Robotics

From Theory to Practice

Chao Chen, Wesley Au, and Shao Liu

CRC Press
Taylor & Francis Group

Robotics

Robotics: From Theory to Practice introduces robotic theories and technologies to audiences, including university students, professionals with engineering backgrounds, and even high-school students interested in building their own robots. We aim to bridge the gap between classic theories and real-world applications of robotic manipulators, which, to date, have far exceeded the domain of conventional industry.

The contents are divided into three parts. The first two cover classic theories of robotics, including kinematics, dynamics, path planning, control, and programming. Specifically, Part I is an introduction targeting junior students, featuring more simplistic topics and examples. Part II provides the senior students and professionals with more in-depth discussions on critical topics and more comprehensive examples. In Part III, we demonstrate how classic robotics theory can be extended to more advanced theoretical frameworks and adopted in real-world applications beyond conventional industries.

This textbook is valuable to broad readers, including those who have limited background in general engineering and wish to explore non-conventional applications of robotic manipulators. The scaffolded contents from Part I to Part III are created to lower the prerequisites and smooth the learning curve.

Chao Chen is the Director of the Laboratory of Motion Generation and Analysis (LMGA), the Academic Supervisor of the Monash Nova Rover Team, and was the Course Director of Robotics and Mechatronics in the Department of Mechanical and Aerospace Engineering at Monash University. His research in robotics was well recognised by a number of awards including Excellence Award of Project of Year 2023 for his harvesting robot by Engineers Australia (Victoria).

Wesley Au is a robotics and automation research engineer at Boeing Research and Technology Australia, designing advanced production systems for the automation of precise and high-rate manufacturing in the aerospace industry. He has a strong background in robotics, especially in intelligent path planning within complex and dynamic workspaces.

Shao Liu is a research fellow in the LMGA at Monash University. He has extensive expertise in robotic design, modelling, and analysis, and has conducted a number of research projects and industry projects in medical robots, manufacturing robots, and special-purpose robots.

Robotics
From Theory to Practice

Chao Chen, Wesley Au, and Shao Liu

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

First edition published 2025
by CRC Press
2385 NW Executive Center Drive, Suite 320, Boca Raton FL 33431

and by CRC Press
4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press is an imprint of Taylor & Francis Group, LLC

ISBN: 978-1-041-01359-4 (hbk)
ISBN: 978-1-041-01323-5 (pbk)
ISBN: 978-1-003-61431-9 (ebk)

DOI: 10.1201/9781003614319

Typeset in CMR10 font
by KnowledgeWorks Global Ltd.

Publisher's note: This book has been prepared from camera-ready copy provided by the authors.

Access the Instructor and Student Resources: www.routledge.com/9781041013235.

To Helen, Jeffrey, Xiaoling, and my parents,
Chao Chen

To Lily, Matthew, Ethan, Jasper, Keith, and Clara,
Wesley Au

To Shuying, my son, Xianjun, and Xiaoyi,
Shao Liu

Contents

Preface **xv**

Glossary **xix**

I Basics of Robotics 1

1 Introduction **3**
- 1.1 Mechanics and Control of Robotics . . . 5
  - 1.1.1 Basics of Robotics . . . 5
  - 1.1.2 Key Topics . . . 6
  - 1.1.3 Advanced Analysis and Case Studies . . . 8
- 1.2 Robot Architectures . . . 9
  - 1.2.1 Cartesian . . . 9
  - 1.2.2 Cylindrical . . . 9
  - 1.2.3 Spherical . . . 11
  - 1.2.4 SCARA . . . 11
  - 1.2.5 Articulated . . . 12
  - 1.2.6 Wrists . . . 14
- 1.3 Conclusion . . . 15
- 1.4 Exercises . . . 15

2 Planar Kinematics, Velocity, and Statics **16**
- 2.1 Two-Dimensional Space . . . 17
- 2.2 Transformations . . . 17
  - 2.2.1 Transforming a Vector . . . 17
  - 2.2.2 Transformation Trees . . . 19
- 2.3 Planar Robot Kinematics . . . 20
  - 2.3.1 Joints . . . 20
  - 2.3.2 Forward Kinematics . . . 21
  - 2.3.3 Inverse Kinematics . . . 24
- 2.4 Velocity Analysis . . . 26
  - 2.4.1 Linear and Angular End Effector Velocity . . . 27
  - 2.4.2 Jacobian . . . 29
  - 2.4.3 Singularity . . . 30
- 2.5 Statics Analysis . . . 31
- 2.6 Workspace . . . 32
- 2.7 Conclusion . . . 33
- 2.8 Exercises . . . 34

3 Trajectory Generation **36**
- 3.1 Interpolation with Polynomials . . . 36

3.2 Interpolation with Linear and Parabolic Functions . . . 42
3.3 Trajectory in Joint Space . . . 45
3.4 Trajectory in Task Space . . . 46
3.5 MATLAB® Examples . . . 48
3.6 Conclusion . . . 54
3.7 Exercises . . . 54

4 Control Schemes 56
4.1 Open-Loop Control . . . 56
4.2 Closed-Loop Control . . . 57
4.2.1 Bang-Bang Control . . . 57
4.2.2 Proportional Controller . . . 59
4.2.3 Integral Controller . . . 61
4.2.4 Derivative Controller . . . 61
4.2.5 PI and PD Controller . . . 62
4.2.6 Proportional-Integral-Derivative (PID) Controller . . . 64
4.3 Pulse Width Modulation Control . . . 65
4.3.1 Relation to Joint Control . . . 65
4.4 Conclusion . . . 66
4.5 Exercises . . . 66

II Key Topics 69

5 General Rotations and Transformations 71
5.1 Position and Orientation . . . 71
5.1.1 Functions of a Rotation Matrix . . . 73
5.2 General Orientation . . . 73
5.3 Fixed Angles . . . 75
5.3.1 Forward Problem . . . 75
5.3.2 Inverse Problem . . . 76
5.4 Euler Angles . . . 77
5.4.1 Forward Problem . . . 78
5.4.2 Inverse Problem . . . 79
5.5 General Transformation . . . 80
5.6 MATLAB® Examples . . . 83
5.6.1 Matrix Arithmetic . . . 83
5.6.2 Inverse Transformation Matrix . . . 84
5.7 Conclusion . . . 87
5.8 Exercises . . . 88

6 Forward Kinematics 91
6.1 Joints . . . 91
6.2 Denavit-Hartenberg Notation . . . 91
6.3 DH Parameters for Prismatic Joints . . . 97
6.4 MATLAB® Examples . . . 102
6.4.1 DH Parameter Summary . . . 105
6.5 Conclusion . . . 106
6.6 Exercises . . . 107

7 Inverse Kinematics **110**
7.1 Basic Techniques . . . 110
7.2 Analytical Solution to Inverse Kinematics . . . 112
7.3 Univariate Polynomial . . . 119
7.4 Dialytic Method . . . 121
7.5 Conclusion . . . 126
7.6 Exercises . . . 127

8 Jacobian Analysis **132**
8.1 Jacobian Matrix . . . 132
8.2 Velocity in Translation and Rotation . . . 136
8.3 Velocity Propagation . . . 140
8.4 Statics . . . 144
8.4.1 Relation between Jacobians and Effort . . . 144
8.5 Workspace . . . 148
8.6 MATLAB® Examples . . . 150
8.7 Conclusion . . . 154
8.8 Exercises . . . 154

9 Path Planning **159**
9.1 Configuration Space . . . 159
9.1.1 Grid . . . 161
9.1.2 Random Sampling . . . 162
9.2 State Connectivity . . . 163
9.2.1 Grids . . . 163
9.2.2 Random Samples . . . 166
9.2.3 Connectivity Matrix . . . 166
9.3 Planning Completeness . . . 168
9.3.1 Measuring Complexity . . . 168
9.4 Complete Planning Algorithms . . . 169
9.4.1 First-Search Algorithms . . . 169
9.4.2 Dijkstra's Algorithm . . . 175
9.4.3 A* . . . 179
9.5 Sample-Based Planning Algorithms . . . 184
9.5.1 Probabilistic Road Map (PRM) . . . 184
9.5.2 Rapidly Exploring Random Trees (RRT) . . . 186
9.6 Potential Field Planners . . . 189
9.7 Conclusion . . . 189

10 Programming **190**
10.1 Modelling . . . 190
10.1.1 Symbolic Functions and Handles . . . 190
10.2 Robot Operating System . . . 191
10.2.1 Operating Paradigm . . . 192
10.2.2 ROS Components . . . 193
10.2.3 ROS System: Case Study . . . 195
10.3 Conclusion . . . 200

11 Lagrangian Dynamics **201**
11.1 Rigid Body Dynamics . . . 201
11.2 Inertia Tensors . . . 202

11.3 Principal Moments of Inertia . . . 204
11.3.1 Parallel Axis Theorem . . . 206
11.4 The Lagrangian Method . . . 207
11.4.1 Mass Matrix . . . 217
11.4.2 Gravity Term . . . 218
11.4.3 Friction Term . . . 218
11.4.4 Load Term . . . 219
11.4.5 MATLAB Example . . . 219
11.5 Conclusion . . . 220
11.6 Exercises . . . 220

12 Newton-Euler Dynamics **222**
12.1 Newton's and Euler's Equations . . . 222
12.1.1 Rigid Body Rotation . . . 223
12.1.2 Intermediate Axis Theorem . . . 225
12.2 Outward Propagation . . . 227
12.3 Inward Propagation . . . 231
12.4 Procedure . . . 233
12.5 Twist, Wrench, and 6×6 Transformation Matrix . . . 236
12.5.1 Transformation of Twist . . . 236
12.5.2 Inverse Transformation of Twist . . . 237
12.5.3 Transformation of Wrench . . . 238
12.5.4 Derivative of ${}^{0}\mathbf{X}_0$ Matrix . . . 238
12.5.5 Physical Interpretation . . . 241
12.6 Applications in Computing . . . 241
12.6.1 Computing Efficiency . . . 241
12.7 Conclusion . . . 248
12.8 Exercises . . . 248

13 Joint Control **252**
13.1 Servo Dynamics . . . 252
13.2 Modelling Servos with a Gear Transmission . . . 255
13.3 Fixed Reference Tracking . . . 261
13.3.1 P Controller . . . 261
13.3.2 PD Controller . . . 264
13.3.3 PID Controller . . . 269
13.4 Error Dynamics . . . 272
13.4.1 First-Order Error Dynamics . . . 273
13.4.2 Second-Order Error Dynamics . . . 273
13.4.3 Velocity Control . . . 275
13.5 Conclusion . . . 276
13.6 Exercises . . . 276

14 Computed Torque Control **278**
14.1 SISO Computed Torque Control . . . 278
14.2 MIMO Computed Torque Control . . . 279
14.3 Controller with Gravity Compensation . . . 282
14.4 Lyapunov Stability . . . 283
14.4.1 Basic Definition . . . 283
14.4.2 Lyapunov's Direct Method . . . 285
14.5 Dynamic Relation . . . 286

14.6 Stability Analysis of Nonlinear Robotic Controllers . . . 288
14.7 Motion Control in Task Space . . . 289
14.7.1 Dynamics in Task Space . . . 290
14.7.2 Task Space Dynamics Partitioning . . . 291
14.7.3 Error Dynamics . . . 291
14.8 Conclusion . . . 292
14.9 Exercises . . . 293

15 Force Control **295**
15.1 Single Axis Control in Task Space . . . 295
15.2 Hybrid Motion-Force Control . . . 297
15.2.1 Natural Constraints . . . 297
15.2.2 Artificial Constraints . . . 298
15.3 Impedance Control . . . 300
15.3.1 Single Axis Control . . . 300
15.3.2 Task Space Control . . . 301
15.3.3 Control with Environmental Dynamics . . . 301
15.3.4 Dynamic Relations . . . 302
15.4 Conclusion . . . 304
15.5 Exercises . . . 304

III Advanced Analysis and Case Studies **307**

16 Mobility Analysis **309**
16.1 Mobility Analysis Based on the Pattern of Transformation Matrix . . . 309
16.1.1 Pattern of Transformation Matrix . . . 309
16.1.2 Case Study—3-UPU Parallel Manipulator . . . 311
16.1.3 Case Study—3R2T Parallel Manipulator . . . 313
16.1.4 Case Study—3R1T Parallel Manipulator . . . 314
16.1.5 Case Study—2-RPS-UPS Parallel Manipulator . . . 315
16.2 The Order of Mobility . . . 316
16.2.1 Local Mobility . . . 317
16.2.2 Case Study—A Novel Spatial Four-Bar Linkage . . . 318
16.3 Conclusion . . . 322
Bibliography . . . 322

17 Orientation Workspace **324**
17.1 Measurement Principles and Quaternions . . . 324
17.2 Generalised Volume of Orientation Workspace . . . 325
17.3 Orientation Workspace Volume of Various Kinematic Descriptions . . . 327
17.3.1 Euler Angles . . . 327
17.3.2 Angle-Axis Representation . . . 328
17.4 Case Study . . . 328
17.5 Conclusion . . . 330
Bibliography . . . 330

18 Constraint Analysis for Underactuated Systems **332**
18.1 Constrained Minimisation . . . 333
18.2 Kinetostatic Constraints . . . 333
18.2.1 Generalised Coordinates and Generalised External Forces . . . 333

18.2.2 General-Form Kinematic Constraint Equations and Lagrange Multipliers . . . 334
18.2.3 Force Constraint Equations . . . 335
18.2.4 Hooke Constitutive Equations . . . 335
18.2.5 Selection Matrices . . . 335
18.2.6 Array of Unknowns . . . 336
18.2.7 Kinetostatic Constraints . . . 336
18.3 Case Study — Soft Robots with Pseudo-Rigid Bodies . . . 337
18.3.1 3R-PRB Model of Backbone Sub-Segments . . . 338
18.3.2 Generalised Coordinates and Generalised External Forces . . . 338
18.3.3 Kinematic Constraint Equations and Lagrange Multipliers . . . 339
18.3.4 Hooke Constitutive Equations . . . 340
18.3.5 Constraint Inequalities . . . 341
18.3.6 Constrained Minimisation . . . 341
18.3.7 Experimental Verification . . . 342
18.4 Case Study — The Compliant Five-Link Epicyclic Finger . . . 343
18.4.1 Generalised Coordinates and Generalised External Forces . . . 344
18.4.2 Kinematic Constraint Equations and Lagrange Multipliers . . . 346
18.4.3 Hooke Constitutive Equations . . . 347
18.4.4 Force Constraints of Contacts . . . 348
18.4.5 Selection Matrices . . . 348
18.4.6 Constrained Minimisation . . . 349
18.4.7 Experimental Verification . . . 350
18.5 Conclusion . . . 351
Bibliography . . . 351

19 Concentric Tube Robot **353**
19.1 Overview of Concentric Tube Robot . . . 353
19.2 Robot-Independent Mapping . . . 354
19.3 Robot-Dependent Mapping . . . 356
19.3.1 Constitutive Equation . . . 356
19.3.2 Static Equilibrium . . . 357
19.3.3 Variations of the Governing Equation . . . 358
19.4 Iteration . . . 360
19.5 Case Studies for Computational Efficiency . . . 361
19.6 Conclusion . . . 363
Bibliography . . . 363

20 Path Planning of Parallel Manipulators **364**
20.1 Parallel Manipulator Kinematics . . . 364
20.1.1 Singularities . . . 365
20.1.2 Example: A 5R Parallel Manipulator . . . 366
20.2 The Path Planning Problem . . . 368
20.3 Methodology . . . 369
20.4 Generating Charts . . . 370
20.4.1 Singularity Locus . . . 370
20.4.2 Identifying a Chart . . . 370
20.4.3 Constructing a Chart's $\mathcal{C}$-space . . . 372
20.5 Constructing the Atlas . . . 373
20.6 Case Study: A 3-RRR Parallel Manipulator . . . 374
20.6.1 Forward Kinematics . . . 375

20.6.2 Continuity Conditions . . . 378
20.6.3 Singularity Analysis . . . 378
20.6.4 Path Planning . . . 380
20.6.5 Performance . . . 383
20.7 Conclusion . . . 383
Bibliography . . . 384

21 Minimally-Invasive Surgical Robot with Remote Centre of Motion 387
21.1 Minimally Invasive Surgery and MIS Robots . . . 387
21.2 The Dual-Triangular Linkage . . . 389
21.2.1 The Design . . . 389
21.2.2 Proof of Remote Centre of Motion . . . 390
21.2.3 The Gear-Constrained Dual-Triangular Linkage . . . 391
21.2.4 The 2R1T RCM Mechanism Based on the DT Linkage . . . 392
21.3 The Cable-Constrained Linkage with Remote Centre of Motion . . . 393
21.3.1 The Design . . . 393
21.3.2 Proof of Functioning . . . 394
21.4 Advantage of DT/CC Linkage over Parallelogram RCM Mechanism . . . 399
21.5 Conclusion . . . 401
Bibliography . . . 401

22 MARS: The Monash Apple Retrieving System 402
22.1 Background and Motivation . . . 402
22.2 Virtual Environment Construction . . . 403
22.2.1 Kinematics . . . 404
22.2.2 Auxiliary Frame Assignments . . . 406
22.3 Pose Optimisation . . . 406
22.3.1 Objective Function . . . 408
22.3.2 Kinematic Constraints . . . 408
22.3.3 Collision Constraints . . . 409
22.3.4 Kinematic Configuration Constraint . . . 410
22.4 Path Planning . . . 410
22.4.1 Planning Algorithm . . . 410
22.4.2 Harvesting Sequence . . . 411
22.5 Results . . . 415
22.5.1 Harvest Success Rate . . . 415
22.5.2 Cycle Time . . . 417
22.6 Conclusion . . . 417
Bibliography . . . 418

IV Appendix 421

23 Appendices for Chapter 5 423
23.1 Angle-Axis . . . 423
23.1.1 Forward Problem . . . 423
23.1.2 Inverse Problem . . . 424
23.2 Quaternion . . . 425
23.3 Alternate Expression of Rodrigues' Formula . . . 427

24 Appendices for Chapter 8 **428**
24.1 Proof of $\mathbf{Q}\,\mathbf{Q}^T$ Being Screw-Symmetric 428
24.2 Proof of a Screw-Symmetric Matrix Being the Crossproduct Matrix of a Vector 428
24.3 Proof of $\boldsymbol{\omega}$ 429

25 Appendices for Chapter 11 **430**
25.1 Inertia Tensor 430
25.2 Euler's Law 430
25.3 Proof of Parallel-Axis Theorem 431

Index **433**

Preface

Robotics is a fascinating subject, which often triggers our deep curiosity and makes us think about how to create robots to improve people's daily lives. Fortunately, I have had the opportunity to teach robotics in the Discipline of Robotics and Mechatronics at Monash University since 2008. My students asked many interesting questions, which contributed significantly to this textbook. For example, a rotation or transformation matrix is described from frame 1 to frame 2 in some textbooks; however, the exact rotation or transformation matrix is described from frame 2 to frame 1 in other textbooks. Clear explanations of the underlying principles to such questions improved the learning of students and enriched this textbook.

Besides the students in my class of robotics, the Monash Nova Rover team has also utilised this textbook (earlier version) as a handbook to develop the robotic manipulators on their rovers. The team is an undergraduate student team founded in 2017, aiming at developing Mars rovers and Lunar rovers for various competitions. As the academic supervisor and founder of this student team, I have observed that the team gradually and successfully developed the robotic manipulators for competitions, by implementing forward kinematics, inverse kinematics, path planning, and remote control. The team won Second Place in the 2022 and 2023 University Rover Challenges in Utah, and First Place in the 2021, 2022, and 2023 Australian Rover Challenges.

This textbook is co-authored by my former PhD students, Dr Wesley Au and Dr Shao Liu. Dr Wesley Au has a strong background in robotics and path planning in complex and dynamic workspaces, and brings his knowledge and expertise in cutting-edge robotic automation for manufacturing into this textbook in both an academic and industrial setting. He has worked in both research and industry-led projects, ranging from critical infrastructure maintenance robots to agricultural robots. He is now an automation research engineer at Boeing Research and Technology Australia, designing advanced production systems for the automation of precise and high-rate manufacturing in the aerospace industry. Dr Shao Liu is a Research Fellow in my lab at Monash University. He has extensive experience and knowledge in robotic design, modelling, and analysis, and has conducted a number of research projects and industry projects in medical robots, manufacturing robots, and special-purpose robots. In particular, Dr Liu's expertise in soft robots and general underactuated systems helps extend the coverage of this textbook beyond classic theories and applications.

This textbook aims to bridge the gap between theories in robotics and real-world applications in modelling, analysis, planning, and control of robotic manipulators. The contents are divided into three parts. The first part targets senior secondary school students and junior university students, by introducing the basics of kinematics, trajectory planning, and control. The second part targets senior university students, by introducing the key topics of robotics, including spatial transformations, kinematics and dynamics, as well as control and programming, which shall also serve as a valuable source to professionals who are interested in robotics and have backgrounds in engineering. The third part targets the researchers in robotics, through the discussions of advanced analysis and case studies in soft robotics, medical robotics, and agricultural robotics. While there exist a number of classic textbooks in robotics to which we pay full respect, the following reasons motivated us to develop this textbook.

Firstly, there is a growing interest of young students in robotics, with an increased number of robotics teams and clubs in secondary schools and universities around the world. However, robotics units or courses are usually taught in the third or fourth year in most undergraduate curricula, preventing junior students from obtaining the necessary knowledge to develop their robotics projects. Part I of this textbook attempts to tackle this gap, by introducing necessary knowledge with minimum prerequisites. Further, completing Part I will enable a smooth transition to more advanced topics in Part II.

Secondly, robotic systems have been applied to much broader fields such as medicine, agriculture, and infrastructure, beyond traditional scenarios in factories. Therefore, Part III attempts to introduce these advanced applications by extending the classic robotic technologies of conventional industrial robotic manipulators in Part II.

Thirdly, there are concepts and theorems in robotics that may be difficult to understand or visualise, particularly those involving three or more dimensions. Therefore, we have included comprehensive examples with MATLAB® code, which can assist in the visualisation and understanding, and also be generalised for various robotic systems. We hope that this feature will make the fundamental mathematics and algorithms more interesting, insightful, and engaging for the readers.

The contents of this textbook are summarised as follows.

Part I introduces the basics of planar robots for simple tasks through modelling, planning, and executing, from Chapters 1–4.

Chapter 1 serves as an introduction to the field of robotics and our textbook. It starts with a brief history of robots in industrial applications. Next, commonly used robot architectures are discussed, including their degrees of freedom (DoF), properties of motion, and usual applications. This chapter also includes a summary of the topics discussed, along with their per-chapter allocations in the textbook.

Chapter 2 tackles the problem of modelling a planar robotic manipulator. Theoretical fundamentals are presented for both the kinematic and the force domains, covering transformations in the planar space, kinematics, velocity analysis, Jacobian, statics, and robot workspace.

Chapter 3 presents the methods to compute a valid trajectory for the planar robotic manipulator to execute. Two approaches are presented: polynomial interpolation and cubic splines. Additionally, trajectory generation in both the joint space and the task space is discussed.

Chapter 4 introduces the basics of control schemes, including open-loop control, closed-loop control, and pulse width modulation. Various controllers realising the closed-loop control schemes are also presented.

Part II covers general transformation, kinematics, dynamics, path planning, programming, and control of spatial robots from Chapters 5–15.

Chapter 5 deals with spatial transformation. The definition of robot pose, i.e., position and orientation, is given. Commonly adopted kinematic descriptions are introduced, with fixed angles and Euler angles in the body part of the chapter, and angle-axis and quaternion in the appendix of this chapter. Both the forward and the inverse problems are discussed for each description.

Chapter 6 is dedicated to the forward kinematics of serial robotic manipulators based on the Denavit-Hartenberg method. The step-by-step implementation of the method is introduced through the analysis of robots with revolute and prismatic joints, a spherical robot wrist, the PUMA robot, and a cylindrical robot.

Chapter 7 focuses on solving the inverse kinematics problem of serial robotic manipulators. Firstly, trigonometric functions that are commonly used to solve inverse kinematics problems are introduced as the foundation. From there, examples with analytical solutions are given, followed by discussions on univariate polynomials and the dialytic method for the cases where analytical solutions are infeasible.

Chapter 8 discusses the Jacobian matrix and velocity analysis. On the Jacobian side, the chapter starts with the definition of a Jacobian matrix, and discusses the end-effector-to-joint velocity and force mappings based on the Jacobian matrix. Workspace singularity analysis is also included. Moreover, the chapter extends the discussion on velocity to present velocity propagation.

Chapter 9 covers the path planning methods and algorithms commonly used in robotics. The concept of configuration space is first introduced. Three path planners are discussed: the complete planners, the sample-based planners, and the potential field planners. The discussion further extends to detailed algorithms belonging to individual planners.

Chapter 10 focuses primarily on the Robot Operating System (ROS). The operating paradigm and ROS components are discussed, followed by a case study.

Chapter 11 is concerned with the Lagrangian dynamics of serial robotic manipulators. Inertia tensors and principal moments of inertia are first introduced. The chapter then presents a step-by-step guide on the implementation of the Lagrangian method.

Chapter 12 is on the Newton-Euler dynamics of serial robotic manipulators. The outward propagation of velocity and acceleration, and the inner propagation of force and moment are described, followed by a step-by-step guide on the implementation of the Newton-Euler method. The chapter also includes an introduction to the twist, wrench, and their corresponding unified transformation matrix.

Chapter 13 covers the basics of actuator control, where a linear controller is derived for a single actuator. Servo dynamics and the modelling of geared servo are presented. The chapter then moves on to fixed reference tracking with P, PD, and PID controllers, and concludes with the discussion on error dynamics.

Chapter 14 focuses on computed torque control that takes into account nonlinearity in robot dynamics. The first part of the chapter discusses SISO and MIMO controllers, as well as controllers with gravity compensation, all in the joint space. The second part is on Lyapunov stability analysis. In the last part of the chapter, motion control in task space is presented.

Chapter 15 introduces the force control strategy. The chapter starts with single-axis force control, and takes it one step further to discuss hybrid motion-force control, where natural and artificial constraints are presented. Finally, an overview of the impedance controller is provided.

Part III investigates advanced robotic modelling and analysis, and the robotic applications in various fields, building upon the knowledge in Parts I and II, from Chapters 16–22.

Chapter 16 is concerned with the mobility analysis of mechanisms. The chapter is divided into two parts. In the first part, the focus is the global mobility of parallel manipulators, where a method that reveals the mobility, property of motion, and actuation pattern based on the manipulator's transformation matrix is presented. In the latter half, the focus is redirected to the analysis of local mobility based on Taylor's theorem.

Chapter 17 presents the investigation of the parameterisation of the orientation workspace of spherical manipulators. An approach based on quaternion is proposed, followed by the mapping to convert parameterisation based on other kinematics descriptions, such as Euler angles and angle-axis, to that based on quaternion to allow like-to-like comparisons among manipulators of different designs.

Chapter 18 introduces a framework to conduct kinetostatic analysis of planar underactuated systems. The derivation of core kinetostatic constraints based on generalised coordinates, kinematic constraint equations, and Lagrange multipliers is presented, along with selection matrices and the constrained minimisation method to complete the formulation and obtain the solution. Two case studies are included, on a tendon-driven robot with a continuum backbone and an adaptive prosthetic finger, respectively.

Chapter 19 covers the modelling of concentric tube robots, a soft robot featuring multiple concentrically arranged continuum superelastic tubes as the body. The modelling is divided into two parts: a robot-independent mapping developed based on the model of strands, and a robot-dependent mapping relating the actuator inputs and the prescribed shapes of individual tubes to the converged shape of the robot. Variations in the governing equations are presented, and their computational efficiencies are compared.

Chapter 20 discusses efficient path planning of parallel manipulators. The chapter starts with the kinematics of parallel manipulators and highlights the challenge in their path planning due to complex singularity profiles. A path planning method is proposed, which actively exploits the workspace singularity loci as gates to connect singularity-free workspace regions for efficient configuration space path planning. A case study based on the 3-RRR manipulator is included and demonstrates successful assembly mode changes of the manipulator.

Chapter 21 presents a novel remote centre of motion mechanism designed for minimally invasive surgery. Two design variants are discussed, which are hybrid mechanisms of gear-linkage and cable-linkage, respectively. Mathematical proofs on the remote centre of motion property are provided, followed by a case study to demonstrate the mechanism's advantage in terms of footprint minimisation compared to the commonly used parallelogram-based designs.

Chapter 22 overviews the design and testing of an edge-cutting robotic apple-harvesting system. The first part of the chapter covers robot design, the construction of the virtual workspace environment, the optimisation of the robot harvesting pose, and harvesting path planning. The second part presents the performance analysis of the robot based on the outcomes of field tests.

This textbook has been influenced by many outstanding books in robotics, in particular, *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms* by Angeles, *Introduction to Robotics* by Craig, *Introduction to Theoretical Kinematics* by McCarthy, *Springer Handbook of Robotics* by Siciliano and Khatib, *Robot Modeling and Control* by Spong, Hutchinson, and Vidyasagar, *Modern Robotics: Mechanics, Planning, and Control* by Lynch and Park, *Robot Dynamics Algorithms* by Featherstone, *Modelling and Control of Robot Manipulators* by Sciavicco and Sicilliano, and *Engineering Applications of Noncommutative Harmonic Analysis: With Emphasis on Rotation and Motion Groups* by Chirikjian and Kyatkin.

To keep the contents of teaching and learning concise, we omitted the list of bibliographies in Parts I and II, where classic theories and well-known technologies are discussed. Bibliographies are included in Part III for readers who desire to conduct a deeper dive into these research topics and case studies.

Finally, I would like to express gratitude to all of the teaching assistants in my class of robotics for their valuable inputs and contributions, especially, Dr Keenan Granland, Mr Eugene Kok, Mr Dariusz Skoraczynski, Dr Sajeeva Abeywardena, Dr Stanley Ip, Dr Charles Treoung, Dr Xing Wang, Dr Hugh Zhou, Dr Zijue Chen, and Dr Hanwen Kang. Moreover, we appreciate the research funds of the Australian Research Council and Perpetual Philanthropy, for the research projects leading to the case studies in Part III of the textbook.

Chao Chen

Glossary

Notations and symbols

$\{0\}$	Ground frame
$\{i\}$	i-th frame
$\{j\}$	j-th frame (following $\{i\}$)
q_i	i-th joint position
${}^i\mathbf{T}_j$	Homogeneous transformation matrix mapping coordinates from $\{j\}$ to $\{i\}$
${}^i\mathbf{R}_j$	Rotation matrix mapping coordinates from $\{j\}$ to $\{i\}$
${}^k_i\mathbf{p}_j$	Position vector pointing from the origin of $\{j\}$ to the origin of $\{i\}$, measured in $\{k\}$
${}^i\mathbf{p}_j \equiv {}^i_i\mathbf{p}_j$	Position vector pointing from the origin of $\{j\}$ to the origin of $\{i\}$, measured in $\{i\}$
${}^i\boldsymbol{\omega}_j \equiv {}^i_0\boldsymbol{\omega}_j$	Angular velocity of $\{j\}$ with respect to ground, measured in $\{i\}$
${}^i_j\boldsymbol{\omega}_C$	Angular velocity of $\{C\}$ with respect $\{j\}$, measured in $\{i\}$
${}^i\mathbf{z}_i$	z axis of frame $\{i\}$, measured in $\{i\}$ (time invariant)
${}^k_i\mathbf{v}_j$	Linear velocity of $\{j\}$ with respect to $\{i\}$, measured in $\{k\}$
${}^i\mathbf{v}_j \equiv {}^i_i\mathbf{v}_j$	Linear velocity of $\{j\}$ (with respect to $\{i\}$), measured in $\{i\}$
${}^i\mathbf{I}$	Inertia tensor of link A, measured in $\{i\}$
${}^i\mathbf{I}_j$	Inertia tensor of link B, measured in $\{i\}$
${}^i\dot{\boldsymbol{\omega}}_j \equiv {}^i_0\dot{\boldsymbol{\omega}}_j$	Angular acceleration of $\{j\}$ with respect to ground, measured in $\{i\}$
${}^i\ddot{\mathbf{p}}_j \equiv {}^i_0\ddot{\mathbf{p}}_j$	Linear acceleration of frame j with respect to ground, measured in $\{i\}$
${}^i\mathbf{f}_j$	Force acting on link j by link $j-1$ at joint j, measured in $\{i\}$
${}^i\mathbf{f}_{cj}$	Inertia force acting on link j at its mass centre, measured in $\{i\}$
${}^i\mathbf{n}_j$	Torque acting on link j by link $j-1$ at joint j, measured in $\{i\}$
${}^i\mathbf{n}_{cj}$	Torque acting on link j, measured in $\{i\}$
$\mathcal{L}$	Lagrangian
$\mathcal{C}$	Configuration
$\mathcal{V}$	Twist
$\mathcal{F}$	Wrench

Part I

Basics of Robotics

1 Introduction

The history of robotics and automation is brief, dating back as recently as the 1960s when the first wave of industrial robots was introduced in manufacturing lines for the first time. During this period, concerns over worker safety in manufacturing environments and the increase in labour costs were becoming significant issues for the manufacturing industry. The introduction of robots in the manufacturing industry was initially touted as a solution to alleviate these concerns. However, after some time in the industry, the advantages of using robots in this sector soon made headlines, as car manufacturers saw an increase in productivity by more than double. This was in part due to minimal downtime and comparably high speed and accuracy in performing repetitive tasks to manual labourers. With the eventual declining costs of industrial robots and general technological advances, robots are now a common staple in modern manufacturing plants.

While modern technological advances in the manufacturing industry can be attributed to improvements in circuit design, processing, and computing, leaps in manufacturing technology can be attributed to advances in mechanical design. These leaps in manufacturing technology can be observed as a series of industrial revolutions, as shown in Table 1.1, with an approximate decade of establishment and key advancements of technologies highlighted. The first industrial revolution in the 1780s saw the use of mechanisms, which were developed to take full advantage of steam and water power by converting a source of power to one that was mechanically useful. This ingenuity formed the basis of intelligent mechanical design that would later bring about further industrial revolutions. Through smart mechanical engineering and the discovery of better power sources, electricity and micro-circuitry, processing and computing, industrial revolutions have accelerated during the 20th century, lending to increased manufacturing capabilities and efficiency that we see today.

The Industrial Revolution 3.0 in the 1960s saw the advancement of both mechanical and electrical circuit design, culminating in the first use of robotics manufacturing in automation. Perhaps the most well-known robot of this era was called the Unimate. It was a hydraulically driven four-degree-of-freedom (four-DoF) robot with two actuators at the base representing the robot's shoulder, and a prismatic (telescopic) and revolute actuator to control the extension and one-axis rotation of the working tool. Its initial role was to perform die casting and welding in the manufacture of cars in an environment that was considered dangerous for humans due to lead fumes and radiant heat and light. These robots were first installed in car manufacturing plants in the late 1960s, which saw the manufacturing speed more than double its existing speed. Other car manufacturers and other industries took note of this milestone and, in a short period of time, revolutionised the manufacturing industry. Its success led the manufacturer of the original Unimate robot to develop a successor, the PUMA robot (Programmable Universal Machine for Assembly). Featuring a similar architecture to the original Unimate robot but on a smaller scale, it has become a standard for robots in the manufacturing industry with typical uses in standard pick-and-place tasks, assembly, and welding, to name a few.

DOI: 10.1201/9781003614319-1

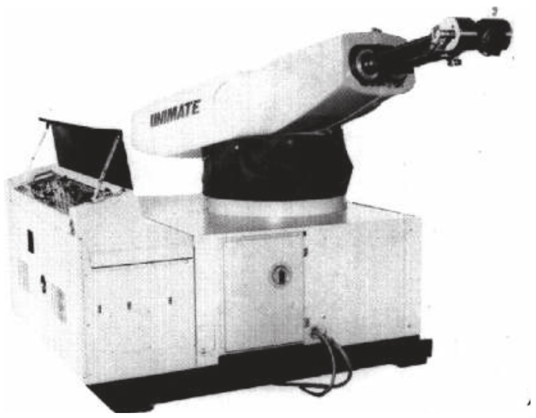


FIGURE 1.1
The original Unimate robot.[1]

TABLE 1.1
Industrial Revolutions, from First Inception in 1784 to Today with Key Technological Advances[2,3]

Industry 1.0 c. 1784	**Industry 2.0** c. 1870	**Industry 3.0** c. 1969	**Industry 4.0** Today
Mechanical production Steam and water power Weaving loom	First assembly line Mass production Electricity Discovery of oil and production of petrol	Computers and IT Microcircuits and processors Automated production Global supply chains	Internet of Things (IoT) Smart devices Macroscale computing Real-time data

Although the manufacturing industry has benefited significantly from the use of robots in the past few decades, it has always met with some resistance. Manufacturing roles that were previously filled by human workers and considered automatable were eventually replaced by robots. However, modern robotics has seen a paradigm shift to more assistive and collaborative robots in manufacturing, as the industry moves towards adaptive manufacturing. Today, we see many robots used in many assistive roles, and they can come in many shapes and forms, such as vehicles, lifters, and exoskeletons. This new era of assistive robotics in the industry is called Industry 4.0, where workers and robots can co-exist in the same workspace, driving up both productivity and manufacturing flexibility. It is the adopted approach in most wide-scale manufacturing plants in Australia and around the world. While artificial intelligence is the primary driver of the latest industrial revolution, in each robot, there is always an aspect of intelligent mechanical design, which, in a literal sense, serves as the backbone for all robotic systems.

[1] Aly, M.F., Abbas, A.T. and Megahed, S.M., 2010. Robot workspace estimation and base placement optimisation techniques for the conversion of conventional work cells into autonomous flexible manufacturing systems. *International Journal of Computer Integrated Manufacturing*, 23(12), pp.1133–1148.

[2] https://www.britannica.com/technology/history-of-technology/The-Industrial-Revolution-1750-1900 (Retrieved 09.10.2024)

[3] https://en.wikipedia.org/wiki/Industrial_Revolution (Retrieved 09.10.2024)

This textbook introduces the field of robotics to any aspiring student who may be interested in this young but ambitious discipline. Each topic in this textbook is introduced in its own chapter, which covers all the fundamental theorems relating to robot analysis, design, and control. Although the field of robotics is relatively new, it covers a vast range of topics, and this textbook simply cannot cover all of them. However, this textbook will provide the foundations of classical robotics theory, which will pave a clear pathway towards other exciting fields in robotics that are not covered by this textbook.

1.1 Mechanics and Control of Robotics

The contents in this textbook are arranged in three parts: fundamental topics, advanced topics, and research and case studies. A summary of the contents is provided below.

1.1.1 Basics of Robotics

Robot Architectures

This topic is covered in this chapter, which introduces the most common robot architectures used in manufacturing and robotics research. A **robot architecture** is defined as a specific configuration of actuators that make up the basic structure of the robot. Each architecture has its own specific strengths, which are fully utilised in application, and will be described here. This will provide a basic understanding of common robot designs, which will be used in many examples and problems to solve throughout this textbook.

Planar Kinematics, Velocity, and Statics

This topic is discussed in Chapter 2 and is relevant to the modelling of robotic systems. It provides a minimum-viable yet complete set of information to understand the kinematic and force-domain behaviour of a robotic manipulator in the two-dimensional space: how do we represent the position and orientation of the "tool", i.e., the robot's end-effector in the space; what is the relation between the position and velocity of the joints and those of the end-effector; how much force or torque is needed at the joints to resist an external disturbance; and what is the reachable area of the manipulator. Said analysis is the very first step in the three-step modelling-planning-execution approach to commanding a robot to perform a certain task.

Trajectory

With the behaviour of the robotics manipulator understood, we move on to discuss how the path, i.e., the trajectory, that the robot end-effector is moved within the workspace in Chapter 3. Trajectory generation is the act of controlling the path of a robot's end-effector between two points in a smooth and controlled fashion. In this instance, the goal is simply to ensure the path from the initial to goal positions is smooth and continuous, with the key assumption that the path is collision-free. This topic is covered in Chapter 3, and will describe interpolation methods used to ensure smooth and continuous trajectories between via points.

Control

The basic control schemes presented in Chapter 4 fill the last piece of information towards the three-step modelling-planning-execution approach to command a robot in the application. We provide a brief overview of the open-loop, closed-loop, and pulse width modulation control schemes, along with various commonly used approaches to realise closed-loop control.

1.1.2 Key Topics

Spatial Transforms

This topic is covered in Chapter 5, which introduces the systems and notation we use to describe the position and orientation, or the **pose** of a body in space. Coordinates or **frames** are then attached to each body that represents the robot. The conventions used for spatial descriptions, and the mathematics of manipulating these quantities will be derived in this chapter.

Kinematics

The term *kinematics* is the science of motion with the exclusion of dynamic effects that cause it. In robotics, an actuator can be represented as a **revolute** joint (rotation about a single axis), or a **prismatic** joint (a sliding mechanism along a single axis). As such, a robot, or **manipulator**, is made up of a series of rigid **links** connected by joints to an **end-effector**, which interacts with the environment with a tool attachment. A single chain of joints and links connecting to a single end-effector is called a **serial robot**, and is the primary type of robot analysed in this textbook. Other names for this type of robot include serial manipulator or serial (kinematic) chain. Multiple serial kinematic chains connecting to a single end-effector is called a **parallel robot**, or parallel (kinematic) manipulator, and will not be covered in this textbook.

In robotics, there are two main spaces in which we perform kinematic analysis: the **joint space** contains all possible actuator positions or *configurations* and maps them to a single point within this space, and the **task space**, which in general represents the end-effector's workspace in free space. A robot's **degrees of freedom** (DoF) represents the number of independent axes at which the end-effector can move in, and this determines the number of dimensions of the task space. The task space can be up to six dimensions, three positions and three orientation axes.

Forward Kinematics

Also known as direct kinematics, forward kinematics is the study of the robot's pose in free space when for a given actuator configuration (all actuator positions defined), and is covered in Chapter 6. We learn how to model a serial robot such that we can fully define the robot's pose and determine the exact pose of the end-effector at a given time. In summary, we are finding the equations of motion of the robot's end-effector as a function of its joint positions, i.e., finding a map from the joint space to the task space.

Inverse Kinematics

In forward kinematics, we map the joint space to the task space. In inverse kinematics, we do the reverse mapping — where for a given end-effector pose, we calculate the actuator configuration required to achieve this end-effector pose. This topic is covered in Chapter 7, and will explore the different methods used to perform inverse kinematics. This is a challenging but critical topic, as many trajectory and path planning applications in robotics occur in the task space.

Jacobians and Statics

In addition to kinematics, where we consider static positions, Jacobians or *velocity analysis* is the study of the end-effector velocities with respect to actuator velocities. This is an important tool for **workspace** analysis, which defines the end-effector's reachable task space. This topic is covered in Chapter 8. Another phenomenon velocity analysis can give further insight into is robot **singularities**, which will be defined in this chapter. The final analysis we can perform by using Jacobians is statics analysis — finding the required actuator torques and forces to maintain a static force and torque at the end-effector.

Path Planning

Path planning is a diverse field that has been studied in both robotics and computer science. It is a primitive yet critical problem to solve in robotics and is by no means a trivial problem to solve. For a given robot starting pose, suppose we would like the end-effector to pick up an object in free space. However, the task space has obstacles that inhibit the robot's ability to follow a shortest-path trajectory to the target. The goal of a path planner is to solve this problem by giving the robot a collision-free path between its initial pose and the goal pose, to allow the robot to pick up the object. Chapter 9 will introduce various path planning algorithms that are currently used in the manufacturing industry and in robotics in general.

Programming

In Chapter 10, we introduce to the students Robot Operation System (ROS), a framework that is becoming the gold-standard for robotic research and witnessing ever-increasing use in real-world applications in the industry. The operating paradigm and ROS components are discussed, followed by a case study.

Dynamics

Dynamics in robotics is a large field of study, devoted to analysing the forces required to cause motion. There are two methods for formulating the dynamics of a serial manipulator: **Lagrangian formulation** is discussed in Chapter 11, and iterative **Newton-Euler**, discussed in Chapter 12. In both methods, we determine the equations of torque and force for each actuator, based on payload conditions at the end-effector. This analysis is very important in manipulator design as it will model torque or force (or **effort**) required to maintain a specified motion at the end-effector, and determine the minimum torque requirements for the servos to drive this load.

Control

Robot control theory is another vast topic in robotics. Chapter 13 covers the basics of actuator control, where a linear controller is derived for a single actuator. Chapter 14 covers computed torque control in which the non-linear dynamics often observed in robotic manipulators can be compensated by clever control system design. Finally, Chapter 15 introduces an alternative mode of end-effector control, where the end-effector is constrained by force rather than position.

1.1.3 Advanced Analysis and Case Studies

The following topics cover cutting-edge methods for robot analysis in robotics. These methods are primarily used in the design phase of a robotic manipulator. These topics are advanced, utilising abstract mathematical theorems to demonstrate the theoretical feasibility of the design of mechanisms.

Mobility Analysis

This fundamental topic covers the study of mobility, or the degrees of freedom (DoF) of mechanisms and robotics, and is often the first step in designing a mechanism to fulfil certain motion requirements. Said study also provides valuable insights regarding the property of motion of a manipulator, as well as constraints on actuator arrangements. While Part II of this book introduces the concept of degrees of freedom, Chapter 16 introduces a more comprehensive methodology with a deep dive into the mathematics behind this analysis.

Orientation Workspace

Chapter 17 provides an in-depth introduction to the orientation workspace, which is vital for the study of manipulators that provide purely rotational output. Instead of representing the orientation workspace in the Euclidean space, we utilise *quaternions* and differential geometry to parameterise an orientation workspace to study its various properties, such as its volume, i.e., the performance index on the range of motion of a manipulator. With our approach, we can make meaningful comparisons of the orientation workspace of manipulators that provide spherical motion, which would otherwise be challenging to derive using traditional methods in Euclidean space.

Constraint Analysis

In Chapter 18, we redirect our focus to soft robots, which are gaining ever-increasing attention in the field of robotics. We tackle the problem of analysing soft robots from the point of view of generalised underactuated mechanisms. By taking advantage of Lagrange multipliers, virtual bodies, and pseudo-rigid bodies, we allow the rigid-link kinematic- and force-domain analysis discussed in the previous part of the textbook to be adopted for planar underactuated systems with continuum bodies, by means of a unified framework of kinetostatic analysis.

Concentric Tube Robot

Chapter 19 discusses the modelling of concentric tube robots, a soft continuum soft robot with a minimalistic mechanical structure of super-elastic tubes. Such a structure, in combination with the compliance, makes the robot suitable for cannula-based minimally invasive surgery. To achieve computationally efficient modelling, our tube robot model is based on a generalised shape model for strands, with a robot-dependent part featuring static equilibrium equations to map the actuator inputs to the cross-section-wise local curvatures.

Advanced Path Planning

Chapter 20 introduces path planning for a different type of mechanism called *parallel manipulators*. Due to their unique kinematic properties, their workspace is often quite complex and full of singularities. As singularities pose a serious threat to their controllability, path planning for these types of manipulators can be quite challenging. This work proposes a method for representing the complexity of a parallel manipulator in a way that path planning can occur efficiently in the configuration space.

Remote Centre of Motion Robot

Chapter 21 looks at the manipulators for non-cannula-type minimally invasive surgery. A new remote centre of motion mechanism is proposed to help surgeons maintain the position of the insertion ports, hence maximising patient safety. Two design variants, based on a gear train and a hybrid link-cable system, respectively, are presented, and their capabilities to rotate around a remote fixed point in the space are proven mathematically.

The Monash Apple Retrieving System

Chapter 22 describes the design and testing of a highly successful apple harvesting robot developed in Dr Chao Chen's Laboratory of Motion Generation and Analysis. The robot's manipulator, vision system, soft gripper, kinematics, and path planning are described in this chapter, along with a comprehensive analysis of its harvesting performance in the field.

1.2 Robot Architectures

In this section, we introduce robot architectures commonly utilised in the manufacturing industry and robotic automation in general.

1.2.1 Cartesian

Cartesian manipulators, e.g., one depicted in Figure 1.2, feature a workspace in which the end-effector's orientation is kept constant while articulating in Cartesian directions. While various mechanical architectures can generate this type of workspace, the simplest form is a gantry-type architecture. This is made up of orthogonal prismatic actuators, connected from the base to the end-effector, representing the x, y, and z Cartesian directions. Each prismatic actuator independently controls the motion along a single axis, which means the forward and inverse kinematic solution is trivial. The gantry-type architecture is mechanically simple, thus reinforcing the structure along each axis by adding passive links between the output and the base is a simple task. The added advantage is that it increases payload strength and end-effector accuracy.

Cartesian manipulators have seen many applications in the industry, ranging from large-scale gantry cranes used in construction and assembly tasks to smaller-scale applications such as printers. Modern-day 3D printers are driven by the aforementioned mechanical advantages, which have revolutionised the way we conduct rapid prototyping (requires citation), giving rise to the term *additive manufacturing*. Latest printing technologies allow the printing of metallic material, which requires micrometre accuracy.

Cartesian manipulators are not without their flaws. Gantry-type structures with prismatic actuators are generally slow, especially in high-precision applications. In addition, the robot typically *envelops* the workspace, meaning the robot workspace volume cannot be increased without also increasing the volume of space occupied by the robot's structure. While this is not an issue for small-scale printers, retrofitting a large Cartesian robot in an industrial workspace can be very challenging if workspace volume is constrained.

1.2.2 Cylindrical

Cylindrical robots have three DoF at the end-effector, forming a cylindrical coordinate system. The robot is actuated by one revolute joint rotating normal to the base surface,

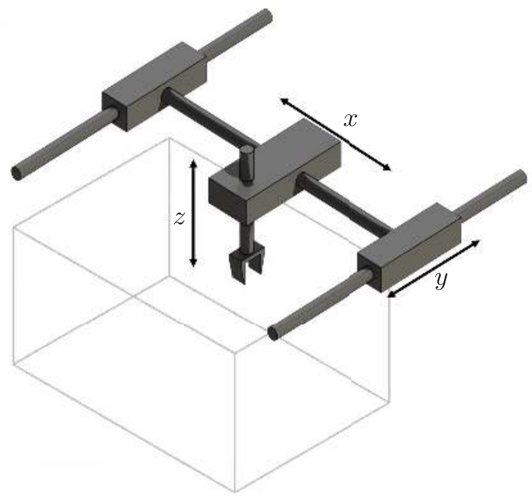


FIGURE 1.2
Cartesian robot workspace.

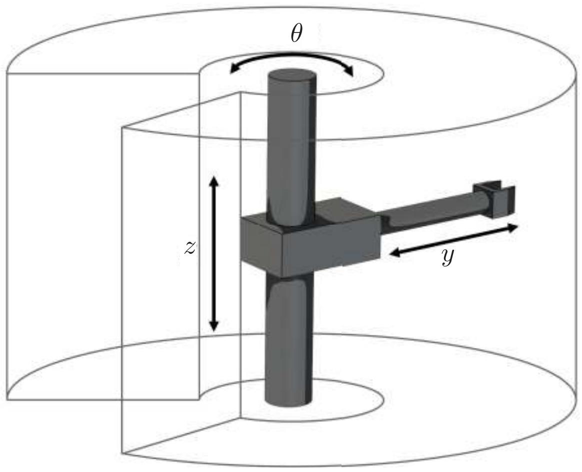


FIGURE 1.3
Cylindrical robot workspace.

and then two linear actuators to control the height and radius of the end-effector's position. As a result, this has a much more compact working area compared to Cartesian robots, albeit at a reduced workspace volume. However, this robot still exhibits high precision and stiffness at the end-effector.

This robot has common uses in pick-and-place and sorting tasks in smaller environments such as laboratories, assembly lines, or as a spot-welder on a relatively planar surface. A sample cylindrical robot and its workspace are illustrated in Figure 1.3.

1.2.3 Spherical

A spherical robot has three DoF at the end-effector, forming a polar coordinate system. It is actuated by two revolute joints at the shoulder to control roll and pitch, with a single linear actuator controlling the radial position of the end-effector. The original Unimate robot, installed in car manufacturing plants in the 1960s, was a spherical robot. As such, it has seen many industrial applications in machine tool manipulation, spot-welding, die casting, and gas and arc welding. A sample spherical robot is shown in Figure 1.4 along with its workspace.

It should be noted that to make full use of the orientation workspace, one more degree of freedom is added to the end of the robot, as seen by the Unimate industrial robot. These additional DoF are known as the *wrist* of the robot, which will be discussed further in this chapter.

1.2.4 SCARA

The SCARA architecture (Selective Compliance Assembly Robot Arm, Figure 1.5) consists of three parallel revolute joints in the main articulated arm, with a prismatic actuator attached to the end of the arm to make up a four-DoF robot. The revolute joints allow agile planar manipulation of the prismatic actuator, which controls the height of the end-effector normal to the workspace plane. The end-effector yields high stiffness, as the three revolute joints do not carry the robot's self-weight.

This robot has a workspace profile similar to that of a cylindrical robot. However, it has an extra degree of freedom to allow planar orientation of the end-effector. Also, it is mechanically very simple, allowing all actuators to be placed towards the base of the arm. This allows for planar motion with high stiffness and speed, which is very suitable for high-precision, quick pick-and-place tasks, such as sorting in a laboratory setting. It can also be used in applications where the workspace is mostly planar, such as drilling, milling, and engraving.

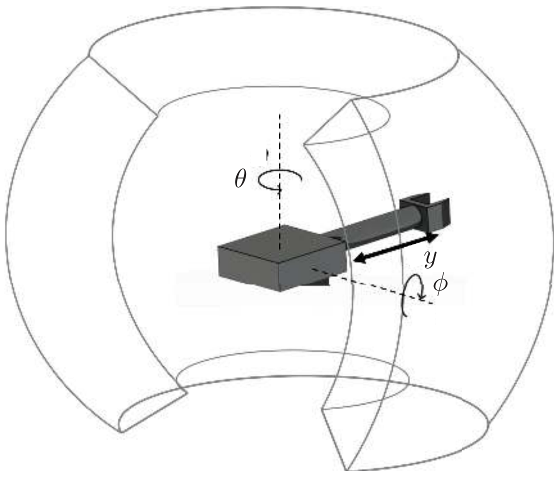


FIGURE 1.4
Spherical robot workspace.

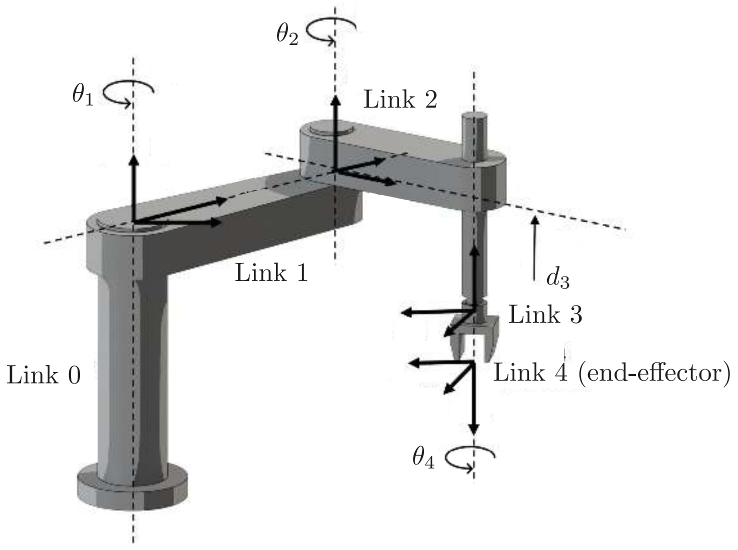


FIGURE 1.5
SCARA architecture.

1.2.5 Articulated

These manipulators consist of two orthogonal *shoulder* joints at the base, and a middle *elbow* joint with a link between the shoulder and elbow joints. The two shoulder actuators at the base control the direction and height of the elbow, and the elbow joint's axis of rotation is parallel to the second shoulder joint, which increases the articulated robot's reach. This architecture is usually implemented with a three-DoF wrist at the end of the elbow link, which allows rotational positioning of the end-effector. The result is a six-dimensional workspace where translational and rotational motion is decoupled, allowing for a simpler kinematic model. Furthermore, this six-DoF implementation yields multiple inverse kinematic solutions, allowing the arm a few options to avoid obstacles within the workspace, thus improving the dexterity of the end-effector under constrained workspaces. Although these robots lack the mechanical stiffness of Cartesian robots, they require much less structural space for the same volume of workspace, making them highly suitable for applications with smaller workspaces and payloads. The articulated architecture is also known as anthropomorphic architecture, because the kinematic configuration (with a three-DoF wrist) closely resembles the kinematic configuration of an anthropomorphic human arm. Figure 1.6 illustrates a sample articulated robot and its workspace.

The PUMA robot (Programmable Universal Machine for Assembly) falls under the category of articulated robots (Figure 1.7), and has seen many uses in the industry, typically in pick-and-place tasks, assembly, and welding. It is the spiritual successor of the famous Unimate robot (Figure 1.1), which revolutionised the manufacturing industry in the 1970s.

Currently, many robotic companies such as ABB, Kuka, and Universal Robots have engineered robotic arms of articulated architecture, and they are popular due to their versatility. Although each company has its own implementation of the articulated arm structure, the first three actuators of the arm, consisting of two shoulder and elbow actuators, are always at the core of each design. The anthropomorphic nature of these arms has put these robots at the forefront of many human-centred applications due to their naturalistic kinematic config-

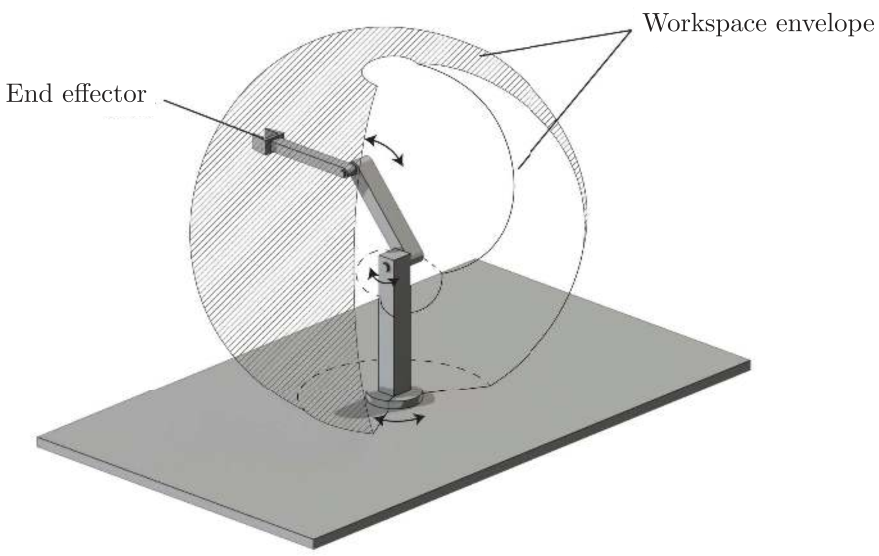


FIGURE 1.6
Articulated robot workspace.

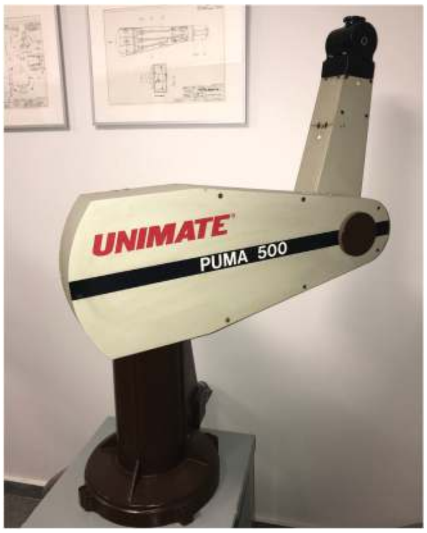


FIGURE 1.7
The PUMA robot.[4]

uration and workspace. This includes heavy-duty cleaning, painting, condition inspection, and even climbing applications. This is also the preferred architecture for human-robot collaborative environments for the aforementioned reasons and in part due to their naturalistic

[4]Guzman-Gimenez, J., Valera Fernandez, A., Mata Amela, V. and Díaz-Rodríguez, M.Á., 2023. Automatic selection of the Groebner Basis' monomial order employed for the synthesis of the inverse kinematic model of non-redundant open-chain robotic systems. Mechanics Based Design of Structures and Machines, 51(5), pp. 2458–2480.

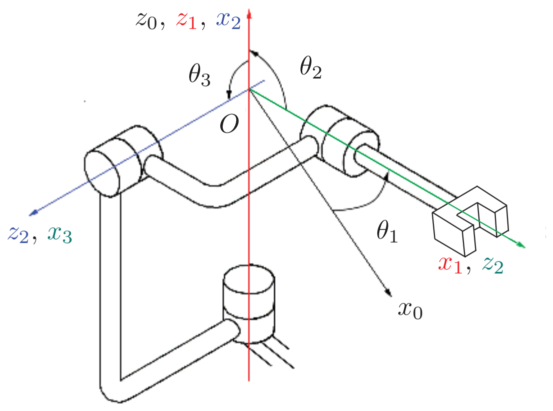


FIGURE 1.8
A spherical wrist with z-axes of rotation, all meeting at a common point O.

aesthetics. Human-robot collaboration is a hot topic in modern robotics research, and has been successfully trialled and implemented in environments such as weight-lifting assistance, rehabilitation, and education.

1.2.6 Wrists

While typical robots focus on the translational workspace, wrists are very important in realising the rotational workspace. Wrists are normally three-DoF, where the rotation axes are orthogonal, which guarantees any orientation of the end-effector can be achieved (assuming no joint limits). This configuration can be simplified in a two-DoF implementation, but a complete orientation workspace is no longer guaranteed. Because the revolute axes are orthogonal, it is very easy to solve the inverse kinematic solutions. Hence, a wrist can be easily attached to the end of an articulated robot without overly complicating its closed-form solutions.

A wrist is critical for allowing rotational motion of the end-effector. While the first three actuators of a robotic arm control the translational positioning of the end-effector, the next two or three actuators within the wrist control end-effector orientation. This is necessary for many manufacturing tasks in which the tool is orientation-constrained, such as welding and assembly.

Because the workspace is mostly rotational, the wrist design is typically more complicated than the rest of the arm. The strength and the number of DoF required at the wrist heavily influence the mechanical complexity. This can adversely affect joint limits, which are typically small and very challenging to maximise. For instance, the joint limits of a gimbal-type wrist are heavily constrained by its internal structure, and require some very clever engineering for only small gains in the joint limits. Alternative wrist configurations feature a serial chain arrangement, such as roll-pitch-roll, where all rotational axes still meet at a single point. This wrist configuration, as shown in Figure 1.8, exhibits an almost infinite joint limit, but suffers from lower strength and stiffness compared to the gimbal counterpart, and its home position is singular, which can cause problems with path planning in this configuration.

Serial chain wrists with non-intersecting rotation axes (non-spherical) can also be implemented. Although they are mechanically simpler with wider joint limits, a closed-form solution to its inverse kinematics is no longer guaranteed, which can affect its real-time performance as numerical solvers must be implemented. This configuration, however, solves the home position singularity problem. The articulated robotic arm by Universal Robots features a non-spherical wrist that has a very wide joint angle limit, but is also singularity-free at its home position and has closed-form solutions. This phenomenon is explained in further detail in Chapter 7.

1.3 Conclusion

In this chapter, we gave a brief overview of robotics in manufacturing, from its brief history to its current state, highlighting the importance of intelligent mechanical design in each industrial revolution. We then introduced the broad field of robotics, focussing on the mechanics and control of robots, which this textbook will cover. Key robotics terms are introduced, and each topic covered in this textbook is briefly introduced. Finally, robot architectures commonly seen in robotics and especially in manufacturing were introduced, where each architecture's strengths and common uses were analysed. These architectures will be consistently referred to in later chapters in problem sets and examples, to link the fundamental theorems of robotics introduced by this textbook back to the manufacturing industry.

1.4 Exercises

Problem 1. What is the degree of freedom of Cartesian manipulators?

Problem 2. The SCARA robot is known for its high payload capacity and agile planar movement. Considering how the joints are configured for this robot, why is the robot capable of such a high payload compared to an articulated architecture?

Problem 3. Is it possible to replicate a Cartesian manipulator with an articulated structure?

Problem 4. Referencing the previous question, can an articulated robot with a wrist attachment achieve the same motion as a pure Cartesian robot?

2

Planar Kinematics, Velocity, and Statics

This chapter will introduce the basic skills needed to define a two-dimensional robotic manipulator in space, analyse its pose and configurations in its workspace, and analyse its general motion. It will cover four main topics of analysis.

Transformations in Planar Space

To foster an understanding of the mechanics of robotic manipulation of objects in its workspace, we introduce the fundamental concepts of two-dimensional space, or *planar* space free space, that are relevant for robotic analysis. Using linear algebra techniques, we learn how to define rigid bodies in free space with the introduction of *pose* (position and orientation), and how to apply transformations (translation and rotation) on them, as if a robot were manipulating them.

Kinematics

Kinematics describes the pose of a robot's links and joints in free space at any given time, given a set of parameters. These parameters are usually joint angles, whereby a robot's kinematic equations allow us to convert a vector of joint angles to an end-effector pose. These kinematic equations also allow us to determine the joint angles of a robot, given a target end-effector pose. Deriving these kinematics equations is usually the first step in robotic modelling.

Velocity

This topic introduces motion into the analysis of the robotic system. It refers to the rate of change of both joint positions and the end-effector, in which point velocity (how fast a robot is moving in a straight line) and angular velocity (how fast a robot is rotating or turning) will be modelled. We will learn to derive the velocity equations of the end-effector from its position in the planar space. Velocity plays a crucial role in describing the speed and direction of movement for robots in navigation, manipulation of objects, and interaction with their environment effectively.

Jacobians and Statics

Deriving the velocity provides us with insight into the Jacobian matrix, which is a useful tool for static analysis, singularity analysis, and workspace analysis. Static analysis involves evaluating the external force exerted by the end-effector from the internal force and torque of the joints. Workspace analysis computes the region a robot can reach and operate within. Singularity analysis identifies and analyses configurations in a robot's workspace where the robot loses certain degrees of freedom or encounters problematic behaviour. This analysis helps engineers and researchers predict and understand how robots will behave under different conditions, loads, or constraints without physically testing them.

DOI: 10.1201/9781003614319-2

2.1 Two-Dimensional Space

Consider a two-dimensional space with a fixed frame defined at its origin. This frame has two orthogonal axes, $\mathbf{x}_0$ and $\mathbf{y}_0$, which defines the space's coordinate system. The subscript of the x and y-axes indicates that the axes belong to frame $\{0\}$. In general convention, we call the 0 frame the *reference frame*, *base frame*, or *world frame* if this frame represents the world in which our objects exist. Mathematically, we write this as $\{0\}$.

Now let us define a rigid body and attach a frame to it called $\{1\}$, which also has orthogonal axes $\mathbf{x}_1$ and $\mathbf{y}_1$. Finally, let us place the rigid body somewhere in our world frame $\{0\}$. To define where this rigid body is in the world, we ask ourselves: *where are* $\mathbf{x}_1$ *and* $\mathbf{y}_1$*, the x and y axes of our rigid body frame B, in relation to our world coordinate system's axes* $\mathbf{x}_0$ *and* $\mathbf{y}_0$? To answer this question, we are looking for ${}^0\mathbf{x}_1$ and ${}^0\mathbf{y}_1$, which means *x-axes and y-axes of* $\{1\}$*, observed in the* $\{0\}$ *coordinate system*, respectively. Typically, when we observe a quantity, we usually mean to *measure* it, so the previous statement could be reworded to *measured in the* $\{0\}$ *coordinate system.* To calculate the quantities ${}^0\mathbf{x}_1$ and ${}^0\mathbf{y}_1$, we can refer to elementary trigonometry and the dot product.

2.2 Transformations

Let us generalise the frame notation based on the above example, using $\{i\}$ and $\{j\}$ as our adjacent frames such that $j = i + 1$, getting our original notation if we set $i = 0$. A two-dimensional transformation matrix describes the *pose* of a frame, or a rigid body with a frame attached, measured in a reference frame. Both a rotation and translation are encoded into a single 3×3 square matrix of the form

$$ {}^i\mathbf{T}_j = \begin{bmatrix} {}^i\mathbf{x}_j & {}^i\mathbf{y}_j & {}^i\mathbf{p}_j \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & {}^ip_{jx} \\ \sin(\theta) & \cos(\theta) & {}^ip_{jy} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1} $$

where ${}^ip_{jx}$ and ${}^ip_{jy}$ represent the translation of O_j, the origin of $\{j\}$, measured along the x_i and y_i axes of the $\{i\}$, respectively, and θ is the angular displacement from $\{i\}$ to $\{j\}$. Figure 2.1 shows how each component of the matrix is physically represented. Note that frame $\{i'\}$ is parallel to $\{i\}$, introduced for the purpose of illustrating the angular displacement θ.

There are two main functions of a transformation matrix. It

- provides a full description of the rigid body's rotation and position, measured in an arbitrary coordinate system, and
- allows us to observe any rigid body in another coordinate system, as long as they are connected in the *transformation tree.*

The latter point is particularly useful in the analysis of robot motion.

2.2.1 Transforming a Vector

Let us begin with a trivial mechanical system consisting of just one rigid link, driven by a single motor at one end (Figure 2.2). The link, as it rotates, drives an external load at the opposite end. Let us define our world coordinate system as $\{0\}$ and place our motor at the

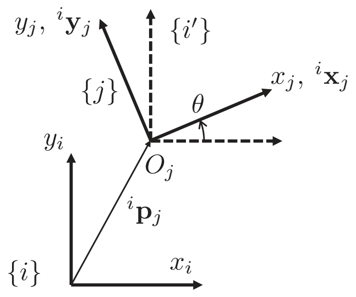


FIGURE 2.1
General transformation of two frames.

origin of this frame. On our rigid link, we will attach frame $\{1\}$ to its base where the motor is attached, where $\mathbf{x}_1$ points along the length of the link. Therefore, as the motor spins, $\{1\}$ will rotate about the origin of $\{0\}$ by an angle θ_1. The transformation matrix between $\{0\}$ and $\{1\}$, according to (2.1) is

$$ {}^0\mathbf{T}_1 = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & {}^0x_1 \\ \sin(\theta_1) & \cos(\theta_1) & {}^0y_1 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2} $$

$$ = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.3} $$

This merely tells us that the base of the rigid link is spinning about the motor's axis, which is intended! However, it is not particularly useful to us. Instead, we want to know the whereabouts of the opposite end of the link as the motor is driven. We should define a point on the rigid link where the load is driven, called E. For simplicity, let us assume the load is located at the opposite end of the rigid link coordinate system. In other words

$$ {}^1\mathbf{p}_E = \begin{bmatrix} l \\ 0 \end{bmatrix} \tag{2.4} $$

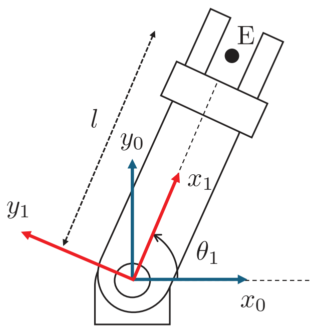


FIGURE 2.2
One-link planar robot.

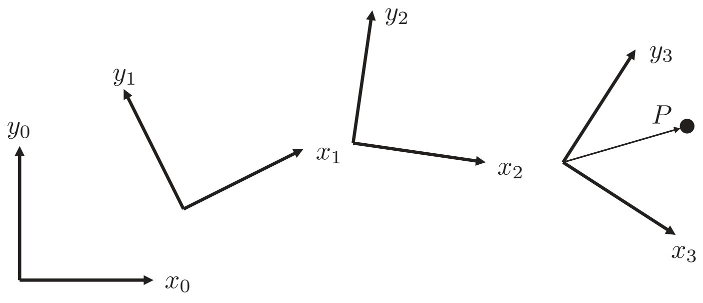


FIGURE 2.3
Transformation tree.

where l is the length of the rigid link which propagates along $\mathbf{x}_1$. To locate point E in the world frame $\{0\}$, we pre-multiply our vector ${}^1\mathbf{p}_E$ with the transformation matrix ${}^0\mathbf{T}_1$, such that

$$
\begin{aligned}
{}^0\mathbf{T}_1\,{}^1\mathbf{p}_E &= {}^0\mathbf{p}_E && (2.5)\\
&= \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^1\mathbf{p}_E \\ 1 \end{bmatrix} && (2.6)\\
&= \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l \\ 0 \\ 1 \end{bmatrix} && (2.7)\\
\begin{bmatrix} {}^0\mathbf{p}_E \\ 1 \end{bmatrix} &= \begin{bmatrix} l\cos(\theta_1) \\ l\sin(\theta_1) \\ 1 \end{bmatrix} && (2.8)
\end{aligned}
$$

Note that in Equation (2.6), we append a 1 to the vector to create a *homogeneous coordinate*. This is required to be able to multiply the vector with the 2D transformation matrix. The formal definition of the homogeneous coordinate is out of this scope in Part I, but is formally introduced in Part II, Section 5.5.

2.2.2 Transformation Trees

Often, we are interested in calculating the pose or position of an object in a reference coordinate system that is not adjacent to the target frame. However, as long as they are connected in a *transformation tree*, then a solution is feasible. A transformation tree is a set of transformation matrices that are linked in the same system. For example, we are given three frames, $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, and $\{4\}$ with the transformation matrices ${}^0\mathbf{T}_1$, ${}^0\mathbf{T}_2$, ${}^2\mathbf{T}_3$, and ${}^3\mathbf{T}_4$. Figure 2.3 represents this system's transformation tree.

To observe any frame measured in another frame in this transformation tree, we apply matrix multiplication, similar to the multiplication of rotation matrices. For example, if we want to observe $\{4\}$ in the $\{0\}$, we apply

$$
{}^0\mathbf{T}_4 = {}^0\mathbf{T}_2\,{}^2\mathbf{T}_3\,{}^2\mathbf{T}_4. \tag{2.9}
$$

We can also observe {4} and the {1} coordinate system as they are connected in the transformation tree. However, the above example implies we need ${}^{1}\mathbf{T}_0$, i.e.,

$$ {}^{1}\mathbf{T}_4 = {}^{1}\mathbf{T}_0 \, {}^{0}\mathbf{T}_2 \, {}^{2}\mathbf{T}_3 \, {}^{2}\mathbf{T}_4 \tag{2.10} $$

but we are only given ${}^{0}\mathbf{T}_1$. Therefore, we need to find the *inverse transformation* of ${}^{1}\mathbf{T}_0$. The generalised form of the inverse transformation matrix is

$$ {}^{i}\mathbf{T}_j = {}^{j}\mathbf{T}_i{}^{-1} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & -{}^{i}x_j\cos(\theta) - {}^{i}y_j\sin(\theta) \\ -\sin(\theta) & \cos(\theta) & {}^{i}x_j\sin(\theta) - {}^{i}y_j\cos(\theta) \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.11} $$

2.3 Planar Robot Kinematics

In this section, we will learn how to describe the position and orientation of a robotic end-effector in terms of the joint angles in a process called *forward kinematics*, otherwise known as direct kinematics. The position and orientation of a robotic end-effector constitute the task space, while the joint angles constitute the joint space. Hence, the forward kinematics is a mapping from the joint space to the direct space. Our focus will be on planar systems, where the robot's end-effector only moves along a plane with a single-parameter orientation.

2.3.1 Joints

There are two commonly used joints in robotic manipulators: revolute (R) joints and prismatic (P) joints, as shown in Figure 2.4. A revolute joint can be simplified into one axis of rotation, which completely defines the relative motion of these two rigid bodies. This axis of rotation is a spatial line with its position and direction. A prismatic joint defines the sliding direction of one body relative to another. A prismatic joint can be replaced by another parallel prismatic joint at a different position, while the relative motion of the two rigid bodies remains the same. Therefore, a prismatic joint can be characterised by a directional vector without a specific position.

In forward kinematics, the shape of a rigid body is of no importance; the only thing that matters is the types and locations of the joints among the bodies. Although there are

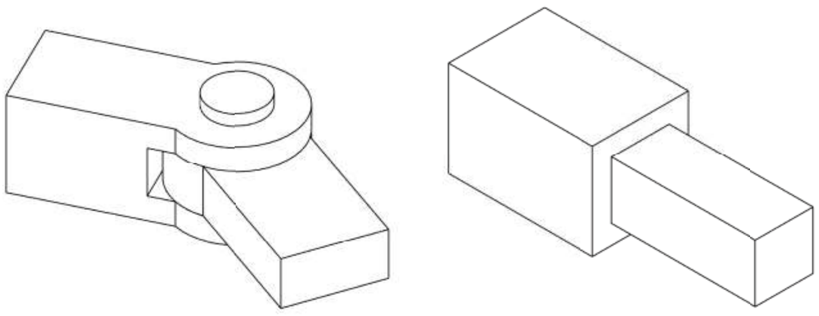

(a) Revolute (b) Prismatic

FIGURE 2.4
Two types of common robotic joints.

many types of actuated joints, our discussion will focus on the serial robotic manipulators with any combination of R and P joints, configured such that the end-effector achieves only planar movement.

In order to precisely describe the position and orientation of each moving link of a robotic manipulator, we desire to assign one frame to each link. Each link's position and orientation can be fully described by the transformation matrix between the ground frame $\{0\}$ and the attached frame to this link. Further, the transformation matrix between two adjacent links is only affected by a single parameter, either a rotation θ for an R joint, or a displacement d for a P joint between these two links.

2.3.2 Forward Kinematics

The process is similar to the one utilised in Section 2.2.1. For each link:

1. Assign the origin of the link's frame to its base, where its actuator will be attached.
2. Assign the x-axis of each frame, such that it points towards the origin of the next frame. If the frame represents a prismatic joint, then the x-axis represents the direction of motion.
3. Assign the y-axis to be 90° clockwise from the x-axis.
4. Assign the base frame, $\{0\}$ to be coincident with the first link's frame $\{1\}$. This means that $\{0\}$ and $\{1\}$ are coincident when θ_1 (or d_1 for prismatic joints) $= 0$.

The four steps are illustrated in Figure 2.5 (a) to (d), respectively.

Example 2.1 (Forward kinematics — 3R):

A planar robot is shown in Figure 2.6. Find the coordinates of Point P in the fixed frame. Assuming $l_1 = 0.25$ m, $l_2 = 0.30$ m, $l_3 = 0.20$ m, $\theta_1 = 15°$, $\theta_2 = 25°$, and $\theta_3 = 30°$.

Solution: The first step is to assign frames to all links, following the convention that the frame's origin is placed on the axis of rotation, with the x-axis pointing along the link towards the next frame's origin. The base frame, $\{0\}$ is determined by aligning it with $\{1\}$ when $\theta_1 = 0$. Therefore, the following transformation matrices are

$$ {}^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

$$ {}^1\mathbf{T}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & l_1 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

$$ {}^2\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & l_2 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

The total transformation matrix is

$$ {}^0\mathbf{T}_3 = {}^0\mathbf{T}_1\, {}^1\mathbf{T}_2\, {}^2\mathbf{T}_3 \tag{2.15}$$

For any given point ${}^3\mathbf{p}$ on the end-effector, its coordinates $\{0\}$ are given by

$$ \begin{bmatrix} {}^0\mathbf{p} \\ 1 \end{bmatrix} = {}^0\mathbf{T}_3 \begin{bmatrix} {}^3\mathbf{p} \\ 1 \end{bmatrix} \tag{2.16}$$

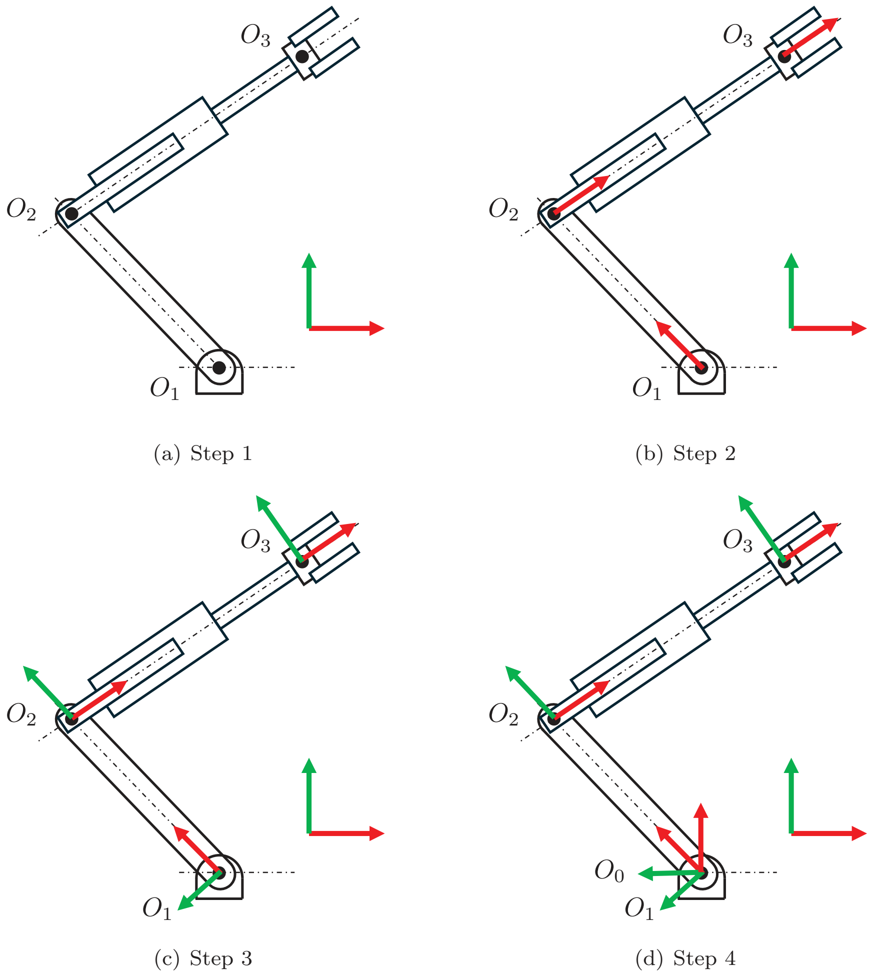


FIGURE 2.5
Frame assignments for a planar RRP robot.

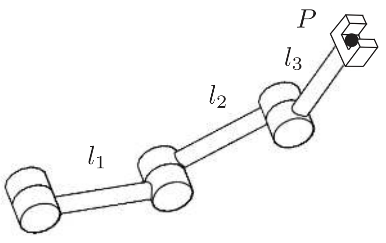


FIGURE 2.6
A planar RRR robot.

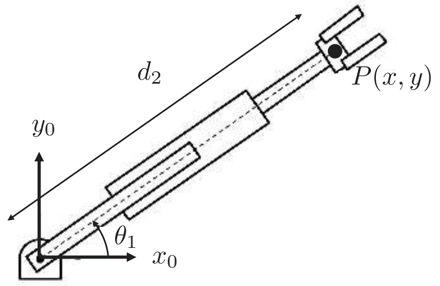


FIGURE 2.7
A planar RP robot.

with ${}^3\mathbf{p} = \begin{bmatrix} l_3 & 0 \end{bmatrix}^T$. Substituting the numerical design parameters yields

$$
{}^0\mathbf{T}_3 = \begin{bmatrix} 0.34 & -0.94 & 0.47 \\ 0.94 & 0.34 & 0.26 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}
$$

and

$$
{}^0\mathbf{p} = \begin{bmatrix} 0.54 \\ 0.45 \end{bmatrix} \tag{2.18}
$$

Example 2.2 (Forward kinematics — RP): A planar robot is shown in Figure 2.7. Find the coordinates of Point P in the fixed frame with $\theta_1 = 30^\circ$, $d_2 = 0.6$ mm, and ${}^2\mathbf{p} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$. **Solution:**

Starting from the base frame {0}, assign frame {1} with the x-axis pointing along the link towards the next frame's origin located at point P. Therefore, the following transformation matrices are

$$
{}^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}
$$

$$
{}^1\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & d_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.20}
$$

The total transformation matrix is

$$
{}^0\mathbf{T}_2 = {}^0\mathbf{T}_1 \, {}^1\mathbf{T}_2 \tag{2.21}
$$

For any given point ${}^2\mathbf{p}$ on the end-effector, its coordinates {0} are given by

$$
\begin{bmatrix} {}^0\mathbf{p} \\ 1 \end{bmatrix} = {}^0\mathbf{T}_2 \begin{bmatrix} {}^2\mathbf{p} \\ 1 \end{bmatrix} \tag{2.22}
$$

Substituting the numerical design parameters yields

$$
{}^0\mathbf{T}_3 = \begin{bmatrix} 0.87 & -0.50 & 0.52 \\ 0.50 & 0.87 & 0.30 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.23}
$$

and since ${}^2\mathbf{p} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, ${}^0\mathbf{p}$ can be extracted from the third column of ${}^0\mathbf{T}_3$, or

$$
{}^0\mathbf{p} = \begin{bmatrix} 0.52 \\ 0.30 \end{bmatrix} \tag{2.24}
$$

2.3.3 Inverse Kinematics

Inverse kinematics is the mathematical process of calculating the configurations of a robot's (or mechanism's) actuators to achieve a particular end-effector pose (position and orientation) relative to a coordinate system attached to the robot's base. This process is the reverse of forward kinematics, but it is generally much more challenging to solve.

There are two ways of solving the inverse kinematics of a robot: the algebraic method and the geometric method. While both methods have their strengths, we will focus on the geometric method for planar robots, which breaks the challenging problem down into an intuitive spatial geometry problem. The benefit of this method is that the methodology can be easily visualised as you are solving the problem, and that the solutions are derived from simple trigonometric equations.

As an introduction, we will utilise a simple 2R robot as seen in Figure 2.8. End effector pose is defined by an (x, y) coordinate, which is achieved by driving its two joint angles (θ_1, θ_2). In the inverse kinematic problem, we wish to solve for joint angles (θ_1, θ_2), given any arbitrary end-effector pose $({}^0x, {}^0y)$ that is within its workspace. Using the geometric method, we will break this problem down into a simple trigonometry geometry problem where we can use trigonometric identities to solve for various angles.

First, we can draw a straight line between the base of the robot to point P (Figure 2.8(a)), creating a triangle with l_1 and l_2. We can utilise the *cosine rule* to calculate the angle α in Figure 2.8(b):

$$x_P{}^2 + y_P{}^2 = l_1{}^2 + l_2{}^2 - 2l_1l_2\cos(\alpha) \tag{2.25}$$

By inspection, we can see that

$$\theta_2 = \pi - \alpha \tag{2.26}$$

and so (2.25) becomes

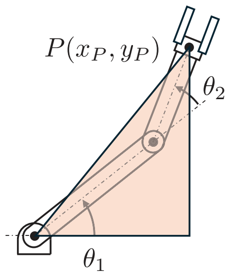
$$x_P{}^2 + y_P{}^2 = l_1{}^2 + l_2{}^2 - 2l_1l_2\cos(180 + \theta_2) \tag{2.27}$$

Because $\cos(180 + \theta_2) = -\cos(\theta_2)$, rearranging, we have

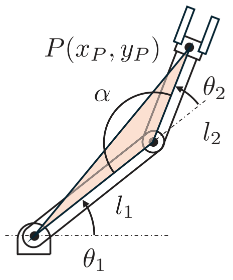
$$\cos(\theta_2) = \frac{x_P{}^2 + y_P{}^2 - l_1{}^2 - l_2{}^2}{2l_1l_2} \tag{2.28}$$

Therefore,

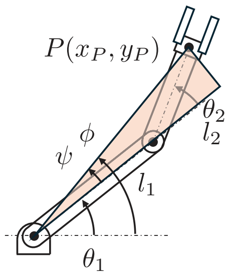
$$\theta_2 = \arccos\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2}\right) \tag{2.29}$$



(a) Step 1



(b) Step 2



(c) Step 3

FIGURE 2.8
Steps in inverse kinematics, using the geometric method.

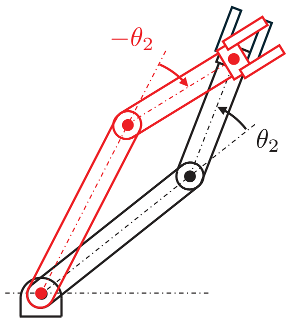


FIGURE 2.9
Kinematic configurations of the 2R.

There are three possible solutions to this equation: $\pm\theta_2$, or no solution.[1] In addition, we must impose the constraint $-\pi \leq \theta_2 \leq 0$ such that the triangle in Figure 2.8(a) can physically exist. To solve for θ_1, we need to solve for angles ψ and ϕ. First, ϕ can be solved using the two-argument arctangent

$$\phi = \text{Atan2}(y_P, x_P) \tag{2.30}$$

then applying the law of cosines again to find ψ

$$\cos(\psi) = \frac{{x_P}^2 + {y_P}^2 - {l_1}^2 - {l_2}^2}{2l_1\sqrt{{x_P}^2 + {y_P}^2}} \tag{2.31}$$

Here, the arccosine must be solved such that $0 \leq \psi \leq \pi$ such that the geometric solution remains physically viable. Finally, θ_1 can be solved with the expression

$$\theta_1 = \phi \pm \psi, \tag{2.32}$$

where the plus sign is used if $\theta_2 < 0$, and minus if $\theta_2 > 0$

2.3.3.1 Kinematic Configurations

In the process of solving the inverse kinematics for a 2R robot, we encountered a situation where two solutions are physically possible: $\pm\theta_2$, which leads to two different solutions for $\pm\theta_1$. When there are multiple solutions for the same end-effector position (or pose), we call each solution a *kinematic configuration*. In the case of our 2R robot, this represents an elbow up or elbow down configuration, as seen in Figure 2.9.

Although this makes for a less straightforward exercise in solving the inverse kinematics, robots do benefit from having multiple kinematic configurations. This feature allows the robot to reach a target in multiple ways, which is useful when its workspace is cluttered with obstacles. For example, if a particular end-effector pose causes the robot to collide with an object, then we can utilise an alternate kinematic configuration to avoid the obstacle (Figure 2.10).

Example 2.3 (Inverse kinematics — RP): Solve the inverse kinematics of the planar RP robot of Figure 2.11 using the geometric method; that is, find equations for θ_1 and d_2 in terms of the arbitrary location of the end-effector (x,y).

[1]What conditions are there no solutions to this equation, and what does that physically mean for the robot?

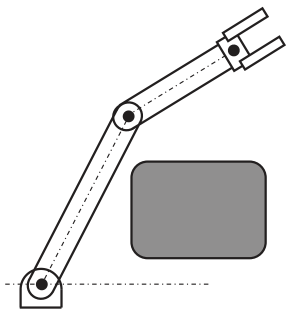


FIGURE 2.10
Utilising alternate kinematic configurations to avoid a collision.

Solution: Both θ_1 and d_2 can be solved by applying a red triangle onto the robot as shown in Figure 2.12, where

$$\theta_1 = \arctan \frac{y}{x} \tag{2.33}$$

$$d_2 = \sqrt{x^2 + y^2} \tag{2.34}$$

2.4 Velocity Analysis

Thus far, our attention has been focused on static manipulator poses, calculating where the end-effector is for a given actuator position, and calculating the actuator positions for a given end-effector pose. Now, we will explore robot motion — the linear and angular velocities of rigid bodies that make up the manipulator and resultant end-effector velocities, and use these concepts to determine the static forces of a manipulator.

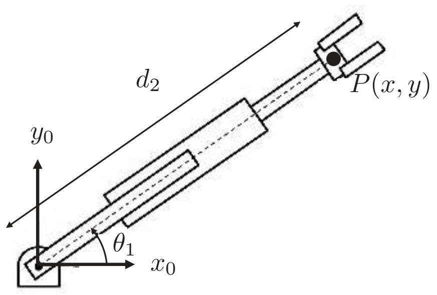


FIGURE 2.11
A planar RP robot.

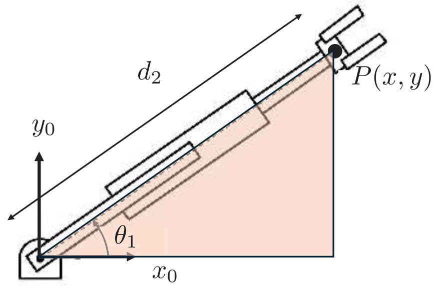


FIGURE 2.12
A planar RP robot (solution).

There are two commonly used methods for calculating the end-effector velocity of a manipulator for a given pose: time derivative of the end-effector output equations, and velocity propagation. In this chapter, we will focus on the time derivative method, with the velocity propagation method to be introduced in Section 8.3.

The study of rigid body motion in robotics is an important precursor to understanding the mechanical behaviour of a mechanical robot system, including its workspace, mobility, and dynamics. Once these are understood, we can apply established control methods to govern the robot's stable motion.

2.4.1 Linear and Angular End Effector Velocity

In Figure 2.13, we have a two-joint RR planar robot. The position of the end-effector in two-dimensional space measured in $\{0\}$, obtained by performing forward kinematics, is given as:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_2\cos(\theta_1+\theta_2) + l_1\cos\theta_1 \\ l_2\sin(\theta_1+\theta_2) + l_1\sin\theta_1 \end{bmatrix} \tag{2.35}$$

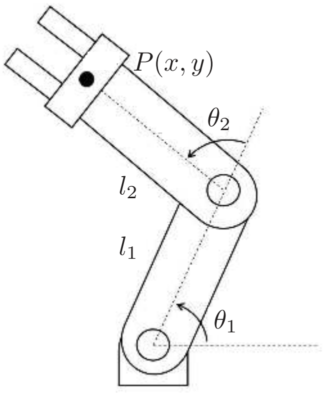


FIGURE 2.13
A planar RR robot.

The joint angles (θ_1 and θ_2) change with time as the joints rotate. Therefore, θ_1 and θ_2 can be expressed as functions with respect to time:

$$\theta_1 = \theta_1(t) \qquad \theta_2 = \theta_2(t) \tag{2.36}$$

Since x and y are functions of θ_1 and θ_2, we can compute the derivative of x and y with respect to time using the chain rule. Putting them into a vector represents the velocity of the end-effector in the Cartesian space

$$\dot{\theta}_i = \frac{d(\theta_i)}{dt} \tag{2.37}$$

$$\mathbf{v} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -l_1\dot{\theta}_1 \sin(\theta_1) - l_2(\dot{\theta}_1 + \dot{\theta}_2)\sin(\theta_1 + \theta_2) \\ l_1\dot{\theta}_1 \cos(\theta_1) + l_2(\dot{\theta}_1 + \dot{\theta}_2)\cos(\theta_1 + \theta_2) \end{bmatrix} \tag{2.38}$$

Example 2.4 (Velocity analysis): A planar robot is shown in Figure 2.14. Given that the total transformation is

$$^0\mathbf{T}_3 = \begin{bmatrix} c_{123} & -s_{123} & l_2c_{12} + l_1c_1 \\ s_{123} & c_{123} & l_2s_{12} + l_1s_1 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.39}$$

where $s_i = \sin\theta_i$, $c_i = \cos\theta_i$, $s_{ij} = \sin(\theta_i + \theta_j)$, and $c_{ij} = \cos(\theta_i + \theta_j)$. Further, θ_i for $i = 1, 2, 3$ are joint angles of this robot. Find the velocity of Point P in the fixed frame with the robot design parameters given in Example 2.1, and joint velocities $\dot{\theta}_1 = \dot{\theta}_2 = \dot{\theta}_3 = 10\ ^\circ/s$

Solution: For any given point $^3\mathbf{p}$ on the end-effector, its coordinates 0 are given by

$$\begin{bmatrix} ^0\mathbf{p} \\ 1 \end{bmatrix} = {^0\mathbf{T}_3} \begin{bmatrix} ^3\mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} l_3c_{123} + l_2c_{12} + l_1c_1 \\ l_3s_{123} + l_2s_{12} + l_1s_1 \\ 1 \end{bmatrix} \tag{2.40}$$

The velocity is the time derivative of the position. Therefore,

$$^0\mathbf{v} = {^0\dot{\mathbf{p}}} = \begin{bmatrix} -(\dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3)l_3s_{123} - (\dot{\theta}_1 + \dot{\theta}_2)l_2s_{12} - \dot{\theta}_1l_1s_1 \\ (\dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3)l_3c_{123} + (\dot{\theta}_1 + \dot{\theta}_2)l_2c_{12} + \dot{\theta}_1l_1c_1 \end{bmatrix} \tag{2.41}$$

Substituting the numerical values results in

$$^0\mathbf{v} = \begin{bmatrix} -0.18 \\ 0.16 \end{bmatrix} \tag{2.42}$$

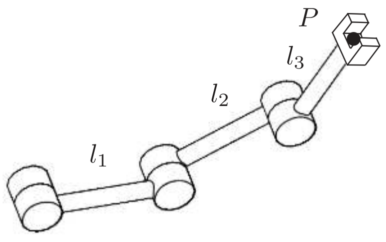
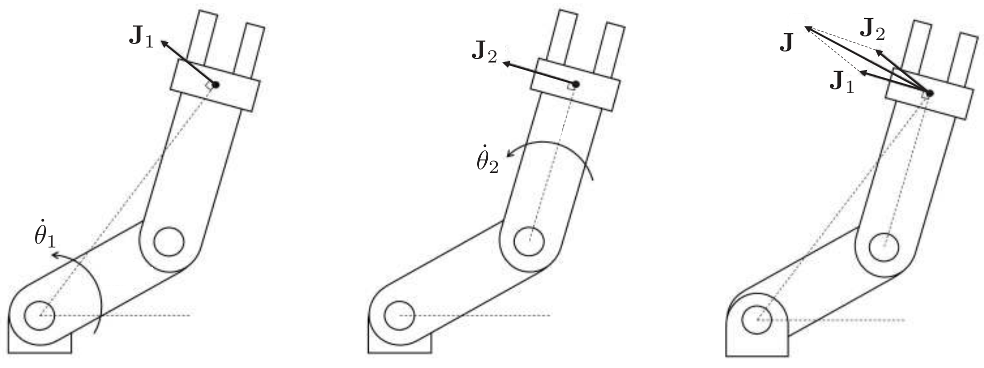


FIGURE 2.14
A planar RRR robot.



(a) Effect of $\mathbf{J}_1$. (b) Effect of $\mathbf{J}_2$. (c) Effect of $\mathbf{J}$.

FIGURE 2.15
Visualisation of Jacobians on an RR manipulator.

2.4.2 Jacobian

A Jacobian is the matrix equivalent of the derivative that maps the joint velocities to the velocity of the end-effector. Think of a robot arm with multiple joints; the Jacobian matrix helps to compute how a small change in each joint's position or angle affects the overall movement of the end-effector. It essentially tells us how the speed and direction of each joint impact the speed and direction of the end of the robot arm. It is also a useful tool in statics analysis that indirectly maps the motor torque inputs and the force and torque output of the end-effector.

Consider the end-effector velocity of the RR robot as derived in (2.38). We can see that the point velocity $\mathbf{v}$ is a function of the joint velocity ($\dot{\theta}_i$), thus we can rewrite the expression as

$$\mathbf{v} = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} \end{bmatrix} \dot{\theta}_1 + \begin{bmatrix} -l_2 s_{12} \\ l_2 c_{12} \end{bmatrix} \dot{\theta}_2 = \mathbf{J}_1(\mathbf{q})\dot{\theta}_1 + \mathbf{J}_2(\mathbf{q})\dot{\theta}_2 \tag{2.43}$$

where $\mathbf{q}$ is a generalised vector comprising all joint variables of the robot (i.e., θ, d). In this example, $\mathbf{q} = \begin{bmatrix} \theta_1 & \theta_2 \end{bmatrix}$. $\mathbf{J}_1(\mathbf{q})$ can be visualised as a vector velocity of the end-effector that is orthogonal to the line connecting from joint 1 to the end-effector when joint 2 is kept constant as shown in Figure 2.15(a). Similarly, $\mathbf{J}_2(\mathbf{q})$ can be visualised as a vector velocity of the end-effector that is orthogonal to the line connecting from joint 2 to the end-effector when joint 1 is kept constant as shown in Figure 2.15(b).

Furthermore, we can express (2.43) in the form

$$\mathbf{v} = \begin{bmatrix} \mathbf{J}_1(\mathbf{q}) & \mathbf{J}_2(\mathbf{q}) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{J}_1(\mathbf{q}) & \mathbf{J}_2(\mathbf{q}) \end{bmatrix} \dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{2.44}$$

where $\mathbf{J}(\mathbf{q})$ is the Jacobian matrix, a linear combination of $\mathbf{J}_1(\mathbf{q})$ and $\mathbf{J}_2(\mathbf{q})$ that maps the joint velocities to the velocity of the end-effector. Therefore, the Jacobian matrix for the RR robot, expressed in the form (2.44), is

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix} \tag{2.45}$$

Example 2.5 (Jacobian): For the same planar robot in Figure 2.14, find the Jacobian matrix in the fixed frame.

Solution: From Example 2.4, we have found that the velocity is

$$ {}^{0}\mathbf{v} = \begin{bmatrix} -(\dot\theta_1+\dot\theta_2+\dot\theta_3)l_3s_{123} - (\dot\theta_1+\dot\theta_2)l_2s_{12} - \dot\theta_1 l_1 s_1 \\ (\dot\theta_1+\dot\theta_2+\dot\theta_3)l_3c_{123} + (\dot\theta_1+\dot\theta_2)l_2c_{12} + \dot\theta_1 l_1 c_1 \end{bmatrix} \tag{2.46}$$

Now, factor out the joint velocities to obtain a term for each joint

$$ {}^{0}\mathbf{v} = \mathbf{J}_1\dot\theta_1 + \mathbf{J}_2\dot\theta_2 + \mathbf{J}_3\dot\theta_3 \tag{2.47}$$

where

$$\mathbf{J}_1 = \begin{bmatrix} -l_3s_{123} - l_2s_{12} - l_1s_1 \\ l_3c_{123} + l_2c_{12} + l_1c_1 \end{bmatrix} \tag{2.48}$$

$$\mathbf{J}_2 = \begin{bmatrix} -l_3s_{123} - l_2s_{12} \\ l_3c_{123} + l_2c_{12} \end{bmatrix} \tag{2.49}$$

$$\mathbf{J}_3 = \begin{bmatrix} -l_3s_{123} \\ l_3c_{123} \end{bmatrix} \tag{2.50}$$

The Jacobian matrix is the linear combination of $\mathbf{J}_1$, $\mathbf{J}_2$, and $\mathbf{J}_3$,

$$\mathbf{J} = \begin{bmatrix}\mathbf{J}_1 & \mathbf{J}_2 & \mathbf{J}_3\end{bmatrix} = \begin{bmatrix} 0.45 & -0.38 & -0.19 \\ 0.54 & 0.30 & 0.07 \end{bmatrix} \tag{2.51}$$

2.4.3 Singularity

A *singularity* in robotics is a condition in which the pose of a robot results in the end-effector motion being compromised. Such an example can be seen in Figure 2.16, where joint angle $\theta_2 = 0$, causing alignment of both $\mathbf{J}_1$ and $\mathbf{J}_2$. In this scenario, the robot loses its degree of freedom where the end-effector can only move in a single vector velocity that is along the line perpendicular to the arm link. Furthermore, unique solutions for $\dot\theta_1$ and $\dot\theta_2$ cannot be solved because there is an infinite combination of $\dot\theta_1$ and $\dot\theta_2$ that can result in a single point velocity of the end-effector.

Studying (2.45) further, in the singularity configuration, we notice that the columns of this matrix become scalar multiples of each other. This is known as a *rank-deficient* matrix, where two or more columns are scalar multiples of each other. When this happens, the *determinant* of the Jacobian matrix is equal to zero ($\det(\mathbf{J}) = 0$).

Example 2.6 (Singularity): Find and give a physical interpretation of the singularities (if any) of the RR robot with the following Jacobian:

$$\mathbf{J} = \begin{bmatrix} -l_1\sin(\theta_1) - l_2\sin(\theta_1+\theta_2) & -l_2\sin(\theta_1+\theta_2) \\ l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) & l_2\cos(\theta_1+\theta_2) \end{bmatrix} \tag{2.52}$$

Solution: Singularity occurs when the determinant of the Jacobian is zero at a particular configuration, such that

$$\det(\mathbf{J}) = l_1l_2(\cos(\theta_1)\sin(\theta_1+\theta_2) - \sin(\theta_1)\cos(\theta_1+\theta_2)) = 0 \tag{2.53}$$

and this happens if and only if θ_2 is equal to 0 or π.

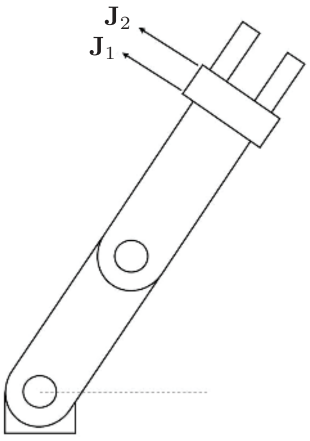


FIGURE 2.16
A planar RR robot with derived velocity.

2.5 Statics Analysis

In addition to the velocity relationship, we are also interested in developing a relationship between the robot joint torques and forces and the force exerted by the end-effector, which is known as static. In physics, we know that the power exerted by the end-effector is the multiplication between the exerted force ($\mathbf{f}$) and its velocity ($\mathbf{v}$). The same power comes from the torque produced from the joints, which can be expressed as the joint velocity ($\dot{\mathbf{q}}$) times the joint torque ($\boldsymbol{\tau}$). Therefore, we have

$$\dot{\mathbf{q}}^T \boldsymbol{\tau} = \mathbf{v}^T \mathbf{f} \tag{2.54}$$

Since the point velocity can be mapped to the joint velocity with the Jacobian matrix, we now have

$$\dot{\mathbf{q}}^T \boldsymbol{\tau} = (\mathbf{J}\, \dot{\mathbf{q}})^T \mathbf{f} \tag{2.55}$$

As we further simplify the equation, we can see that the joint torque can be mapped directly to the force exerted from the end-effector with the transpose of the Jacobian matrix

$$\boldsymbol{\tau} = \mathbf{J}^T \mathbf{f} \tag{2.56}$$

This relationship is useful in force control, as we can design the required torque from each joint for a desired force at the tip of the end-effector at a particular configuration.

Example 2.7 (Statics): Find the joint torques required by the robot in Figure 2.14 to maintain a static force vector ${}^0\mathbf{f} = \begin{bmatrix}1 & 1\end{bmatrix}^T$, when $\theta_1 = 45^\circ, \theta_2 = 45^\circ$, and $\theta_1 = 0^\circ$. Adopt the design parameters l_1, l_2, and l_3 from the previous examples.

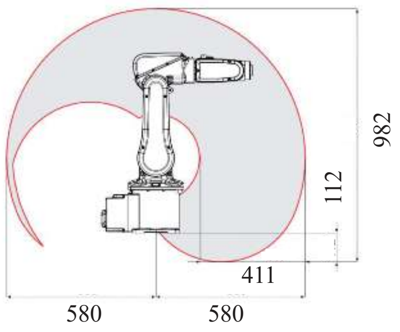


FIGURE 2.17
ABB IRB 120 six-axis industrial robotic arm workspace.[2]

Solution: From Example 2.5, we found that the Jacobian of the planar robot is

$$\begin{aligned}\mathbf{J} &= \begin{bmatrix} -l_3s_{123} - l_2s_{12} - l_1s_1 & -l_3s_{123} - l_2s_{12} & -l_3s_{123} \\ l_3c_{123} + l_2c_{12} + l_1c_1 & l_3c_{123} + l_2c_{12} & l_3c_{123} \end{bmatrix} \\ &= \begin{bmatrix} -0.68 & -0.50 & -0.20 \\ 0.18 & 0 & 0 \end{bmatrix}\end{aligned} \tag{2.57}$$

The transpose of the Jacobian can be used to map static force vectors to joint torques, such that

$$\boldsymbol{\tau} = \mathbf{J}^T\ \mathbf{f} \tag{2.58}$$

Therefore,

$$\boldsymbol{\tau} = \begin{bmatrix} -l_3s_{123} - l_2s_{12} - l_1s_1 & l_3c_{123} + l_2c_{12} + l_1c_1 \\ -l_3s_{123} - l_2s_{12} & l_3c_{123} + l_2c_{12} \\ -l_3s_{123} & l_3c_{123} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.59}$$

Substituting $\theta_1 = 45°, \theta_2 = 45°$, and $\theta_1 = 0°$, we get

$$\boldsymbol{\tau} = \begin{bmatrix} -l_3 - l_2 \\ -l_3 - l_2 \\ -l_3 \end{bmatrix} = \begin{bmatrix} -0.50 \\ -0.50 \\ -0.20 \end{bmatrix} \tag{2.60}$$

This means that joint 1 and 2 require a torque of -0.50 Nm, whilst joint 3 requires a torque of -0.20 Nm.

2.6 Workspace

In robotics, the workspace refers to the specific physical area or region where a robot can perform its tasks effectively. It is a concept that's crucially important in designing and programming robots, especially when considering their movement, range, and capabilities. Here, Figure 2.17 shows the workspace of the ABB IRB 120 six-axis industrial robotic arm.

[2]https://library.e.abb.com/public/7aa0711a20fa41c49a8fdf3bbc3d5bb0/IRB120-Rev.J-ROB0149EN_D.pdf (Retrieved 18.9.2024)

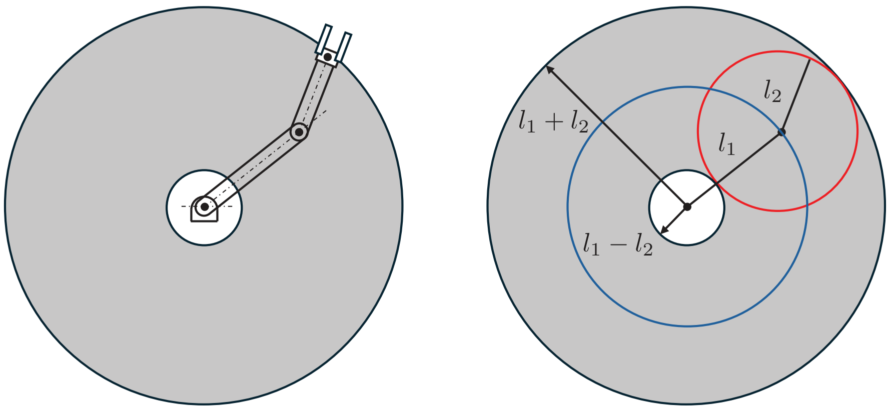


FIGURE 2.18
Planar RR robot workspace.

The area highlighted in grey represents the reachable area by the end-effector of the robot, which is defined by the robot's maximal reach (outer boundary), and self-imposed joint limits to avoid self-collision (inner boundary). Any points outside the grey area, the end-effector will not be able to reach, and there should exist no solution for inverse kinematics.

For our planar 2R robot, the workspace is shown in Figure 2.18. The blue circle denotes the path where the first link is able to move, whilst the red circle centre around the endpoint of the first link denotes the path where the second link can move. With these two paths combined, we have the workspace of the robot, illustrated in grey. The maximum boundary of the workspace is reached when the robotic arm is fully stretched ($l_1 + l_2$). The minimum boundary of the workspace is achieved by folding the second link inwards, such that both links are parallel to each other ($l_2 - l_1$). To mathematically express the workspace boundary, we can say that

$$(l_1 - l_2) \leq x^2 + y^2 \leq (l_1 + l_2) \tag{2.61}$$

where x and y indicate the position of the end-effector in Cartesian space.

2.7 Conclusion

In this chapter, we learnt about planar kinematics, velocity, and statics for basic robot analysis. In planar kinematics, we learnt to describe the position and orientation of a frame or a rigid body from a reference frame using a two-dimensional transformation matrix. The transformation matrix is used to map a point on a rigid link from one frame to another. Multiple matrices can form a transformation tree that correlates adjacent frames. We introduced two concepts in planar robot kinematics, which are forward and inverse kinematics, respectively. Forward kinematics finds the transformation matrix that represents the cumulative effect of all joint movements on the end-effector 's position and orientation. Inverse kinematics determines the joint configurations of a robotic system that will result in a

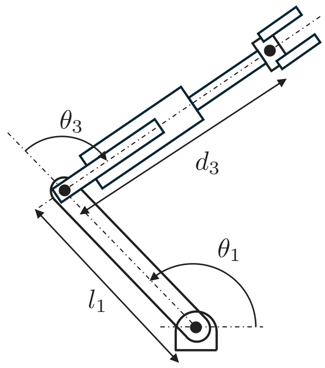


FIGURE 2.19
RRP manipulator.

specific position and orientation of its end-effector. This chapter focuses on using the geometric method for inverse kinematics.

Following that, we learnt to obtain the velocity of the robot manipulator from its position using the time derivative method. The joint space and the Cartesian space velocities can be mapped with the Jacobian matrix. We discussed how to use the Jacobian matrix to perform static analysis that provides a mapping between the joint load and the load acting on the end-effector in a stationary position. We introduced the concept of singularity and how to find singularity in a robotic system using the Jacobian matrix. Lastly, we introduced the concept of workspace and the basic analysis of the workspace of a robot.

2.8 Exercises

Problem 1. Define ${}^0\mathbf{T}_1$ that translate 5 units and 10 units along the x and y axes, and rotates by 30° in angular displacement. Then, use ${}^0\mathbf{T}_1$ to map a vector $\begin{bmatrix}1 & 2 & 0\end{bmatrix}^T$ from 1 to 0.

Problem 2. Find the ${}^1\mathbf{T}_3$ by using the following frame definitions:

$$
{}^1\mathbf{T}_0 = \begin{bmatrix} 0 & -1 & -10 \\ 1 & 0 & 10 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^3\mathbf{T}_0 = \begin{bmatrix} 0.5 & -0.866 & 10 \\ 0.866 & 0.5 & 8 \\ 0 & 0 & 1 \end{bmatrix}
$$

Problem 3. A RRP planar robot is shown in Figure 2.19. Solve the inverse kinematics of the robot using the geometric method; that is, find equations for θ_1, θ_2, and d_3 in terms of the arbitrary location of the end-effector (x, y).

Problem 4. Derive the point velocity and angular velocity, represented in $\{0\}$ (frame 0) of the end-effector of the three-link manipulator shown in Figure 2.20 by using the **time derivative** of the point position.

The transformation matrix ${}^0T_3$ is given by:

$$
{}^0T_3 = \begin{bmatrix} c_{123} & -s_{123} & l_1c_1 + l_2c_{12} \\ s_{123} & c_{123} & l_1s_1 + l_2s_{12} \\ 0 & 0 & 1 \end{bmatrix}
$$

where c and s represent cosine and sine, respectively, and multi-digit subscripts indicate the summation of angles.

Problem 5. Derive the **Jacobian** mapping the joint velocity inputs to the point velocity of the end-effector tips for the three-link manipulator of Figure 2.20. Represent the Jacobian in $\{0\}$ attached to the base.

Problem 6. What happens to the rank of a square Jacobian matrix under singularity configuration? Under that assumption, would singularities in the force domain exist in the same configuration as singularities in the position domain? Explain the physical meaning of a singularity in the force domain.

Problem 7. Find the joint torques required by the robot in Figure 2.20 to maintain a static force vector ${}^0\mathbf{f} = \begin{bmatrix} f_x & f_y \end{bmatrix}^T$, represent the result as a matrix equation.

Problem 8. Find the symbolic equation which represents the workspace boundary of the robot in Figure 2.20.

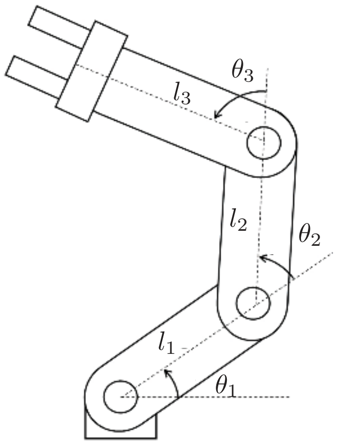


FIGURE 2.20
RRR manipulator.

3 Trajectory Generation

In this chapter, we introduce basic methods for computing trajectories to achieve the desired motion of a manipulator. A trajectory consists of a set of time-stamped instructions for each actuator in a robot to achieve a certain end-effector motion. However, a trajectory must also be feasible for a manipulator to execute. The minimum requirements for a smooth and feasible trajectory include:

- Positional requirements
- Continuous position profile
- Continuous velocity
- Maximum acceleration does not exceed payload ratings on each actuator

Trajectory generation is an essential function in industrial robots used in applications such as welding, painting, pick-and-place, and assembly. All of these applications require a degree of speed and precision. Therefore, the robot's trajectory must be smooth and continuous to maintain a standard of quality while minimising wear and tear.

Here, we introduce two methods of trajectory generation: polynomial interpolation, and cubic splines. While both are viable trajectory generation methods, one method may be favoured over the other, depending on the required trajectory constraints. In both methods, we treat a trajectory as a continuous time-dependent function that is calculated independently *for a single actuator*. Therefore, to achieve the desired end-effector trajectory, we execute the trajectory for each actuator simultaneously.

Trajectory generation methods are generalisable for both revolute and prismatic joints; hence, we will use the generalised variable q_i to represent the position of the robot's i-th joint. Also, note that references to the variable T in this chapter represent a time quantity rather than a transformation matrix.

3.1 Interpolation with Polynomials

Example 3.1 (Cubic splines): Figure 3.1 shows a trivial robot with only one R joint, which can be called the R robot. We want to command the robot to move the end-effector from a start joint angle q_s at time T_s to the final joint angle q_f at time T_f. The desired start and final angular velocities are $\dot{q}_s$ and $\dot{q}_f$, respectively. Find the smooth function to describe the trajectory of this joint angle, satisfying the above boundary conditions.

Solution: Assume the smooth function we seek is $q(t)$. There are infinitely many functions that can be used to generate this trajectory. Generally, polynomials are convenient for this work because they are infinitely differentiable. The required order of a polynomial depends on the number of

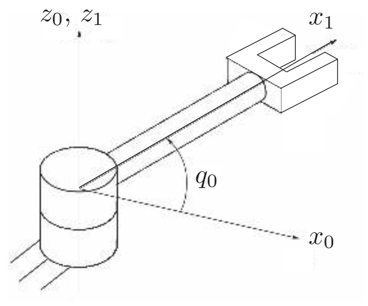


FIGURE 3.1
A single R robot.

conditions to be satisfied. In this problem, there are four conditions: the desired positions at the start and final moments, and the desired velocities at the start and final moments

$$q(T_s) = q_s, \qquad q(T_f) = q_f, \qquad \dot{q}(T_s) = \dot{q}_s, \qquad \dot{q}(T_f) = \dot{q}_f \tag{3.1}$$

Equations in (3.1) represent four constraints, while the variables are the coefficients of the polynomial at hand. To have exact solutions, four coefficients are required, which can be found in cubic polynomial

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{3.2}$$

The time derivative of (3.2) yields the velocity profile, given by

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{3.3}$$

Substituting (3.2) and (3.3) into (3.1) yields four linear equations below

$$q(T_s) = a_0 + a_1 T_s + a_2 T_s^2 + a_3 T_s^3 = q_s \tag{3.4}$$
$$q(T_f) = a_0 + a_1 T_f + a_2 T_f^2 + a_3 T_f^3 = q_f \tag{3.5}$$
$$\dot{q}(T_s) = a_1 + 2a_2 T_s + 3a_3 T_s^2 = \dot{q}_s \tag{3.6}$$
$$\dot{q}(T_f) = a_1 + 2a_2 T_f + 3a_3 T_f^2 = \dot{q}_f \tag{3.7}$$

Equations (3.4)–(3.7) can be further written in a matrix form

$$\begin{bmatrix} 1 & T_s & T_s^2 & T_s^3 \\ 1 & T_f & T_f^2 & T_f^3 \\ 0 & 1 & 2T_s & 3T_s^2 \\ 0 & 1 & 2T_f & 3T_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_s \\ q_f \\ \dot{q}_s \\ \dot{q}_f \end{bmatrix} \tag{3.8}$$

Solving (3.8) gives the coefficients of the cubic function $q(t)$ satisfying all conditions. In general, the start time T_s is chosen to be zero. Therefore, (3.8) becomes

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & T_f & T_f^2 & T_f^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2T_f & 3T_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_s \\ q_f \\ \dot{q}_s \\ \dot{q}_f \end{bmatrix} \tag{3.9}$$

Solving (3.9) yields

$$a_0 = q_s \tag{3.10}$$
$$a_1 = \dot{q}_s \tag{3.11}$$

$$a_2 = -\frac{2\dot{q}_s T_f + \dot{q}_f T_f + 3q_s - 3q_f}{T_f^2} \tag{3.12}$$

$$a_3 = \frac{\dot{q}_s T_f + \dot{q}_f T_f + 2q_s - 2q_f}{T_f^3} \tag{3.13}$$

With the coefficients given in (3.10)–(3.13), the polynomial $q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$ satisfies all four conditions given by (3.1).

One special case of the above problem is that the desired start and final velocities are zeros. In this case, the solutions can be obtained by substituting $\dot{q}_s = 0$ and $\dot{q}_f = 0$ into (3.10)–(3.13), such that

$$a_0 = q_s \tag{3.14}$$

$$a_1 = 0 \tag{3.15}$$

$$a_2 = \frac{3(q_f - q_s)}{T_f^2} \tag{3.16}$$

$$a_3 = \frac{2(q_s - q_f)}{T_f^3} \tag{3.17}$$

Although the start and final velocities are zeros, the start and final accelerations still present. Further, the instant transitions between zero acceleration and finite accelerations at the start and end cause large *jerk*, which is the time derivative of the acceleration. In order to restrict the jerk, sometimes, we require the start and final accelerations to be zeros as well. In this case, the conditions become

$$q(0) = q_s, \quad q(T_f) = q_f, \quad \dot{q}(0) = 0, \quad \dot{q}(T_f) = 0, \quad \ddot{q}(0) = 0, \quad \ddot{q}(T_f) = 0 \tag{3.18}$$

The six constraints in (3.18) require six variables (coefficients of a polynomial) to get the exact solutions. Therefore, a fifth-order polynomial of the form

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{3.19}$$

is applied here. The velocity and acceleration profiles of the joint angle $q(t)$ can be found as

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \tag{3.20}$$

$$\ddot{q}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \tag{3.21}$$

Substituting (3.19)–(3.21) into (3.18) gives six linear equations

$$q(0) = a_0 = q_s \tag{3.22}$$

$$q(T_f) = a_0 + a_1 T_f + a_2 T_f^2 + a_3 T_f^3 + a_4 T_f^4 + a_5 T_f^5 = q_f \tag{3.23}$$

$$\dot{q}(0) = a_1 = 0 \tag{3.24}$$

$$\dot{q}(T_f) = a_1 + 2a_2 T_f + 3a_3 T_f^2 + 4a_4 T_f^3 + 5a_5 T_f^4 = 0 \tag{3.25}$$

$$\ddot{q}(0) = 2a_2 = 0 \tag{3.26}$$

$$\ddot{q}(T_f) = 2a_2 + 6a_3 T_f + 12a_4 T_f^2 + 20a_5 T_f^3 = 0 \tag{3.27}$$

Solving (3.22)–(3.27) gives

$$a_0 = q_s \tag{3.28}$$

$$a_1 = 0 \tag{3.29}$$

$$a_2 = 0 \tag{3.30}$$

$$a_3 = \frac{10(q_f - q_s)}{T_f^3} \tag{3.31}$$

$$a_4 = \frac{15(q_s - q_f)}{T_f^4} \tag{3.32}$$

$$a_5 = \frac{6(q_f - q_s)}{T_f^5} \tag{3.33}$$

In theory, a polynomial can satisfy an arbitrary number of constraints as long as the order of the polynomial is high enough. However, a high-order polynomial can yield unexpected and unwanted deviations from the desired path. Therefore, cubic polynomials are most commonly used in robotic path generation.

In many occasions, the desired path contains not only the start and final positions, but also some intermediate positions, which are called the *via points*. In order to deal with additional conditions introduced by the via points, more than one cubic polynomial can be used.

Example 3.2 (Cubic splines with via points): We want to command the R robot in Example 3.1 to move the end-effector from a start joint angle q_s at time T_s to a via position q_v at time T_v, and then to the final position q_f at time T_f. Use two polynomials to generate a smooth and continuous trajectory in the joint space, where the desired start and final angular velocities are zero.

Solution: Assume the two cubic polynomials are $q_1(t)$ and $q_2(t)$. The time periods of $q_1(t)$ and $q_2(t)$ are $[T_s, T_v]$ and $[T_v, T_f]$, respectively. For convenience, we want to first derive the trajectories on the periods of $[0, T_v - T_s]$ and $[0, T_f - T_v]$, respectively, such that the equations can be simplified. This can be done by shifting a function along the time axis, i.e., $f(t-T)$ is the function by moving $f(t)$ along the time axis with a shift distance of T. Let us first define two polynomials $q_1(t_1)$ and $q_2(t_2)$ on their local time periods, $[0, T_{f1}]$ and $[0, T_{f2}]$, respectively, with $T_{f1} = T_v - T_s$ and $T_{f2} = T_f - T_v$. After the polynomials are found based on the conditions, the trajectories are then shifted back to the correct start time.

In this problem, there are eight conditions, given by

$$\begin{aligned} q_1(0) = q_s, \quad \dot{q}_1(0) = 0, \quad q_2(T_{f2}) = q_f, \quad \dot{q}_2(T_{f2}) = 0 \\ q_1(T_{f1}) = q_v, \quad q_2(0) = q_v, \quad \dot{q}_1(T_{f1}) = \dot{q}_2(0), \quad \ddot{q}_1(T_{f1}) = \ddot{q}_2(0) \end{aligned} \tag{3.34}$$

where the equations in the first row represent the start and final conditions, while those in the second represent the conditions at the via position. The two desired cubic functions are assumed to be

$$\begin{aligned} q_1(t_1) &= a_0 + a_1 t_1 + a_2 t_1^2 + a_3 t_1^3 \\ q_2(t_2) &= b_0 + b_1 t_2 + b_2 t_2^2 + b_3 t_2^3 \end{aligned} \tag{3.35}$$

with the time derivative giving the velocity profiles

$$\begin{aligned} \dot{q}_1(t_1) &= a_1 + 2a_2 t_1 + 3a_3 t_1^2 \\ \dot{q}_2(t_2) &= b_1 + 2b_2 t_2 + 3b_3 t_2^2 \end{aligned} \tag{3.36}$$

A further time derivative yields the acceleration profiles

$$\begin{aligned} \ddot{q}_1(t_1) &= 2a_2 + 6a_3 t_1 \\ \ddot{q}_2(t_2) &= 2b_2 + 6b_3 t_2 \end{aligned} \tag{3.37}$$

Substituting (3.35)–(3.37) into the conditions (3.34) yields eight linear equations

$$\begin{aligned}
q_1(0) &= a_0 = q_s \\
\dot{q}_1(0) &= a_1 = \dot{q}_s \\
q_2(T_{f2}) &= b_0 + b_1 T_{f2} + b_2 T_{f2}^2 + b_3 T_{f2}^3 = q_f \\
\dot{q}_2(T_{f2}) &= b_1 + 2b_2 T_{f2} + 3b_3 T_{f2}^2 = \dot{q}_f \\
q_1(T_{f1}) &= a_0 + a_1 T_{f1} + a_2 T_{f1}^2 + a_3 T_{f1}^3 = q_v \\
q_2(0) &= b_0 = q_v \\
\dot{q}_1(T_{f1}) &= a_1 + 2a_2 T_{f1} + 3a_3 T_{f1}^2 = \dot{q}_2(0) = b_1 \\
\ddot{q}_1(T_{f1}) &= 2a_2 + 6a_3 T_{f1} = \ddot{q}_2(0) = 2b_2
\end{aligned} \tag{3.38}$$

Solving equations (3.38) gives

$$\begin{aligned}
a_0 &= q_s \\
a_1 &= 0 \\
a_2 &= \frac{-3T_{f1}^2 \Delta q_2 + 6T_{f1}T_{f2}\Delta q_1 + 3T_{f2}^2 \Delta q_1}{2T_{f1}^2 T_{f2}(T_{f1} + T_{f2})} \\
a_3 &= \frac{3T_{f1}^2 \Delta q_2 - 4T_{f1}T_{f2}\Delta q_1 - T_{f2}^2 \Delta q_1}{2T_{f1}^3 T_{f2}(T_{f1} + T_{f2})} \\
b_0 &= q_v \\
b_1 &= \frac{3(T_{f1}^2 \Delta q_2 + T_{f2}^2 \Delta q_1)}{2T_{f1}T_{f2}(T_{f1} + T_{f2})} \\
b_2 &= \frac{3(T_{f1}\Delta q_2 - T_{f2}\Delta q_1)}{T_{f1}T_{f2}(T_{f1} + T_{f2})} \\
b_3 &= \frac{-T_{f1}^2 \Delta q_2 - 4T_{f1}T_{f2}\Delta q_2 + 3T_{f2}^2 \Delta q_1}{2T_{f1}T_{f2}^3(T_{f1} + T_{f2})}
\end{aligned} \tag{3.39}$$

where $\Delta q_1 = q_v - q_s$ and $\Delta q_2 = q_f - q_v$. With the coefficients given in equations (3.39) and the time shifts of two cubic functions, the final trajectory can be written as

$$q(t) = \begin{cases} a_0 + a_1(t - T_s) + a_2(t - T_s)^2 + a_3(t - T_s)^3, & T_s \le t \le T_v \\ b_0 + b_1(t - T_v) + b_2(t - T_v)^2 + b_3(t - T_v)^3, & T_v \le t \le T_f \end{cases} \tag{3.40}$$

In many applications, the time intervals throughout the whole period are chosen to be the same, such that $T_{f1} = T_{f2} = T$. Hence, the linear constraints given by (3.38) can be readily written in the matrix form

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & T & T^2 & T^3 & 0 & 0 & 0 & 0 \\
0 & 1 & 2T & 3T^2 & 0 & -1 & 0 & 0 \\
0 & 0 & 2 & 6T & 0 & 0 & -2 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & T & T^2 & T^3 \\
0 & 0 & 0 & 0 & 0 & 1 & 2T & 3T^2
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}
=
\begin{bmatrix} q_s \\ 0 \\ q_v \\ 0 \\ 0 \\ q_v \\ q_f \\ 0 \end{bmatrix} \tag{3.41}$$

where the square matrix is a tri-diagonal matrix with a bandwidth of 2, which is counted as the maximum offset of nonzero elements from the diagonal. The inverse of such a matrix can be computed efficiently by computational software. Solving (3.41) gives

$$a_0 = q_s \tag{3.42}$$

$$a_1 = 0 \tag{3.43}$$

$$a_2 = \frac{-3\Delta q_2 + 9\Delta q_1}{4T^2} \tag{3.44}$$

$$a_3 = \frac{3\Delta q_2 - 5\Delta q_1}{4T^3} \tag{3.45}$$

$$b_0 = q_v \tag{3.46}$$

$$b_1 = 3\frac{\Delta q_2 + \Delta q_1}{4T} \tag{3.47}$$

$$b_2 = 3\frac{\Delta q_2 - \Delta q_1}{2T^2} \tag{3.48}$$

$$b_3 = \frac{-5\Delta q_2 + 3\Delta q_1}{4T^3} \tag{3.49}$$

In practice, a trajectory of one joint contains a large number of via points. Hence, an adequate number of cubic functions are required to construct the trajectory. The coefficients of these cubic functions can be found by expanding the matrix equation (3.41) into

$$\begin{bmatrix} \mathbf{A} & \cdots & \cdot \\ \vdots & \ddots & \vdots \\ \cdot & \cdots & \mathbf{B} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ b_0 \\ b_1 \\ \vdots \\ v_2 \\ v_3 \\ w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} q_s \\ 0 \\ q_{v1} \\ 0 \\ 0 \\ q_{v1} \\ \vdots \\ q_{vn} \\ 0 \\ 0 \\ q_{vn} \\ q_f \\ 0 \end{bmatrix} \tag{3.50}$$

where $\mathbf{A}$ and $\mathbf{B}$ form a $(4n) \times (4n)$ square matrix, such that

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdot & \cdot \\ 0 & 1 & 0 & 0 & 0 & \cdot \\ 1 & T & T^2 & T^3 & 0 & 0 \\ 0 & 1 & 2T & 3T^2 & 0 & -1 \\ \cdot & 0 & 2 & 6T & 0 & 0 \\ \cdot & \cdot & 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.51}$$

and

$$\mathbf{B} = \begin{bmatrix} T^2 & T^3 & 0 & 0 & \cdot & \cdot \\ 2T & 3T^2 & 0 & -1 & 0 & \cdot \\ 2 & 6T & 0 & 0 & -2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \cdot & 0 & 1 & T & T^2 & T^3 \\ \cdot & \cdot & 0 & 1 & 2T & 3T^2 \end{bmatrix} \tag{3.52}$$

while $a_0, a_1, \ldots, w_3$ are the coefficients of n piecewise cubic functions, which can be obtained by solving the above equation. The trajectory described by the piecewise cubic functions is also called a cubic spline.

3.2 Interpolation with Linear and Parabolic Functions

The simplest interpolation on a set of start, via, and final points over a period of time is to use linear functions to link these points together. However, at each point, there is a discontinuity of velocity, which causes infinitely large acceleration. This can be both dangerous and infeasible for the capability of actuators. In order to handle this issue, a parabolic function is introduced to each point with a constant acceleration to allow a smooth transition between two adjacent linear paths.

Consider the single R robot in Example 3.1, where the joint is commanded to move from the start position q_s at time 0 to the final position q_f at time t_f, through a number of via points q_i at t_i, for $i = 1, \ldots, n$, in order. The basic concept is to use straight lines to connect these points and then use parabolic functions to generate smooth transitions based on the given magnitudes of constant accelerations at all points. Consider three points, (t_{i-1}, q_{i-1}), (t_i, q_i), and (t_{i+1}, q_{i+1}), as shown in Figure 3.2. Using linear interpolation, two linear paths through these three points are obtained as

$$q_{i-1}(t) = q_{i-1} + \dot{q}_{i-1}(t - t_{i-1}), \qquad t_{i-1} \le t \le t_i \tag{3.53}$$

$$q_i(t) = q_i + \dot{q}_i(t - t_i), \qquad t_i \le t \le t_{i+1} \tag{3.54}$$

where $\dot{q}_{i-1} = \dfrac{q_i - q_{i-1}}{t_i - t_{i-1}}$ and $\dot{q}_i = \dfrac{q_{i+1} - q_i}{t_{i+1} - t_i}$. Given the magnitude of the acceleration through Point i being $|\ddot{q}_i|$, this acceleration can be determined upon the adjacent velocities as

$$\ddot{q}_i = \operatorname{sign}(\dot{q}_i - \dot{q}_{i-1})\,|\ddot{q}_i| \tag{3.55}$$

The time period of transition is then given by

$$T_i = \frac{\dot{q}_i - \dot{q}_{i-1}}{\ddot{q}_i} \tag{3.56}$$

Consider one-half of the transition happens at the end of the first linear path while the other half happens at the beginning of the second linear path. Therefore, the linear paths given (3.53) and (3.54) becomes

$$q_{i-1}(t) = q_{i-1} + \dot{q}_{i-1}(t - t_{i-1}), \qquad t_{i-1} \le t \le t_i - \frac{T_i}{2} \tag{3.57}$$

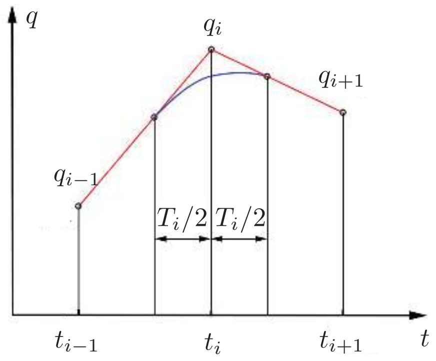


FIGURE 3.2
Three via points.

$$q_i(t) = q_i + \dot{q}_i(t - t_i), \qquad t_i + \frac{T_i}{2} \leq t \leq t_{i+1} \tag{3.58}$$

At the end of the first linear path given by (3.57) and (3.58), the joint position is given by

$$\phi_{i-1} = q_{i-1} + \dot{q}_{i-1}\left(t_i - \frac{T_i}{2} - t_{i-1}\right) \tag{3.59}$$

Therefore, the parabolic transition is

$$q_{t(i-1)}(t) = \phi_{i-1} + \dot{q}_{i-1}\left(t - t_i + \frac{T_i}{2}\right) + \frac{\ddot{q}_i}{2}\left(t - t_i + \frac{T_i}{2}\right)^2 \tag{3.60}$$

Note that (3.60) is formulated based on the start position ϕ_{i-1}. It can be shown readily that the end position of this parabolic trajectory at $t_i + \frac{T_i}{2}$ is exactly the same as the start position of the following linear trajectory given by (3.57)–(3.58) at the same instant. This can be easily understood in the velocity profile of the joint angle, where the parabolic transition is a straight line connecting two-step functions. Since the removed and added triangles in the integration area are the same, the joint travelling distance remains unchanged. Further, the transition time T_{i-1} and T_{i+1} are also required at t_{i-1} and t_{i+1}, respectively. In summary, the two linear paths and one parabolic transition can be written as

$$q_{i-1}(t) = q_{i-1} + \dot{q}_{i-1}(t - t_{i-1}), \qquad t_{i-1} + \frac{T_{i-1}}{2} \leq t \leq t_i - \frac{T_i}{2} \tag{3.61}$$

$$q_{t(i-1)}(t) = \phi_{i-1} + \dot{q}_{i-1}\left(t - t_i + \frac{T_i}{2}\right) + \frac{\ddot{q}_i}{2}\left(t - t_i + \frac{T_i}{2}\right)^2, \qquad t_i - \frac{T_i}{2} \leq t \leq t_i + \frac{T_i}{2} \tag{3.62}$$

$$q_i(t) = q_i + \dot{q}_i(t - t_i), \qquad t_i + \frac{T_i}{2} \leq t \leq t_{i+1} - \frac{T_{i+1}}{2} \tag{3.63}$$

Note that the above trajectory does not pass through via points exactly. In the case of high acceleration, the deviation of the generated trajectory is close enough to the via points. As for the start and final points, the robot must start and finish at them exactly. Hence, they need to be treated slightly differently.

In order to force the trajectory to pass through the start point $(0, q_s)$, an offset point $(t_s + \frac{T_s}{2}, q_s)$ is introduced as shown in Figure 3.3. With this offset point, two linear lines can connect the start point, offset point, and the first via point (t_1, q_1). We also desire to use a transition time T_s to change the velocity smoothly according to a given $|\ddot{q}_s|$. The acceleration is

$$\ddot{q}_s = \text{sign}(q_1 - q_s)|\ddot{q}_s| \tag{3.64}$$

Consider the R robot starts from stationary with the constant acceleration $\ddot{q}_s$ for a time period of T_s to reach the velocity defined by the second linear path. The relation can be written as

$$\ddot{q}_s T_s = \dot{q}_s \tag{3.65}$$

where $\dot{q}_s = \frac{q_1 - q_s}{t_1 - \frac{T_s}{2}}$. Equation (3.65) gives two solutions of T_s as

$$T_s = t_1 \pm \sqrt{t_1^2 - \frac{2}{\ddot{q}_s}(q_1 - q_s)} \tag{3.66}$$

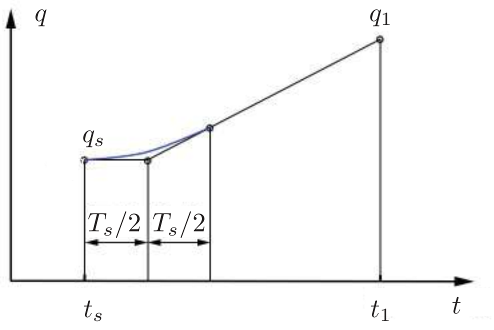


FIGURE 3.3
The start two points.

Since T_s must be less than t_1, the only solution of T_s is $t_1 - \sqrt{t_1^2 - \frac{2}{\ddot{q}_s}(q_1 - q_s)}$. Hence, the trajectory from the start point is

$$q_{ts}(t) = q_s + \ddot{q}_s t^2/2, \qquad 0 \le t \le T_s \tag{3.67}$$
$$q_s(t) = q_1 - \dot{q}_s(t_1 - t), \qquad T_s \le t \le t_1 - T_1/2 \tag{3.68}$$

Similarly, an offset point $(t_f - \dfrac{T_f}{2}, q_f)$ is introduced to generate the path at the final point, as shown in Figure 3.4. Given $|\ddot{q}_f|$, the acceleration is

$$\ddot{q}_f = \text{sign}(q_f - q_n)|\ddot{q}_f| \tag{3.69}$$

The continuous velocity at $t_f - T_f$ yields

$$\ddot{q}_f T_f = -\dot{q}_n \tag{3.70}$$

where $\dot{q}_n = \frac{q_f - q_n}{t_f - \frac{T_f}{2} - t_n}$. Solving (3.70) gives

$$T_f = (t_f - t_n) - \sqrt{(t_f - t_n)^2 - \frac{2}{\ddot{q}_f}(q_n - q_f)} \tag{3.71}$$

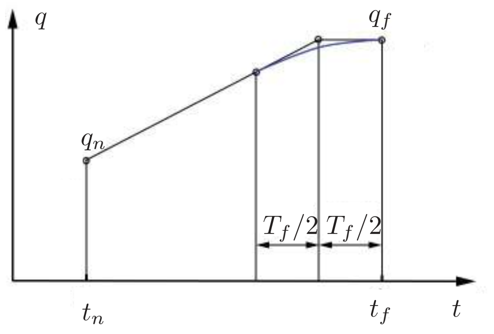


FIGURE 3.4
The final two points.

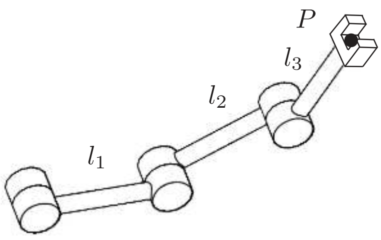


FIGURE 3.5
A planar 3R robot.

The trajectory to the final point is

$$q_n(t) = q_n + \dot{q}_n(t - T_n), \qquad t_n + \frac{T_n}{2} \le t \le t_f - T_f \tag{3.72}$$

$$q_{tf}(t) = q_f - \frac{\ddot{q}_f}{2}(t_f - t)^2, \qquad t_f - T_f \le t \le t_f \tag{3.73}$$

In summary, the complete trajectory is described by (3.67)–(3.68), (3.61)–(3.63), and (3.72)–(3.73) at the start, via, and final points, respectively. All intermediate parameters are defined in the above corresponding equations. The advantage of the interpolation based on linear and quadratic functions is that it does not need to solve a large equation system and can be obtained efficiently. Further, the capability of the actuators can be considered to predefine the bending accelerations.

3.3 Trajectory in Joint Space

The previous discussion was about the trajectory generation on a trivial R robot. In general, a robotic manipulator has more than three DoF. Hence, the trajectories must be created for all joint angles simultaneously.

Example 3.3 (Joint space trajectory): Given a three-DoF planar robotic manipulator shown in Figure 3.5, develop a trajectory of the robot such that it starts at a configuration (q_{1s}, q_{2s}, q_{3s}) at time 0, and stops at a configuration (q_{1f}, q_{2f}, q_{3f}) at time t_f. Both initial and final velocities are zeros.

Solution: The trajectories can be conveniently developed in the joint space (q_1, q_2, q_3). The joint space of a serial robotic manipulator is also called the *configuration space*, which is defined as a set of all configurations for a general robot, which can be a biped robot, robotic vehicle, or robotic aircraft. Any position and orientation of a robot can be represented as a single point in its configuration space, which simplifies the complex problem of general path planning.

In this example, the joint space is a three-dimensional space with three axes (q_1, q_2, q_3). The trajectories can be independently derived with respect to all axes respectively. Therefore, (3.14)–(3.17) can be used to obtain three cubic functions

$$\begin{aligned} a_{i0} &= q_{is} \\ a_{i1} &= 0 \\ a_{i2} &= \frac{3(q_{if} - q_{is})}{t_f^2} \\ a_{i3} &= \frac{2(q_{is} - q_{if})}{t_f^3} \end{aligned} \tag{3.74}$$

where $i = 1, 2, 3$. The polynomials $q_i(t) = a_{i0} + a_{i1}t + a_{i2}t^2 + a_{i3}t^3$ generate a trajectory of the robot satisfying all boundary conditions.

The approach in Example 3.3 can be applied to a serial robotic manipulator with an arbitrary DoF. If there are a number of via points, either the polynomial solution given by (3.50) or the linear and quadratic interpolation given by (3.67)–(3.68), (3.61)–(3.63), and (3.72)–(3.73) can be used. However, joint space trajectory generation is unsuitable in instances where the path of the end-effector must be constrained to a specific trajectory, such as in collision avoidance or for mission-specific tasks. A solution to this is to define via points in the task space, which are then converted into the joint space before connecting them into a single trajectory with splines or parabolic blends. However, there is no guarantee that the end-effector strictly follows the desired task space path, because the trajectory is still defined in the joint space. To achieve this, the entire trajectory should be defined in the task space.

3.4 Trajectory in Task Space

Generating trajectories in the task space follows the same principles as joint space trajectory generation, except that motion is generated through control of the end-effector's pose. Although trajectories generated in the task space are intuitive, they may not always be feasible due to kinematic constraints. Task space trajectories are usually converted into joint space for execution. Hence, an inverse kinematic solution along each point of the task space must exist for a path to be feasible. Furthermore, there is a risk of the robot approaching singularities at the workspace limit, lending to potentially dangerous joint velocities. Therefore, a task space trajectory should always be checked for safety before execution.

Example 3.4 (Task space trajectory): Given a two-DoF planar 2R robotic manipulator shown in Figure 3.6, develop a trajectory of the robot such that its end-effector starts at a position (p_{xs}, p_{ys}) at time 0, and stops at a position (p_{xf}, p_{yf}) at time t_f. Both initial and final velocities are zeros.

Solution: This trajectory can be developed in the task space (p_x, p_y). The trajectories in x and y axes can be independently derived. Again, (3.14)–(3.17) can be used to obtain two cubic

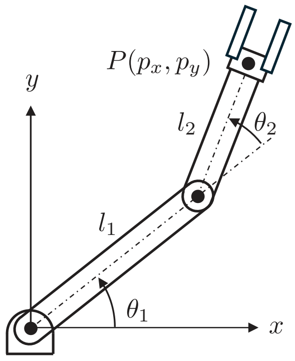


FIGURE 3.6
A planar 2R robot.

functions

$$
\begin{aligned}
a_{i0} &= p_{is} \\
a_{i1} &= 0 \\
a_{i2} &= \frac{3(p_{if} - p_{is})}{t_f^2} \\
a_{i3} &= \frac{2(p_{is} - p_{if})}{t_f^3}
\end{aligned}
\tag{3.75}
$$

where $i = x, y$. The polynomials $p_i(t) = a_{i0} + a_{i1}t + a_{i2}t^2 + a_{i3}t^3$ generate a trajectory of the robot satisfying all boundary conditions.

Example 3.5 (Orientation space trajectory): Given a three-DoF spherical robotic wrist shown in Figure 3.7, develop a trajectory of the robot such that its end-effector starts at an orientation $\mathbf{R}_s$ at time 0 and stops at another orientation $\mathbf{R}_f$ at time t_f. Both initial and final angular velocities are zeros.

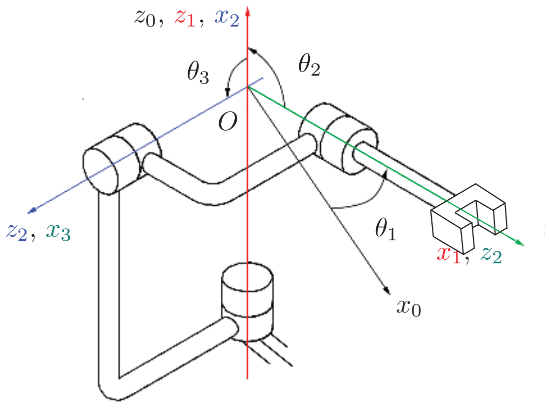


FIGURE 3.7
A spherical robotic wrist.

Solution: There are three different independent-parameter representations of orientation. If we use $X-Y-Z$ Euler angles to represent the orientation of the end-effector of this robotic wrist, the task space is then defined by (ϕ_x, ϕ_y, ϕ_z). Hence, the start and final orientations can be derived from $\mathbf{R}_s$ and $\mathbf{R}_f$ as $(\phi_{xs}, \phi_{ys}, \phi_{zs})$ and $(\phi_{xf}, \phi_{yf}, \phi_{zf})$, respectively. The trajectories in these three Euler-angle axes can be independently derived. Again, Equations (3.14)–(3.17) can be used to obtain three cubic functions

$$\begin{aligned} a_{i0} &= \phi_{is} \\ a_{i1} &= 0 \\ a_{i2} &= \frac{3(\phi_{if} - \phi_{is})}{t_f^2} \\ a_{i3} &= \frac{2(\phi_{is} - \phi_{if})}{t_f^3} \end{aligned} \tag{3.76}$$

where $i = x, y, z$. The polynomial $\phi_i(t) = a_{i0} + a_{i1}t + a_{i2}t^2 + a_{i3}t^3$ generates a trajectory of the robot satisfying all boundary conditions. Notably, if another representation of orientation is used, the derived trajectory will differ even though the boundary conditions are satisfied.

A trajectory of a robotic manipulator with six-DoF can be developed in a similar way, as long as six independent parameters are well defined in the task space: three for the position and the other three for the orientation.

3.5 MATLAB® Examples

Example M3.1 (Cubic splines): For a single actuator, calculate the cubic splines required to create a single path that starts, visits and ends at the following positions:

Point	Position (°)	Time (s)
q_{init}	0	0
q_{via}	90	2
q_{final}	45	5

Solution: We require two cubic splines to connect these three points. Each cubic has a valid time defined in Table 3.1. Working with normalised variables is preferred because it simplifies the system using zero-time constraints. It creates many zero entries in the system matrix, cancelling out variables and simplifying overall equations. The resulting cubic constraint equations are:

TABLE 3.1
Time of each cubic spline

Spline	Time		
	Global	Normalised	
		Variable	Value
Cubic 1	$0 \le t \le 2$	$t_{s1} \le t_1 \le t_{f1}$	$0 \le t_1 \le 2$
Cubic 2	$2 \le t \le 5$	$t_{s2} \le t_2 \le t_{f2}$	$0 \le t_2 \le 3$

1. Cubic 1 position at $t_{s1} = q_{init}$, where $t_{s1} = 0$

$$a_1 + a_2 t_{s1} + a_3 t_{s1}^2 + a_4 t_{s1}^3 = q_{init} \tag{3.77}$$
$$a_1 = 0 \tag{3.78}$$

2. Cubic 1 velocity at $t_{s1} = 0$

$$a_2 + 2a_3 t_{s1} + 3a_4 t_{s1}^2 = \dot{q}_{init} \tag{3.79}$$
$$a_2 = 0 \tag{3.80}$$

3. Cubic 1 position at $t_{f1} = q_{via}$, where $t_{f1} = 2$

$$a_1 + a_2 t_{f1} + a_3 t_{f1}^2 + a_4 t_{f1}^3 = q_{via} \tag{3.81}$$
$$a_1 + 2a_2 + 4a_3 + 8a_4 = 90 \tag{3.82}$$

4. Cubic 2 velocity at $t_{s2} =$ Cubic 1 velocity at t_{f1}, where $t_{f1} = 2$ and $t_{s2} = 0$

$$b_2 + 2b_3 t_{s2} + 3b_4 t_{s2}^2 = a_2 + a_3 t_{f1} + 3a_4 t_{f1}^2 \tag{3.83}$$
$$a_2 + 4a_3 + 12a_4 - b_2 = 0 \tag{3.84}$$

5. Cubic 2 acceleration at $t_{s2} =$ Cubic 1 acceleration at t_{f1}, where $t_{f1} = 2$ and $t_{s2} = 0$

$$2b_3 + 6b_4 t_{s2} = 2a_3 + 6a_4 t_{f1} \tag{3.85}$$
$$2a_3 + 12a_4 - 2b_3 = 0 \tag{3.86}$$

6. Cubic 2 position at $t_{s2} = q_{via}$, where $t_{s2} = 0$

$$b_1 + b_2 t_{s2} + b_3 t_{s2}^2 + b_4 t_{s2}^3 = q_{via} \tag{3.87}$$
$$b_1 = 90 \tag{3.88}$$

7. Cubic 2 position at $t_{f2} = q_{final}$, where $t_{f2} = 3$

$$b_1 + b_2 t_{f2} + b_3 t_{f2}^2 + b_4 t_{f2}^3 = q_{final} \tag{3.89}$$
$$b_1 + 3b_2 + 9b_3 + 27b_4 = 45 \tag{3.90}$$

8. Cubic 2 velocity at $t_{f2} = 0$, where $t_{f2} = 2$

$$b_2 + 2b_3 t_{f2} + 3b_4 t_{f2}^2 = \dot{q}_{final} \tag{3.91}$$
$$b_2 + 6b_3 + 27b_4 = 0 \tag{3.92}$$

These equations can be represented as a linear system in matrix form $\mathbf{A}\ \mathbf{x} = \mathbf{b}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 12 & 0 & -1 & 0 & 0 \\ 0 & 0 & 2 & 12 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 9 & 27 \\ 0 & 0 & 0 & 0 & 0 & 1 & 6 & 27 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 90 \\ 0 \\ 0 \\ 90 \\ 45 \\ 0 \end{bmatrix} \tag{3.93}$$

The following inline script generates the system defined in (3.93).

```
% Define time vectors
time = [0 2 5]
dtime = diff(time)                % Normalise time

% Initialise variables
n_cubics = 2;
A = zeros(4*n_cubics);
b = zeros(4*n_cubics, 1);

% Define proposed boundary conditions
pos_init = 0;
pos_via = 90;
pos_final = 45;
vel_init = 0;
vel_final = 0;

% Function handles to define coefficients
coeff_pos = @(t) [1     t   t^2     t^3];   % Position coefficients
coeff_vel = @(t) [0     1   2*t     3*t^2]; % Velocity
coeff_acc = @(t) [0     0   2       6*t];   % Acceleration

%% Cubic 1 equations
t_s1 = 0;                         % Cubic 1 initial time
A(1,1:4) = coeff_pos(t_s1);
b(1) = pos_init;

A(2,1:4) = coeff_vel(t_s1);
b(2) = vel_init;

t_f1 = dtime(1);                  % Cubic 1 final time
A(3,1:4) = coeff_pos(t_f1);
b(3) = pos_via;

%% Cubic 1+2 equations
t_s2 = 0;                         % Cubic 2 initial time
A(4,1:8) = [coeff_vel(t_f1) -coeff_vel(t_s2)];
b(4) = 0;

A(5,1:8) = [coeff_acc(t_f1) -coeff_acc(t_s2)];
b(5) = 0;

%% Cubic 2 equations
A(6,5:8) = coeff_pos(t_s2);
b(6) = pos_via;

t_f2 = dtime(2);                  % Cubic 2 final time
A(7,5:8) = coeff_pos(t_f2);
b(7) = pos_final;

A(8,5:8) = coeff_vel(t_f2);
b(8) = vel_final;

%% Solve system
cfs = A \ b;                      % This is the fastest solve method

% Display variables
A
b
cfs

%% Plot system
t1 = t_s1:0.1:t_f1;
p1 = polyval(cfs(4:-1:1), t1);
t2 = t_s2:0.1:t_f2;
p2 = polyval(cfs(8:-1:5), t2);
plot(t1, p1, t1(end)+t2, p2, 'LineWidth', 3)

% Always label plots
title('Cubic Spline Example')
ylabel('Position (deg)')
xlabel('Time (s)')
grid on
```

```
time =

     0     2     5


dtime =

     2     3

```

```
A =

     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     1     2     4     8     0     0     0     0
     0     1     4    12     0    -1     0     0
     0     0     2    12     0     0    -2     0
     0     0     0     0     1     0     0     0
     0     0     0     0     1     3     9    27
     0     0     0     0     0     1     6    27


b =

     0
     0
    90
     0
     0
    90
    45
     0


cfs =

         0
         0
   51.7500
  -14.6250
   90.0000
   31.5000
  -36.0000
    6.8333
```

Therefore, the two cubic splines that will pass through the proposed positions are

$$q(t) = \begin{cases} -14.63{t_1}^3 + 51.75{t_1}^2 & 0 \le t_1 \le 2 \text{ when } 0 \le t \le 2 \\ 6.83{t_2}^3 - 36{t_2}^2 + 31.5t_2 + 90 & 0 \le t_2 \le 3 \text{ when } 2 \le t \le 5 \end{cases} \tag{3.94}$$

Example M3.2 (Cubic splines for SCARA): A common motion for 3C (Computer, Communications, and Consumer) product assembly is to translate a part while rotating it by 180 degrees. The SCARA robot (Figure 3.8) is well-suited to perform this trajectory and task. Assume the robot's end-effector is currently at

$$^0\mathbf{p}_{EE} = \begin{bmatrix}0.70 & 0.00 & -0.25\end{bmatrix} \text{ m}, \qquad q_4 = 0 \tag{3.95}$$

and the next pose in the assembly task for the end-effector in 3 seconds is

$$^0\mathbf{p}_{EE} = \begin{bmatrix}-0.20 & 0.30 & -0.25\end{bmatrix} \text{ m}, \qquad q_4 = \pi \tag{3.96}$$

Generate a cubic spline trajectory in the end-effector coordinates that connects these two poses in a single, smooth transition.

Solution: A trajectory is desired in the end-effector coordinates, or *task space coordinates*. This implies that, later, inverse kinematics will be used to translate this task space trajectory into a joint space trajectory for robot control. In this problem, the task space coordinate consists of a Cartesian point and a rotation. Therefore, we will define $\mathbf{X}$ as the vector of task space co-ordinates, such that

$$\mathbf{X} = \begin{bmatrix}x & y & z & q_4\end{bmatrix}^T \tag{3.97}$$

Also

$$\mathbf{X}_{init} = \begin{bmatrix}-0.7 & 0.0 & -0.25 & 0\end{bmatrix}^T \tag{3.98}$$

$$\mathbf{X}_{final} = \begin{bmatrix}-0.2 & 0.3 & -0.25 & \pi\end{bmatrix}^T \tag{3.99}$$

Hence, we need only three cubics since only three actuators move between two points, each from $t_s = 0$ to $t_f = 3$. Therefore, for each dimension, $p \in \mathbf{X}$ that contains movement, we define the following constraint equations:

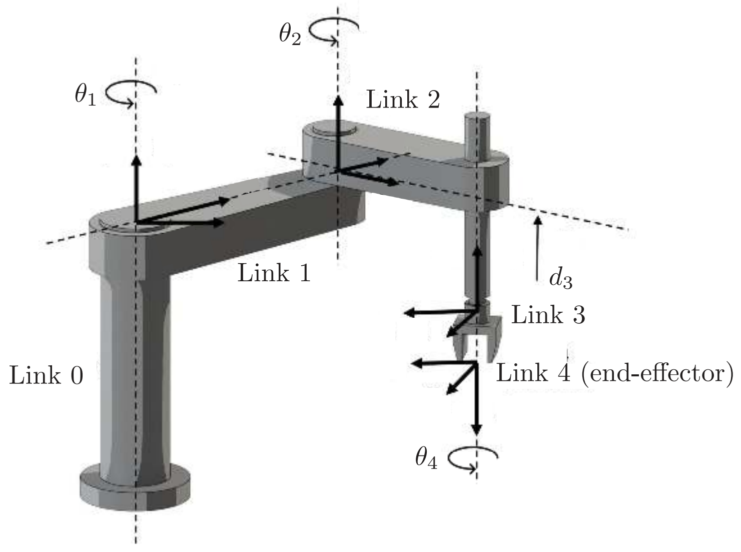


FIGURE 3.8
A SCARA robot.

1. Cubic for initial position at $t_s = 0$

$$a_1 + a_2 t_s + a_3 {t_s}^2 + a_4 {t_s}^3 = p_{init} \tag{3.100}$$
$$a_1 = p_{init} \tag{3.101}$$

2. Cubic for initial velocity at $t_s = 0$

$$a_2 + 2a_3 t_s + 3a_4 {t_s}^2 = \dot{p}_{init} \tag{3.102}$$
$$a_2 = 0 \tag{3.103}$$

3. Cubic for final position at $t_f = 3$

$$a_1 + a_2 t_f + a_3 {t_f}^2 + a_4 {t_f}^3 = p_{final} \tag{3.104}$$
$$a_1 + 2a_2 + 9a_3 + 27a_4 = p_{final} \tag{3.105}$$

4. Cubic for final velocity at $t_f = 3$

$$a_2 + 2a_3 t_f + 3a_4 {t_f}^2 = \dot{p}_{final} \tag{3.106}$$
$$a_2 + 6a_3 + 27a_4 = 0 \tag{3.107}$$

Notably, there is no mention of acceleration. This is because acceleration constraints are only applied when transitioning between via points. We do not constrain acceleration at the very start and very end of the trajectory. The following inline script generates the system of equations for each cubic. The use of a `for..loop` construct here will perform this task more efficiently.

```
% Define time vector
t = [0 3];

% Initialise variables
n_cubics = 2;
A = zeros(4*n_cubics);
b = zeros(4*n_cubics, 1);

% Define position boundary conditions
X_init = [-0.7, 0.0, -0.25, 0];
```

```
X_final = [-0.2, 0.3, -0.25, pi];

% Function handles to define coefficients
coeff_pos = @(t) [1     t   t^2     t^3];   % Position coefficients
coeff_vel = @(t) [0     1   2*t     3*t^2]; % Velocity

% This task can be done more efficiently in a for-loop construct
for c = [1 2 4]          % Create cubics for 1st, 2nd and 4th vars in X_
    A = zeros(4);        % Initialise 4x4 matrix
    b = zeros(4,1);      % Initialise 4x1 vector
    % Cubic pos at t = 0
    A(1,:) = coeff_pos(t(1));
    b(1) = X_init(c);
    % Cubic vel at t = 0
    A(2,:) = coeff_vel(t(1));
    b(2) = 0;
    % Cubic pos at t = 3
    A(3,:) = coeff_pos(t(2));
    b(3) = X_final(c);
    % Cubic vel at t = 3
    A(4,:) = coeff_vel(t(2));
    b(4) = 0;

    fprintf('Cubic data for variable %i\n', c)
    % Display variables
    A
    b

    % Solve system
    coefficients = A \ b        % This is the fastest solve method

end
```

```
Cubic data for variable 1

A =

     1     0     0     0
     0     1     0     0
     1     3     9    27
     0     1     6    27


b =

   -0.7000
         0
   -0.2000
         0


coefficients =

   -0.7000
    0.0000
    0.1667
   -0.0370

Cubic data for variable 2

A =

     1     0     0     0
     0     1     0     0
     1     3     9    27
     0     1     6    27


b =

         0
         0
    0.3000
         0


coefficients =

         0
   -0.0000
    0.1000
   -0.0222
```

```
Cubic data for variable 4

A =

     1     0     0     0
     0     1     0     0
     1     3     9    27
     0     1     6    27


b =

         0
         0
    3.1416
         0


coefficients =

         0
    0.0000
    1.0472
   -0.2327
```

Therefore, the trajectory of $\mathbf{X}$, generated by cubic splines that satisfies point and time constraints, is

$$\mathbf{X(t)} = \begin{bmatrix} -0.037t^3 + 0.1667t^2 - 0.7 \\ -0.022t^3 + 0.1t^2 \\ -0.25 \\ -0.2327t^3 + 1.0472t^2 \end{bmatrix} \qquad 0 \leq t \leq 3 \tag{3.108}$$

3.6 Conclusion

We investigated two methods for generating trajectories between two or more points: cubic splines and linear interpolation with parabolic blends. While both are effective in generating smooth trajectories, cubic splines is the preferred method when there are via points in the trajectory. The system of equations to solve for the coefficients of the cubic splines was defined, and MATLAB examples were given to show the method's simplicity and practicality in trajectory generation for robotics.

There is a major caveat when performing trajectory planning using splines: We always assume the trajectory is collision-free and singularity-free. These assumptions may not always be feasible in practice. However, this will be addressed in the next chapter.

3.7 Exercises

Problem 1. Given an RRR robot moves its end-effector along a cubic spline path through three waypoints to reach its goal point, how many constraints are required to solve for all the coefficients of the individual cubics? Name the constraints required for one joint.

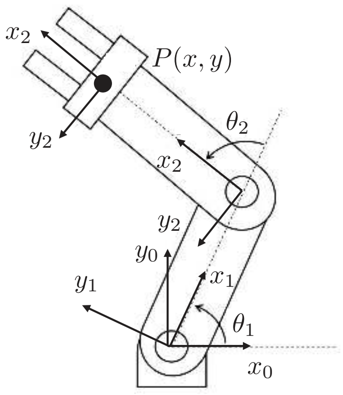


FIGURE 3.9
A two-link RR manipulator.

Problem 2. A single-joint robot performs the following task in sequence:

1. start at a joint angle of 10° at rest.
2. pick up an object at 30° after 1 s.
3. drop the object at 50° after 2 s.

The smooth motion is achieved with two continuous cubic functions. Solve the coefficients of the cubics by applying appropriate constraints.

Problem 3. Use the linear function with parabolic blends to generate a smooth trajectory from $x = 0$ m at $t = 0$ s to $x = 0.9$ m at $t = 1$ s. The required acceleration in the blend region is $\ddot{x} = 4.8$ m/s^2. Write the equations for the whole trajectory.

Problem 4. Frame {3} of the two-link manipulator shown in Figure 3.9 is projected to move to the following locations:

$$\overset{t_0 = 0}{{}^0\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & l_1 + l_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}, \quad \overset{t_1 = 2}{{}^0\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & l_2 - l_1\frac{\sqrt{2}}{2} \\ 0 & 1 & l_1\frac{\sqrt{2}}{2} \\ 0 & 0 & 1 \end{bmatrix}}$$

$$\overset{t_2 = 4}{{}^0\mathbf{T}_3 = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} & l_2\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & l_1 + l_2\frac{\sqrt{2}}{2} \\ 0 & 0 & 1 \end{bmatrix}}$$

l_1 and l_2 being the lengths of the first and the second links, respectively. Based on the transformation matrices at each point, if $l_1 = l_2 = 1$ m, find the cubic functions that describe each joint angle as the end-effector moves from rest at its starting point at time t_0, to the mid-point at t_1, before coming to rest at its end point at t_2. Ensure the rate of change of the angles of joints 1 and 2 at t_1 are continuous.

4 Control Schemes

This chapter will introduce basic control schemes used in servos and sensors. In addition, we will focus on *fixed reference control*, whereby we define a static input to the control system, known as a *set point* (SP). The SP dictates the desired state in which we want the system to reach in finite time, e.g., the temperature of an oven or the set speed of a car's cruise control. In robotics, this is typically a joint position q_d to which we want the manipulator to move. Fundamentally, there are two types of control loops: open-loop control (feedforward), and closed-loop control (feedback).

4.1 Open-Loop Control

An open-loop controller, also known as a non-feedback controller, computes its output based on a fixed input state. That is, the output of an open-loop control system depends only on the input it is given.

An open-loop controller is often used in simple processes or systems where feedback is not critical, thanks to its simplicity and low cost. Devices that utilise open-loop controls include on-off switching of valves, machinery, lights, heaters, and motors. An example of an open-loop controlled process is a belt conveyor system that transports materials in a warehouse. These systems are usually set to a fixed speed, but to maintain this speed, the conveyor operates under its rated maximum load. In an open-loop control scheme, the system will not compensate (i.e., attempt to increase power to the conveyor drive system to maintain speed) if there is a slow-down due to increased load on the conveyor system. In general, the output of an open-loop controlled process is simple and usually approximately sufficient under normal conditions without the need for feedback.

The advantage of using open-loop control is the reduction in component count and complexity, as a predictable output is always assumed. However, when operating outside of its rated conditions, also known as *disturbance*, its output may start to deviate from its intended set point. This is called *error*, and in an open loop control system, this error cannot be corrected automatically without manually adjusting its input. In other words, an open-loop control system cannot correct internal errors or counter external disturbances.

Figure 4.1 shows a block diagram of how different components in a system interact with each other in an open loop control system.

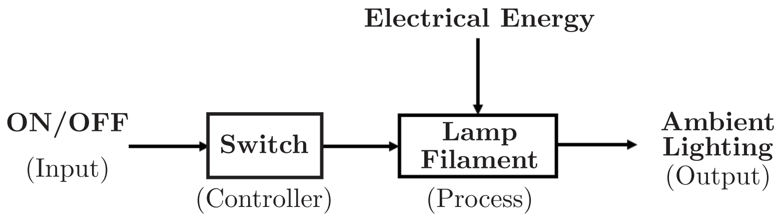


FIGURE 4.1
Open loop control system.

4.2 Closed-Loop Control

A closed-loop controller, also known as a feedback controller, is a control system where a portion of the output signal is fed back into the control system to determine the next output state.

The goal of the output of the system is to eliminate the difference between the system's desired output and actual output, or *error*. A typical instance of this is a thermostat for a heater, where the environment's temperature is fed into the system, compared to the thermostat's set temperature, then tells the heater whether to remain on or to turn off once the target temperature is reached. A cruise control system in a car is also a closed-loop control system, where the driver sets a target speed, and the vehicle matches said speed. If the car encounters resistance, such as driving up a hill and slows down, the control system detects an error between its current speed and set speed and applies an additional input to compensate for the error.

Closed-loop controllers have many advantages over their open-loop counterparts, the main advantage being their ability to reject disturbances from external sources. This makes them much more stable relative to their set points when conditions change (refer to the cruise control system example). They can also handle unstable inputs such as constant switching or varying set points. This is particularly useful when the set point of a servo is constantly varying to achieve path objectives.

There are many types of closed-loop controllers, all varying in performance and complexity. In this section, we will introduce four types of control: bang-bang, proportional (P), proportional-integral (PI), and proportional-integral-derivative (PID) controller. We will analyse how they work, analyse their typical out response for a set point input, and use common-practice tuning techniques to optimise their output response characteristics.

There are many types of closed-loop controllers, all varying in performance and complexity. In this section, we will introduce four types of control: bang-bang, proportional (P), proportional-integral (PI), and proportional-integral-derivative (PID) controller. We will analyse how they work, analyse their typical out response for a set point input, and use common-practice tuning techniques to optimise their output response characteristics. Figure 4.2 shows a block diagram of how different components in a system interact with each other in a closed-loop control system.

4.2.1 Bang-Bang Control

A bang-bang control scheme utilises discrete output states to achieve a target setpoint. A typical use of a bang-bang control algorithm is a thermostat in a heating or cooling system.

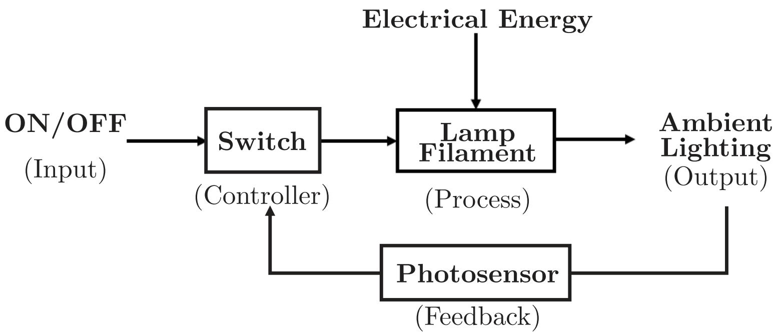


FIGURE 4.2
Closed loop control system.

When the temperature is below a target temperature, the heating system will try to heat the room to that temperature until the room is at the target temperature. If the room is above the target temperature, the heating system will correspondingly cool the room until the target temperature is met. To prevent quick oscillations between the heating and cooling state, *hysteresis* can be utilised, where a room would be cooled or heated beyond its setpoint by a fixed amount before switching to an idle state. The disadvantage of this method is that the system state may exhibit a ”zig-zag” or ”wiggle” behaviour, which might be unwanted under specific circumstances.

While adequate in some situations, bang-bang control is inadequate for more difficult tasks like speed control, or where system state oscillations are unsatisfactory. For instance, one does not want a cruise control system to constantly apply an on-off behaviour (on state, meaning applying maximum acceleration) to achieve the target speed, as it will make for a very uncomfortable and unsafe trip!

Example 4.1 (Bang-bang control): A household water heater is a common application of a bang-bang controller. Given that the heater has a set temperature of 30°C with a hysteresis error of 1°C, provide an illustration of the measured temperature graph and the corresponding state of the controller.

Solution: In a closed loop system such as the bang-bang control, the state of the controller depends on the feedback error, given as

$$e = q_d - q, \tag{4.1}$$

where e is the feedback error, q and q_d are the measured and desired temperatures, respectively.

Hysteresis is utilised where the heater would turn on or off when the measured temperature is beyond its setpoint by has a 1°C, such that

$$State = \begin{cases} ON, & \text{if } e < -1. \\ OFF, & \text{if } e > 1. \end{cases} \tag{4.2}$$

The state of the controller at a given feedback error could translate into an illustration of the measured temperature and its corresponding controller state, as shown in Figure 4.3.

Explanation: In the beginning, the water is sitting at room temperature (24° C), and the heater turns on to increase the temperature to its setpoint. The heater will turn off as the temperature 31° C, that is, 1° C beyond the setpoint. The heater will turn back on at 29° C, that is, 1° C below the setpoint. The cycle continues, which creates a steady state for the controller.

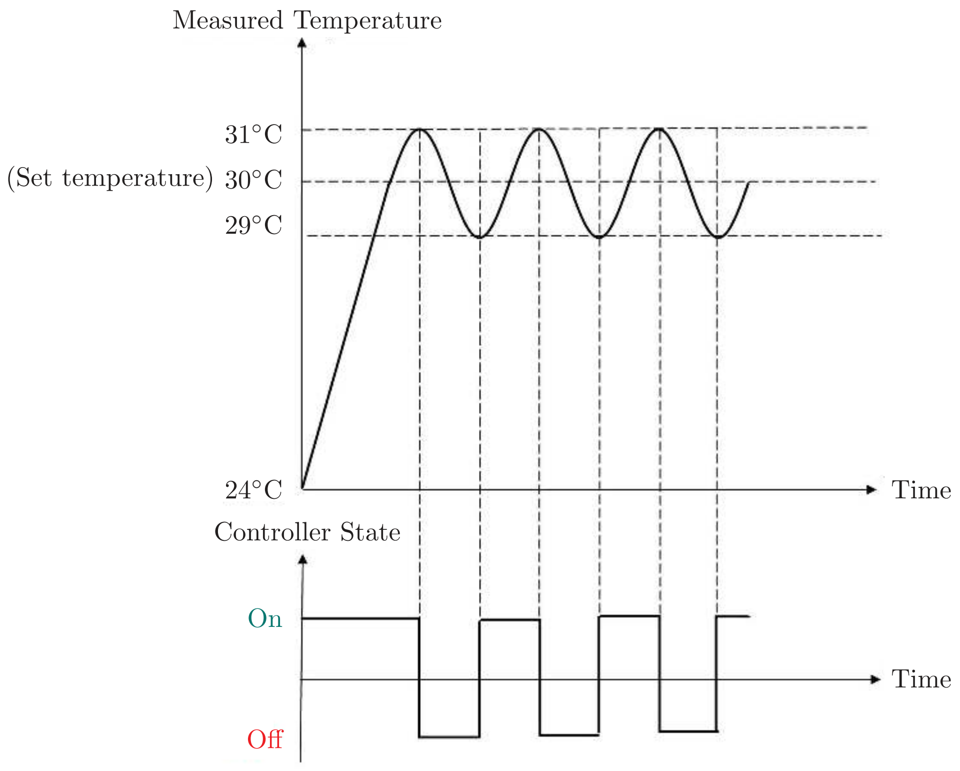


FIGURE 4.3
Measured temperature graph and the corresponding state of the controller.

4.2.2 Proportional Controller

One of the simplest forms of feedback controller is the proportional (P) gain controller, depicted in Figure 4.4. The action of the proportional controller is expressed as follows:

$$c(t) = K_p e(t) + b, \tag{4.3}$$

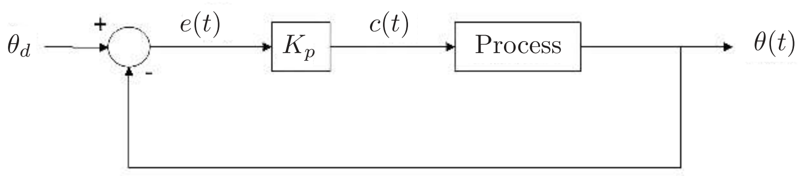


FIGURE 4.4
Proportional controller.

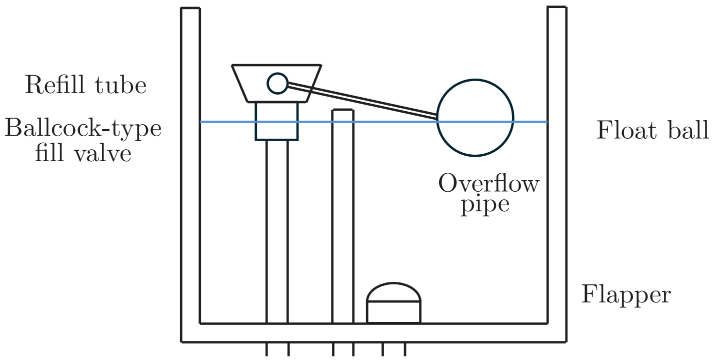


FIGURE 4.5
Toilet tank.

$$e(t) = q_d - q(t) \tag{4.4}$$

where $c(t)$ is the controller output, K_p is the proportional gain, $e(t)$ is the error between the current ($q(t)$) and desired (q_d) actuator positions respectively, and b is the bias term that defines the controller output when the error is zero. Unlike the bang-bang controller, which operates at either 0% or 100 % depending on the sign of the error, the proportional controller action is directly proportional to the sign and magnitude of the error. This means that the correction needed gets smaller as the output of the system approaches closer to the setpoint, and the fluctuation during steady-state can be minimised. However, fluctuation still exists, which creates an offset in the system output.

Example 4.2 (Proportional controller): Figure 4.5 shows a flush toilet tank with the ballcock mechanism design. Explain the correlation between a toilet tank and a proportional controller, and define the controller output.

Solution: The float ball, lift arm, and fill valve formulate a proportional control system. The error of the system (e) is the difference between the current ($h(t)$) and desired (h_d) water level, where the current water level is obtained through the position of the float ball. The controller output ($c(t)$) is the flow rate of water ($\dot{V}(t)$) coming into the tank through the opening of the fill valve. The proportional gain (K_p) is inversely proportional to the length of the lift arm (l). When the arm length decreases, the opening of the fill valve becomes more sensitive to the change in water level. Since we would like to stop the water from flowing into the tank when it reaches the desired water level, there will be no bias.

Therefore, the controller output, which is also the water flow rate, can be expressed as follows:

$$\dot{V}(t) = \frac{e(t)}{l}, \tag{4.5}$$

where

$$e(t) = h_d - h(t). \tag{4.6}$$

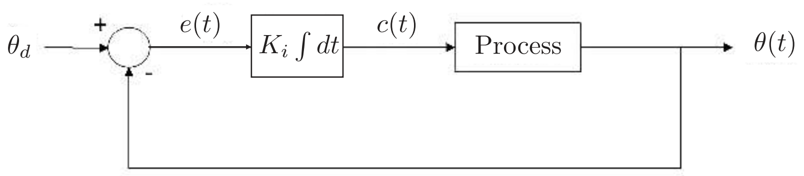


FIGURE 4.6
Integral controller.

4.2.3 Integral Controller

The issue with a proportional controller arises when the proportional gain is too high, leading to the system generating a consistent error at equilibrium, i.e., the steady-state error, or even risking instability due to increased gain. Consequently, the integral controller, illustrated in Figure 4.6, steps in to address this concern. The action of the integral controller is expressed as follows:

$$c(t) = K_i \int_0^t e(t)dt, \tag{4.7}$$

$u(t)$, K_i, and $e(t)$ denoting the integral control action, the integral constant, and the error of system output, respectively. The integral function measures the cumulative errors over time. If the error is not zero, the integral function continually builds up its output. Consequently, this output drives the actuator harder to minimise fluctuations in the system's output, thereby reducing the error to zero.

4.2.4 Derivative Controller

Derivative controller, shown in Figure 4.7, functions as a form of feedforward control that aims to minimise the change of error, thereby maintain the system stability. The signal output of the controller is determined based on the rate of error change over time, with a more significant rate of error change resulting in a more pronounced controller response. This can be achieved by correlating the derivative of the error to the controller output, where the derivative of the error is taken with respect to time. Thus, the derivative control

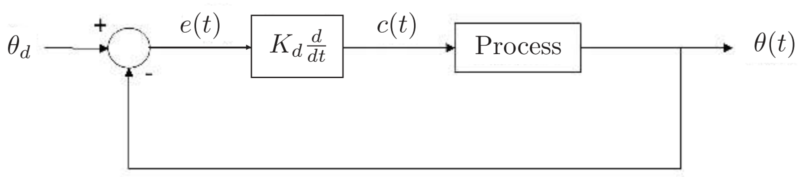


FIGURE 4.7
Derivative controller.

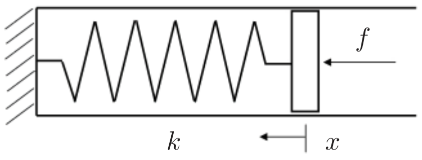


FIGURE 4.8
Spring loaded piston.

is mathematically illustrated as follows:

$$c(t) = K_d \frac{de(t)}{dt}, \tag{4.8}$$

where $u(t)$ is the derivative control action, K_d is the derivative constant, and $e(t)$ is the error of the system output.

The primary advantage of the derivative controller is its ability to counteract system changes, particularly oscillations. In contrast to proportional and integral controllers, derivative controllers do not lead the system directly to a steady state as it does not know where the setpoint is. Consequently, D controllers must be combined with P, I, or PI controllers for effective system regulation.

4.2.5 PI and PD Controller

An effective controller requires combining multiple controllers to achieve different purposes. The common combinations are Proportional-Integral (PI) controller, Proportional-Derivative (PD) controller, and Proportional-Integral-Derivative (PID) controller. Proportional action is necessary for quickly driving the system to the setpoint. The PI-control is a feedback control that provides a faster response time than an I-only control. This integral action helps eliminate fluctuations in a P-only controller. The behaviour of a PI controller can be mathematically expressed as

$$c(t) = K_p e(t) + K_i \int_0^t e(t)dt, \tag{4.9}$$

where the bias term of the proportional-only controller is replaced with the signal output of the integral action.

Example 4.3 (PI controller): Figure 4.8 shows a spring-loaded piston, where the piston stroke length measured along the x axis is controlled by the force F exerted on the piston surface. Assuming that the piston surface is massless, design a PI force controller to control the piston stroke length.

Solution:

Let x_d be the desired displacement of the piston, and $x(t)$ be the recorded piston displacement that is being fed back into our closed-loop system. We can design a block diagram for the PI controller, as shown in Figure 4.9. The behaviour of the PI force controller can be mathematically expressed as

$$F(t) = K_p e(t) + K_i \int_0^t e(t)dt, \tag{4.10}$$

where $e(t)$ is the difference between x_d and $x(t)$.

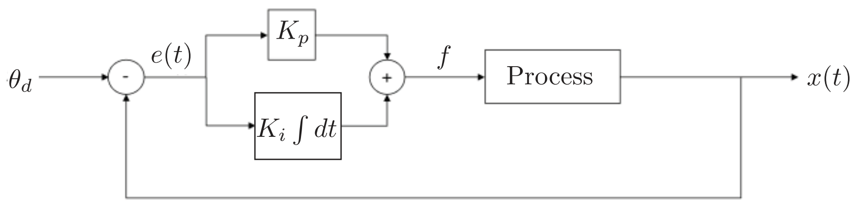


FIGURE 4.9
PI controller for the spring-loaded piston.

Explanation: Figure 4.10 shows that the I controller helps to compensate the required force as the error minimises to drive the piston to the desired position where the forces from both sides(acting force and the repulsive force from the loaded-spring) reaches equilibrium. Note that this will not work with a single P-controller, as the force output decreases whilst the spring force increases as the piston goes closer to the desired position. The result is that the error will not be minimised beyond $x_d/2$, and thus, a steady state cannot be reached.

PD control is a combination of feedforward and feedback control. The purpose of the derivative action is to predict future errors to improve the stability of the system. However, the steady-state error is not minimised without the integral term.

The proportional component is incorporated to accelerate the transient phase based on the system output, whilst the derivative component predicts the error in order to increase the stability of the closed-loop system.

The behaviour of a PD-controller can be mathematically expressed as

$$c(t) = K_p e(t) + K_d \frac{de(t)}{dt}, \tag{4.11}$$

where the bias term of the proportional-only controller is replaced with the signal output of the integral action.

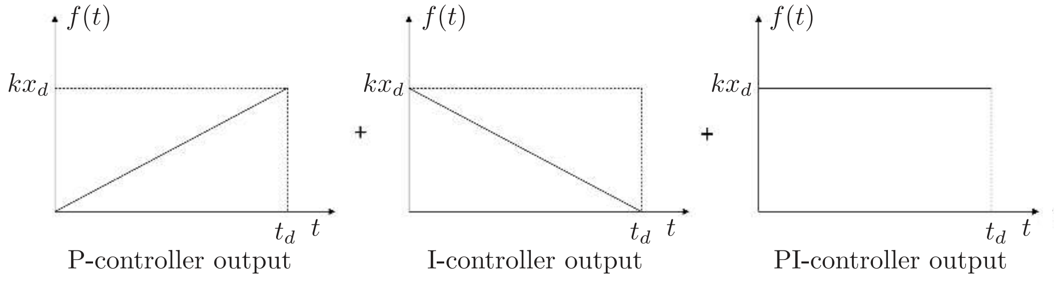


FIGURE 4.10
Force outputs of P, I, and PI controller.

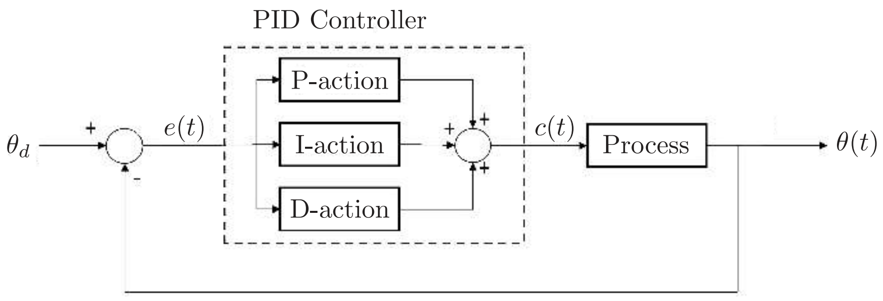


FIGURE 4.11
PID controller.

4.2.6 Proportional-Integral-Derivative (PID) Controller

The Proportional-Integral-Derivative (PID) Controller of Figure 4.11 is the most commonly used as it combines the best of both worlds. Aside from the inherent advantages of a PD controller, the addition of the Integral controller continuously integrates errors over time, eliminating steady-state errors and ensuring the system reaches and maintains the desired setpoint in a shorter period of time. PID controller can be mathematically expressed as

$$c(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}, \tag{4.12}$$

where the bias term of the proportional-only controller is replaced with the summation of the signal outputs from the integral and derivative actions.

Another advantage of a PID controller is its flexibility in tuning parameters to adapt to specific system requirements — it can be readily transformed into a PI or PD controller by setting K_d or K_i to zero.

The main characteristics that define the system's response include the Rise time (T_R), Settling time (T_S), overshoot, and steady-state error. An illustration of these characteristics can be seen in Figure 4.12. Rise time is the time taken for the response to first reach its

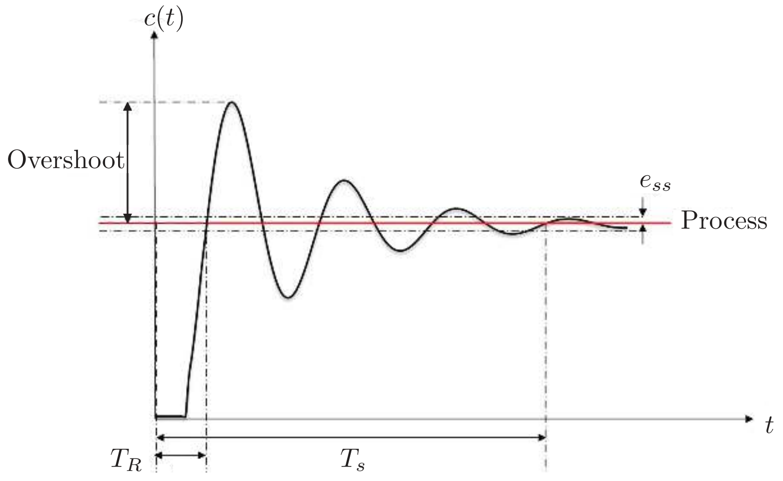


FIGURE 4.12
System response characteristics.

final steady-state value. Settling time is the time required for the system to converge to its steady state. Overshoot is the amount by which system output goes past the setpoint value. Steady-state error is the constant deviation from the setpoint value during steady-state.

The system response can be tailored by tuning K_p, K_i, and K_d. Increasing K_p would tend to increase the overshoot and decrease rise time and steady-state error. Increasing K_i tends to increase the overshoot and settling time and decrease the rise time and steady-state error. Increasing K_d tends to decrease overshoot and settling time. Nevertheless, while the PID controller often provides the optimal performance, it is also the most costly. As a result, it is only employed when the precision and stability offered by the PID controller are essential for the particular requirements of the process.

4.3 Pulse Width Modulation Control

Load devices are engineered to function efficiently and execute their specific tasks when given a specific voltage and consuming a particular current. Modifying the parameters of the load usually involves reducing the voltage. However, said adjustments are often associated with adverse effects, such as diminishing a motor's torque or dropping the voltage below the forward bias level of a transistor or LED series. An alternative approach is necessary to enable variable control without compromising operational capacity.

A method known as PWM (pulse width modulation) signal is employed to curtail the electrical power delivered to an electrical device by rapidly switching the signal on and off. The average voltage of the signal is adjusted by altering the relative on-time of the signal. This average voltage delivers a reduced power equivalent while ensuring that full voltage is maintained during the on-state duration of the pulse. One way to regulate the PWM signal is by controlling the relative duration of the on-time, referred to as the "duty cycle".

The duty cycle identifies the proportion (in percentage) of active time within a PWM signal. A 100% duty cycle maintains the signal continuously active, whereas a reduction to 50% entails the signal being active for half of the pulse duration and inactive for the remaining half. When managing motors or heaters, we rely on the duty cycle to govern power output. For instance, if our PWM controller generates a 12-volt DC voltage, a 50% duty cycle results in delivering an effective power of 6 volts DC to operate the load. Corresponding graphical representation is shown in Figure 4.13.

4.3.1 Relation to Joint Control

Pulse width modulation is commonly employed in DC motor speed control. However, we can take a step back and consider the implications of PWM control for an LED. Since pulse width modulation controls how long a device is turned on, one will observe this as a flashing LED under low frequencies. However, as frequency increases, this flashing eventually becomes invisible to the eye and becomes constant in brightness. This brightness is then controlled by the PWM's duty cycle. In the case of DC motors, the brightness of the LED and the rapid switching of the PWM signal are the analogues of the motor input voltage and motor speed control, respectively.

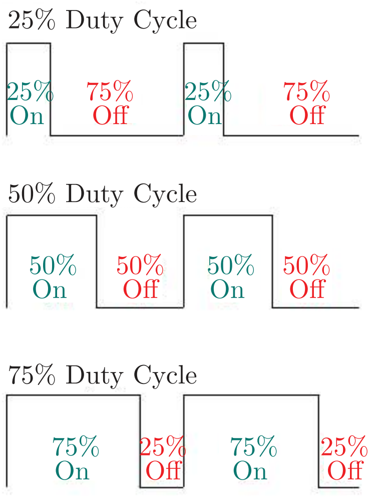


FIGURE 4.13
Pulse-width modulation.

4.4 Conclusion

In this chapter, we introduced open-loop and closed-loop systems as the two fundamental concepts in control schemes. An open-loop system operates on a fixed input state without actively monitoring the output; a closed-loop system uses the feedback from the system output to adjust the input so that the system can reach the desired output. We also discussed some of the most common closed-loop systems, which include bang-bang control, proportional (P) control, integral (I) control, derivative (D) control, PI control, PD control, and PID control. We discussed the working principles behind each of these controllers, as well as their advantages and disadvantages. The applications of some of these controllers were given as examples in the chapter. Lastly, we talked about Pulse Width Modulation (PWM) Control and how PWM can be used to modulate the power output through duty cycles.

4.5 Exercises

Problem 1. What is the difference between an open-loop and a closed-loop system? List one advantage and disadvantage of each system.

Problem 2. Draw a block diagram that represents the control system of a flush toilet tank.

Problem 3. Figure 4.14 shows the step response of a system. The input step has amplitude 1. Use the figure and determine

1. Steady state value
2. Overshoot

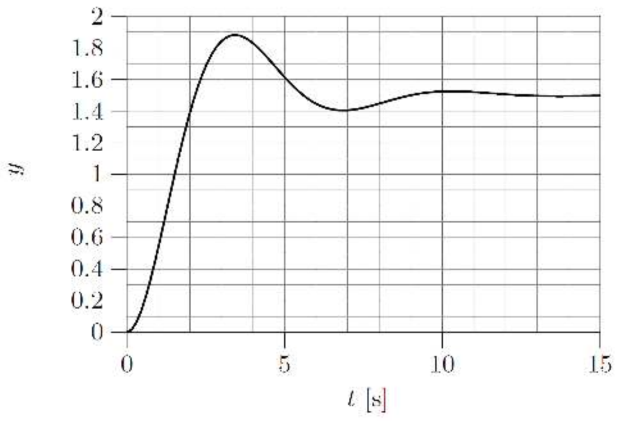


FIGURE 4.14
Step response of a system.

3. Rise time
4. Settling time

Problem 4. Given that the output state, $q(t) = 2ct + 4$, where c is the controller output of the proportional controller. Calculate the required proportional gain, K_d, to achieve the desired output state of 4.

Problem 5. Pulse Width Modulation Control is an effective method to control the brightness of an LED. Given an LED that operates at 50 Hz with 75% brightness, what is the time duration the LED is on within one cycle?

Part II

Key Topics

5

General Rotations and Transformations

In Chapter 2, we introduced the concept of position and orientation (pose) in a two-dimensional space. Here, we will generalise the discussion to describe the position and orientation of an object in three-dimensional space. These concepts will lay the foundation for understanding and performing robotic analysis for general spatial robotic manipulators.

5.1 Position and Orientation

A point and a Cartesian coordinate system, frame $\{0\}$, is shown in Figure 5.1(a). Its position can be represented by a vector

$$
{}^{0}\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{5.1}
$$

where the front upper index A indicates that this position vector is measured in $\{0\}$. Since the positions of the same point in different frames are different, the corresponding frame of a position vector must be specified.

Consider that a rigid body can rotate around a pivot in all three axes. The position of this rigid body can be treated as fixed at the pivot. As shown in Figure 5.1(b), where $\{0\}$ and $\{1\}$ located at the pivot are attached to the ground and the moving rigid body, respectively. The orientation of this rigid body can be described by a rotation matrix between these two frames

$$
{}^{0}\mathbf{R}_1 = \begin{bmatrix} {}^{0}\mathbf{x}_1 & {}^{0}\mathbf{y}_1 & {}^{0}\mathbf{z}_1 \end{bmatrix} \tag{5.2}
$$

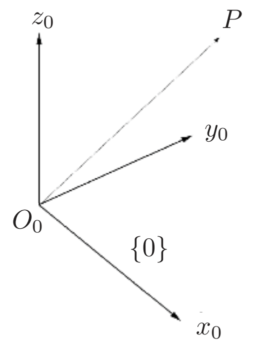
where the indices A and B in ${}^{0}\mathbf{R}_1$ indicate the orientation is about $\{1\}$ with respect to $\{0\}$. ${}^{0}\mathbf{x}_1$, ${}^{0}\mathbf{y}_1$, and ${}^{0}\mathbf{z}_1$ are the unit axis-directions of $\{1\}$ measured in $\{0\}$, as position vectors.

Example 5.1 (Rotation matrices): As shown in Figure 5.2, rotations of roll, pitch, and yaw are around the axes of x, y, and z, respectively. Find the orientations of $\{1\}$ in $\{0\}$.

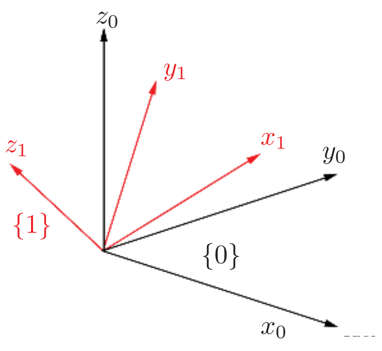
Solution: According to (5.2), the orientations due to these three types of simple rotations are given by

$$
{}^{0}\mathbf{R}_{1x}(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{5.3}
$$

DOI: 10.1201/9781003614319-5



(a) A point mass in space



(b) A rigid body in space

FIGURE 5.1
Frame definitions.

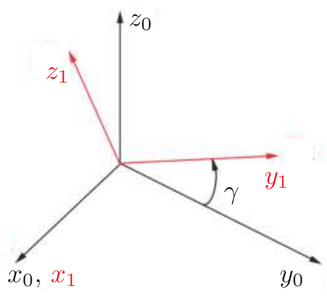
$$
{}^0\mathbf{R}_{1y}(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{5.4}
$$

$$
{}^0\mathbf{R}_{1z}(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.5}
$$

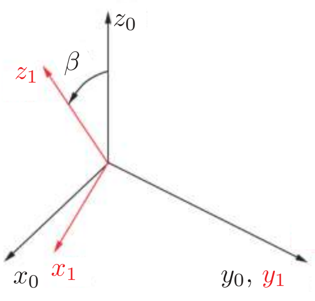
Rotation matrices consist of a special group of *orthonormal matrices* with the following properties:

1. all column vectors and row vectors are unit vectors,
2. all column vectors are orthogonal to each other,
3. all row vectors are orthogonal to each other,
4. the determinant of any rotation matrix is positive 1[1], and

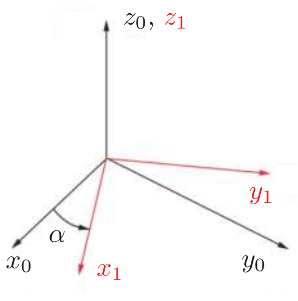
[1]Another group of orthonormal matrices with the determinant of -1 is the reflection matrices.



(a) Roll



(b) Pitch



(c) Yaw

FIGURE 5.2
Axes of rotation.

5. the inverse of a rotation matrix is its transpose such that

$$ {}^0\mathbf{R}_1^{-1} = {}^1\mathbf{R}_0 = {}^0\mathbf{R}_1^T \tag{5.6} $$

In particular, the last property is advantageous in analytical and numerical computations, as a transpose-derived inverse is more efficient than generalised square matrix inverse methods.

5.1.1 Functions of a Rotation Matrix

There are two important physical functions of a rotation matrix. The first is that a rotation matrix ${}^0\mathbf{R}_1$ maps the coordinates of a position vector measured in $\{1\}$, ${}^1\mathbf{p}$, to the coordinates of the same vector measured in $\{0\}$, ${}^0\mathbf{p}$, where

$$ {}^0\mathbf{p} = {}^0\mathbf{R}_1 \, {}^1\mathbf{p} \tag{5.7} $$

Example 5.2 (Rotation matrix operator — axes): Frames $\{0\}$ and $\{1\}$ are initially coincident. If a rotation is applied to $\{1\}$ about $\{0\}$'s z-axis, find the mapping of $\mathbf{x}_1$ to $\{0\}$.

Solution: A rotation about the z-axis is applied. Therefore, we can refer to the rotation matrix (5.5), which provides this operation. Also note that $\mathbf{x}_1$ refers to $\{1\}$'s x-axis relative to its own frame, such that ${}^1\mathbf{x}_1 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$. By inspection, we find that the mapping of the x-axis relative to its initial frame $\{0\}$ is indicated by the first column, which is $\begin{bmatrix}\cos\alpha & \sin\alpha & 0\end{bmatrix}^T$. We can also find this by using the equation

$$ {}^0\mathbf{x}_1 = {}^0\mathbf{R}_{1z}(\alpha)\, {}^1\mathbf{x}_1 \tag{5.8} $$

The second physical function of a rotation matrix is that ${}^0\mathbf{R}_1$ maps the position of $\mathbf{p}$ before rotation to its position after rotation, $\mathbf{p}'$, i.e.,

$$ \mathbf{p}' = {}^0\mathbf{R}_1 \, \mathbf{p} \tag{5.9} $$

where $\{0\}$ and $\{1\}$ represent the orientations of the rigid body before and after rotation, respectively. Furthermore, $\mathbf{p}$ and $\mathbf{p}'$ are both measured in the ground frame.

Consider the same figure, Figure 5.2(c). The x-axis is carried from x_0 to x_1 by the rotation around the z axis. Hence, we have $\mathbf{p} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$ and $\mathbf{p}' = \begin{bmatrix}\cos\alpha & \sin\alpha & 0\end{bmatrix}^T$. Both vectors are measured in the ground frame $\{0\}$ while the rotation matrix is exactly the same as the one in Example 5.2.

5.2 General Orientation

The orientation of one arbitrary frame $\{1\}$ with respect to another arbitrary frame $\{0\}$, as shown in Figure 5.3, is given by

$$ {}^0\mathbf{R}_1 = \begin{bmatrix}{}^0\mathbf{x}_1 & {}^0\mathbf{y}_1 & {}^0\mathbf{z}_1\end{bmatrix} = \begin{bmatrix} \mathbf{x}_1\cdot\mathbf{x}_0 & \mathbf{y}_1\cdot\mathbf{x}_0 & \mathbf{z}_1\cdot\mathbf{x}_0 \\ \mathbf{x}_1\cdot\mathbf{y}_0 & \mathbf{y}_1\cdot\mathbf{y}_0 & \mathbf{z}_1\cdot\mathbf{y}_0 \\ \mathbf{x}_1\cdot\mathbf{z}_0 & \mathbf{y}_1\cdot\mathbf{z}_0 & \mathbf{z}_1\cdot\mathbf{z}_0 \end{bmatrix} \tag{5.10} $$

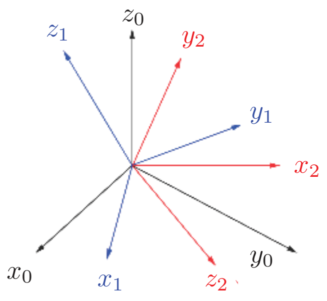


FIGURE 5.3
Three arbitrary frames located at the same point.

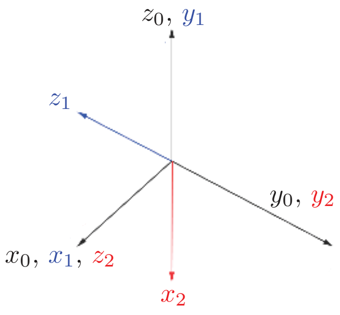


FIGURE 5.4
Three frames with their axes aligned with each other.

Example 5.3 (Rotation matrix operation): Find the rotation matrix ${}^0\mathbf{R}_1$, according to the frames shown in Figure 5.4.

Solution: The axes of {0} measured in {0} are given by

$$ {}^0\mathbf{x}_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad {}^0\mathbf{y}_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad {}^0\mathbf{z}_0 = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \tag{5.11} $$

The axes of {1} measured in {0} are given by

$$ {}^0\mathbf{x}_1 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, \quad {}^0\mathbf{y}_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad {}^0\mathbf{z}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{5.12} $$

According to (5.10), we have

$$ {}^0\mathbf{R}_1 = \begin{bmatrix} \mathbf{x}_1 \cdot \mathbf{x}_0 & \mathbf{y}_1 \cdot \mathbf{x}_0 & \mathbf{z}_1 \cdot \mathbf{x}_0 \\ \mathbf{x}_1 \cdot \mathbf{y}_0 & \mathbf{y}_1 \cdot \mathbf{y}_0 & \mathbf{z}_1 \cdot \mathbf{y}_0 \\ \mathbf{x}_1 \cdot \mathbf{z}_0 & \mathbf{y}_1 \cdot \mathbf{z}_0 & \mathbf{z}_1 \cdot \mathbf{z}_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.13} $$

An orientation can be well represented by a rotation matrix. However, due to the property of an orthonormal matrix, six constraints are applied to the nine entries in the matrix, which means that only three entries are independent. Hence, three-parameter representations are desired.

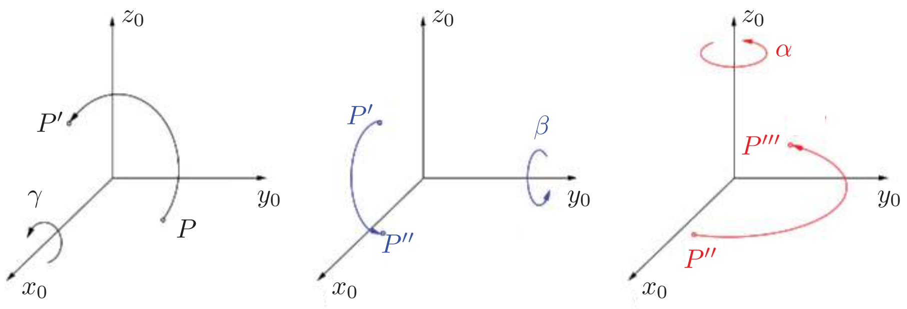


FIGURE 5.5
Fixed angles.

5.3 Fixed Angles

As shown in Figure 5.5, any arbitrary orientation of one object can be obtained by a sequence of rotations with angles γ, β, and α, around the fixed axes of $\{0\}$, x_0, y_0, and z_0, respectively. These angles are called the X-Y-Z fixed angles. In other words, any arbitrary orientation can be obtained by a sequence of roll, pitch, and yaw motions.

5.3.1 Forward Problem

The forward problem is to find the rotation matrix represented by three fixed angles. Consider a vector $\mathbf{p}$ before all three rotations. According to the second meaning of a rotation matrix (5.9), the first rotation around x-axis brings $\mathbf{p}$ to $\mathbf{p}'$, the second rotation around y-axis brings $\mathbf{p}'$ to $\mathbf{p}''$, and the third rotation around z-axis brings $\mathbf{p}''$ to $\mathbf{p}'''$, i.e.,

$$\mathbf{p}' = \mathbf{R}_x(\gamma)\,\mathbf{p}, \quad \mathbf{p}'' = \mathbf{R}_y(\beta)\,\mathbf{p}', \quad \mathbf{p}''' = \mathbf{R}_z(\alpha)\,\mathbf{p}''$$

which yields

$$\mathbf{p}''' = \mathbf{R}_z(\alpha)\;\mathbf{R}_y(\beta)\;\mathbf{R}_x(\gamma)\,\mathbf{p}$$

Again, according to the second meaning (5.9) and the above equation, the total rotation matrix is given by

$$\begin{aligned}
\mathbf{R}_{XYZ}(\gamma,\beta,\alpha) &= \mathbf{R}_z(\alpha)\;\mathbf{R}_y(\beta)\;\mathbf{R}_x(\gamma) \\
&= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \\
&= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \qquad (5.14)
\end{aligned}$$

Example 5.4 (Fixed angle rotation): Given the X-Y-Z fixed angles, $\gamma = 90°$, $\beta = 180°$, and $\alpha = 90°$, find the corresponding rotation matrix.

Solution: Substituting the above values into (5.14) yields

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.15}$$

5.3.2 Inverse Problem

The inverse problem in fixed angles is to find the fixed angles, γ, β, and α, to represent an arbitrary orientation (rotation matrix). The formulation of this problem is given by

$${}^0\mathbf{R}_1 = {}^0\mathbf{R}_{1XYZ} \tag{5.16}$$

where

$${}^0\mathbf{R}_1 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \quad {}^0\mathbf{R}_{1XYZ} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \tag{5.17}$$

Consider the entry (3,1) of (5.16), we have the solutions of β as $\arcsin(-r_{31})$ or $180° - \arcsin(-r_{31})$. For a unique solution of β, we restrict β within the range of $[-90°, 90°]$. Hence, the only solution of β is $\arcsin(-r_{31})$. Using the entries (1,1) and (2,1) of (5.16), we have

$$\begin{aligned} c\alpha &= r_{11}/c\beta \\ s\alpha &= r_{21}/c\beta \end{aligned} \tag{5.18}$$

We consider $(c\alpha, s\alpha)$ as the coordinates of a point on a unit circle. Therefore, α is the polar angle uniquely defined by this point. Atan2 is a function that finds the polar angle of an arbitrary point (except the origin), (x, y), on a plane. It is defined as

$$\text{Atan2}(y, x) = \begin{cases} \arctan(y/x) & x \geq 0 \\ \pi + \arctan(y/x) & x < 0 \end{cases} \tag{5.19}$$

Applying (5.19) to (5.18) yields a unique solution of α, i.e.,

$$\alpha = \text{Atan2}(r_{21}/c\beta, r_{11}/c\beta) \tag{5.20}$$

Since $c\beta$ is positive within the valid range of β except for the boundaries and Atan2 does not require a point to be bounded to a unit circle, the solution can be simplified as

$$\alpha = \text{Atan2}(r_{21}, r_{11}) \tag{5.21}$$

Similarly, γ can be obtained by solving the entries (3,2) and (3,3) of (5.16), i.e.,

$$\gamma = \text{Atan2}(r_{32}, r_{33}) \tag{5.22}$$

In summary, the solutions are given by

$$\beta = \arcsin(-r_{31}) \tag{5.23}$$

$$\alpha = \text{Atan2}(r_{21}, r_{11}) \tag{5.24}$$

$$\gamma = \text{Atan2}(r_{32}, r_{33}) \tag{5.25}$$

Example 5.5 (Fixed angles — inverse problem): Find out the fixed angles for the rotation matrix

$$
{}^{0}\mathbf{R}_1 = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.26}
$$

Solution: Directly applying (5.23) to (5.25) to yields

$$
\begin{aligned}
\beta &= \arcsin(-r_{31}) = \arcsin(0) = 0^\circ \\
\alpha &= \arctan 2(r_{21}, r_{11}) = \arctan 2(-1, 0) = -90^\circ \\
\gamma &= \arctan 2(r_{32}, r_{33}) = \arctan 2(-1, 0) = -90^\circ
\end{aligned}
$$

Note that the solutions differ from the fixed angles in Example 5.4 despite being in the same orientation.

The solutions (5.23) to (5.25) are invalid on the boundaries, $\beta = \pm 90^\circ$ because it is a *singularity*. This is caused by the alignment of the first x-axis and the last z-axis resulting from the right-angle rotation around the y-axis. Therefore, the first and last rotations are coupled, and only the sum of these two angles of rotation matters. Therefore, there are infinitely many solutions. In the case of $\beta = 90^\circ$, (5.14) becomes

$$
\mathbf{R}_{XYZ}(\gamma, \beta, \alpha) = \begin{bmatrix} 0 & \sin(\gamma - \alpha) & \cos(\gamma - \alpha) \\ 0 & \cos(\gamma - \alpha) & -\sin(\gamma - \alpha) \\ -1 & 0 & 0 \end{bmatrix} \tag{5.27}
$$

By assuming $\alpha = 0^\circ$, the special solutions are given by

$$
\begin{aligned}
\beta &= 90^\circ \\
\alpha &= 0^\circ \\
\gamma &= \text{Atan2}(r_{12}, r_{22})
\end{aligned}
$$

Similarly, in the case of $\beta = -90^\circ$, the special solutions are given by

$$
\begin{aligned}
\beta &= -90^\circ \\
\alpha &= 0^\circ \\
\gamma &= -\text{Atan2}(r_{12}, r_{22})
\end{aligned}
$$

It should be noted that the X-Y-Z fixed angles is not the only convention in the representation of fixed angles. There are in total 12 different conventions: X-Y-X, X-Z-X, Y-Z-Y, Y-X-Y, Z-X-Z, Z-Y-Z, X-Y-Z, Y-Z-X, Z-X-Y, X-Z-Y, Z-Y-X, and Y-X-Z. Each convention has its own solutions to the direct and inverse problems.

5.4 Euler Angles

Any arbitrary orientation of one object can also be obtained by a sequence of rotations with angles α, β, and γ, around the *moving axes*, z_1, y_1, and x_1, respectively, as shown in Figure 5.6. Similarly, we have two problems: direct and inverse problems. The former is to find a rotation matrix according to the given Euler angles, while the latter is to obtain the corresponding Euler angles based on a given rotation matrix.

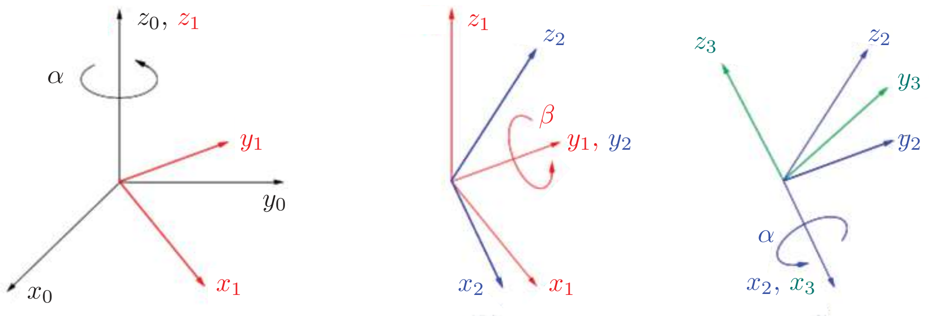


FIGURE 5.6
The *Z*-*Y*-*X* Euler angles.

5.4.1 Forward Problem

The solution to the forward problem is derived from the first meaning of a rotation matrix (5.7), which is the mapping between coordinates. As shown in Figure 5.6, {0}, {1}, {2}, and {3} represent the frames before all rotations, after the first rotation around the *z*-axis, after the second rotation around the *y*-axis, and after the third rotation around the *z*-axis, respectively. Considering an arbitrary position vector $\mathbf{p}$, we have its coordinates in {0} as

$$^{0}\mathbf{p} = {}^{0}\mathbf{R}_1\,{}^{1}\mathbf{p} = {}^{0}\mathbf{R}_1\,{}^{1}\mathbf{R}_2\,{}^{2}\mathbf{p} = {}^{0}\mathbf{R}_1\,{}^{1}\mathbf{R}_2\,{}^{2}\mathbf{R}_3\,{}^{3}\mathbf{p} = \mathbf{R}_z\,\mathbf{R}_y\,\mathbf{R}_x\,{}^{3}\mathbf{p} \tag{5.28}$$

Again, utilising the first meaning (5.7) and the above equation, we have the total rotation matrix as

$$\begin{aligned}
\mathbf{R}_{ZYX}(\alpha,\beta,\gamma) &= \mathbf{R}_z(\alpha)\,\mathbf{R}_y(\beta)\,\mathbf{R}_x(\gamma) \\
&= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \\
&= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
\end{aligned}$$

Note that this *Z*-*Y*-*X* Euler-angle rotation matrix is the same as the *X*-*Y*-*Z* fixed-angle rotation matrix. However, the meanings of these rotation angles are entirely different.

Example 5.6 (Euler angles): Find the rotation matrix (orientation) by knowing the *Z*-*Y*-*X* Euler angles, $\alpha = 90°$, $\beta = 180°$, and $\gamma = 90°$.

Solution: The orientation is obtained by applying (5.29):

$$^{0}\mathbf{R}_1 = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.29}$$

5.4.2 Inverse Problem

The formulation of the inverse problem to find the *Z*-*Y*-*X* Euler angles, α, β, and γ, from an arbitrary orientation (rotation matrix) is given by

$$\begin{aligned} {}^0\mathbf{R}_1 &= {}^0\mathbf{R}_{1ZYX} \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \\ &= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \end{aligned} \tag{5.30}$$

Equation (5.30) is exactly the same as (5.16) in the inverse problem of the fixed angles; therefore, the solutions must also be the same as below, despite different meanings.

$$\beta = \arcsin(-r_{31}) \tag{5.31}$$

$$\alpha = \mathrm{Atan2}(r_{21}, r_{11}) \tag{5.32}$$

$$\gamma = \mathrm{Atan2}(r_{32}, r_{33}) \tag{5.33}$$

Similarly, there are singularities on the boundaries. In case of $\beta = 90°$, we have

$$\begin{aligned} \beta &= 90° \\ \alpha &= 0° \\ \gamma &= \arctan 2(r_{12}, r_{22}) \end{aligned}$$

In case of $\beta = -90°$, we have

$$\begin{aligned} \beta &= -90° \\ \alpha &= 0° \\ \gamma &= -\arctan 2(r_{12}, r_{22}) \end{aligned}$$

Example 5.7 (Euler angles): Find the Euler angles for the rotation matrix

$${}^0\mathbf{R}_1 = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.34}$$

Solution: Equations (5.31) to (5.33) give

$$\begin{aligned} \beta &= \mathrm{Atan2}(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}) = \mathrm{Atan2}(0, 1) = 0° \\ \alpha &= \mathrm{Atan2}(r_{21}/c\beta, r_{11}/c\beta) = \mathrm{Atan2}(-1, 0) = -90° \\ \gamma &= \mathrm{Atan2}(r_{32}/c\beta, r_{33}/c\beta) = \mathrm{Atan2}(-1, 0) = -90° \end{aligned}$$

Note that these solutions have entirely different meanings to the solution in Example 5.5, although the values are the same.

Similar to the fixed angles, Euler angles also have 12 conventions: *X*-*Y*-*X*, *X*-*Z*-*X*, *Y*-*Z*-*Y*, *Y*-*X*-*Y*, *Z*-*X*-*Z*, *Z*-*Y*-*Z*, *X*-*Y*-*Z*, *Y*-*Z*-*X*, *Z*-*X*-*Y*, *X*-*Z*-*Y*, *Z*-*Y*-*X*, and $Y - X - Z$[2]. Euler angles and fixed angles are dual to each other, i.e., the same angles in the opposite order of rotations yield the same rotation matrix.

[2]Some texts may distinguish three axes rotations as *Bryan-Tait* angles, which refer to the last six conventions in the list.

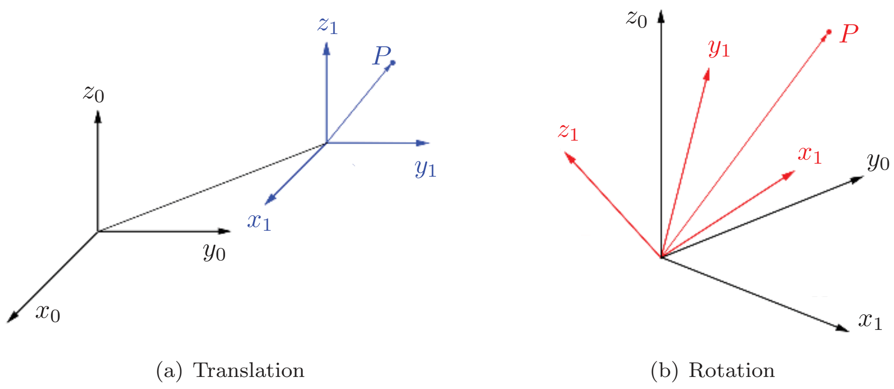


FIGURE 5.7
Modes of transformation.

5.5 General Transformation

Translation is illustrated in Figure 5.7(a) where {0} and {1} are parallel to each other. The coordinates of a position vector **p** are mapped by

$$^0\mathbf{p} = {}^0\mathbf{o}_1 + {}^1\mathbf{p} \tag{5.35}$$

where $^0\mathbf{o}_1$ is the position of the origin of {1} measured in {0}. Rotation has been studied intensively so far. As shown in Figure 5.7(b), the coordinates of a position vector **p** are mapped from {1} to {0} by

$$^0\mathbf{p} = {}^0\mathbf{R}_1 \, {}^1\mathbf{p} \tag{5.36}$$

Regarding two general frames, {0} and {1}, as shown in Figure 5.8. The coordinate mapping is given by

$$^0\mathbf{p} = {}^0\mathbf{o}_1 + {}^0\mathbf{R}_1 \, {}^1\mathbf{p} \tag{5.37}$$

which is a combination of rotation and translation in sequence. Equation (5.37) can be more simply represented by introducing homogeneous coordinates, which is defined as

$$^0\underline{\mathbf{p}} = \begin{bmatrix} ^0\mathbf{p} \\ 1 \end{bmatrix} \tag{5.38}$$

Hence, (5.37) can be written as

$$^0\underline{\mathbf{p}} = {}^0\mathbf{T}_1 \, {}^1\underline{\mathbf{p}} \tag{5.39}$$

where $^1\underline{\mathbf{p}} = \begin{bmatrix} ^1\mathbf{p} & 1 \end{bmatrix}^T$ and

$$^0\mathbf{T}_1 = \begin{bmatrix} ^0\mathbf{R}_1 & ^0\mathbf{o}_1 \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{5.40}$$

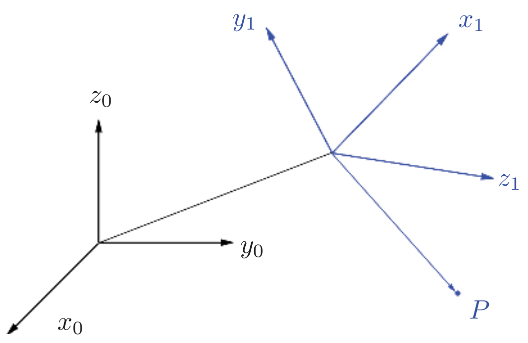


FIGURE 5.8
General transformation of two frames.

Example 5.8 (Transformations): Find ${}^0\mathbf{p}$, given

$$ {}^0\mathbf{R}_1 = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}, \quad {}^1\mathbf{p} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}, \quad {}^0\mathbf{o}_1 = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \tag{5.41} $$

Solution: This can be solved in two ways.

1) Three-dimensional vectors

$$ {}^0\mathbf{p} = {}^0\mathbf{o}_1 + {}^0\mathbf{R}_1\,{}^1\mathbf{p} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ -2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{5.42} $$

2) Homogeneous vectors

$$ {}^0\mathbf{T}_1 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^1\mathbf{p} = \begin{bmatrix} 2 \\ 1 \\ -1 \\ 1 \end{bmatrix} \tag{5.43} $$

Hence

$$ {}^0\mathbf{p} = {}^0\mathbf{T}_1\,{}^1\mathbf{p} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} \tag{5.44} $$

The inverse problem of the general mapping is to find the ${}^1\mathbf{p}$ by knowing ${}^0\mathbf{p}$, ${}^0\mathbf{o}_1$, and the rotation matrix ${}^0\mathbf{R}_1$ between $\{0\}$ and $\{1\}$. According to (5.37), we have

$$ {}^1\mathbf{p} = {}^0\mathbf{R}_1^{-1}({}^0\mathbf{p} - {}^0\mathbf{o}_1) = {}^0\mathbf{R}_1^T({}^0\mathbf{p} - {}^0\mathbf{o}_1) \tag{5.45} $$

which can be further understood as a sequence of translation and rotation backward. Using homogeneous coordinates, we obtain

$$ {}^1\mathbf{p} = {}^0\mathbf{T}_1^{-1}\,{}^0\mathbf{p} \tag{5.46} $$

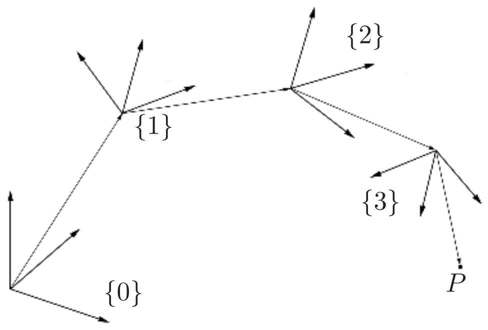


FIGURE 5.9
Multiple transformations.

where

$$
{}^0\mathbf{T}_1^{-1} = {}^1\mathbf{T}_0 = \begin{bmatrix} {}^0\mathbf{R}_1^T & -{}^0\mathbf{R}_1^T\,{}^0\mathbf{o}_1 \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{5.47}
$$

Example 5.9 (Transformation matrices): Find ${}^1\mathbf{p}$, given

$$
{}^0\mathbf{R}_1 = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}, \quad {}^0\mathbf{p} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, \quad {}^0\mathbf{o}_1 = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \tag{5.48}
$$

Solution: According to (5.46), we have

$$
-\,{}^0\mathbf{R}_1^T\,{}^0\mathbf{o}_1 = -\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} \tag{5.49}
$$

$$
{}^0\mathbf{T}_1^{-1} = \begin{bmatrix} 0 & -1 & 0 & 2 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.50}
$$

Hence,

$$
{}^1\mathbf{p} = {}^0\mathbf{T}_1^{-1}\,{}^0\mathbf{p} = \begin{bmatrix} 0 & -1 & 0 & 2 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ -1 \\ 1 \end{bmatrix} \tag{5.51}
$$

If multiple transformations are involved, as shown in Figure 5.9, the problem can be extremely tedious with normal 3D vectors. Homogeneous-vector expression provides an elegant way to describe such transform, where

$$
{}^0\mathbf{p} = {}^0\mathbf{T}_1\,{}^1\mathbf{T}_2\,{}^2\mathbf{T}_3\,{}^3\mathbf{p} \tag{5.52}
$$

Example 5.10 (Application of transformations): A helmet with LEDs is used to determine the pose of an object. Assume that Frame H is attached to the helmet, and the positions of the LEDs are known in $\{H\}$: ${}^{H}\mathbf{p}_i$ for $i = 1, 2, 3, 4$. Frame $\{C\}$ is the camera-based frame. The LEDs' positions are measured by cameras in $\{C\}$: ${}^{C}\mathbf{p}_i$ for $i = 1, 2, 3, 4$. With this setup, find the rotation matrix ${}^{C}\mathbf{R}_H$

Solution: According to (5.37), we have

$$ {}^{C}\mathbf{p}_i = {}^{C}\mathbf{o}_H + {}^{C}\mathbf{R}_H \, {}^{H}\mathbf{p}_i \tag{5.53}$$

for $i = 1, 2, 3, 4$. Subtracting one equation from another in (5.53) yields

$$ {}^{C}\mathbf{p}_i - {}^{C}\mathbf{p}_{i+1} = {}^{C}\mathbf{R}_H ({}^{H}\mathbf{p}_i - {}^{H}\mathbf{p}_{i+1}) \tag{5.54}$$

for $i = 1, 2, 3$. The above equations can be written in a matrix form as

$$ \mathbf{A} = {}^{C}\mathbf{R}_H \, \mathbf{B} \tag{5.55}$$

where

$$ \mathbf{A} = \begin{bmatrix} {}^{C}\mathbf{p}_1 - {}^{C}\mathbf{p}_2 & {}^{C}\mathbf{p}_2 - {}^{C}\mathbf{p}_3 & {}^{C}\mathbf{p}_3 - {}^{C}\mathbf{p}_4 \end{bmatrix} \tag{5.56}$$

$$ \mathbf{B} = \begin{bmatrix} {}^{H}\mathbf{p}_1 - {}^{H}\mathbf{p}_2 & {}^{H}\mathbf{p}_2 - {}^{H}\mathbf{p}_3 & {}^{H}\mathbf{p}_3 - {}^{H}\mathbf{p}_4 \end{bmatrix} \tag{5.57}$$

Solving the above matrix equation yields

$$ {}^{C}\mathbf{R}_H = \mathbf{A} \, \mathbf{B}^{-1} \tag{5.58}$$

Note that the minimum number of LEDs for the orientation detection is 4. Further, if all LEDs are on the same plane, $\mathbf{B}$ will be singular, which yields no solution.

5.6 MATLAB® Examples

5.6.1 Matrix Arithmetic

MATLAB can be used to perform matrix arithmetic for linear transforms. However, variable names must be consistent to decrease the chance of computational errors. The examples in this section follow a logical variable-naming convention that is clear and unambiguous and recommended for this unit. This will become especially important when submitting MATLAB code used in any assessments.

Example M5.1 (Rotation matrices): Find rotation matrices ${}^1\mathbf{R}_2$ and ${}^1\mathbf{R}_3$, given,

$$ {}^0\mathbf{R}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad {}^0\mathbf{R}_2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad {}^2\mathbf{R}_3 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{5.59} $$

Solution: The following script solves this problem.

```
% Define rotation matrices
R_0_1 = [1 0 0; 0 0 -1; 0 1 0];
R_0_2 = [-1 0 0; 0 -1 0; 0 0 1];
R_2_3 = [0 0 -1; 0 1 0; 1 0 0];

R_1_0 = R_0_1'            % R_0_1 inverse
R_1_2 = R_1_0 * R_0_2     % R_1_2 answer

R_1_3 = R_1_2 * R_2_3     % R_1_3 answer
```

```
>> ch2_1

R_1_0 =

     1     0     0
     0     0     1
     0    -1     0


R_1_2 =

    -1     0     0
     0     0     1
     0     1     0


R_1_3 =

     0     0     1
     1     0     0
     0     1     0
```

5.6.2 Inverse Transformation Matrix

The following MATLAB code calculates the inverse of a 4×4 homogeneous transformation matrix, according to (5.46). Note that this method cannot be used to calculate the inverse of general 4×4 matrices.

```
function [ T_B_A ] = invT( T_A_B )
%INVT Transformation matrix inverse
% Input
%   T_A_B   Transformation matrix of {B} measured in {A}
% Output
%   T_B_A   Transformation matrix of {A} measured in {B} (inverse)

R_A_B = T_A_B(1:3,1:3);      % Rotation matrix
p_B = T_A_B(1:3,4);          % Translation vector
T_B_A = [R_A_B', -R_A_B'*p_B; 0 0 0 1];  % Inverse

end
```

Example M5.2 (Matrix inverse): Find ${}^1\mathbf{T}_0$, given the transformation matrix

$$ {}^0\mathbf{T}_1 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.60} $$

Solution: There are a couple of ways to solve this. One may use the in-built function `inv()` in MATLAB, or use `invT()` as defined in Inline 5.3. The MATLAB command window shows the output of each result.

```
>> T_A_B = [0 0 1 1; -1 0 0 2; 0 -1 0 0; 0 0 0 1]

T_A_B =

     0     0     1     1
    -1     0     0     2
     0    -1     0     0
     0     0     0     1

>> T_B_A1 = invT(T_A_B)      % Homog matrix inverse

T_B_A1 =

     0    -1     0     2
     0     0    -1     0
     1     0     0    -1
     0     0     0     1

>> T_B_A2 = inv(T_A_B)       % General inverse

T_B_A2 =

     0    -1     0     2
     0     0    -1     0
     1     0     0    -1
     0     0     0     1
```

While we expected the result to be identical, in the application of linear transforms, we *always* want to use the homogeneous matrix inverse rather than the general inverse. The calculation of the general inverse introduces residual errors and is two to three times slower than the homogeneous matrix inverse operation. For more information, view the MATLAB documentation on `inv()`.

Example M5.3 (Transformation matrices):
In Figure 5.10, a robot is detecting an apple to be picked. Find the position of the detected apple in the tree relative to the gripper coordinates so that a trajectory for the gripper can be planned to grasp this apple successfully. In addition, find the position of the apple relative to the robot arm base frame so that the kinematics can be solved. The following details are known:

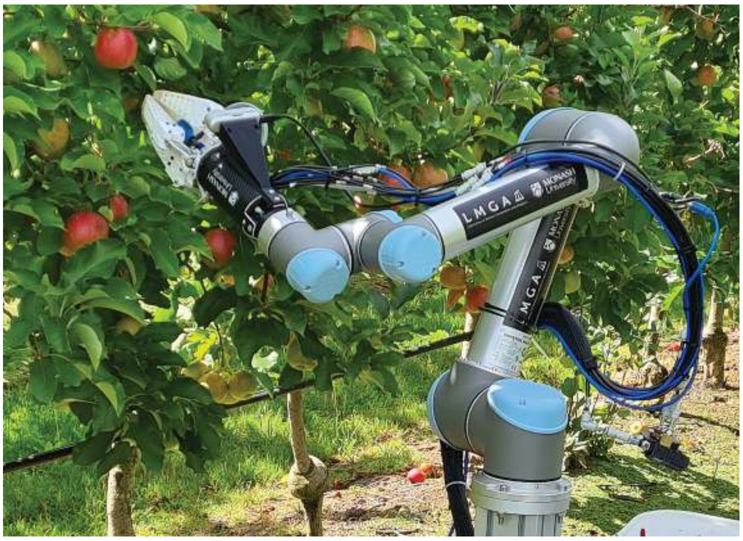


FIGURE 5.10
A UR5 robot inspecting an apple in a tree.

End-effector frame $\{E\}$ relative to the robot arm base $\{0\}$

$$
{}^{0}\mathbf{T}_E = \begin{bmatrix} 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0.2 \\ 0 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0.2 \\ 1 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.61}
$$

Gripper frame $\{G\}$ relative to the end-effector frame $\{E\}$

$$
{}^{E}\mathbf{T}_G = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.62}
$$

Camera frame $\{C\}$ relative to the end-effector frame $\{E\}$

$$
{}^{E}\mathbf{T}_C = \begin{bmatrix} 1 & 0 & 0 & 0.1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.63}
$$

Apple position in the camera coordinate frame

$$
{}^{C}\mathbf{p}_A = \begin{bmatrix} -0.05 \\ 0.5 \\ 0.4 \end{bmatrix} \tag{5.64}
$$

Solution: The position of the apple in the gripper frame is ${}^{G}\mathbf{p}_A$, and the position of the apple in the arm base frame is ${}^{0}\mathbf{p}_A$. These vectors can be found by

$$
\begin{aligned}
{}^{G}\mathbf{p}_A &= {}^{G}\mathbf{T}_E \, {}^{E}\mathbf{T}_C \, {}^{C}\mathbf{p}_A \\
&= {}^{E}\mathbf{T}_G^{-1} \, {}^{E}\mathbf{T}_C \, {}^{C}\mathbf{p}_A
\end{aligned} \tag{5.65}
$$

and

$$
{}^{0}\mathbf{p}_A = {}^{0}\mathbf{T}_E \, {}^{E}\mathbf{T}_C \, {}^{C}\mathbf{p}_A \tag{5.66}
$$

The following script solves this problem.

```
% End-effector frame {E} relative to arm base frame {0}
T_0_E = [   0    sqrt(2)/2    sqrt(2)/2    0.2;
            0    -sqrt(2)/2   sqrt(2)/2    0.2;
            1    0            0            0.3;
            0    0            0            1];

% Camera frame {C} relative to end-effector frame {E}
T_E_C = [1 0 0 0.1; 0 1 0 0; 0 0 1 0.1; 0 0 0 1];

% Gripper frame {G} relative to end-effector frame {E}
T_E_G = [0 0 1 0; 1 0 0 0; 0 1 0 0.2; 0 0 0 1];

% Detected position of the apple in the camera frame
P_C_A = [-0.05 -0.5 0.4 1]';

% Apple position in gripper co-ordinates
T_G_E = invT(T_E_G)              % T_E_G inverse
P_G_A = T_G_E * T_E_C * P_C_A    % Answer

% Apple position in the arm base frame co-ordinates
P_0_A = T_0_E * T_E_C * P_C_A    % Answer
```

```
T_G_E =

         0    1.0000         0         0
         0         0    1.0000   -0.2000
    1.0000         0         0         0
         0         0         0    1.0000


P_G_A =

   -0.5000
    0.3000
    0.0500
    1.0000


P_0_A =

    0.2000
    0.9071
    0.3500
    1.0000
```

Therefore, the apple location in gripper coordinates (in metres) is

$$^{G}\mathbf{p}_A = \begin{bmatrix} -0.5 \\ 0.2 \\ 0.05 \end{bmatrix} \tag{5.67}$$

and

$$^{0}\mathbf{p}_A = \begin{bmatrix} 0.2 \\ 0.9071 \\ 0.35 \end{bmatrix} \tag{5.68}$$

5.7 Conclusion

In this chapter, we described how to define a rigid body's pose in space. The position of a rigid body can be described as a three-dimensional vector representing a point in Cartesian space, and its orientation can be described with a 3×3 orthogonal *rotation* matrix. Furthermore, with the use of *frames* and matrix arithmetic, we explored how we can use these matrices to apply rotational operations to three-dimensional vectors.

The representation for a rotation is not unique, where rotations can be applied relative to a fixed coordinate system (*fixed-angle*), or relative to its own moving frame (*Euler angle*). In robotics, either mode of applying rotations is valid, and its use is simply dependent on the convenience of the application. Finally, the general transformation was introduced as a 4×4 homogeneous matrix, which encapsulates both the position and orientation of a rigid body. The transformation matrix is used in frame definitions, and applying translational and rotational transformations to other frames.

The orthogonal nature of rotation matrices and the homogeneity of transformation matrices make them very intuitive to use mathematically, such as applying rotations or transforms in a sequence, or finding their inverse. Examples of rotation and transformation matrices were given as MATLAB examples at the end of this chapter.

5.8 Exercises

Problem 1. Given that vector ${}^1\mathbf{p}$ undergoes two separate cases of rotations in the given orders:

1. x_1 by θ degrees and is followed by rotation about y_1 by ϕ degrees.
2. y_1 by $30°$ degrees and is followed by rotation about x_1 by $30°$ degrees.

Calculate the rotation matrix and the state of the rotation style for each case.

Problem 2. A frame $\{2\}$ is initially coincident with another frame, $\{1\}$. Frame $\{2\}$ is rotated about x_2 by θ, and then the resulting frame is rotated about x_2 by ϕ. What style of rotation is this? Find the rotation matrix that will change the descriptions of vectors from ${}^2\mathbf{p}$ to ${}^1\mathbf{p}$.

Problem 3. Find ${}^1\mathbf{T}_3$ by using the following frame definitions:

$$
{}^1\mathbf{T}_0 = \begin{bmatrix} 1 & 0 & 0 & -10 \\ 0 & 0 & -1 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^3\mathbf{T}_0 = \begin{bmatrix} 0.1268 & -0.6124 & 0.7803 & 10 \\ 0.9268 & 0.3536 & 0.1268 & 8 \\ -0.3536 & 0.7071 & 0.6124 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Problem 4. Frame $\{2\}$ is initially coincident with frame $\{1\}$. Frame $\{2\}$ is rotated about z_2 by $60°$, then about x_2 by $45°$ and then about y_2 by $30°$. Finally, the origin of $\{2\}$ is translated to $\begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T = \begin{bmatrix} 3 & 8 & -6 \end{bmatrix}^T$.

1. What is the transformation matrix ${}^1\mathbf{T}_2$?
2. What is the transformation matrix ${}^2\mathbf{T}_1$?
3. The position of a point in $\{2\}$ is ${}^2\mathbf{p} = \begin{bmatrix} 1 & -2 & 3 \end{bmatrix}^T$, find the coordinates of this point in $\{1\}$.
4. The position of a point in $\{1\}$ is ${}^1\mathbf{p} = \begin{bmatrix} 6 & 4 & -1 \end{bmatrix}^T$, find the coordinates of this point in $\{2\}$.

Problem 5. An Angle-Axis describes a fixed rotation about a fixed vector. The formula to generate a transformation matrix using the Angle-Axis representation is given below:

$$
{}^1\mathbf{T}_2 = \begin{bmatrix} k_x k_x(1-c\theta)+c\theta & k_x k_y(1-c\theta)-k_z s\theta & k_x k_z(1-c\theta)+k_y s\theta & 0 \\ k_x k_y(1-c\theta)+k_z s\theta & k_y k_y(1-c\theta)+c\theta & k_y k_z(1-c\theta)-k_x s\theta & 0 \\ k_x k_z(1-c\theta)-k_y s\theta & k_y k_z(1-c\theta)+k_x s\theta & k_z k_z(1-c\theta)+c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

There are two assumptions made when using this formula:

1. The vector $\mathbf{k}$ is a unit vector.
2. The axis of rotation passes through the origin of the current frame.

Given frame {2} is initially coincident with frame {1}. Frame {2} is rotated by 35° about a unit vector $\mathbf{k}$, where $\mathbf{k}$ passes through the point P. Find the transformation matrix ${}^1\mathbf{T}_2$ for each of the following circumstances:

1. $\mathbf{k} = \begin{bmatrix}0.8018 & -0.2673 & 0.5345\end{bmatrix}^T$, the position of point P is ${}^1\mathbf{p}{=} \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$.
2. $\mathbf{k} = \begin{bmatrix}0.8018 & -0.2673 & 0.5345\end{bmatrix}^T$, the position of point P is ${}^1\mathbf{p}{=} \begin{bmatrix}0 & 0 & 3\end{bmatrix}^T$.

Problem 6. A velocity vector is given by

$${}^0\mathbf{v} = \begin{bmatrix}0\\0\\10\end{bmatrix}$$

Given

$${}^0\mathbf{T}_1 = \begin{bmatrix}1 & 0 & 0 & 5\\0 & 0.5 & -0.866 & 1\\0 & -0.866 & 0.5 & 1\\0 & 0 & 0 & 1\end{bmatrix}$$

Compute ${}^1\mathbf{v}$. Hint: Consider the properties of velocity.

Problem 7.

1. Two frames: {0} and {1} are related through a transformation matrix given by

$${}^0\mathbf{T}_1 = \begin{bmatrix}\sqrt{2}/2 & -\sqrt{2}/2 & 0 & 3\\0 & 0 & -1 & 2\\\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

   Find transformation matrix ${}^1\mathbf{T}_0$.

2. Point P is located at ${}^1\mathbf{p} = \begin{bmatrix}1 & 2 & 4\end{bmatrix}^T$ with respect to the origin of frame {1}, as shown in Figure 5.11. Frame {1} is related to {0} through the transformation matrix ${}^0\mathbf{T}_1$ given previously. Evaluate the position of point P with respect to the origin of {0}, i.e., evaluate ${}^0\mathbf{p}$.

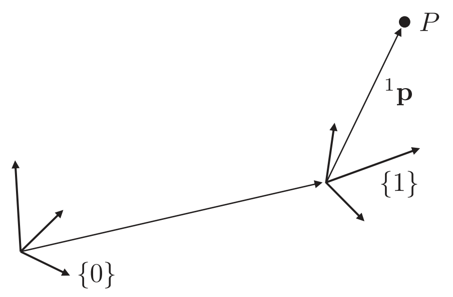


FIGURE 5.11
Frame 0, frame 1, and point P.

Problem 8. A rigid body rotates around a pivot point. Two frames sitting on the pivot point are assigned: $\{0\}$ is fixed to the ground, and $\{1\}$ is attached to the rigid body. Initially, both frames coincide with each other. Rotate rigid body around the x axis of $\{0\}$ of θ_1, and then rotate the rigid body around the z axis of $\{0\}$ of θ_2. Three markers attached to the rigid body are used to track the rotation. The markers' positions in $\{1\}$ are given by

$$^{1}\mathbf{p}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, {}^{1}\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, {}^{1}\mathbf{p}_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

After the above two rotations, the three markers' positions in Frame A are recorded as

$$^{0}\mathbf{p}_1 = \begin{bmatrix} 0.500 \\ 0.866 \\ 0.000 \end{bmatrix}, {}^{0}\mathbf{p}_2 = \begin{bmatrix} -0.250 \\ 1.299 \\ 0.500 \end{bmatrix}, {}^{0}\mathbf{p}_3 = \begin{bmatrix} 0.933 \\ 0.616 \\ 0.866 \end{bmatrix}$$

Find θ_1 and θ_2.

6

Forward Kinematics

In this chapter, we will expand the discussion of kinematics from Chapter 2 to cover generalised serial chain manipulators. By using the Denavit-Harternberg notation, we will explore how it can be used to model the kinematics of general serial chain manipulators.

6.1 Joints

There are two commonly used joints in robotic manipulators: revolute (R) joints and prismatic (P) joints, as shown in Figure 6.1. A revolute joint can be simplified into one axis of rotation, which completely defines the relative motion of these two rigid bodies. This axis of rotation is a spatial line with its position and direction. A prismatic joint defines the sliding direction of one body relative to another. A prismatic joint can be replaced by another parallel prismatic joint at a different position while the relative motion of the two rigid bodies remains the same. Therefore, a prismatic joint can be characterised by a directional vector without a specific position.

In the eyes of a kinematician, the shape of a rigid body is of no importance.[1] The only thing that matters is the types and locations of the joints among the bodies. There are typically six types of joints, cylindrical, planar, screw, spherical, and the above two. Our discussion will focus on the serial robotic manipulators with any combination of R and P joints. A serial robotic manipulator has an open-loop structure, as shown in Figure 6.2, where a number (n) of rigid bodies are linked serially together by a number (n−1) of one degree-of-freedom (DoF) joints. Since each joint removes five-DoF from this robot and one body (except the ground) has six-DoF, the total DoF of the robot is $6(n-1)-5(n-1) = n-1$, which is simply the number of one-DoF joints the robot has.

In order to precisely describe the position and orientation of each moving link of a robotic manipulator, we desire to assign one frame to each link. Each link's position and orientation can be fully described by the transformation matrix between the ground frame and the attached frame to this link. Further, the transformation matrix between two adjacent links is only affected by the joint angle of the R joint between these two links. A technique for assigning frames effectively to a robotic manipulator is described in the following section.

6.2 Denavit-Hartenberg Notation

The Denavit-Hartenberg (DH) notation is a four-parameter representation between the relative position and orientation of one link with respect to its adjacent link. Here, we first

[1]On the other hand, it is important to a robot designer.

DOI: 10.1201/9781003614319-6

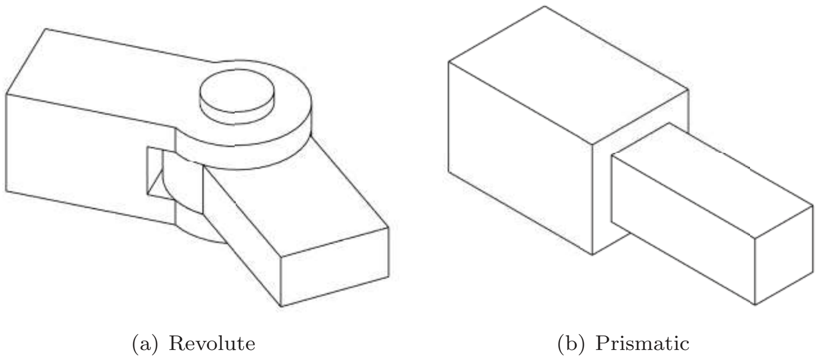


FIGURE 6.1
Two types of common robotic joints.

assume a robot with R joints only. Each R joint can be represented by a single spatial line. One important geometric property of spatial lines is applied here to formulate the relations among these lines: any two general spatial lines have one unique common perpendicular, defined as a line intersecting and being perpendicular to both given lines. There are some special cases with no unique solution, which will be discussed later. The DH notation can be best described as a procedure.

As shown in Figure 6.3, two links on a serial robotic manipulator with three revolute joints are depicted. The first step is to assign a z-axis along each joint axis. According to the above geometric property, there is always a unique common perpendicular between two adjacent z-axes. The x-axis is defined along this common perpendicular, directed from the current z-axis to the next z-axis, i.e., x_i points from z_i to z_{i+1}. Once x_i and z_i are determined, the frame $\{i\}$ is fully defined because $y_i = z_i \times x_i$. Note that $\{i\}$ is attached to link i between Joint i and Joint $i+1$, i.e., the coordinates of any point on link i measured in $\{i\}$ are invariant. This frame is called the DH frame on link i.

Since one DH frame is determined by adjacent joint axes, the ground frame $\{0\}$ and the end-effector frame $\{n\}$ cannot be fully determined. We will introduce additional rules after explaining the DH parameters. Given two DH frames, $\{i-1\}$ and $\{i\}$, the four DH parameters are defined according to Figure 6.3.

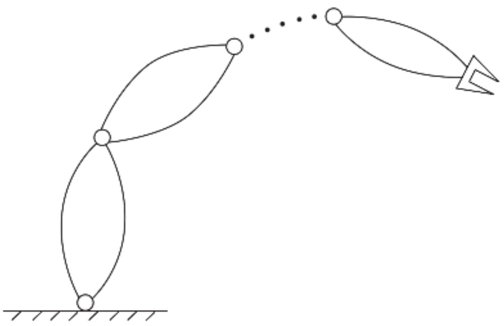


FIGURE 6.2
A schematic of a serial robotic manipulator.

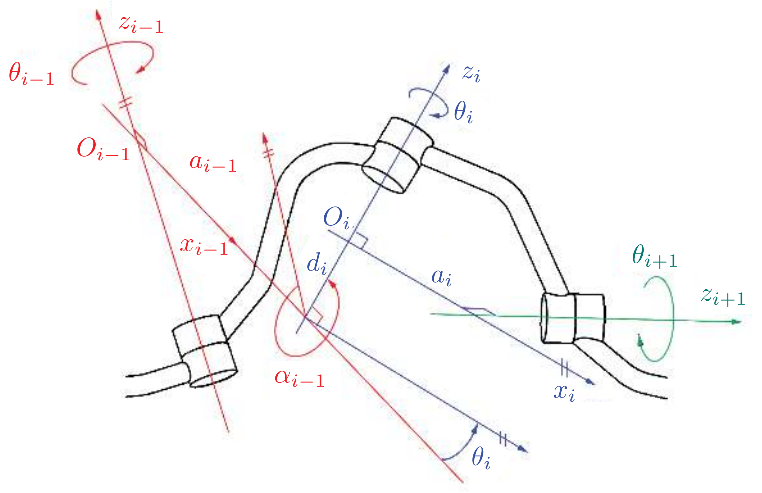


FIGURE 6.3
DH notation.

- a_{i-1} is the *link length* from z_{i-1} to z_i along the positive direction of x_{i-1}. a_{i-1} is a nonnegative invariant.
- α_{i-1} is the *link twist* from z_{i-1} to z_i along the positive direction of x_{i-1} (use right-hand rule). α_{i-1} is an invariant between 0 and 2π[2].
- d_i is the *link offset*, measured from x_{i-1} to x_i along the positive direction of z_i. d_i is an invariant that can be positive, zero, or negative depending on the positions of the intersection between z_i and x_{i-1} and the intersection between z_i and x_i.
- θ_i is the *joint angle* from x_{i-1} to x_i along the positive direction of z_i (use right-hand rule). θ_i is the only variable among the four DH parameters. Its range is typically between 0 and 2π.

The following rules are suggested to achieve the uniqueness and simplicity of the DH frames in special cases.

1. If z_i and z_{i+1} are parallel to each other, x_i is chosen to intersect with x_{i-1}.
2. If z_i and z_{i+1} intersect with each other, choose the direction of x_i such that α_i is between 0 and π.
3. If z_i and z_{i+1} are aligned, locate and direct x_i such that the values of the corresponding DH parameters are the simplest.
4. Keep the z-axis of the ground frame $\{0\}$ align with the first joint z-axis, i.e., $z_0 = z_1$.
5. Define x_n in the end-effector frame $\{n\}$ such that it points at the point of interest on the end-effector.

[2]You may see negative twist angle somewhere. That does not mean it is wrong. Rather, it is due to the different definitions of the range of the link twist.

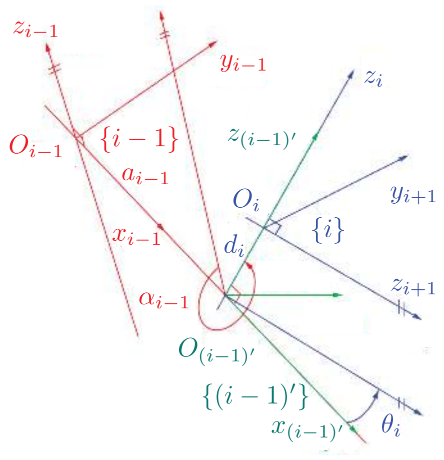


FIGURE 6.4
Transformation mapping T_{i-1} between $\{i-1\}$ and $\{i\}$.

The transformation matrix between $\{i-1\}$ and $\{i\}$ can be derived by introducing an auxiliary frame $\{(i-1)'\}$ consisting of x_{i-1} and z_i, as shown in Figure 6.4. The transform from $\{i-1\}$ to $\{(i-1)'\}$ is the combination of a translation along x_{i-1} and a rotation around x_{i-1}, which can be written as

$$ {}^{i-1}\mathbf{T}_{(i-1)'} = \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & \cos\alpha_{i-1} & -\sin\alpha_{i-1} & 0 \\ 0 & \sin\alpha_{i-1} & \cos\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1} $$

Similarly, the transform from $\{i-1\}'$ to $\{i\}$ is the combination of a translation along z_i and a rotation around z_i, i.e.,

$$ {}^{(i-1)'}\mathbf{T}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.2} $$

Combining (6.1) and (6.2) yields the transformation matrix between $\{i\}$ and $\{i-1\}$,

$$ \begin{aligned} {}^{i-1}\mathbf{T}_i &= {}^{i-1}\mathbf{T}_{(i-1)'}\,{}^{(i-1)'}\mathbf{T}_i \\ &= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{6.3} $$

The transformation between a robot's base and its end-effector can be expressed as:

$$ {}^0\mathbf{T}_n = {}^0\mathbf{T}_1\,{}^1\mathbf{T}_2 \ldots {}^{n-1}\mathbf{T}_n \tag{6.4} $$

where n is the DoF of the robotic manipulator.

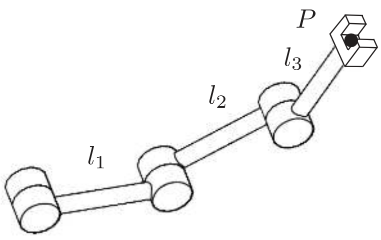


FIGURE 6.5
A planar RRR robot.

Example 6.1 (Transformations — frames): The following frames are defined on a robotic system: the base frame $\{B\}$, the workstation frame $\{S\}$, the wrist frame $\{W\}$, the tool frame $\{T\}$, and the goal frame $\{G\}$. Given ${}^B\mathbf{T}_W(\theta_1, \dots, \theta_n)$, ${}^W\mathbf{T}_T$, ${}^B\mathbf{T}_S$, and ${}^S\mathbf{T}_G$, find the position and orientation of the tool measured in the goal frame.

Solution:

$$\begin{aligned} {}^G\mathbf{T}_T &= {}^G\mathbf{T}_S\ {}^S\mathbf{T}_B\ {}^B\mathbf{T}_W\ {}^W\mathbf{T}_T \\ &= {}^S\mathbf{T}_G^{-1}\ {}^S\mathbf{T}_G^{-1}\ {}^B\mathbf{T}_S^{-1}\ {}^B\mathbf{T}_W\ {}^W\mathbf{T}_T \end{aligned} \tag{6.5}$$

Any point on the end-effector can be located in the fixed frame by

$$ {}^0\mathbf{p} = {}^0\mathbf{T}_n\ {}^n\mathbf{p} \tag{6.6}$$

where ${}^n\mathbf{p}$ and ${}^0\mathbf{p}$ are the coordinates of a point on the end-effector, measured in $\{n\}$ and $\{0\}$.

Example 6.2 (Forward kinematics — 3R): A planar robot is shown in Figure 6.5. Find the coordinates of Point P in the fixed frame, assuming: $l_1 = 0.25$ m, $l_2 = 0.30$ m, $l_3 = 0.20$ m, $\theta_{1e} = 15°$, $\theta_{2e} = 25°$, and $\theta_{3e} = 30°$.

Solution: The first step is to assign z_1, z_2, and z_3 along the axes of three R joints, respectively, as shown in Figure 6.6. For each axis, there are two possible directions to choose. Since all joints are parallel, we keep the directions of all z-axes the same. Note that each z-axis is floating along the axis of rotation without a specific position yet. Following the definition of the x-axis, there are infinitely many common perpendiculars for x_1, since z_1 and z_2 are parallel. Here, x_1 is located at an arbitrary point along z_1, which is the origin of $\{1\}$. Similarly, x_2 between z_2 and z_3 is chosen to sit at the intersection between x_1 and z_2, which is the origin of $\{2\}$. x_3 is an arbitrary vector perpendicular to z_3. For simplicity, x_3 is chosen at the intersection between x_2 and z_3, which is the origin of $\{3\}$. Finally, $\{0\}$ is determined by z_0 aligned with z_1 and x_0 perpendicular to z_0 at the origin of $\{1\}$, which is also the origin of $\{0\}$.

According to Figure 6.6, the DH parameters are derived in Table 6.1. Suppose incremental encoders are attached to each joint, so joint positions are monitored. If the home position is the fully stretched-out position along x_0, we have the joint angles at this home position as

$$\theta_{10} = 0°, \quad \theta_{20} = 0°, \quad \theta_{30} = 0° \tag{6.7}$$

The θ angles in Table 6.1 can be written as

$$\theta_1 = \theta_{1e} + \theta_{10} = \theta_{1e} + 0° \tag{6.8}$$

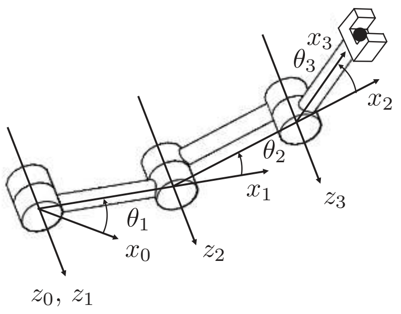


FIGURE 6.6
DH frames assigned to the RRR robot.

$$\theta_2 = \theta_{2e} + \theta_{20} = \theta_{2e} + 0^\circ \tag{6.9}$$

$$\theta_3 = \theta_{3e} + \theta_{30} = \theta_{3e} + 0^\circ \tag{6.10}$$

If the home positions are along x_0, 90°, and 90°, respectively, the θ angles in Table 6.1 can be written as

$$\theta_1 = \theta_{1e} + \theta_{10} = \theta_{1e} + 0^\circ \tag{6.11}$$

$$\theta_2 = \theta_{2e} + \theta_{20} = \theta_{2e} + 90^\circ \tag{6.12}$$

$$\theta_3 = \theta_{3e} + \theta_{30} = \theta_{3e} + 90^\circ \tag{6.13}$$

Substituting the values in Table 6.1 into Equation (6.3) yields

$$^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.97 & -0.26 & 0 & 0 \\ 0.26 & 0.97 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.14}$$

$$^1\mathbf{T}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_3 \\ \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.91 & -0.42 & 0 & 0.25 \\ 0.42 & 0.81 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.15}$$

$$^2\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & l_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.87 & -0.50 & 0 & 0.30 \\ 0.50 & 0.87 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.16}$$

The total transformation matrix is

$$^0\mathbf{T}_3 = {}^0\mathbf{T}_1\, {}^1\mathbf{T}_2\, {}^2\mathbf{T}_3 = \begin{bmatrix} 0.34 & -0.94 & 0 & 0.47 \\ 0.94 & 0.34 & 0 & 0.26 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.17}$$

TABLE 6.1
DH parameters of the RRR robot

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	l_3	0	θ_2
3	0	l_2	0	θ_3

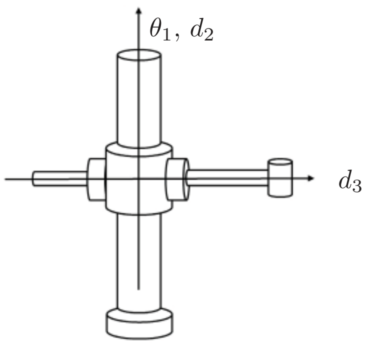


FIGURE 6.7
A cylindrical robot with z-axes defined.

For any given point ${}^3\mathbf{p}$ on the end-effector, its coordinates $\{0\}$ are given by

$$ {}^0\mathbf{p} = {}^0\mathbf{T}_3 \, {}^3\mathbf{p} \tag{6.18}$$

where ${}^0\mathbf{p}$ and ${}^3\mathbf{p}$ are the homogeneous coordinates. With ${}^3\mathbf{p} = \begin{bmatrix} 0.20 & 0 & 0 & 1 \end{bmatrix}^T$ (homogeneous form), the corresponding numerical solution is

$$ {}^0\mathbf{p} = \begin{bmatrix} 0.54 \\ 0.45 \\ 0 \\ 1 \end{bmatrix} \tag{6.19}$$

6.3 DH Parameters for Prismatic Joints

The previous definitions of the DH parameters work well for revolute joints. In the case of a prismatic joint, the z-axis is defined in the direction of sliding, which does not have a fixed position. To fully determine the z-axis, we can choose the location of the z-axis such that it passes through the origin of the previous or next frame, O_{i_1} or O_i, for simplicity. The x-axis is determined the same as before. The transformation matrix (6.3) remains the same.

Example 6.3 (DH parameters — prismatic joints): Assign the DH parameters for the cylindrical robot shown in Figure 6.7.

Solution: The actuated variables for this robot are revolute θ_1, and prismatic d_2 and d_3, along z_1, z_2, and z_3 respectively. According to frame assignment rules, the base frame $\{0\}$, first frame $\{1\}$, and second frame $\{2\}$ should all be coincident where the common perpendicular between z_2 and z_3 intersect (the dotted line). Therefore, x_0, x_1, and x_2 should be pointing along this common perpendicular. The origin of the final frame $\{3\}$, for simplicity, lies at the intersection of x_2 and z_3. This means x_3 also points along x_2. With these frames and axes defined, the DH parameters for the cylindrical robot in Figure 6.7 are

where a_3 is the length of the common perpendicular between z_2 and z_3.

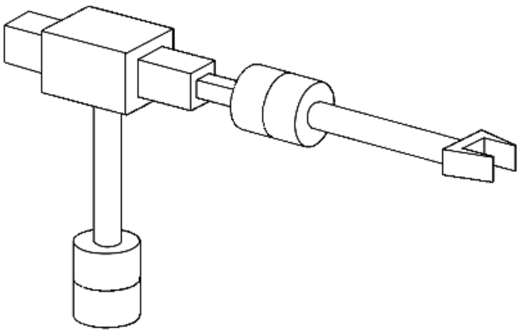


FIGURE 6.8
An RPR robot.

Example 6.4 (Forward kinematics — RPR): An RPR robot is shown in Figure 6.8. Find the transformation matrix between the base and the end-effector, assuming $\theta_1 = 35°$, $d_2 = 0.50$ m, $\theta_3 = 30°$, $l_1 = l_2 = 0.30$ m.

Solution: The DH frames are assigned to the RPR robotic manipulator as shown in Figure 6.9. The DH parameters are derived in Table 6.2 where the home positions are 90°, d_2, and 180°, respectively.

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	0	d_2	0
3	90°	a_3	d_3	0

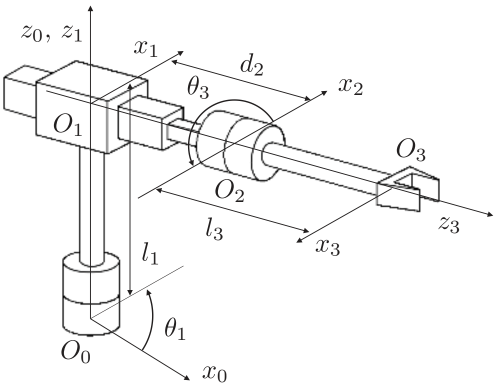


FIGURE 6.9
DH frames assigned to the RPR robot.

TABLE 6.2
DH parameters of the RPR robot

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	l_3	θ_1
2	90°	0	d_2	0
3	0	0	l_2	θ_3

All transform matrices are given by

$$^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.82 & -0.57 & 0 & 0 \\ 0.57 & 0.82 & 0 & 0 \\ 0 & 0 & 1 & 0.30 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.20}$$

$$^1\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -d_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -0.50 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.21}$$

$$^2\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & l_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.87 & -0.50 & 0 & 0 \\ -0.50 & 0.87 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.22}$$

The total transformation matrix is

$$^0\mathbf{T}_3 = {}^0\mathbf{T}_1\,{}^1\mathbf{T}_2\,{}^2\mathbf{T}_3 = \begin{bmatrix} c_1c_3 & -c_1s_3 & s_1 & s_1\,(l_2+d_2) \\ c_3s_1 & -s_1s_3 & -c_1 & -c_1\,(l_2+d_2) \\ s_3 & c_3 & 0 & l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.71 & -0.41 & 0.57 & 0.46 \\ 0.50 & -0.29 & -0.82 & -0.66 \\ 0.50 & 0.87 & 0 & 0.30 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.23}$$

Example 6.5 (Forward kinematics — spherical wrist): A spherical wrist of the PUMA robot is shown in Figure 6.10, where the axes of three joints are concurrent. Further, these joint axes are mutually perpendicular at the current configuration (home position). Find the total transformation matrix between the end-effector and the base at the manipulator's home position where the joints are at 30°, 90°, and 90°, respectively.

Solution: This is called the spherical wrist because the axes of all R joints interest at a single point. Since the axis of rotation is invariant for a given rotation, none of the three rotations due to the R joints can vary the position of this point, i.e., all three axes are geometrically constrained to interest at this point at any instant. Therefore, the end-effector is restricted to a three-DoF spherical motion centred at this point, which is called the spherical centre of this wrist.

Axes z_1, z_2, and z_3 are first assigned along the axes of the three R joints, as shown in Figure 6.10. Since the axes of the joints are mutually perpendicular, x_1 is aligned with z_3 to have the link twist between 0 and π, according to the definition of the x-axis. Similarly, x_2 is aligned with z_1. Axis x_3 can be chosen freely as long as it is perpendicular to z_3. Here, x_3 is chosen to be aligned with z_2. Finally, z_0 is aligned with z_1 while x_0 is at the spherical centre and has an angle of 45° from x_1 as shown in Figure 6.10. One important feature is that all assigned frames are at the same position, the spherical centre, which indicates that this robotic wrist can only perform rotation around this centre.

According to the DH frames assigned in Figure 6.10, the DH parameters are derived in Table 6.3, where the home positions are 30°, 90°, and 90°, respectively. All transform matrices are given by

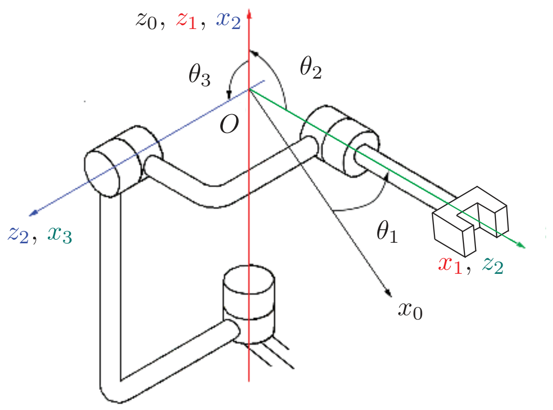


FIGURE 6.10
A spherical wrist with DH frames assigned.

TABLE 6.3
DH parameters of the spherical wrist

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	90°	0	0	θ_2
3	90°	0	0	θ_3

$$^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.87 & -0.50 & 0 & 0 \\ -0.50 & -0.86 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.24}$$

$$^1\mathbf{T}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.25}$$

$$^2\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.26}$$

The total transformation matrix is then given by

$$^0\mathbf{T}_3 = {^0\mathbf{T}_1}\,{^1\mathbf{T}_2}\,{^2\mathbf{T}_3} = \begin{bmatrix} s_1s_3 + c_1c_2c_3 & c_3s_1 - c_1c_3s_3 & c_1s_2 & 0 \\ c_2c_3s_1 - c_1s_3 & -c_1c_3 - c_2s_1s_3 & s_1s_2 & 0 \\ c_3s_2 & -s_2s_3 & -c_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.50 & 0 & 0.87 & 0 \\ -0.87 & 0 & 0.50 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.27}$$

Example 6.6 (Forward kinematics — PUMA): The PUMA robot represents a significant milestone in the history of industrial robots. The major advantage is that its position and orientation are decoupled in inverse kinematics, making analytical solutions possible. As shown in Figure 6.11, the PUMA robot consists of an RRR robot (the first three joint axes) and a robotic spherical wrist (the last three joint axes intersecting at a single point P). The offsets, d_3 and d_4, in the

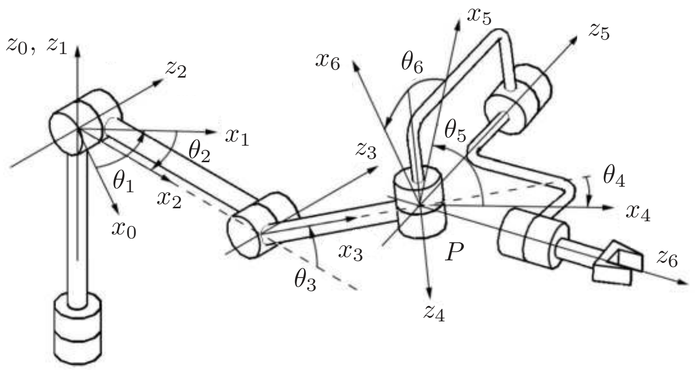


FIGURE 6.11
PUMA robot with assigned axes.

actual PUMA robot, are assumed to be zero. Find DH parameters and compute all the individual transformation matrices.

Solution:
The DH parameters are shown in Table 6.4. Therefore, the transformation between the frame attached to the end-effector and the ground frame can be expressed as

$$^0\mathbf{T}_6 = {}^0\mathbf{T}_1(\theta_1)\ {}^1\mathbf{T}_2(\theta_2)\ {}^2\mathbf{T}_3(\theta_3)\ {}^3\mathbf{T}_4(\theta_4)\ {}^4\mathbf{T}_5(\theta_5)\ {}^5\mathbf{T}_6(\theta_6) \tag{6.28}$$

where

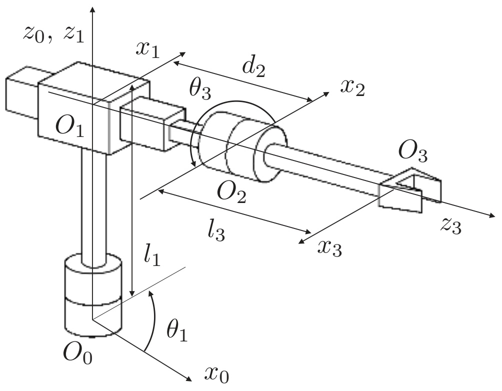
$$^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^1\mathbf{T}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_2 & -\cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^2\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & a_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^3\mathbf{T}_4 = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & a_3 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_3 & -\cos\theta_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^4\mathbf{T}_5 = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^5\mathbf{T}_6 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_6 & -\cos\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

TABLE 6.4
DH parameters of the PUMA robot

i	α_{i-1} (°)	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	-90	0	0	θ_2
3	0	a_2	0	θ_3
4	-90	a_3	0	θ_4
5	90	0	0	θ_5
6	-90	0	0	θ_6



i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	l_3	θ_1
2	90°	0	d_2	0
3	0	0	l_2	θ_3

FIGURE 6.12
RPR manipulator with DH table.

6.4 MATLAB® Examples

In this chapter, we investigated how to convert a DH table to transformation matrices that map the pose of each frame along a manipulator. The nature of DH parameters is that the actuated variables should remain as variables (say, θ) within the table, such that the transformation matrices ${}^{i-1}\mathbf{T}_i$ become functions of these variables, ${}^{i-1}\mathbf{T}_i(\theta)$. To facilitate unknown variables in MATLAB when solving forward kinematics problems, we should define *symbolic variables*[3] via MATLAB's Symbolic Math Toolbox. The following example shows how MATLAB can be used to obtain symbolic forward kinematics equations for serial manipulators.

Example M6.1 (Forward kinematics — RPR): Find the transformation matrix of the end-effector ${}^0\mathbf{T}_3$ for the RPR robot shown in Figure 6.12.

Solution: To do this, we need to convert each row of DH parameters to their respective transformation matrices, according to Equation (6.3). To code this more efficiently, we can write a MATLAB function called `dh2T()` (Inline 6.1) that converts a single row of DH parameters to a transformation matrix, then calls it for each row in the table.

```
function T_i_j = dh2T(dh_row)
%DH2T Converts row of DH parameters into a transformation matrix

al = dh_row(1);    % alpha rotation
a = dh_row(2);     % a displacement
d = dh_row(3);     % d displacement
th = dh_row(4);    % theta displacement

T_i_j = [
    cos(th)             -sin(th)            0            a;
    sin(th)*cos(al)     cos(th)*cos(al)     -sin(al)     -sin(al)*d;
    sin(th)*sin(al)     cos(th)*sin(al)     cos(al)      cos(al)*d;
    0                   0                   0            1];

end
```

From here, we can use the following script (Inline 6.2) to solve this problem. Note that in line 3, we end the `syms` definition with the keyword `real`, which tells MATLAB these variables can only

[3] https://au.mathworks.com/help/symbolic/create-symbolic-numbers-variables-and-expressions.html

contain real numbers. If any of these variables involve a transpose operation, the conjugate is not required, simplifying the resultant symbolic expressions.

```
% Define symbolic variables
% Add keyword "real" to assume these variables always contain real numbers
syms L1 L2 th1 d2 th3 real

% Define DH table (alpha, a, d, theta)
DH_Table = [
    0       0       L1      th1;
    pi/2    0       d2      0;
    0       0       L2      th3]

T_0_1 = dh2T(DH_Table(1,:))   % Get t-matrix from row 1 DH
T_1_2 = dh2T(DH_Table(2,:))   % Get t-matrix from row 2 DH
T_2_3 = dh2T(DH_Table(3,:))   % Get t-matrix from row 3 DH

T_0_3 = T_0_1 * T_1_2 * T_2_3   % EE t-matrix measured in base frame
```

The command window outputs the answers for ${}^0\mathbf{T}_1$, ${}^1\mathbf{T}_2$, ${}^2\mathbf{T}_3$, and ${}^0\mathbf{T}_3$ from the script.

```
DH_Table =

[    0, 0, L1, th1]
[ pi/2, 0, d2,   0]
[    0, 0, L2, th3]


T_0_1 =

[ cos(th1), -sin(th1), 0,  0]
[ sin(th1),  cos(th1), 0,  0]
[        0,         0, 1, L1]
[        0,         0, 0,  1]


T_1_2 =

[ 1, 0,  0,   0]
[ 0, 0, -1, -d2]
[ 0, 1,  0,   0]
[ 0, 0,  0,   1]


T_2_3 =

[ cos(th3), -sin(th3), 0,  0]
[ sin(th3),  cos(th3), 0,  0]
[        0,         0, 1, L2]
[        0,         0, 0,  1]


T_0_3 =

[ cos(th1)*cos(th3), -cos(th1)*sin(th3),  sin(th1),   L2*sin(th1) + d2*sin(th1)]
[ cos(th3)*sin(th1), -sin(th1)*sin(th3), -cos(th1), - L2*cos(th1) - d2*cos(th1)]
[          sin(th3),           cos(th3),         0,                          L1]
[                 0,                  0,         0,                           1]
```

Note that the expression for ${}^0\mathbf{T}_3$ is not fully simplified. Since ${}^0\mathbf{T}_3$ represents the final answer of this example, we wish to express it in the most simplified form. We can condense its length by using the MATLAB function `simplify()`. This will attempt to group and factorise symbolic variables as well as utilise simple trigonometric identities to simplify the expression. We recommend the use of `simplify()` on final answers only, and not on intermediate variables such as `T_0_1` etc.

```
>> simplify(T_0_3)

ans =

[ cos(th1)*cos(th3), -cos(th1)*sin(th3),  sin(th1),  sin(th1)*(L2 + d2)]
[ cos(th3)*sin(th1), -sin(th1)*sin(th3), -cos(th1), -cos(th1)*(L2 + d2)]
[          sin(th3),           cos(th3),         0,                  L1]
[                 0,                  0,         0,                   1]
```

Reading T_0_3 in Inline 6.4, the transformation matrix of the end-effector, relative to its base, is

$$
{}^0\mathbf{T}_3 = \begin{bmatrix} \cos\theta_1\cos\theta_3 & -\cos\theta_1\sin\theta_3 & \sin\theta_1 & \sin\theta_1(l_2+d_2) \\ \sin\theta_1\cos\theta_3 & -\sin\theta_1\sin\theta_3 & -\cos\theta_1 & -\cos\theta_1(l_2+d_2) \\ \sin\theta_3 & \cos\theta_3 & 0 & l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Example M6.2 (Forward kinematics — cylindrical robot): In Example 6.3, we showed that the DH parameters for a cylindrical robot shown in Figure 6.7 were

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	0	d_2	0
3	90°	a_3	d_3	0

Use MATLAB to find the transformation matrix of the end-effector relative to the base frame when the joint values are $\theta_1 = 90°$, $d_2 = 30$ cm, $d_3 = 10$ cm, and offset $a_3 = 5$ cm.

Solution: We need to find ${}^0\mathbf{T}_3$ before substituting known values into this transformation matrix. This can be easily done in MATLAB by finding the symbolic solution for ${}^0\mathbf{T}_3$ before converting it into a function handle to apply the known values.

```matlab
% Define symbolic variables
% Add keyword "real" to assume these variables always contain real numbers
syms th1 d2 d3 a3 real

% Define DH table for cylindrical robot (alpha, a, d, theta)
DH_Table = [
    0       0       0       th1;
    0       0       d2      0;
    pi/2    a3      d3      0]

T_0_1 = dh2T(DH_Table(1,:))    % Get t-matrix from row 1 DH
T_1_2 = dh2T(DH_Table(2,:))    % Get t-matrix from row 2 DH
T_2_3 = dh2T(DH_Table(3,:))    % Get t-matrix from row 3 DH

T_0_3 = T_0_1 * T_1_2 * T_2_3   % EE t-matrix measured in base frame

% Define function handle for T_0_3 so we can sub in values
% Note 'Vars' input, where we can define variables in a specific order
T_0_3_f = matlabFunction(T_0_3, 'Vars', {th1 d2 d3 a3});

% Find T_0_3 with values subbed
T_0_3_v = T_0_3_f(pi/2, 30e-2, 10e-2, 5e-2)
```

```
DH_Table =

[    0,   0,   0, th1]
[    0,   0, d2,    0]
[ pi/2, a3, d3,    0]


T_0_1 =

[ cos(th1), -sin(th1), 0, 0]
[ sin(th1),  cos(th1), 0, 0]
[        0,         0, 1, 0]
[        0,         0, 0, 1]


T_1_2 =

[ 1, 0, 0,  0]
[ 0, 1, 0,  0]
[ 0, 0, 1, d2]
[ 0, 0, 0,  1]
```

```
T_2_3 =

[ 1, 0,  0,  a3]
[ 0, 0, -1, -d3]
[ 0, 1,  0,   0]
[ 0, 0,  0,   1]


T_0_3 =

[ cos(th1), 0,  sin(th1), a3*cos(th1) + d3*sin(th1)]
[ sin(th1), 0, -cos(th1), a3*sin(th1) - d3*cos(th1)]
[        0, 1,         0,                        d2]
[        0, 0,         0,                         1]


T_0_3_v =

    0.0000         0    1.0000    0.1000
    1.0000         0   -0.0000    0.0500
         0    1.0000         0    0.3000
         0         0         0    1.0000
```

Therefore, the transformation matrix of the end-effector relative to the base frame is

$$
{}^{0}\mathbf{T}_3 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & a_3\cos\theta_1 + d_3\sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 & a_3\sin\theta_1 - d_3\cos\theta_1 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0.1 \\ 1 & 0 & 0 & 0.05 \\ 0 & 1 & 0 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.29}
$$

6.4.1 DH Parameter Summary

Assigning DH parameters follows a strict set of rules to reduce the chance of ambiguity in the frame assignments. This process is summarised as follows, where i is the frame number, with $i = 0$ being the base frame. In assigning DH parameters, start from frame 1 ($i = 1$), not from the base frame. n is the number of links.

1. Draw a rough sketch of the manipulator under analysis.
2. Define the locations of all z axes from frames 1 to n. If the base frame $\{0\}$ is not defined, do *not* define it here.
3. Link all z axes as follows:
   (a) Identify and mark where any z_i and z_{i+1} axes intersect, or if they don't intersect,
   (b) Draw a common perpendicular line between the two z axes. Identify and mark where these perpendicular lines intersect on both z axes.
4. For each z_i axis from frame 1 to n, assign the origin of frame i to an identified point on the axis. That is either,
   (a) Where your current z_i axis intersects with the next z_{i+1} axis, or
   (b) Where your z_i axis meets the common perpendicular line to the next z_{i+1} axis.
5. Assign the x axis for each frame. The following rules apply:
   (a) At frame $\{i\}$, if the current z_i axis and next z_{i+1} intersect, the x_i axis must point *normal* to the plane containing the two z axes, or
   (b) If there is a common perpendicular line to z_{i+1}, the x_i axis is coincident with this line.

6. Assign the remaining y axis to each frame for completion.
7. Assign the base frame $\{0\}$ to be coincident with frame $\{1\}$ if the base frame has not been arbitrarily defined. This is done so that moving from frame $\{0\}$ to $\{1\}$ is purely rotational or translation with no other offsets.
8. Fill out the rows of the DH table as follows. Starting from frame $\{1\}$ ($i = 1$):
   - $a_{i-1} =$ the distance from z_{i-1} to z_i measured along x_{i-1}; *
   - $\alpha_{i-1} =$ the angle from z_{i-1} to z_i measured about x_{i-1}; *
   - $d_i =$ the distance from x_{i-1} to x_i measured along z_i; and
   - $\theta_i =$ the angle from x_{i-1} to x_i measured about z_i.

   (*) The index of the rule has been reduced by 1 to avoid confusion.
9. Use Equation (6.3) to convert DH parameters into a transformation matrix, starting with $i = 1$.

$$ {}^{i-1}\mathbf{T}_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.30} $$

The short-hand notation used is as follows:

$$ \sin\theta_i = s_i \tag{6.31} $$
$$ \cos\theta_i = c_i \tag{6.32} $$

The following trigonometry identities can be used to simplify answers:

$$ \cos(\theta_1 + \theta_2) = c_{12} = c_1 c_2 - s_1 s_2 \tag{6.33} $$
$$ \sin(\theta_1 + \theta_2) = s_{12} = c_1 s_2 + s_1 c_2 \tag{6.34} $$
$$ \cos(\theta_1 - \theta_2) = c_1 c_2 + s_2 s_2 \tag{6.35} $$
$$ \sin(\theta_1 - \theta_2) = s_1 c_2 - c_1 s_2 \tag{6.36} $$

6.5 Conclusion

In this chapter, we describe how the forward kinematics of a serial manipulator is solved. First, frames are systematically assigned to the system of rigid bodies that make up a serial manipulator. The position of these frames is described using Denavit-Hartenberg notation, which describes the kinematic configuration of a robot with the least parameters. This systematic process can be applied to robots with both revolute and prismatic actuators. From the table of DH parameters, we derived transformation matrices from one link to another, which fully describes the rigid body motion of a serial robot, thus solving the forward kinematics of the robot in analytical form.

The definition of a robot using DH parameters is not unique, where the direction of assigned axes affects the orientation of assigned frames. However, the final transformation matrix, which maps the pose of the robot's end-effector to a fixed universal frame, should remain consistent amongst all combinations of valid DH parameters for the same robot.

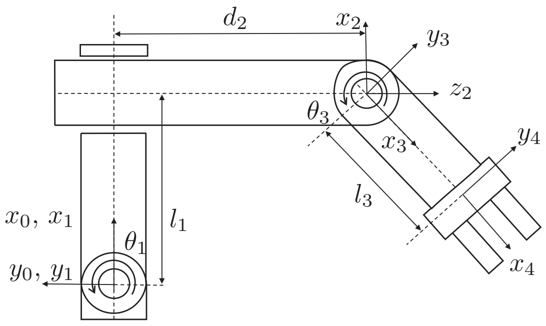


FIGURE 6.13
Planar robot for Problem 1.

6.6 Exercises

Problem 1. The robot is shown in Figure 6.13.

1. Identify the Denavit-Hartenberg parameters and tabulate the results, indicating the variables which are actuated (**Note**: You do not need to account for frame $\{4\}$ in this part).
2. Compute the individual transformation matrices which relate frame $\{i\}$ to frame $\{i-1\}$ according to the frames shown in the Figure 6.13. (i.e, ${}^0\mathbf{T}_1$, ${}^1\mathbf{T}_2$, ${}^2\mathbf{T}_3$)
3. Find the transformation matrix that relates frame $\{3\}$ to frame $\{0\}$ (i.e., ${}^0\mathbf{T}_3$) in its simplest form.
4. Find the vector expression that describes the position of the tool tip relative to $\{1\}$. Assume the tool tip position, expressed in $\{4\}$, is $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$.

Problem 2. Orthopaedic surgery is concerned with conditions involving the musculoskeletal system. Often, the mode of treatment involves the insertion of a prosthesis in place of diseased bone. Cavities need to be created within the bone segment of interest to house

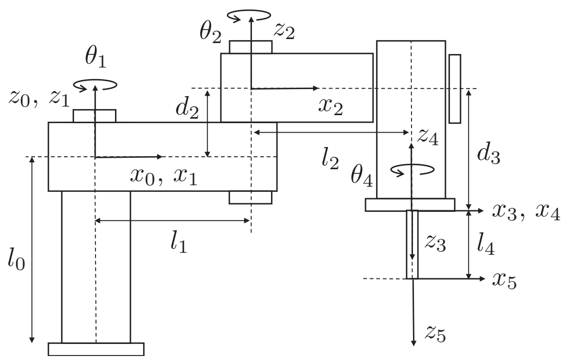


FIGURE 6.14
Orthopaedic robot for Problem 2.

this prosthesis. Traditionally, this was achieved by using jigs and mallets to hammer away at the bone. However, this technique is inherently inaccurate and does not cater for the variation in bone characteristics, such as size within a set of patients. This could lead to prosthesis misalignment, which affects the life span of the implant. This led to research and development into robotic systems for orthopaedic surgery . The philosophy is that a robot could mill a cavity with superior accuracy and repeatability and allow for variation in patient bone characteristics compared to manual techniques. In these systems, high-speed cutting tools are attached to the end-effector. The robot in Figure 6.13 is an example of a robot that has been used in hip and knee replacement surgery. For this robot:

1. Please identify the Denavit-Hartenberg parameters and tabulate the results, indicating the variables which are actuated (**Note**: You do not need to account for frame $\{5\}$ in this part).
2. Please obtain the individual transformation matrices which relate frame $\{i\}$ to frame $\{i-1\}$ according to the frames shown in the Figure 6.14. (i.e., ${}^{0}\mathbf{T}_1$, ${}^{1}\mathbf{T}_2$, ${}^{2}\mathbf{T}_3$, ${}^{3}\mathbf{T}_4$)
3. Find the transformation matrix that relates frame $\{4\}$ to frame $\{0\}$ (i.e., ${}^{0}\mathbf{T}_4$) in its simplest form.
4. Find the inverse of the matrix ${}^{0}\mathbf{T}_4$. What is the physical meaning of this transformation matrix?
5. Consider a drill bit being attached to the wrist. If the length of the drill bit is l_4, find a vector expression which describes the position of the drill bit relative to $\{0\}$ (i.e., $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$ where $\mathbf{p}$ is the origin of the frame $\{5\}$).

Problem 3. A serial manipulator with four revolute joints is shown in Figure 6.15. The initial frame attached to the base is given by $x_0 - y_0 - z_0$. The axes of all revolute joints are given by z_1, z_2, z_3, and z_4. At the instant configuration shown in Figure 6.15, these joint axes are either parallel or perpendicular. Furthermore, z_1, z_3, and z_4 are in the $y_0 - O - z_0$ plane, while z_2 indicated by $\odot$ is perpendicular to the $y_0 - O - z_0$ plane. The link frames have been assigned to all moving links of this manipulator (y axes are not displayed). IMPORTANT: $\odot$ indicates a vector pointing out from the sheet plane (upwards), while $\otimes$ indicates a vector pointing into the sheet plane (downwards).

1. Construct a table of the Denavit-Hartenberg parameters that describe the serial mechanism, based on the link frames shown in Figure 6.15. Note that the joint angles θ_1, θ_2, θ_3, and θ_4 are variables, measured around z_1, z_2, z_3, and z_4, respectively.
2. Based on the DH parameters defined, find the homogeneous transformation matrices between every two adjacent frames, i.e., ${}^{0}\mathbf{T}_1$, ${}^{1}\mathbf{T}_2$, ${}^{2}\mathbf{T}_3$, and ${}^{3}\mathbf{T}_4$.
3. Upon the results of the previous part, derive the analytic expression of the position of the origin of the 4th frame $\{4\}$ (attached to the end-effector), given in the initial frame $\{0\}$, by knowing $\theta_2 = 0°$ and $\theta_4 = 0°$.
4. If the homogeneous coordinates of the origin of $\{4\}$ in $\{0\}$ is given by ${}^{0}\mathbf{p}_{O4} = \begin{bmatrix} 10 & 10 & 6 & 1 \end{bmatrix}^T$, find the corresponding joint angles θ_1 and θ_3, by knowing $\theta_2 = 0°$ and $\theta_4 = 0°$.

Problem 4. A serial manipulator with three revolute joints is shown in Figure 6.16. The tool frame T is also given in Figure 6.16.

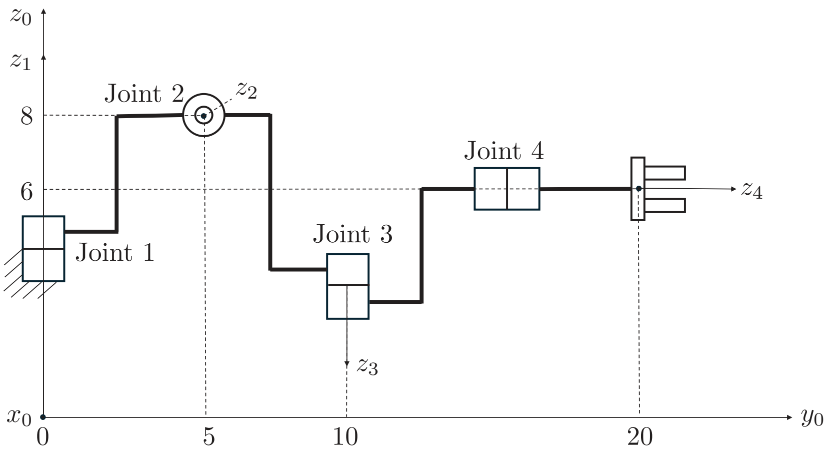


FIGURE 6.15
A four-link robot.

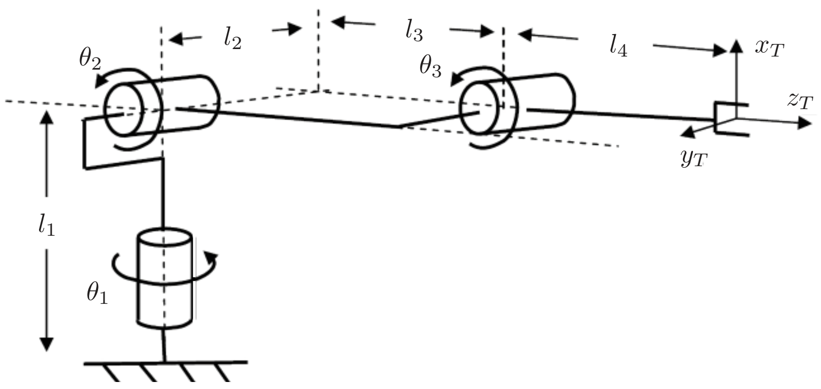


FIGURE 6.16
A serial spherical robotic manipulator.

1. Assign the link frames — i.e., attach the frames (you must assign the z and x axes of each frame) to the moving links of the mechanism.
2. Construct a table of the Denavit-Hartenberg parameters that describe the serial manipulator in terms of the link lengths as given in Figure 6.16.
3. Find the individual transformation matrices to describe the motion of the links due to the joint θ_1, θ_2, and θ_3. You do not need to multiply them together.

7

Inverse Kinematics

As discussed in Chapter 2, forward kinematics is the process of mapping the joint space to the end effector space or task space. The counter problem is known as the inverse kinematics, in which we map the end effector task space to the joint space. We know that the solution to the forward kinematics serves as the starting point of the inverse kinematics. However, this chapter will cover advanced inverse kinematics techniques applicable to general serial manipulators. Inverse kinematics for general serial chain manipulators can be quite challenging and requires a bit of forward-thinking and experience to be able to handle all types of transcendental equations that may be encountered in kinematic equations for robots with revolute joints.

The inverse kinematics problem of serial robotic manipulators was considered one of the most challenging problems in robotics research in the 1980s, where in some cases, analytical solutions are impossible. Although we do not discuss these types of manipulators in this chapter, the advancement of computational power has allowed the inverse kinematic solutions to be numerically calculated efficiently using gradient-descent methods.

7.1 Basic Techniques

In inverse kinematics, we frequently encounter some forms of trigonometric functions. Knowing the solutions to these functions is beneficial for solving the inverse kinematics of many types of robots.

Example 7.1 (Dual-parameter arc tangent: Atan2()**):** Given the expressions

$$\cos\theta = kA, \quad \sin\theta = kB \tag{7.1}$$

where k, A, B are three constraints, solve for θ.

Solution: Use Atan2 to find the solution:

$$\theta = \text{Atan2}(kB, kA) \tag{7.2}$$

If k is positive, (A, B) and (kA, kB) are in the same quadrant. Hence, the solution can be further simplified into

$$\theta = \text{Atan2}(B, A) \tag{7.3}$$

Example 7.2 (Solving transcendental equations): Given equation

$$A\cos\theta + B\sin\theta = C \tag{7.4}$$

DOI: 10.1201/9781003614319-7

where A, B, C are three constants, solve for θ.

Solution: There are different ways to solve (7.4). One method is to divide (7.4) by D, such that

$$\frac{A}{D}\cos\theta + \frac{B}{D}\sin\theta = \frac{C}{D} \tag{7.5}$$

where

$$D = \sqrt{A^2 + B^2} \tag{7.6}$$

Also, note that

$$\frac{A}{D} = \cos\phi \qquad \text{and} \qquad \frac{B}{D} = \sin\phi \tag{7.7}$$

Therefore, (7.5) becomes

$$\cos\phi\cos\theta + \sin\phi\sin\theta = \frac{C}{D} \tag{7.8}$$

which is simplified to

$$\cos(\theta - \phi) = \frac{C}{D} \tag{7.9}$$

There are two solutions for (7.9):

$$\theta = \pm \arccos\left(\frac{C}{D}\right) + \phi \tag{7.10}$$

where $\phi = \text{Atan2}(B, A)$.

Besides the above solutions to some common functions, there are also some tips for handling functions in inverse kinematics.

Pythagorean Identity

To remove some variables from equations, we can use

$$\sin(\alpha)^2 + \cos(\alpha)^2 = 1 \tag{7.11}$$

For example, to simplify the following equations

$$x = c_1 + c_{12}, \quad y = s_1 + s_{12} \tag{7.12}$$

they can be written as

$$c_{12} = x - c_1, \quad s_{12} = y - s_1 \tag{7.13}$$

We can then substitute c_{12} and s_{12} into (7.11) to get

$$(x - c_1)^2 + (y - s_1)^2 = 1 \tag{7.14}$$

The original two functions on θ_1 and θ_2 are simplified into the above single function on θ_1 only, which can be readily solved.

Equation Balancing

To reduce the entanglement of variables, keep the numbers of variables on both sides of equations as close as possible. This is typically done by inversing transformation matrices. Consider a general six-DoF robotic manipulator has the kinematics relation

$$ {}^{0}\mathbf{T}_6 = {}^{0}\mathbf{T}_1(\theta_1)\ {}^{1}\mathbf{T}_2(\theta_2)\ {}^{2}\mathbf{T}_3(\theta_3)\ {}^{3}\mathbf{T}_4(\theta_4)\ {}^{4}\mathbf{T}_5(\theta_5)\ {}^{5}\mathbf{T}_6(\theta_6) \tag{7.15}$$

where ${}^{0}\mathbf{T}_6$ is known in the inverse kinematics. To simplify the process of solving for θ_i with $i = 1 \dots 6$ through these matrices, a better way to arrange the equations to reduce the entanglement of the variables could be

$$ {}^{2}\mathbf{T}_3^{-1}(\theta_3)\ {}^{1}\mathbf{T}_2^{-1}(\theta_2)\ {}^{0}\mathbf{T}_1^{-1}(\theta_1)\ {}^{0}\mathbf{T}_6 = {}^{3}\mathbf{T}_4(\theta_4)\ {}^{4}\mathbf{T}_5(\theta_5)\ {}^{5}\mathbf{T}_6(\theta_6) \tag{7.16}$$

or

$$ {}^{3}\mathbf{T}_2(\theta_3)\ {}^{2}\mathbf{T}_1(\theta_2)\ {}^{1}\mathbf{T}_0(\theta_1)\ {}^{0}\mathbf{T}_6 = {}^{3}\mathbf{T}_4(\theta_4)\ {}^{4}\mathbf{T}_5(\theta_5)\ {}^{5}\mathbf{T}_6(\theta_6) \tag{7.17}$$

This is not the only way to balance the number of variables on both sides of this equation. Other alternatives can be any of the following

$$ {}^{2}\mathbf{T}_1(\theta_2)\ {}^{1}\mathbf{T}_0(\theta_1)\ {}^{0}\mathbf{T}_6\ {}^{6}\mathbf{T}_5(\theta_6) = {}^{2}\mathbf{T}_3(\theta_3)\ {}^{3}\mathbf{T}_4(\theta_4)\ {}^{4}\mathbf{T}_5(\theta_5) \tag{7.18}$$

$$ {}^{1}\mathbf{T}_0(\theta_1)\ {}^{0}\mathbf{T}_6\ {}^{6}\mathbf{T}_5(\theta_6)\ {}^{5}\mathbf{T}_4(\theta_5) = {}^{1}\mathbf{T}_2(\theta_2)\ {}^{2}\mathbf{T}_3(\theta_3)\ {}^{3}\mathbf{T}_4(\theta_4) \tag{7.19}$$

$$ {}^{0}\mathbf{T}_6\ {}^{6}\mathbf{T}_5(\theta_6)\ {}^{5}\mathbf{T}_4(\theta_5)\ {}^{4}\mathbf{T}_3(\theta_4) = {}^{0}\mathbf{T}_1(\theta_1)\ {}^{1}\mathbf{T}_2(\theta_2)\ {}^{2}\mathbf{T}_3(\theta_3) \tag{7.20}$$

Note that these are not randomly balanced equations but have particular meanings in terms of frame transformations.

7.2 Analytical Solution to Inverse Kinematics

Analytical solutions to inverse kinematics allow fast real-time updates of the joints solutions for any given position and orientation of the end-effector, by simply substituting the numerical position and orientation into the analytical solutions. Furthermore, analytical solutions differentiate the different sets of solutions belonging to different regions in the workspace. On the other hand, analytical solutions to nonlinear equations do not exist in general. In order to achieve analytical solutions, most industrial robots are designed in such a way that analytical solutions can be derived. PUMA robot is an outstanding example of this. In the absence of an analytical solution, we must rely on gradient-based solvers that do not guarantee solution convergence and can be prone to large errors in problematic configurations.

Example 7.3 (IK of an RR robot): An RR robot is shown in Figure 7.1, where both link lengths are 1 m. Find the analytical solutions of joint angles in terms of the position of the end-effector (x_p, y_p). Assume $l_1 = l_2 = 1$ m.

Solution: The forward kinematic solution of this robot is

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} c_1 + c_{12} \\ s_1 + s_{12} \end{bmatrix} \tag{7.21}$$

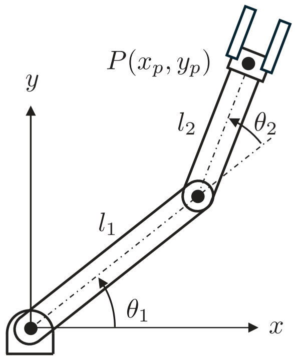


FIGURE 7.1
An RR robot.

We can use the Pythagorean identity (7.11) to reduce two equations into one equation by removing θ_2:

$$(x_p - c_1)^2 + (y_p - s_1)^2 = 1$$
$$2x_p c_1 + 2y_p s_1 = x_p^2 + y_p^2 \tag{7.22}$$

Equation (7.22) is now is the form of Example 7.2, where $A = 2x_p$, $B = 2y_p$, and $C = x_p^2 + y_p^2$. According to (7.6), $D = 2\sqrt{x_p^2 + y_p^2}$, which yields solutions

$$\theta_{1s} = \pm \arccos\left(\frac{C}{D}\right) + \phi \tag{7.23}$$

where

$$\frac{C}{D} = \frac{1}{2}\sqrt{x_p^2 + y_p^2} \qquad \text{and} \qquad \phi = \text{Atan2}(y_p, x_p) \tag{7.24}$$

Substituting the solutions (7.23) into (7.21) yields

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} c_{1s} + c_{12} \\ s_{1s} + s_{12} \end{bmatrix} \tag{7.25}$$

where $c_{1s} = \cos\theta_{1s}$ and $s_{1s} = \sin\theta_{1s}$. Here, θ_{1s} is used to represent the given solution of θ_1 to avoid writing down the full expression given in (7.23). If $\theta_{1s} + \theta_2$ is treated as a single variable, (7.23) is in the form of Example 7.1, where $k = 1$, $A = x_p - c_{1s}$, and $B = y_p - s_{1s}$. Hence, the solution of $\theta_{1s} + \theta_2$ can be readily found as

$$\theta_{1s} + \theta_2 = \text{Atan2}(y_p - s_{1s}, x_p - c_{1s}) \tag{7.26}$$

which gives the solution for θ_2

$$\theta_{2s} = \text{Atan2}(y_p - s_{1s}, x_p - c_{1s}) - \theta_{1s} \tag{7.27}$$

In summary, there are two sets of analytical solutions to this problem:

$$\theta_{1s} = \arccos\left(\frac{C}{D}\right) + \phi \qquad \theta_{2s} = \text{Atan2}(y_p - s_{1s}, x_p - c_{1s}) - \theta_{1s} \tag{7.28}$$

$$\theta_{1s} = -\arccos\left(\frac{C}{D}\right) + \phi \qquad \theta_{2s} = \text{Atan2}(y_p - s_{1s}, x_p - c_{1s}) - \theta_{1s} \tag{7.29}$$

with variables C, D, and ϕ defined in (7.24).

If we consider a particular position of the end-effector at $(1, 1)$, (7.21) yields the root of $\theta_1 = 0$, $\theta_2 = \pi/2$, and $\theta_1 = \pi/2$, $\theta_2 = 0$, as the numeric solutions to the inverse kinematics at this position.

TABLE 7.1
The DH parameters are defined below in Table 7.1

i	α_{i-1} ($^\circ$)	a_{i-1}	d_i	θ_i
1	0	1	0	θ_1
2	-45	0	0	θ_2
3	0	0	$\sqrt{2}$	θ_3
4	0	$\sqrt{2}$	0	θ_4

Example 7.4 (IK of a 4R robot): For a 4R robot, find the numerical solution of joint angles such that

$$ {}^0\mathbf{p}_4 = \begin{bmatrix} 0.7 & 1.2 & 1.5 \end{bmatrix}^T \tag{7.30}$$

And the position of the end-effectoris given by

$$\begin{aligned} {}^0\mathbf{p}_4 &= {}^0\mathbf{T}_4\,{}^4\mathbf{p}_4 \\ &= \begin{bmatrix} \frac{1}{2}\left(2c_1 + \left(\sqrt{2}-1\right)c_{123} + \left(1+\sqrt{2}\right)c_{123} + 2s_{12}\right) \\ c_1\left(\sqrt{2}c_3s_2 + c_2\left(-1+s_3\right)\right) + s_1\left(1+\sqrt{2}c_2c_3 + s_2 - s_2s_3\right) \\ 1+s_3 \end{bmatrix} \end{aligned} \tag{7.31}$$

Solution:

One approach is to equate the analytical solution and the given numerical one directly, which can be complicated. Let us try to balance the number of variables.

$$ {}^0\mathbf{T}_1^{-1}\,{}^0\mathbf{p}_4 = {}^1\mathbf{T}_2\,{}^2\mathbf{T}_3\,{}^3\mathbf{p}_4 \tag{7.32}$$

where

$$LHS = \begin{bmatrix} 0.7c_1 + 1.2s_1 \\ 1.2c_1 - 0.7s_1 \\ 1.5 \\ 1 \end{bmatrix} \quad \text{and} \quad RHS = \begin{bmatrix} 1+\sqrt{2}c_2c_3 + s_2 - s_2s_3 \\ \sqrt{2}s_2c_3 + c_2(-1+s_3) \\ 1+s_3 \\ 1 \end{bmatrix} \tag{7.33}$$

Joint angle θ_3 can firstly be solved by comparing Elements (3,1) on RHS and LHS.

$$\begin{aligned} s_3 &= 0.5 \\ \theta_3 &= \pm\arcsin(0.5) = 30^\circ \text{ or } 120^\circ \end{aligned} \tag{7.34}$$

When $\theta_3 = 30^\circ$, by summing the square of Elements (1,1) and (2,1) from RHS,

$$1.93 = 2.75 + 2.45c_2 + s_2 \tag{7.35}$$

Let

$$\cos\phi = \frac{2.45}{D} \text{ and } \sin\phi = \frac{1}{D} \tag{7.36}$$

where

$$D = \sqrt{2.45^2 + 1^2} \tag{7.37}$$

Therefore,

$$\begin{aligned} \theta_2 + \phi = \theta_2 - 22.2 &= \pm 108.1 \\ \theta_2 &= 130.3^\circ \text{ or } -85.85^\circ \end{aligned} \tag{7.38}$$

Substituting $\theta_3 = 30^\circ$ and $\theta_2 = 130.3^\circ$ into Elements (1,1) and (2,1) from RHS yields two simultaneous equations of c_1 and s_1. Solving the equations gives

$$\begin{aligned} c_1 &= 0.9955 \text{ or } s_1 = -0.09 \\ \theta_1 &= -5.149^\circ \end{aligned} \tag{7.39}$$

Substituting $\theta_3 = 30^\circ$ and $\theta_2 = -85.85^\circ$ into Elements (1,1) and (2,1) from RHS yields two simultaneous equations of c_1 and s_1. Solving the equations gives

$$\begin{aligned} c_1 &= -0.568 \text{ or } s_1 = 0.823 \\ \theta_1 &= 124.6^\circ \end{aligned} \tag{7.40}$$

When $\theta_3 = 120^\circ$, by summing the square of Elements (1,1) and (2,1) from RHS,

$$1.93 = 1.518 - \sqrt{2}c_2 + 0.268s_2 \tag{7.41}$$

Let

$$\cos\phi = \frac{-\sqrt{2}}{D} \text{ and } \sin\phi = \frac{0.268}{D} \tag{7.42}$$

where

$$D = \sqrt{(-\sqrt{2})^2 + 0.268^2} \tag{7.43}$$

Therefore,

$$\begin{aligned} \theta_2 + \phi = \theta_2 - 169.3 &= \pm 73.37 \\ \theta_2 &= 242.7^\circ \text{ or } 95.94^\circ \end{aligned} \tag{7.44}$$

Substituting $\theta_3 = 120^\circ$ and $\theta_2 = 242.7^\circ$ into Elements (1,1) and (2,1) from RHS yields two simultaneous equations of c_1 and s_1. Solving the equations gives

$$\begin{aligned} c_1 &= 0.866 \text{ or } s_1 = 0.499 \\ \theta_1 &= 29.96^\circ \end{aligned} \tag{7.45}$$

Substituting $\theta_3 = 120^\circ$ and $\theta_2 = 95.94^\circ$ into Elements (1,1) and (2,1) from RHS yields two simultaneous equations of c_1 and s_1. Solving the equations gives

$$\begin{aligned} c_1 &= 0.0089 \text{ or } s_1 = 1 \\ \theta_1 &= 89.49^\circ \end{aligned} \tag{7.46}$$

Example 7.5 (IK of PUMA): The PUMA robot shown in Figure 7.2(a) is a milestone in the history of industrial robots. The major advantage is that its position and orientation are decoupled in inverse kinematics, making analytical solutions possible. As shown in Figure 7.2(b), the PUMA robot consists of an RRR robot (the first three joint axes) and a robotic spherical wrist (the last three joint axes intersecting at a single point P). The DH table is shown in Table 7.2, where the offsets, d_3 and d_4, in the actual PUMA robot, are assumed to be zeros. The problem here is to find the solutions to the inverse kinematics of the PUMA robot shown in Figure 7.2, i.e., find the joint angles in terms of the position and orientation of the end-effector.

Solution: Since the three joints in the wrist only change the orientation of the end-effector and have no effect on the position of Point P, we can readily use P to represent the position of the end-effector. Therefore, the position of the end-effector can be first used to solve for the first three joint angles, while the orientation of the end-effector is then used to solve for the last three joint angles. This is the feature of the decoupling of the PUMA robot. The total solutions will be derived

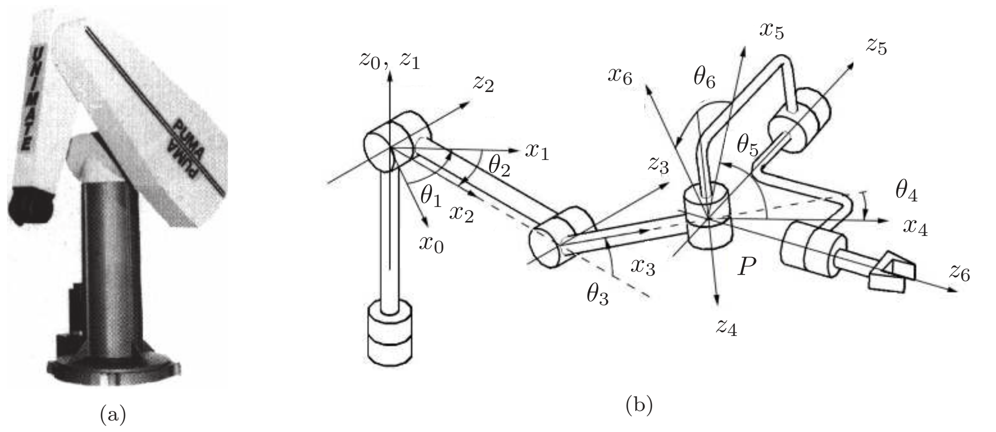


FIGURE 7.2
PUMA robot and its kinematic model.[1]

TABLE 7.2
DH parameters of the PUMA robot

i	α_{i-1} (°)	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	-90	0	0	θ_2
3	0	a_2	0	θ_3
4	-90	a_3	0	θ_4
5	90	0	0	θ_5
6	-90	0	0	θ_6

in two steps: 1) to solve the position of the end-effector for the first three joint angles and 2) to solve the orientation of the end-effector for the last three joint angles.

According to the DH parameters in Table 7.2, the transformation between the frame attached to the end-effector and the ground frame can be expressed as

$$^0\mathbf{T}_6 = {}^0\mathbf{T}_1(\theta_1)\ {}^1\mathbf{T}_2(\theta_2)\ {}^2\mathbf{T}_3(\theta_3)\ {}^3\mathbf{T}_4(\theta_4)\ {}^4\mathbf{T}_5(\theta_5)\ {}^5\mathbf{T}_6(\theta_6) \tag{7.47}$$

where

$$^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^1\mathbf{T}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_2 & -\cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^2\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & a_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^3\mathbf{T}_4 = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & a_3 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_3 & -\cos\theta_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^4\mathbf{T}_5 = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^5\mathbf{T}_6 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_6 & -\cos\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[1]Aly, M.F., Abbas, A.T. and Megahed, S.M., 2010. Robot workspace estimation and base placement optimisation techniques for the conversion of conventional work cells into autonomous flexible manufacturing systems. *International Journal of Computer Integrated Manufacturing*, 23(12), pp. 1133–1148.

1. Position:

Since Point P is the origin of $\{4\}$, $\{5\}$, and $\{6\}$, its position in $\{0\}$ is given by

$$^0\mathbf{p} = {}^0\mathbf{T}_1(\theta_1)\,{}^1\mathbf{T}_2(\theta_2)\,{}^2\mathbf{T}_3(\theta_3)\,{}^3\mathbf{T}_4(\theta_4)\,{}^4\mathbf{p} \tag{7.48}$$

where $^0\mathbf{p} = \begin{bmatrix} x_p & y_p & z_p & 1 \end{bmatrix}^T$ and $^4\mathbf{p} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$ while $^0\mathbf{p}$ is known in the problem of inverse kinematics. Note that $^3\mathbf{p} = {}^3\mathbf{T}_4(\theta_4)\,{}^4\mathbf{p} = \begin{bmatrix} a_3 & 0 & 0 & 1 \end{bmatrix}^T$, which is the origin of $\{4\}$ measured in $\{3\}$ and invariant. Therefore, (7.48) can be simplified as

$$^0\mathbf{p} = {}^0\mathbf{T}_1(\theta_1)\,{}^1\mathbf{T}_2(\theta_2)\,{}^2\mathbf{T}_3(\theta_3)\,{}^3\mathbf{p} \tag{7.49}$$

where only three variables present. Applying equation balancing to (7.49) yields

$$^0\mathbf{T}_1^{-1}(\theta_1)\,{}^0\mathbf{p} = {}^1\mathbf{T}_2(\theta_2)\,{}^2\mathbf{T}_3(\theta_3)\,{}^3\mathbf{p} \tag{7.50}$$

or

$$\begin{bmatrix} \cos\theta_1 x_p + \sin\theta_1 y_p \\ -\sin\theta_1 x_p + \cos\theta_1 y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_2\cos\theta_3 a_3 - \sin\theta_2\sin\theta_3 a_3 + a_2\cos\theta_2 \\ 0 \\ -\sin\theta_2\cos\theta_3 a_3 - \cos\theta_2\sin\theta_3 a_3 - a_2\sin\theta_2 \\ 1 \end{bmatrix} \tag{7.51}$$

Applying Example 7.2 to the second row of (7.51) yields

$$\theta_{1s} = \pm\pi/2 + \text{Atan2}(-x_p, y_p) \tag{7.52}$$

where θ_{1s} as the analytical solution of θ_1. Substituting this solution into the remaining two equations in (7.51) yields

$$\cos\theta_{1s} x_p + \sin\theta_{1s} y_p = \cos\theta_2\cos\theta_3 a_3 - \sin\theta_2\sin\theta_3 a_3 + a_2\cos\theta_2 \tag{7.53}$$

$$z_p = -\sin\theta_2\cos\theta_3 a_3 - \cos\theta_2\sin\theta_3 a_3 - a_2\sin\theta_2 \tag{7.54}$$

To simplify the expressions of the equations, we introduce a lumped constant, $k_1 = \cos\theta_{1s} x_p + \sin\theta_{1s} y_p$, such that (7.53) can be written as

$$k_1 - a_2\cos\theta_2 = \cos(\theta_2 + \theta_3)a_3 \tag{7.55}$$

$$-z_p - a_2\sin\theta_2 = \sin(\theta_2 + \theta_3)a_3 \tag{7.56}$$

According to the Pythagorean identity, the sum of the squares of the two equations in (7.55) gives

$$(k_1 - a_2\cos\theta_2)^2 + (-z_p - a_2\sin\theta_2)^2 = a_3^2 \tag{7.57}$$

Introducing more lumped constants, we can write (7.57) as

$$k_2\cos\theta_2 + k_3\sin\theta_2 = k_4 \tag{7.58}$$

where $k_2 = -2k_1a_2$, $k_3 = -2z_pa_2$, and $k_4 = a_3^2 - k_1^2 - a_2^2 - z_p^2$. Applying the solution of Example 7.2 to (7.58) yields the solution of θ_2, i.e.,

$$\theta_{2s} = \pm\arccos(k_4/D_2) + \phi_2 \tag{7.59}$$

where $D_2 = \sqrt{k_2^2 + k_3^2}$ and $\phi_2 = \text{Atan2}(k_3, k_2)$. Substituting the solution back into (7.55) gives the solution of θ_3 directly:

$$\theta_{3s} = \text{Atan2}(-z_p - a_2\sin\theta_{2s}, k_1 - a_2\cos\theta_{2s}) - \theta_{2s} \tag{7.60}$$

There are, in total, four solutions for one position of Point P.

Orientation:

The orientation of the end-effector is given by the orientation part of (7.47)

$$ {}^0\mathbf{R}_6 = {}^0\mathbf{R}_1(\theta_1)\,{}^1\mathbf{R}_2(\theta_2)\,{}^2\mathbf{R}_3(\theta_3)\,{}^3\mathbf{R}_4(\theta_4)\,{}^4\mathbf{R}_5(\theta_5)\,{}^5\mathbf{R}_6(\theta_6) \tag{7.61}$$

Since $\theta_1, \theta_2, \theta_3$ have been solved already, (7.61) simplifies into

$$ {}^3\mathbf{R}_6 = {}^3\mathbf{R}_4(\theta_4)\,{}^4\mathbf{R}_5(\theta_5)\,{}^5\mathbf{R}_6(\theta_6) \tag{7.62}$$

where

$$ {}^3\mathbf{R}_6 = {}^3\mathbf{R}_2(\theta_{3s})\,{}^2\mathbf{R}_1(\theta_{2s})\,{}^1\mathbf{R}_0(\theta_{1s})\,{}^0\mathbf{R}_6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{7.63}$$

To balance the number of variables on both sides, (7.62) can be written as

$$ {}^3\mathbf{R}_4^T(\theta_4)\,{}^3\mathbf{R}_6 = {}^4\mathbf{R}_5(\theta_5)\,{}^5\mathbf{R}_6(\theta_6) \tag{7.64}$$

Substituting the transformation matrices in (7.47) into (7.64) gives

$$ \begin{bmatrix} c_4 r_{11} - s_4 r_{31} & c_4 r_{12} - s_4 r_{32} & c_4 r_{13} - s_4 r_{33} \\ -s_4 r_{11} - c_4 r_{31} & -s_4 r_{12} - c_4 r_{32} & -s_4 r_{13} - c_4 r_{33} \\ r_{21} & r_{22} & r_{23} \end{bmatrix} = \begin{bmatrix} c_5 c_6 & -c_5 s_6 & -s_5 \\ s_6 & c_6 & 0 \\ s_5 c_6 & -s_5 s_6 & c_5 \end{bmatrix} \tag{7.65}$$

Entry (3,3) of (7.65) gives

$$ \theta_{5s} = \pm \arccos(r_{23}) \tag{7.66}$$

Entries (3,1) and (3,2) of (7.65) give

$$ \theta_{6s} = \text{Atan2}(-r_{22}/\sin\theta_{5s}, r_{21}/\sin\theta_{5s}) \tag{7.67}$$

Substituting the above solution into Entries (2,1) and (2,2) of (7.65) yields

$$ c_4 = (-s_{6s} r_{12} + c_{6s} r_{11})/(r_{12} r_{31} - r_{11} r_{32}) \tag{7.68}$$

$$ s_4 = (s_{6s} r_{32} - c_{6s} r_{31})/(r_{12} r_{31} - r_{11} r_{32}) \tag{7.69}$$

Finally, the solution of θ_4 is obtained as

$$ \theta_{4s} = \text{Atan2}(s_{6s} r_{32} - c_{6s} r_{31}, -s_{6s} r_{12} + c_{6s} r_{11}) \tag{7.70}$$

All eight solutions are summarised below.

$$ \begin{aligned} \theta_{1s} &= \pm\pi/2 + \text{Atan2}(-x_p, y_p) \\ \theta_{2s} &= \pm \arccos(k_4/D_2) + \phi_2 \\ \theta_{3s} &= \text{Atan2}(-z_p - a_2 \sin\theta_{2s}, k_1 - a_2\cos\theta_{2s}) - \theta_{2s} \\ \theta_{4s} &= \text{Atan2}(s_{6s} r_{32} - c_{6s} r_{31}, -s_{6s} r_{12} + c_{6s} r_{11}) \\ \theta_{5s} &= \pm \arccos(r_{23}) \\ \theta_{6s} &= \text{Atan2}(-r_{22}/\sin\theta_{5s}, r_{21}/\sin\theta_{5s}) \end{aligned} \tag{7.71}$$

where

$$ \begin{aligned} k_1 &= \cos\theta_{1s} x_p + \sin\theta_{1s} y_p \\ k_2 &= -2k_1 a_2 \\ k_3 &= -2 z_p a_2 \\ k_4 &= a_3^2 - k_1^2 - a_2^2 - z_p^2 \\ D_2 &= \sqrt{k_2^2 + k_3^2} \\ \phi_2 &= \text{Atan2}(k_3, k_2) \end{aligned}$$

$$ \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = {}^2\mathbf{R}_3(\theta_{3s})\,{}^1\mathbf{R}_2(\theta_{2s})\,{}^0\mathbf{R}_1(\theta_{1s})\,{}^0\mathbf{R}_6 $$

7.3 Univariate Polynomial

It is often impossible to find analytical solutions to the IK of some general high-DoF robotic manipulators. One approach is to reduce the kinematics equations into a *univariant polynomial.* Solving this polynomial gives all possible solutions of one variable, which can then be used to find out all sets of solutions to the IK. The order of the univariant polynomial varies depending on the complexity of the robot.

In order to derive a univariate polynomial of a robot, the basic kinematics relation based on the transformation matrices must be written in the form of polynomials. The common way is to utilise the tangent half-angle formulas given by

$$\sin\theta = \frac{2\tau}{1+\tau^2}, \quad \cos\theta = \frac{1-\tau^2}{1+\tau^2} \tag{7.72}$$

where

$$\tau = \tan\left(\frac{\theta}{2}\right) \tag{7.73}$$

Given the range of θ being $[-\pi, \pi)$, the range of τ is $(-\infty, +\infty)$. This feature is convenient because there is no additional constraint on the value of τ when polynomials are solved. Further, since $1+\tau^2$ is always positive, Formulas (7.72) do not have any singularity.

Example 7.6 (Univariate expression): Express the general DH transformation matrix in terms of polynomials.

Solution: The general transformation matrix is given by (6.3). Applying (7.72) to (6.3), we obtain

$$ {}^{i-1}_{i}T = \frac{1}{A}\begin{bmatrix} B & -2\tau_i & 0 & A\,a_{i-1} \\ 2\tau_i\cos\alpha_{i-1} & B\cos\alpha_{i-1} & -A\sin\alpha_{i-1} & -A\sin\alpha_{i-1}d_i \\ 2\tau_i\sin\alpha_{i-1} & B\sin\alpha_{i-1} & A\cos\alpha_{i-1} & A\cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.74}$$

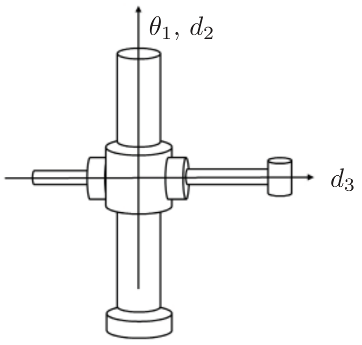
where $\tau_i = \tan\left(\frac{\theta_i}{2}\right)$, $A = 1+\tau_i^2$, and $B = 1-\tau_i^2$.

Example 7.7 (IK of a cylindrical robot): Find all valid joint configurations for a cylindrical robot defined Figure 7.3 when the end-effector is located at

$$ {}^{0}\mathbf{p}_3 = \begin{bmatrix} 0.1 \\ 0.05 \\ 0.3 \end{bmatrix} \text{m} \tag{7.75}$$

Solution: Here, we simply equate the position of the end-effector to the analytical solution:

$$\begin{bmatrix} 0.05\cos\theta_1 + d_3\sin\theta_1 \\ 0.05\sin\theta_1 - d_3\cos\theta_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.05 \\ 0.3 \end{bmatrix} \tag{7.76}$$



$$ {}^0\mathbf{p}_3 = \begin{bmatrix} 0.05\cos\theta_1 + d_3\sin\theta_1 \\ 0.05\sin\theta_1 - d_3\cos\theta_1 \\ d_2 \end{bmatrix} $$

FIGURE 7.3
A cylindrical robot with z-axes defined and end effector position ${}^0\mathbf{p}_3$.

By inspection, we already see that prismatic extension $d_2 = 0.3$ m. To solve for θ_1 and d_3, we must utilise the two upper equations in Equation (7.76). We can try the sum-squared method on the first two equations to try and isolate the unknown variables:

$$\begin{aligned}
(0.05\cos\theta_1 + d_3\sin\theta_1) + (0.05\sin\theta_1 - d_3\cos\theta_1) &= 0.1^2 + 0.05^2 \\
0.0025\left(\cos{\theta_1}^2 + \sin{\theta_1}^2\right) + {d_3}^2\left(\cos{\theta_1}^2 + \sin{\theta_1}^2\right) &= 0.0125 \\
{d_3}^2 + 0.0025 &= 0.0125 \\
d_3 &= \pm 0.1
\end{aligned} \tag{7.77}$$

Therefore, prismatic extension $d_3 = \pm 0.1$ m. However, we generally only consider positive values for prismatic actuators. Hence, we will choose to keep only the positive solution $d_3 = 0.1$ m. For the final variable θ_1, we choose the first equation from Equation (7.76) and use the tangent half-angle substitution in variable u (Equation (7.72)) to solve for θ_1

$$\begin{aligned}
0 &= 0.05\cos\theta_1 + d_3\sin\theta_1 - 0.1 \\
&= 0.05\left(\frac{1-u^2}{1+u^2}\right) + \left(d_3\frac{2u}{1+u^2}\right) - 0.1 \\
&= (2d_3 - 0.15)u^2 - 0.05
\end{aligned} \tag{7.78}$$

Substituting $d_3 = 0.1$, therefore,

$$u = \pm 1 \tag{7.79}$$

and

$$\begin{aligned}
\theta_1 &= 2\text{Atan2}(u) \\
&= 2\text{Atan2}(\pm 1) \\
&= \pm\frac{\pi}{2}\ \text{rad}
\end{aligned} \tag{7.80}$$

Note that there are two solutions for θ_1 due to the tangent half-angle substitution. It is important to validate the solutions found using this method because the roots of a quadratic may not have any physical meaning in the context of a robotic system. We do this by substituting the inverse kinematic solution into the forward kinematic solutions as defined in Figure 7.3. Substituting into forward kinematics ${}^0\mathbf{p}_3 = {}^0\mathbf{p}_3(\theta_1, d_2, d_3)$

$$ {}^0\mathbf{p}_3\left(\frac{\pi}{2}, 0.3, 0.1\right) = \begin{bmatrix} 0.1 \\ 0.05 \\ 0.3 \end{bmatrix} \quad \text{and} \quad {}^0\mathbf{p}_3\left(-\frac{\pi}{2}, 0.3, 0.1\right) = \begin{bmatrix} -0.1 \\ -0.05 \\ 0.3 \end{bmatrix} \tag{7.81}$$

As observed, the first solution matches the target solution. Therefore, the inverse kinematic solution to this problem is

$$\theta_1 = \frac{\pi}{2} \text{ rad} \tag{7.82}$$

$$d_2 = 0.3 \text{ m} \tag{7.83}$$

$$d_3 = 0.1 \text{ m} \tag{7.84}$$

7.4 Dialytic Method

The *dialytic method* efficiently reduces the number of polynomials into a simpler form by removing one or more variables. It can eliminate more than one variable simultaneously.

Example 7.8 (Solving simultaneous equations): Reduce the two equations in (x, y) given by (7.85) and (7.86) into a univariant polynomial in x.

$$y^2 + xy + 1 = 0 \tag{7.85}$$

$$x^2 y - 4 = 0 \tag{7.86}$$

Solution: This problem can be readily solved by substituting the expression of y based on (7.86) into (7.85). Consider a vector of $\begin{bmatrix} y^2 & y & 1 \end{bmatrix}^T$ containing only y. Then (7.85) is the dot-product of this vector with $\begin{bmatrix} 1 & x & 1 \end{bmatrix}^T$, while (7.86) is the dot-product of this vector with $\begin{bmatrix} 0 & x^2 & -4 \end{bmatrix}^T$. Create a new equation by multiplying (7.86) with y

$$x^2 y^2 - 4y = 0 \tag{7.87}$$

which is the dot product of $\begin{bmatrix} y^2 & y & 1 \end{bmatrix}^T$ and $\begin{bmatrix} x^2 & -4 & 0 \end{bmatrix}^T$. Therefore, (7.85)–(7.87) can be written as

$$\begin{bmatrix} 1 & x & 1 \\ 0 & x^2 & -4 \\ x^2 & -4 & 0 \end{bmatrix} \begin{bmatrix} y^2 \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.88}$$

where the square matrix must be singular since $\begin{bmatrix} y^2 & y & 1 \end{bmatrix}^T$ is a non-zero vector. The singularity of this square matrix yields

$$x^4 + 4x^3 + 16 = 0 \tag{7.89}$$

which is the univariant polynomial we are seeking. The answer can be readily verified.

Example 7.9 (IK of a 6R robot): The solutions to the IK of a general 6R robot are given in the following four equations,

$$C_i \tau_2^2 \tau_3 + 2B_i \tau_2 \tau_3 + A_i \tau_3 + F_i \tau_2^2 + 2E_i \tau_2 + D_i = 0 \tag{7.90}$$

where i= 1, 2, 3, 4, $\tau_2 = \tan(\theta_2/2)$, $\tau_3 = \tan(\theta_3/2)$, and $A_i, \ldots, F_i$ are quadratic functions in $\tau_1 = \tan(\theta_1/2)$. Reduce (7.90) into an univariant polynomial in terms of τ_1.

Solution: Multiplying τ_2 on both sides of (7.90) gives

$$C_i \tau_2^3 \tau_3 + 2B_i \tau_2^2 \tau_3 + A_i \tau_2 \tau_3 + F_i \tau_2^3 + 2E_i \tau_2^2 + D_i \tau_2 = 0 \tag{7.91}$$

Rewrite (7.90) and (7.91) in a matrix form:

$$\mathbf{M}\,\mathbf{x} = \mathbf{0} \tag{7.92}$$

where

$$\mathbf{M} = \begin{bmatrix} 0 & C_1 & 2B_1 & 0 & F_1 & A_1 & 2E_1 & D_1 \\ 0 & C_2 & 2B_2 & 0 & F_2 & A_2 & 2E_2 & D_2 \\ 0 & C_3 & 2B_3 & 0 & F_3 & A_3 & 2E_3 & D_3 \\ 0 & C_4 & 2B_4 & 0 & F_4 & A_4 & 2E_4 & D_4 \\ C_1 & 2B_1 & A_1 & F_1 & 2E_1 & 0 & D_1 & 0 \\ C_2 & 2B_2 & A_2 & F_2 & 2E_2 & 0 & D_2 & 0 \\ C_3 & 2B_3 & A_3 & F_3 & 2E_3 & 0 & D_3 & 0 \\ C_4 & 2B_4 & A_4 & F_4 & 2E_4 & 0 & D_4 & 0 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \tau_2^3\tau_3 \\ \tau_2^2\tau_3 \\ \tau_2\tau_3 \\ \tau_2^3 \\ \tau_2^2 \\ \tau_3 \\ \tau_2 \\ 1 \end{bmatrix} \tag{7.93}$$

Equation (7.92) indicates that $\mathbf{M}$ is singular, i.e.,

$$\det(\mathbf{M}) = 0 \tag{7.94}$$

Since $A_i, \ldots, F_i$ are all quadratic functions in τ_1, it can be readily shown that

$$\det(\mathbf{M}) = \sum_{k=0}^{16} a_k \tau_1^k = 0 \tag{7.95}$$

which is the univariant polynomial we are seeking. Solving (7.95) gives all solutions of θ_1. Once θ_1 is obtained, θ_2 and θ_3 are to be solved similarly upon (7.90).

MATLAB can be used to solve the inverse kinematics of a serial manipulator algebraically, but only on a case-by-case basis as there is no unified method to solve the inverse kinematics of general serial manipulators.[2] The method presented here utilises the *tangent half-angle substitution* method as expressed in Equation(7.72), which is very useful for solving transcendental equations normally seen in forward kinematic equations. The rest of the IK method requires a general understanding of both geometry and algebraic equations, such as using squared-sum operations to cancel out variables.

Example M7.1 (RR manipulator): Solve for θ_1 and θ_2 when given a position P for RR robot shown in Figure 7.4.

Solution: We are presented with two forward kinematic expressions that describe the end-effector position as a function of θ_1 and θ_2, x_P and y_P. Observing these two equations, we can already see that θ_1 can be eliminated by taking their squared-sum:

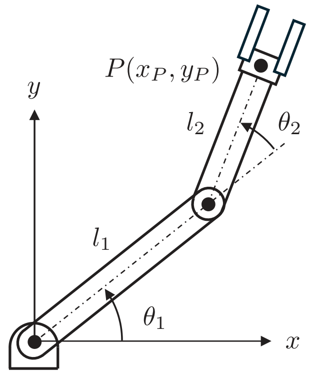
$$\begin{aligned} r &= (l_1 \cos\theta_1 + l_2\cos(\theta_1+\theta_2))^2 + (l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2))^2 \\ &= {l_1}^2 + 2l_1l_2\cos\theta_2 + {l_2}^2 \end{aligned} \tag{7.98}$$

where $r = {x_P}^2 + {y_P}^2$. Therefore, the obvious answer for θ_2 is

$$\theta_2 = \pi \pm \arccos\left(\frac{{l_1}^2 + {l_2}^2 - r}{2l_1l_2}\right) \tag{7.99}$$

Now that θ_2 is solved, we can solve for θ_1. This can be calculated from either x_P or y_P equations but is more challenging to solve using the algebraic method because of coupled sine and cosine terms. However, the tangent half-angle substitution can make light work of this problem for solving θ_1.

[2]There are research papers that attempt to generalise the IK for six-DoF serial manipulators, but the methodology is quite complex and is definitely outside the scope of this textbook.



$$x_P = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \quad (7.96)$$
$$y_P = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \quad (7.97)$$

FIGURE 7.4
RR manipulator with the forward kinematic equations.

First, choose either x_P or y_P equations to work with, then expand it. For this example, we will work with the x_P equation,

$$\begin{aligned} x_P &= l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ &= l_1 \cos\theta_1 + l_2 \cos\theta_1 \cos\theta_2 - l_2 \sin\theta_1 \sin\theta_2 \end{aligned} \quad (7.100)$$

Once expanded, we can perform the tangent half-angle substitution

$$\cos\theta_1 = \frac{u^2 - 1}{u^2 + 1} \qquad \sin\theta_1 = \frac{2u}{u^2 + 1} \quad (7.101)$$

where

$$u = \tan\frac{\theta_1}{2} \quad (7.102)$$

After substituting Equations (7.101) into Equation (7.100), the univariate polynomial for θ_1, after simplification is

$$(-l_1 - l_2 \cos\theta_2)u^2 + (l_1 + l_2 \cos\theta_2)u - 2l_2 \sin\theta_2 \quad (7.103)$$

Notice that Equation (7.103) is a quadratic, made up of all known quantities (θ_2 is solved first). Therefore, we can use the quadratic formula to solve for u. Hence

$$u = \frac{l_2 \cos\theta_2 \pm \sqrt{2({l_1}^2 + 2l_1 l_2 \cos\theta_2 + {l_2}^2)}}{l_1 + l_2 \cos\theta_2} \quad (7.104)$$

Finally, we can solve for θ_1 using Equation (7.102). Because Equation (7.104) is a fraction, we can actually use the two-parameter Atan2(y, x) so that the answer for is reflected in the correct quadrant:

$$\theta_1 = 2\text{Atan2}\left(l_1 + l_2 \cos\theta_2, l_2 \cos\theta_2 \pm \sqrt{2({l_1}^2 + 2l_1 l_2 \cos\theta_2 + {l_2}^2)}\right) \quad (7.105)$$

Therefore, our inverse kinematic solution for the RR manipulator, noting the dependent variables, is:

$$\theta_2(x_P, y_P) = \pi \pm \arccos\left(\frac{{l_1}^2 + {l_2}^2 - r(x_P, y_P)}{2l_1 l_2}\right) \quad (7.106)$$

$$\theta_1(\theta_2) = 2\text{Atan2}\left(l_1 + l_2 \cos\theta_2, l_2 \cos\theta_2 \pm \sqrt{2({l_1}^2 + 2l_1 l_2 \cos\theta_2 + {l_2}^2)}\right) \quad (7.107)$$

Our IK solution has a total of four solutions, two for each angle. Notice that θ_1 has two solutions because of the $\pm$ in the second parameter because the solution for θ_1 is derived from the single equation x_P. So, although the answer for θ_1 satisfies the x_P constraint, it may not satisfy the y_P constraint. Hence, two solutions (out of four in total) will be invalid because they do not meet the y_P constraint. This is one of the disadvantages of using the tangent half-angle substitution, as it can result in invalid extra solutions without checks.

The following MATLAB code performs the above steps to solve the inverse kinematics for this problem, with the ensuing command window output.

```matlab
clear
syms l1 l2 th1 th2 x y r real

% Kinematic equations
x_p = l1*cos(th1)+l2*cos(th1+th2)
y_p = l1*sin(th1)+l2*sin(th1+th2)

% Solve for th2 by squared sum of x and y
% r = x^2 + y^2 (radial length of position)
r_s = simplify(x_p^2+y_p^2)
t2 = solve(r_s == r, th2)

% Use tangent half-angle substitution to solve th1
syms u real
x_s = x_p - x
x_u = subs(expand(x_s), ...
    {cos(th1), sin(th1)}, ...
    {(1-u^2)/(1+u^2), 2*u/(1+u^2)});
x_u = simplifyFraction(x_u*(1+u^2))

% Extract coefficients (quadratic, lowest order first)
x_c = coeffs(x_u, u)

% Use quadratic formula to solve for u
a = x_c(3); b = x_c(2); c = x_c(1);
u1 = [(-b + sqrt(b^2 - 4*a*c)); (-b - sqrt(b^2 - 4*a*c))]

% Convert roots of u into angle
t1 = 2*atan2(u1, 2*a)

% Create function handles
t2_f = matlabFunction(t2)
t1_f = matlabFunction(t1)
```

```
x_p =

l2*cos(th1 + th2) + l1*cos(th1)


y_p =

l2*sin(th1 + th2) + l1*sin(th1)


r_s =

l1^2 + 2*cos(th2)*l1*l2 + l2^2


t2 =

 pi - acos((l1^2 + l2^2 - r)/(2*l1*l2))
 pi + acos((l1^2 + l2^2 - r)/(2*l1*l2))


x_s =

l2*cos(th1 + th2) - x + l1*cos(th1)


x_u =

l1 - x - l1*u^2 + l2*cos(th2) - u^2*x - 2*l2*u*sin(th2) - l2*u^2*cos(th2)


x_c =

[ l1 - x + l2*cos(th2), -2*l2*sin(th2), - l1 - x - l2*cos(th2)]
```

```
u1 =

 2*(((l1 - x + l2*cos(th2))*(4*l1 + 4*x + 4*l2*cos(th2)))/4 + l2^2*sin(th2)^2)^(1/2) +
     2*l2*sin(th2)
 2*l2*sin(th2) - 2*(((l1 - x + l2*cos(th2))*(4*l1 + 4*x + 4*l2*cos(th2)))/4 + l2^2*sin(
     th2)^2)^(1/2)


t1 =

 2*atan2(2*(((l1 - x + l2*cos(th2))*(4*l1 + 4*x + 4*l2*cos(th2)))/4 + l2^2*sin(th2)^2)
     ^(1/2) + 2*l2*sin(th2), - 2*l1 - 2*x - 2*l2*cos(th2))
 2*atan2(2*l2*sin(th2) - 2*(((l1 - x + l2*cos(th2))*(4*l1 + 4*x + 4*l2*cos(th2)))/4 + l2
     ^2*sin(th2)^2)^(1/2), - 2*l1 - 2*x - 2*l2*cos(th2))


t2_f =

  function_handle with value:

    @(l1,l2,r)[pi-acos((-r+l1.^2+l2.^2)./(l1.*l2.*2.0));pi+acos((-r+l1.^2+l2.^2)./(l1.*
     l2.*2.0))]


t1_f =

  function_handle with value:

    @(l1,l2,th2,x)[atan2(sqrt(((l1-x+l2.*cos(th2)).*(l1.*4.0+x.*4.0+l2.*cos(th2).*4.0))
     ./4.0+l2.^2.*sin(th2).^2).*2.0+l2.*sin(th2).*2.0,l1.*-2.0-x.*2.0-l2.*cos(th2).*2.0)
     .*2.0;atan2(sqrt(((l1-x+l2.*cos(th2)).*(l1.*4.0+x.*4.0+l2.*cos(th2).*4.0))./4.0+l2
     .^2.*sin(th2).^2).*-2.0+l2.*sin(th2).*2.0,l1.*-2.0-x.*2.0-l2.*cos(th2).*2.0).*2.0]
```

Example M7.2 (RR manipulator): For the same RR manipulator in Example M7.1, solve for θ_1 and θ_2 if $P = (1.0, 0.5)$, $l_1 = 1.0$ and $l_2 = 0.5$.

Solution: Use the function handles defined in Example M7.1 to find the four solutions for the inverse kinematics. The following script solves this problem, assuming script Inline 7.1 was executed first.

```
x = 1; y = 0.5;      % Define goal position
l1 = 1; l2 = 0.5;    % Link lengths
r = x^2 + y^2;       % Radial length r

% Solve theta_2
t2 = t2_f(l1, l2, r)

% Solve theta_1
t1 = t1_f(l1, l2, t2, x)

% Collate angles
t = [t1 reshape([t2 t2]',[],1)];
% Wrap angles so that we only get within range [-pi pi]
t_all = mod(t+pi, 2*pi) - pi

% Check answers
p_f = matlabFunction([x_p y_p])      % Function handle for EE pos

% Find position of EE for each 4 solutions
pos = cellfun(@(th) p_f(l1,l2,th(1),th(2)), num2cell(t_all, 2), 'Uni', 0);
pos = vertcat(pos{:})         % Convert cell to matrix

% Find sum squared error
err = sum((pos - [x y]).^2, 2)

% Remove answers that have EE error > 1e-4
t_final = t_all(err < 1e-4, :)
```

```
t2 =

    1.5708
    4.7124


t1 =
    5.3559
   -6.2832
    6.2832
   -5.3559


t_all =

   -0.9273    1.5708
         0    1.5708
         0   -1.5708
    0.9273   -1.5708


p_f =

  function_handle with value:

    @(l1,l2,th1,th2)[l2.*cos(th1+th2)+l1.*cos(th1),l2.*sin(th1+th2)+l1.*sin(th1)]


pos =

    1.0000   -0.5000
    1.0000    0.5000
    1.0000   -0.5000
    1.0000    0.5000


err =

    1.0000
         0
    1.0000
    0.0000


t_final =

         0    1.5708
    0.9273   -1.5708
```

The variable `t_all` contains all four IK solutions. However, it is known that two of them are invalid, hence error-checking is required. By substituting the four solutions into the forward kinematic equations (line 20), we arrive at the matrix `pos`, which contains the end-effector positions for the four IK solutions. By calculating the sum squared error (line 24), we find that IK solutions 1 and 3 are invalid. Hence the two valid IK solutions that remain in `t_final` (in degrees) are

TABLE 7.3
IK solutions for the RR manipulator in Figure 7.4 for $P = (1.0, 0.5)$.

IK Solution	θ_1	θ_2
1	$0°$	$90°$
2	$53.13°$	$-90°$

7.5 Conclusion

In this chapter, we described the inverse kinematic problem, where, for a serial manipulator, we find the joint configurations required to achieve a particular end-effector pose. This pose can be described by a point or transformation matrix. This is an important and highly practical problem to solve in robotics, as we typically describe the robot's end-effector pose in the task space coordinates rather than joint space.

In general, the inverse kinematics problem is difficult to solve compared to the direct kinematics of serial robotic manipulators, and there is no universal approach to solving such problems. However, we have introduced some mathematical techniques which can be used to solve these difficult problems.

1. **Balancing**: Keep the numbers of variables on both sides of equations as balanced as possible.
2. **Decoupling**: Identify a subset of solvable equations containing fewer variables and decouple them from the rest of the equations.
3. **Lumping**: construct lumped constants to simplify the expressions of the equations.
4. **Forming**: Form the equations into familiar forms and utilise the given solutions, such as (7.10).

In addition to these techniques, we also find that the geometric method is a viable approach to solving the IK problem. This is an intuitive method that requires a geometrical understanding of the robot's kinematic motion. Another mathematical method to solve transcendental expressions is the tangent half-angle substitution method. The advantage of this method is that it can be somewhat generalised for programming purposes so that it can be solved in numerical software such as MATLAB.

7.6 Exercises

Problem 1. Derive the inverse kinematics solutions for the three-link manipulator shown in Figure 7.5, when

1. the target transformation matrix ${}^0\mathbf{T}_3$ is given

$$
{}^0\mathbf{T}_3 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

2. the target position of an end effector positioned ${}^3\mathbf{p}_g = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$ in Frame $\{3\}$ is given in Frame $\{0\}$ as ${}^0\mathbf{p}_g = \begin{bmatrix} x & y & z \end{bmatrix}^T$

The DH parameters of the manipulator are as follows

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	90°	l_1	0	θ_2
3	0	l_2	0	θ_3

Problem 2.
For the planar robot shown in Figure 7.6, the transformation matrices between the base and the tool frames are:

$$
{}^0\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^1\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & l_1 \\ 0 & 0 & -1 & -d_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

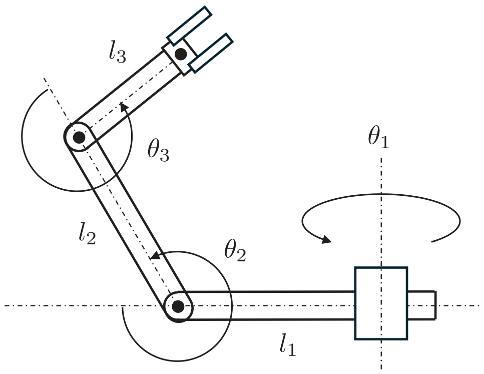


FIGURE 7.5
The RRR manipulator.

$$
{}^2\mathbf{T}_3 = \begin{bmatrix} s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ c_3 & -s_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^3\mathbf{T}_4 = \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Note: Your answers should reflect the use of these transformation matrices, or otherwise the correct ${}^0\mathbf{T}_4$ generated by the matrices above. Do not reassign frames and use a different set of matrices, or they will be marked as incorrect.

1. Solve the inverse kinematics of the robot using the **algebraic method**; that is find equations for θ_1, d_2, and q_3 in terms of the arbitrary location and orientation of the end effector shown in ${}^0\mathbf{T}_4$.

$$
{}^0\mathbf{T}_4 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

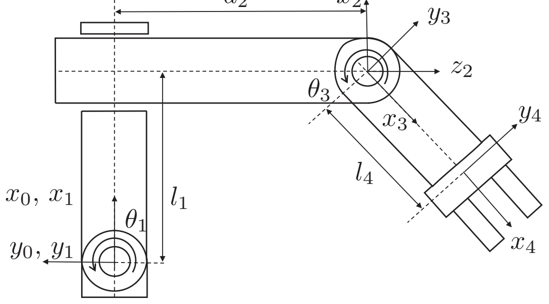


FIGURE 7.6
Planar robot for Problem 2.

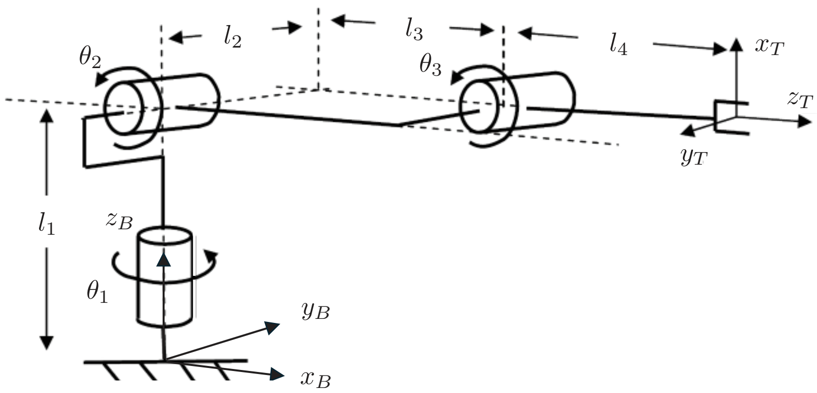


FIGURE 7.7
A serial spherical robotic manipulator.

2. If the transformation matrix relating the base and tool frames is:

$$^0\mathbf{T}_4 = \begin{bmatrix} -0.2588 & 0.9659 & 0 & 3.69680 \\ -0.9659 & -0.2588 & 0 & -8.0599 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Find all valid solutions for the joint variables, if one exists. Let $l_1 = 2$ and $l_3 = 4$, and assume $d_2 \geq 0$.

Problem 3. Consider a serial manipulator with three revolute joints as shown in Figure 7.7.

1. Frame $\{B\}$ is defined for the base of this manipulator, as shown in Figure 7.7. Find the position of the origin of frame T with respect to $\{B\}$, i.e., find $^B\mathbf{p}_{T_{org}}$.
2. With the forward kinematics found in (a) and link lengths: $l_1 = 0.5$ m, $l_2 = 0.2$ m, $l_3 = 0.4$ m, and $l_4 = 0.3$ m, evaluate the position of the Tool point T_{ORG} when $\theta_1 = 0°$, $\theta_2 = 30°$, and $\theta_3 = 90°$.
3. Assume that the position of the Tool with respect to $\{B\}$ is given by

$$^B\mathbf{p}_{T_{org}} = \begin{bmatrix} -0.058 \\ -0.250 \\ -0.566 \end{bmatrix}$$

Let the link lengths be: $l_1 = 0.5$ m, $l_2 = 0.4$ m, $l_3 = 0.4$ m, and $l_4 = 0.2$ m. It is also known that $q_3 = 30°$. Conduct the inverse kinematics to find the joint displacements θ_1 and θ_2.

Problem 4. Given a robotic manipulator. The position of the end-effector with respect to the ground frame is given by

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} c_1(l_2c_2 + l_3c_{23}) \\ s_1(l_2c_2 + l_3c_{23}) \\ -l_2s_2 - l_3s_{23} \end{bmatrix}$$

where θ_i, for $i = 1, 2, 3$ are joint angles.

1. Derive the solution to the inverse kinematics of this manipulator based on the above formula. Identify how many different solutions in total.
2. Given $l_2 = 3$, $l_3 = 2$, $p_x = 4$, $p_y = 2$, and $p_z = 1$, find the numerical values of q_i, for $i = 1, 2, 3$.

Problem 5.

The schematic structure, frames, dimensions of links and home positions of joints of the legs of the Bioloid robot are shown in Figure 7.8. The configuration of the right knee (Joint ID 13) of the Boiloid robot with respect to a reference frame $\{ORL\}$ (origin-right-leg, $\{O\}$ hereafter for simplicity) on its body is given by the following transformation matrix

$$ {}^{O}\mathbf{T}_{13} = \begin{bmatrix} {}^{O}\mathbf{R}_{13} & {}^{O}\mathbf{p}_{13} \\ 0 & 1 \end{bmatrix} $$

where

$$ {}^{O}\mathbf{R}_{13} = \begin{bmatrix} -s_7\sin(\theta_{11}-\theta_{13}) - c_7c_9\cos(\theta_{11}-\theta_{13}) & s_7\cos(\theta_{11}-\theta_{13}) - c_7s_9\sin(\theta_{11}-\theta_{13}) & c_7c_9 \\ c_7\sin(\theta_{11}-\theta_{13}) - s_7s_9\cos(\theta_{11}-\theta_{13}) & -c_7\cos(\theta_{11}-\theta_{13}) - s_7s_9\sin(\theta_{11}-\theta_{13}) & c_7s_9 \\ c_9\cos(\theta_{11}-\theta_{13}) & c_9\sin(\theta_{11}-\theta_{13}) & s_9 \end{bmatrix} $$

$$ {}^{O}\mathbf{p}_{13} = \begin{bmatrix} -76.98s_7s_{11} - 76.98c_7s_9c_{11} \\ 76.98c_7s_{11} - 76.98s_7s_9c_{11} \\ 120.75 + 76.98c_9c_{11} \end{bmatrix} $$

1. Derive the analytical solution for Joints 7, 9, 11, and 13, so that the right knee reaches a given position

$$ {}^{O}\mathbf{p}_{13} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} $$

while x_{13} and z_{13} are in the same directions as z_O and x_O, respectively.

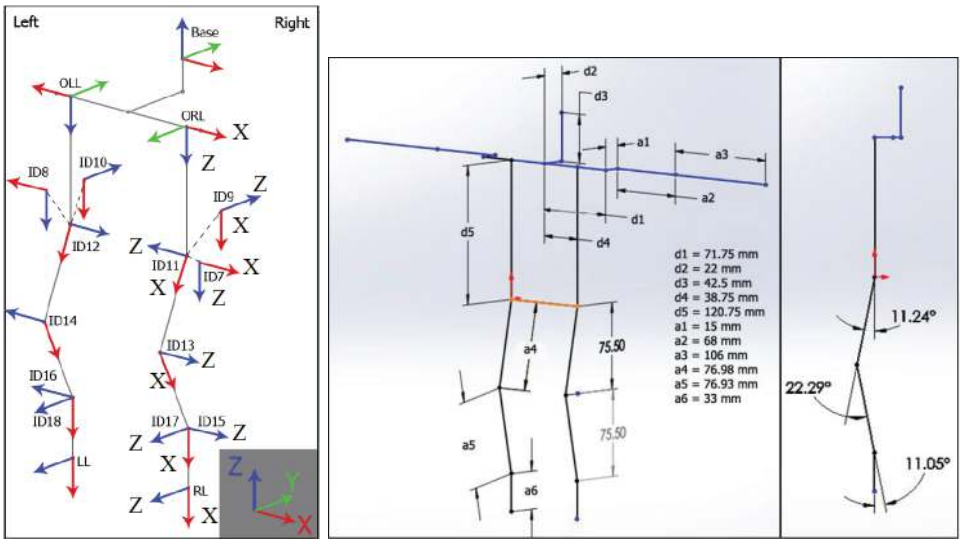


FIGURE 7.8
Legs of the Bioloid robot.

2. From the analytical solution, calculate the numerical solution when

$$^{O}\mathbf{p}_{13} = \begin{bmatrix} -19.245 \\ 66.667 \\ 154.083 \end{bmatrix}$$

Problem 6. In deriving the solution of the inverse kinematics of a general three-DoF robot, the following equations are formulated

$$\Gamma_1{}^2\Gamma_2{}^2 + 3\Gamma_1{}^2\Gamma_2 - 4\Gamma_1\Gamma_2{}^2 + 4 = 0$$

$$2\Gamma_1{}^2\Gamma_2{}^2 + \Gamma_1{}^2\Gamma_2 - 3\Gamma_1\Gamma_2{}^2 + 1 = 0$$

Please reduce the above two equations into a univariant polynomial in Γ_1.

Problem 7. The position of the end-effector of the JACO manipulator, with respect to its proximal joint, is given by

$$^{3}\mathbf{p}_7 = \begin{bmatrix} (228.5 - 80.4c_5)s_4 - 160.7c_4s_5 \\ 424.8 + 139.2c_5 \\ (228.5 - 80.4c_5)c_4 + 160.7s_4s_5 \end{bmatrix}$$

Assuming only θ_4 and θ_5 are actuated, determine all sets of numerical solutions (θ_4 and θ_5) for the end-effector to reach

$$^{3}\mathbf{p}_7 = \begin{bmatrix} 0 \\ 494.4 \\ 1 \end{bmatrix}$$

8

Jacobian Analysis

In Chapter 2, the time derivative method was used directly on the forward kinematic equations of a serial robot, which we then used to derive the robot's Jacobian matrix. However, we will see that this method becomes more complicated as robots exhibit more degrees of freedom (DoF). To alleviate the challenge, this chapter will introduce a new method for velocity analysis called *velocity propagation*, an iterative method that can simplify the derivation of velocity equations.

We will further expand on Chapter 2 and show that static analysis, singularity analysis, and workspace analysis of a robotic manipulator are all linked to its Jacobian matrix for the general serial manipulator case.

8.1 Jacobian Matrix

A *Jacobian matrix* is defined as a multidimensional form of the derivatives of a set of functions. For example, given a set of functions

$$\begin{aligned} f_1 &= f_1(x_1, \ldots, x_n) \\ &\vdots \\ f_m &= f_m(x_1, \ldots, x_n) \end{aligned} \tag{8.1}$$

The time derivatives of the above functions are given by

$$\begin{aligned} \dot{f}_1 &= \frac{\partial f_1}{\partial x_1}\dot{x}_1 + \ldots + \frac{\partial f_1}{\partial x_n}\dot{x}_n \\ &\vdots \\ \dot{f}_m &= \frac{\partial f_m}{\partial x_1}\dot{x}_1 + \ldots + \frac{\partial f_m}{\partial x_n}\dot{x}_n \end{aligned}$$

which can be rewritten in a matrix form, i.e.,

$$\dot{\mathbf{f}} = \mathbf{J}\,\dot{\mathbf{x}} \tag{8.2}$$

where $\dot{\mathbf{f}} = \begin{bmatrix}\dot{f}_1 & \ldots & \dot{f}_m\end{bmatrix}^T$, $\dot{\mathbf{x}} = \begin{bmatrix}\dot{x}_1 & \ldots & \dot{x}_n\end{bmatrix}^T$, and

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{8.3}$$

DOI: 10.1201/9781003614319-8

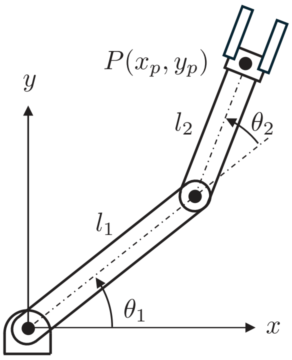


FIGURE 8.1
RR robot.

Here **J** is the Jacobian matrix of the functions in (8.1). In general, the Jacobian for a serial robot can be obtained by the definition (8.2) directly. If the functions represent the position of an end-effector, the time derivatives are its velocity. We will also show that the Jacobian is quite useful in static and workspace analysis later.

Notation

From this point, unless stated otherwise, these common mathematical symbols represent the following vectors when handling Jacobians

$\mathbf{q}$	Generalised vector of joint space variables
$\mathbf{x}$	Generalised vector of task space variables

Example 8.1 (Linear velocity via differentiation): Find the velocity of the end-effector of the 2R robot shown in Figure 8.1. Assume $l_1 = l_2 = 1$ m.

Solution: The forward kinematics of this robot was derived previously as

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} c_1 + c_{12} \\ s_1 + s_{12} \end{bmatrix}$$

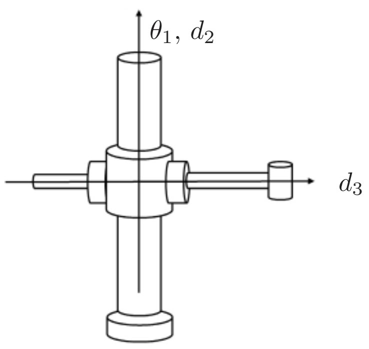
The velocity of the end-effector is obtained by the differentiation as

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \begin{bmatrix} -s_1 - s_{12} & -s_{12} \\ c_1 + c_{12} & c_{12} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} -s_1\dot{\theta}_1 - s_{12}(\dot{\theta}_1 + \dot{\theta}_2) \\ c_1\dot{\theta}_1 + c_{12}(\dot{\theta}_1 + \dot{\theta}_2) \end{bmatrix} \tag{8.4}$$

Example 8.2 (Jacobian matrix): Find the Jacobian matrix for the end-effector for the cylindrical robot shown in Figure 8.2.

Solution: Take the time derivative of the end-effector position vector ${}^0\mathbf{p}_E$ such that we get

$$\frac{d({}^0\mathbf{p}_E)}{dt} = {}^0\dot{\mathbf{p}}_E = \begin{bmatrix} \sin\theta_1\dot{d}_3 + \cos\theta_1 d_3\dot{\theta}_1 - a_3\sin\theta_1\dot{\theta}_1 \\ -\cos\theta_1\dot{d}_3 + \sin\theta_1 d_3\dot{\theta}_1 + a_3\cos\theta_1\dot{\theta}_1 \\ \dot{d}_2 \end{bmatrix} \tag{8.5}$$



$$
{}^{0}\mathbf{p}_E = \begin{bmatrix} a_3\cos\theta_1 + d_3\sin\theta_1 \\ a_3\sin\theta_1 - d_3\cos\theta_1 \\ d_2 \end{bmatrix}
$$

FIGURE 8.2
A cylindrical robot with end-effector position ${}^{0}\mathbf{p}_E$ derived from forward kinematics.

Rewrite Equation (8.5) in the following form

$$
\begin{aligned}
{}^{0}\dot{\mathbf{p}}_E &= \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} && (8.6)\\
&= \begin{bmatrix} d_3\cos\theta_1 - a_3\sin\theta_1 & 0 & \sin\theta_1 \\ a_3\cos\theta_1 + d_3\sin\theta_1 & 0 & -\cos\theta_1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{d}_2 \\ \dot{d}_3 \end{bmatrix} && (8.7)
\end{aligned}
$$

where the Jacobian is the 3×3 matrix $\mathbf{J}(\mathbf{q})$ which maps joint velocities $\dot{\mathbf{q}}$ to the end-effector velocity ${}^{0}\dot{\mathbf{p}}_E$ for a given joint configuration $\mathbf{q}$.

Example 8.3 (six-DoF Jacobian matrix): Find the general structure of the Jacobian matrix for the six-DoF PUMA robot with a spherical wrist, as shown in Figure 8.3, that maps joint velocity to end-effector linear velocity.

Solution: Assume the position of the end-effector is given by point P. Then the position of P in $\{0\}$ is given by

$$
{}^{0}\mathbf{p} = {}^{0}\mathbf{T}_1(\theta_1)\ {}^{1}\mathbf{T}_2(\theta_2)\ {}^{2}\mathbf{T}_3(\theta_3)\ {}^{3}\mathbf{T}_4(\theta_4)\ {}^{4}\mathbf{T}_5(\theta_5)\ {}^{5}\mathbf{T}_6(\theta_6)\ {}^{6}\mathbf{p} \qquad (8.8)
$$

where ${}^{i}\mathbf{T}_{i+1}\theta_{i+1}$ are derived from forward kinematics as described in Example 6.6. The above

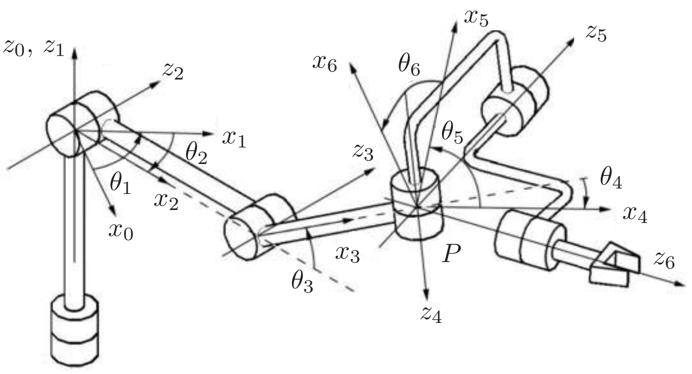


FIGURE 8.3
A six-DoF PUMA (articulated) robot with a spherical wrist.

equation can then be expanded as

$$x_p = x_p(\theta_1, \dots, \theta_6) \tag{8.9}$$

$$y_p = y_p(\theta_1, \dots, \theta_6) \tag{8.10}$$

$$z_p = z_p(\theta_1, \dots, \theta_6) \tag{8.11}$$

which describe the position of the end-effector, given all joint positions θ_1 to θ_6. Therefore, the overall structure of the Jacobian is

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x_p}{\partial \theta_1} & \frac{\partial x_p}{\partial \theta_2} & \cdots & \frac{\partial x_p}{\partial \theta_6} \\ \frac{\partial y_p}{\partial \theta_1} & \frac{\partial y_p}{\partial \theta_2} & \cdots & \frac{\partial y_p}{\partial \theta_6} \\ \frac{\partial z_p}{\partial \theta_1} & \frac{\partial z_p}{\partial \theta_2} & \cdots & \frac{\partial z_p}{\partial \theta_6} \end{bmatrix} \tag{8.12}$$

which is a 3×6 matrix that maps joint velocities to end-effector linear velocities at a given joint configuration by

$$^0\dot{\mathbf{p}} = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{8.13}$$

where $\boldsymbol{\theta} = [\theta_1, \dots, \theta_6]^T$.

Example 8.4 (Linear velocity via Jacobians): Referencing the same PUMA robot in Figure 8.3, find the linear velocity of the wrist frame for any given joint configuration and joint velocity, where the link lengths are $L_1 = 0.5$ m between z_2 and z_3, and $L_2 = 0.4$ m between z_3 and z_4.

Solution: To find the linear velocity of the wrist frame, we need to find the Jacobian, which maps joint velocities to the linear velocity of $\{4\}$ for any given joint configuration. We know from Example 6.6 that

$$^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^1\mathbf{T}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_2 & -\cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.14}$$

$$^2\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 0.5 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.15}$$

and the position of the wrist relative to $\{3\}$ is

$$^3\mathbf{p}_w = \begin{bmatrix} 0.4 \\ 0 \\ 0 \end{bmatrix}$$

Therefore, the position of the wrist relative to ground Frame 0 is

$$^0\mathbf{p}_w = {}^0\mathbf{T}_1\,{}^1\mathbf{T}_2\,{}^2\mathbf{T}_3\,{}^3\mathbf{p}_w \tag{8.16}$$

$$= \begin{bmatrix} 0.4\cos\theta_1\cos(\theta_2+\theta_3) + 0.5\cos\theta_2 \\ 0.4\sin\theta_1\cos(\theta_2+\theta_3) + 0.5\cos\theta_2 \\ 0.4\sin(\theta_2+\theta_3) + 0.5\sin\theta_2 \end{bmatrix} \tag{8.17}$$

Find the Jacobian by taking the time derivative of $^0\mathbf{p}_w$ and rewriting it into the form as follows

$$^0\dot{\mathbf{p}}_w = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}$$

$$= \begin{bmatrix} -s_1(0.4c_{23} + 0.5c_2) & -c_1(0.4s_{23} + 0.5s_2) & -0.4s_{23}c_1 \\ c_1(0.4c_{23} + 0.5c_2) & -s_1(0.4s_{23} + 0.5s_2) & 0.4s_{23}s_1) \\ 0 & -0.4c_{23} - 0.5c_2 & -0.4c_{23} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{8.18}$$

Therefore, the linear velocity of the wrist frame for any given joint configuration and velocity is given by Equation (8.18), where the Jacobian is represented by the 3×3 matrix.

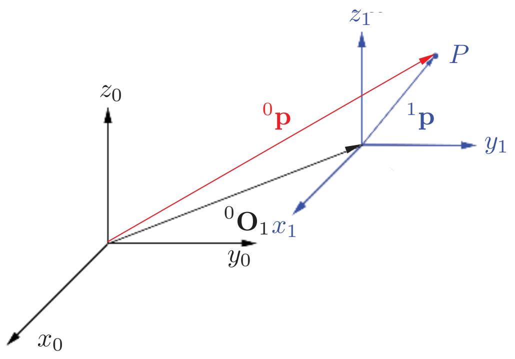


FIGURE 8.4
Pure translation.

8.2 Velocity in Translation and Rotation

Note: Please refer to the Glossary for notation on linear and angular velocities.

To gain in-depth insight into a robotic system's velocity, we conduct the analysis in an alternative way here. Firstly, we derive basic formulas for translation and rotation.

In the case of pure translation, as shown in Figure 8.4, the moving frame {1} is always parallel to the ground frame {0}. The position of arbitrary point P attached to {1} is given by

$$ {}^0\mathbf{p} = {}^0\mathbf{O}_1 + {}^1\mathbf{p} \tag{8.19} $$

where ${}^1\mathbf{p}$ is an invariant vector. The velocity of Point P is obtained by differentiating the above equation with respect to time, i.e,

$$ {}^0\dot{\mathbf{p}} = {}^0\dot{\mathbf{O}}_1 \tag{8.20} $$

its physical meaning is that all points on this body have the same translational velocity.

Angular velocity is associated with a body or a frame with rotation as shown in Figure 8.5, where a moving frame {1} rotates around a pivot O in the ground frame {0}. The orientation of {1} in {0} is noted as ${}^0\mathbf{R}_1(t)$. Further consider a point P attached to {1}, whose position is given by

$$ {}^0\mathbf{p} = {}^0\mathbf{R}_1(t)\ {}^1\mathbf{p} \equiv \mathbf{R}(t)\ {}^1\mathbf{p} \tag{8.21} $$

where ${}^1\mathbf{p}$ is an invariant vector. The velocity of Point P is the time derivative of its position, i.e.,

$$ {}^0\dot{\mathbf{p}} = \dot{\mathbf{R}}(t)\ {}^1\mathbf{p} \tag{8.22} $$

Since ${}^1\mathbf{p} = \mathbf{R}^T(t)\ {}^0\mathbf{p}$, we have

$$ {}^0\dot{\mathbf{p}} = \dot{\mathbf{R}}(t)\ {}^1\mathbf{p} = \dot{\mathbf{R}}(t)\ \mathbf{R}^T\ {}^0\mathbf{p} \tag{8.23} $$

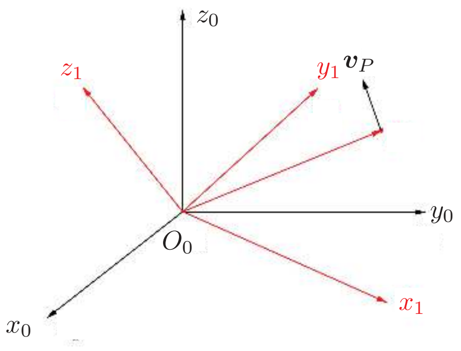


FIGURE 8.5
Pure rotation.

Define $\boldsymbol{\Omega} \equiv \dot{\mathbf{R}}\ \mathbf{R}^T$ such that the above equation becomes

$$^0\dot{\mathbf{p}} = \boldsymbol{\Omega}\ ^0\mathbf{p} \tag{8.24}$$

It can be shown that $\boldsymbol{\Omega}$ is *skew-symmetric*, i.e., $\boldsymbol{\Omega}^T = -\boldsymbol{\Omega}$. The proof is in Appendix 24.1. Further, a skew-symmetric matrix can always be written as

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times$$

where $\boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ is called the angular velocity, while $\boldsymbol{\Omega}$ is called the crossproduct matrix of the angular velocity. The proof is in Appendix 24.2. Hence, the relation between a rotation matrix and the corresponding angular velocity is given by

$$\boldsymbol{\Omega} \equiv \dot{\mathbf{R}}\ \mathbf{R}^T = [\boldsymbol{\omega}\times] \tag{8.25}$$

where $[\boldsymbol{\omega}\times]$ stands for the crossproduct matrix of $\boldsymbol{\omega}$. Imposing the frame indices on the rotation matrix, (8.25) becomes

$$^0\boldsymbol{\Omega}_1 \equiv {}^0\dot{\mathbf{R}}_1\ {}^0\mathbf{R}_1^T = [{}^0_0\boldsymbol{\omega}_1\times] \tag{8.26}$$

where ${}^0_0\boldsymbol{\omega}_1$ is the angular velocity of Frame $\{1\}$ with respect to Frame $\{0\}$, and also measured in Frame $\{0\}$. This nomenclature is explained below.

A position vector of a point is defined as the vector from the origin of the frame to this point. This frame is called the reference frame. The coordinates of this vector are obtained by projecting this vector onto all axes of the same frame. We call this frame the measuring frame. Implicitly, the reference frame and the measure frame for a position vector are always the same. For example, the reference frame and the measuring frame of $^0\mathbf{p}$ are $\{0\}$, while those of $^1\mathbf{p}$ are $\{1\}$.

Different from a position vector, a velocity vector may have different reference and measuring frames.

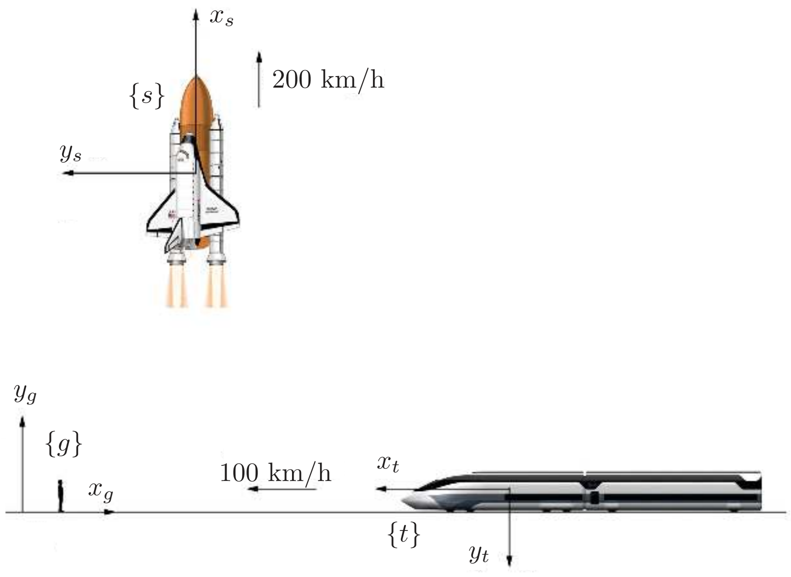


FIGURE 8.6
Relative velocities.

Example 8.5 (Velocity frames): As shown in Figure 8.6, there are three observers: one on the ground using a ground frame $\{g\}$, one in a train using a train-attached frame $\{t\}$, and one in a spaceship using a ship-attached frame $\{s\}$. All the frames are illustrated in Figure 8.6. Given that the train is travelling west at a speed of 100 km/h, while the spaceship is flying upwards at a speed of 200 km/h. Find the velocity of the train with the reference frame and the measuring frame being any one or two frames from $\{g\}$, $\{t\}$, and $\{s\}$.

Solution: Let us start with ${}^s_g\mathbf{v}_t$, where s, g, t refer to the measure frame $\{s\}$, the reference frame $\{g\}$, and the frame of moving body $\{t\}$. The relative speed between the train and the ground is 100 km/h west. Since $\{s\}$ is the measure frame, this velocity is projected onto the axes of $\{s\}$ to obtain ${}^s_g\mathbf{v}_t$, i.e.,

$$
{}^s_g\mathbf{v}_t = \begin{bmatrix} 0 \\ 100 \end{bmatrix} \tag{8.27}
$$

The velocities of the train in different frames are summarised in Table 8.1. Note that when the reference frame is $\{t\}$, the train's speed is always zero. However, if the measuring frame is $\{t\}$, the train's speed does not have to be zero. If interested, you can try to find the velocity of the spaceship in different frames.

A general motion consists of simultaneous translation and rotation as shown in Figure 8.7, where $\{1\}$ moves with respect to $\{0\}$. The position of a point $\mathbf{p}$ attached to $\{1\}$ is given by

$$
\begin{aligned}
{}^0_0\mathbf{p} &= {}^0_0\mathbf{O}_1 + {}^0_1\mathbf{p} \\
&= {}^0_0\mathbf{O}_1 + {}^0\mathbf{R}_1\, {}^1_1\mathbf{p}
\end{aligned} \tag{8.28}
$$

TABLE 8.1
Velocities represented in different frames

Ref. / Mea.	$\{g\}$	$\{t\}$	$\{s\}$
$\{g\}$	${}^g_g\mathbf{v}_t = \begin{bmatrix}-100\\0\end{bmatrix}$	${}^t_g\mathbf{v}_t = \begin{bmatrix}100\\0\end{bmatrix}$	${}^s_g\mathbf{v}_t = \begin{bmatrix}0\\100\end{bmatrix}$
$\{t\}$	${}^g_t\mathbf{v}_t = \begin{bmatrix}0\\0\end{bmatrix}$	${}^t_t\mathbf{v}_t = \begin{bmatrix}0\\0\end{bmatrix}$	${}^s_t\mathbf{v}_t = \begin{bmatrix}0\\0\end{bmatrix}$
$\{s\}$	${}^g_s\mathbf{v}_t = \begin{bmatrix}-100\\-200\end{bmatrix}$	${}^t_s\mathbf{v}_t = \begin{bmatrix}100\\200\end{bmatrix}$	${}^s_s\mathbf{v}_t = \begin{bmatrix}-200\\100\end{bmatrix}$

The velocity of point $\mathbf{p}$ with respect to ground is obtained by differentiating the above equation, i.e.,

$$\begin{aligned}
{}^1\dot{\mathbf{p}} &= {}^0\dot{\mathbf{O}}_1 + {}^0\dot{\mathbf{R}}_1\, {}^1_1\mathbf{p} \\
&= {}^0\dot{\mathbf{O}}_1 + {}^0\dot{\mathbf{R}}_1\, {}^0\mathbf{R}_1^T\, {}^0_1\mathbf{p} \\
&= {}^0\dot{\mathbf{O}}_1 + {}^0\boldsymbol{\omega}_1 \times {}^0_1\mathbf{p} \qquad (8.29)
\end{aligned}$$

which is measured in $\{0\}$. This velocity can also be measured in $\{1\}$ by premultiplying the rotation matrix on (8.29), i.e.,

$$\begin{aligned}
{}^1\dot{\mathbf{p}} &= {}^1\mathbf{R}_0\, {}^1\dot{\mathbf{p}} \\
&= {}^1\mathbf{R}_0\, {}^0\dot{\mathbf{O}}_1 + {}^1\mathbf{R}_0({}^0\boldsymbol{\Omega}_1 \times {}^0_1\mathbf{p}) \\
&= {}^1\mathbf{R}_0\, {}^0\dot{\mathbf{O}}_1 + {}^1\mathbf{R}_0\, {}^0\boldsymbol{\Omega}_1 \times {}^1\mathbf{R}_0\, {}^0_1\mathbf{p} \\
&= {}^1\dot{\mathbf{O}}_1 + {}^1\boldsymbol{\Omega}_1 \times {}^1_1\mathbf{p} \qquad (8.30)
\end{aligned}$$

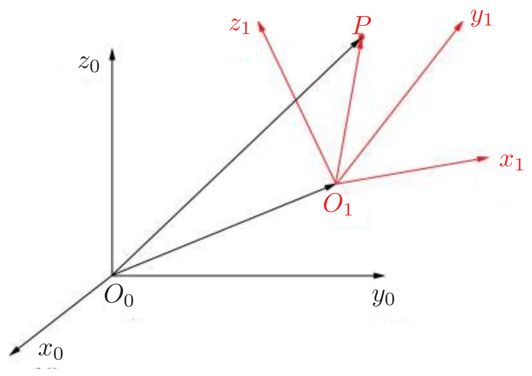


FIGURE 8.7
General motion.

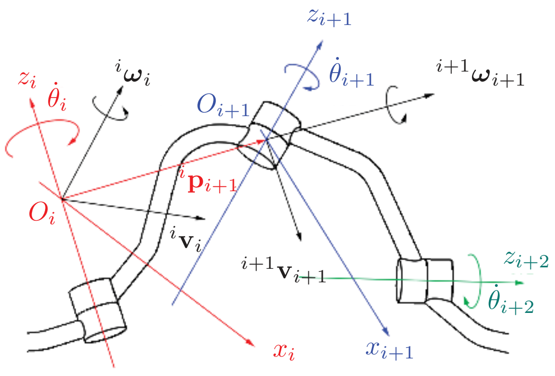


FIGURE 8.8
Velocities of connected links.

8.3 Velocity Propagation

The idea of velocity propagation is to find the velocity of the origin and the angular velocity of the $i+1$th frame, $^{i+1}\boldsymbol{\omega}_{i+1}$ and $^{i+1}\mathbf{v}_{i+1}$, based on the information of the velocity of the origin and the angular velocity of the ith frame, $^{i}\boldsymbol{\omega}_i$ and $^{i}\mathbf{v}_i$, and the joint velocity, $\dot{q}_{i+1}$, as shown in Figure 8.8. Angular velocities are easy to handle, in which the general formula is given by

$$^{m}\boldsymbol{\omega}_k = {}^{m}_{i}\boldsymbol{\omega}_j + {}^{m}_{j}\boldsymbol{\omega}_k \tag{8.31}$$

where $\{i\}$, $\{j\}$, $\{k\}$, and $\{m\}$ are four different frames. The meaning of (8.31) is that the angular velocity of $\{k\}$ with respect to $\{i\}$ equals the sum of the angular velocity of $\{j\}$ with respect to $\{i\}$ and the angular velocity of $\{k\}$ with respect to $\{j\}$. This is true for any uniform measure frame $\{m\}$. Applying (8.31) to the system shown in Figure 8.8, we have

$$^{i}\boldsymbol{\omega}_{i+1} = {}^{i}\boldsymbol{\omega}_i + {}^{i}_{i}\boldsymbol{\omega}_{i+1} \tag{8.32}$$

where the measure frame is $\{i\}$, and the omitted reference frame is $\{0\}$. Premultiplying $^{i+1}\mathbf{R}_i$ on both sides of (8.32) yields,

$$^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^{i}\boldsymbol{\omega}_i + {}^{i+1}_{i}\boldsymbol{\omega}_{i+1} \tag{8.33}$$

Note that $^{i+1}_{i}\boldsymbol{\omega}_{i+1}$ is the angular velocity of Link $i+1$ with respect to Link i, measured in $\{i+1\}$. Hence, its direction is along the Z axis of $\{i+1\}$, while its magnitude is the joint velocity, i.e.,

$$^{i+1}_{i}\boldsymbol{\omega}_{i+1} = \dot{q}_{i+1} \;\; \mathbf{k} \tag{8.34}$$

where $\mathbf{k} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$. Therefore, (8.33) can be further written as

$$^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^{i}\boldsymbol{\omega}_i + \dot{q}_{i+1} \;\; \mathbf{k} \tag{8.35}$$

which is the formula for the *propagation of angular velocities.* We can see that the angular velocity of $\{i\}$ and the joint velocity $\dot{\theta}_{i+1}$ are transformed into the angular velocity of $\{i+1\}$ in (8.33).

Point (origin) velocity propagation is to find ${}^{i+1}\mathbf{v}_{i+1}$, by knowing ${}^{i}\boldsymbol{\omega}_i$ and ${}^{i}\mathbf{v}_i$. According to (8.29), the point velocity of the origin $\mathbf{O}_{i+1}$ shown in Figure 8.8 can be written as

$$ {}^{i}\mathbf{v}_{i+1} = {}^{i}\mathbf{v}_i + {}^{i}\boldsymbol{\omega}_i \times {}^{i}\mathbf{p}_{i+1} \tag{8.36}$$

This equation can be described in $\{i+1\}$ by multiplying ${}^{i+1}\mathbf{R}_i$ on both sides of the above equation, which gives

$$ {}^{i+1}\mathbf{v}_{i+1} = {}^{i+1}\mathbf{R}_i({}^{i}\mathbf{v}_i + {}^{i}\boldsymbol{\omega}_i \times {}^{i}\mathbf{p}_{i+1}) \tag{8.37}$$

Equation (8.37) is the formula for the *propagation of point velocities.*

Example 8.6 (Velocity of 2R via propagation): A 2R robot is shown in Figure 8.9. Use velocity propagation to find the point velocity of O_3 on the end-effector. Assume $l_1 = l_2 = 1$ m.

Solution:

According to the previous example on the DK problem of this robot, we have the following transform matrices.

$$ {}^{0}\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; {}^{1}\mathbf{T}_2 = \begin{bmatrix} c_2 & -s_2 & 0 & 1 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; {}^{2}\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

According to the formulas (8.35) and (8.37) and the above transformation matrices, we have velocities in Table 8.2. Note that the point velocity and angular velocity of the end-effector obtained are measured in the third frame. To get the velocities measured in the ground frame, premultiply the rotation matrix on them, i.e.,

$$ {}^{0}\boldsymbol{\omega}_3 = {}^{0}\mathbf{R}_3\, {}^{3}\boldsymbol{\omega}_3 = \begin{bmatrix} 0 \\ 0 \\ \dot\theta_1 + \dot\theta_2 \end{bmatrix}, \quad {}^{0}\mathbf{v}_3 = {}^{0}\mathbf{R}_3\, {}^{3}\mathbf{v}_3 = \begin{bmatrix} -s_1\dot\theta_1 - s_{12}(\dot\theta_1+\dot\theta_2) \\ c_1\dot\theta_1 + c_{12}(\dot\theta_1+\dot\theta_2) \\ 0 \end{bmatrix} \tag{8.38}$$

One can readily compare the obtained results to the solution (8.4) obtained by the Jacobian in Example 8.1.

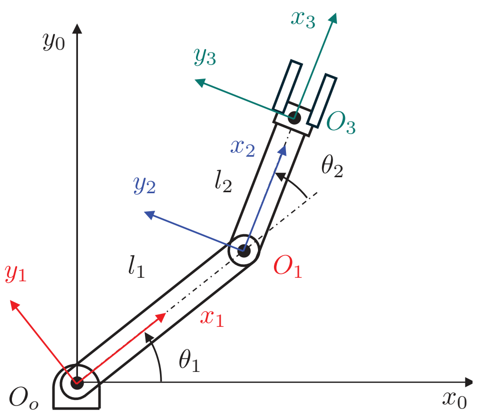


FIGURE 8.9
2R robot, with joint angles θ.

TABLE 8.2
Velocity propagation of the 2R robot

i	${}^{i}\boldsymbol{\omega}_i$	${}^{i}\mathbf{v}_i$
0	$\mathbf{0}$	$\mathbf{0}$
1	$\begin{bmatrix}0 & 0 & \dot{\theta}_1\end{bmatrix}^T$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$
2	$\begin{bmatrix}0 & 0 & \dot{\theta}_1+\dot{\theta}_2\end{bmatrix}^T$	$\begin{bmatrix}s_2\dot{\theta}_1 & c_2\dot{\theta}_1 & 0\end{bmatrix}^T$
3	$\begin{bmatrix}0 & 0 & \dot{\theta}_1+\dot{\theta}_2\end{bmatrix}^T$	$\begin{bmatrix}s_2\dot{\theta}_1 & c_2\dot{\theta}_1+(\dot{\theta}_1+\dot{\theta}_2) & 0\end{bmatrix}^T$

The formulas of velocity propagation (8.35) and (8.37) can utilise the ground frame as the measuring frame. Premutiplying ${}^{0}\mathbf{R}_{i+1}$ on (8.35) and (8.37) yields

$$ {}^{0}\boldsymbol{\omega}_{i+1} = {}^{0}\boldsymbol{\omega}_i + \dot{\theta}_{i+1}\,{}^{0}\mathbf{z}_{i+1} \tag{8.39}$$

$$ {}^{0}\mathbf{v}_{i+1} = {}^{0}\mathbf{v}_i + {}^{0}\boldsymbol{\omega}_i \times {}^{0}_{i}\mathbf{p}_{i+1} \tag{8.40}$$

Without ambiguity, we can readily rewrite the above equations as

$$ \boldsymbol{\omega}_{i+1} = \boldsymbol{\omega}_i + \dot{\theta}_{i+1}\,\mathbf{z}_{i+1} \tag{8.41}$$

$$ \mathbf{v}_{i+1} = \mathbf{v}_i + \boldsymbol{\omega}_i \times {}^{i}\mathbf{p}_{i+1} \tag{8.42}$$

Example 8.7 (Linear velocity via propagation 2): Find the angular velocity and the point velocity of the end-effector by using velocity propagation.

Solution:

According to (8.41), we have

$$\begin{aligned}
\boldsymbol{\omega}_0 &= \mathbf{0}\\
\boldsymbol{\omega}_1 &= \boldsymbol{\omega}_0 + \dot{\theta}_1\,\mathbf{z}_1 = \dot{\theta}_1\,\mathbf{z}_1\\
\boldsymbol{\omega}_2 &= \boldsymbol{\omega}_1 + \dot{\theta}_2\,\mathbf{z}_2 = \dot{\theta}_1\,\mathbf{z}_1 + \dot{\theta}_2\,\mathbf{z}_2\\
\vdots &= \vdots\\
\boldsymbol{\omega}_6 &= \boldsymbol{\omega}_5 + \dot{\theta}_6\,\mathbf{z}_6 = \dot{\theta}_1\,\mathbf{z}_1 + \ldots + \dot{\theta}_6\,\mathbf{z}_6
\end{aligned} \tag{8.43}$$

According to (8.42), we have

$$\begin{aligned}
\mathbf{v}_p &= \boldsymbol{\omega}_6 \times {}^{6}\mathbf{p}_p + \mathbf{v}_6\\
&= \boldsymbol{\omega}_6 \times {}^{6}\mathbf{p}_p + \boldsymbol{\omega}_5 \times {}^{5}\mathbf{p}_6 + \mathbf{v}_5\\
&= \vdots\\
&= \boldsymbol{\omega}_6 \times {}^{6}\mathbf{p}_p + \boldsymbol{\omega}_5 \times {}^{5}\mathbf{p}_6 \ldots + \boldsymbol{\omega}_1 \times {}_{1}\mathbf{p}_2
\end{aligned} \tag{8.44}$$

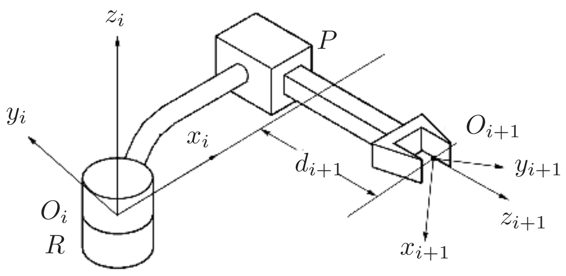


FIGURE 8.10
Velocity propagation through a prismatic joint.

Substituting (8.43) into (8.44) yields

$$\begin{aligned}\mathbf{v}_p &= \left(\dot\theta_1\,\mathbf{z}_1 + \ldots + \dot\theta_6\,\mathbf{z}_6\right) \times {}^6\mathbf{p}_p + \left(\dot\theta_1\;\mathbf{z}_1 + \ldots + \dot\theta_5\;\mathbf{z}_5\right) \times {}^5\mathbf{p}_6 \\ &\quad \ldots + \dot\theta_1\,\mathbf{z}_1 \times {}^1\mathbf{p}_2 \\ &= \mathbf{z}_1 \times \left({}^1\mathbf{p}_2 + {}^2\mathbf{p}_3 + \ldots + {}^6\mathbf{p}_p\right)\dot\theta_1 + \mathbf{z}_2 \times ({}^2\mathbf{p}_3 + \ldots + {}^6\mathbf{p}_p)\dot\theta_2 \\ &\quad \ldots + \mathbf{z}_6 \times {}^6\mathbf{p}_p\dot\theta_6 \\ &= \mathbf{z}_1 \times {}^1\mathbf{p}_p\dot\theta_1 + \mathbf{z}_2 \times {}^2\mathbf{p}_p\dot\theta_2 + \ldots + \mathbf{z}_6 \times {}^6\mathbf{p}_p\dot\theta_6\end{aligned} \tag{8.45}$$

The results shown in (8.43) and (8.45) can be written in a matrix form:

$$\begin{bmatrix}\boldsymbol{\omega}_6 \\ \mathbf{v}_p\end{bmatrix} = \begin{bmatrix}\mathbf{z}_1 & \mathbf{z}_2 & \ldots & \mathbf{z}_6 \\ \mathbf{z}_1 \times {}^1\mathbf{p}_p & \mathbf{z}_2 \times {}^2\mathbf{p}_p & \ldots & \mathbf{z}_6 \times {}^6\mathbf{p}_p\end{bmatrix}\begin{bmatrix}\dot\theta_1 \\ \vdots \\ \dot\theta_6\end{bmatrix} \tag{8.46}$$

The Jacobian can be extracted from (8.46) as

$$\mathbf{J} = \begin{bmatrix}\mathbf{z}_1 & \mathbf{z}_2 & \ldots & \mathbf{z}_6 \\ {}^p\mathbf{p}_1 \times \mathbf{z}_1 & {}^p\mathbf{p}_2 \times \mathbf{z}_2 & \ldots & {}^p\mathbf{p}_6 \times \mathbf{z}_6\end{bmatrix} \tag{8.47}$$

where each column is the *Plücker coordinates* of the corresponding joint axis with Point P as the reference. The Plücker coordinates are used for uniquely defining a line in Euclidean space. The 6D Plücker coordinates consist of two 3D vectors: the first one is the direction of the line, while the second one is the moment of line. (8.46) shows that the velocity of the end-effector is actually the sum of the joint angular velocities around their own instantaneous axes. These instantaneous axes are also the *screw axes* of the joints in the *screw theory.* An in-depth discussion on the screws is out of the scope of the content here.

Example 8.8 (Velocity propagation with prismatic joints): Find the formulas for the velocity propagation through a prismatic joint, as shown in Figure 8.10.

Solution: Since the prismatic joint does not permit a relative rotation between the two connected links, the two links must have the same angular velocity. From (8.35), we have the propagation formula of angular velocity for the P joint as

$${}^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i\,{}^i\boldsymbol{\omega}_i \tag{8.48}$$

On the other hand, a linear velocity along Axis z_{i+1} is introduced by the prismatic joint. From (8.37), we have the propagation formula of point velocity for the P joint as

$$^{i+1}\mathbf{v}_{i+1} = {}^{i+1}\mathbf{R}_i \left({}^{i}\mathbf{v}_i + {}^{i}\boldsymbol{\omega}_i \times {}^{i}_{i}\mathbf{p}_{i+1} \right) + \dot{d}_{i+1}\, \mathbf{k} \tag{8.49}$$

where $\dot{d}_{i+1}$ is the joint velocity of the prismatic joint.

8.4 Statics

Statics analysis of a serial robot is the study of the mapping between the motor torque inputs and the force and torque output of the end-effector. Collectively, we can call these quantities the actuator *effort*.

8.4.1 Relation between Jacobians and Effort

We will show that the statics of a robot is related to its Jacobian matrix. Let us start with a simple example without considering the masses of the links.

Example 8.9 (Effort via free-body analysis): Find the required joint torques, τ_1 and τ_2, of the 2R robot shown in Figure 8.11 to generate the desired force output, f_x and f_y, by the end-effector.

Solution: In this example, we will conduct *free-body analysis* on this 2R robot to find the required joint torques. Isolating Link 2 as shown in Figure 8.12(a), we can readily find

$$\tau_2 = -s_{12} f_x + c_{12} f_y \tag{8.50}$$

Note that the directions of two forces are reversed because the desired force is defined as the *action force by* the end-effector while the free-body analysis of Link 2 requires the force *acting on* the end-effector.

Similarly, using the analysis on Link 1 in Figure 8.12(b), we have

$$\tau_1 = -s_1 f_x + c_1 f_y + \tau_2 = -s_1 f_x + c_1 f_y - s_{12} f_x + c_{12} f_y \tag{8.51}$$

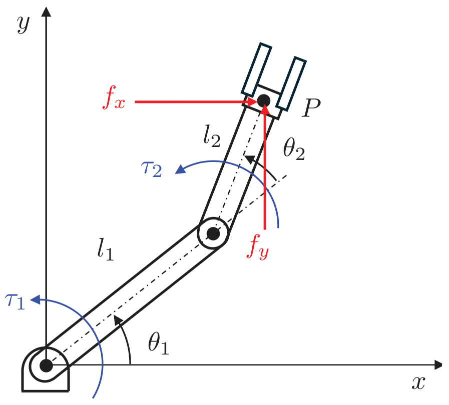


FIGURE 8.11
A 2R robot.

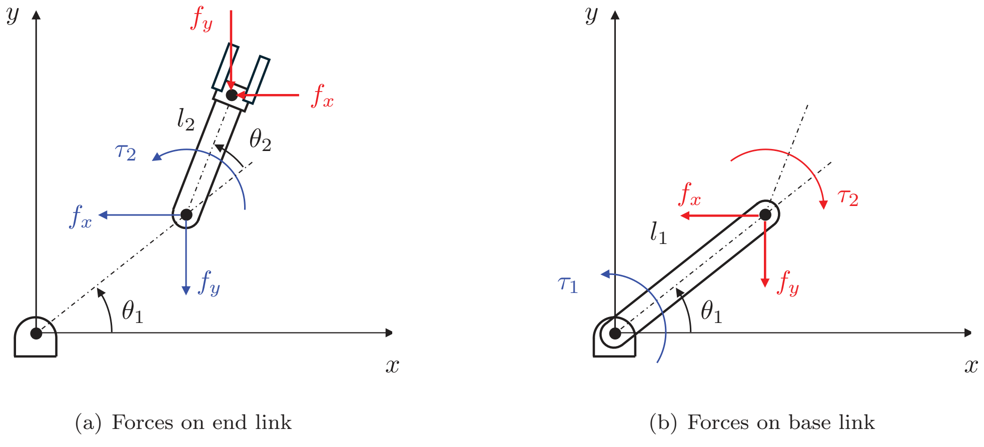


FIGURE 8.12
Free-body analysis of 2R robot.

$$= (-s_1 - s_{12})f_x + (c_1 + c_{12})f_y \tag{8.52}$$

Write (8.50) and (8.51) in the matrix form as

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \begin{bmatrix}-s_1 - s_{12} & c_1 + c_{12}\\ -s_{12} & c_{12}\end{bmatrix}\begin{bmatrix}f_x\\ f_y\end{bmatrix} \tag{8.53}$$

Recall the velocity mapping (8.4) in Example 8.1 given below:

$$\begin{bmatrix}\dot{x}_p\\ \dot{y}_p\end{bmatrix} = \begin{bmatrix}-s_1 - s_{12} & -s_{12}\\ c_1 + c_{12} & c_{12}\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} \tag{8.54}$$

From Example 8.9, we find that the relation between the matrices from Equations (8.53) and (8.4) is a simple transposition. We will show this is always true by analysing a general case.

Example 8.10 (Effort of a general 6R robot): A general 6R robot is shown in Figure 8.13. Find the joint torques according to the output effort by the end-effector.

Solution: According to the *principle of virtual work*, we have

$$\tau_1\delta\theta_1 + \ldots + \tau_6\delta\theta_6 = f_{ex}\delta d_x + f_{ex}\delta d_y + f_{ez}\delta d_z + \tau_{ex}\delta\gamma + \tau_{ey}\delta\beta + \tau_{ez}\delta\alpha \tag{8.55}$$

where $\delta\theta_i$ is the virtual displacement of θ_i, and $\delta d_x, \delta d_y, and\, \delta d_z$ are the virtual displacements of d_x, d_y and d_z, respectively. $\delta\gamma$, $\delta\beta$, and $\delta\alpha$ are the virtual angular displacements of the end-effector around the axes in the end-effector attached frame, x, y, and z, respectively. Dividing (8.55) by a time interval δt gives the relation in terms of velocities as

$$\boldsymbol{\tau}^T\,\dot{\boldsymbol{\theta}} = \mathbf{f}_e^T\,\mathbf{v} + \boldsymbol{\tau}_e^T\,\boldsymbol{\omega} = \begin{bmatrix}\mathbf{f}_e^T & \boldsymbol{\tau}_e^T\end{bmatrix}\begin{bmatrix}\mathbf{v}\\ \boldsymbol{\omega}\end{bmatrix} \tag{8.56}$$

where

$$\boldsymbol{\tau} = \begin{bmatrix}\tau_1 & \ldots & \tau_6\end{bmatrix}^T \qquad \dot{\boldsymbol{\theta}} = \begin{bmatrix}\delta\theta_1/\delta t & \ldots & \delta\theta_6/\delta t\end{bmatrix}^T \tag{8.57}$$

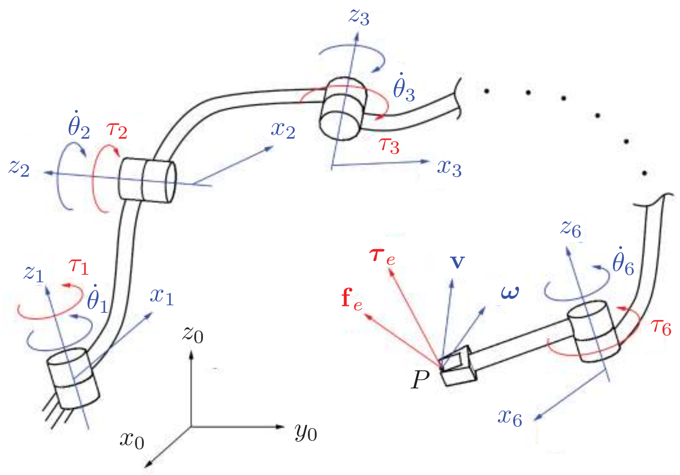


FIGURE 8.13
A general 6R robot.

$$\mathbf{f}_e = \begin{bmatrix} f_{ex} & f_{ey} & f_{ez} \end{bmatrix}^T \qquad \mathbf{v} = \begin{bmatrix} \delta d_x/\delta t & \delta d_y/\delta t & \delta d_z/\delta t \end{bmatrix} \tag{8.58}$$

$$\boldsymbol{\tau}_e = \begin{bmatrix} \tau_{ex} & \tau_{ey} & \tau_{ez} \end{bmatrix}^T \qquad \boldsymbol{\omega} = \begin{bmatrix} \delta\gamma/\delta t & \delta\beta/\delta t & \delta\alpha/\delta t \end{bmatrix}^T \tag{8.59}$$

These terms of velocities are obtained directly from the definition of velocity. The angular velocity term $\boldsymbol{\omega}$ is proven in Appendix 24.3. The kinematics relation is now known as

$$\begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v} \end{bmatrix} = \mathbf{J}\ \dot{\boldsymbol{\theta}} \tag{8.60}$$

Substituting (8.60) into (8.56) gives

$$\boldsymbol{\tau}^T\ \dot{\boldsymbol{\theta}} = \begin{bmatrix} \mathbf{f}_e^T & \boldsymbol{\tau}_e^T \end{bmatrix} \mathbf{J}\ \dot{\boldsymbol{\theta}} \tag{8.61}$$

Since the above equation is valid for an arbitrary joint velocity vector $\dot{\boldsymbol{\theta}}$, we have

$$\boldsymbol{\tau} = \mathbf{J}^T \begin{bmatrix} \mathbf{f}_e \\ \boldsymbol{\tau}_e \end{bmatrix} \tag{8.62}$$

Equation (8.62) indicates that the transpose of the Jacobian matrix of a serial robot maps the output effort by the end-effector to the input joint torques.

Example 8.11 (Static load balancing): A robotic spherical wrist is shown in Figure 8.14, where the axes of three joints are concurrent. There is a force $\mathbf{f}_e$ exerted at Point P on the end-effector, where ${}^3\mathbf{p} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$. Find the input torques to balance this load.

Solution: The problem can be solved by free-body static analysis, which can be a bit tedious. We will use the Jacobian to tackle this problem. According to the assigned DH frames, we derived previously the transform matrices as

$$ {}^0\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^1\mathbf{T}_2 = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$ {}^2\mathbf{T}_3 = \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.63}$$

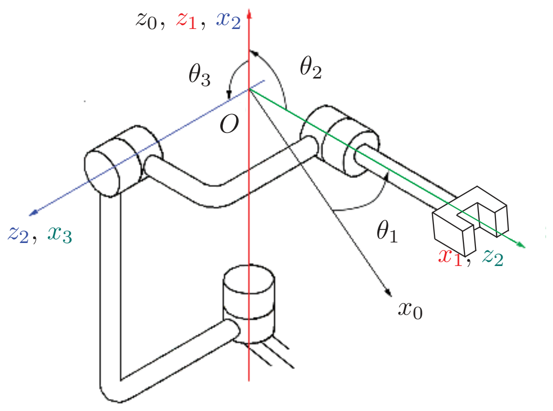


FIGURE 8.14
Spherical wrist.

Given ${}^3\mathbf{p} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, we have

$$
{}^0\mathbf{p} = {}^0\mathbf{T}_1\, {}^1\mathbf{T}_2\, {}^2\mathbf{T}_3\, {}^3\mathbf{p} = \begin{bmatrix} \cos\theta_1 \sin\theta_2 \\ \sin\theta_1 \sin\theta_2 \\ -\cos\theta_2 \\ 1 \end{bmatrix} \tag{8.64}
$$

Hence

$$
{}^0\dot{\mathbf{p}} = \begin{bmatrix} -\sin\theta_1 \sin\theta_2 \dot{\theta}_1 + \cos\theta_1 \cos\theta_2 \dot{\theta}_2 \\ \cos\theta_1 \sin\theta_2 \dot{\theta}_1 + \sin\theta_1 \cos\theta_2 \dot{\theta}_2 \\ \sin\theta_2 \dot{\theta}_2 \end{bmatrix} = \mathbf{J}\, \dot{\boldsymbol{\theta}} \tag{8.65}
$$

where $\dot{\boldsymbol{\theta}} = \begin{bmatrix} \dot{\theta}_1 & \dot{\theta}_2 & \dot{\theta}_3 \end{bmatrix}^T$, and the Jacobian is given by

$$
\mathbf{J} = \begin{bmatrix} -\sin\theta_1 \sin\theta_2 & \cos\theta_1 \cos\theta_2 & 0 \\ \cos\theta_1 \sin\theta_2 & \sin\theta_1 \cos\theta_2 & 0 \\ 0 & \sin\theta_2 & 0 \end{bmatrix} \tag{8.66}
$$

Hence, the joint torques are given by

$$
\boldsymbol{\tau}_q = \mathbf{J}^T(-\,\mathbf{f}_x) = -\begin{bmatrix} -\sin\theta_1 \sin\theta_2 & \cos\theta_1 \sin\theta_2 & 0 \\ \cos\theta_1 \cos\theta_2 & \sin\theta_1 \cos\theta_2 & \sin\theta_2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_{ex} \\ f_{ey} \\ f_{ez} \end{bmatrix} \tag{8.67}
$$

that is

$$
\begin{aligned}
\tau_1 &= \sin\theta_1 \sin\theta_2 f_{ex} - \cos\theta_1 \sin\theta_2 f_{ey} \\
\tau_2 &= -\cos\theta_1 \cos\theta_2 f_{ex} - \sin\theta_1 \cos\theta_2 f_{ey} - \sin\theta_2 f_{ez} \\
\tau_3 &= 0
\end{aligned} \tag{8.68}
$$

Note that the required torque on the third joint is always zero because the force is acting on the axis of the third joint.

The previous statics analysis does not consider the masses of the links of the robotic manipulator. In the case that the masses of the links are not negligible as compared to the loads, the Jacobian matrices for mass centres are required. Considering Example 8.10, the mass and mass centre of each link are m_i and $\mathbf{c}_i \equiv {}^0\mathbf{c}_i$, respectively, for $i = 1, \ldots, 6$. According to the principle of virtual work, (8.56) becomes

$$\boldsymbol{\tau}^T\, \dot{\boldsymbol{\theta}} + \sum m_i \mathbf{g}^T \dot{\mathbf{c}}_i = \begin{bmatrix} \mathbf{f}_e^T & \boldsymbol{\tau}_e^T \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} \tag{8.69}$$

which can be further written as

$$\boldsymbol{\tau}^T\, \dot{\boldsymbol{\theta}} + \sum m_i \mathbf{g}^T\, \mathbf{J}_{ci}\, \dot{\boldsymbol{\theta}} = \begin{bmatrix} \mathbf{f}_e^T & \boldsymbol{\tau}_e^T \end{bmatrix} \mathbf{J}\, \dot{\boldsymbol{\theta}} \tag{8.70}$$

Since the above equation is valid for an arbitrary joint velocity vector $\dot{\boldsymbol{\theta}}$, we have

$$\boldsymbol{\tau} = \mathbf{J}^T \begin{bmatrix} \mathbf{f}_e \\ \boldsymbol{\tau}_e \end{bmatrix} - \sum_{i=1}^{6} m_i\, \mathbf{J}_{ci}^T \mathbf{g} \tag{8.71}$$

which are the total torques required to overcome gravity and produce the desired output force and torque by the end-effector.

8.5 Workspace

The workspace evaluation of a robotic manipulator is fundamental in the design process and allows a designer to choose proper design parameters for the optimum workspace. A simple index can be the volume of the workspace, although the shape of a robotic workspace is often important to particular applications. Usually, the position workspace other than the orientation workspace is considered because it is intuitive and relatively simple. Further, there are different types of position workspace. Here, we will restrict our study to the *maximum reachable workspace*, which is defined as the set of all positions that can be reached by the end-effector of a robotic manipulator. Here, we introduce a theorem below.

Theorem 8.1. *The end-effector of a two-DoF planar or three-DoF spatial robotic manipulator is on the boundary of its maximum reachable position workspace if its Jacobian is singular.*

The following discussion gives some insight into this theorem instead of providing complete proof of this theorem. Consider a three-DoF spatial robot with the joint space in $\mathcal{R}^3$ and the task space in $\mathcal{R}^3$ as well. In one configuration, the 3×3 Jacobian $\mathbf{J}$ maps the 3D joint velocity into the 3D point velocity. If $\mathbf{J}$ is singular, the *nullspace* of $\mathbf{J}$ is not empty. According to the *rank-nullity theorem*, the null space of $\mathbf{J}^T$ is too not empty and is perpendicular to the *image* of $\mathbf{J}$. The union of the null space of $\mathbf{J}^T$ and the image of $\mathbf{J}$ is all possible 3D point velocity at this position in the task space. The null space of $\mathbf{J}^T$ and the image of $\mathbf{J}$ are the *subspaces* of the unreachable point velocities and the reachable point velocities, respectively. Consider a unit vector $\mathbf{n}$ inside the null space of $\mathbf{J}^T$ and define the travel distance of the end-effector in the direction of $\mathbf{n}$ as $d = \mathbf{p}^T\mathbf{n}$, where $\mathbf{p}$ is the point of interest on the end-effector. Then we have

$$\frac{\partial d}{\partial\, \boldsymbol{\theta}} = (\frac{\partial \mathbf{p}}{\partial\, \boldsymbol{\theta}})^T \mathbf{n} = \mathbf{J}^T\, \mathbf{n} \tag{8.72}$$

Since $\mathbf{n}$ inside the null space of $\mathbf{J}^T$, we have $\partial d/\partial\, \boldsymbol{\theta} = \mathbf{0}$, which means that the singular position of the end-effector is at its local extrema (minima or maxima) in the direction of $\mathbf{n}$.

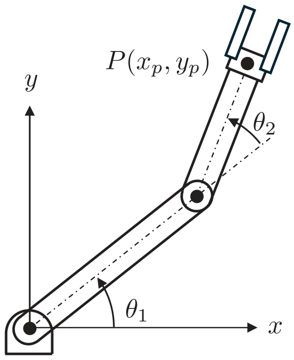


FIGURE 8.15
A 2R robot.

Hence, the end-effector is on the boundary. Further, the *image* of $\mathbf{J}$ represents the tangent plane to the boundary surface, while the null space of $\mathbf{J}^T$ is normal to the boundary.

Example 8.12 (Workspace boundary analysis): Find the boundary of the workspace of the 2R Robot shown in Figure 8.15. Assume $l_1 = l_2 = 1$ m.

Solution: From the previous example, we know the Jacobian as

$$\mathbf{J} = \begin{bmatrix} -s_1 - s_{12} & -s_{12} \\ c_1 + c_{12} & c_{12} \end{bmatrix} \tag{8.73}$$

whose singular condition is given by $\det(\mathbf{J}) = 0$. This is

$$\begin{vmatrix} -s_1 - s_{12} & -s_{12} \\ c_1 + c_{12} & c_{12} \end{vmatrix} = -c_{12}s_1 + s_{12}c_1 = s_2 = 0 \tag{8.74}$$

The solutions of the above equation are

$$\theta_2 = 0^\circ, \quad \theta_2 = 180^\circ \tag{8.75}$$

which means that the end-effector is at its boundary when the second joint is either fully folded or fully stretched out. This is true for this 2R robot. The position of the end-effector is given by

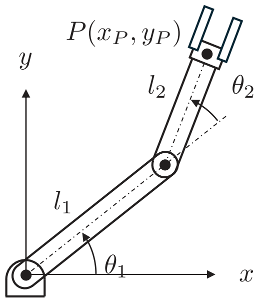
$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} c_1 + c_{12} \\ s_1 + s_{12} \end{bmatrix} \tag{8.76}$$

Substituting the singular conditions into the above position yields

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} 2c_1 \\ 2s_1 \end{bmatrix}, \quad \begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{8.77}$$

This represents the workspace boundaries of the robot, which is a circle with a radius of 2 and a point in the middle. In the singular configuration, a serial manipulator loses mobility and gains infinitely high stiffness in the direction of the load.[1]

[1] A parallel manipulator can also have parallel singularities.



$$x_P = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$y_P = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

FIGURE 8.16
RR manipulator with the forward kinematic equations.

8.6 MATLAB® Examples

There are two methods in which MATLAB can calculate Jacobians for a serial manipulator: velocity propagation as introduced in this chapter, and the use of the `Jacobian ()` function built into the Symbolic Math Toolbox. The examples in this section will cover both methods. Note that the velocity propagation method will yield velocity equations only, and require additional code to convert to Jacobian matrix form. However, this method can also be used to calculate the instantaneous numerical velocity of any defined frame within the robot iteratively, which is one of the advantages of this method if the closed-form solution to the velocity equations is unobtainable or complex.

Example M8.1 (Built-in Jacobian function): Using the `Jacobian ()` function in MATLAB, find the end-effector Jacobian matrix ${}^0J$ for the RR manipulator in Figure 8.16.

Solution: The following MATLAB code performs the above steps to solve the inverse kinematics for this problem, with the ensuing command window output.

```
syms th1 th2 l1 l2 real
x = l1*cos(th1) + l2*cos(th1+th2);
y = l1*sin(th1) + l2*sin(th1+th2);

p02 = [x; y]

J02 = jacobian(p02, [th1, th2])
```

```
p02 =

 l2*cos(th1 + th2) + l1*cos(th1)
 l2*sin(th1 + th2) + l1*sin(th1)


J02 =

[ - l2*sin(th1 + th2) - l1*sin(th1), -l2*sin(th1 + th2)]
[   l2*cos(th1 + th2) + l1*cos(th1),  l2*cos(th1 + th2)]
```

According to variable `J`, the Jacobian matrix of the end-effector measured at frame $\{0\}$ is

$$ {}^0\mathbf{J} = \begin{bmatrix} -l_2 \sin(\theta_1+\theta_2) - l_1 \sin\theta_1 & -l_2 \sin(\theta_1+\theta_2) \\ l_2 \cos(\theta_1+\theta_2) + l_1 \cos\theta_1 & l_2 \cos(\theta_1+\theta_2) \end{bmatrix} \tag{8.78}$$

Example M8.2 (Statics): For the same RR manipulator in Figure 8.16, find the joint torques required to statically hold a $-5\hat{Y}_2$ N force at the end-effector when the robot is under the pose

$$\theta_1 = \frac{\pi}{4} \qquad \theta_2 = \frac{\pi}{5}$$

with link lengths

$$l_1 = l_2 = 0.5 \text{ m}$$

Solution: The joint torque $\boldsymbol{\tau}$ is calculated by

$$\boldsymbol{\tau}_q = {}^0\mathbf{J}^T \, {}^0\mathbf{f}_x \tag{8.79}$$

where ${}^0F$ is the force applied at the end-effector, measured at the base frame $\{0\}$.

```
% Create function handle for Jacobian matrix J02
f_J02 = matlabFunction(J02,'Vars',[th1 th2 l1 l2]);

% Substitute known values for [th1, th2, l1, l2]
v_J02 = f_J02(pi/4, pi/6, 0.5, 0.5)

% Calculate static torque
F02 = [0 -5]';             % Force at end effector {2} wrt {0}
torque = v_J02'*F02
```

```
v_J02 =

   -0.8365    -0.4830
    0.4830     0.1294


torque =

   -2.4148
   -0.6470
```

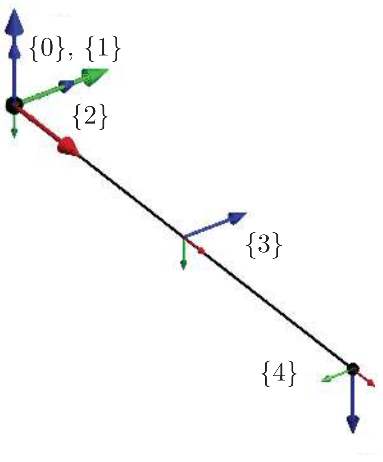
Therefore, the torques required for each actuator to maintain the static force of $-5\hat{Y}_0$ N at the end-effector is

$$\boldsymbol{\tau}_q = \begin{bmatrix} -2.4148 \\ -0.6470 \end{bmatrix} \text{ Nm}$$

Example M8.3 (Velocity propagation): Use the velocity propagation method to find the Jacobian matrix ${}^4\mathbf{J}$ of an articulated robot represented in Figure 8.17.

Solution: The following MATLAB code performs the above steps to solve the inverse kinematics for this problem, with the ensuing command window output.

```
syms th1 th2 th3 l1 l2 real % Joint and constant terms
syms dth1 dth2 dth3 real    % Joint velocity terms

% Frame definitions
T_0_1 = [
    cos(th1)     -sin(th1)     0    0;
    sin(th1)     cos(th1)      0    0;
    0            0             1    0;
    0            0             0    1];

T_1_2 = [
    cos(th2)     -sin(th2)     0    0;
    0            0             1    0;
    -sin(th2)    -cos(th2)     0    0;
    0            0             0    1];

T_2_3 = [
    cos(th3)     -sin(th3)     0    l1;
    sin(th3)     cos(th3)      0    0;
    0            0             1    0;
    0            0             0    1];
```



$$^{0}\mathbf{T}_1 = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{1}\mathbf{T}_2 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin(\theta_2) & -\cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{2}\mathbf{T}_3 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & l_1 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{3}\mathbf{T}_4 = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

FIGURE 8.17
An articulated manipulator with transformation matrices.

```

T_3_4 = [
    1    0    0    l2;
    0    0    1    0;
    0    -1   0    0;
    0    0    0    1];

% Initialise variables
w0 = [0 0 0]'  % Base frame zero angular velocity
v0 = [0 0 0]'  % Base frame zero linear velocity
Z = [0 0 1]';    % Unit vector in Z

% Propagation i = 1 (Frame {0} to {1} via revolute joint 1)
R01 = T_0_1(1:3,1:3);              % Rotation matrix of {1} measured in {0}
R10 = R01';                        % Rotation matrix of {0} measured in {1}
P01 = T_0_1(1:3,4);                % Position of {1} measured in {0}
w1 = R10*w0 + dth1*Z               % Angular vel of {1} measured in {1}
v1 = R10*(v0 + cross(w0, P01))     % Linear vel of {1} measured in {1}

% Propagation i = 2 (Frame {1} to {2} via revolute joint 2)
R12 = T_1_2(1:3,1:3);              % Rotation matrix of {2} measured in {1}
R21 = R12';
P12 = T_1_2(1:3,4);                % Position of {2} measured in {1}
w2 = R21*w1 + dth2*Z               % Angular vel of {2} measured in {2}
v2 = R21*(v1 + cross(w1, P12))     % Linear vel of {2} measured in {2}

% Propagation i = 3 (Frame {2} to {3} via revolute joint 3)
R23 = T_2_3(1:3,1:3);
R32 = R23';
P23 = T_2_3(1:3,4);
w3 = R32*w2 + dth3*Z               % Angular vel of {3} measured in {3}
v3 = R32*(v2 + cross(w2, P23))     % Linear vel of {3} measured in {3}

% Propagation i = 4 (Frame {3} to {4} via displacement only)
R34 = T_3_4(1:3,1:3);
R43 = R34';
P34 = T_3_4(1:3,4);
w4 = R43*w3                        % Angular vel of {4} measured in {4}
v4 = R43*(v3 + cross(w3, P34))     % Linear vel of {4} measured in {4}

% Find Jacobian matrix J44 (Jacobian of {4} measured in {4}
% Create function handle of v4
f_v4 = matlabFunction(v4);  % Input vars: dth1,dth2,dth3,l1,l2,th2,th3

% Eliminate variables using function handle
c = {l1,l2,th2,th3};               % Constants
J44 = simplify([f_v4(1,0,0,c{:}) f_v4(0,1,0,c{:}) f_v4(0,0,1,c{:})])
```

```
w0 =

     0
     0
     0


v0 =

     0
     0
     0


w1 =

    0
    0
 dth1


v1 =

 0
 0
 0


w2 =

 -dth1*sin(th2)
 -dth1*cos(th2)
           dth2


v2 =

 0
 0
 0


w3 =

 - dth1*cos(th2)*sin(th3) - dth1*cos(th3)*sin(th2)
   dth1*sin(th2)*sin(th3) - dth1*cos(th2)*cos(th3)
                                       dth2 + dth3


v3 =

 dth2*l1*sin(th3)
 dth2*l1*cos(th3)
 dth1*l1*cos(th2)


w4 =

 - dth1*cos(th2)*sin(th3) - dth1*cos(th3)*sin(th2)
                                     - dth2 - dth3
   dth1*sin(th2)*sin(th3) - dth1*cos(th2)*cos(th3)


v4 =

                                                             dth2*l1*sin(th3)
 - l2*(dth1*cos(th2)*cos(th3) - dth1*sin(th2)*sin(th3)) - dth1*l1*cos(th2)
                                       l2*(dth2 + dth3) + dth2*l1*cos(th3)


J44 =

[                                0,       l1*sin(th3),  0]
[ - l2*cos(th2 + th3) - l1*cos(th2),                0,  0]
[                                0, l2 + l1*cos(th3), l2]
```

According to variable `J44`, the Jacobian matrix of the end-effector measured at frame {4} is

$$
{}^4\mathbf{J} = \begin{bmatrix} 0 & l_1 \sin\theta_3 & 0 \\ -l_2\cos(\theta_2+\theta_3) - l_1\cos\theta_2 & 0 & 0 \\ 0 & l_2 + l_1\cos\theta_3 & l_2 \end{bmatrix} \tag{8.80}
$$

Example M8.4 (Jacobian matrix): Find the end-effector Jacobian matrix ${}^4\mathbf{J}$ for the articulated manipulator in MATLAB Example M8.3 using the `Jacobian ()` function in MATLAB.

Solution:

```
% Find EE frame {4} measured in base frame {0}
T_0_4 = T_0_1 * T_1_2 * T_2_3 * T_3_4;
R04 = T_0_4(1:3,1:3);                   % Rotation matrix of {4} wrt {0}
P04 = T_0_4(1:3,4);                     % Position of {4} wrt {0}
J04 = jacobian(P04,[th1 th2 th3]);      % Jacobian of EE {4} wrt {0}
R40 = R04';                             % Rotation matrix of {0} wrt {4}

% Jacobian of EE {4} wrt {4}
J44 = simplify(R40*J04)
```

```
J44 =

[                                0,      l1*sin(th3),  0]
[ - l2*cos(th2 + th3) - l1*cos(th2),                0,  0]
[                                0, l2 + l1*cos(th3), l2]
```

Note that the variable `J44` representing ${}^4\mathbf{J}$ in this example is exactly the same as the one generated in Example M8.3 as Equation (8.80), verifying that both the velocity propagation method and MATLAB `Jacobian ()` functions produce the same answers.

8.7 Conclusion

Velocity analysis of a robot is an important tool for analysing end-effector motion. Two methods were introduced for deriving the velocity of the end-effector relative to its base frame.

The *time derivative* of the analytic forward kinematic equations is the most direct method for modelling end-effector velocity. The linear velocity can be found by taking the time derivative of the end-effector position, and the angular velocity can be found by finding the skew-symmetric matrix, and extracting its x, y, and z components. This method can be quite difficult for robots with high DoF, because the analytic expressions can be very complex.

The *velocity propagation* method is an iterative method for deriving the velocity of the end-effector. Although the process is less direct, the method is ideal for robots with many DoF, or when the analytical expression for the forward kinematics is difficult to analyse.

The expression for the end-effector velocity can be represented as a Jacobian matrix J, which maps the joint velocity in the joint space to the end-effector velocity in the task space. This matrix can then be used for singularity and workspace analysis, and statics analysis by taking J^T.

8.8 Exercises

Problem 1. Derive the Jacobian for the manipulators mapping the joint velocity inputs to the point velocity of the end-effector tips for both three-link manipulators from Figure 8.18. Represent the Jacobian in

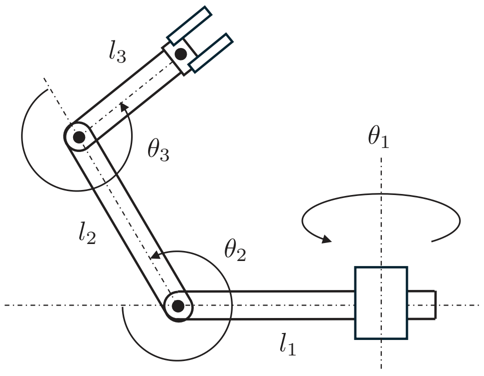


FIGURE 8.18
A 3R non-planar robot.

1. $\{0\}$ attached to the base.
2. $\{3\}$ attached to the third link.

Problem 2. A simplified model of a personnel lifter mechanism is shown in Figure 8.19 that has a working platform as the end-effector, which for safety purposes is always oriented in the direction of $\hat{z}_0$. Joints 1, 2, and 4 are revolute, with the first two coincident and the third joint prismatic, offset from joint 2 by distance l with variable extension d. You can assume a load in any direction on the platform translates to a point force in the same direction coincident at the fourth joint. Hence find equations for the torques, τ_1 , τ_2 and prismatic joint force f_3 such that the mechanism can support any force ${}^0\mathbf{f}_e = \begin{bmatrix} f_{ex} & f_{ey} & f_{ez} \end{bmatrix}^T$, applied to the platform. What are the singularities of this system, and what does it physically mean? Assume the robot is at home position in Figure 8.18, therefore an offset is needed in row 2 of the DH table. Note: The following equations for a prismatic joint can be adopted for velocity propagation:

$$ {}^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i \quad {}^{i+1}\mathbf{v}_{i+1} = {}^{i+1}\mathbf{R}_i \left({}^i\mathbf{v}_i + {}^i\boldsymbol{\omega}_i \times {}^i_i\mathbf{p}_{i+1} \right) + \dot{d}_{i+1} \, \mathbf{k} $$

Problem 3. For the robot shown in Figure 8.20.

1. Using the propagation method, find the velocity (linear and angular) of the end-effector in the tool frame, i.e., find ${}^4\mathbf{v}_4$ and ${}^4\boldsymbol{\omega}_4$.

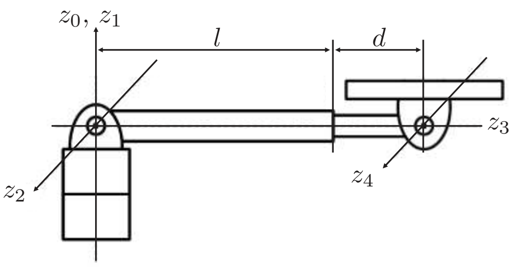


FIGURE 8.19
Schematic of a four-DoF RRPR mechanism.

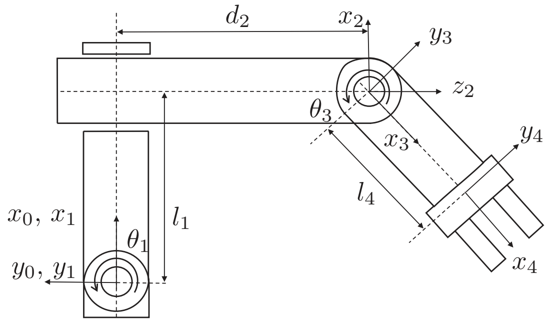


FIGURE 8.20
Planar robot.

2. Using the results obtained in (a), find the velocity (linear and angular) of the end-effector in the base frame in its simplest form, i.e. find ${}^0\mathbf{v}_4$ & ${}^0\boldsymbol{\omega}_4$
3. Find the Jacobian of the end-effector in the tool frame, i.e., ${}^4\mathbf{J}$ (include only linear velocity terms).
4. Find the Jacobian of the end-effector in the base frame, i.e., ${}^0\mathbf{J}$ (include only linear velocity terms).
5. Find the joint torques required to maintain a static force vector ${}^0\mathbf{f}_e = \begin{bmatrix} f_{ex} & f_{ey} & f_{ez} \end{bmatrix}^T$, represent the result as a matrix equation.

The transformation matrices between the base and tool frame for the robot in Figure 8.20 are as follows:

$$
{}^0\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & l_1 \\ 0 & 0 & -1 & -d_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^2\mathbf{T}_3 = \begin{bmatrix} s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ c_3 & -s_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^3\mathbf{T}_4 = \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Note: Your answers should reflect the use of these transformation matrices or otherwise correct ${}^4\mathbf{T}_0$ generated by the matrices above. Do not reassign frames and use a different set of matrices.

Problem 4. The kinematics of a 3R robotic manipulator is given by

$$
{}^0\mathbf{T}_3 = \begin{bmatrix} c_1c_{23} & -c_1s_{23} & s_1 & 2c_1 + c_1c_2 \\ s_1c_{23} & -s_1s_{23} & -c_1 & 2s_1 + s_1c_2 \\ s_{23} & c_{23} & 0 & s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

where $s_i = \sin\theta_i$, $c_i = \cos\theta_i$, $s_{ij} = \sin(\theta_i + \theta_j)$, and $c_{ij} = \cos(\theta_i + \theta_j)$. Further, θ_i for $i = 1, 2, 3$ are joint angles of this robot.

1. The position of the tip of the tool held by this robot is given by $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$ in the third frame, $\{3\}$. Find the joint angles such that this point can reach a required

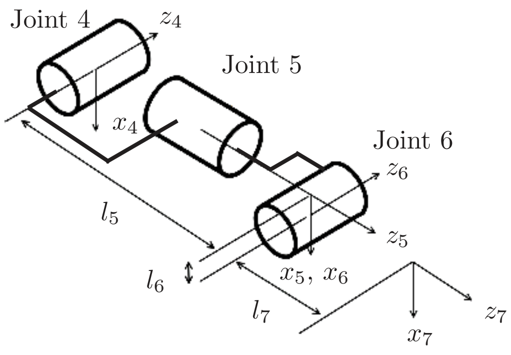


FIGURE 8.21
Wrist and end-effector of Baxter robot.

position $\begin{bmatrix}1.75 & 3.03 & 0.87\end{bmatrix}^T$ given in the ground frame, $\{0\}$. Here, $\theta_1 = 60°$ is given as the part of the answer to reduce the complexity of calculation.

2. At the above position, a force of 10N is acting at the tip of the tool by a workpiece in the negative direction of the y-axis of the ground frame. Find the required torques on each joint in order to balance the robotic arm. If you have multiple configurations of the robot for this position, choose any one of them to get your answer.

Problem 5. In a two-DoF planar robot, the position of a point P on the end-effector is given by

$$x = 10\cos\theta_1 \sin\theta_2 - 10\sin\theta_1 \cos\theta_2 + 10\cos\theta_1$$
$$y = 10\sin\theta_1 \sin\theta_2 + 10\cos\theta_1 \cos\theta_2 + 10\sin\theta_1$$

At a static configuration with $\theta_1 = 45°$ and $\theta_2 = 90°$, an external force $\mathbf{f}_e = \begin{bmatrix}f_{ex} & f_{ey}\end{bmatrix}^T = \begin{bmatrix}-5\sqrt{2} & 10\sqrt{2}\end{bmatrix}^T$ N is exerted at the point P on the robot. Calculate the required joint torques, τ_1 and τ_2, to resist this external force.

Problem 6. The orientation of one object with respect to the ground is represented by the quaternions (a, b, c, d) given by

$$\mathbf{R} = \begin{bmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{bmatrix}$$

At one instant, we detect $a = b = c = d = 0.5$ and $\dot{a} = 10$, $\dot{b} = 40$, $\dot{c} = -20$, $\dot{d} = -30$. Please find the instantaneous angular velocity of this object with respect to the ground, measured in the ground frame. Units are not required.

Problem 7. The wrist of the Baxter robot is illustrated in Figure 8.21. It consists of two revolute joints, Joints 5 and 6. Link 4 is assumed to be fixed in this question. Frames F_4 to F_7 are used to describe the configuration of such wrist and the end-effector. The overall transformation matrix ${}^4\mathbf{T}_7$ is given by

$${}^4\mathbf{T}_7 = \begin{bmatrix} c_5c_6 & -s_5 & c_5s_6 & c_5(l_6+l_7s_6) \\ s_6 & 0 & -c_6 & -l_5-l_7c_6 \\ s_5c_6 & c_5 & s_5s_6 & s_5(l_6+l_7s_6) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1. Derive the Jacobian, $\mathbf{J}$, mapping the joint velocity to the point velocity of origin of $\{7\}$.
2. Determine the wrist joint loads, τ_5 and τ_6, for the manipulator to reach static equilibrium under an external load $\begin{bmatrix} f_{ex} & f_{ey} & f_{ez} \end{bmatrix}^T$ at the origin of $\{7\}$.
3. Calculate the numerical solutions of τ_5 and τ_6, when

$$\begin{aligned}
\theta_5 &= \frac{\pi}{3} \\
\theta_6 &= \frac{\pi}{6} \\
l_5 &= 0.375 \text{ m} \\
l_6 &= 0.1 \text{ m} \\
l_7 &= 0.5 \text{ m} \\
f_x &= f_y = f_z = 1 \text{ N}
\end{aligned}$$

9 Path Planning

In Chapter 3, we investigated methods for generating trajectories between two points in both joint space and task space. In longer trajectories, we used via points to link the initial and goal states of the robot via cubic splines. While the selection of via points seemed arbitrary in the previous chapter, in application, the selection of via points is not a trivial task. In addition, it is not always guaranteed that a path exists from the robot's initial position to the goal position. In this chapter, we answer two important questions: *does a valid path exist between the robot's initial state and the goal state?* If a path exists, then *how do we move the robot from its initial state to the goal state?* The act of path planning is a critical task in robotics, which attempts to answer these questions, but this is generally not an easy task with many solutions to the problem.

In this chapter, we will introduce path planning methods and algorithms commonly used in robotics and path planning in general. Starting with the most basic path-finding algorithms, each algorithm is introduced with increasing complexity, with the final path planners introduced representing state-of-the-art algorithms that are commonly used in manufacturing robotics and mobile field robots.

9.1 Configuration Space

Most path planners are implemented in the discretised $\mathcal{C}$-space. The $\mathcal{C}$-space can be generalised to identifying the robot's current *state*, such positions in 3D space, robot arm positions or even assembly states. As long as the $\mathcal{C}$-space is non-empty and has connected states, these path planning algorithms can be implemented. Hence, the goal of a path planner is to determine a feasible path between the initial state and the goal state as represented by the $\mathcal{C}$-space. It is important that the elements in the $\mathcal{C}$-space are countable and finite in the path planning context (i.e., there are a limited number of states a robot can exist in).

In a manipulator context, a *configuration space* (or $\mathcal{C}$-space) represents the set of all possible configurations of the robot, which can be mapped *one-to-one* to the joint space. This means a single point in the configuration space maps to a single point in the joint space. Path planning is generally performed in the $\mathcal{C}$-space because, by definition, it fully describes the configuration of the robot. This is important for:

- Simplifying the path planning scheme,
- Utilising the maximum usable workspace of a robot, and
- Fully defining a robot configuration for collision avoidance in the workspace.

Consider a 2R planar manipulator such as that shown in Figure 9.1, along with sampled $\mathcal{C}$-space points and the projection of these points into the task space. In the task space, we have defined obstacles, represented by black dots. This translates to invalid points in

DOI: 10.1201/9781003614319-9

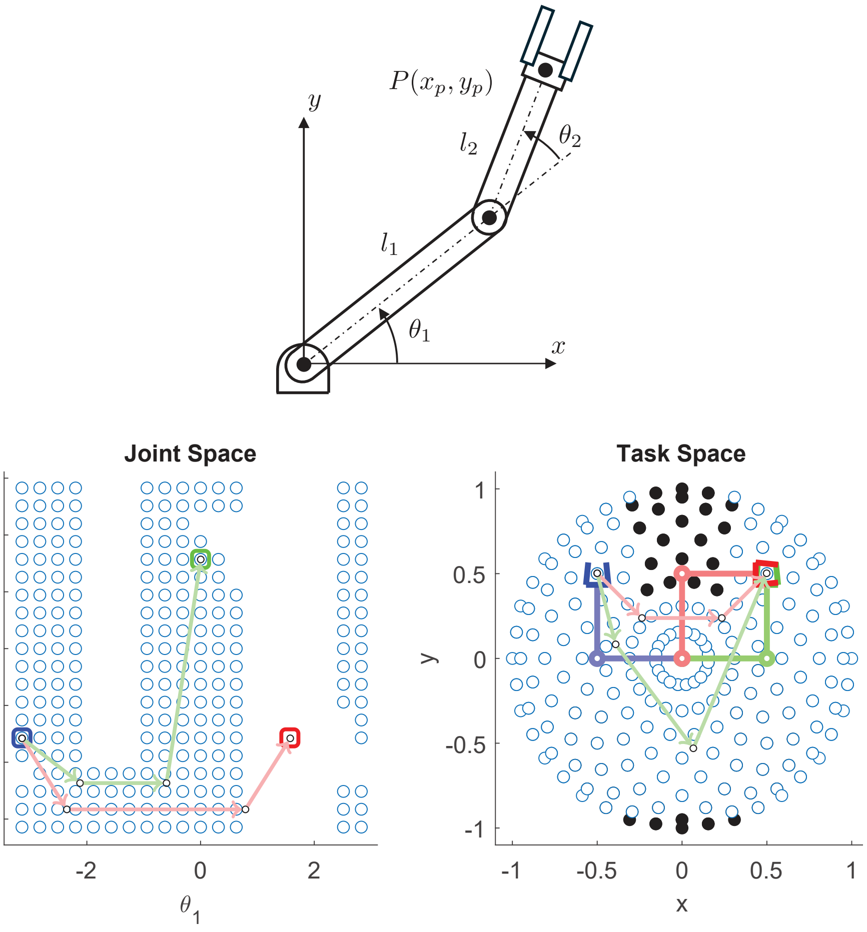


FIGURE 9.1
The valid joint space and task space with obstacles of a 2R manipulator. Three configurations are represented in both spaces, with proposed paths shown as arrows.

the $\mathcal{C}$-space, which are subsequently removed (as voids in the $\mathcal{C}$-space). Now, we propose to plan a path from the blue to the green configuration. We have two ways to represent the $\mathcal{C}$-space: in the joint space variables (θ_1 and θ_2) or the task space variables (x and y).

Suppose we use the task space variables to represent the $\mathcal{C}$-space, which is a logical step, given that obstacle definition is usually expressed in task space variables. A task space path can be easily observed, as represented by the red-arrow trajectory. However, if this trajectory is followed, then the end configuration will be represented in red, which is infeasible due to its elbow being in collision. This is a direct result of a $\mathcal{C}$-space not having a one-to-one mapping with the joint space.

Now, suppose we use the joint space variables to represent the $\mathcal{C}$-space. The goal is represented in two unique positions in this space — at the green marker, which was the original goal, and the red marker, which, although it represents the same point in the task space, is a different configuration to the green. In addition, we also observe the red marker as not being a valid configuration because it does not lie in the discretised valid region. If a proposed path, represented in green arrows, is followed, then we are guaranteed to finish at the green configuration as intended.

The reason why task space planning is not guaranteed to work for serial robots is that there may be more than one inverse kinematic solution for each point in this space. Therefore, to successfully plan a path between these two configurations, we need to apply kinematic constraints, which, depending on the manipulator involved, can be quite a complex problem. However, there may be highly valid reasons why we should perform path planning in the task space variables. These reasons usually involve path constraints, which are defined in the task space variables, such as the need to perform linear trajectories in manufacturing tasks. Therefore, we can conclude the following points.

Case for Task Space Planning

Where the trajectory of the end-effector is critical, such as executing pre-defined linear paths or task space collision avoidance, then task space planning is desirable.

Case for $\mathcal{C}$-space Planning

If we are only concerned with linking configurations A and B, and the trajectory of the end-effector does not matter, then planning in a $\mathcal{C}$-space is preferred.

For serial robots, the joint space *always* fully describes a serial robot's configuration. Hence, the $\mathcal{C}$-space of a serial robot is made up of joint space variables. Unless path planning constraints are required, for the rest of this chapter, any reference to the $\mathcal{C}$-space of a robot represents the joint space of a serial robot.

The $\mathcal{C}$-space, or any planning space, is generally a continuous space in $\mathbb{R}^n$, where n is the number of dimensions. For most path planning algorithms, we utilise discrete samples of the $\mathcal{C}$-space. These samples can be taken from a grid-based discretisation or randomly sampled.

9.1.1 Grid

In a grid-based discretisation scheme, each dimension of the $\mathcal{C}$-space is discretised uniformly, such that all states in the $\mathcal{C}$-space are represented in a grid-like structure. This is a basic

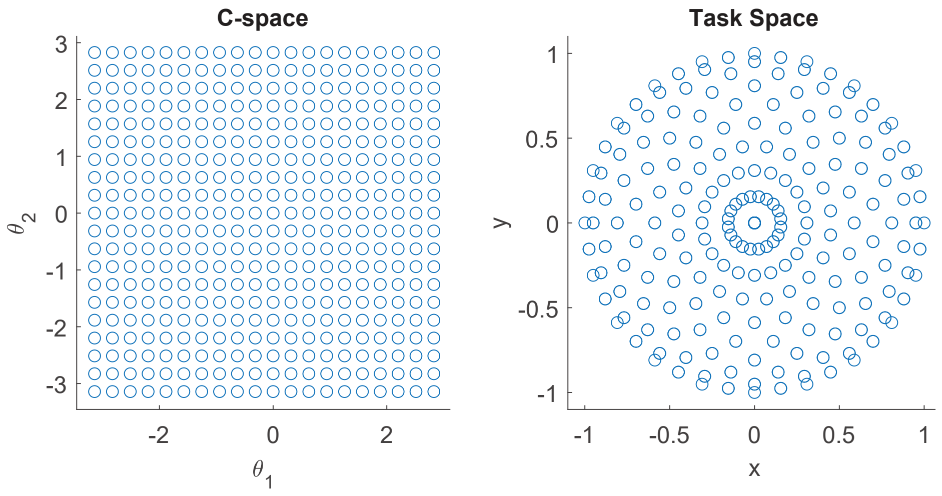


FIGURE 9.2
The discretised $\mathcal{C}$-space and equivalent points in the task space of a planar 2R manipulator.

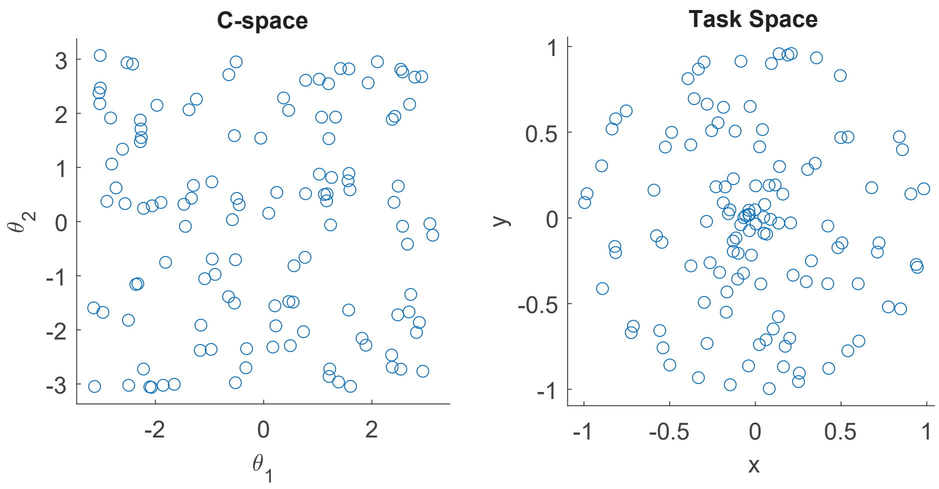


FIGURE 9.3
A $\mathcal{C}$-space that is randomly sampled, and equivalent points in the task space of a planar 2R manipulator.

scheme that is very easy to implement in a programming environment, and is the simplest way to represent the $\mathcal{C}$-space, particularly when visualising important contours of the workspace. Figure 9.2 shows the grid-based discretisation scheme applied to a planar 2R manipulator.

An important property of this type of discretisation to consider is the memory usage associated with saving each element in the $\mathcal{C}$-space. As the dimensionality of the $\mathcal{C}$-space increases, the memory usage to store this space exponentially grows. In a six-dimensional $\mathcal{C}$-space, which is quite common in robotics because it represents Cartesian and rotational co-ordinates, even a coarse discretisation of 30 elements per dimension results in $30^6 = 729$ million elements in the $\mathcal{C}$-space. If each element held a single, double-precision floating-point variable, which is 8 bytes, then we require 5.83 GB of memory to store the $\mathcal{C}$-space. While this is a non-issue on dedicated computer systems, it is problematic on embedded systems where resources are very constrained. Finally, such a large $\mathcal{C}$-space can adversely affect the speed of complete path planners, as they generally search through the entire $\mathcal{C}$-space for a feasible solution.

9.1.2 Random Sampling

Random sampling is not necessarily a discretisation method, but it describes how the points in the $\mathcal{C}$-space are progressively taken for path planning. This method is regularly used by state-of-the-art sample-based path planners, where the general assumption is that a general $\mathcal{C}$-space is relatively smooth, even with the presence of obstacles. In this case, a gridded approach may result in many discretised points that do not represent any critical features within the $\mathcal{C}$-space. This means a significant amount of memory is wasted by storing trivial points in the $\mathcal{C}$-space, that could have otherwise been identified by linear interpolation between two sampled points.

Random sampling solves this problem by progressively adding randomly sampled points in the $\mathcal{C}$-space, where the more points are sampled, the better the $\mathcal{C}$-space is represented. The user defines when to stop sampling, based on measured confidence that the $\mathcal{C}$-space and potential obstacles are fully defined by these randomly sampled points. If enough points are sampled, then collision-free paths can be planned effectively and efficiently. This method of discretisation has been shown to work well in higher-dimensional $\mathcal{C}$-spaces.

As each point is randomly sampled, the chance of sampling exactly the same point is minimal. Hence, paths will always take different points from start to goal configurations. Therefore, paths found with this method are non-deterministic in nature. In some cases, this can be viewed as a limitation where path repeatability is a priority. In addition, the chance of finding a successful path between two configurations is *probabilistically complete*, in that the probability of resolving whether a path exists or not increases as more samples are taken. This concept will be expanded upon in Section 9.5. Figure 9.3 shows the random sampling scheme applied to a planar 2R manipulator.

9.2 State Connectivity

For multi-query planners, the randomly sampled states should be connected via a *connectivity scheme*. These connections define all potential paths between states *before* they are validated for obstacles and other constraints. In other words, a connectivity scheme defines a set of rules in which states in a $\mathcal{C}$-space are considered adjacent and therefore connected by default.

9.2.1 Grids

In a grid discretisation scheme, we can consider the volume of the $\mathcal{C}$-space being divided into *cells*, such that each discretised point is located at the centre of its cell. A cell's boundaries along each dimension are called a *polytope*, which represents the "flat" sides of a cell, such as the edges of a square in two dimensions, or faces of a box in three dimensions. The $\mathcal{C}$-space can, therefore, be represented as a volume that is equidistantly filled with these cells, such that each cell's polytopes are in contact with an adjacent cell. Based on this abstract representation, several levels of connectivity can be applied to a cell's polytopes, which affect the complexity of the path planning problem.

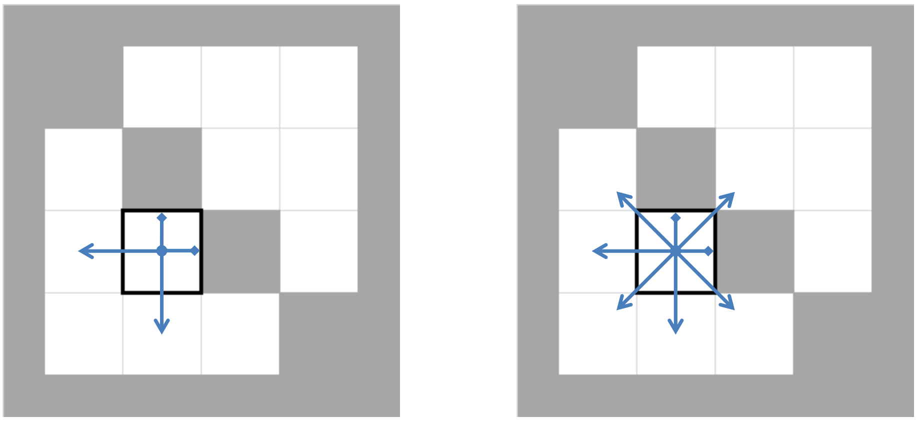
9.2.1.1 Definition

Consider an $\mathbb{R}^n$ $\mathcal{C}$-space, where n is the number of dimensions. Assume that this $\mathcal{C}$-space is grid-discretised into cells, made up of $2n$, $(n-1)$-dimensional polytopes. Then, for an n-dimensional $\mathcal{C}$-space, there is a minimum of n-possible connectivity schemes. Also, each connectivity scheme defines which adjacent cells are connected via an $(n-m)$-dimension polytope of a cell, where $\{m \in \mathbb{Z}^+ : m \leq n\}$.

A *minimum connectivity* scheme allows only $(n-1)$-dimension polytopes to be connected to an adjacent cell.

A *maximum connectivity* scheme allows all $(0, 1, .., n-1)$-dimension polytopes to be connected to an adjacent cell.

Example 9.1 (2D connectivity): Consider a two-dimensional $\mathcal{C}$-space, where $n = 2$ that has been grid-discretised with free space and obstacles defined. Define the minimum, maximum, and



(a) 4-connectivity via 1-D polytopes.

(b) 8-connectivity via 1-D and 0-D polytopes.

FIGURE 9.4
Connectivity schemes of a two-dimensional $\mathcal{C}$-space.

any other useful connectivity schemes for this $\mathcal{C}$-space.

Solution: By definition, there are two possible connectivity schemes for $n = 2$, connectivity where 1-D polytopes are connected, and connectivity where both 1-D and 0-D polytopes are connected. These connectivity schemes can be observed in Figure 9.4.

4-connectivity

This scheme defines that states in the grid should be connected, at most, at their adjacent 1-dimensional polytope or adjacent edges. It is called 4-connectivity because each cell can transition to another state in four directions, via the four edges of each cell. This is the minimum connectivity scheme.

8-connectivity

This scheme defines that states in the grid should be connected at their adjacent edges (1-D polytope) as well as at their vertices (0-D polytope), essentially allowing diagonal state transitions. It is called 8-connectivity because each cell can transition to another state in eight directions, via the four edges and vertices of each cell. This is the maximum connectivity scheme.

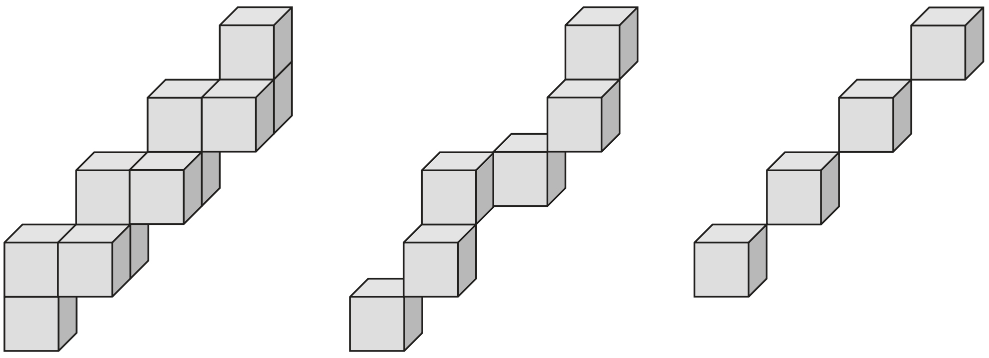
Other Connectivity Schemes

A connectivity scheme exists where cells are only connected at their vertices (diagonal state transitions only), but this is not a very practical scheme for path planning purposes.

Example 9.2 (3d connectivity): Consider a grid-discretised three-dimensional $\mathcal{C}$-space, where $n = 3$. Define the minimum, maximum, and any other fundamental connectivity schemes for this $\mathcal{C}$-space.

Solution: There are a minimum of three possible connectivity schemes for $n = 3$ as observed in Figure 9.5:

- Connected 2-D polytopes (faces) — 6-connectivity.
- Connected 2-D and 1-D polytopes (faces and edges) — 18-connectivity.
- Connected 2-D, 1-D, and 0-D polytopes (faces, edges and vertices) — 26-connectivity.



(a) 6-connectivity. (b) 18-connectivity. (c) 26-connectivity.

FIGURE 9.5
Fundamental connectivity schemes of a three-dimensional $\mathcal{C}$-space.

6-connectivity

This scheme defines that states in the grid should be connected, at most, at their adjacent two-dimensional polytope, or adjacent faces. This is the minimum connectivity scheme.

18-connectivity

This scheme defines that states in the grid should be connected at their adjacent faces (2-D polytope) as well as at their edges (1-D polytope), essentially allowing a two-variable state transition. This is a *fundamental* connectivity scheme.

26-connectivity

This scheme defines that states in the grid should be connected at their adjacent faces, edges, and vertices (0-D polytope), allowing a three-variable (diagonal) state transition. This is the maximum connectivity scheme.

Other Connectivity Schemes

A connectivity scheme exists that makes use of any other combination of m-dimensional polytopes. However, they are not practical for path planning purposes.

9.2.1.2 Obstacle Definition

One thing to consider when deciding which connectivity scheme to use, is how obstacles are defined in the $\mathcal{C}$-space. As observed in Examples 9.1 and 9.2, we see that introducing a connectivity of polytopes with dimensions less than $n-1$ results in multi-variable state changes. During these state transitions, we can pass through infinitesimally small areas of the $\mathcal{C}$-space where obstacles are ill-defined. This can be observed particularly in Figure 9.4(b). Although multi-variable (diagonal) state changes results in smoother paths, obstacle definitions for these connectivity schemes must be adjusted to avoid unintentionally passing through them.

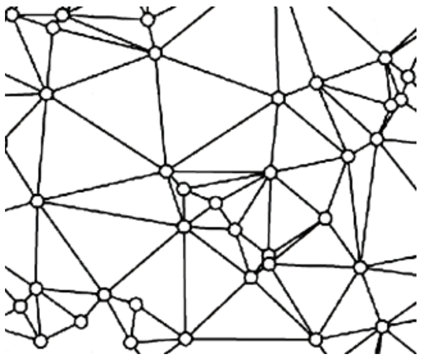


FIGURE 9.6
A 2D randomly sampled $\mathcal{C}$-space connected using Delaunay triangulation.

9.2.2 Random Samples

When the saved states of a $\mathcal{C}$-space are randomly sampled, rather than laid out in a grid, there is no unified method for determining initial connectivity between states. However, *triangulation* methods are popular, as they are generally efficient and can minimise the number of edges needed to connect all random states in the $\mathcal{C}$-space. The Delaunay triangulation method (Figure 9.6) guarantees that a state is at least connected to its nearest neighbour, and is commonly used in randomly sampled environments in field robotics. These triangulation methods can be generalised for n-dimensions, hence can work for $\mathcal{C}$-spaces representing many DoFs of a robot.

9.2.3 Connectivity Matrix

Any n-dimensional $\mathcal{C}$-space can be represented as a connected graph, where each vertex represents a state, and edges represent linear paths between each state as generated by the connectivity scheme in a grid discretisation, or triangulation or other numerical methods for randomly-sampled states. This finite graph can be represented as a *connectivity matrix* M (or adjacency matrix in linear algebra), which is a square matrix indicating whether pairs of vertices are connected or not in the $\mathcal{C}$-space *before* path validation. The connectivity matrix M is a fundamental construct in graph theory and has many uses in path planning and analysis.

Example M9.1 (Connectivity matrix): Generate a connectivity matrix for the connected graph in Figure 9.7, representing a simple $\mathcal{C}$-space. Verify using MATLAB.

Solution: The connectivity matrix M for the $\mathcal{C}$-space defined in Figure 9.7 is

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{9.1}$$

The easiest way to generate this in MATLAB is to define a list of vertex-pair connections. Inline 9.1 shows the command window output, where the vertex-pair connections matrix `E` is defined. We then convert this into a connectivity matrix `M` by using a custom function `Edge2CMatrix`, as is

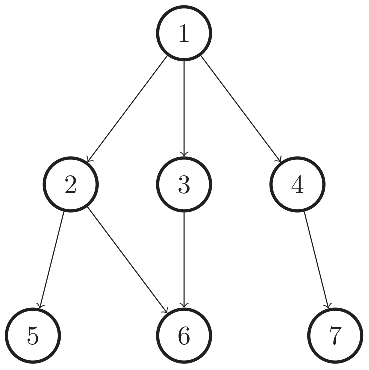


FIGURE 9.7
A graph of connected states in a simple $\mathcal{C}$-space.

shown in Inline 9.2. The initial output of `M` is in *sparse matrix* form, where only non-zero elements of a matrix are stored. This ensures that very large connectivity matrices with many zero-elements are efficiently stored in memory. To see the full matrix of `M`, we simply use MATLAB's **`full()`** function.

```
>> E = [1 2; 1 3; 1 4; 2 5; 2 6; 3 6; 4 7]

E =

     1     2
     1     3
     1     4
     2     5
     2     6
     3     6
     4     7

>> M = Edge2CMatrix(E)

M =

   (2,1)        1
   (3,1)        1
   (4,1)        1
   (1,2)        1
   (5,2)        1
   (6,2)        1
   (1,3)        1
   (6,3)        1
   (1,4)        1
   (7,4)        1
   (2,5)        1
   (2,6)        1
   (3,6)        1
   (4,7)        1

>> full(M)

ans =

     0     1     1     1     0     0     0
     1     0     0     0     1     1     0
     1     0     0     0     0     1     0
     1     0     0     0     0     0     1
     0     1     0     0     0     0     0
     0     1     1     0     0     0     0
     0     0     0     1     0     0     0
```

```
function M = Edge2CMatrix(edges)
%EDGE2CMATRIX Convert a list of connections to a sparse matrix

v1 = edges(:,1);
v2 = edges(:,2);
```

```
if size(edges,2) == 3
    cost = edges(:,3);
else
    cost = ones(size(v1));
end

% Define the sparse matrix.
% Entries are flipped to ensure connections are bi-directional.
M = sparse([v1;v2], [v2;v1], [cost;cost]);

end
```

As seen in Inline 9.1, we obtain the connectivity matrix M as shown in Equation (9.1).

9.3 Planning Completeness

There are two types of path planning algorithms that will be mentioned in this review: *complete* and *sample-based* algorithms. The key difference between the two algorithms is that the former will always return the same result, given the exact same $\mathcal{C}$-space and boundary conditions, whereas the latter result may differ slightly in different runs. Early classical path planning algorithms go for complete approaches; the entire $\mathcal{C}$-space is analysed, and the best result is always returned. However, where the $\mathcal{C}$-space is large, complex or multi-dimensional, handling such a large amount of data is infeasible and results in long running times and heavy memory usage. More recent sample-based planners accept a weaker notion of completeness; the path planner will return a valid solution as quickly as possible at the expense of sampling only a portion of the $\mathcal{C}$-space, but will continue to improve on the result as more samples of the $\mathcal{C}$-space are taken. With these algorithms, as time approaches infinity, the solution will converge to the best solution (one that a complete algorithm will produce). This need for a fast and valid solution is driven by the need for path-planning that is fast enough for real-time applications. This is one of the benefits of using sample-based planners, in that there is a degree of flexibility in accuracy versus time performance. The probabilistic road map (PRM) is one such sample-based algorithm that has been used successfully in robotics path planning, and has also served as the basis for further development for other sample-based algorithms, such as the rapidly exploring random trees (RRT).

9.3.1 Measuring Complexity

The following path planning algorithms are introductory discrete algorithms, general enough to be applied in any $\mathcal{C}$-space regardless of what quantifiable measure they represent. These algorithms are well-known in the computer science field and thus can be found in many textbooks. Their relative time performance is measured as order-of-time complexity $O(F(n))$, expressed as a function F of the $\mathcal{C}$-space parameters n. Typically, the lower the time complexity, the better-performing the algorithm is in terms of time for a given $\mathcal{C}$-space.

9.4 Complete Planning Algorithms

Complete algorithms will systematically work through a connected graph of nodes, such as Figure 9.7 representing states in the $\mathcal{C}$-space, until the goal is found. These algorithms typically perform an exhaustive search of the $\mathcal{C}$-space and construct a tree for each visited state. Many of these algorithms are classical graph theory searches and are well known for their simplicity.

9.4.1 First-Search Algorithms

One of the most primitive methods for searching a connected tree is to use either a *breadth-first search* (BFS) or *depth-first search* (DFS). Although they are simplistic by nature, they are worth noting as they form the basis of some more advanced, well-known planning algorithms. The difference between the two search methods is the order in which each edge is traversed.

9.4.1.1 Breadth-First Search

The breadth-first search (BFS) algorithm searches the entire state-space in a propagating wave, resulting in an exhaustive radial search to find the goal (Figure 9.8). It utilises a first-in-first-out queue Q to track and sample states, and has been successfully used in determining shortest-distance problems (shortest number of nodes or states visited to goal). However, it is inefficient in very large graphs and $\mathcal{C}$-spaces, because the order at which states are traversed is fixed by the queue structure, with minimal opportunity for search optimisation. However, in whole $\mathcal{C}$-space searches where *flood-fill* algorithms are necessary, BFS is a viable traversal method.

The time complexity of this algorithm is $O(|V|+|E|)$, where V and E are the number of vertices and edges. Because the number of vertices increases exponentially with dimensions of the $\mathcal{C}$-space, the time taken for this algorithm to search also increases exponentially with the number of dimensions.

Example 9.3 (Breadth-first search): Apply breadth-first search on the $\mathcal{C}$-space shown in Figure 9.7 to traverse all states from State 1, known as *flood-filling*.

Solution: Flood-filling is a useful technique for calculating the overall volume of the $\mathcal{C}$-space. Figure 9.8 shows the flood-fill result using BFS on the $\mathcal{C}$-space. Starting from State 1, a queue construct Q is used to keep track of sampled states. As Q behaves like a queue, new states are *pushed* to the rear, and items are *popped* from the head of Q. Hence, states are sampled in the order that they are pushed into Q, and *popping* Q will remove and sample the state at the head of the queue. This procedure repeats until all states in the $\mathcal{C}$-space have been visited.

Example M9.2 (Breadth-first search in MATLAB): Use MATLAB to apply breadth-first search the $\mathcal{C}$-space shown in Figure 9.7 to find a path from State 1 to State 6.

Solution: Inline 9.3 shows the MATLAB command window output on solving this problem. We can directly use the method used to solve Example M9.1, to provide the inputs required for the custom MATLAB function `bfs()`, defined in Inline 9.4. According to the output variable `pth`, the BFS algorithm suggests the path: State 1 $\rightarrow$ State 2 $\rightarrow$ State 6. This path is shown in Figure 9.9, where solid arrows indicate edges that have been pushed onto the queue (dotted arrows were not observed by the search algorithm).

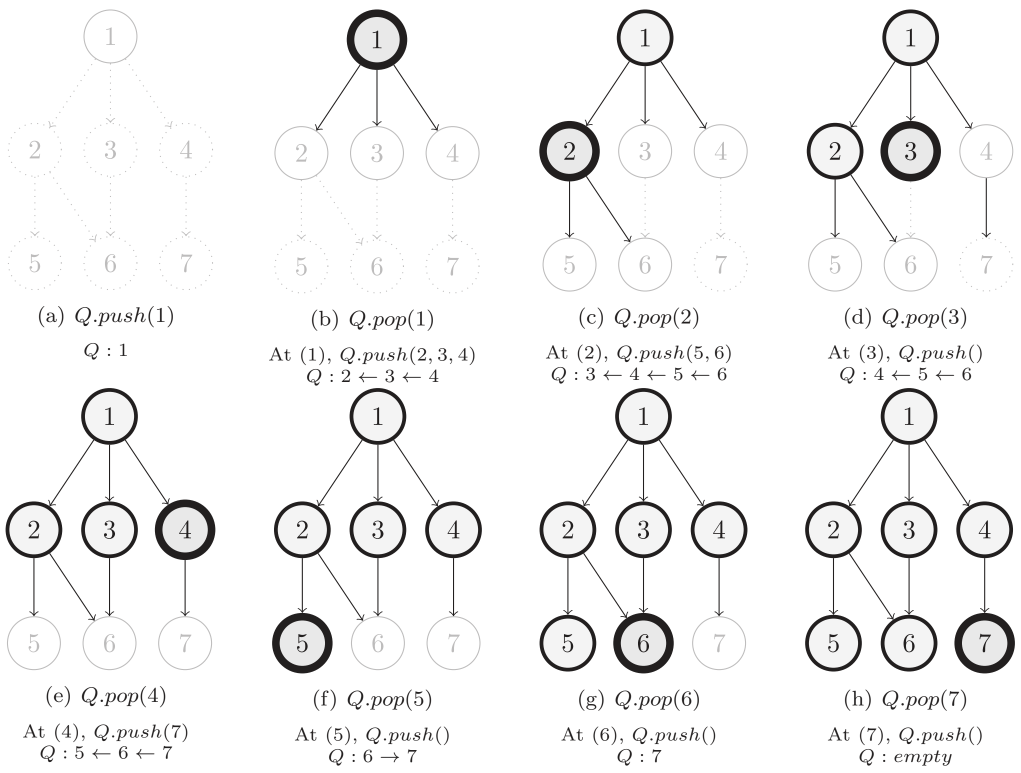


FIGURE 9.8
Breadth-first search expansion of the $\mathcal{C}$-space in Figure 9.7 with a running tally of elements in queue Q. The arrows point towards the head of the queue.

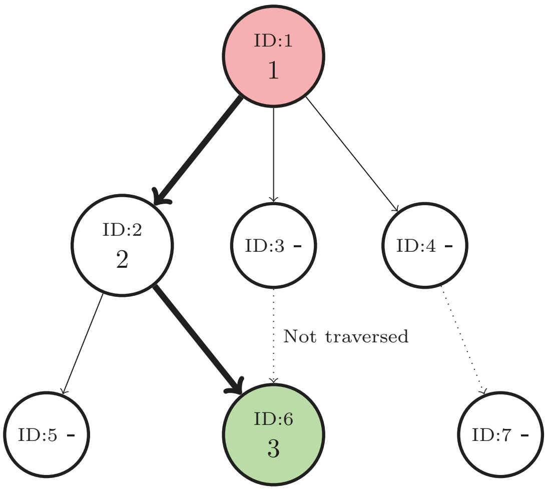


FIGURE 9.9
Breadth-first search solution with order of states visited.

```
>> E = [1 2; 1 3; 1 4; 2 5; 2 6; 3 6; 4 7];
>> M = Edge2CMatrix(E);
>> pth = bfs(M,1,6)
```

```
Starting BFS from State 1 to State 6.
  Added to states to Queue: 2 3 4
  Pushed states: 1 2 3 4
Moving to state: 2
  Added to states to Queue: 5 6 (goal found)
  Pushed states: 1 2 3 4 5 6
Moving to state: 6
Reconstructing path: (Goal) 6 -> 2 -> 1 (Init).
pth =

     1
     2
     6
```

```
function StatePath = bfs(M, InitState, GoalState)
%BFS Breadth First Search

fprintf('Starting BFS from State %i to State %i.\n', InitState, GoalState);

% Define initial conditions
PushedStates = InitState;
Queue = clsEdge.empty;          % Queue construct
CurrentState = InitState;
E = [];
GoalFound = false;

% Loop until the goal state is reached
while CurrentState ~= GoalState
    % Find all states connected to current state
    NextStateAll = find(M(:,CurrentState));
    % Remove connected states already pushed to queue previously
    NextState = setdiff(NextStateAll, PushedStates);
    fprintf('  Added to states to Queue:');
    % Add connected states to queue
    for i = NextState(:)'
        NewEdge = clsEdge(CurrentState, i, E);
        fprintf(' %i', i);
        % SHORTCUT: Terminate if goal state is added to queue
        if i == GoalState
            fprintf(' (goal found)');
            GoalFound = true;
            break
        else
            % Otherwise, push edge containing next state to queue
            Queue = Push(Queue, NewEdge);
        end
    end
    if GoalFound
        % If goal found by adding to queue, set next edge to newly added edge
        E = NewEdge;
    else
        % Otherwise, get next edge from head of queue. Pop edge off queue.
        [E, Queue] = PopQueue(Queue);
    end
    % Add pushed states to list of pushed states
    fprintf('\n  Pushed states:');
    PushedStates = Push(PushedStates, NextState);
    fprintf(' %i', PushedStates);
    % Set next state as defined by the end of the next edge
    CurrentState = E.NextState;
    fprintf('\nMoving to state: %i\n', CurrentState);
end

% Reconstruct path from goal state to initial state
EdgePath = E;
fprintf('Reconstructing path: (Goal)');
while ~isempty(E.ParentEdge)
    fprintf(' %i ->', E.NextState);
    EdgePath = Push(EdgePath, E.ParentEdge);
    E = E.ParentEdge;
end
fprintf(' %i -> %i (Init).', E.NextState, E.ParentState);

% Flip path so it goes from initial to goal state
EdgePath = flipud(EdgePath);
StatePath = [EdgePath.ParentState EdgePath(end).NextState]';

end

% Implementation of pushing an object to a list
function obj = Push(obj, item)
obj = [obj; item];
end
```

```

% Implementation of popping an object off a queue
function [item, obj] = PopQueue(obj)
item = obj(1);
obj(1) = [];
end
```

This function uses an *edge class*, `clsEdge` in Line 22 so that we can implement object referencing during path reconstruction (Inline 9.5). This allows us to back-track edges via the `ParentEdge` field through object referencing to reconstruct paths when the goal is reached.

```
classdef clsEdge < handle
    %CLSEDGE Edge objects for path planning.

    properties (SetAccess = immutable)
        ParentState     % Node ID of parent state
        NextState       % Node ID of next state
        ParentEdge      % clsEdge object which preceeded this edge
                        % Allows for back-tracking.
        Cost            % Costs associated with this path (edge)
    end

    methods
        function obj = clsEdge(s1,s2,pe,c)
            %CLSEDGE Construct an instance of this class
            obj.ParentState = s1;
            obj.NextState = s2;
            obj.ParentEdge = pe;
            if nargin < 4
                c = 1;
            end
            obj.Cost = c;
        end
    end
end
```

9.4.1.2 Depth-First Search

The depth-first search (DFS) algorithm adopts a depth-first initiative where branches are searched in-depth first until the end is reached (Figure 9.10). This utilises a first-in-last-out stack S in a programming sense to systematically build the tree. While the tendency for this algorithm is to dive in head-first quickly into a graph, this algorithm, in fact, shares the same time complexity as the breadth-first search of $O(|V|+|E|)$, where n is the number of dimensions of the $\mathcal{C}$-space. However, this algorithm will not always detect the shortest path possible between states like the BFS.

Example 9.4 (Depth-first search): Apply depth-first search on the $\mathcal{C}$-space shown in Figure 9.7 to flood-fill all states from State 1.

Solution: Figure 9.10 shows the flood-fill result using DFS on the $\mathcal{C}$-space. Starting from State 1, a stack construct S is used to keep track of sampled states. As opposed to a queue in a BFS, in a stack, new states are *pushed* to the top of the S, and states are also *popped* off from the top of S. Hence, states are sampled in the order that they were last pushed onto S, and *popping* S will remove and sample the state at the top of the stack. This procedure repeats until all states in the $\mathcal{C}$-space have been visited.

Example M9.3 (Depth-first search in MATLAB): Use MATLAB to apply depth-first search the $\mathcal{C}$-space shown in Figure 9.7 to find a path from State 1 to State 6.

Solution: Inline 9.6 shows the MATLAB command window output on solving this problem. We follow the same procedure used in Example M9.2, but using the custom MATLAB function `dfs()` instead, defined in Inline 9.7. According to the output variable `pth`, the DFS algorithm suggests the path: State 1 → State 3 → State 6. This path is shown in Figure 9.11, where solid arrows indicate edges that have been pushed onto the queue (dotted arrows were not observed by the search algorithm). Notice that in this scenario, DFS needed more iterations than BFS to find a solution.

```
>> E = [1 2; 1 3; 1 4; 2 5; 2 6; 3 6; 4 7];
>> M = Edge2CMatrix(E);
>> pth = dfs(M,1,6)
Starting DFS from State 1 to State 6.
  Added to states to Stack: 2 3 4
  Pushed states: 1 2 3 4
Moving to state: 4
  Added to states to Stack: 7
  Pushed states: 1 2 3 4 7
Moving to state: 7
  Added to states to Stack:
  Pushed states: 1 2 3 4 7
Moving to state: 3
  Added to states to Stack: 6 (goal found)
  Pushed states: 1 2 3 4 7 6
Moving to state: 6
Reconstructing path: (Goal) 6 -> 3 -> 1 (Init).
pth =

     1
     3
     6
```

```
function StatePath = dfs(M, InitState, GoalState)
%DFS Depth First Search

```

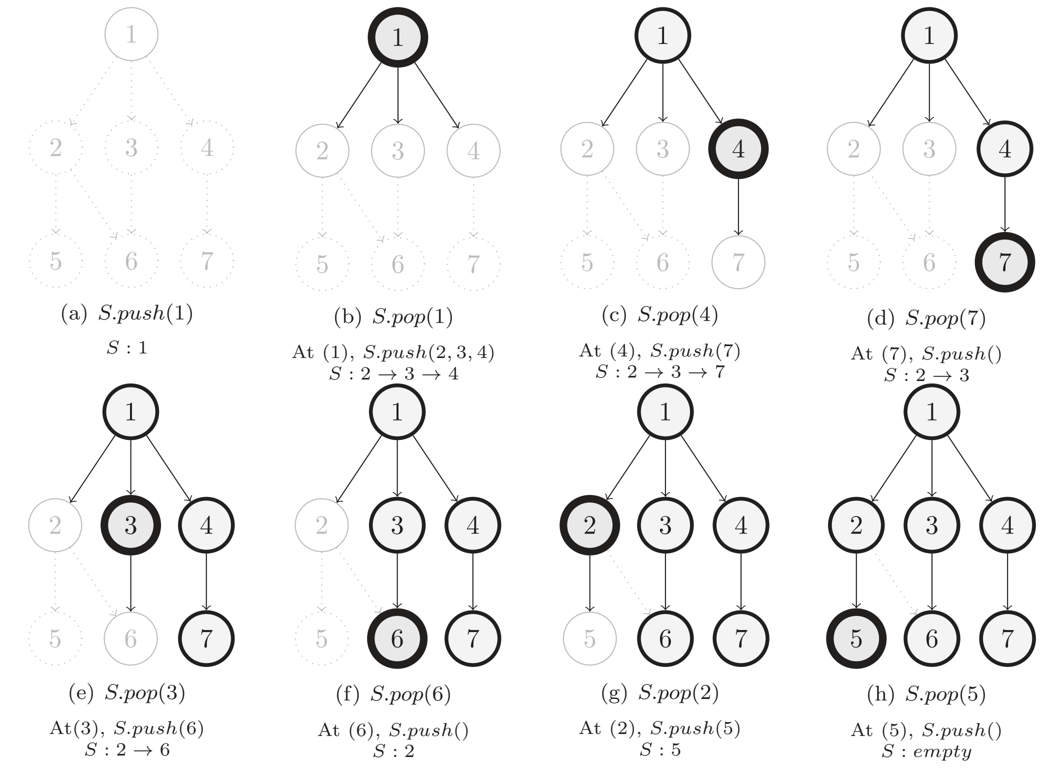


FIGURE 9.10
Depth-first search expansion of the $\mathcal{C}$-space in Figure 9.7 with a running tally of elements in stack S. The arrows point towards the top of the stack.

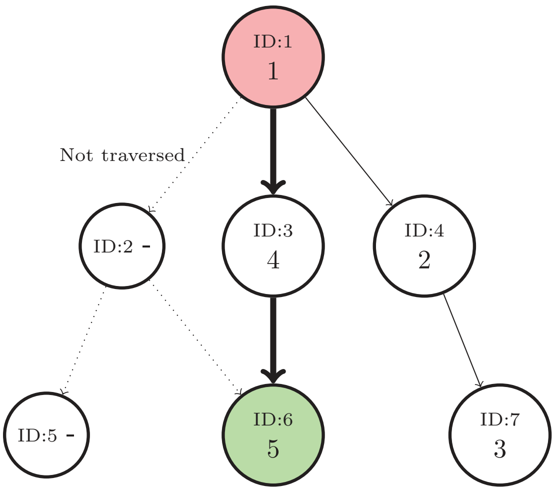


FIGURE 9.11
Depth-first search solution with order of states visited.

```
fprintf('Starting DFS from State %i to State %i.\n', InitState, GoalState);

% Define initial conditions
PushedStates = InitState;
Stack = clsEdge.empty;           % Stack construct
CurrentState = InitState;
E = [];
GoalFound = false;

% Loop until the goal state is reached
while CurrentState ~= GoalState
    % Find all states connected to current state
    NextStateAll = find(M(:,CurrentState));
    % Remove connected states already pushed to stack previously
    NextState = setdiff(NextStateAll, PushedStates);
    fprintf('  Added to states to Stack:');
    % Add connected states to stack
    for i = NextState(:)'
        NewEdge = clsEdge(CurrentState, i, E);
        fprintf(' %i', i);
        % SHORTCUT: Terminate if goal state is added to stack
        if i == GoalState
            fprintf(' (goal found)');
            GoalFound = true;
            break
        else
            % Otherwise, push edge containing next state to stack
            Stack = Push(Stack, NewEdge);
        end
    end
    if GoalFound
        % If goal found by adding to stack, set next edge to newly added edge
        E = NewEdge;
    else
        % Otherwise, get next edge from end of stack. Pop edge off stack.
        [E, Stack] = PopStack(Stack);
    end
    % Add pushed states to list of pushed states
    fprintf('\n  Pushed states:');
    PushedStates = Push(PushedStates, NextState);
    fprintf(' %i', PushedStates);
    % Set next state as defined by the end of the next edge
    CurrentState = E.NextState;
    fprintf('\nMoving to state: %i\n', CurrentState);
end

% Reconstruct path from goal state to initial state
EdgePath = E;
fprintf('Reconstructing path: (Goal)');
while ~isempty(E.ParentEdge)
    fprintf(' %i ->', E.NextState);
    EdgePath = Push(EdgePath, E.ParentEdge);
    E = E.ParentEdge;
end
fprintf(' %i -> %i (Init).', E.NextState, E.ParentState);

% Flip path so it goes from initial to goal state
EdgePath = flipud(EdgePath);
StatePath = [EdgePath.ParentState EdgePath(end).NextState]';

end

% Implementation of pushing an object to a list
function obj = Push(obj, item)
obj = [obj; item];
end

% Implementation of popping an object off a stack
function [item, obj] = PopStack(obj)
item = obj(end);
obj(end) = [];
end
```

Note that this function also uses an *edge class*, `clsEdge` in Line 22 as defined in Inline 9.5. Finally, the command window output shows how we can use MATLAB to solve this problem.

As you can expect, the way these searchers find the goal is by chance, as the order at which each edge is traversed is fixed, so there is no chance of optimisation when using these algorithms. These algorithms should only be used in smaller $\mathcal{C}$-spaces, and are generally only implemented because they are very simple to implement in any programming language.

9.4.2 Dijkstra's Algorithm

First conceived by Edsger Dijkstra in 1956, this algorithm implements a *cost to go* function for each state transition during its planning. Therefore, each path that is planned by the algorithm will have an associated overall cost and, at the end of execution, will return the path with the lowest cost.

The algorithm functions on a *best search* paradigm, where the states with the lowest path cost will be visited first, rather than simply searching the next edge in a queue or stack. This helps directionalise the search towards lowest cost paths, and ensures that the shortest path is always found when the goal is reached. As a result, the time complexity becomes $O(|V|\log(|V|) + |E|)$. However, more data is needed, as the costs of each state must be maintained.

Process

1. **Initialisation.** Starting with an initial state x_{init}, goal state x_{goal}, and a finite space S, set $x_{current} = x_{init}$ and define two lists:

    - Next state list N
    - Visited states list V

    Add x_{init} to V

2. **Add next states to N.** From $x_{current}$, add all of its connected states that are not in V, to N.

3. **Evaluate costs for each connected state.** From $x_{current}$'s cost, evaluate costs for all connected states by evaluating state transition costs to the next state h. Update any previously calculated scores for any states if they were higher.

4. **Find next state in N.** Search for the next state with the lowest cost in list N, x_{next}.
5. **Move to lowest next state.** Set $x_{current} = x_{next}$.
6. **Terminate condition.** If $x_{current} = x_{goal}$, then the goal is found. Otherwise, go to Step 2.

The resulting search propagates like a wave-front, similar to the breadth-first search, but the path costs will directionalise the search in favour of taking shorter paths. This does not necessarily mean that this will directionalise the search towards the goal. This rapid exploration *outwards* from the initial state, rather than *towards* the goal, is this algorithm's weakness.

Example M9.4 (Dijkstra's algorithm): Find the shortest path from State 1 to State 6 using Dijkstra's algorithm, given the following connectivity matrix M with path costs integrated. Use MATLAB to verify the result.

$$M = \begin{bmatrix} 0 & 5 & 1 & 2 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 6 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix} \tag{9.2}$$

Solution: Connectivity matrix M will yield the following graph with state transition costs.

Figure 9.13 shows the search result for the $\mathcal{C}$-space in Figure 9.12, for each iteration of Dijkstra's algorithm. The solution suggests the path: State 1 $\rightarrow$ State 2 $\rightarrow$ State 6 with an overall path cost of 6. We can observe that although the goal node (State 6)'s path cost was initially calculated as 7 from State 3, State 6 was not visited after calculating its cost because there were other states of lower costs to visit first. When approaching State 6 from State 2, we find that the path cost is decreased from 7 to 6.

Notice that this algorithm required 6 iterations to solve this problem which is the worst-case scenario for this algorithm; which is worse than both BFS and DFS searches. This highlights the main weakness of this search algorithm.

Inline 9.8 shows the MATLAB command window output that verifies the result obtained in Figure 9.13 for each iteration in detail. The output variable `pth` verifies the final path in Figure M9.4.

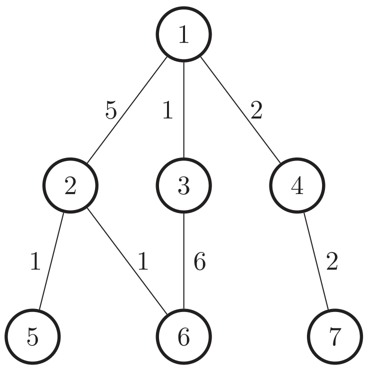


FIGURE 9.12
$\mathcal{C}$-space and state transition costs based on Equation (9.2) for Example M9.4.

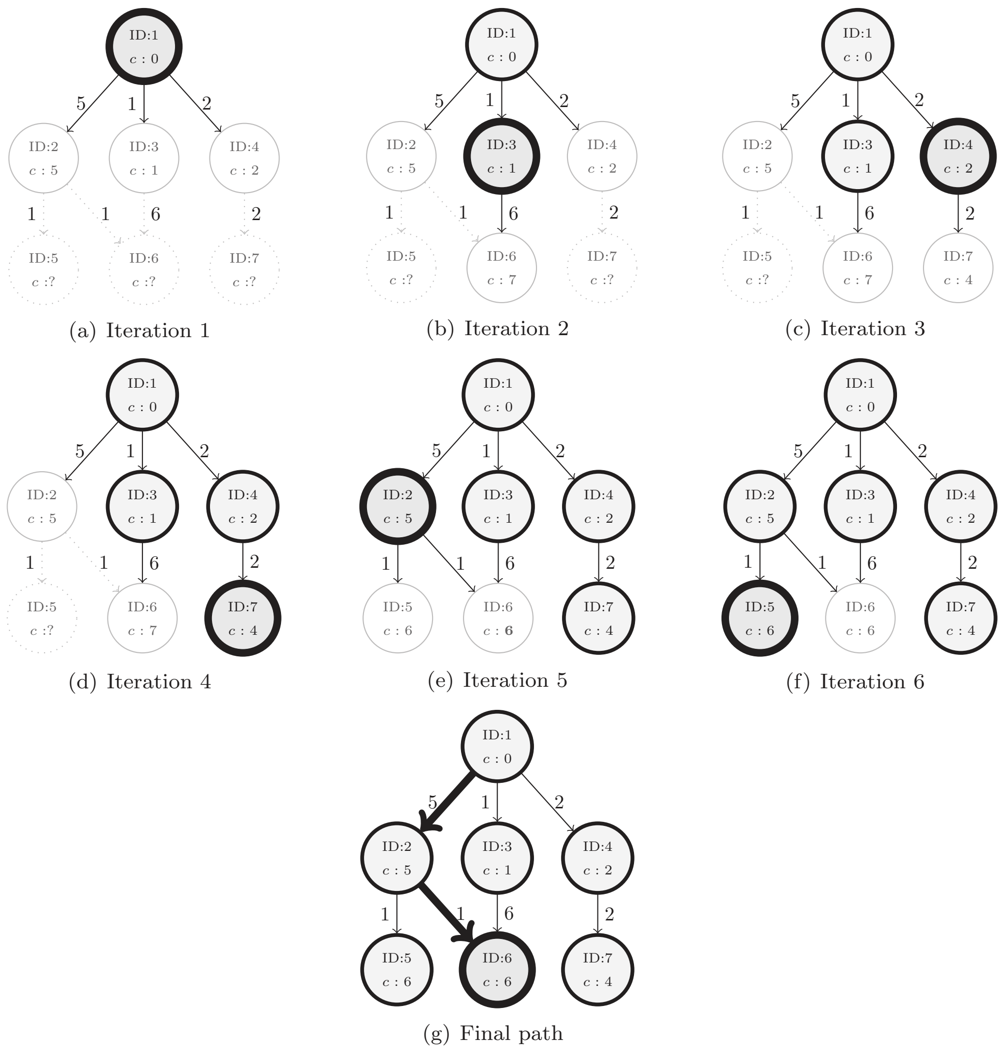


FIGURE 9.13
Dijkstra's search of the $\mathcal{C}$-space in each iteration as shown in Inline 9.8.

Inline 9.7 shows the MATLAB code to implement Dijkstra's algorithm for a given connectivity matrix, used in Inline 9.8.

```
M = [
     0     5     1     2     0     0     0
     5     0     0     0     1     1     0
     1     0     0     0     0     6     0
     2     0     0     0     0     0     2
     0     1     0     0     0     0     0
     0     1     6     0     0     0     0
     0     0     0     2     0     0     0];

>> [pth, cost] = dijkstra(M,1,6)
Starting Dijkstra search from State 1 to State 6.
[Iteration 1]
    States that have been visited: 1
    States pushed onto the list:  (c:)
    Added to states to List: 2 (c:5) 3 (c:1) 4 (c:2)
    Moving to state with lowest cost: 3 (c:1)
[Iteration 2]
    States that have been visited: 1 3
    States pushed onto the list: 2 (c:4) 5 (c:2)
    Added to states to List: 6 (c:7)
    Moving to state with lowest cost: 4 (c:2)
[Iteration 3]
    States that have been visited: 1 3 4
    States pushed onto the list: 2 (c:6) 5 (c:7)
    Added to states to List: 7 (c:4)
    Moving to state with lowest cost: 7 (c:4)
[Iteration 4]
    States that have been visited: 1 3 4 7
    States pushed onto the list: 2 (c:6) 5 (c:7)
    Added to states to List:
    Moving to state with lowest cost: 2 (c:5)
[Iteration 5]
    States that have been visited: 1 3 4 7 2
    States pushed onto the list: 6 (c:7)
    Added to states to List: 5 (c:6) 6 (c:7 -> 6)
    Moving to state with lowest cost: 5 (c:6)
[Iteration 6]
    States that have been visited: 1 3 4 7 2 5
    States pushed onto the list: 6 (c:6)
    Added to states to List:
    Moving to state with lowest cost: 6 (c:6)
Reconstructing path: (Goal) 6 (6) -> 2 (5) -> 1 (Init).
pth =

     1
     2
     6


cost =

     0
     5
     6
```

```
function [StatePath, Cost] = dijkstra(M, InitState, GoalState)
%DIJKSTRA Dijkstra's search algorithm

fprintf('Starting Dijkstra search from State %i to State %i.\n', InitState, GoalState);

% Define initial conditions
VisitedStates = InitState;       % States that have been visited
SampledStates = [];              % States added to list
List = clsEdge.empty;            % Unordered List construct
StateCost = NaN(size(M,1),1);    % Initialise all costs to NaN
StateCost(InitState) = 0;        % Set initial state score as 0
CurState = InitState;
E = [];

iteration = 0;
% Loop until the goal state is reached
while CurState ~= GoalState
    iteration = iteration + 1;
    fprintf('[Iteration %i]\n', iteration)
    fprintf('    States that have been visited:');
    fprintf(' %i', VisitedStates);
    fprintf('\n    States pushed onto the list:');
    fprintf(' %i (c:%i)', SampledStates, StateCost(SampledStates));
    fprintf('\n')
    % Find all states connected to current state
    NxtStateAll = find(M(:,CurState));
    % Remove connected states that were visited previously
    NxtState = setdiff(NxtStateAll, VisitedStates);
    NxtStateCost = StateCost(CurState) + M(NxtState,CurState);
    fprintf('    Added to states to List:');
    % Add connected states to list
    for i = 1:length(NxtState)
        % Check if next state has been sampled
        if ismember(NxtState(i),SampledStates)
            % Check if next state has been sampled but at a higher G-score
            if NxtStateCost(i) < StateCost(NxtState(i))
                % If yes, then update next state cost with the current lower G-score
                fprintf(' %i (c:%i -> %i)', NxtState(i), StateCost(NxtState(i)), ...
                    NxtStateCost(i))
                StateCost(NxtState(i)) = NxtStateCost(i);
                NewEdge = clsEdge(CurState, NxtState(i), E, ...
                    StateCost(NxtState(i)));
                List(([List.NxtState]' == NxtState)) = []; % Remove old edge from list
                List = Push(List, NewEdge);                % Push edge with lower cost
            end
        else
```

```
47              StateCost(NxtState(i)) = NxtStateCost(i);
48              % Push edge containing next state to list, along with F-score to get there
49              NewEdge = clsEdge(CurState, NxtState(i), E, StateCost(NxtState(i)));
50              fprintf(' %i (c:%i)', NxtState(i), StateCost(NxtState(i)));
51              List = Push(List, NewEdge);
52              SampledStates = Push(SampledStates, NxtState(i));
53          end
54      end
55      fprintf('\n')
56      % Get next edge whose next state F-score is lowest. Pop it from the list.
57      [E, List] = PopLowest(List);
58      % Set next state as defined by the end of the next edge
59      CurState = E.NxtState;
60      % Added current state to visited state list
61      VisitedStates = Push(VisitedStates, CurState);
62      SampledStates = setdiff(SampledStates, CurState);
63      fprintf('    Moving to state with lowest cost: %i (c:%i)\n', CurState, E.Cost);
64  end
65  
66  % Reconstruct path from goal state to initial state
67  EdgePath = E;
68  fprintf('Reconstructing path: (Goal)');
69  while ~isempty(E.ParentEdge)
70      fprintf(' %i (%i) ->', E.NxtState, E.Cost);
71      EdgePath = Push(EdgePath, E.ParentEdge);
72      E = E.ParentEdge;
73  end
74  fprintf(' %i (%i) -> %i (Init).', E.NxtState, E.Cost, E.ParentState);
75  
76  % Flip path so it goes from initial to goal state
77  EdgePath = flipud(EdgePath);
78  StatePath = [EdgePath.ParentState EdgePath(end).NxtState]';
79  
80  % Set cost as an output variable
81  Cost = StateCost(StatePath);
82  end
83  
84  % Implementation of pushing an object to a list
85  function obj = Push(obj, item)
86  obj = [obj; item];
87  end
88  
89  % Implementation of popping the lowest cost object from a list
90  function [item, obj] = PopLowest(obj)
91  [~, i] = min([obj.Cost]);
92  item = obj(i);
93  obj(i) = [];
94  end
```

Notice that function also uses an *edge class*, `clsEdge` in Line 41 as defined in Inline 9.5

9.4.3 A*

The A* search is based on Dijkstra's algorithm, which further improves performance with the introduction of *heuristics*. It has been used in a wide range of applications, but more importantly, it has applications in robot motion planning for discrete data sets.

The key feature that differentiates the A* search from a *greedy* best-first search is that it takes into account both the distance already travelled and the estimated distance to the goal to determine which state to branch from next. The result is that A* is a *best-first search* that finds a lowest-cost path from an initial state to a goal state, building a partial tree where branches are explored based on minimising the cost function

$$f(n) = g(n) + h(n) \tag{9.3}$$

where $g(n)$ is the cumulative path cost from the initial state to state n, $h(n)$ is a heuristic estimate to get from state n to the goal state, and $f(n)$ is the total score for the state which is used to determine which state to visit next. This is also called the *F-score*. In general, the heuristic function should reliably give an estimate of the distance between state n and the goal state. A common heuristic implemented in the algorithm is the *as-the-crow-flies* heuristic, which is a straight-line distance heuristic that forces A* to find the shortest path. Another common heuristic used in grid $\mathcal{C}$-spaces is called the *Manhattan* heuristic, which is simply the sum of the distances along each dimension towards the goal state.

This algorithm has a time complexity of $\mathcal{O}(b^d)$, where d is the length of the shortest path and b is the average number of adjacent states per state. If a perfect heuristic[1] is used, then the time complexity is assumed to be polynomial.

This algorithm is guaranteed to return a pass or fail result, which is a desirable feature depending on the dimensionality of the problem. Since the algorithm only terminates upon finding the endpoint, the algorithm can take a significant amount of time to complete in higher-dimensional space due to its exponential time complexity in the worst case. Furthermore, a heuristic is very difficult to calculate or model in non-homogeneous $\mathcal{C}$-spaces, such as ones that feature both angular rotations and stroke length from revolute and prismatic actuators, respectively. Due to these limitations, this algorithm is best used for smaller, less complicated spaces at lower dimensions.

Process

1. **Initialisation.** Starting with an initial state x_{init}, goal state x_{goal}, and a finite space S, set $x_{current} = x_{init}$ and define two lists:

   - Next state list N
   - Visited states list V

   Add x_{init} to V
2. **Add next states to** N**.** From $x_{current}$, add all of its connected states that are not in V, to N.
3. **Evaluate F-scores for each connected state.** From $x_{current}$'s G-score, evaluate F-scores for all connected states by evaluating state transition costs g and heuristic at the next state h. Update any previously calculated F-scores for any states if they were higher.
4. **Find next state in** N**.** Search for the next state with the lowest F-score in list N, x_{next}.
5. **Move to lowest next state.** Set $x_{current} = x_{next}$.
6. **Terminate condition.** If $x_{current} = x_{goal}$, then the goal is found. Otherwise, go to Step 2.

Example M9.5 (A-star search): Given the graph in Figure 9.14, find the shortest path between State 1 and State 6 using A* search. Verify the result in MATLAB.

Solution: Figure 9.15 shows the search result for the $\mathcal{C}$-space in Figure 9.14, for each iteration of the A* algorithm. The solution suggests the path: State 1 $\rightarrow$ State 2 $\rightarrow$ State 6 with an overall path cost of 6 as shown in Figure M9.5, similar to the result of Dijkstra's search in Example M9.4. However, the A* search was able to find the shortest path after three iterations, rather than six, which highlights the usefulness of heuristics to optimise searches in finite graphs.

Inline 9.10 shows the MATLAB command window output that verifies the result obtained in Figure 9.15 for each iteration in detail. The output variable `pth` verifies the final path in Figure M9.5. To use A* in MATLAB, we need the connectivity matrix M and heuristic information. From the $\mathcal{C}$-space graph from Figure 9.14, we can define vertex-pair connections as defined by variable `E`, and the path costs (`PathCost`). This is concatenated with `E` to then convert to a connectivity matrix

[1] Implies that the estimation of the distance between the sampled node and the goal node is exact.

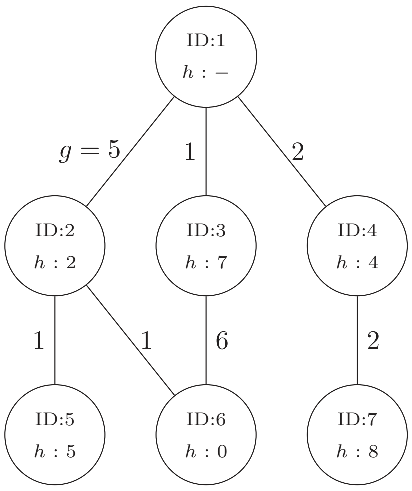


FIGURE 9.14
$\mathcal{C}$-space, state transition costs and distance heuristic h from goal State 6, for Example M9.5.

`M` via function `Edge2CMatrix()`. We also define the heuristic for each state as variable `Heuristic`. This, matrix M, initial and goal states, are all required inputs to the `astar()` function.

Inline 9.11 shows the MATLAB code to implement the A* algorithm as the function `astar()` for a given connectivity matrix, used in Inline 9.10.

```
>> PathCost = [5 1 2 1 1 6 2]

PathCost =

     5     1     2     1     1     6     2

>> E = [[1 2; 1 3; 1 4; 2 5; 2 6; 3 6; 4 7] PathCost']

E =

     1     2     5
     1     3     1
     1     4     2
     2     5     1
     2     6     1
     3     6     6
     4     7     2

>> M = Edge2CMatrix(E)

M =

   (2,1)        5
   (3,1)        1
   (4,1)        2
   (1,2)        5
   (5,2)        1
   (6,2)        1
   (1,3)        1
   (6,3)        6
   (1,4)        2
   (7,4)        2
   (2,5)        1
   (2,6)        1
   (3,6)        6
   (4,7)        2

>> Heuristic = [0 2 7 4 5 0 8]

Heuristic =

     0     2     7     4     5     0     8

>> [pth, c] = astar(M,1,6,Heuristic)
```

```
Starting A* search from State 1 to State 6.
[Iteration 1]
    States that have been visited: 1
    States pushed onto the list:  (f:)
    Added to states to List: 2 (f:7 g:5 h:2) 3 (f:8 g:1 h:7) 4 (f:6 g:2 h:4)
    Moving to state with lowest F-score: 4 (f:6)
[Iteration 2]
    States that have been visited: 1 4
    States pushed onto the list: 2 (f:3) 7 (f:8)
    Added to states to List: 7 (f:12 g:4 h:8)
    Moving to state with lowest F-score: 2 (f:7)
[Iteration 3]
    States that have been visited: 1 4 2
    States pushed onto the list: 3 (f:7) 8 (f:12)
    Added to states to List: 5 (f:11 g:6 h:5) 6 (f:6 g:6 h:0)
    Moving to state with lowest F-score: 6 (f:6)
Reconstructing path: (Goal) 6 (6) -> 2 (5) -> 1 (Init).
pth =

     1
     2
     6
```

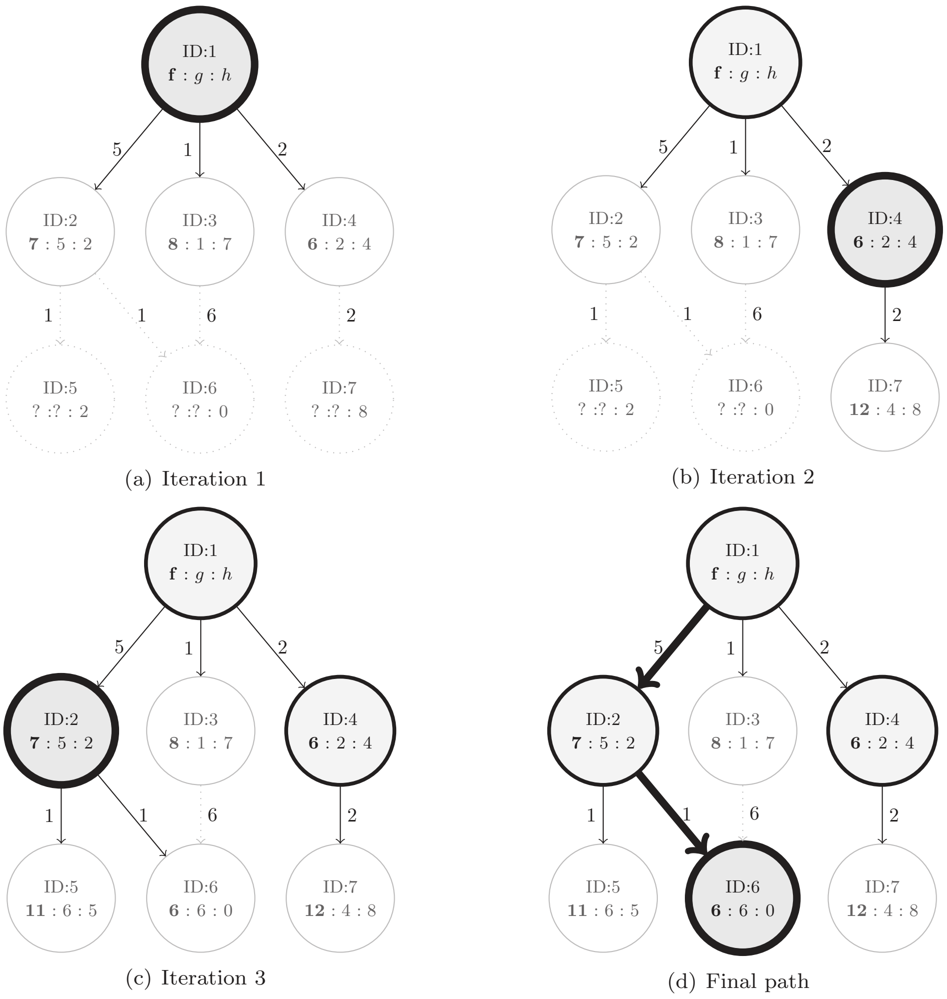


FIGURE 9.15
A* search of the $\mathcal{C}$-space in each iteration. Refer to the output in Inline 9.8 for list and score details.

```
c =

     0
     5
     6
```

```
function [StatePath, Cost] = astar(M, InitState, GoalState, StateHeuristic)
%ASTAR A* search algorithm

fprintf('Starting A* search from State %i to State %i.\n', InitState, GoalState);

% Define initial conditions
VisitedStates = InitState;      % States that have been visited
SampledStates = [];             % States added to list
List = clsEdge.empty;           % Unordered List construct
% F, G, and H scores for each state
StateScore = struct('F', 0, 'G', NaN, 'H', num2cell(StateHeuristic));
StateScore(InitState).G = 0;    % Set initial state score as 0
CurState = InitState;     % Current state
E = [];

iteration = 0;
% Loop until the goal state is reached
while CurState ~= GoalState
    iteration = iteration + 1;
    fprintf('[Iteration %i]\n', iteration)
    fprintf('    States that have been visited:');
    fprintf(' %i', VisitedStates);
    fprintf('\n    States pushed onto the list:');
    fprintf(' %i (f:%i)', SampledStates, [StateScore(SampledStates).F]);
    fprintf('\n')
    % Find all states connected to current state
    NxtStateAll = find(M(:,CurState));
    % Remove connected states that were visited previously
    NxtState = setdiff(NxtStateAll, VisitedStates);
    NxtStateG = StateScore(CurState).G + M(NxtState,CurState);
    fprintf('    Added to states to List:');
    % Add connected states to list
    for i = 1:length(NxtState)
        % Check if next state has been sampled
        if ismember(NxtState(i),SampledStates)
            % Check if next state has been sampled but at a higher G-score
            if NxtStateG(i) < StateScore(NxtState(i)).G
                % If yes, then update next state cost with the current lower G-score
                StateScore(NxtState(i)).G = NxtStateG(i);
                % Update next state F-score
                StateScore(NxtState(i)).F = StateScore(NxtState(i)).G + ...
                    StateScore(NxtState(i)).H;
                NewEdge = clsEdge(CurState, NxtState(i), E, ...
                    StateScore(NxtState(i)).F);
                List(([List.NxtState]' == NxtState)) = []; % Remove old edge from list
                List = Push(List, NewEdge);                % Push edge with lower cost
            end
        else
            StateScore(NxtState(i)).G = NxtStateG(i);
            % Update next state F-score
            StateScore(NxtState(i)).F = StateScore(NxtState(i)).G + ...
                StateScore(NxtState(i)).H;
            % Push edge containing next state to list, along with F-score to get there
            NewEdge = clsEdge(CurState, NxtState(i), E, StateScore(NxtState(i)).F);
            fprintf(' %i (f:%i g:%i h:%i)', NxtState(i), StateScore(NxtState(i)).F, ...
                StateScore(NxtState(i)).G, StateScore(NxtState(i)).H);
            List = Push(List, NewEdge);
            SampledStates = Push(SampledStates, NxtState(i));
        end
    end
    fprintf('\n')
    % Get next edge whose next state F-score is lowest. Pop it from the list.
    [E, List] = PopLowest(List);
    % Set next state as defined by the end of the next edge
    CurState = E.NxtState;
    % Added current state to visited state list
    VisitedStates = Push(VisitedStates, CurState);
    SampledStates = setdiff(SampledStates, CurState);
    fprintf('    Moving to state with lowest F-score: %i (f:%i)\n', CurState, E.Cost);
end

% Reconstruct path from goal state to initial state
EdgePath = E;
fprintf('Reconstructing path: (Goal)');
while ~isempty(E.ParentEdge)
    fprintf(' %i (%i) ->', E.NxtState, StateScore(E.NxtState).G);
    EdgePath = Push(EdgePath, E.ParentEdge);
    E = E.ParentEdge;
end
fprintf(' %i (%i) -> %i (Init).', E.NxtState, StateScore(E.NxtState).G, E.ParentState);

% Flip path so it goes from initial to goal state
EdgePath = flipud(EdgePath);
StatePath = [EdgePath.ParentState EdgePath(end).NxtState]';

% Set cost as an output variable
Cost = [StateScore(StatePath).G]';
end

% Implementation of pushing an object to a list
function obj = Push(obj, item)
obj = [obj; item];
end

% Implementation of popping the lowest cost object from a list
function [item, obj] = PopLowest(obj)
[~, i] = min([obj.Cost]);
item = obj(i);
obj(i) = [];
end
```

Notice that function also uses an *edge class*, `clsEdge` in Line 43 as defined in Inline 9.5

9.5 Sample-Based Planning Algorithms

As the name suggests, sample-based algorithms will take random samples from the discretised space to create a connected roadmap structured like a tree. Where a path is found in the network of branches between the initial and goal nodes, then the path is returned. Due to the randomness in the sampling, a different tree will be generated every time, which is why solutions will differ on repeated runs on the same problem. The main concept of sample-based planning is to avoid mapping the entire discretised space, but rather to take samples of this *black box* to obtain the overall characteristic of the $\mathcal{C}$-space progressively.

There are two goal paradigms when it comes to sample-based algorithms: *single* and *multiple* query algorithms. For single query algorithms, the goal is to find a path between two states as quickly as possible, disregarding the potential for better solutions if one exists. No $\mathcal{C}$-space pre-processing is performed for single-query algorithms. Multiple query algorithms assume that the same space will be sampled regularly, and so the process involves pre-processing of the $\mathcal{C}$-space to obtain better solutions over multiple queries. While this is beneficial for a static $\mathcal{C}$-space, it becomes challenging in dynamic environments, where single-query algorithms may perform better.

In the last decade, the most prominent sample-based algorithms are the PRM and the RRT. Both algorithms have seen industry use and possess a theoretical guarantee of probabilistic completeness. The PRM is a multiple-query algorithm, while RRT is considered the single-query counterpart.

9.5.1 Probabilistic Road Map (PRM)

As the number of dimensions in a map increases, the time it takes for a graph-based search to execute grows exponentially. In recent years, PRM has been the go-to method for pathfinding in robotics as it performs relatively well in higher-dimensional workspaces. The PRM algorithm is a sampling-based multiple-query path planning scheme that samples random points within a connected set of points and builds a tree based on collision-free straight-line trajectories between these sampled points. Sampling-based path planners have the advantage of speed over *complete* algorithms in that weaker notions of completeness are tolerated (resolution and probabilistic completeness). Where higher dimensions are considered (such as a manipulator with a high DOF), complete algorithms will take a long time to find a solution, due to time complexity and the large memory footprint required to visit and store every single state visited. Sampling-based planners are iteration-based, and in general, the probability of finding a solution converges to zero as the number of samples approaches infinity. As such, it was shown that PRM planners work well in solving difficult motion planning problems. This algorithm has a time complexity of $\mathcal{O}(n \log n)$, and was shown to be probabilistically complete, but not asymptotically optimal.

The PRM algorithm has seen widespread use in robotics, in both vision and actuator based planning. It is capable of handling higher dimensional configuration spaces which is important for high degree-of-freedom serial robot structures. It has also been regularly applied in a research setting, such as gait generation for climbing robots, and complex singularity avoidance for higher degree-of-freedom parallel mechanisms.

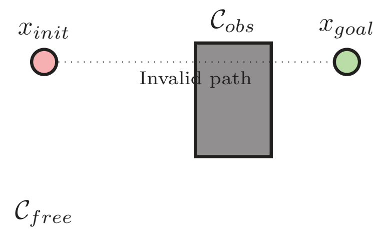
Process

This can be visualised in Figure 9.16.

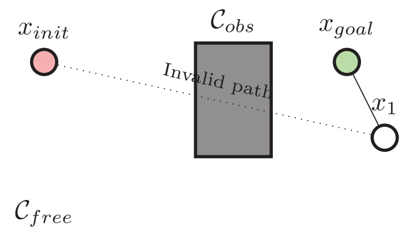
1. **Initialisation.** Start with an initial state x_{init}, goal state x_{goal}, and a finite space $\mathcal{C}_{free}$ with obstacles $\mathcal{C}_{obs}$ defined. Set $x_{new} = x_{init}$.
2. **Connect to existing states.** Find linear paths between x_{new} and all existing states in $\mathcal{C}_{free}$[2] that do not intersect $\mathcal{C}_{obs}$. Valid paths between states are marked as connected on a connectivity matrix M.
3. **Check for path between x_{init}, goal state x_{goal}.** Evaluate M for connectivity between x_{init}, goal state x_{goal}. If a path exists and other constraints are met, the algorithm stops. Otherwise, go to the next step.
4. **Sample new state.** Randomly sample a new state x_{new} anywhere in $\mathcal{C}_{free}$. Loop back to Step 2.

As more points are sampled in the discretised space, the solution will converge to a shortest-path solution. Although feasible paths are normally found in a matter of milliseconds in smooth and well-conditioned data sets, the execution of the PRM algorithm is generally allowed to continue to smooth out the path. If weighting is factored into the connectivity matrix, then the quality of the solution depends on the cost criteria of the solutions found in the connectivity/weighting matrix. For example, if paths are weighted to avoid boundaries of workspace patches, then the lowest-scoring solution will contain the path that is optimised in both path length and boundary avoidance. The simplest way to cost each linear path is to associate a cost based on distance from an obstacle for each cell within the connected map, also known as adding a heuristic function to the planner. Heuristics is regularly used to help the path planner generate better-conditioned paths in the workspace in addition to obstacle avoidance. While this has no implication on the time taken to find a valid path in the connected space, with continuing iterations beyond the path-existence termination condition, the path found will eventually converge to a balanced solution regarding path cost and path length. This will generally smooth out trajectories generated by the PRM.

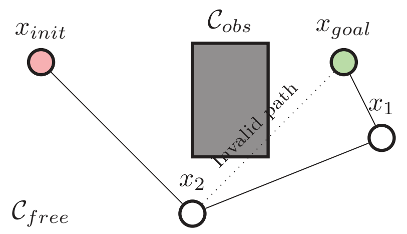
[2]Optional: within a radius r.



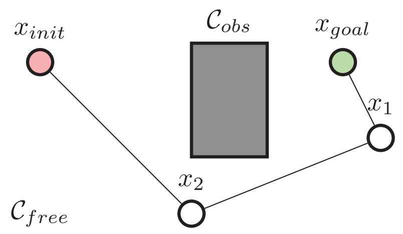
(a) Initialise and try to connect x_{init} and x_{goal}.



(b) Sample new point x_1, find connections to all other states.



(c) Sample new point x_2, find connections to all other states.



(d) Final connected states.

FIGURE 9.16
Connecting states x_{init} to x_{goal} using PRM in finite free space $\mathcal{C}_{free}$ with obstacle $\mathcal{C}_{obs}$. New randomly sampled states are added to $\mathcal{C}_{free}$ and paths that avoid $\mathcal{C}_{obs}$ are connected to existing states to find a collision-free path.

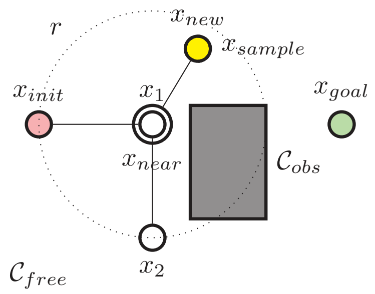
9.5.2 Rapidly Exploring Random Trees (RRT)

PRMs are considered to be efficient in planning in higher dimensional $\mathcal{C}$-spaces that are time-invariant and holonomic. However, there are many path planning problems that are *non-holonomic*, that involve kinodynamics, which by nature is used in mobile vehicles and some robotic trajectory planning. Non-holonomic problems[3] involve the constant changing of the $\mathcal{C}$-space, either through the system itself or driven by external means.

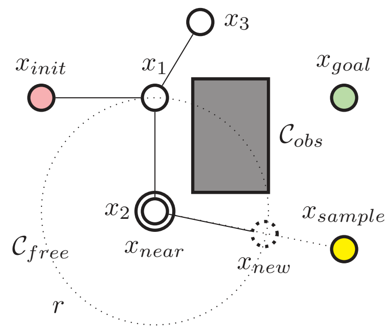
Let us consider a vehicle with a two-DoF input — acceleration and orientation as controlled by a steering wheel. The motion, or state of the vehicle, will be controlled through these inputs. However, we observe that the vehicle output state has three DoF — the $x - y$ position on a plane, and vehicle orientation ϕ. Path planning for this vehicle is a standard non-holonomic problem because the vehicle's state involves some form of integration of its input. Hence, checking for valid connections between states during path planning, considering collisions, can be computationally expensive. The rapidly-exploring random tree method aims to solve this problem by constructing state trees sequentially, such that it derives a new state from a previously known state.

Because the $\mathcal{C}$-space of a non-holonomic problem is not static, the states sampled within the tree are only valid for a given initial state. Once a new planning query is called, if the initial state is different from the previous plan, then the old sample states are no longer valid and should be discarded. This routine is called a single-query planner because a new tree is constructed every time a new path is planned. Although this may seem inefficient, the core idea of RRT is that speed is favoured over quality; that results should be generated *as quickly as possible* without any prior knowledge of the $\mathcal{C}$-space. As the $\mathcal{C}$-space is not

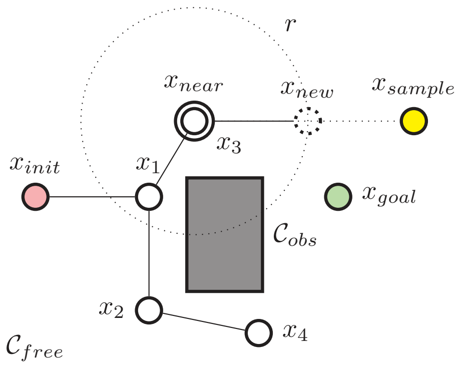
[3]Problems of the form $\dot{q} = f(q, u)$ (the classical $\mathcal{C}$-space function) are non-holonomic problems.



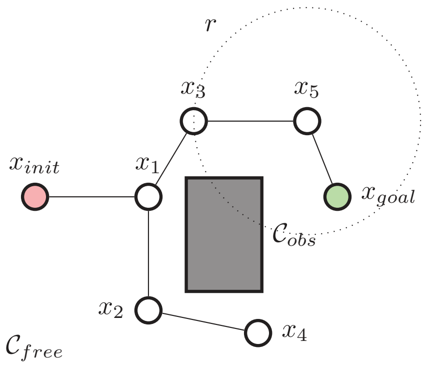
(a) Repeat for new x_{sample}. Since it is within r of x_{near}, add it directly as x_3 to T.



(b) Repeat the process for x_4.



(c) Repeat the process for x_5.



(d) Since x_5 is within r of x_{goal}, connect them. A path from x_{init} to x_{goal} is found.

FIGURE 9.17
RRT expansion from states x_{init} to x_{goal}, in finite free space $\mathcal{C}_{free}$ with obstacle $\mathcal{C}_{obs}$.

.

constructed at all, the memory footprint of RRT planning is minimal, which is ideal for embedded systems where processing power is limited.

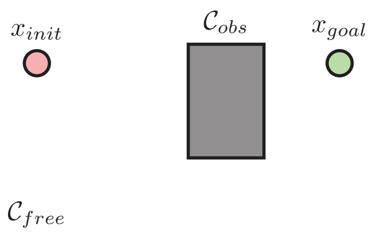
Process

The process of generating an RRT for path planning is similar to the PRM. However, there is one key difference — each new sampled state is connected to only one existing sampled state to ensure that states are connected sequentially between one another. This can be visualised in Figure 9.17.

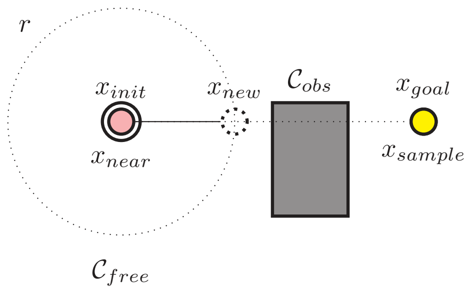
1. **Initialisation.** Start with an initial state x_{init}, goal state x_{goal} and a finite space $\mathcal{C}_{free}$ with obstacles $\mathcal{C}_{obs}$. Define a connected tree of states T, and add x_{init} to it. Set a growth radius r for the tree T. Set $x_{sample} = x_{goal}$
2. **Find x_{near}.** Find the nearest state in T that is closest to x_{sample} that is *not* x_{goal}. Set this state as x_{near}.
3. **Define x_{new}.** If distance between x_{sample} and x_{near} exceeds growth radius r, then define a linear path from x_{near} and x_{sample}, and set x_{new} to be on this path at distance r from x_{near}. Otherwise, set $x_{new} = x_{sample}$.

4. **Connect x_{new} tree T.** Verify that the linear path between x_{new} and x_{near} does not intersect $\mathcal{C}_{obs}$. If it does, discard x_{new}, and go to Step 6. If there is a valid connection, add x_{new} to the tree T via a branch from x_{near}.
5. **Connect to x_{goal}.** From x_{new}, if distance to x_{goal} is less than radius r, find a collision-free path from x_{new} to x_{goal}. If no path exists, move to the next step. If a path exists, add a branch to T, from x_{new} to x_{goal} and a path from x_{init}, goal state x_{goal} is found. Stop.
6. **Sample new state x_{sample}.** Randomly sample a new state x_{sample} anywhere in $\mathcal{C}_{free}$. Loop back to Step 2.

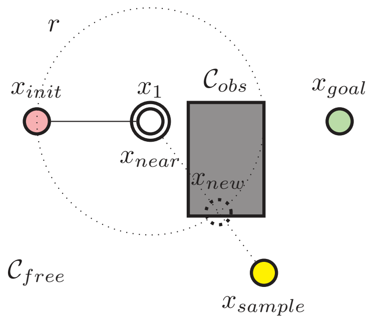
Due to RRT's ability to handle non-holonomic problems, it is very popular amongst the robotics community as the initial go-to planner for unknown $\mathcal{C}$-spaces. Its ability to handle planning almost any $\mathcal{C}$-space makes it very suitable for use in field robotics, but it also has uses in robotics research where robot prototyping can continuously change its $\mathcal{C}$-space. It is also used in some manufacturing plants that adopt human-robot collaboration, where collision objects (such as the human) are time-variant within the $\mathcal{C}$-space. However, in cases where the $\mathcal{C}$-space is predictable and time-invariant, such as plants utilising robots in trivial pick-and-place and assembly tasks, a multiple-query planner is preferred for its efficiency and deterministic plans. In such a scenario, A* or PRM in a predefined $\mathcal{C}$-space would be suitable candidates.



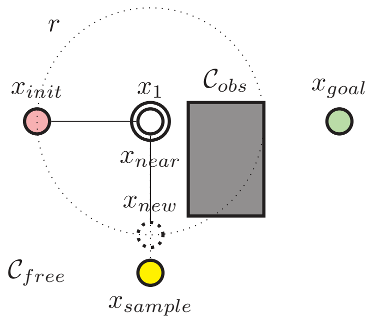
(a) Initialise with x_{init} and x_{goal} in $\mathcal{C}_{free}$ with obstacle $\mathcal{C}_{obs}$.



(b) Set x_{goal} as new sample state x_{sample}, find the nearest state x_{near} at x_{init}, and find x_{new} at radius r. Add x_{new} as x_1 to tree T.



(c) Sample new state x_{sample}, find the nearest state x_{near} at x_1, and find x_{new} at radius r. x_{new} is invalid due to collision with $\mathcal{C}_{obs}$.



(d) Repeat process for new state x_{sample}, adding it as x_2 to T.

9.6 Potential Field Planners

Potential field algorithms represent a more mathematical solution to the path planning problem. Before planning begins in the $\mathcal{C}$-space, the potential field is constructed over the entire $\mathcal{C}$-space whereby weights are applied to discourage paths in problematic areas (such as collision objects). Initially, the entire space is discretised coarsely, and the potential field is constructed. If a solution cannot be found, the space is discretised further, potential fields re-evaluated, and the algorithm is run again. The algorithm will stop when the goal state is met (success), or the maximum resolution has been attained (failure). This is a single-query planner, where the main advantage of this method is that the potential field (known as the heuristic function or guiding function) can be calculated very quickly, and adapted for specific problems. Potential field planners for robots with up to 8 DoF have been successfully implemented for the inspection of buildings but have seen limited use in modern industry with the advancements of sample-based planners.

For guaranteed convergence, the potential field must be *perfect* in that no local minima should exist except for one located at the goal state. This is typically very difficult to achieve. Hence, many applications of this planner are prone to local minima problems without assistance, such as introducing random walks. Calculating the overall path cost from gradients is also expensive to implement over spaces of high dimensions.

9.7 Conclusion

In this chapter, we defined how to set up a path planning problem for manipulator path planning. The concept of the configuration space ($\mathcal{C}$-space) is used to fully describe the pose or configuration of a robot at a given time. This space can be built using either the task space or joint space variables, depending on the proposed path planning constraints. We observed that for serial manipulators, the joint space always fully describes the configuration of the robot, hence a $\mathcal{C}$-space derived from joint space variables is a viable candidate for path planning. In addition, the $\mathcal{C}$-space can be discretised in two ways, grid or random sampling, with connectivity schemes initially defining all possible state transitions within the $\mathcal{C}$-space. Selecting an effective connectivity scheme is critically important for planning efficiency, as it directly affects the complexity of the path planning problem.

Two main types of planning algorithms were introduced to perform path planning in the $\mathcal{C}$-space: complete planners and sample-based planners. Complete planners include depth-first and breadth-first searches, which are primitive but easy to implement for simple planning problems, and Dijkstra's algorithm and A*, which are more advanced planners that implement path costings and heuristics for solving shortest path problems. Complete planners work very well in $\mathcal{C}$-spaces defined as finite graphs, but are limited to problems of lower dimensions. Sample-based planners solve the problem of higher dimensionality by taking random samples of the $\mathcal{C}$-space to formulate a problem such that a solution is probabilistically complete. Probabilistic road maps (PRM) and rapidly-exploring random trees (RRT) fall under this category, and are introduced in this chapter.

MATLAB implementations for many these planning algorithms were given in the form of path planning examples. The supplied code, which is commented in detail, supplements the brief descriptions of the process of each planning algorithm.

10

Programming

In this chapter, we introduce programming methods to model, simulate, and control serial chain manipulators. Many development environments and programming languages are used in the industry, research, and broader hobby and enthusiast environments. However, in the context of introductory robotics, we will focus on software and programming languages that are already familiar to undergraduate students: MATLAB, C++, and Python. Furthermore, we introduce the Robot Operating System (ROS), which is a commonly used middleware for programming and controlling a physical robot.

10.1 Modelling

Throughout this textbook, we used a variety of MATLAB functions and features to model various aspects of a serial robot. MATLAB provides the perfect environment for learning robotics, not only to model the kinematics and dynamics of a robot but also to provide visualisations using MATLAB's built-in graphical functions. This chapter will demonstrate how to use these tools to assist in the modelling and visualisation of robots.

10.1.1 Symbolic Functions and Handles

In Chapter 6, we utilised the Denavit-Hartenberg parameters set in tables to model the forward kinematics of a serial manipulator. Although the link lengths and offsets can be predefined by numerical values within the table, rows that contain an actuated variable, either in the q column for revolute or d column for prismatic actuators, will always remain variable. Hence, our forward kinematic modelling will be a function of symbolic variables q or d, collectively known as joint space vector $\mathbf{q}$. In MATLAB, we can define joint variables as *symbolic variables*. For example, if a RP robot of length L_1 is made up of a revolute and prismatic actuator in sequence, then we can define variables q_1 and d_2 and unknown L_1 in MATLAB by

```
syms theta_1 d_2 L_1 real
```

Note that we include the keyword `real`, which forces MATLAB to assume these variables will always contain real numbers. This means MATLAB will not calculate a complex conjugate for these variables when applying transpose operations to them. If we assume the DH parameters for this robot are

i	α_{i-1}	a_{i-1}	d_i	q_i
1	0	0	L_1	q_1
2	0	0	d_2	0

DOI: 10.1201/9781003614319-10

and apply Equation (6.3) (defined as a MATLAB function `dh2T()` in Inline 6.1), then we obtain the forward kinematic equations ${}^0\mathbf{T}_2$ in MATLAB in symbolic form.

```
>> T_0_1 = dh2T( [0, 0, L_1, theta_1] );
>> T_1_2 = dh2T( [0, 0, d_2, 0] );
>> T_0_2 = T_0_1 * T_1_2

T_0_2 =

[ cos(theta_1), -sin(theta_1), 0,         0]
[ sin(theta_1),  cos(theta_1), 0,         0]
[            0,             0, 1, L_1 + d_2]
[            0,             0, 0,         1]
```

While this clearly demonstrates MATLAB's ability to generate analytic solutions effortlessly for simple serial chains, it is not particularly useful if we want fast numerical solutions. To get a numerical answer from MATLAB's symbolic equation, one simple solution is to copy and paste the analytic expression for `T_0_2` into the command window and replace all variables with known values. However, this is inefficient if we evaluate over multiple points in the joint space, especially for long analytic expressions seen in manipulators of more than three DoF. To efficiently substitute values into `T_0_2`, we can convert it into a *function handle* using `matlabFunction()` which effectively converts it to MATLAB a function, similar to `sin()` or `cos()`.

```
>> T_0_2_function = matlabFunction(T_0_2)

T_0_2_function =

function_handle with value:

@(L_1,d_2,theta_1)reshape([cos(theta_1),sin(theta_1),0.0,0.0,-sin(theta_1),cos(theta_1
   ),0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,L_1+d_2,1.0],[4,4])
```

The function handle `T_0_2_function` is used like any other function in MATLAB, but has three inputs, L_1, d_2, and q_1 in that order. To find ${}^0\mathbf{T}_2$ at $q_1 = \frac{\pi}{4}$, $d_2 = 0.2$, with $L_1 = 0.1$ simply call `T_0_2_function()` with these known inputs.

```
>> T_0_2_function(0.1, 0.2, pi/4)

ans =

0.7071   -0.7071    0         0
0.7071    0.7071    0         0
0         0         1.0000    0.3000
0         0         0         1.0000
```

By utilising `matlabFunction()` to convert symbolic expressions into function handles, we now have a fully programmatic way to numerically model the forward kinematics of any robotic manipulator from its DH parameters.

10.2 Robot Operating System

The field of robotics has made impressive gains in recent years, both in hardware and software, and the algorithms that help these robots run with increasing levels of intelligence and autonomy. However, the increasing availability of inexpensive robotic components, each with its own hardware implementation and communications interface, is causing some significant challenges for software developers — which is to write software for a robotic system that allows seamless communication between *any* low-level hardware with higher-level control and decision-making algorithms. To add to the wish list, the flexibility of using multiple programming languages, ease of code expandability and portability, and a robust debugging platform may seem like a pipe dream. However, the introduction of the *Robot Operating System*, or *ROS*, is intended to address many of these challenges.

The official description of ROS is as follows:

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

More succinctly, ROS is known as a *middleware*, which serves to provide a standardised framework for the communications between hardware and software, to promote collaboration, expansion, and robust debugging through distributed computing, and to encourage code sharing through package management. Let us take a look at the first two benefits more closely.

Standardised Communication

This is one of the main benefits of ROS, as it provides a standardised platform for communication between *all* components within the robotic system via the ROS messaging interface, implemented on an *Ethernet/IP protocol.* This means the messaging system can function internally in a standalone computer system, utilising a network in a local host, or distributed across multiple devices on a network with messages transferred through an Ethernet cable or WiFi.

Distributed Computation

By allowing a ROS network to expand through the Ethernet infrastructure, we allow the natural expansion of a robotic system, similar to a *plug and play* implementation. This is particularly useful if we are adding new hardware, such as additional sensors, or if one were to run multiple computing systems that host their own subset of code, algorithms or hardware.

10.2.1 Operating Paradigm

Before describing the fundamental components of ROS, it is important to know the core mechanics, or *paradigm* of how ROS operates. Although ROS programs are generally written in C++ or Python, which are *object-oriented* programming languages, ROS functions primarily through an *event-driven* framework.

The flow of an **event-driven** program is determined by *events* that are raised outside the program loop. Examples of events can be sensor readings, a user clicking an element on the user interface, or even a physical switch changing states. The main program loop listens for raised events, where each loop iteration is of very low computational cost. When the program detects a raised event, it triggers a *callback function*[1] that is assigned with the event. The callback function handles the event along with any associated data, processes it, and then releases its resources back to the main program loop.

The main advantage of an event-driven paradigm is that functions are only called upon a change of state, or when new data becomes available. This means that the event handling process will always capture raised events that are non-periodic, and that polling rates do not have to be configured to match the refresh rate of incoming data. Also, if events are

[1]A callback function is also known as an *event handler*.

raised faster than event handlers can process them, they are simply placed in a finite queue to be processed when resources become available. Examples of non-periodic raised events include

- Non-periodic sensors, such as velocity sensors that raises an event for every "tick" that they are moving, but stop when stationary.
- New-data triggers, such as when a thread has finished analysing complex vision data.
- Hardware triggers, such as pressing a button to detect when a gripper has successfully closed.

By not processing computationally expensive code at each program loop, program efficiency is increased, and processing power requirements can be lowered.

10.2.2 ROS Components

ROS is not a traditional operating system that replaces another, nor is it a new programming language. It is a framework designed to standardise communications between devices and programs that may exist in a single standalone system or be interconnected through an Ethernet/IP network topology. This is achieved through a messaging system, where data packages in the form of messages are relayed between devices in a ROS network. Although this seems abstract, it is quite analogous to a traditional standalone monolith program.

A typical computer program may consist of the following components

- *Variables* that store data within a program, such as a counter.
- *Functions* that process raw input data, and outputs it as useful data.
- *Methods* that execute tasks in relation to the program, such as saving data to a file.

A simple way to understand the fundamental components of ROS is to find direct comparisons of these components to those of a computer program.

Nodes

Nodes are analogous to traditional computer programs that contain all the code, algorithms, and ROS components to function and communicate. For instance, a node can be setup to keep track of all sensor information and update the readings for the entire ROS system to see. Another node can be to solve the inverse kinematics of a robot. A node communicates to other nodes through a messaging system, in which the *ROS master* keeps track of where relevant messages should go. When a node starts, it advertises its *topics* and *services*, which the master will make available for all other connected nodes. These components will be explained further in this section.

Master

The ROS master is the server for the entire ROS system. Whenever new nodes connect to the ROS network, they will always connect to the ROS master. The master will keep track of all ROS components and forward messages to the right nodes. Only one ROS master can exist per ROS network.

Messages

All data transmission between nodes and devices is implemented through messages. Although this kind of abstract data transmission does not occur in a normal computer program, comparisons can still be made whenever we write a value to a variable, or call a function in a program with an input value. In each of these cases, we are providing a copy (or reference) of the data and storing it in the variable, or passing it to the function to then process. In ROS, a definite copy of the data is made, encapsulated as a ROS message, and then transmitted through the ROS network to the relevant receiver, such as a *topic* or *service*. Furthermore, a message can be made up of a single data type such as a `float` or an array of `uint8`, or multiple data types such as a `bool`, `int32`, and an array of `float`. Each message is defined as its own type, and senders and receivers must use the same message type to communicate.

Topics

Topics are analogous to global variables in a computer program. It is a mechanism for sharing data amongst all nodes within the ROS network. For instance, the `/joint_state` topic may contain the latest information on the joint states of a robot. A topic is created by a node and maintained and advertised by the ROS master. A ROS topic is updated by *publishing* a message to the topic with a *publisher* created from any node. However, for a node to receive updates from a topic, it must create a *subscriber* handle to the topic. The subscriber listens to the topic, and will only receive updates when a new message is published to the topic.

Publishers

A publisher publishes to a topic by sending a message (of the correct message type) to the topic. For instance, a thread may update the joint configuration of a robot at 10 Hz, so the rest of the nodes who may be *subscribed* to the topic can get the most up-to-date information.

Subscribers

A subscriber subscribes to a topic by creating an event handler and an associated callback function. When a publisher publishes a new message to the topic, all subscribers to the node are notified, and their respective callback functions are triggered. This ensures that code is only executed on new data, promoting program efficiency. Sometimes, a topic is updated while a subscriber's callback code is still executing. If this happens, later callbacks are queued (finitely) so that new information is not missed.

Services

A ROS service is analogous to a function call or method in a computer program. A service is created by starting a *service server* on the hosting node, which is then advertised by the ROS master. To call a ROS service, a *service client* is created from the client node, which allows calling of the service by sending it a *service message*. Service messages work on the same principle as regular messages but exist only to transfer messages between the service server and the client. When a service is called, execution on the calling node is blocked until a return message from the hosting service is received. This is known as a *blocking call*, whereby code execution is paused while waiting for a routine to finish. While this behaviour is common among single-threaded programs, there are scenarios where this can cause problems. For instance, on services that require long processing times, this can cause

ROS programs to become unresponsive and cause unnecessary delays in other critical tasks such as driving actuators or reading sensors. If a blocking service call is undesirable, then service implementation through an *action service* is recommended.

Action Service

An action service is the same as a regular ROS service, with the key exception that action service calls are *non-blocking* to the client node. Action services are implemented on servo-control routines so that a ROS program does not have to wait while the robot is moving. This is an example of *asynchronous programming*. While asynchronous execution of services seems desirable in all cases, it unnecessarily complicates the ROS program, especially when determining the *state* of the program.[2]

10.2.3 ROS System: Case Study

Consider the following robotic system, whose task is to autonomously sort recyclable waste that is randomly stored in a bin. The system comprises three main subsystems:

- a vision system to identify and locate objects in the bin,
- a gripper to grasp onto objects, and
- a robotic manipulator to manipulate the gripper during the sorting process.

Figure 10.1 shows a possible ROS implementation of this robotic system. There are four main processors (computers or control boxes) that handle each subsystem of the robot. This is called distributed computing, whereby each processor is configured to handle a task as efficiently as possible. There are many advantages of this setup.

- Each subsystem, in hardware or code, can be developed individually without affecting other subsystems
  - This feature significantly simplifies collaboration among programmers
  - Debugging can be performed without shutting down the entire system
- If a subsystem or individual node fails or crashes, it will not bring down the entire system
- The system is easily expandable, simply by adding additional processors to the ROS network

In addition, each computer can host multiple nodes that specialise in a particular task, such as solving robot kinematics, communicating with sensors, or detecting objects using convoluted neural network algorithms. Each node can also be launched independently on separate threads of a processor to improve performance through parallel processing.

Finally, the hardware peripherals associated with each computer and subsystem are usually connected via USB or other high-bandwidth connections, and certain nodes are dedicated to communications to these devices. For example, on *Computer B*, the `camera` node allows communication to the camera via USB, and exposes two topics to the ROS network: `/camera/point_cloud` and `/camera/image_RGB`. These topics will be updated by the `camera` node at a constant rate, such as 10 Hz, for example.

[2]Determining the state of a program is critical for the implementation of a *state machine*. A state machine is a behavioural model, which dictates a finite number of connected states of a program. Based on the program's current state and input, the machine performs state transitions to produce the next output and move to the next state. Because asynchronous threads are difficult to manage due to their parallel execution, state machines are difficult to implement on such programs.

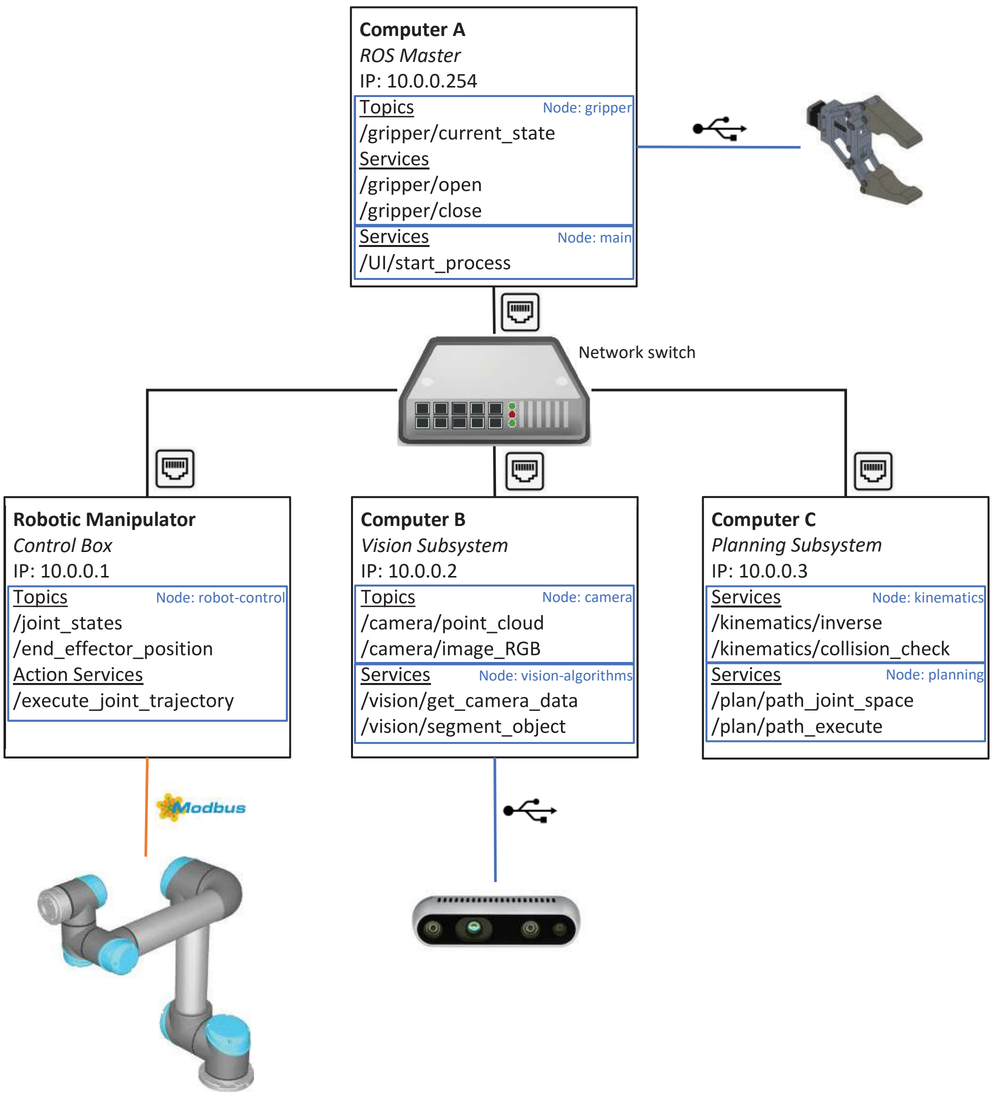


FIGURE 10.1
A ROS implementation of a waste-sorting robot.

ROS Processors

Computer A

hosts the *ROS master* node as well as the `gripper` and `main` program nodes. The `gripper` node handles all communications with the gripper via the USB interface and exposes the following ROS components.

- Topics
  - `/gripper/current_state`. Any subscribers to this topic will receive updates on the current state of the gripper, whether it is open, closed, or grasping an object.

- Services
  - `/gripper/open`, `/gripper/close`. Calling these services opens or closes the gripper.

The `main` node controls the state of the program and generates a user interface to allow interaction with the user and robot. It also contains the main program loop, and exposes a single ROS service `/UI/current_state`. This service is called only when a user presses "Start" on the user interface, for example.

Robotic Manipulator: Control Box

Control Box contains all the hardware relating to the control of the robotic manipulator, as well as having an onboard processor that hosts the `robot-control` node. This node handles all low-level communications with the robot's actuators, as well as provides the current status of the robot's joints and end effector positions through the following ROS components.

- Topics
  - `/joint_states`, `/end_effector_position`. Any subscribers to these topics will receive updates on the joint positions (as a vector), or end effector position (as a transformation matrix), respectively.
- Action services
  - `/execute_joint_trajectory`. Calling this action service with a given list of joint velocities and positions will actuate the robot's joints accordingly. This is a non-blocking call, which will allow the calling node to continue executing while the robot is moving.

Computer B

Computer B hosts two nodes. The `camera` node communicates with the camera via the USB interface and exposes the following ROS topics.

- `/camera/point_cloud`, `/camera/image_RGB`. Subscribers to these topics will receive point cloud data or an RGB image from the camera at a fixed rate. Note that the messages sent and received from these topics can be quite large, depending on the image resolution. Hence, only nodes requiring this information should subscribe to it.

Running in parallel, the `vision-algorithms` node implements object-segmentation algorithms to detect and categorise objects, and returns data based on the service called. This node exposes the following ROS services.

- `/vision/get_camera_data` returns the camera's point cloud data and RGB image on demand. Because subscribing to a stream of RGB and point cloud data can be computationally expensive and utilise unnecessary bandwidth within the ROS network, using a service to get one-off data from each topic can be more efficient.
- `/vision/segment_object` returns a list of transformation matrices of all objects and their type (as an integer) based on given camera sensor data.

Computer C

Computer C hosts two nodes. The `kinematics` node calculates the kinematics of the robotic manipulator by hosting the following ROS services.

- `/kinematics/inverse`. This service solves the inverse kinematics of the robot, given a transformation matrix of the gripper. It will return a message containing a list of joint states that satisfy the gripper's pose.
- `/kinematics/collision_check`. Calling this service, given a joint space point, will calculate whether the robotic manipulator is in collision with itself.

The `planning` node implements path planners for the robotic manipulator by exposing the following ROS services.

- `/plan/path_joint_space`. Calling this service, given an initial and goal joint state, will calculate a collision-free trajectory and return a list of joint velocities and positions that link the initial and goal positions.
- `/plan/path_execute`. Calling this service will call the action service to execute the given list of joint velocities and positions.

Robot operation: Autonomous Trash Sorting

Assume the system is ready to start autonomous trash sorting and is in the home position q_0 with the camera facing inside the box of trash.

1. Wait for user interface trigger

The ROS program begins at the `main` ROS node, which polls for any `/UI/start_process` calls. This is triggered whenever the "Start" button is pressed on the user interface generated by the node. Once the `/UI/start_process` service is called, the main routine begins from the next step.

2. Get camera data

The first service called within the `/UI/start_process` service is `/vision/get_camera_data` the service. Within this service, subscribers are made to the `/camera/point_cloud` and `/camera/image_RGB` to get the latest camera data. When execution of the `/vision/get_camera_data` service finishes, point cloud data and its associated RGB image are returned, and the subscribers are destroyed to save resources. Flow returns back to the `/UI/start_process` service.

3. Identify objects

Next, the `/vision/segment_object` service is called by sending it a service message with point cloud and RGB image data retrieved in Step 2. The service will run object-detection algorithms to identify all objects in the box (if any exist). Any items identified and classified are returned as a list of transformation matrices ${}^{B}\mathbf{T}_{I_1}, {}^{B}\mathbf{T}_{I_2}, ..., {}^{B}\mathbf{T}_{I_n}$ where B is the robot base frame and I_n is the n-th item's frame. In addition, a list of integers is also returned, which identifies how the item should be sorted. If there are no objects detected, the `/UI/start_process` service terminates, and we return to Step 1 where control is returned to the main program loop in the `main` ROS node.

4. Verify reachability of Item 1

Once a list of items' transformation matrices is obtained, the robot will attempt to move the gripper to the first object to grasp (we will call this *Item 1*). The `/kinematics/inverse` service is called by sending a service message with the proposed transformation matrix of the gripper to grasp Item 1, called ${}^{B}\mathbf{T}_{G}$. In a grasp scenario, ${}^{B}\mathbf{T}_{G} = {}^{B}\mathbf{T}_{I_1}$. The return message will be a list of joint space configurations, $Q = \{q_1, q_2, ...q_n\}$, that satisfies ${}^{B}\mathbf{T}_{G}$. The `/kinematics/collision_check` service is called by sending a service message with the list of configurations Q, to check whether any of these joint space configurations result in robot self-collisions. The return service message will be the list of joint space configurations, $Q' \subseteq Q$ that are collision-free.

5. Plan to grasp Item 1

The `main` node has a subscriber to the `/joint_states` topic, which continuously updates the current joint position of the robot, and stores it in a variable `js`. With a list of joint configurations Q' ready from Step 4, a path from the robot's initial state `js` to the first configuration in Q' (q_1) can be planned. The `/plan/path_joint_space` service is called by sending it a service message with the initial joint state `js` to the goal joint state q_1. The path planner within the service will calculate the joint trajectories required to link the initial and goal states. If a path to q_1 is found, a list of joint velocities and joint positions (Q_v, Q_p) is returned. If a path is not found, the next goal position in Q' (q_2) is planned instead, and the process continues for each failed path plan. If no paths are found among all elements in Q', then Item 1 cannot be grasped, and we return to Step 1.

6. Execute plan to grasp Item 1

Before execution of the planned path from Step 5 to reach q_1, the `/gripper/open` service is called to open the gripper. Then `/plan/path_execute` is called with (Q_v, Q_p) included in the service message. Within the `/plan/plan_execute` service, the `/execute_joint_trajectory` action service is called, which will directly communicate with the robot in executing the (Q_v, Q_p) trajectory. Note that this is a non-blocking call within the `/plan/plan_execute` service. Hence, this service will continue to execute while the robot is moving. However, the parent service, `/UI/start_process`, is blocked from continuing until the robot has finished moving to q_1. This will ensure the robot has reached its final position before continuing, especially when the next step is to grasp Item A!

7. Depositing Item 1

Once `/plan/plan_execute` has finished executing, the robot should be at q_1. The `/gripper/close` service is called to close the gripper to grasp Item 1. A temporary subscriber to the `/gripper/current_state` topic is created to check whether the grasp is successful. From here, there are two possible outcomes.

1. If the grasp is *not* successful, the gripper is opened (by calling `/gripper/open`) in case an object was grasped, but incorrectly.
2. If the grasp *is* successful, a path is planned from q_1 to q_b, where q_b is the joint state of the robot that places the gripper over the correct depositing box, as defined by the object's sorting criteria defined in Step 3. The `/plan/plan_execute` service is called to move the robot to q_b, and finally, the gripper is opened by calling `/gripper/open` to drop Item 1 into the deposit box.

8. Returning home

The `/plan/path_joint_space` and `/plan/plan_execute` services are called to move the robot from its current position defined by `js` (which should be updated by the topic `/joint_states`) to the home position q_0. The `/UI/start_process` service loops back to Step 2.

10.3 Conclusion

In this chapter, we discussed how to model robotic systems efficiently in MATLAB, utilising symbolic equations and function handles to effortlessly calculate the forward kinematics of a general serial manipulator. The functions and techniques discussed in modelling can be generalised to any system that requires the modelling of frames, given the system's current state.

Also, ROS or Robot Operating System was introduced. It is an open source middleware that provides the framework to standardise communcation between hardware and software components of a robotic system using the Ethernet/IP protocol. Its operating paradigm was discussed, and each component of what makes ROS function were described in detail. Finally, a realistic case study was used to demonstrate how all of the ROS concepts and components work together to create a functional robotic system.

11

Lagrangian Dynamics

In Chapter 8, we studied how to use Jacobians to calculate a robot's joint torques and forces under a static load at the end-effector. However, when a robot begins to accelerate, the dynamics of the rigid links and their effects on the actuators come into play.

Dynamics is the study of the relation between the force and the motion of an object. When the study is applied to a robotic system, we investigate how the torques and forces exerted on its rigid links (such as from end-effector payload, self-inertia, and gravity) affect the robot's motion via its actuators. This analysis is very important for all robots, and especially for agile robots or those that carry significant payloads, as dynamic torques and forces can be significant relative to static loading. Suppose a robot's dynamics are not factored into its control system. In that case, it can introduce unwanted vibrations or oscillations, destabilising the robot's motion and causing the loss of control of the end-effector.

This chapter will introduce the dynamic analysis of robotics through Lagrangian mechanics. It is centred around the *Lagrangian*, which is a single equation that encapsulates the dynamics of the entire robotic system as the sum of all kinetic and potential energies. The resultant equations of motion, which model the torques and forces as seen by the actuators from the robot's dynamics, is solved by using the *Lagrange equation*.

11.1 Rigid Body Dynamics

The mass of one rigid body is its intrinsic dynamic parameter, which links the movement of the body and the force applied to the body. According to Newton's second law

$$\mathbf{f} = m\,\mathbf{a} \tag{11.1}$$

$\mathbf{f}$ is the inertial force applied by a body with mass m accelerates at $\mathbf{a}$. Note here that mass m is assumed to be a point mass, such as that shown in Figure 11.1.

Because rigid bodies such as robotic links are too big to be considered as a point, the total mass of the body can be condensed to a single point called the *centre of mass* (CoM). The CoM of a rigid body is the mass-weighted geometric centre of the distributed mass in space, which is given by

$$^{0}\mathbf{c} = \frac{\int_B {^{0}\mathbf{r}}\,dm}{\int_B dm} = \frac{\int_B {^{0}\mathbf{r}}\,dm}{M} \tag{11.2}$$

where $^{0}\mathbf{c}$ is the position of the CoM measured in a universal fixed frame $\{0\}$, $^{0}\mathbf{r}$ is the position vector measured in $\{0\}$, B is the rigid body region, dm is an infinitesimally small mass element, and M is the total mass of the rigid body.

This calculation can be simplified by attaching the fixed frame to the rigid body such that the vector components are constant, i.e., frame $\{0'\}$, resulting in an invariant vector

DOI: 10.1201/9781003614319-11

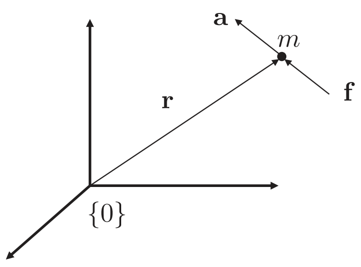


FIGURE 11.1
A point mass.

${}^{0'}\mathbf{c}$. Further, if the attached frame is located at the CoM of this body, we must have ${}^{0'}\mathbf{c} = 0$. According to Equation (11.2), we have

$$\int_B {}^{0'}\mathbf{r}\, dm = 0 \tag{11.3}$$

In many cases, the mass of a rigid body is condensed into a point mass located at the CoM such that the behaviour of the point mass is governed by Equation (11.1). In the case that the distributed masses cannot be neglected, the *inertia tensor* describing the mass distribution of the rigid body is required.

11.2 Inertia Tensors

Consider a rigid body rotating around a pivot point with an angular velocity $\boldsymbol{\alpha}$ under a torque $\boldsymbol{\tau}$, as shown in Figure 11.2. With all vectors measured in the ground frame, Newton's second law can be applied to an arbitrary infinitesimal mass dm

$$d\,\mathbf{f} = dm\ \ \mathbf{a} \tag{11.4}$$

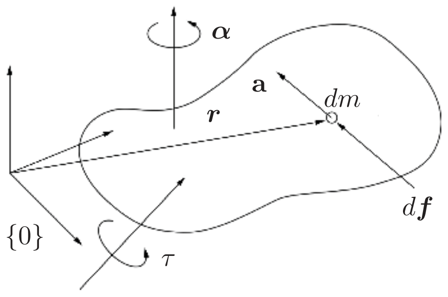


FIGURE 11.2
A rigid body.

where $d\,\mathbf{f}$ and $\mathbf{a}$ are the equivalent force applied on this point mass and its acceleration, respectively. Combining the following relations

$$d\,\boldsymbol{\tau} = \mathbf{r} \times d\,\mathbf{f} \qquad\qquad \mathbf{a} = \boldsymbol{\alpha} \times \mathbf{r} \tag{11.5}$$

with Equation (11.4) yields

$$\begin{aligned} d\,\boldsymbol{\tau} &= -dm\ \ \mathbf{r} \times (\mathbf{r} \times \boldsymbol{\alpha}) \\ &= -dm\ [\mathbf{r}][\mathbf{r}]\ \boldsymbol{\alpha} \\ &= -dm\ [\mathbf{r}]^2\ \boldsymbol{\alpha} \end{aligned} \tag{11.6}$$

where $[\mathbf{r}]$ is the cross product of $\mathbf{r}$. Hence, the total torque acting on the rigid body is given by

$$\boldsymbol{\tau} = \int d\,\boldsymbol{\tau} = -\int [\mathbf{r}]^2 dm\ \ \boldsymbol{\alpha} \tag{11.7}$$

which is the dynamic relation between $\boldsymbol{\tau}$ and $\boldsymbol{\alpha}$. The term $-\int [\mathbf{r}]^2 dm$ is a 3×3 matrix and is known as the inertia tensor, or

$$\mathbf{I} = -\int [\mathbf{r}]^2 dm \tag{11.8}$$

We can further expand $\mathbf{I}$ to define its individual elements. Assuming $\mathbf{r} = \begin{bmatrix} x & y & z \end{bmatrix}^T$, then Equation (11.8) becomes

$$\begin{aligned} \mathbf{I} &= \int \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -xy & x^2+z^2 & -yz \\ -xz & -yz & x^2+y^2 \end{bmatrix} \\ &= \begin{bmatrix} \int (y^2+z^2)dm & -\int xydm & -\int xzdm \\ -\int xydm & \int (x^2+z^2)dm & -\int yzdm \\ -\int xzdm & -\int yzdm & \int (x^2+y^2)dm \end{bmatrix} \\ &= \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \end{aligned} \tag{11.9}$$

where I_{xx}, I_{yy}, and I_{zz} are the *mass moments of inertia*, while I_{xy}, I_{xz}, and I_{yz} are the *mass products of inertia*. Typically, the larger the moment of inertia around a given axis, the more torque is required to accelerate or decelerate the rotation of a body about this axis.

Example 11.1 (Inertia tensor): Find the inertia tensor of a robotic link in the frame shown in Figure 11.3. Assume it has a mass of $m = 1$ kg and a corresponding density of $\rho = 0.125$ kg/m^3.

Solution: According to Equation (11.9), the mass moment I_{xx} of inertia of the block is given by

$$I_{xx} = \int_0^{0.1}\int_0^{0.8}\int_0^{0.1} (y^2+z^2)\rho dxdydz = 0.217 \tag{11.10}$$

Similarly, we have $I_{yy} = 0.007$ and $I_{zz} = 0.217$. According to Equation (11.9), the mass product of inertia, I_{xy}, is given by

$$I_{xy} = \int_0^{0.1}\int_0^{0.8}\int_0^{0.1} xydxdydz = 0.020 \tag{11.11}$$

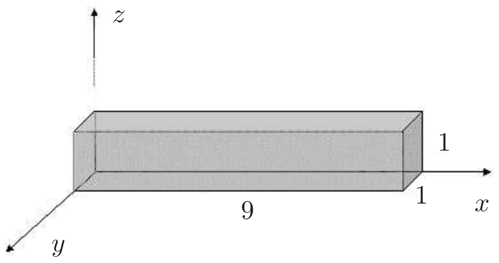


FIGURE 11.3
A robotic link.

Similarly, we have $I_{xz} = 0.003$ and $I_{yz} = 0.020$. Therefore in this reference frame $\{A\}$, the inertia tensor is given by

$$^{A}\mathbf{I} = \begin{bmatrix} 0.217 & -0.020 & -0.003 \\ -0.020 & 0.007 & -0.020 \\ -0.003 & -0.020 & 0.217 \end{bmatrix} \tag{11.12}$$

11.3 Principal Moments of Inertia

The values of the inertia tensor for a rigid body depend on the frame in which it is measured. Therefore, it is possible to select a measurement frame such that all mass products in the upper and lower triangle of the inertia tensor vanish. The axes of this frame are called the *principal axes* of this body, and the mass moments are the *principal moments* of the inertia, where

$$^{A}\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{11.13}$$

The eigenvalues of an inertia tensor are the principal moments for the body. The associated eigenvectors are orthogonal to each other due to the symmetric inertia tensor, as proven in Appendix 25.1. Hence, these eigenvectors are called the principal axes, which constitute the principal frame.

Example 11.2 (Principal moment and frame): For the robot link in Figure 11.4, find its principal moments and the corresponding principal frame $\{P\}$ at the origin of the given frame $\{A\}$.

Solution: Based on Equation (11.12), the eigenvalues are

$$\lambda_1 = 0.003 \qquad \lambda_2 = 0.218 \qquad \lambda_3 = 0.219 \tag{11.14}$$

and associated eigenvectors are

$$\mathbf{c}_1 = \begin{bmatrix} -0.094 & -0.991 & -0.094 \end{bmatrix}^T \tag{11.15}$$

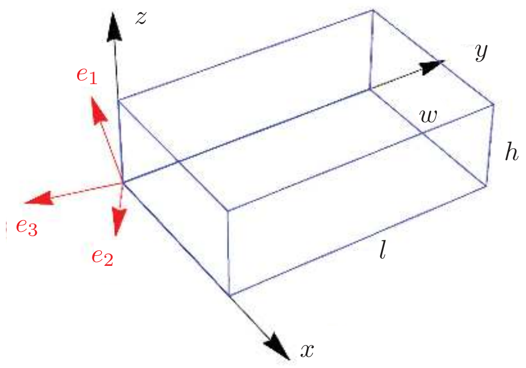


FIGURE 11.4
A robot link with an indicative principal frame $\{P\}$ located at its origin frame $\{A\}$.

$$\mathbf{c}_2 = \begin{bmatrix} -0.701 & 0.133 & -0.701 \end{bmatrix}^T \tag{11.16}$$

$$\mathbf{c}_3 = \begin{bmatrix} 0.707 & 0 & 0.707 \end{bmatrix}^T \tag{11.17}$$

If we set eigenvectors $\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$ as the x, y, and z axes of the principal frame $\{P\}$, then we have the inertia tensor measured in the $\{P\}$ as

$$^P\mathbf{I} = \begin{bmatrix} 0.003 & 0 & 0 \\ 0 & 0.218 & 0 \\ 0 & 0 & 0.219 \end{bmatrix} \tag{11.18}$$

and the orientation of $\{P\}$ measured $\{A\}$ is

$$^A\mathbf{R}_P = \begin{bmatrix} -0.094 & -0.701 & 0.707 \\ -0.991 & 0.133 & 0 \\ -0.094 & -0.701 & -0.707 \end{bmatrix} \tag{11.19}$$

You can check the validity of the rotation matrix $^A\mathbf{R}_P$ by ensuring $\det(^A\mathbf{R}_P) = 1$ and $^A\mathbf{R}_P\ ^A\mathbf{R}_P^T = \mathbf{I}_3$.

The relation of the inertia tensors of the same rigid body measured in two different frames with the same origin can be expressed by the *similarity transformation*,

$$^A\mathbf{I} = {^A\mathbf{R}_B}\ {^B\mathbf{I}}\ {^A\mathbf{R}_B^T} \tag{11.20}$$

For example, the inertia tensor in the frame $\{P\}$ consisting of the eigenvectors in Example 11.2 can be derived as

$$\begin{aligned} ^P\mathbf{I} &= {^P\mathbf{R}_A}\ {^A\mathbf{I}}\ {^P\mathbf{R}_A^T} \\ &= \begin{bmatrix} 0.094 & -0.991 & -0.094 \\ -0.701 & 0.133 & -0.701 \\ 0.707 & 0 & -0.707 \end{bmatrix} \begin{bmatrix} 0.217 & -0.020 & -0.003 \\ -0.020 & 0.007 & -0.020 \\ -0.003 & -0.020 & 0.217 \end{bmatrix} \begin{bmatrix} -0.094 & -0.701 & 0.707 \\ -0.991 & 0.133 & 0 \\ -0.094 & -0.701 & -0.707 \end{bmatrix} \\ &= \begin{bmatrix} 0.003 & 0 & 0 \\ 0 & 0.218 & 0 \\ 0 & 0 & 0.219 \end{bmatrix} \end{aligned} \tag{11.21}$$

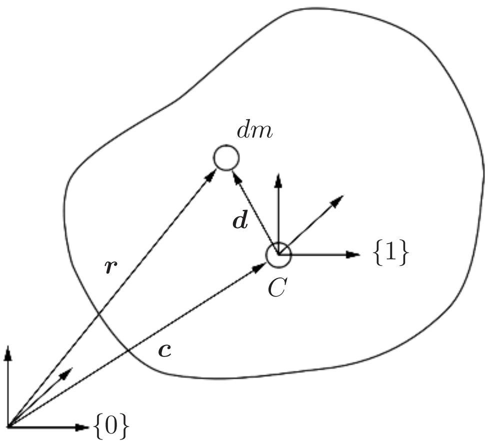


FIGURE 11.5
Two parallel frames associated with a rigid body.

11.3.1 Parallel Axis Theorem

The *parallel-axis theorem* relates the inertia tensors of the same body measured in two parallel frames with different locations, as shown in Figure 11.5. It is expressed as

$$ {}^{0}\mathbf{I} = {}^{1}\mathbf{I} - m[\mathbf{c}]^2 = {}^{1}\mathbf{I} + m\left(\mathbf{c}^T\ \mathbf{c}\ \mathbf{1}_3 - \mathbf{c}\ \mathbf{c}^T\right) \tag{11.22} $$

where m, $\mathbf{1}_3$, and $\mathbf{c}$ are the total mass, the 3×3 identity matrix and the vector from the origin of $\{0\}$ to that of $\{1\}$, respectively. Further, $\{0\}$ and $\{1\}$ are parallel, while $\{1\}$ sits at the mass centre. The proof of this theorem is given in Appendix 25.3.

Example 11.3 (Parallel axis theorem): Find the inertia tensor of the block shown in Figure 11.3 in a frame $\{1\}$ at the mass centre and parallel to $\{0\}$.

Solution: In Example 11.1, the inertia tensor in $\{0\}$ is derived as

$$ {}^{0}\mathbf{I} = \begin{bmatrix} 0.217 & -0.020 & -0.003 \\ -0.020 & 0.007 & -0.020 \\ -0.003 & -0.020 & 0.217 \end{bmatrix} \tag{11.23} $$

The mass centre in $\{0\}$ is given by $\mathbf{c} = \begin{bmatrix} 0.05 & 0.40 & 0.05 \end{bmatrix}^T$. According to Equation (11.22), we have

$$ \begin{aligned} {}^{1}\mathbf{I} &= {}^{0}\mathbf{I} - m(\mathbf{c}^T\ \mathbf{c}\ \mathbf{1}_3 - \mathbf{c}\ \mathbf{c}^T) \\ &= \begin{bmatrix} 0.217 & -0.020 & -0.003 \\ -0.020 & 0.007 & -0.020 \\ -0.003 & -0.020 & 0.217 \end{bmatrix} - \left(\begin{bmatrix} 0.165 & 0 & 0 \\ 0 & 0.165 & 0 \\ 0 & 0 & 0.165 \end{bmatrix} - \begin{bmatrix} 0.003 & 0.020 & 0.003 \\ 0.020 & 0.160 & 0.020 \\ 0.003 & 0.020 & 0.003 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0.054 & 0 & 0 \\ 0 & 0.002 & 0 \\ 0 & 0 & 0.054 \end{bmatrix} \end{aligned} \tag{11.24} $$

In this case, all mass products vanish. Hence, $\{1\}$ is the principal frame, while the mass moments are the principal moments of the inertia at the CoM.

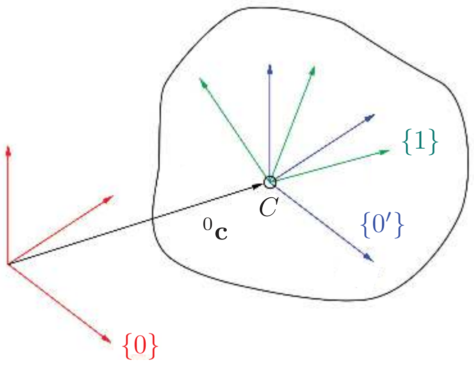


FIGURE 11.6
Two general frames associated with a rigid body.

Combining the similarity transformation and the parallel-axis theorem, one can obtain the inertia tensor of a rigid body measured in an arbitrary frame based on the inertia tensor measured in a frame located at the CoM. As shown in Figure 11.6, {0} is not parallel to {1} that sits at the CoM of the body. By introducing an auxiliary frame {0′} parallel to {0} and sitting at the CoM, we can readily have

$$
{}^0\mathbf{I} = {}^0\mathbf{R}_1\, {}^1\mathbf{I}\, {}^0\mathbf{R}_1^T - m[{}^0\mathbf{c}]^2 \tag{11.25}
$$

Inertia tensors also have other important properties.

- Where a plane of symmetrical mass distribution can be identified, and that two reference frame axes are on said plane, the inertia or mass product relevant to the axes the will vanish.
- Moments of inertia must be positive.
- The sum of the three moments of inertia, the trace of the inertia tensor, is invariant under orientation changes in the reference frame.

11.4 The Lagrangian Method

Consider a system consisting of n rigid bodies. The mass and inertia tensor of each body are m_i and $\mathbf{I}_i$, respectively, where $i = 1, \ldots, n$. Note that m_i is frame-invariant while $\mathbf{I}_i$ is frame-variant. The Lagrangian of the system is defined as the difference between the total kinetic energy and the total potential energy

$$
\mathcal{L} = K - V. \tag{11.26}
$$

The Lagrange equation is

$$
\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = f_i, \quad i = 1, \ldots, n \tag{11.27}
$$

where q_i and f_i are the generalised coordinate and force applied along q_i, respectively. In robotic applications that utilise only revolute actuators, we typically choose q_i as joint angles

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_i}\right) - \frac{\partial \mathcal{L}}{\partial \theta_i} = \tau_i, \quad i = 1, \ldots, n \tag{11.28}$$

where θ_i and τ_i are the input joint angle and input torque of the i-th actuator, respectively.

Kinetic Energy K

The kinetic energy of the system is given by

$$K = \sum_{1}^{n} K_i = \sum_{1}^{n} \left(\frac{1}{2} m_i \, \mathbf{v}_{ci}^T \, \mathbf{v}_{ci} + \frac{1}{2} \, \boldsymbol{\omega}_i^T \, \mathbf{I}_i \, \boldsymbol{\omega}_i\right) \tag{11.29}$$

where K_i, $\mathbf{v}_{ci}$, and $\boldsymbol{\omega}_i$ are the kinetic energy, velocity at mass centre, and angular velocity, respectively, of Body i. Note that $\mathbf{I}_i$ and $\boldsymbol{\omega}_i$ must be measured in the same frame. The velocity of the mass centre is given by

$$\mathbf{v}_{ci} = {}^0_0\mathbf{v}_{ci} = \frac{d\left({}^0_0\mathbf{p}_{ci}\right)}{dt} = \frac{d\left({}^0\mathbf{T}_1(\theta_1) \ldots {}^{i-1}\mathbf{T}_i(\theta_i) \, {}^i_i\mathbf{p}_{ci}\right)}{dt} \tag{11.30}$$

The angular velocity is given by

$${}^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i(\theta_{i+1}) \, {}^i\boldsymbol{\omega}_i + \dot{\theta}_{i+1} \, {}^{i+1}\mathbf{z}_{i+1} \tag{11.31}$$

with the initial condition ${}^0\boldsymbol{\omega}_0 = \mathbf{0}$. If we choose the frame of the inertia tensor $\mathbf{I}_i$ is parallel to the $\{i\}$, we have

$$K_i = \frac{1}{2} m_i \, \mathbf{v}_{ci}^T \, \mathbf{v}_{ci} + \frac{1}{2} \, {}^i\boldsymbol{\omega}_i^T \, {}^i\mathbf{I}_i \, {}^i\boldsymbol{\omega}_i \tag{11.32}$$

If we choose the principle frame to measure the inertia tensor ${}^p\mathbf{I}_i$, which is not parallel to the $\{i\}$, we have

$${}^p\boldsymbol{\omega}_i = {}^p\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i \tag{11.33}$$

which yields

$$\begin{aligned} K_i &= \frac{1}{2} m_i \, \mathbf{v}_{ci}^T \, \mathbf{v}_{ci} + \frac{1}{2} \, {}^p\boldsymbol{\omega}_i^T \, {}^p\mathbf{I}_i \, {}^p\boldsymbol{\omega}_i \\ &= \frac{1}{2} m_i \, \mathbf{v}_{ci}^T \, \mathbf{v}_{ci} + \frac{1}{2} \left({}^p\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i\right)^T \, {}^p\mathbf{I}_i \left({}^p\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i\right) \\ &= \frac{1}{2} m_i \, \mathbf{v}_{ci}^T \, \mathbf{v}_{ci} + \frac{1}{2} \, {}^i\boldsymbol{\omega}_i^T \, {}^p\mathbf{R}_i^T \, {}^p\mathbf{I}_i \, {}^p\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i \end{aligned} \tag{11.34}$$

Another way to interpret the term $\frac{1}{2} \, {}^i\boldsymbol{\omega}_i^T \, {}^p\mathbf{R}_i^T \, {}^p\mathbf{I}_i \, {}^p\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i$ is that the inertia tensor is mapped from $\{p\}$ to $\{i\}$, i.e.,

$$\frac{1}{2} \, {}^i\boldsymbol{\omega}_i^T \, {}^i\mathbf{I}_i \, {}^i\boldsymbol{\omega}_i = \frac{1}{2} \, {}^i\boldsymbol{\omega}_i^T \left({}^p\mathbf{R}_i^T \, {}^p\mathbf{I}_i \, {}^p\mathbf{R}_i\right) {}^i\boldsymbol{\omega}_i \tag{11.35}$$

where

$${}^i\mathbf{I}_i = {}^p\mathbf{R}_i^T \, {}^p\mathbf{I}_i \, {}^p\mathbf{R}_i \tag{11.36}$$

which is called the similarity transformation of the inertia tensor.

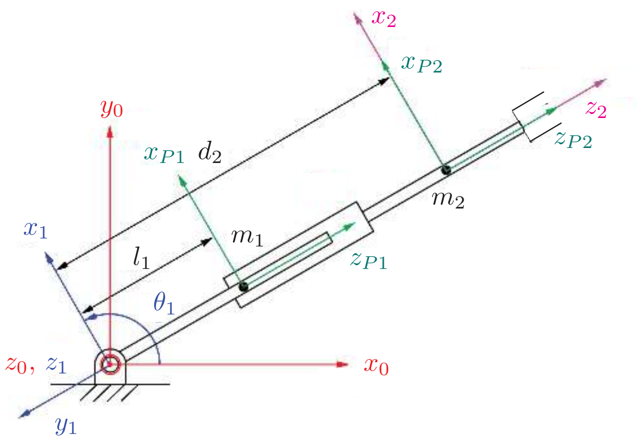


FIGURE 11.7
A RP robot.

Potential Energy V

The potential energy of the system is given by

$$V = \sum_{1}^{n} V_i = \sum_{1}^{n} (-m_i\, {}^0\mathbf{g}^T\, {}^0\mathbf{p}_{ci}) \tag{11.37}$$

where V_i, and ${}^0\mathbf{p}_{ci}$ are the potential energy and position vector of the mass centre of Body i, respectively, while ${}^0\mathbf{g}$ is the vector of gravity acceleration.

Example 11.4 (PR robot): Given a two-DoF PR robotic manipulator with assigned reference frames shown in Figure 11.7, derive the dynamic equations governing this system. Assume that the inertia tensor of each link in the attached frame is given by, respectively,

$${}^{P1}\mathbf{I}_1 = \begin{bmatrix} I_{xx1} & 0 & 0 \\ 0 & I_{yy1} & 0 \\ 0 & 0 & I_{zz1} \end{bmatrix}, \quad {}^{P2}\mathbf{I}_2 = \begin{bmatrix} I_{xx2} & 0 & 0 \\ 0 & I_{yy2} & 0 \\ 0 & 0 & I_{zz2} \end{bmatrix} \tag{11.38}$$

Solution:

Step 1: Transformation matrices

According to the frames assigned to the RP robot as shown in Figure 11.7, the DH parameters in Table 11.1 can be derived. Hence, the transformation matrices are given by

TABLE 11.1
DH parameters of the RP robot

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	90	0	d_2	0

$$
{}^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -d_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Step 2: Angular velocities

The angular velocities of two links, measured in their own frames, are derived by using velocity propagation as

$$
{}^1\boldsymbol{\omega}_1 = {}^1\mathbf{R}_0\,{}^0\boldsymbol{\omega}_0 + \dot\theta_1 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dot\theta_1 \end{bmatrix} \qquad {}^2\boldsymbol{\omega}_2 = {}^2\mathbf{R}_1\,{}^1\boldsymbol{\omega}_1 + \dot\theta_2 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \dot\theta_1 \\ 0 \end{bmatrix} \tag{11.39}
$$

where $\dot\theta_2 = 0$ because $\theta_2 = 0$. As shown in Figure 11.7, the rotation matrices between the DH frames and the corresponding principle frames are given by

$$
{}^1\mathbf{R}_{P1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad {}^2\mathbf{R}_{P2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11.40}
$$

Therefore, the angular velocities in the principle frames are given by

$$
{}^{P1}\boldsymbol{\omega}_1 = {}^{P1}\mathbf{R}_1\,{}^1\boldsymbol{\omega}_1 = \begin{bmatrix} 0 \\ \dot\theta_1 \\ 0 \end{bmatrix} \qquad {}^{P2}\boldsymbol{\omega}_2 = {}^2\boldsymbol{\omega}_2 \tag{11.41}
$$

Step 3: Velocities at mass centres

The positional homogeneous coordinates of the mass centres of the two links measured in their own frames are given by

$$
{}^1\mathbf{c}_1 = \begin{bmatrix} 0 \\ -l_1 \\ 0 \\ 1 \end{bmatrix} \qquad {}^2\mathbf{c}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{11.42}
$$

Hence, the positional homogeneous coordinates of the mass centres of the two links measured in the ground frame are

$$
{}^0\mathbf{c}_1 = {}^0\mathbf{T}_1\,{}^1\mathbf{c}_1 = l_1 \begin{bmatrix} s_1 \\ -c_1 \\ 0 \\ 1 \end{bmatrix} \qquad {}^0\mathbf{c}_2 = {}^0\mathbf{T}_2\,{}^2\mathbf{c}_2 = d_2 \begin{bmatrix} s_1 \\ -c_1 \\ 0 \\ 1 \end{bmatrix} \tag{11.43}
$$

The velocities of mass centers are found by using the time derivatives of the positions

$$
{}^0\dot{\mathbf{c}}_1 = l_1\dot\theta_1 \begin{bmatrix} c_1 \\ s_1 \\ 0 \end{bmatrix} \qquad {}^0\dot{\mathbf{c}}_2 = \dot d_2 \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix} + d_2\dot\theta_1 \begin{bmatrix} c_1 \\ s_1 \\ 0 \end{bmatrix} \tag{11.44}
$$

Step 4: Kinetic energy

The kinetic energies of the two links are given by

$$
\begin{aligned}
K_1 &= \frac{1}{2}m_1\,\dot{\mathbf{c}}_1^T\,\dot{\mathbf{c}}_1 + \frac{1}{2}\,{}^{P1}\boldsymbol{\omega}_1^T\,{}^{P1}\mathbf{I}_1\,{}^{P1}\boldsymbol{\omega}_1 \\
&= \frac{1}{2}m_1\left(l_1^2\dot\theta_1^2\right) + \frac{1}{2}I_{yy1}\dot\theta_1^2
\end{aligned} \tag{11.45}
$$

$$\begin{aligned} K_2 &= \frac{1}{2} m_2\, \dot{\mathbf{c}}_2^T\, \dot{\mathbf{c}}_2 + \frac{1}{2}\, {}^{P2}\boldsymbol{\omega}_2^T\, {}^{P2}\mathbf{I}_2\, {}^{P2}\boldsymbol{\omega}_2 \\ &= \frac{1}{2} m_2 \left(\dot{d}_2^2 + d_2^2\dot{\theta}_1^2\right) + \frac{1}{2} I_{yy2}\dot{\theta}_1^2 \end{aligned} \tag{11.46}$$

Step 5: Potential energy

According to Figure 11.7, the gravity can be written as ${}^0\mathbf{g} = \begin{bmatrix}0 & -g & 0\end{bmatrix}^T$ where $g = 9.8$ m/s^2. Therefore, the potential energies of the two bodies are given are

$$\begin{aligned} V_1 &= -m_1\, {}^0\mathbf{g}^T\, {}^0\mathbf{c}_1 \\ &= -m_1 g l_1 \cos(\theta_1) \end{aligned} \tag{11.47}$$

$$\begin{aligned} V_2 &= -m_2\, {}^0\mathbf{g}^T\, {}^0\mathbf{c}_2 \\ &= -m_2 g d_2 \cos(\theta_1) \end{aligned} \tag{11.48}$$

Step 6: Lagrangian formula

The Lagrangian is

$$\begin{aligned} \mathcal{L} &= K - V \\ &= K_1 + K_2 - (V_1 + V_2) \end{aligned} \tag{11.49}$$

Therefore, the dynamics of the RP robot through the Lagrangian method is

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_i}\right) - \frac{\partial \mathcal{L}}{\partial \theta_i} = \tau_i \tag{11.50}$$

for actuators $i = 1, 2$. Because $i = 1$ is a revolute actuator and $i = 2$ is prismatic, τ_1 and τ_2 represents actuator torque and force, respectively. Substituting the derived Lagrangian into Equation (11.50), the dynamic equations for the RP robot are

$$\tau_1 = (m_1 l_1^2 + I_{yy1} + I_{yy2} + m_2 d_2^2)\ddot{\theta}_1 + 2m_2 d_2 \dot{\theta}_1 \dot{d}_2 + (m_1 l_1 + m_2 d_2) g \sin\theta_1 \tag{11.51}$$

$$f_2 = m_2 \ddot{d}_2 - m_2 d_2 \dot{\theta}_1^2 - m_2 g \cos\theta_1 \tag{11.52}$$

The dynamic equations found in Example 11.4 have explicit physical meanings. The force terms of the prismatic actuator defined in Equation (11.52): f_2, $m_2\ddot{d}_2$, $m_2 d_2 \dot{\theta}_1^2$, and $m_2 g \cos\theta_1$ represent the actuating force, inertia force, centrifugal force, and gravity force along the direction of d_2, respectively, as shown in Figure 11.8. One can derive this equation simply by adding all these terms. Equation (11.52) can be further rewritten in matrix form

$$\boldsymbol{\tau} = \mathbf{M}(\boldsymbol{\theta})\,\ddot{\boldsymbol{\theta}} + \mathbf{V}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) + \mathbf{G}(\boldsymbol{\theta}) \tag{11.53}$$

where $\boldsymbol{\tau} = \begin{bmatrix}\tau_1 & f_2\end{bmatrix}^T$, $\boldsymbol{\theta} = \begin{bmatrix}\theta_1 & d_2\end{bmatrix}^T$, $\dot{\boldsymbol{\theta}}$, and $\ddot{\boldsymbol{\theta}}$ are the vectors of the input torques, joint angles, joint velocities, and joint accelerations, respectively. $\mathbf{M}$, $\mathbf{V}$, and $\mathbf{G}$ are the mass matrix, the centrifugal and Coriolis term, and the gravity term, respectively, which are given by

$$\mathbf{M} = \begin{bmatrix} (m_1 l_1^2 + I_{yy1} + I_{yy2} + m_2 d_2^2) & 0 \\ 0 & m_2 \end{bmatrix} \tag{11.54}$$

$$\mathbf{V} = \begin{bmatrix} 2m_2 d_2 \dot{\theta}_1 \dot{d}_2 \\ -m_2 d_2 \dot{\theta}_1^2 \end{bmatrix} \tag{11.55}$$

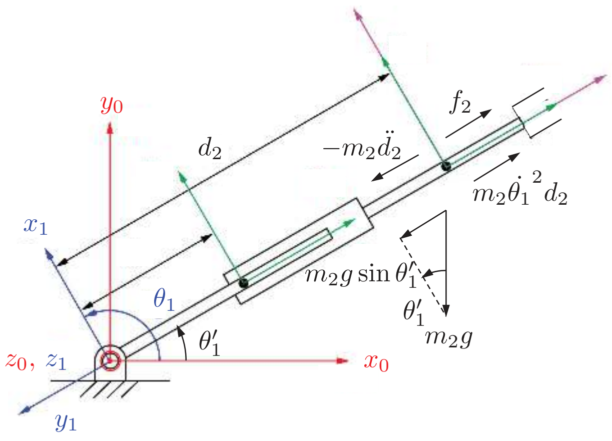


FIGURE 11.8
Force analysis of the end-effector along the prismatic joint of the PR robot.

$$\mathbf{G} = \begin{bmatrix} (m_1 l_1 + m_2 d_2) g \sin\theta_1 \\ -m_2 g \cos\theta_1 \end{bmatrix} \tag{11.56}$$

The Lagrangian method is valid for any arbitrary serial robot, such as the six-DoF robot shown in Figure 11.9. Further, the governing dynamics of a general robot can always be written in the form of Equation (11.53).

Example 11.5 (Cylindrical robot): Calculate the joint torques and forces for the cylindrical robot shown in Figure 11.10. Assume all masses are point masses and are located at the origin of each frame in homogeneous coordinates

$${}^1\mathbf{c}_1 = {}^2\mathbf{c}_2 = {}^3\mathbf{c}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{11.57}$$

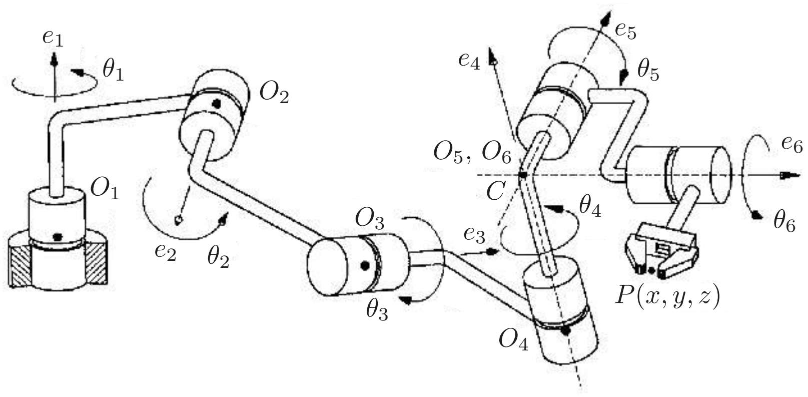
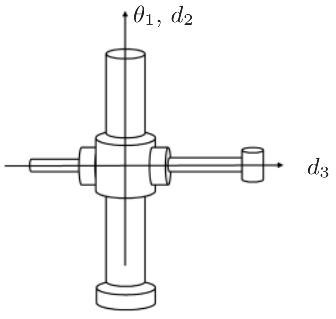


FIGURE 11.9
A six-DOF manipulator.



$$ {}^0\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.58} $$

$$ {}^1\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.59} $$

$$ {}^2\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 & 0.05 \\ 0 & 0 & -1 & -d_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.60} $$

FIGURE 11.10
A cylindrical robot with transformation matrices derived from forward kinematics.

Solution: Note that we have point masses in this example. This results in a $[\mathbf{0}]$ mass moment of inertia matrix, and therefore the kinetic energy terms to do with angular velocities equal zero. Hence, we do not need to find the angular velocities of the mass centres in this example.

Step 1:

Start by finding the mass centres relative to the base frame.

$$ {}^0\mathbf{c}_1 = {}^0\mathbf{T}_1\, {}^1\mathbf{c}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad {}^0\mathbf{c}_2 = {}^0\mathbf{T}_1\, {}^1\mathbf{T}_2\, {}^2\mathbf{c}_2 = \begin{bmatrix} 0 \\ 0 \\ d_2 \\ 1 \end{bmatrix} $$

$$ {}^0\mathbf{c}_3 = {}^0\mathbf{T}_1\, {}^1\mathbf{T}_2\, {}^2\mathbf{T}_3\, {}^3\mathbf{c}_3 = \begin{bmatrix} 0.05c_1 + d_3 s_1 \\ 0.05s_1 - d_3 c_1 \\ d_2 \\ 1 \end{bmatrix} \tag{11.61} $$

Step 2:

Take the time derivative of Equation (11.61) to find centre of mass linear velocities.

$$ {}^0\dot{\mathbf{c}}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad {}^0\dot{\mathbf{c}}_2 = \begin{bmatrix} 0 \\ 0 \\ \dot{d}_2 \end{bmatrix} \qquad {}^0\dot{\mathbf{c}}_3 = \begin{bmatrix} s_1\dot{d}_3 - 0.05s_1\dot{\theta}_1 + c_1 d_3\dot{\theta}_1 \\ -c_1\dot{d}_3 + 0.05c_1\dot{\theta}_1 + s_1 d_3\dot{\theta}_1 \\ \dot{d}_2 \end{bmatrix} \tag{11.62} $$

Step 3:

Find kinetic energies using $K_i = \frac{1}{2}m_i\, {}^0\mathbf{v}_i^T\, {}^0\mathbf{v}_i$. In this example, ${}^0\mathbf{v}_i$ is equal to the CoM velocity ${}^0\dot{\mathbf{c}}_i$ defined in Equation (11.62).

$$ \begin{aligned} K_1 &= 0 \\ K_2 &= \frac{1}{2}m_2\dot{d}_2^2 \\ K_3 &= \frac{1}{2}m_3\left(\dot{d}_2^2 + \left(s_1\dot{d}_3 - 0.05s_1\dot{\theta}_1 + c_1 d_3\dot{\theta}_1\right)^2\right. \\ &\qquad \left. + \left(-c_1\dot{d}_3 + 0.05c_1\dot{\theta}_1 + s_1 d_3\dot{\theta}_1\right)^2\right) \end{aligned} \tag{11.63} $$

Step 4:

Find potential energies using $V_i = m_i\, {}^0\mathbf{g}^T\, {}^0\mathbf{c}_i$, where ${}^0\mathbf{c}_i$ is the position of the CoM in the ground frame in Equation (11.61), and ${}^0\mathbf{g} = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T$, the direction of gravity also in the ground frame. Remember to multiply using the 3×1 non-homogeneous position vectors (ignore the trailing 1).

$$\begin{aligned} V_1 &= 0 \\ V_2 &= \frac{1}{2} m_2 \dot{d}_2^2 \\ V_3 &= \frac{1}{2} m_3 \left(\dot{d}_2^2 + \left(s_1 \dot{d}_3 - 0.05 s_1 \dot{\theta}_1 + c_1 d_3 \dot{\theta}_1 \right)^2 \right. \\ &\quad \left. + \left(-c_1 \dot{d}_3 + 0.05 c_1 \dot{\theta}_1 + s_1 d_3 \dot{\theta}_1 \right)^2 \right) \end{aligned} \tag{11.64}$$

Step 5:

Find the Lagrangian function, using $\mathcal{L}(q, \dot{q}) = \sum K(q, \dot{q}) - \sum V(q)$, where q is the joint space vector $q = \begin{bmatrix} \theta_1 & d_2 & d_3 \end{bmatrix}$.

$$\begin{aligned} \mathcal{L} &= K_1 + K_3 + K_3 - V_1 - V_2 - V_3 \\ &= \frac{1}{2} \left(\dot{d}_2 \left(m_2 + m_3 \right) + m_3 \left(s_1 \dot{d}_3 - 0.05 s_1 \dot{\theta}_1 + c_1 d_3 \dot{\theta}_1 \right)^2 \right. \\ &\quad \left. + m_3 \left(-c_1 \dot{d}_3 - 0.05 c_1 \dot{\theta}_1 + s_1 d_3 \dot{\theta}_1 \right)^2 \right) - g d_2 \left(m_2 + m_3 \right) \end{aligned} \tag{11.65}$$

Step 6:

Find joint torque and forces (effort), using $\dfrac{d}{dt} \dfrac{d\mathcal{L}}{\partial \dot{q}} - \dfrac{\partial \mathcal{L}}{\partial q}$.

Joint torque τ_1:

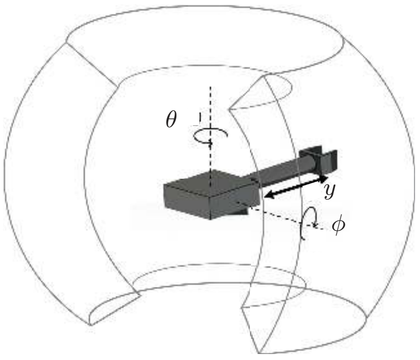
$$\begin{aligned} \tau_1 &= \left[\frac{d}{dt} \frac{d\mathcal{L}}{\partial \dot{\theta}_1} \right] - \left[\frac{\partial \mathcal{L}}{\partial \theta_1} \right] \\ &= m_3 \left[0.0025 \ddot{\theta}_1 - \ddot{d}_3 + {d_3}^2 \ddot{\theta}_1 + 2 m_3 d_3 \dot{d}_3 \dot{\theta}_1 \right] - [0] \end{aligned} \tag{11.66}$$

Joint force f_2:

$$\begin{aligned} f_2 &= \left[\frac{d}{dt} \frac{d\mathcal{L}}{\partial \dot{d}_2} \right] - \left[\frac{\partial \mathcal{L}}{\partial d_2} \right] \\ &= \ddot{d}_2 \left[m_2 + m_3 \right] + g \left[m_2 + m_3 \right] \\ &= \left(m_2 + m_3 \right) \left(g + \ddot{d}_2 \right) \end{aligned} \tag{11.67}$$

Joint force f_3:

$$\begin{aligned} f_3 &= \left[\frac{d}{dt} \frac{d\mathcal{L}}{\partial \dot{d}_3} \right] - \left[\frac{\partial \mathcal{L}}{\partial d_3} \right] \\ &= m_3 \left[\ddot{d}_3 - 0.05 \ddot{\theta}_1 \right] + \left[m_3 d_3 \dot{\theta}_1^2 \right] \\ &= m_3 \left(\ddot{d}_3 - 0.05 \ddot{\theta}_1 - d_3 \dot{\theta}_1^2 \right) \end{aligned} \tag{11.68}$$



$$ {}^{0}\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.70} $$

$$ {}^{1}\mathbf{T}_2 = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.71} $$

$$ {}^{2}\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -d_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.72} $$

FIGURE 11.11
A spherical robot with transformation matrices derived from forward kinematics.

Example 11.6 (Spherical robot): Calculate the joint torques and forces for the spherical robot shown in Figure 11.11. Assume all masses are point masses and are located at the following positions in homogeneous coordinates

$$ {}^{1}\mathbf{c}_1 = {}^{2}\mathbf{c}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad {}^{3}\mathbf{c}_3 = \begin{bmatrix} 0 \\ 0 \\ 0.25 \\ 1 \end{bmatrix} \tag{11.69} $$

Solution: Note that we have point masses in this example. This results in a $[\mathbf{0}]$ mass moment of inertia matrix, and therefore the kinetic energy terms to do with angular velocities equal zero. Hence, we do not need to find the angular velocities of the mass centres in this example.

Step 1:

Start by finding the mass centres relative to the base frame.

$$ {}^{0}\mathbf{c}_1 = {}^{0}\mathbf{T}_1\,{}^{1}\mathbf{c}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad {}^{0}\mathbf{c}_2 = {}^{0}\mathbf{T}_1\,{}^{1}\mathbf{T}_2\,{}^{2}\mathbf{c}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} $$

$$ {}^{0}\mathbf{c}_3 = {}^{0}\mathbf{T}_1\,{}^{1}\mathbf{T}_2\,{}^{2}\mathbf{T}_3\,{}^{3}\mathbf{c}_3 = \begin{bmatrix} c_1 s_2 (d_3 + \frac{1}{4}) \\ s_1 s_2 (d_3 + \frac{1}{4}) \\ -c_2 (d_3 + \frac{1}{4}) \\ 1 \end{bmatrix} \tag{11.73} $$

Step 2:

Take the time derivative of Equation (11.73) to find centre of mass linear velocities.

$$ {}^{0}\dot{\mathbf{c}}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad {}^{0}\dot{\mathbf{c}}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} $$

$$ {}^{0}\dot{\mathbf{c}}_3 = \frac{1}{4} \begin{bmatrix} -s_1 s_2 \dot{\theta}_1 + c_1 c_2 \dot{\theta}_2 + 4 c_1 s_2 \dot{d}_3 + 4 c_1 c_2 d_3 \dot{\theta}_2 - 4 s_1 s_2 d_3 \dot{\theta}_1 \\ c_1 s_2 \dot{\theta}_1 + c_1 s_2 \dot{\theta}_2 + 4 s_1 s_2 \dot{d}_3 + 4 c_1 s_2 d_3 \dot{\theta}_1 + 4 s_1 c_2 d_3 \dot{\theta}_2 \\ s_2 \dot{\theta}_2 - 4 c_2 \dot{d}_3 + s_2 d_3 \dot{\theta}_2 \end{bmatrix} \tag{11.74} $$

Step 3:

Find kinetic energies using $K_i = \frac{1}{2} m_i \, {}^0\mathbf{v}_i^T \, {}^0\mathbf{v}_i$. Again, ${}^0\mathbf{v}_i$ is equal to the CoM velocity ${}^0\dot{\mathbf{c}}_i$ defined in Equation (11.74).

$$\begin{aligned} K_1 &= K_2 = 0 \\ K_3 &= \frac{1}{32} m_3 \Bigg(\left(s_2\dot{\theta}_2 - 4c_2\dot{d}_3 + 4s_2 d_3 \dot{\theta}_2 \right)^2 \\ &\quad + \left(c_1c_2\dot{\theta}_2 + 4c_1s_2\dot{d}_3 - s_1s_2\dot{\theta}_1 + 4c_1c_2d_3\dot{\theta}_2 - 4s_1s_2d_3\dot{\theta}_1 \right)^2 \\ &\quad + \left(c_1s_2\dot{\theta}_1 + s_1c_2\dot{\theta}_2 + 4s_1s_2\dot{d}_3 + 4c_1s_2d_3\dot{\theta}_1 + 4s_1c_2d_3\dot{\theta}_2 \right)^2 \Bigg) \end{aligned} \tag{11.75}$$

Step 4:

Find potential energies using $V_i = m_i \, {}^0\mathbf{g}^T \, {}^0\mathbf{c}_i$, where ${}^0\mathbf{c}_i$ is the position of the CoM in the ground frame in Equation (11.61), and ${}^0g = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T$, the direction of gravity also in the ground frame.

$$\begin{aligned} V_1 &= V_2 = 0 \\ V_3 &= -\frac{1}{4} m_3 g c_2 \left(1 + d_3\right) \end{aligned} \tag{11.76}$$

Step 5:

Find the Lagrangian function, using $\mathcal{L}(q,\dot{q}) = \sum K(q,\dot{q}) - \sum V(q)$, where q is the joint space vector $q = \begin{bmatrix} \theta_1 & \theta_2 & d_3 \end{bmatrix}$.

$$\begin{aligned} \mathcal{L} &= K_1 + K_3 + K_3 - V_1 - V_2 - V_3 \\ &= \frac{1}{32} m_3 \Bigg(\left(s_2\dot{\theta}_2 - 4c_2\dot{d}_3 + 4s_2 d_3 \dot{\theta}_2 \right)^2 \\ &\quad + \left(c_1c_2\dot{\theta}_2 + 4c_1s_2\dot{d}_3 - s_1s_2\dot{\theta}_1 + 4c_1c_2d_3\dot{\theta}_2 - 4s_1s_2d_3\dot{\theta}_1 \right)^2 \\ &\quad + \left(c_1s_2\dot{\theta}_1 + s_1c_2\dot{\theta}_2 + 4s_1s_2\dot{d}_3 + 4c_1s_2d_3\dot{\theta}_1 + 4s_1c_2d_3\dot{\theta}_2 \right)^2 \Bigg) \\ &\quad + \frac{1}{4} m_3 g c_2 \left(1 + d_3\right) \end{aligned} \tag{11.77}$$

Step 6:

Find joint torque and forces (effort), using $\dfrac{d}{dt}\dfrac{d\mathcal{L}}{\partial \dot{q}} - \dfrac{\partial \mathcal{L}}{\partial q}$.

Joint torque τ_1:

$$\begin{aligned} \tau_1 &= \left[\frac{d}{dt} \frac{d\mathcal{L}}{\partial \dot{\theta}_1} \right] - \left[\frac{\partial \mathcal{L}}{\partial \theta_1} \right] \\ &= \frac{1}{16} \Big[m_3 {s_2}^2 (4d_3 + 1)\ddot{\theta}_1 + 2c_2\dot{\theta}_1\dot{\theta}_2 + 4s_2d_3\ddot{\theta}_1 + 8s_2\dot{d}_3\dot{\theta}_1 \\ &\qquad + 8c_2d_3\dot{\theta}_1\dot{\theta}_2 \Big] - [0] \end{aligned} \tag{11.78}$$

Joint torque τ_2:

$$\begin{aligned} \tau_2 &= \left[\frac{d}{dt} \frac{d\mathcal{L}}{\partial \dot{\theta}_2} \right] - \left[\frac{\partial \mathcal{L}}{\partial \theta_2} \right] \\ &= \frac{m_3}{16} \left(4d_3 + 1\right) \Bigg(\ddot{\theta}_2 + 4d_3\ddot{\theta}_2 + 8\dot{d}_3\dot{\theta}_2 + 4gs_2 - \frac{\sin(2\theta_2)\dot{\theta}_1^2}{2} \end{aligned}$$

$$-2\sin(2\theta_2)d_3\dot{\theta}_1^2\Big) \tag{11.79}$$

Joint force f_3:

$$\begin{aligned} f_3 &= \left[\frac{d}{dt}\frac{d\mathcal{L}}{\partial \dot{d}_3}\right] - \left[\frac{\partial \mathcal{L}}{\partial d_3}\right] \\ &= m_3\left(\frac{1}{4}\left({c_2}^2\dot{\theta}_1^2 - \dot{\theta}_1^2 - \dot{\theta}_2^2\right) + \ddot{d}_3 - d_3\dot{\theta}_1^2 - d_3\dot{\theta}_2^2\right. \\ &\quad \left. + {c_2}^2 d_3\dot{\theta}_1^2 - gc_2\right) \end{aligned} \tag{11.80}$$

11.4.1 Mass Matrix

The kinetic energy of a robot can be expressed in terms of the mass matrix

$$K = \frac{1}{2}\,\dot{\boldsymbol{\theta}}^T\,\mathbf{M}(\boldsymbol{\theta})\,\dot{\boldsymbol{\theta}} \tag{11.81}$$

Meanwhile, the kinetic energy is also given by Equation (11.29). Given the position of the CoM of Link i, we have

$$\mathbf{v}_{ci} = \frac{\mathbf{p}_{ci}}{dt} = \mathbf{J}_{ci}\,\dot{\boldsymbol{\theta}} \tag{11.82}$$

As discussed in Chapter 8, the angular velocity of Link i is given by

$${}^i\boldsymbol{\omega}_i = \sum_{j=1}^{i} {}^i\mathbf{z}_j\dot{\theta}_j = {}^i\mathbf{J}_{zi}\,\dot{\boldsymbol{\theta}} \tag{11.83}$$

where $\dot{\theta}_j = 0$ if Joint j is a prismatic joint. Substituting Equations (11.82) and (11.83) into (11.29) yields

$$\begin{aligned} K &= \sum\left(\frac{1}{2}m_i\,\dot{\boldsymbol{\theta}}^T\,\mathbf{J}_{ci}^T\,\mathbf{J}_{ci}\,\dot{\boldsymbol{\theta}} + \frac{1}{2}\,\dot{\boldsymbol{\theta}}^T\,{}^i\mathbf{J}_{zi}^T\,{}^i\mathbf{I}_i\,{}^i\mathbf{J}_{zi}\,\dot{\boldsymbol{\theta}}\right) \\ &= \frac{1}{2}\,\dot{\boldsymbol{\theta}}^T\sum\left(m_i\,\mathbf{J}_{ci}^T\,\mathbf{J}_{ci} + {}^i\mathbf{J}_{zi}^T\,{}^i\mathbf{I}_i\,{}^i\mathbf{J}_{zi}\right)\dot{\boldsymbol{\theta}} \end{aligned} \tag{11.84}$$

Comparing Equation (11.81) with (11.84) gives

$$\mathbf{M}(\boldsymbol{\theta}) = \sum\left(m_i\,\mathbf{J}_{ci}^T\,\mathbf{J}_{ci} + {}^i\mathbf{J}_{zi}^T\,{}^i\mathbf{I}_i\,{}^i\mathbf{J}_{zi}\right) \tag{11.85}$$

Example 11.7 (RP robot mass matrix): Consider the RP robot in Example 11.4. Find its mass matrix without deriving the complete dynamics equations.

Solution: According to the CoM of Links 1 and 2, $\mathbf{c}_1$ and $\mathbf{c}_2$, we have

$$\mathbf{J}_{c1} = \begin{bmatrix} l_1c_1 & 0 \\ l_1s_1 & 0 \\ 0 & 0 \end{bmatrix} \qquad \mathbf{J}_{c2} = \begin{bmatrix} d_2c_1 & s_1 \\ d_2s_1 & -c_1 \\ 0 & 0 \end{bmatrix} \tag{11.86}$$

The angular velocities are given by

$${}^1\boldsymbol{\omega}_1 = {}^1\mathbf{z}_1\dot{\theta}_1 \qquad {}^2\boldsymbol{\omega}_2 = {}^2\mathbf{z}_1\dot{\theta}_1 \tag{11.87}$$

which yields

$$\mathbf{J}_{z1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad \mathbf{J}_{z2} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{11.88}$$

Furthermore, ${}^1\mathbf{I}_1$ and ${}^2\mathbf{I}_2$ can be obtained from the inertia tensors in their principal frames by the similarity transformation. Substituting the above Jacobian matrices and inertia tensors into Equation (11.85), we have

$$\mathbf{M} = \begin{bmatrix} m_1 l_1^2 + I_{yy1} + I_{yy2} + m_2 d_2^2 & 0 \\ 0 & m_2 \end{bmatrix} \tag{11.89}$$

which is the same as the result in Example 11.4.

11.4.2 Gravity Term

The gravity term that appears in Equation (11.53) is the joint torques required to resist the gravity forces on the links of the robot. According to statics analysis, the required joint torque to resist the gravity at the CoM of Link i can be written as

$$\boldsymbol{\tau}_{gi} = -\,\mathbf{J}_{ci}^T m_i\,\mathbf{g} \tag{11.90}$$

where $\mathbf{J}_{ci}$ is the Jacobian matrix between the CoM and the joint angles, i.e., $\mathbf{J}_{ci} = \partial \mathbf{c}_i \partial \boldsymbol{\theta}$. Hence, the overall gravity term is given by

$$\mathbf{G}(\boldsymbol{\theta}) = \sum \boldsymbol{\tau}_{gi} = -\sum (\mathbf{J}_{ci}^T m_i)\,\mathbf{g} \tag{11.91}$$

Example 11.8 (RP robot gravity term): Consider the RP robot in Example 11.4. Find its gravity term in joint space without deriving the complete dynamics equations.

Solution: According to the CoM of Links 1 and 2, $\mathbf{c}_1$ and $\mathbf{c}_2$, we have

$$\mathbf{J}_{c1} = \begin{bmatrix} l_1 c_1 & 0 \\ l_1 s_1 & 0 \\ 0 & 0 \end{bmatrix} \qquad \mathbf{J}_{c2} = \begin{bmatrix} d_2 c_1 & s_1 \\ d_2 s_1 & -c_1 \\ 0 & 0 \end{bmatrix} \tag{11.92}$$

Substituting the above Jacobian matrices and $\mathbf{g} = \begin{bmatrix} 0 & -g & 0 \end{bmatrix}^T$ into Equation (11.91), we have

$$\mathbf{G} = \begin{bmatrix} (m_1 l_1 + m_2 d_2) g \sin\theta_1 \\ -m_2 g \cos\theta_1 \end{bmatrix} \tag{11.93}$$

which is exactly the same as the result in Example 11.4.

11.4.3 Friction Term

Friction forces and torques typically appear in the joint space, since they are caused by the relative movements. The friction forces are usually modelled in terms of joint positions and velocities, such as $\mathbf{F}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$. The friction forces are position-dependent because the contact forces at joints and meshing gears vary in different configurations.

11.4.4 Load Term

The load or external torques and forces acting on the body of the robot can be modelled similarly to the gravity term. The load can be mapped into joint torques, which are then resisted by the actuator torques. The required joint torques to resist the load is given by

$$\mathbf{L}(\boldsymbol{\theta}) = -\,\mathbf{J}_L^T\,\mathbf{w} \tag{11.94}$$

where $\mathbf{J}_L$ is the Jacobian matrix between the *twist* of the link where the load is applied to and the joint velocities while $\mathbf{w}$ is the load expressed in terms of *wrench*. Therefore, the complete dynamics of a robot under the assumption of rigid body can be written as

$$\boldsymbol{\tau} = \mathbf{M}(\boldsymbol{\theta})\,\ddot{\boldsymbol{\theta}} + \mathbf{V}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) + \mathbf{G}(\boldsymbol{\theta}) + \mathbf{F}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) + \mathbf{L}(\boldsymbol{\theta}) \tag{11.95}$$

11.4.5 MATLAB Example

Example M11.1 (Lagrangian dynamics): Find the torques and forces exerted on each joint for the cylindrical robot in Example 11.5 under the following conditions in SI units (radians, metres, seconds):

$$q = \begin{bmatrix} \frac{\pi}{2} \\ 0.2 \\ 0.1 \end{bmatrix} \qquad \dot{q} = \begin{bmatrix} 0.5 \\ -0.05 \\ 0.05 \end{bmatrix} \qquad \ddot{q} = \begin{bmatrix} 0 \\ -2 \\ 1 \end{bmatrix} \tag{11.96}$$

with masses

$$m_1 = 0.5 \text{ kg} \qquad m_2 = 1 \text{ kg} \qquad m_3 = 0.5 \text{ kg} \tag{11.97}$$

.

Solution: The following MATLAB code solves the problem, with the ensuing command window output.

```
% Symbolic variables
syms th1 dth1 ddth1 d2 dd2 ddd2 d3 dd3 ddd3 real

% Masses (kg)
m1 = 0.5;
m2 = 1;
m3 = 0.5;

% Gravity (m/s^2)
g = 9.81;

% Dynamic equations for each actuator
torque1 = (m3*ddth1)/400 - (m3*ddd3)/20 + m3*d3^2*ddth1 + 2*m3*d3*dd3*dth1;
force2 = (m2 + m3)*(g + ddd2);
force3 = m3*ddd3 - (m3*ddth1)/20 - m3*d3*dth1^2;

% Define effort vector
effort = [torque1; force2; force3];

% Group symbolic vectors based on time diff
vars = [th1 d2 d3];
dvars = [dth1 dd2 dd3];
ddvars = [ddth1 ddd2 ddd3];

% Create a function handle with the following inputs
% E_f(vars, dvars, ddvars)
E_f = matlabFunction(effort, 'Vars', {vars, dvars, ddvars});

% Defined conditions
vars = [pi/2, 0.2, 0.1];
dvars = [0.5, -0.05, 0.05];
ddvars = [0, -2, 1];

% Calculate effort
Effort = E_f(vars, dvars, ddvars)
Effort =

   -0.0225
   11.7150
    0.4875
```

According to the output, the effort seen in each joint is

$$\begin{bmatrix}\tau_1\\ f_2\\ f_3\end{bmatrix} = \begin{bmatrix}-0.0225 \text{ N/m}\\ 11.715 \text{ N}\\ 0.4875 \text{ N}\end{bmatrix} \tag{11.98}$$

11.5 Conclusion

In this chapter, we introduced dynamics analysis of robotic systems through Lagrangian mechanics. This is an energy-based method, where the entire dynamics of a robot is encapsulated in the *Lagrangian*. This is a function that is made up of the summation of all kinetic energy terms minus the sum of all potential energy terms. With the Lagrangian, we apply Lagrange's equation to solve for the equations of motion, which yields the analytical expression of the torques and forces exerted on the robot's actuators by these dynamic effects.

Dynamics analysis of a robotic system is vital from both a robot design perspective and in motion control. With the equations of motion, we can model the joint torques and forces exerted at the actuators, and use it directly in the robot's control system. By knowing the dynamic effects of the robot, its control system can be adequately tuned to keep the robot stable in all modes of operation. Further information on this topic is provided in Chapters 13 and 14.

11.6 Exercises

Problem 1. Derive the dynamic equations for the two-link manipulators shown in Figure 11.12 by means of the Lagrangian formulation. Assume only point masses (no inertia tensors, therefore no angular velocity component of K), m_1 and m_2, lie at the middle-point of each link.

Problem 2. Find the inertia tensor of a cylinder of homogeneous density, with respect to a frame attached to the centre of its bottom face. Hints: Convert coordinates from a Cartesian system (xyz) into a cylindrical system ($r\theta z$). Formulas for changing the integration between sets of variables will be required.

Problem 3. For the robot in Figure 11.13, obtain the dynamics of the system using the Lagrangian method. Represent the dynamic equations in state-space (i.e., matrix) form.

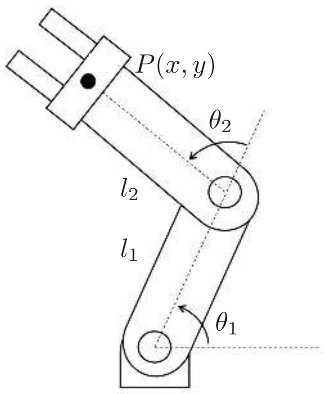


FIGURE 11.12
Planar RR robot.

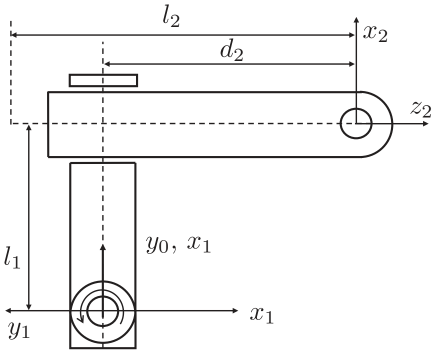


FIGURE 11.13
Planar RP robot.

Assume that the centres of masses are point masses located midway along each link. Use the following transformation matrices:

$$
{}^0\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^1\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & L_1 \\ 0 & 0 & -1 & -d_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

12

Newton-Euler Dynamics

Historically, the Newton-Euler formulation was developed in parallel with the Lagrangian formulation for robotic dynamics. Newton-Euler formulation utilises the outward kinematics propagation and inward force/torque propagation to obtain the final dynamic governing equations. During the derivation, the terms of the internal states of the robot, such as velocity, acceleration, and constraint forces, are all developed. Further, the Newton-Euler method was reported to be more efficient than the Lagrangian formulation with respect to high-DoF robotic systems. Therefore, the Newton-Euler method is equivalently important as the Lagrangian formulation in robotic analysis.

Our following discussion will be in the order of the outward kinematics propagation, the inward force/torque propagation, and the derivation of the inverse dynamics equations.

12.1 Newton's and Euler's Equations

These two equations form the basis of Newton-Euler dynamics formulation. Assuming we know the centre of mass of each link of a robot and its inertia tensor, then its dynamic properties are fully described. In order to move these links, we apply forces and torques to accelerate and decelerate them, such as through its actuators. Therefore, all forces and torques acting on this body can be reduced to a resultant force $\mathbf{f}$ acting at the mass centre and a resultant moment $\mathbf{n}$ acting on the body.

Newton's Equation

Consider a rigid body depicted in Figure 12.1. Newton's second law can be expressed as:

$$\mathbf{f} = m\,\dot{\mathbf{v}}_c \tag{12.1}$$

where m and $\mathbf{v}_c$ are the mass and the velocity at the mass centre of this body measured in $\{c\}$, respectively.

Euler's Equation

Euler's second law governing the rotation of the rigid body is given by

$$\mathbf{n} = \mathbf{I}_c\,\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}_c\,\boldsymbol{\omega} \tag{12.2}$$

where $\mathbf{I}_c$ is the moment of inertia measured in $\{c\}$, ω and $\dot{\omega}$ are angular velocity and acceleration, both also measured in $\{c\}$, and $\mathbf{n}$ is the moment acting on the rotating body. Euler's law is derived in Appendix 25.2.

DOI: 10.1201/9781003614319-12

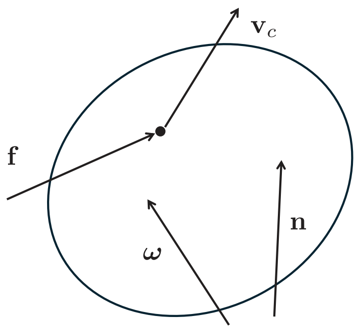


FIGURE 12.1
A rigid body under the resultant force and torque.

12.1.1 Rigid Body Rotation

In Section 11.3, we introduced the concept of the *principal axes* and its associated principal moments of inertia for a rigid body. To understand how the principal moments of inertia affects the rotation of a rigid body, we apply Euler's second law in Equation 12.2.

Example 12.1 (Rigid body rotation): Consider a rigid body with the inertia tensor measured in its principal frame ${}^{P}\mathbf{I}$. Assume the rigid body is rotating around its mass centre at a constant angular velocity $\boldsymbol{\omega}$. Find the moment required to maintain this rotation.

Solution:

Assume $\{0\}$ and $\{1\}$ as the ground frame and moving frame to the body at the mass centre. Initially, $\{0\}$ and $\{1\}$ are coincident with each other. Since the constant angular velocity $\boldsymbol{\omega}$ must be along the axis of rotation between $\{0\}$ and $\{1\}$, we must have ${}^{0}\boldsymbol{\omega} = {}^{1}\boldsymbol{\omega}$. According to the Euler formula, we have the required moment expressed in $\{1\}$ as:

$$
{}^{1}\mathbf{n} = {}^{1}\boldsymbol{\omega} \times {}^{1}\mathbf{I}_c\, {}^{1}\boldsymbol{\omega} = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} \times \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} (I_z - I_y) w_z w_y \\ (I_x - I_z) w_x w_z \\ (I_y - I_x) w_y w_x \end{bmatrix} \tag{12.3}
$$

where I_x, I_y, and I_z are the principal moments of inertia. If the body is a sphere or cube with a uniform density, $I_x = I_y = I_z$, which leads to zero moment required. If ${}^{1}\mathbf{n}$ is a nonzero vector, the direction of moment spins at the angular velocity $\boldsymbol{\omega}$, which is the source of vibration.

Example 12.2 (Wheel dynamics): Wheels on a vehicle must be dynamically balanced along its spinning axis to reduce the vibration on the road. A wheel on the balance-test machine is shown in Figure 12.2. $\{0\}$ and $\{1\}$ are the stationary frame and the moving frame attached to the wheel, respectively. Both frames are located at the geometric centre of the wheel, which can be approximated as the mass centre. Force sensors are mounted on the shafts to measure ${}^{1}\mathbf{f}_l$ and ${}^{1}\mathbf{f}_r$ when the wheel spins at a constant angular velocity along the x-axis. Find the unbalanced components of the wheel's inertia tensor.

Solution:

Ideally, $\{1\}$ should be the principal frame of the inertia tensor of a dynamically balanced wheel. In general, we assume

$$
{}^{1}\mathbf{I}_c = \begin{bmatrix} I_x & -I_{xy} & -I_{xz} \\ -I_{xy} & I_y & -I_{yz} \\ -I_{xz} & -I_{yz} & I_z \end{bmatrix} \tag{12.4}
$$

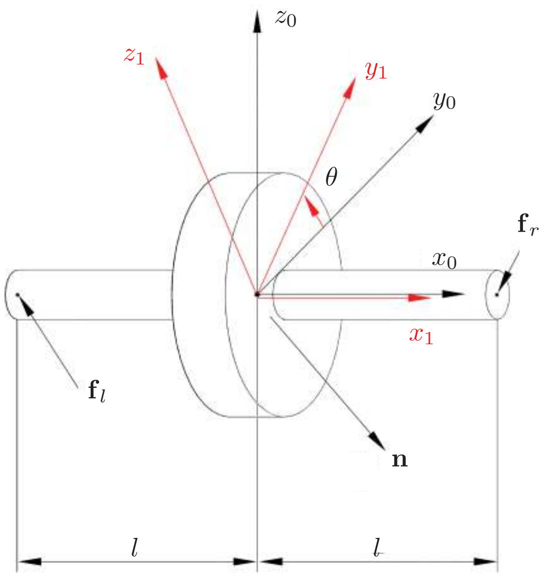


FIGURE 12.2
A wheel under the balance test.

where I_{xy} and I_{xz} are the unbalanced components we are looking for. I_{yz} does not yield vibration along the wheel axis because it represents the asymmetric mass distribution with respect to the y-z plane. According to the Euler's formula, we have

$$^1\mathbf{n} = {}^1\boldsymbol{\omega} \times {}^1\mathbf{I}_c\, {}^1\boldsymbol{\omega} = \begin{bmatrix} w_x \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} I_x & -I_{xy} & -I_{xz} \\ -I_{xy} & I_y & -I_{yz} \\ -I_{xz} & -I_{yz} & I_z \end{bmatrix} \begin{bmatrix} w_x \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ I_{xz} \\ -I_{xy} \end{bmatrix} w_x^2 \tag{12.5}$$

On the other hand, the moments around the mass centre caused by the constraint forces $^1\mathbf{f}_l$ and $^1\mathbf{f}_r$ are given by, respectively,

$$^1\mathbf{n}_l = \begin{bmatrix} -l \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} f_{lx} \\ f_{ly} \\ f_{lz} \end{bmatrix} = \begin{bmatrix} 0 \\ lf_{lz} \\ -lf_{ly} \end{bmatrix} \tag{12.6}$$

$$^1\mathbf{n}_r = \begin{bmatrix} l \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} f_{rx} \\ f_{ry} \\ f_{rz} \end{bmatrix} = \begin{bmatrix} 0 \\ -lf_{rz} \\ lf_{ry} \end{bmatrix} \tag{12.7}$$

The moment balance at the mass centre gives

$$^1\mathbf{n} + {}^1\mathbf{n}_l + {}^1\mathbf{n}_r = \begin{bmatrix} 0 \\ I_{xz}w_x^2 + lf_{lz} - lf_{rz} \\ -I_{xy}w_x^2 - lf_{ly} + lf_{ry} \end{bmatrix} = \mathbf{0} \tag{12.8}$$

which yields the solution

$$I_{xz} = (-lf_{lz} + lf_{rz})/w_x^2, \quad I_{xy} = (-lf_{ly} + lf_{ry})/w_x^2 \tag{12.9}$$

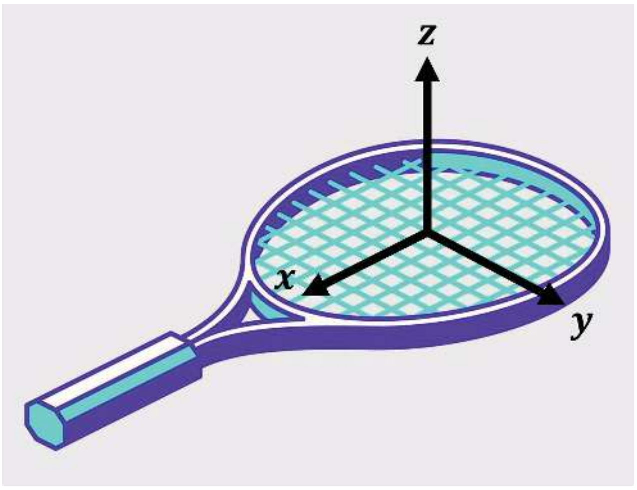


FIGURE 12.3
A tennis racquet with three principal axes.

12.1.2 Intermediate Axis Theorem

Commonly known as the *tennis racquet theorem*, this theorem states that the rotation about the first and third principal axes is stable, while that about the second, or *intermediate axis* is unstable. This theorem can be demonstrated physically with a tennis racquet. The racquet has three principal axes as shown in Figure 12.3 and that for each axis, assume its principal moment of inertia has the relation $I_1 < I_2 < I_3$. Therefore under torque-free conditions ($\mathbf{n} = 0$), the equations of motion as derived from Equation (12.2) are

$$I_1\dot{\omega}_1 = (I_2 - I_3)\,\omega_2\omega_3 \tag{12.10}$$
$$I_2\dot{\omega}_2 = (I_3 - I_1)\,\omega_3\omega_1 \tag{12.11}$$
$$I_3\dot{\omega}_3 = (I_1 - I_2)\,\omega_1\omega_2 \tag{12.12}$$

Note that one can perform this experiment on any object that has three different moments of inertia, such as a remote control, a book, or even a smartphone (please be careful!). For any rectangular object of uniform mass, $\hat{e}_1$ is parallel with the long edge, $\hat{e}_2$ is parallel with the short edge, and $\hat{e}_3$ is normal to the plane of the object.

Stable Rotation About First and Third Principal Axes

To demonstrate stable rotation about the first principal axis $\hat{e}_1$, first hold the racquet with the face horizontal to the ground. Now attempt to throw the racquet while twisting the handle, such that when one catches the racquet, its face has rotated 360°. This should be fairly easy to achieve because this is a stable axis of rotation, resulting in little residual rotation about any other axes.

To analyse the stability of this rotation, we utilise Equations (12.10)–(12.12). Under this scenario, we assume that the applied angular velocity about $\hat{e}_2$ and $\hat{e}_3$ is small, such that $\omega_2 \approx \omega_3 \approx 0$. Evaluating Equation (12.10) with these two angular velocities implies that $\dot{\omega}_1 \approx 0$. Therefore, under equilibrium conditions of this scenario, angular acceleration $\dot{\omega}_1$ is small and therefore ω_1 is relatively constant.

To analyse the stability of this rotation, take the time derivative of Equation (12.11)

$$\ddot{\omega}_2 = \frac{I_3 - I_1}{I_2}\left(\dot{\omega}_3\omega_1 + \omega_3\dot{\omega}_1\right) \tag{12.13}$$

provided that $\dot{\omega}_1 \approx 0$, substitute Equation (12.12) into the above equation leads to

$$\begin{aligned}\ddot{\omega}_2 &= \frac{I_3 - I_1}{I_2}\left[\frac{I_1 - I_2}{I_3}\omega_1\omega_2\right]\omega_1 \\ &= \frac{I_3 - I_1}{I_2}\frac{I_1 - I_2}{I_3}{\omega_1}^2\omega_2 \end{aligned} \tag{12.14}$$

the two fractions being positive and negative, respectively, which further gives the second-order ordinary differential equation

$$\ddot{\omega}_2 = -\lambda\omega_2, \qquad \lambda \in \mathbb{R}^+ \tag{12.15}$$

Due to the $I_1 < I_2 < I_3$ relation, the first fractional term in Equation (12.14) is always positive, and the second fraction term is always negative, resulting in Equation (12.15). This ODE has the general solution

$$\omega_2 = c_1 \sin(\sqrt{\lambda}t) + c_2 \cos(\sqrt{\lambda}t) \tag{12.16}$$

which represents a stable oscillatory motion over time. The conclusion is that any perturbation of ω_2 during rotation about the principal axis $\hat{e}_1$ will not cause further rotations about the $\hat{e}_2$ axis. The same relation for ω_3 can be found, starting with the time derivative Equation (12.12) instead.

Unstable Rotation About the Second Principal Axes

This time, to demonstrate unstable rotation about the second principal axis $\hat{e}_2$, first hold the racquet with the face horizontal to the ground as previously. Now throw the racquet such that it flips about the horizontal axis 360°, catching it at the handle. One may also notice that the racquet will perform a 180° flip about the handle, or first principal axis $\hat{e}_1$. This residual rotation about $\hat{e}_1$ is very difficult to suppress because rotation about the $\hat{e}_2$ axis is unstable.

We can prove its instability by again utilising Equations (12.10)–(12.12). In this scenario, we assume that the applied angular velocity about $\hat{e}_1$ and $\hat{e}_3$ is small, such that $\omega_1 \approx \omega_3 \approx 0$. Evaluating Equation (12.11) with these two angular velocities implies that $\dot{\omega}_2 \approx 0$, removing the time dependence on ω_2 (constant over time) under equilibrium. Now take the time derivative of Equation (12.10)

$$\ddot{\omega}_1 = \frac{I_2 - I_3}{I_1}\left(\dot{\omega}_2\omega_3 + \omega_2\dot{\omega}_3\right) \tag{12.17}$$

with $\dot{\omega}_3$ vanishes according to the assumption, substitute Equation (12.12) into the above equation results in

$$\begin{aligned}\ddot{\omega}_1 &= \frac{I_2 - I_3}{I_1}\left[\frac{I_1 - I_2}{I_3}\omega_1\omega_2\right]\omega_2 \\ &= \frac{I_2 - I_3}{I_1}\frac{I_1 - I_2}{I_3}{\omega_2}^2\omega_1 \end{aligned} \tag{12.18}$$

which gives the second-order ODE

$$\ddot{\omega}_1 = \lambda\omega_1, \qquad \lambda \in \mathbb{R}^+ \tag{12.19}$$

Due to the $I_1 < I_2 < I_3$ relation, all fractional terms in Equation (12.14) are always negative, yielding a positive product in Equation (12.19). This ODE has the general solution

$$\omega_1 = c_1 e^{\sqrt{\lambda} t} + c_2 e^{\sqrt{\lambda} t} \tag{12.20}$$

which shows unstable rotation due to exponential growth over time. The conclusion in this scenario is that any perturbation of ω_1 during rotation about intermediate axis $\hat{e}_2$ will cause increasing residual rotations about the $\hat{e}_1$ axis, causing the almost-unavoidable "flip" around the handle. Similarly, the same relation for instability of ω_3 can be found by repeating the above steps, but starting with the time derivative Equation (12.12) instead.

According to Equation (12.20), it is possible to achieve rotation about the intermediate axis without residual rotations in the principal axes, but only by keeping ω_1 and ω_3 as close to zero as possible. However, this is very difficult to achieve in the tennis racquet experiment.

Finally, one may notice in this experiment that the racquet tends to favour residual rotation about $\hat{e}_1$ (the handle axis) rather than $\hat{e}_3$. This is because $I_3 > I_1$, hence less torque is required to rotate the racquet about the handle than about the racquet face normal $\hat{e}_3$. This means $\dot{\omega}_1 > \dot{\omega}_3$, resulting in higher angular acceleration about $\hat{e}_1$ than $\hat{e}_3$.

12.2 Outward Propagation

Assume $\{1\}$ translates and rotates with respect to $\{0\}$ simultaneously, as shown in Figure 12.4. According to our previous discussion, the position and velocity of Point P are given by the following equations

$$ {}^0_0\mathbf{p} = {}^0_0\mathbf{O}_1 + {}^0_1\mathbf{p} = {}^0_0\mathbf{O}_1 + {}^0\mathbf{R}_1\, {}^1_1\mathbf{p} \tag{12.21}$$

$$ {}^0_0\dot{\mathbf{p}} = {}^0_0\dot{\mathbf{O}}_1 + {}^0_1\dot{\mathbf{p}} = {}^0_0\dot{\mathbf{O}}_1 + {}^0_0\boldsymbol{\omega}_1 \times {}^0_1\mathbf{p} \tag{12.22}$$

Taking the time derivative of the above velocity, we have the acceleration of Point P given by

$$ {}^0_0\ddot{\mathbf{p}} = {}^0_0\ddot{\mathbf{O}}_1 + {}^0_0\dot{\boldsymbol{\omega}}_1 \times {}^0_1\mathbf{p} + {}^0_0\boldsymbol{\omega}_1 \times {}^0_1\dot{\mathbf{p}}$$

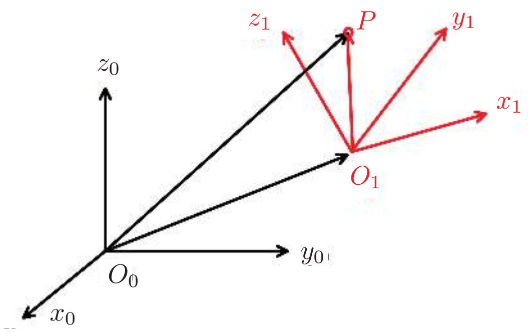


FIGURE 12.4
$\{1\}$ moves with respect to $\{0\}$.

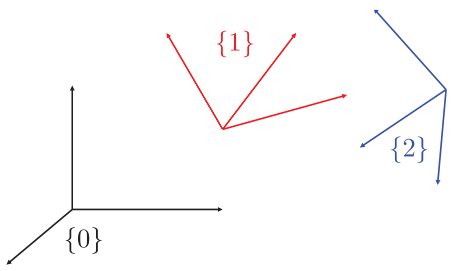


FIGURE 12.5
Frame {1} ({2}) moves with respect to {0} ({1}).

$$= {}^0_0\ddot{\mathbf{O}}_1 + {}^0_0\dot{\boldsymbol{\omega}}_1 \times {}^0_1\mathbf{p} + {}^0_0\boldsymbol{\omega}_1 \times \left({}^0_0\boldsymbol{\omega}_1 \times {}^0_1\mathbf{p}\right) \tag{12.23}$$

Consider the reference frame {0}, measure frame {0}, moving frame {1}, and Point P in (12.23) become the ground frame {0}, the frame attached to Link i-1 $\{i-1\}$, the frame attached to Link i $\{i\}$, and the origin of the frame attached to Link i+1 $\{i+1\}$, respectively. (12.23) becomes

$${}^{i-1}_{\ 0}\ddot{\mathbf{O}}_{i+1} = {}^{i-1}_{\ 0}\ddot{\mathbf{O}}_i + {}^{i-1}_{\ 0}\dot{\boldsymbol{\omega}}_i \times {}^{i-1}_{\ i}\mathbf{O}_{i+1} + {}^{i-1}_{\ 0}\boldsymbol{\omega}_i \times \left({}^{i-1}_{\ 0}\boldsymbol{\omega}_i \times {}^{i-1}_{\ i}\mathbf{O}_{i+1}\right) \tag{12.24}$$

Premultiplying ${}^{i+1}\mathbf{R}_{i-1}$ on (12.24) yields

$${}^{i+1}_{\ 0}\ddot{\mathbf{O}}_{i+1} = {}^{i+1}\mathbf{R}_i \left({}^i_0\ddot{\mathbf{O}}_i + {}^i_0\dot{\boldsymbol{\omega}}_i \times {}^i_i\mathbf{O}_{i+1} + {}^i_0\boldsymbol{\omega}_i \times \left({}^i_0\boldsymbol{\omega}_i \times {}^i_i\mathbf{O}_{i+1}\right)\right) \tag{12.25}$$

Note that the front-lower index of zero of all velocities of accelerations can be safely removed in (12.25), since the reference frames of velocity and acceleration are defined as the ground frame by default. That is

$${}^{i+1}\ddot{\mathbf{O}}_{i+1} = {}^{i+1}\mathbf{R}_i \left({}^i\ddot{\mathbf{O}}_i + {}^i\dot{\boldsymbol{\omega}}_i \times {}^i_i\mathbf{O}_{i+1} + {}^i\boldsymbol{\omega}_i \times \left({}^i\boldsymbol{\omega}_i \times {}^i_i\mathbf{O}_{i+1}\right)\right) \tag{12.26}$$

which serves as the point velocity and acceleration propagation on a robot.

Assume {1} ({2}) translates and rotates with respect to {0} ({1}) simultaneously, as shown in Figure 12.5. The relation among angular velocities is given by

$${}^0_0\boldsymbol{\omega}_2 = {}^0_0\boldsymbol{\omega}_1 + {}^0_1\boldsymbol{\omega}_2 = {}^0_0\boldsymbol{\omega}_1 + {}^0\mathbf{R}_1\, {}^1_1\boldsymbol{\omega}_2 \tag{12.27}$$

The time derivative of the above velocity yields the angular acceleration given by

$${}^0_0\dot{\boldsymbol{\omega}}_2 = {}^0_0\dot{\boldsymbol{\omega}}_1 + {}^0\dot{\mathbf{R}}_1\, {}^1_1\boldsymbol{\omega}_2 + {}^0\mathbf{R}_1\, {}^1_1\dot{\boldsymbol{\omega}}_2 = {}^0_0\dot{\boldsymbol{\omega}}_1 + {}^0_0\boldsymbol{\omega}_1 \times {}^0_1\boldsymbol{\omega}_2 + {}^0_1\dot{\boldsymbol{\omega}}_2 \tag{12.28}$$

Replace the measure frame {0}, and the moving frames {1} and {2} in (12.28) with the frame attached Link i-1 $\{i-1\}$, the frame attached to Link i $\{i\}$, and the frame attached to Link i+1 $\{i+1\}$, respectively. (12.28) becomes

$${}^{i-1}_{\ 0}\dot{\boldsymbol{\omega}}_{i+1} = {}^{i-1}_{\ 0}\dot{\boldsymbol{\omega}}_i + {}^{i-1}_{\ 0}\boldsymbol{\omega}_i \times {}^{i-1}_{\ i}\boldsymbol{\omega}_{i+1} + {}^{i-1}_{\ i}\dot{\boldsymbol{\omega}}_{i+1} \tag{12.29}$$

Premultiplying ${}^{i+1}\mathbf{R}_{i-1}$ on (12.29) yields

$${}^{i+1}_{\ 0}\dot{\boldsymbol{\omega}}_{i+1} = {}^{i+1}\mathbf{R}_i \left({}^i_0\dot{\boldsymbol{\omega}}_i + {}^i_0\boldsymbol{\omega}_i \times {}^i_i\boldsymbol{\omega}_{i+1} + {}^i_i\dot{\boldsymbol{\omega}}_{i+1}\right)$$

$$= {}^{i+1}\mathbf{R}_i \, {}^i_0\dot{\boldsymbol{\omega}}_i + \left({}^{i+1}\mathbf{R}_i \, {}^i_0\boldsymbol{\omega}_i\right) \times {}^{i+1}_{i}\boldsymbol{\omega}_{i+1} + {}^{i+1}_{i}\dot{\boldsymbol{\omega}}_{i+1}$$
$$= {}^{i+1}\mathbf{R}_i \, {}^i_0\dot{\boldsymbol{\omega}}_i + \left({}^{i+1}\mathbf{R}_i \, {}^i_0\boldsymbol{\omega}_i\right) \times \left(\dot{\theta}_{i+1}\,\mathbf{k}\right) + \ddot{\theta}_{i+1}\,\mathbf{k} \tag{12.30}$$

Recall the angular velocity propagation we derived previously as

$${}^{i+1}_{0}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i_0\boldsymbol{\omega}_i + {}^{i+1}_{i}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i_0\boldsymbol{\omega}_i + \dot{\theta}_{i+1}\,\mathbf{k} \tag{12.31}$$

Note again that the reference frames for velocity and acceleration are $\{0\}$ by default. Hence, we can write (12.31) and (12.30) as

$${}^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i + {}^{i+1}_{i}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i + \dot{\theta}_{i+1}\,\mathbf{k} \tag{12.32}$$
$${}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i\dot{\boldsymbol{\omega}}_i + \left({}^{i+1}\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i\right) \times \left(\dot{\theta}_{i+1}\,\mathbf{k}\right) + \ddot{\theta}_{i+1}\,\mathbf{k} \tag{12.33}$$

Equations (12.32) and (12.33) serve as the angular velocity and acceleration propagation formulas for a robot.

The outward velocity and acceleration propagation utilises the following equations as derived previously

$${}^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i + \dot{\theta}_{i+1}\,\mathbf{k} \tag{12.34}$$
$${}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i\dot{\boldsymbol{\omega}}_i + \left({}^{i+1}\mathbf{R}_i \, {}^i\boldsymbol{\omega}_i\right) \times \left(\dot{\theta}_{i+1}\,\mathbf{k}\right) + \ddot{\theta}_{i+1}\;\;\mathbf{k} \tag{12.35}$$
$${}^{i+1}\ddot{\mathbf{O}}_{i+1} = {}^{i+1}\mathbf{R}_i \left({}^i\ddot{\mathbf{O}}_i + {}^i\dot{\boldsymbol{\omega}}_i \times {}^i_i\mathbf{O}_{i+1} + {}^i\boldsymbol{\omega}_i \times \left({}^i\boldsymbol{\omega}_i \times {}^i_i\mathbf{O}_{i+1}\right)\right) \tag{12.36}$$

For prismatic joints, the propagation formulas are given by

$${}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} = {}^{i+1}\mathbf{R}_i \, {}^i\dot{\boldsymbol{\omega}}_i \tag{12.37}$$
$$\begin{aligned}{}^{i+1}\ddot{\mathbf{O}}_{i+1} = {}^{i+1}\mathbf{R}_i &\left({}^i\ddot{\mathbf{O}}_i + {}^i\dot{\boldsymbol{\omega}}_i \times {}^i_i\mathbf{O}_{i+1} + {}^i\boldsymbol{\omega}_i \times \left({}^i\boldsymbol{\omega}_i \times {}^i_i\mathbf{O}_{i+1}\right)\right) + \\ &2\,{}^{i+1}\boldsymbol{\omega}_{i+1} \times \dot{d}_{i+1}\,\mathbf{z} + \ddot{d}_{i+1}\,\mathbf{z}\end{aligned} \tag{12.38}$$

The acceleration of the mass centre of a link can be readily derived from (12.26) as

$${}^i\ddot{\mathbf{p}}_{ci} = {}^i\ddot{\mathbf{O}}_i + {}^i\dot{\boldsymbol{\omega}}_i \times {}^i_i\mathbf{p}_{ci} + {}^i\boldsymbol{\omega}_i \times \left({}^i\boldsymbol{\omega}_i \times {}^i_i\mathbf{p}_{ci}\right) \tag{12.39}$$

where ${}_i\mathbf{p}_{ci}$ is the position vector from the origin of $\{i\}$ to the mass centre of Link i. The boundary conditions of angular velocity and acceleration for the outward propagation are given by

$${}^0\boldsymbol{\omega}_0 = {}^0\dot{\boldsymbol{\omega}}_0 = \mathbf{0} \tag{12.40}$$

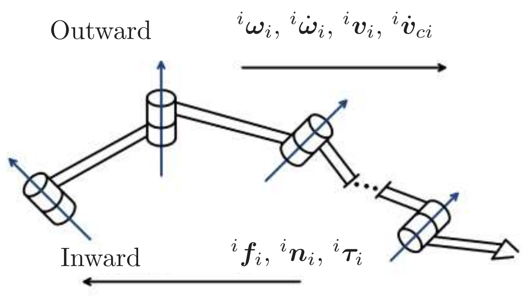

Outward
${}^i\boldsymbol{\omega}_i,\ {}^i\dot{\boldsymbol{\omega}}_i,\ {}^i\boldsymbol{v}_i,\ {}^i\dot{\boldsymbol{v}}_{ci}$
Inward
${}^i\boldsymbol{f}_i,\ {}^i\boldsymbol{n}_i,\ {}^i\boldsymbol{\tau}_i$

FIGURE 12.6
Propagation of velocities, accelerations, forces, and torques of a serial robotic manipulator.

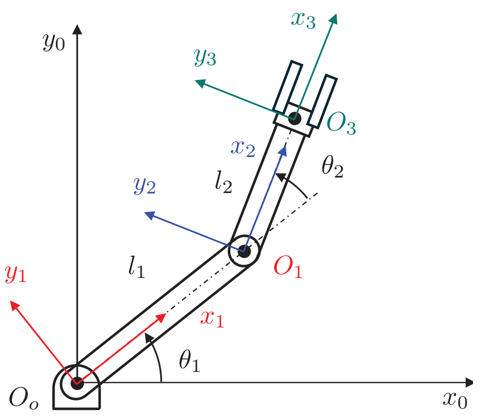


FIGURE 12.7
2R robot.

Regarding the point acceleration for the outward propagation, we adopt the boundary condition

$$ {}^{0}_{-1}\ddot{\mathbf{O}}_0 = -\,\mathbf{g} \tag{12.41}$$

where $\{-1\}$ is an artificial frame being absolutely stationary and parallel to $\{0\}$ while $\mathbf{g}$ is the vector of gravity. We can imagine that the robot is positioned in an elevator $\{0\}$ which is translating with respect to $\{-1\}$ with an acceleration of $-\,\mathbf{g}$ in a zero-gravity environment. The purpose of this boundary condition is to simplify the force and torque computations by excluding gravity terms in the ensuing analysis because the boundary condition automatically considers gravity.

Example 12.3 (2R robot kinetics):

Consider a 2R robot shown in Figure 12.7, where the mass centres are at the tips of the links. Find the angular velocity, angular acceleration, and acceleration of each link using the outward propagation. Assume $l_1 = l_2 = 1$ m.

Solution:

Step 1: Initialisation

The mass centres of the two links are given by

$$ {}^{1}\mathbf{p}_{c1} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T \qquad {}^{2}\mathbf{p}_{c2} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T \tag{12.42}$$

The boundary conditions in Frame 0 are

$$ {}^{0}\boldsymbol{\omega}_0 = {}^{0}\dot{\boldsymbol{\omega}}_0 = \mathbf{0}, \qquad {}^{0}\ddot{\mathbf{O}}_0 = \begin{bmatrix}0 & g & 0\end{bmatrix}^T \tag{12.43}$$

where $g = 9.8$ m/s^2. It is known from previous examples that the transform matrices of the frames $(i = 1, 2)$ are

$$ {}^{i-1}\mathbf{R}_i = \begin{bmatrix}\cos\theta_i & -\sin\theta_i & 0 \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1\end{bmatrix} \tag{12.44}$$

Step 2: Frame 1

According to the propagation formula and the conditions on Frame 0, we have angular velocity and acceleration, point acceleration, and the acceleration at the mass centre of Link 1:

$$ {}^1\boldsymbol{\omega}_1 = {}^1\mathbf{R}_0\, {}^0\boldsymbol{\omega}_0 + \dot{\theta}_1\, {}^1\mathbf{z}_1 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \tag{12.45} $$

$$ {}^1\dot{\boldsymbol{\omega}}_1 = {}^1\mathbf{R}_0\, {}^0\dot{\boldsymbol{\omega}}_0 + \left({}^1\mathbf{R}_0\, {}^0\boldsymbol{\omega}_0\right) \times \left(\dot{\theta}_1\, {}^1\mathbf{z}_1\right) + \ddot{\theta}_1\, {}^1\mathbf{z}_1 = \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_1 \end{bmatrix} \tag{12.46} $$

$$ \begin{aligned} {}^1\ddot{\mathbf{O}}_1 &= {}^1\mathbf{R}_0 \left({}^0\ddot{\mathbf{O}}_0 + {}^0\dot{\boldsymbol{\omega}}_0 \times {}^0_0\mathbf{O}_1 + {}^0\boldsymbol{\omega}_0 \times \left({}^0\boldsymbol{\omega}_0 \times {}^0_0\mathbf{O}_1\right)\right) \\ &= \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ g \\ 0 \end{bmatrix} = \begin{bmatrix} gs_1 \\ gc_1 \\ 0 \end{bmatrix} \end{aligned} \tag{12.47} $$

$$ {}^1\ddot{\mathbf{p}}_{c1} = {}^1\ddot{\mathbf{O}}_1 + {}^1\dot{\boldsymbol{\omega}}_1 \times {}^1_1\mathbf{p}_{c1} + {}^1\boldsymbol{\omega}_1 \times \left({}^1\boldsymbol{\omega}_1 \times {}^1_1\mathbf{p}_{c1}\right) = \begin{bmatrix} -\dot{\theta}_1^2 + gs_1 \\ \ddot{\theta}_1 + gc_1 \\ 0 \end{bmatrix} \tag{12.48} $$

Step 3: Frame 2

According to the results in Frame 1, we have the angular velocity, angular acceleration, point acceleration, and acceleration at the mass centre of Link 2, given by, respectively,

$$ {}^2\boldsymbol{\omega}_2 = {}^2\mathbf{R}_1\, {}^1\boldsymbol{\omega}_1 + \dot{\theta}_2\, {}^2\mathbf{z}_2 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix} \tag{12.49} $$

$$ {}^2\dot{\boldsymbol{\omega}}_2 = {}^2\mathbf{R}_1\, {}^1\dot{\boldsymbol{\omega}}_1 + \left({}^2\mathbf{R}_1\, {}^1\boldsymbol{\omega}_1\right) \times \left(\dot{\theta}_2\, {}^2\mathbf{z}_2\right) + \ddot{\theta}_2\, {}^2\mathbf{z}_2 = \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_1 + \ddot{\theta}_2 \end{bmatrix} \tag{12.50} $$

$$ \begin{aligned} {}^2\ddot{\mathbf{O}}_2 &= {}^2\mathbf{R}_1 \left({}^1\ddot{\mathbf{O}}_1 + {}^1\dot{\boldsymbol{\omega}}_1 \times {}^1_1\mathbf{O}_2 + {}^1\boldsymbol{\omega}_1 \times \left({}^1\boldsymbol{\omega}_1 \times {}^1_1\mathbf{O}_2\right)\right) \\ &= \begin{bmatrix} \ddot{\theta}_1 s_2 - \dot{\theta}_1^2 c_2 + gs_{12} \\ \ddot{\theta}_1 c_2 + \dot{\theta}_1^2 s_2 + gc_{12} \\ 0 \end{bmatrix} \end{aligned} \tag{12.51} $$

$$ \begin{aligned} {}^2\ddot{\mathbf{p}}_{c2} &= {}^2\ddot{\mathbf{O}}_2 + {}^2\dot{\boldsymbol{\omega}}_2 \times {}^2_2\mathbf{p}_{c2} + {}^2\boldsymbol{\omega}_2 \times \left({}^2\boldsymbol{\omega}_2 \times {}^2_2\mathbf{p}_{c2}\right) \\ &= \begin{bmatrix} -\left(\dot{\theta}_1 + \dot{\theta}_2\right)^2 + \ddot{\theta}_1 s_2 - \dot{\theta}_1^2 c_2 + gs_{12} \\ \left(\ddot{\theta}_1 + \ddot{\theta}_2\right) + \ddot{\theta}_1 c_2 + \dot{\theta}_1^2 s_2 + gc_{12} \\ 0 \end{bmatrix} \end{aligned} \tag{12.52} $$

12.3 Inward Propagation

The essential problem in the inward force/torque propagation is to derive the equilibrium equation of an isolated link in a robotic manipulator. Referencing the robotic in Figure 12.6, the force and torque equilibrium equations are given by

$$ {}^{i}\mathbf{f}_i + {}^{i}\mathbf{f}_{ci} - {}^{i}\mathbf{f}_{i+1} = 0 \tag{12.53}$$

$$ {}^{i}\mathbf{n}_i + {}^{i}\mathbf{n}_{ci} - {}^{i}\mathbf{n}_{i+1} + {}^{i}\mathbf{p}_{ci} \times {}^{i}\mathbf{f}_{ci} + {}^{i}\mathbf{O}_{i+1} \times \left(-\,{}^{i}\mathbf{f}_{i+1}\right) = 0 \tag{12.54}$$

where $\mathbf{f}_i$, $\mathbf{f}_{i+1}$, $\mathbf{n}_i$, $\mathbf{n}_{i+1}$, $\mathbf{f}_{ci}$, and $\mathbf{n}_{ci}$ are the joint force by Link $i-1$ on Link i, joint force by Link i on Link $i+1$, joint torque by Link $i-1$ on Link i, joint torque by Link i on Link $i+1$, inertia force at mass centre, and inertia moment, respectively. All forces and torques are expressed in $\{i\}$. With proper transformations, the equilibrium equations can be written in the form of propagation

$$ {}^{i}\mathbf{f}_i = -\,{}^{i}\mathbf{f}_{ci} + {}^{i}\mathbf{R}_{i+1}\,{}^{i+1}\mathbf{f}_{i+1} \tag{12.55}$$

$$ {}^{i}\mathbf{n}_i = -\,{}^{i}\mathbf{n}_{ci} + {}^{i}\mathbf{R}_{i+1}\,{}^{i+1}\mathbf{n}_{i+1} - {}^{i}\mathbf{p}_{ci} \times {}^{i}\mathbf{f}_{ci} + {}^{i}\mathbf{O}_{i+1} \times {}^{i}\mathbf{R}_{i+1}\,{}^{i+1}\mathbf{f}_{i+1} \tag{12.56}$$

where the inertia force and torque are obtained by using Newton and Euler equations

$$ {}^{i}\mathbf{f}_{ci} = -m_i\,{}^{i}\ddot{\mathbf{p}}_{ci} \tag{12.57}$$

$$ {}^{i}\mathbf{n}_{ci} = -\,{}^{i}\mathbf{I}_{ci}\,{}^{i}\dot{\boldsymbol{\omega}}_i - {}^{i}\boldsymbol{\omega}_i \times {}^{i}\mathbf{I}_{ci}\,{}^{i}\boldsymbol{\omega}_i \tag{12.58}$$

(12.55) and (12.56) are utilised to obtain the joint forces ${}^{i}\mathbf{f}_i$ and torques ${}^{i}\mathbf{n}_i$ from the end-effector to the base, as shown in Figure 12.6. For a revolute joint, the joint torque consists of the joint constraint torque and motor input torque. The latter is along the axis of rotation, which is expressed as $\mathbf{k}$ in $\{i\}$. Therefore, the motor's torque input is given by

$$ \tau_i = {}^{i}\mathbf{n}_i^{T}\,\mathbf{k} \tag{12.59}$$

Example 12.4 (Dynamic propagation): Consider a 2R robot shown in Figure 12.7 in Example 12.3. Derive the dynamic equations by propagation. Assume point mass for the links.

Solution:
The outward kinematics propagation was done in Example 12.3. We utilise the results in Example 12.3 to conduct the inward propagation here.

Step 1: Inertia forces and torques

According to the kinematics, the inertia forces of Link 1 and Link 2 are

$$ {}^{1}\mathbf{f}_{c1} = -m_1\,{}^{1}\ddot{\mathbf{p}}_{c1} = -\begin{bmatrix} -m_1\dot{\theta}_1^2 + m_1 g s_1 \\ m_1\ddot{\theta}_1 + m_1 g c_1 \\ 0 \end{bmatrix} \tag{12.60}$$

$$ {}^{2}\mathbf{f}_{c2} = -m_2\,{}^{2}\ddot{\mathbf{p}}_{c2} = -\begin{bmatrix} -m_2(\dot{\theta}_1+\dot{\theta}_2)^2 + m_2\ddot{\theta}_1 s_2 - m_2\dot{\theta}_1^2 c_2 + m_2 g s_{12} \\ m_2(\ddot{\theta}_1+\ddot{\theta}_2) + m_2\ddot{\theta}_1 c_2 + m_2\dot{\theta}_1^2 s_2 + m_2 g c_{12} \\ 0 \end{bmatrix} \tag{12.61}$$

and the torques for Link 1 and 2 are

$$ {}^{1}\mathbf{n}_{c1} = -\,{}^{1}\mathbf{I}_{c1}\,{}^{1}\dot{\boldsymbol{\omega}}_1 - {}^{1}\boldsymbol{\omega}_1 \times {}^{1}\mathbf{I}_{c1}\,{}^{1}\boldsymbol{\omega}_1 = \mathbf{0} \tag{12.62}$$

$$ {}^{2}\mathbf{n}_{c2} = -\,{}^{2}\mathbf{I}_{c2}\,{}^{2}\dot{\boldsymbol{\omega}}_2 - {}^{2}\boldsymbol{\omega}_2 \times {}^{2}\mathbf{I}_{c2}\,{}^{2}\boldsymbol{\omega}_2 = \mathbf{0} \tag{12.63}$$

where the inertia tensors are zero matrices due to the assumption of point masses.

Step 2: Link 2

Force and torque propagations are given by

$$\begin{aligned} {}^2\mathbf{f}_2 &= -\,{}^2\mathbf{f}_{c2} + {}^2\mathbf{R}_3\,{}^3\mathbf{f}_3 = -\,{}^2\mathbf{f}_{c2} \\ &= \begin{bmatrix} -m_2\left(\dot\theta_1+\dot\theta_2\right)^2 + m_2\ddot\theta_1 s_2 - m_2\dot\theta_1^2 c_2 + m_2 g s_{12} \\ m_2\left(\ddot\theta_1+\ddot\theta_2\right) + m_2\ddot\theta_1 c_2 + m_2\dot\theta_1^2 s_2 + m_2 g c_{12} \\ 0 \end{bmatrix} \end{aligned} \tag{12.64}$$

$$\begin{aligned} {}^2\mathbf{n}_2 &= -\,{}^2\mathbf{n}_{c2} + {}^2\mathbf{R}_3\,{}^3\mathbf{n}_3 - {}^2\mathbf{p}_{c2}\times{}^2\mathbf{f}_{c2} + {}^2\mathbf{O}_3\times{}^2\mathbf{R}_3\,{}^3\mathbf{f}_3 \\ &= \begin{bmatrix} 0 \\ 0 \\ m_2\left(\left(\ddot\theta_1+\ddot\theta_2\right) + \ddot\theta_1 c_2 + \dot\theta_1^2 s_2 + g c_{12}\right) \end{bmatrix} \end{aligned} \tag{12.65}$$

Therefore, the motor's torque input to Link 2 is given by

$$\tau_2 = {}^2\mathbf{n}_2^T\,{}^2\mathbf{z}_2 = m_2\left(\left(\ddot\theta_1+\ddot\theta_2\right) + \ddot\theta_1 c_2 + \dot\theta_1^2 s_2 + g c_{12}\right) \tag{12.66}$$

Step 3: Link 1

Since Link 1 is the first link, only torque propagation is required here, i.e.,

$${}^1\mathbf{n}_1 = -\,{}^1\mathbf{n}_{c1} + {}^1\mathbf{R}_2\,{}^2\mathbf{n}_2 - {}^1\mathbf{p}_{c1}\times{}^1\mathbf{f}_{c1} + {}^1\mathbf{O}_2\times{}^1\mathbf{R}_2\,{}^2\mathbf{f}_2 \tag{12.67}$$

Therefore, the motor's torque input to Link 1 is

$$\begin{aligned} \tau_1 &= {}^1\mathbf{n}_1^T\,{}^1\mathbf{z}_1 \\ &= m_2^2\left(\ddot\theta_1+\ddot\theta_2\right) + m_2 c_2\left(2\ddot\theta_1+\ddot\theta_2\right) + (m_1+m_2)\ddot\theta_1 - \\ &\qquad m_2\dot\theta_2^2 s_2 - 2m_2\dot\theta_1\dot\theta_2 s_2 + m_2 g c_{12} + (m_1+m_2) g c_1 \end{aligned} \tag{12.68}$$

Step 4: Final Dynamics

The final dynamic equations are

$$\begin{aligned} \tau_1 &= m_2\left(\ddot\theta_1+\ddot\theta_2\right) + m_2 c_2\left(2\ddot\theta_1+\ddot\theta_2\right) + (m_1+m_2)\ddot\theta_1 - \\ &\qquad m_2\dot\theta_2^2 s_2 - 2m_2\dot\theta_1\dot\theta_2 s_2 + m_2 g c_{12} + (m_1+m_2) g c_1 \end{aligned} \tag{12.69}$$

$$\tau_2 = m_2\left(\left(\ddot\theta_1+\ddot\theta_2\right) + \ddot\theta_1 c_2 + \dot\theta_1^2 s_2 + g c_{12}\right) \tag{12.70}$$

which can be written in the matrix form:

$$\boldsymbol{\tau} = \mathbf{M}(\boldsymbol{\theta})\,\ddot{\boldsymbol{\theta}} + \mathbf{V}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) + \mathbf{G}(\boldsymbol{\theta}) \tag{12.71}$$

where $\mathbf{M}(\boldsymbol{\theta})$ is the mass matrix, $\mathbf{V}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}})$ is the centrifugal and Coriolis term, and $\mathbf{G}(\boldsymbol{\theta})$ is the gravity term expressed in the joint space. Leave it as an exercise to find all these terms in this example.

12.4 Procedure

The procedure to derive the dynamic equations governing a general robotic manipulator such as Figure 12.8 by dynamic propagation is summarised here.

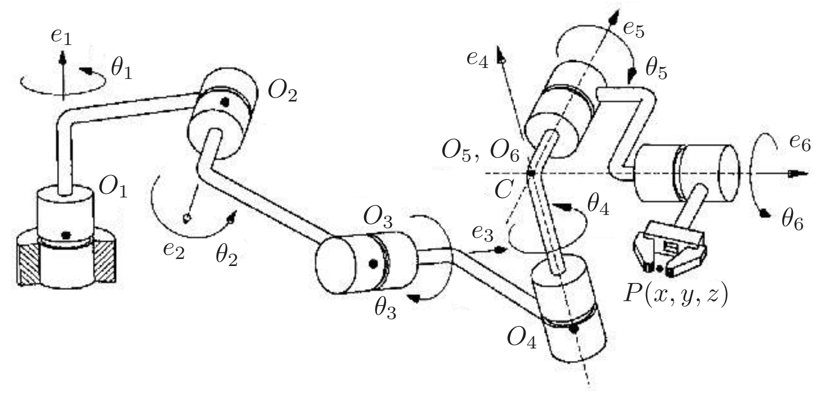


FIGURE 12.8
A general robotic manipulator.

Step 1

Prepare all quantities for each i-th link, for an n-link manipulator:

Inertia tensor of each link in its principal frame	${}^{P_i}\mathbf{I}_i$
Mass of the link	m_i
Location of the link's mass centre	${}^{i}\mathbf{c}_i$
Force applied by the end-effector	${}^{n}\mathbf{f}_n$
Torque applied by the end-effector	${}^{n}\mathbf{n}_n$
Transformation matrices via forward kinematics	${}^{i}\mathbf{T}_{i+1}$

Step 2

Perform outward propagation of link kinematics, where Frame 0 is stationary

$$ {}^0\boldsymbol{\omega}_0 = {}^0\dot{\boldsymbol{\omega}}_0 = \mathbf{0} \tag{12.72}$$

but has a nonzero linear acceleration due to gravity

$$ {}^0\ddot{\mathbf{O}}_0 = -\,\mathbf{g} \tag{12.73}$$

Then, for each subsequent Frame $i+1$, propagate the link kinematics with the generalised equations (for revolute or prismatic joints):

$$ {}^{i+1}_{\ 0}\boldsymbol{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i\, {}^{i}_{0}\boldsymbol{\omega}_i + \dot{\theta}_{i+1}\ \mathbf{k} \tag{12.74}$$

$$ {}^{i+1}_{\ 0}\dot{\boldsymbol{\omega}}_{i+1} = {}^{i+1}\mathbf{R}_i\, {}^{i}_{0}\dot{\boldsymbol{\omega}}_i + \left({}^{i+1}\mathbf{R}_i\, {}^{i}_{0}\boldsymbol{\omega}_i \times \dot{\theta}_{i+1}\ \mathbf{k}\right) + \ddot{\theta}_{i+1}\ \mathbf{k} \tag{12.75}$$

$$ {}^{i+1}_{\ 0}\ddot{\mathbf{O}}_{i+1} = {}^{i+1}\mathbf{R}_i \left({}^{i}_{0}\ddot{\mathbf{O}}_i + {}^{i}_{0}\dot{\boldsymbol{\omega}}_i \times^{i}_{0} \mathbf{O}_i + {}^{i}_{0}\boldsymbol{\omega}_i \times \left({}^{i}_{0}\boldsymbol{\omega}_i \times {}^{i}_{0}\mathbf{O}_i\right)\right) \tag{12.76}$$

Step 3

Inertia force and torque on each Link i from its own mass can now be found with known link velocities and accelerations:

$$ {}^{i}\mathbf{f}_{ci} = -m_i\, {}^{i}\ddot{\mathbf{p}}_{ci} \tag{12.77}$$

$$ {}^{i}\mathbf{n}_{ci} = -\,{}^{i}\mathbf{I}_{ci}\,{}^{i}\dot{\boldsymbol{\omega}}_{i} - {}^{i}\boldsymbol{\omega}_{i} \times {}^{i}\mathbf{I}_{ci}\,{}^{i}\boldsymbol{\omega}_{i} \tag{12.78}$$

where the point acceleration at the mass centre is

$$ {}^{i}_{0}\ddot{\mathbf{p}}_{ci} = {}^{i}_{0}\ddot{\mathbf{O}}_{i} + {}^{i}_{0}\dot{\boldsymbol{\omega}}_{i} \times {}^{i}_{i}\mathbf{p}_{ci} + {}^{i}_{0}\boldsymbol{\omega}_{i} \times \left({}^{i}_{0}\boldsymbol{\omega}_{i} \times {}^{i}_{i}\mathbf{p}_{ci}\right) \tag{12.79}$$

Step 4

Perform inward propagation to find the force and torque exerted by each link, where the n-th frame represents force and torque exerted by the end-effector to the environment. The equations for force and torque, respectively are

$$ {}^{i}\mathbf{f}_{i} = -\,{}^{i}\mathbf{f}_{ci} + {}^{i}\mathbf{R}_{i+1}\,{}^{i+1}\mathbf{f}_{i+1} \tag{12.80}$$

$$ {}^{i}\mathbf{n}_{i} = -\,{}^{i}\mathbf{n}_{ci} + {}^{i}\mathbf{R}_{i+1}\,{}^{i+1}\mathbf{n}_{i+1} - {}^{i}\mathbf{p}_{ci} \times {}^{i}\mathbf{f}_{ci} + {}^{i}\mathbf{O}_{i+1} \times {}^{i}\mathbf{R}_{i+1}\,{}^{i+1}\mathbf{f}_{i+1} \tag{12.81}$$

Step 5

The resulting servo effort is

$$ \tau_i = {}^{i}\mathbf{n}_i^T\ \mathbf{k} \tag{12.82}$$

for revolute joints, or

$$ f_i = {}^{i}\mathbf{f}_i^T\ \mathbf{k} \tag{12.83}$$

for prismatic actuators, where $\mathbf{k}$ is the axis of actuation, usually $\mathbf{z}$ as defined in DH notation.

Effects of friction

Two types of friction can affect servos. Viscous friction τ_{fv} for revolute actuators is modelled as

$$ \tau_{fv} = \nu\dot{\theta} \tag{12.84}$$

and Coulomb friction τ_{fc} for prismatic actuators, which is modelled by

$$ \tau_{fc} = c\,\mathrm{sgn}(\dot{\theta}) \tag{12.85}$$

Constants ν and c, are the viscous friction and Coulomb friction coefficients, respectively. If the friction is modelled, the final dynamic equations become

$$ \boldsymbol{\tau} = \mathbf{M}(\boldsymbol{\theta})\,\ddot{\boldsymbol{\theta}} + \mathbf{V}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) + \mathbf{G}(\boldsymbol{\theta}) + \mathbf{F}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \tag{12.86}$$

where $\mathbf{F}$ is the term of frictions projected into the joint space. Equation (12.86) is called the inverse dynamics of the given robot. If the desired trajectory is known, substituting all information of joint positions, velocities, and accelerations into the inverse dynamics, the desired torques can be computed. If the dynamics model is perfect, the torque commands according to the computed torques will drive the robot to follow the desired trajectory precisely. Such a strategy is called the open-loop control. Apparently, it is naive because the derived dynamic model cannot be perfect. However, the inverse dynamics serves as the building block for more advanced control strategies.

Equation (12.86) can be written differently as

$$ \ddot{\boldsymbol{\theta}} = \mathbf{M}^{-1}(\boldsymbol{\theta})\left[\boldsymbol{\tau} - \mathbf{V}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) - \mathbf{G}(\boldsymbol{\theta}) - \mathbf{F}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\right] \tag{12.87}$$

which is called the direct dynamics of the given robot. It can be used to generate a simulation according to a set of torque inputs from initial conditions $\boldsymbol{\theta}(0)$ and $\dot{\boldsymbol{\theta}}(0)$, to predict the motion of the robotic system offline. Euler integration formulas can be applied, such as

$$\dot{\boldsymbol{\theta}}(t+\Delta t)=\dot{\boldsymbol{\theta}}(t)+\ddot{\boldsymbol{\theta}}(t)\Delta t \tag{12.88}$$

$$\boldsymbol{\theta}(t+\Delta t)=\boldsymbol{\theta}(t)+\dot{\boldsymbol{\theta}}(t)\Delta t+\frac{1}{2}\,\ddot{\boldsymbol{\theta}}(t)\Delta t^2 \tag{12.89}$$

where Δt is the time step. Other numerical integration techniques such as midpoint, Heun's, or Runge-Kutta methods can be applied for better accuracy. Furthermore, the simulation will depend on the selection of the time step. Generally, a smaller time step yields higher accuracy, while the computational cost is increased.

12.5 Twist, Wrench, and 6×6 Transformation Matrix

So far in the textbook, we always discussed the velocity- and force-domain information with two equations, i.e., linear and angular velocities, respectively, for the velocity domain, and force and moment, respectively, for the force domain. On the other hand, it may be more elegant and beneficial to express the transformation with a unified equation. This could be achieved by adopting the concept of twist (velocity-domain) and wrench (force-domain), whose transformations are presented in the ensuing subsections.

12.5.1 Transformation of Twist

Consider two arbitrary frames, {1} and {2}, where {1} is an arbitrary reference frame, and {2} is attached to a point on a moving rigid body, such as a link of a robotic manipulator. A twist is used to fully describe the velocity of the rigid body, which comprises angular and linear components. The angular velocity applies to the entire rigid body. The linear velocity is the velocity of a point that is attached to the rigid body and instantaneously coincident with the origin of the frame, where the twist is represented.

Following such a definition, the twist is represented in {2} as

$${}^2\mathbf{V}_2=\begin{bmatrix}{}^2\boldsymbol{\omega}_2\\ {}^2\mathbf{v}_2\end{bmatrix} \tag{12.90}$$

where ${}^2v_2$ is the linear velocity of the origin of {2}. To represent the twist in {1}, the linear velocity of Point 1', which is attached to the rigid body and instantaneously coincident with the origin of {1} is used. Such linear velocity, represented in {2}, is

$$\begin{aligned}{}^2\mathbf{v}_{1'}&={}^2\mathbf{v}_2+{}^2\boldsymbol{\omega}_2\times{}^2_2\mathbf{r}_{1'}={}^2\mathbf{v}_2+{}^2\boldsymbol{\omega}_2\times{}^2_2\mathbf{r}_1\\ &={}^2\mathbf{v}_2+{}^2_1\mathbf{r}_2\times{}^2\boldsymbol{\omega}_2\end{aligned} \tag{12.91}$$

where the $\mathbf{r}$ terms are position vectors. ${}_2\mathbf{r}_{1'}$ is identical to ${}_2\mathbf{r}_1$, as Point 1' is instantaneously coincident with Point 1 (origin of {1}). Furthermore, representing such linear velocity in {1} is

$${}^1\mathbf{v}_{1'}={}^1\mathbf{R}_2\,{}^2\mathbf{v}_2+{}^1\mathbf{R}_2\left({}^2_1\mathbf{r}_2\times{}^2\boldsymbol{\omega}_2\right) \tag{12.92}$$

Angular velocity can be readily transformed with a rotation matrix

$${}^1\boldsymbol{\omega}_2={}^1\mathbf{R}_2\,{}^2\boldsymbol{\omega}_2 \tag{12.93}$$

Therefore, twist $\mathbf{V}_2$ represented in $\{1\}$ is written as

$$^1\mathbf{V}_2 = \begin{bmatrix} ^1\boldsymbol{\omega}_2 \\ ^1\mathbf{v}_{1'} \end{bmatrix} = \begin{bmatrix} ^1\mathbf{R}_2\,{}^2\boldsymbol{\omega}_2 \\ ^1\mathbf{R}_2\,{}^2\mathbf{v}_2 + {}^1\mathbf{R}_2\left({}^2_1\mathbf{r}_2 \times {}^2\boldsymbol{\omega}_2\right) \end{bmatrix} \tag{12.94}$$

Rearranging the expression into matrix form yields

$$^1\mathbf{V}_2 = \begin{bmatrix} ^1\boldsymbol{\omega}_2 \\ ^1\mathbf{v}_{1'} \end{bmatrix} = \begin{bmatrix} ^1\mathbf{R}_2 & 0 \\ ^1\mathbf{R}_2\,{}^2_1\mathbf{r}_2\times & {}^1\mathbf{R}_2 \end{bmatrix} \begin{bmatrix} ^2\boldsymbol{\omega}_2 \\ ^2\mathbf{v}_2 \end{bmatrix} = {}^1\mathbf{X}_2\,{}^2\mathbf{V}_2 \tag{12.95}$$

Alternatively, given

$$\begin{aligned} ^1\mathbf{R}_2\,{}^2_1\mathbf{r}_2\times &= {}^1\mathbf{R}_2\left({}^2\mathbf{R}_1\,{}^1_1\mathbf{r}_2\right)\times = {}^1\mathbf{R}_2\,{}^1\mathbf{R}_2\,{}^1_1\mathbf{r}_2 \times {}^1\mathbf{R}_2 \\ &= {}^1_1\mathbf{r}_2 \times {}^1\mathbf{R}_2 \end{aligned} \tag{12.96}$$

$^1\mathbf{X}_2$ can be equivalently expressed as

$$^1\mathbf{X}_2 = \begin{bmatrix} ^1\mathbf{R}_2 & 0 \\ ^1_1\mathbf{r}_2 \times {}^1\mathbf{R}_2 & {}^1\mathbf{R}_2 \end{bmatrix} \tag{12.97}$$

12.5.2 Inverse Transformation of Twist

Consider the same physical system as described in the derivation of $^1\mathbf{X}_2$. Given the twist represented in $\{1\}$ as

$$^1\mathbf{V}_2 = \begin{bmatrix} ^1\boldsymbol{\omega}_2 \\ ^1\mathbf{v}_{1'} \end{bmatrix} \tag{12.98}$$

it is desired that the inverse transformation $^2\mathbf{X}_1$ is derived. The linear velocity of the origin of $\{2\}$, represented in $\{1\}$, is given by

$$^1\mathbf{v}_2 = {}^1\mathbf{v}_{1'} + {}^1\boldsymbol{\omega}_2 \times {}^1\mathbf{r}_2 = {}^1\mathbf{v}_{1'} + {}^1\boldsymbol{\omega}_2 \times {}^1_1\mathbf{r}_2 = {}^1\mathbf{v}_{1'} + {}^1_2\mathbf{r}_1 \times {}^1\boldsymbol{\omega}_2 \tag{12.99}$$

Furthermore, representing such linear velocity in $\{2\}$ yields

$$\begin{aligned} ^2\mathbf{v}_2 &= {}^2\mathbf{R}_1\,{}^1\mathbf{v}_{1'} + {}^2\mathbf{R}_1({}^1_2\mathbf{r}_1 \times {}^1\boldsymbol{\omega}_2) \\ &= {}^2\mathbf{R}_1\,{}^1\mathbf{v}_{1'} + {}^2\mathbf{R}_1\,{}^1_2\mathbf{r}_1 \times {}^1\boldsymbol{\omega}_2 \end{aligned} \tag{12.100}$$

The angular velocity can be readily transformed with a rotation matrix

$$^2\boldsymbol{\omega}_2 = {}^2\mathbf{R}_1{}^1\omega_2 \tag{12.101}$$

The twist $\mathbf{V}_2$ represented in $\{2\}$ is written as

$$\begin{aligned} ^2\mathbf{V}_2 = \begin{bmatrix} ^2\boldsymbol{\omega}_2 \\ ^2\mathbf{v}_2 \end{bmatrix} &= \begin{bmatrix} ^2\mathbf{R}_1{}^1\omega_2 \\ ^2\mathbf{R}_1\,{}^1\mathbf{v}_{1'} + {}^2\mathbf{R}_1\,{}^1_2\mathbf{r}_1 \times {}^1\boldsymbol{\omega}_2 \end{bmatrix} \\ &= \begin{bmatrix} ^2\mathbf{R}_1 & 0 \\ ^2\mathbf{R}_1\,{}^1_2\mathbf{r}_1\times & {}^2\mathbf{R}_1 \end{bmatrix} \begin{bmatrix} ^1\boldsymbol{\omega}_2 \\ ^1\mathbf{v}_{1'} \end{bmatrix} \\ &= {}^2\mathbf{X}_1\,{}^1\mathbf{V}_2 \end{aligned} \tag{12.102}$$

Given

$$\begin{aligned} ^2\mathbf{R}_1\,{}^1_2\mathbf{r}_1\times &= {}^2\mathbf{R}_1({}^1\mathbf{R}_2{}^2\,{}_2\mathbf{r}_1)\times = {}^2\mathbf{R}_1\,{}^1\mathbf{R}_2\,{}^2_2\mathbf{r}_1 \times {}^2\mathbf{R}_1 \\ &= {}^2_2\mathbf{r}_1 \times {}^2\mathbf{R}_1 \end{aligned} \tag{12.103}$$

$^2\mathbf{X}_1$ can be equivalently expressed as

$$^2\mathbf{X}_1 = \begin{bmatrix} ^2\mathbf{R}_1 & 0 \\ ^2_2\mathbf{r}_1 \times {}^2\mathbf{R}_1 & {}^2\mathbf{R}_1 \end{bmatrix} \tag{12.104}$$

12.5.3 Transformation of Wrench

Similar to the relation between velocities and a twist, effort (force and moment) can be represented by a 6×1 wrench. Consider an arbitrary wrench at the origin of Frame {2} represented as

$$ {}^{2}\mathbf{L}_2 = \begin{bmatrix} {}^{2}\mathbf{n}_2 \\ {}^{2}\mathbf{m}_2 \end{bmatrix} \tag{12.105}$$

where ${}^{2}\mathbf{n}_2$ and ${}^{2}\mathbf{m}_2$ are arbitrary force and moment (dimensions 3×1) applied at the origin of {2}. To represent said wrench in frame {1}, the transformation matrix must be written in such a way that the physical effect of the wrench stays unchanged during the transformation, thus

$$ {}^{1}\mathbf{L}_2 = \begin{bmatrix} {}^{1}\mathbf{R}_2\, {}^{2}\mathbf{n}_2 \\ {}^{1}\mathbf{R}_2\, {}^{2}\mathbf{m}_2 + {}^{1}\mathbf{R}_2 \left({}^{2}_{1}\mathbf{r}_2 \times {}^{2}\mathbf{n}_2\right) \end{bmatrix} \tag{12.106}$$

given

$$\begin{aligned} {}^{1}\mathbf{R}_2 \left({}^{2}_{1}\mathbf{r}_2 \times {}^{2}\mathbf{n}_2\right) &= \left({}^{1}\mathbf{R}_2\, {}^{2}_{1}\mathbf{r}_2\right) \times \left({}^{1}\mathbf{R}_2\, {}^{2}\mathbf{n}_2\right) \\ &= {}^{1}\mathbf{R}_2\, {}^{2}_{1}\mathbf{r}_2 \times {}^{2}\mathbf{R}_1\, {}^{1}\mathbf{R}_2\, {}^{2}\mathbf{n}_2 \\ &= {}^{1}\mathbf{R}_2\, {}^{2}_{1}\mathbf{r}_2 \times {}^{2}\mathbf{n}_2 \end{aligned} \tag{12.107}$$

${}^{1}\mathbf{L}_2$ can be rearranged into matrix form

$$ {}^{1}\mathbf{L}_2 = \begin{bmatrix} {}^{1}\mathbf{R}_2 & 0 \\ {}^{1}\mathbf{R}_2\, {}^{2}_{1}\mathbf{r}_2 \times & {}^{1}\mathbf{R}_2 \end{bmatrix} \begin{bmatrix} {}^{2}\mathbf{n}_2 \\ {}^{2}\mathbf{m}_2 \end{bmatrix} = {}^{1}\mathbf{X}_2\, {}^{2}\mathbf{L}_2 \tag{12.108}$$

where ${}^{1}\mathbf{X}_2$ is a 6×6 transformation matrix for the wrench. Alternatively, given

$$\begin{aligned} {}^{1}\mathbf{R}_2\, {}^{2}_{1}\mathbf{r}_2 \times &= {}^{1}\mathbf{R}_2 ({}^{2}\mathbf{R}_1\, {}^{1}_{1}\mathbf{r}_2) \times \\ &= {}^{1}\mathbf{R}_2\, {}^{2}\mathbf{R}_1\, {}^{1}_{1}\mathbf{r}_2 \times {}^{1}\mathbf{R}_2 \end{aligned} \tag{12.109}$$

${}^{1}\mathbf{X}_2$ can be written as

$$ {}^{1}\mathbf{X}_2 = \begin{bmatrix} {}^{1}\mathbf{R}_2 & 0 \\ {}^{1}_{1}\mathbf{r}_2 \times {}^{1}\mathbf{R}_2 & {}^{1}\mathbf{R}_2 \end{bmatrix} \tag{12.110}$$

12.5.4 Derivative of ${}^{0}\mathbf{X}_0$ Matrix

Consider the physical system used in the previous derivation, with an additional arbitrary stationary reference frame {0}, where the transformation matrix between {0} and {2} is

$$ {}^{0}\mathbf{X}_2 = \begin{bmatrix} {}^{0}\mathbf{R}_2 & 0 \\ {}^{0}_{0}\mathbf{r}_2 \times {}^{0}\mathbf{R}_2 & {}^{0}\mathbf{R}_2 \end{bmatrix} \tag{12.111}$$

Taking the time derivative of an X matrix gives

$$ \frac{d}{dt}\, {}^{0}\mathbf{X}_2 = \begin{bmatrix} \frac{d}{dt}\, {}^{0}\mathbf{R}_2 & 0 \\ \frac{d}{dt}({}^{0}_{0}\mathbf{r}_2) \times {}^{0}\mathbf{R}_2 + {}^{0}_{0}\mathbf{r}_2 \times \frac{d}{dt}({}^{0}\mathbf{R}_2) & \frac{d}{dt}\, {}^{0}\mathbf{R}_2 \end{bmatrix} \tag{12.112}$$

Given

$$ \frac{d}{dt}({}^{0}\mathbf{R}_2)\, {}^{0}\mathbf{R}_2^{T} = {}^{0}\boldsymbol{\omega}_2 \times \tag{12.113}$$

the derivative of the rotation matrix is

$$\frac{d}{dt}\,{}^0\mathbf{R}_2 = {}^0\boldsymbol{\omega}_2 \times {}^0\mathbf{R}_2 \tag{12.114}$$

The derivative of r is the linear velocity of the origin of $\{2\}$ with respect to the origin of F_0. Knowing that,

$$\begin{aligned} {}^0\mathbf{v}_{0'} &= {}^0\mathbf{v}_2 + {}^0\boldsymbol{\omega}_2 \times {}^0_2\mathbf{r}_{0'} \\ &= {}^0\mathbf{v}_2 + {}^0\boldsymbol{\omega}_2 \times {}^0_2\mathbf{r}_0 \end{aligned} \tag{12.115}$$

where $0'$ is a point that moves with the rigid body and is instantaneously coincident with Point 0. Element $(2, 1)$ in the derivative of X can be written as

$$\begin{aligned} &\frac{d}{dt}({}^0_0\mathbf{r}_2) \times {}^0\mathbf{R}_2 + {}^0_0\mathbf{r}_2 \times \frac{d}{dt}({}^0\mathbf{R}_2) \\ &\qquad = {}^0\mathbf{v}_2 \times {}^0\mathbf{R}_2 + {}^0_0\mathbf{r}_2 \times ({}^0\boldsymbol{\omega}_2 \times {}^0\mathbf{R}_2) \\ &\qquad = ({}^0\mathbf{v}_{0'} - {}^0\boldsymbol{\omega}_2 \times {}^0_2\mathbf{r}_0) \times {}^0\mathbf{R}_2 + {}^0_0\mathbf{r}_2 \times ({}^0\boldsymbol{\omega}_2 \times {}^0\mathbf{R}_2) \\ &\qquad = {}^0\mathbf{v}_{0'} \times {}^0\mathbf{R}_2 - ({}^0\boldsymbol{\omega}_2 \times {}^0_2\mathbf{r}_0) \times {}^0\mathbf{R}_2 + {}^0_0\mathbf{r}_2 \times ({}^0\boldsymbol{\omega}_2 \times {}^0\mathbf{R}_2) \\ &\qquad = {}^0\mathbf{v}_{0'} \times {}^0\mathbf{R}_2 + {}^0\mathbf{R}_2 \times ({}^0\boldsymbol{\omega}_2 \times {}^0_2\mathbf{r}_0) + {}^0_0\mathbf{r}_2 \times ({}^0\boldsymbol{\omega}_2 \times {}^0\mathbf{R}_2) \\ &\qquad = {}^0\mathbf{v}_{0'} \times {}^0\mathbf{R}_2 + {}^0\mathbf{R}_2 \times ({}^0_0\mathbf{r}_2 \times {}^0\boldsymbol{\omega}_2) + {}^0_0\mathbf{r}_2 \times ({}^0\boldsymbol{\omega}_2 \times {}^0\mathbf{R}_2) \end{aligned} \tag{12.116}$$

The last two terms in the last line of the expression can be simplified using Jacobi identity as

$${}^0\mathbf{R}_2 \times ({}^0_0\mathbf{r}_2 \times {}^0\boldsymbol{\omega}_2) + {}^0_0\mathbf{r}_2 \times ({}^0\boldsymbol{\omega}_2 \times {}^0\mathbf{R}_2) = -\,{}^0\boldsymbol{\omega}_2 \times ({}^0\mathbf{R}_2 \times {}^0_0\mathbf{r}_2) \tag{12.117}$$

Thus

$$\begin{aligned} \frac{d}{dt}({}^0_0\mathbf{r}_2) \times {}^0\mathbf{R}_2 + {}^0_0\mathbf{r}_2 \times \frac{d}{dt}({}^0\mathbf{R}_2) &= {}^0\mathbf{v}_{0'} \times {}^0\mathbf{R}_2 - {}^0\boldsymbol{\omega}_2 \times ({}^0\mathbf{R}_2 \times {}^0_0\mathbf{r}_2) \\ &= {}^0\mathbf{v}_{0'} \times {}^0\mathbf{R}_2 + {}^0\boldsymbol{\omega}_2 \times ({}^0_0\mathbf{r}_2 \times {}^0\mathbf{R}_2) \end{aligned} \tag{12.118}$$

Rearranging the derivative of ${}^0\mathbf{X}_2$ into matrix form yields

$$\begin{aligned} \frac{d}{dt}\,{}^0\mathbf{X}_2 &= \begin{bmatrix} {}^0\boldsymbol{\omega}_2\times & 0 \\ {}^0\mathbf{v}_{0'}\times & {}^0\boldsymbol{\omega}_2\times \end{bmatrix} \begin{bmatrix} {}^0\mathbf{R}_2 & 0 \\ {}^0_0\mathbf{r}_2 \times {}^0\mathbf{R}_2 & {}^0\mathbf{R}_2 \end{bmatrix} \\ &= \begin{bmatrix} {}^0\boldsymbol{\omega}_2\times & 0 \\ {}^0\mathbf{v}_{0'}\times & {}^0\boldsymbol{\omega}_2\times \end{bmatrix} {}^0\mathbf{X}_2 \end{aligned} \tag{12.119}$$

Borrowing the operation between ω and $\omega\times$, the 6×6 matrix in the equation above, which contains the cross-product of the velocities, can be represented as

$$\begin{bmatrix} {}^0\boldsymbol{\omega}_2\times & 0 \\ {}^0\mathbf{v}_{0'}\times & {}^0\boldsymbol{\omega}_2\times \end{bmatrix} = \begin{bmatrix} {}^0\boldsymbol{\omega}_2 \\ {}^2\mathbf{v}_{0'} \end{bmatrix} \times \tag{12.120}$$

Alternatively, given

$$\frac{d}{dt}\,{}^0\mathbf{R}_2 = {}^0\boldsymbol{\omega}_2 \times {}^0\mathbf{R}_2 = {}^0\mathbf{R}_2\,{}^2\boldsymbol{\omega}_2\times \tag{12.121}$$

and

$$\frac{d}{dt}({}^0_0\mathbf{r}_2) \times {}^0\mathbf{R}_2 + {}^0_0\mathbf{r}_2 \times \frac{d}{dt}({}^0\mathbf{R}_2)$$

$$\begin{aligned}
&= {}^0\mathbf{v}_{0'} \times {}^0\mathbf{R}_2 + {}^0\boldsymbol{\omega}_2 \times ({}^0_0\mathbf{r}_2 \times {}^0\mathbf{R}_2) \\
&= {}^0\mathbf{v}_2 \times {}^0\mathbf{R}_2 + {}^0_0\mathbf{r}_2 \times {}^0\mathbf{R}_2\, {}^2\boldsymbol{\omega}_2 \times \\
&= ({}^0\mathbf{R}_2\, {}^2\mathbf{v}_2) \times {}^0\mathbf{R}_2 + ({}^0\mathbf{R}_2 {}^2_0 r_2) \times {}^0\mathbf{R}_2\, {}^2\boldsymbol{\omega}_2 \times \\
&= ({}^0\mathbf{R}_2\, {}^2\mathbf{v}_2) \times {}^0\mathbf{R}_2 + {}^0\mathbf{R}_2 {}^2_0 r_2 \times {}^2\boldsymbol{\omega}_2 \times \qquad (12.122)
\end{aligned}$$

the derivative of ${}^0\mathbf{X}_2$ can be equivalently expressed as

$$\begin{aligned}
\frac{d}{dt}\,{}^0\mathbf{X}_2 &= \begin{bmatrix} {}^0\mathbf{R}_2 & 0 \\ {}^0\mathbf{R}_2 {}^2_0 r_2 \times & {}^0\mathbf{R}_2 \end{bmatrix} \begin{bmatrix} {}^2\boldsymbol{\omega}_2 \times & 0 \\ {}^2\mathbf{v}_2 \times & {}^2\boldsymbol{\omega}_2 \times \end{bmatrix} \\
&= {}^0\mathbf{X}_2 \begin{bmatrix} {}^2\boldsymbol{\omega}_2 \times & 0 \\ {}^2\mathbf{v}_2 \times & {}^2\boldsymbol{\omega}_2 \times \end{bmatrix} \\
&= {}^0\mathbf{X}_2 \begin{bmatrix} {}^2\boldsymbol{\omega}_2 \\ {}^2\mathbf{v}_2 \end{bmatrix} \times \qquad (12.123)
\end{aligned}$$

Now consider the case where $\{1\}$ is not stationary, e.g., being the frame of the previous link of $\{2\}$ on a robotic manipulator. The derivative of ${}^1\mathbf{X}_2$ is

$$\frac{d}{dt}\,{}^1\mathbf{X}_2 = \begin{bmatrix} \frac{d}{dt}\,{}^1\mathbf{R}_2 & 0 \\ \frac{d}{dt}({}^1_1\mathbf{r}_2) \times {}^1\mathbf{R}_2 + {}^1_1\mathbf{r}_2 \times \frac{d}{dt}({}^1\mathbf{R}_2) & \frac{d}{dt}\,{}^1\mathbf{R}_2 \end{bmatrix} \qquad (12.124)$$

For the case where the joint (Joint 2) is revolute, r is a constant. Therefore

$${}^2\mathbf{X}_1 \frac{d}{dt}\,{}^1\mathbf{X}_2 = \begin{bmatrix} {}^2\mathbf{R}_1 & 0 \\ {}^2\mathbf{R}_1\, {}^1_2\mathbf{r}_1 \times & {}^2\mathbf{R}_1 \end{bmatrix} \begin{bmatrix} \frac{d}{dt}\,{}^1\mathbf{R}_2 & 0 \\ {}^1_1\mathbf{r}_2 \times \frac{d}{dt}\,{}^1\mathbf{R}_2 & \frac{d}{dt}\,{}^1\mathbf{R}_2 \end{bmatrix} \qquad (12.125)$$

Since

$$\frac{d}{dt}\,{}^1\mathbf{R}_2 = {}^1_1\boldsymbol{\omega}_2 \times {}^1\mathbf{R}_2 \qquad (12.126)$$

where ${}^1_1\boldsymbol{\omega}_2$ is the angular velocity of Joint 2 with respect $\{1\}$, the equation becomes

$${}^2\mathbf{X}_1 \frac{d}{dt}\,{}^1\mathbf{X}_2 = \begin{bmatrix} {}^2\mathbf{R}_1\, {}^1_1\boldsymbol{\omega}_2 \times {}^1\mathbf{R}_2 & 0 \\ {}^2\mathbf{R}_1\, {}^1_2\mathbf{r}_1 \times \left({}^1_1\boldsymbol{\omega}_2 \times {}^1\mathbf{R}_2\right) + {}^2\mathbf{R}_1\, {}^1_1\mathbf{r}_2 \times \left({}^1_1\boldsymbol{\omega}_2 \times {}^1\mathbf{R}_2\right) & {}^2\mathbf{R}_1\, {}^1_1\boldsymbol{\omega}_2 \times {}^1\mathbf{R}_2 \end{bmatrix} \qquad (12.127)$$

It can be readily seen that Element $(3,1)$ is zero, as ${}_1r_2$ and ${}_2\mathbf{r}_1$ cancel out with each other. Thus

$$\begin{aligned}
{}^2\mathbf{X}_1 \frac{d}{dt}\,{}^1\mathbf{X}_2 &= \begin{bmatrix} {}^2\mathbf{R}_1\, {}^1_1\boldsymbol{\omega}_2 \times {}^1\mathbf{R}_2 & 0 \\ 0 & {}^2\mathbf{R}_1\, {}^1_1\boldsymbol{\omega}_2 \times {}^1\mathbf{R}_2 \end{bmatrix} \\
&= \begin{bmatrix} {}^2_1\boldsymbol{\omega}_2 \times & 0 \\ 0 & {}^2_1\boldsymbol{\omega}_2 \times \end{bmatrix} \\
&= \begin{bmatrix} {}^2_1\boldsymbol{\omega}_2 \\ 0 \end{bmatrix} \times \qquad (12.128)
\end{aligned}$$

For the case where the joint is a prismatic joint, ${}^1\mathbf{R}_2$ is a constant. Therefore

$${}^2\mathbf{X}_1 \frac{d}{dt}\,{}^1\mathbf{X}_2 = \begin{bmatrix} {}^2\mathbf{R}_1 & 0 \\ {}^2\mathbf{R}_1\, {}^1_2\mathbf{r}_1 \times & {}^2\mathbf{R}_1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ \frac{d}{dt}({}^1_1\mathbf{r}_2) \times {}^1\mathbf{R}_2 & 0 \end{bmatrix} \qquad (12.129)$$

Since the derivative of ${}^{1}_{1}\mathbf{r}_2$ is merely the linear velocity of the joint with respect to $\{1\}$

$$
\begin{aligned}
{}^{2}\mathbf{X}_1 \frac{d}{dt}\,{}^{1}\mathbf{X}_2 &= \begin{bmatrix} {}^{2}\mathbf{R}_1 & 0 \\ {}^{2}\mathbf{R}_1\,{}^{1}_{2}\mathbf{r}_1\times & {}^{2}\mathbf{R}_1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ {}^{1}_{1}\mathbf{v}_2\times\,{}^{1}\mathbf{R}_2 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 0 \\ {}^{2}_{1}\mathbf{v}_2\times & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ {}^{2}_{1}\mathbf{v}_2 \end{bmatrix}\times
\end{aligned} \tag{12.130}
$$

12.5.5 Physical Interpretation

An important aspect relevant to the physical interpretation of a twist or a wrench is that they can use one physical quantity to fully represent the velocity or force domain information of an entire rigid body. Given a twist or a wrench of a point on a rigid body, the angular velocity or force applies to the entire rigid body, while the linear velocity or moment is dedicated to said point. From there, given an arbitrary point with a prescribed relative position (r in the previous part of this section), the velocity and effective force/moment at the point can be conveniently computed by means of the $\mathbf{X}$ transformation matrix. We will showcase the use of this characteristic in Chapter 19, where we project the forces and moments at different parts of a soft continuum robot to establish the static equilibrium equations.

12.6 Applications in Computing

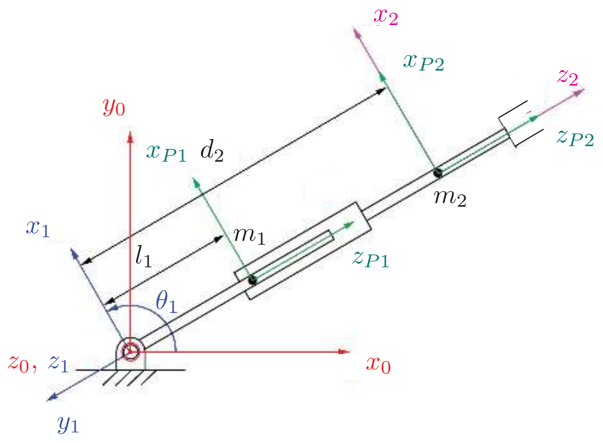
The dynamics of a serial manipulator can be quite complex, especially for manipulators of six or more DoF, where closed-form equations are observed to be highly complex and non-linear. In a robotic control system that utilises dynamics, the computational costs for real-time performance must be considered. As the dynamics discussed so far covers the joint space, we will restrict our attention to this space.

12.6.1 Computing Efficiency

The Newton-Euler method for calculating dynamics is tailor-made for numerical computing due to its iterative nature. In addition, variables can be substituted as symbolic quantities in MATLAB, such that we obtain analytical expressions for the dynamic equations. Compared to the Lagrangian formulation, the Newton-Euler method is more efficient, as fewer computational steps are required to obtain dynamic equations in symbolic form.

Example M12.1 (Newton-Euler dynamics): Compute the torque and force equations for the RP manipulator in Figure 12.9. Assume m_1 and m_2 are point masses, there are no external forces applied at the end-effector, the robot base is stationary, and gravity is applied in the y_0 direction. Calculate the required numerical actuator input force and torque when the manipulator is stationary at $\theta_1 = 30^\circ$ and $d_2 = 0.40$ m. Assume $l_1 = 0.15$ m, and m_1 and m_2 are 4.0 kg and 3.0 kg, respectively.

Solution: The following script sets up the problem to solve via the Newton-Euler method. To convert the DH table to transformation matrices ${}^{0}\mathbf{T}_1$, ${}^{1}\mathbf{T}_2$, and ${}^{2}\mathbf{T}_3$, we make use of `dh2T()` as defined in Inline 6.1. Notice that the last line of the DH table is made of zeroes to indicate the end-



i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	$90°$	0	d_2	0
3	0	0	0	0

FIGURE 12.9
RP manipulator with DH table.

effector frame is coincident with Frame 2, and is where an external force will be applied if one exists. The variables defined in this script are `Q_1` and `Q_2`, which represent θ_1 and d_2, respectively. Variables `dQ` and `ddQ` represent the variable's time and double-time derivatives (velocity and acceleration).

The setup of this problem can be quite long, as indicated by 58 lines and the Newton-Euler code does not begin until line 60. In this example, the problem is solved without a for-loop construct so that each step can be inspected. However, in application, a for-loop solution is highly recommended to shorten code and minimise errors.

```
 1 % Vector definition
 2 X = [1 0 0]';
 3 Y = [0 1 0]';
 4 Z = [0 0 1]';
 5 
 6 %Symbolic definitions
 7 syms l_1 g t real
 8 
 9 % Mass definition
10 m = sym('m_',[1 2],'real');
11 
12 % Time-dependent variables
13 Q = sym('Q_',[1 2],'real');
14 dQ = sym('dQ_',[1 2],'real');
15 ddQ = sym('ddQ_',[1 2],'real');
16 
17 % Centre of mass location
18 P1_C1 = l_1 * -Y;
19 P2_C2 = [0 0 0]';
20 
21 % Inertia tensors (point masses)
22 I_C1_1 = zeros(3);
23 I_C2_2 = zeros(3);
24 
25 % End-effector forces and torques
26 f33 = [0 0 0]';
27 n33 = [0 0 0]';
28 
29 % No ang velocity or accel at robot base
30 w00 = [0 0 0]';
31 dw00 = [0 0 0]';
32 % Linear accel at robot base (due to gravity)
33 dv00 = g * Y;
34 
35 % DH table
36 dh_table = [
37     0       0       0       Q(1);
38     pi/2    0       Q(2)    0;
39     0       0       0       0]    % End effector frame (to apply external force)
40 
41 % Transformation matrices
42 T01 = dh2T(dh_table(1,:));
```

```
T12 = dh2T(dh_table(2,:));
T23 = dh2T(dh_table(3,:));

% Rotation matrices
R01 = T01(1:3,1:3);
R12 = T12(1:3,1:3);
R23 = T23(1:3,1:3);
R10 = R01';
R21 = R12';
R32 = R23';

% Positions
P01 = T01(1:3,4);
P12 = T12(1:3,4);
P23 = T23(1:3,4);

%% ======================================
% OUTWARD ITERATIONS (i = 0, 1)
% Calculate forces and torques exerted on each link due to the linear accel
% of the centre of mass.
% ==========
% i = 0
% Velocities and accels

% Propagate ang velocity of frame 1 from base, relative to frame 1
w11 = R10*w00 + dQ(1)*Z

% Propagate ang accel of frame 1 from base, relative to frame 1
dw11 = R10*dw00 + cross(R10*w00, dQ(1)*Z) + ddQ(1)*Z

% Propagate linear accel at frame 1 from base, relative to frame 1
dv11 = R10*(cross(dw00, P01) + cross(w00, cross(w00, P01)) + dv00)

% Propagate linear accel at centre of mass 1 from frame 1, relative to frame 1
dvc11 = cross(dw11, P1_C1) + cross(w11, cross(w11, P1_C1)) + dv11

% =====
% Force and torque acting on link 1 from CoM 1
% Find inertial force from CoM 1 acting on link 1, relative to frame 1
F11 = m(1)*dvc11

% Find inertial torque from CoM 1 acting on link 1, relative to frame 1
N11 = I_C1_1*dw11 + cross(w11, I_C1_1*w11)

% ==========
% i = 1
% Velocities and accels
% Propagate ang vel of frame 2 from frame 1, relative to frame 2 (prismatic joint)
w22 = R21*w11

% Propagate ang accel of frame 2 from frame 1, relative to frame 2 (prismatic joint)
dw22 = R21*dw11

% Propagate linear accel at frame 2 from frame 1, relative to frame 2 (prismatic j)
dv22 = R21*(cross(dw11, P12) + cross(w11, cross(w11, P12)) + dv11 ) ...
                + 2*cross(w22, dQ(2)*Z ) + ddQ(2)*Z

% Propagate linear accel at centre of mass 2 from frame 2, relative to frame 2
dvc22 = cross(dw22, P2_C2) + cross(w22, cross(w22, P2_C2)) + dv22

% =====
% Force and torque acting on link 2 from CoM 2
% Find inertial force from CoM 2 acting on link 2, relative to frame 2
F22 = m(2)*dvc22

% Find inertial torque from CoM 2 acting on link 2, relative to frame 2
N22 = I_C2_2*dw22 + cross(w22, I_C2_2*w22)

%% ======================================
% INWARD ITERATIONS (i = 2, 1)
% Calculate forces and torques exerted on each link due to forces and torques
% exerted by the neighbouring link.
% i = 2
%Force exerted on link 2 by CoM 2 and external force
f22 = R23*f33 + F22

%Torque exerted on link 2 by CoM 2 external torque
n22 = N22 + R23*n33 + cross(P2_C2, F22) + cross(P23, R23*f33)

% i = 1
%Force exerted on link 1 by CoM 1 and link 2
f11 = R12*f22 + F11

%Torque exerted on link 1 by CoM 1 and link 2
```

```
n11 = N11 + R12*n22 + cross(P1_C1, F11) + cross(P12, R12*f22)

%% =====================================
% Actuator Effort
%Torque on actuator (i = 1)
E(1,1) = n11'*Z;
%Force on actuator (i = 2)
E(2,1) = f22'*Z
```

```
dh_table =

[     0, 0,    0, Q_1]
[ pi/2, 0, Q_2,    0]
[     0, 0,    0,    0]


w11 =

    0
    0
 dQ_1


dw11 =

     0
     0
 ddQ_1


dv11 =

 g*sin(Q_1)
 g*cos(Q_1)
          0


dvc11 =

  ddQ_1*l_1 + g*sin(Q_1)
 l_1*dQ_1^2 + g*cos(Q_1)
                       0


F11 =

  m_1*(ddQ_1*l_1 + g*sin(Q_1))
 m_1*(l_1*dQ_1^2 + g*cos(Q_1))
                             0


N11 =

 0
 0
 0


w22 =

    0
 dQ_1
    0


dw22 =

     0
 ddQ_1
     0


dv22 =

 Q_2*ddQ_1 + 2*dQ_1*dQ_2 + g*sin(Q_1)
                                    0
    - Q_2*dQ_1^2 + ddQ_2 - g*cos(Q_1)


dvc22 =

 Q_2*ddQ_1 + 2*dQ_1*dQ_2 + g*sin(Q_1)
                                    0
    - Q_2*dQ_1^2 + ddQ_2 - g*cos(Q_1)


F22 =

 m_2*(Q_2*ddQ_1 + 2*dQ_1*dQ_2 + g*sin(Q_1))
                                           0
     -m_2*(Q_2*dQ_1^2 - ddQ_2 + g*cos(Q_1))


N22 =

 0
 0
 0


f22 =

 m_2*(Q_2*ddQ_1 + 2*dQ_1*dQ_2 + g*sin(Q_1))
                                           0
     -m_2*(Q_2*dQ_1^2 - ddQ_2 + g*cos(Q_1))


n22 =

 0
 0
 0


f11 =

 m_1*(ddQ_1*l_1 + g*sin(Q_1)) + m_2*(Q_2*ddQ_1 + 2*dQ_1*dQ_2 + g*sin(Q_1))
     m_1*(l_1*dQ_1^2 + g*cos(Q_1)) + m_2*(Q_2*dQ_1^2 - ddQ_2 + g*cos(Q_1))
                                                                         0


n11 =

                                                                                 0
                                                                                 0
 Q_2*m_2*(Q_2*ddQ_1 + 2*dQ_1*dQ_2 + g*sin(Q_1)) + l_1*m_1*(ddQ_1*l_1 + g*sin(Q_1))


E =

 Q_2*m_2*(Q_2*ddQ_1 + 2*dQ_1*dQ_2 + g*sin(Q_1)) + l_1*m_1*(ddQ_1*l_1 + g*sin(Q_1))
                                           -m_2*(Q_2*dQ_1^2 - ddQ_2 + g*cos(Q_1))
```

Remembering that Q_1 and Q_2 represent θ_1 and d_2, respectively, according to effort variable E, the dynamic equations for the revolute and prismatic actuators, respectively, are

$$\tau_1 = d_2 m_2(d_2\ddot{\theta}_1 + 2\dot{\theta}_1\dot{d}_2 + g\sin\theta_1) + l_1 m_1(\ddot{\theta}_1 l_1 + g\sin\theta_1) \tag{12.131}$$

$$f_2 = -m_2(d_2\dot{\theta}_1^{\,2} - \ddot{d}_2 + g\cos\theta_1) \tag{12.132}$$

Appending the code below to the end of the previous script conducts the substitution for the numerical solution:

```
% NUMERICAL SUBSTITUTION
%Insert numerical parameters
Qn = [deg2rad(30), 0.40]; % joint position Q_1 = theta_1 and Q_2 = d_2
dQn = [0, 0]; % joint velocity dQ_1 = dQ_2 = 0
ddQn = [0, 0]; % joint acceleration ddQ_1 = ddQ_2 = 0
mn = [4.0 3.0]; % mass m_1 = 4.0 kg and m_2 = 3.0 kg
gn = -9.8; % g acceleration
l_1n = 0.15; % CoM 1 position l_1

% Compute numerical joint force and torque
w11n = eval(subs(w11, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

dw11n = eval(subs(dw11, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

dv11n = eval(subs(dv11, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

dvc11n = eval(subs(dvc11, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

F11n = eval(subs(F11, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

N11n = eval(subs(N11, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

w22n = eval(subs(w22, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

dw22n = eval(subs(dw22, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

dv22n = eval(subs(dv22, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

dvc22n = eval(subs(dvc22, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

F22n = eval(subs(F22, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

N22n = eval(subs(N22, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

f22n = eval(subs(f22, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

n22n = eval(subs(n22, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

f11n = eval(subs(f11, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

n11n = eval(subs(n11, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))

En = eval(subs(E, ...
    [Q, dQ, ddQ, m, g, l_1], ...
    [Qn, dQn, ddQn, mn, gn, l_1n]))
```

```
w11n =

     0
     0
     0


dw11n =

     0
     0
     0


dv11n =

   -4.9000
   -8.4870
         0


dvc11n =

   -4.9000
   -8.4870
```

```
         0


F11n =

  -19.6000
  -33.9482
         0


N11n =

     0
     0
     0


w22n =

     0
     0
     0


dw22n =

     0
     0
     0


dv22n =

   -4.9000
         0
    8.4870


dvc22n =

   -4.9000
         0
    8.4870


F22n =

  -14.7000
         0
   25.4611


N22n =

     0
     0
     0


f22n =

  -14.7000
         0
   25.4611


n22n =

     0
     0
     0


f11n =

  -34.3000
  -59.4093
         0


n11n =

         0
         0
   -8.8200


En =

   -8.8200
   25.4611
```

12.7 Conclusion

In this chapter, we derived Newton-Euler's equations to formulate an iterative method for analysing the dynamics of robotic systems. It consists of two phases: outward and inward propagation. In the outward propagation, the torque and force acting on the centre of mass of each link as a result of inertia are calculated based on the link's velocities and accelerations from its actuators and gravity. The inward propagation resolves the torque and force acting upon each link as a result of external forces from the end-effector. The torque and force seen at each actuator is the sum of all forces and torques as calculated on each link after each iteration.

Due to the iterative nature of this method, it is more efficient to apply this method for dynamic analysis in a computational setting, rather than using Lagrangian dynamics, which focuses on deriving the analytical equations of motion. As seen in the MATLAB example, although setting up the system can be a long process, the process of calculating the equations of motion is quite efficient, especially when combined with loop constructs. The Newton-Euler method can also be used to calculate direct numerical values, if all quantities are known at a particular time.

12.8 Exercises

Problem 1. Compute the joint torque and force of the system defined in Figure 12.10 using the iterative Newton-Euler algorithm. Note that z_2 and z_3 point out of the paper, as indicated by the $\odot$ marks at the respective joint locations. Assume zero initial conditions, but include the gravity term g in the $-z_0$ direction. Also assume point masses m_1 and m_2 located at the midpoints of the links.

Problem 2. Using MATLAB, form the symbolic derivation of the dynamic equations to the plotting of the trajectory and torque profiles, for the three-DoF robot (assume $\{4\}$ is locked) shown in Figure 12.11. Using the Newton-Euler method, obtain the dynamics equations of the system. Assume no forces or moments are acting on the end-effector.

$$
{}^0\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & l_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^1\mathbf{T}_2 = \begin{bmatrix} c_1 & -s_1 & 0 & l_1 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^2\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & -d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Assume the centre of mass of each link is located at the centre of the link. The inertia of

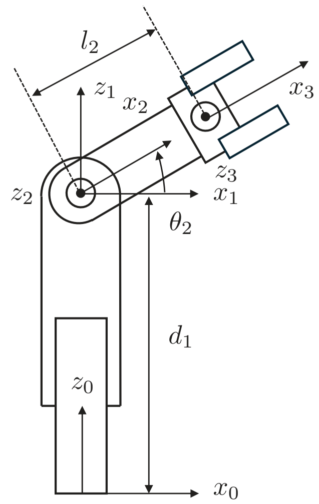


FIGURE 12.10
Two-DOF PR manipulator.

each link is given as:

$$
{}^{c1}\mathbf{I}_1 = \frac{1}{12}m_1 l_1^2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad {}^{c2}\mathbf{I}_2 = \frac{1}{12}m_2 l_2^2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad {}^{c3}\mathbf{I}_3 = \frac{1}{12}m_3 l_3^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}
$$

Problem 3. The schematic structure, frames and home positions of joints of the legs of the Bioloid robot are shown in Figure 12.12.

Consider the right leg of the robot. The hip joints (ID7, 9 and 11) and one of the ankle joints (ID17) are assumed stationary, while the knee (ID13) and the ankle (ID15) are manipulated. The transformation matrices are given by (neglecting the home positions of Joints 11 and 13 as shown in the figure, assuming x_{11} and x_{13} point vertically down at the

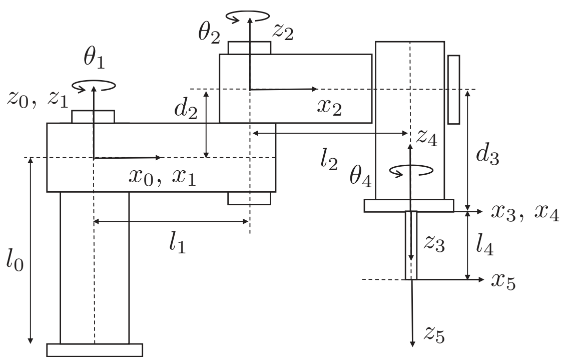


FIGURE 12.11
A 3R non-planar robot.

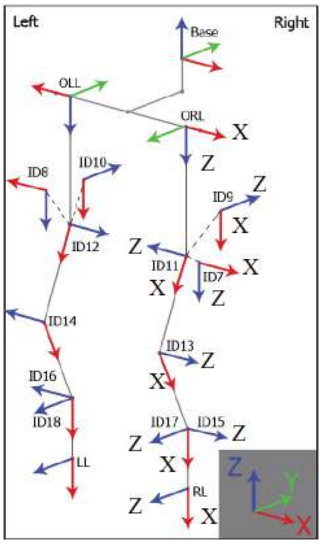


FIGURE 12.12
Legs of Bioloid robot.

home position to simplify calculation)

$$^{11}\mathbf{T}_{13} = \begin{bmatrix} c_{13} & -s_{13} & 0 & a_4 \\ -s_{13} & -c_{13} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{11}\mathbf{T}_{15} = \begin{bmatrix} c_{1315} & -s_{1315} & 0 & a_4 + a_5 c_{13} \\ -s_{1315} & -c_{1315} & 0 & -a_5 s_{13} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the subscripts c_{1315} and s_{1315} stand for the cosine and sine of $(\theta_{13}+\theta_{15})$, respectively.

The mass of the lower leg and foot, m_A and m_B, is assumed to be point mass, located at the origins of frames $\{15\}$ and $\{RL\}$, respectively. The two point masses are represented as

$$^{13}C_A = \begin{bmatrix} a_5 & 0 & 0 & 1 \end{bmatrix}^T$$
$$^{15}C_B = \begin{bmatrix} a_6 & 0 & 0 & 1 \end{bmatrix}^T$$

Use the following physical quantities to construct the dynamics equations of the system.

$$m_A = 0.1 \text{ kg}$$
$$m_B = 0.1 \text{ kg}$$
$$q_{13} = \frac{\pi}{3}$$
$$q_{15} = \frac{\pi}{4}$$
$$a_4 = 0.07698 \text{ m}$$
$$a_5 = 0.07693 \text{ m}$$
$$a_6 = 0.033 \text{ m}$$
$$g = 10 \text{ m/s}^2$$

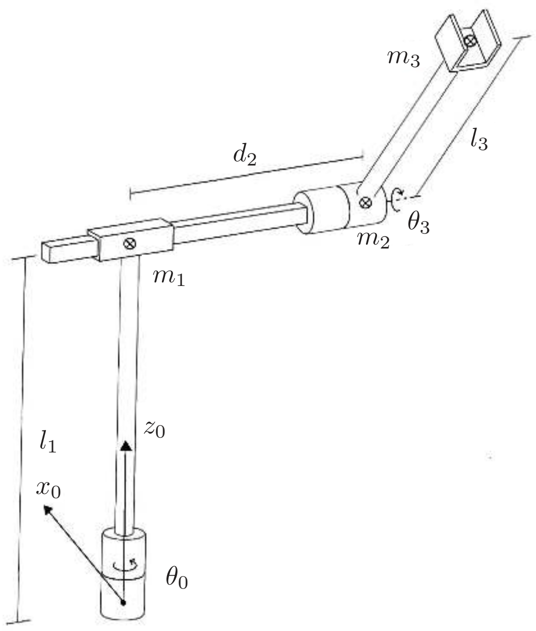


FIGURE 12.13
RPR manipulator.

Problem 4. A RPR manipulator is shown in Figure 12.13.

1. Using the Newton-Euler method, derive the dynamics of this robot. Show the equation used and the analytic solution for each outward iteration. Show the final dynamic equations in matrix format.
2. Give an example of when you would use:
   (a) Newton Euler Method
   (b) Lagrangian Method

13 Joint Control

The purpose of robotic control is to drive the joint servos such that the end-effector follows a prescribed path. Typically, the desired trajectory is given in the task space, as it is simpler to associate a path to the task's coordinate system. The task space path is then mapped back into the joint space via inverse kinematics, which yields a path in the joint space that the joint controllers should follow. If each servo follows its desired trajectory in joint space, then the end-effector should pass through all desired points in the task space.

In Chapter 4, we discussed some simple control schemes suitable for implementation on simple servos. This chapter will expand on these control schemes, which include detailed analyses of their response to different input signals, their stability, and steady-state errors. These analyses are performed in the frequency domain.

A typical single-input/single-output feedback (closed-loop) control system is shown in Figure 13.1, where there are a summing point, compensator, amplifier, plant, and sensor. The reference signal serves as the input to this control system, while the plant generates the output, which is measured by the sensor. The reference signal and the sensed output are compared to create the error. The compensator produces signals that intend to eliminate the error. These signals are amplified by the amplifier to drive the plant. The input signals to the plant may be different from the output signals from the amplifier due to disturbances.

Generally, the compensator, together with the amplifier, is called a controller. The key objectives of the controller design are 1) to allow the output to track the reference input, and 2) to reject the effects of the disturbances on the plant. The former is called the **tracking** while the latter is called the **disturbance rejection**. However, a closed-loop control system can become unstable (oscillating violently), potentially leading to disastrous consequences. Therefore, checking **stability** is the first task of any controller design.

13.1 Servo Dynamics

Most actuators, or servos, in a robotic system, are based on a DC motor (Figure 13.2), where torque is generated at the rotor based on the voltage applied to the armature circuit. By constructing a voltage balance equation around the armature circuit, we can model the torque provided by the rotor and, thus, the rotor position as a function of the voltage applied to the armature circuit.

The voltage loop equation for the armature circuit is

$$V_a = R_a i_a + L_a \dot{i}_a + V_b \tag{13.1}$$

where V_a is the voltage applied at the armature circuit, R_a is the armature wire resistance, L_a is the coil inductance, i_a is the armature current, and V_b is the *back-EMF* induced by the rotor. The back-EMF voltage is produced as the rotor coils pass the fixed magnetic field. Hence, at steady-state stall conditions (when the rotor is stationary), $V_b = 0$, which

DOI: 10.1201/9781003614319-13

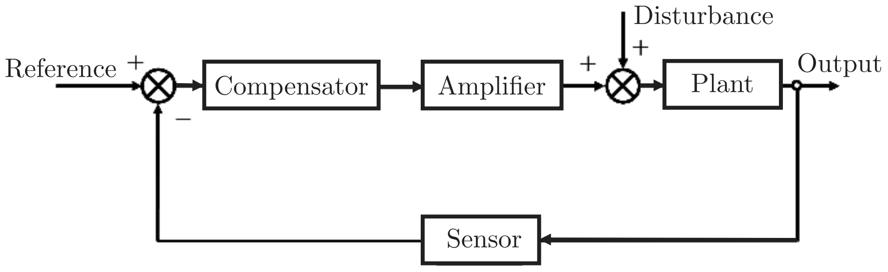


FIGURE 13.1
Linear closed-loop control.

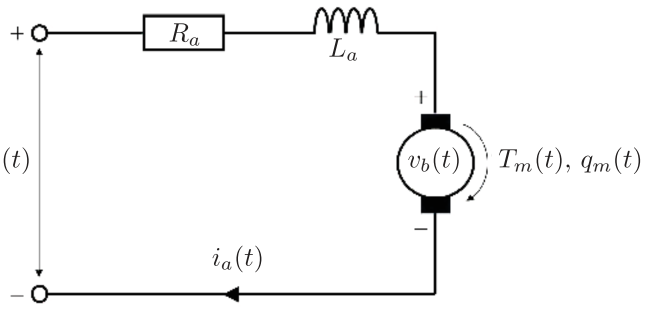


FIGURE 13.2
Armature circuit of a DC motor.

means i_a is maximum, and the magnetic field produced in the rotor is the highest strength, producing maximum torque.[1] The torque produced at the rotor is linearly proportional to the current flow in the armature

$$\tau_m = K_t i_a \tag{13.2}$$

where K_t is a torque constant. This torque causes the rotor to rotate, producing back-EMF voltage as the rotor coil passes the fixed magnetic field. The back-EMF voltage is proportional to the speed of the rotor, given by

$$V_b = K_b \dot{q}_m \tag{13.3}$$

where K_b is the EMF constant. We know that the dynamics of a multi-link robot comes in the form of

$$\tau = M(q)\ddot{q} + V(\dot{q}, q) + B(\dot{q}, q)\dot{q} + G(q) \tag{13.4}$$

where M is the manipulator mass matrix, V is the torque exerted from velocity and Coriolis effects, B is the internal torque due to friction of the actuators, and G is the torque exerted due to gravity. Further, we can identify the torque on each i-th actuator as

$$\tau_i = M_i(q)\ddot{q} + V_i(\dot{q}, q) + B_i(\dot{q}, q)\dot{q}_i + G_i(q) \tag{13.5}$$

[1]This is why DC motors typically *burn out* if they experience stall conditions for extended periods of time, due to increased current flow in the armature coil.

where M_i is a mass vector for the i-th link. Let j be the index along vector M_i, such that the element $j = i$ (i.e., the diagonal element of M, M_{ii}) represents the total inertia as seen at link i from itself and inertia from subsequent links. Therefore, we can rewrite the torque equations as a summation of torque exerted by its own link inertia and actuator friction and define the rest of the torques as *disturbance* torques

$$\begin{aligned}\tau_i &= M_{ii}(q)\ddot{q}_i + \sum_{j=1,j\neq i}^{n} M_{ij}\ddot{q}_j + V_i + G_i + B_i(\dot{q},q)\dot{q}_i \\ &= M_{ii}(q)\ddot{q}_i + D_i(\dot{q},q) + B_i(\dot{q},q)\dot{q}_i \qquad (13.6)\end{aligned}$$

By combining (13.6) with (13.2), we find armature current for actuator i_a

$$i_a = \frac{1}{K_t}\left(M_{ii}(q)\ddot{q}_i + D_i(\dot{q},q) + B_i(\dot{q},q)\dot{q}_i\right) \qquad (13.7)$$

which we can substitute into (13.1) along with (13.3)

$$\begin{aligned}V_a &= R_a i_a + L_a \dot{i_a} + V_b \qquad (13.8)\\ &= \frac{R_a}{K_t}\left(M_{ii}(q)\ddot{q}_i + D_i(\dot{q},q) + B_i(\dot{q},q)\dot{q}_i\right) + L_a\dot{i_a} + K_b\dot{q}_m\end{aligned}$$

$L_a\dot{i_a}$ being zero. Noting that the term $L_a\dot{i_a}$ is negligible as coil inductance is usually very small. In direct drive systems, we set $\tau_m = \tau_i$ and $q_m = q_i$. Therefore, the governing equation of motion modelled as a function of armature voltage V_a is

$$\begin{aligned}V_a &= \frac{R_a}{K_t}M_{ii}(q)\ddot{q}_m + \left(\frac{R_a}{K_t}B_i(\dot{q},q) + K_b\right)\dot{q}_m + \frac{R_a}{K_t}D_i(\dot{q},q)\\ &= J(q)\ddot{q}_m + B(\dot{q},q)\dot{q}_m + \frac{R_a}{K_t}D_i(\dot{q},q) \qquad (13.9)\end{aligned}$$

where J is the inertia function of the link as seen by the driving actuator, B represents the friction functions and EMF constant, and D_i is the disturbance due to the motion of other actuators and links along the manipulator.

Example 13.1 (Inertia contribution): Consider a two-DoF RP robotic manipulator with assigned reference frames shown in Figure 13.3. The dynamic equations governing this system were derived previously. Assume: $m_i = 1$, $l_1 = 1$, and $I_{xxi} = I_{yyi} = I_{zzi} = 1$, for $i = 1, 2$. Find the contributions of the robotic arm to the inertia and coefficient at the first motor's output.
Solution: Dynamic equation at the first joint was derived as

$$\tau_1 = (m_1 l_1^2 + I_{yy1} + I_{yy2} + m_2 d_2^2)\ddot{q}_1 + 2m_2 d_2 \dot{q}_1 \dot{d}_2 + (m_1 l_1 + m_2 d_2) g \sin q_1 \qquad (13.10)$$

Substituting the dynamic parameters into the above equation yields

$$\tau_1 = (3 + d_2^2)\ddot{q}_1 + (2d_2\dot{d}_2)\dot{q}_1 + (1 + d_2) g \sin q_1 \qquad (13.11)$$

Therefore, contributions of the robotic arm to the total inertia and the total damping coefficient of joint 1 are $J_r = (3 + d_2^2)$ and $B_r = (2d_2\dot{d}_2)$, respectively. The results are illustrated in the following table.

Given the dynamic parameters J_a and B_a at armature, the lumped dynamic parameters are given by $J = J_a + J_r$ and $B = B_a + B_r$, respectively. Considering $J_a = 1$ and $B_a = 1$, the induced changes by the robotic arm in J and B are significant, as indicated in Table 13.1.

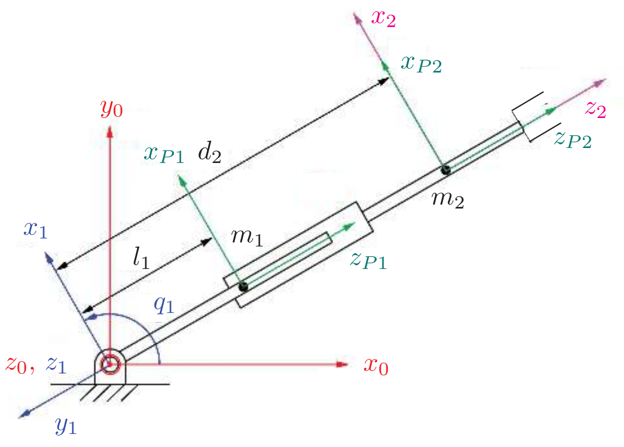


FIGURE 13.3
An RP robot.

13.2 Modelling Servos with a Gear Transmission

A gear transmission can improve the robustness of the control system. Not only will this amplify the torque output of the motor, but it will also increase the precision of sensing and suppress the dynamic disturbance in control (Figure 13.4). The effective inertia and damping factors in this scenario can be derived from the dynamic equations. The torque exerted by the motor behind Gear 1 is

$$\tau_m = J_m\ddot{q}_m + B_m\dot{q}_m + fr_1 \tag{13.12}$$

and the torque after Gear 2 is

$$fr_2 = J_r\ddot{q}_r + B_r\dot{q}_r \tag{13.13}$$

combining the aforementioned two equations yields a torque equation

$$\tau_m = J_{em}\ddot{q}_m + B_{em}\dot{q}_m \tag{13.14}$$

where the effective inertia J_{em} from the motor's perspective is defined as

$$J_{em} = J_m + \frac{J_r}{\eta^2} \tag{13.15}$$

TABLE 13.1
The lumped dynamic parameters of the RP robot

	$d_2 = 0.6$	$d_2 = 0.8$	$d_2 = 1.0$	$d_2 = 1.2$
$\dot{d}_2 = -0.1$	$J_r = 3.36$ $B_r = -0.12$	$J_r = 3.64$ $B_r = -0.16$	$J_r = 4$ $B_r = -0.2$	$J_r = 4$ $B_r = -0.24$
$\dot{d}_2 = 0$	$J_r = 3.36$ $B_r = 0$	$J_r = 3.64$ $B_r = 0$	$J_r = 4$ $B_r = 0$	$J_r = 4$ $B_r = 0$
$\dot{d}_2 = 0.1$	$J_r = 3.36$ $B_r = 0.12$	$J_r = 3.64$ $B_r = 0.16$	$J_r = 4$ $B_r = 0.2$	$J_r = 4$ $B_r = 0.24$

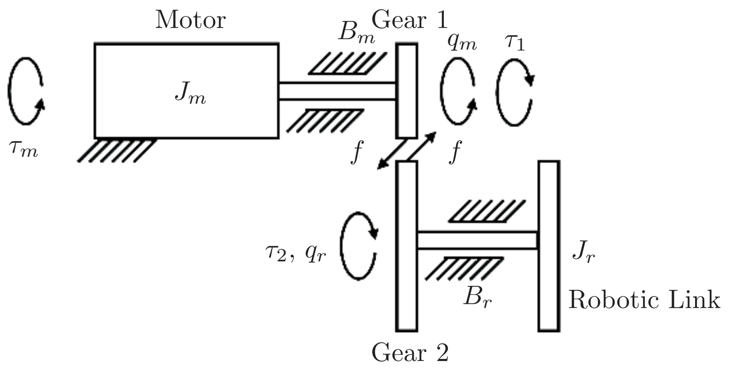


FIGURE 13.4
Robotic link driven by a geared motor.

and the corresponding effective damping factor given by

$$B_{em} = B_m + \frac{B_r}{\eta^2} \tag{13.16}$$

where η denoting the gear ratio

$$\eta = \frac{r_1}{r_2} > 1 \tag{13.17}$$

r_1 and r_2 identify the gear radii, respectively. Considering the output torque of the transmission, $\tau = \eta\tau_m$, we have

$$\frac{\tau}{\eta} = (J_m + \frac{J_r}{\eta^2})\ddot{q}\eta + (B_m + \frac{B_r}{\eta^2})\dot{q}\eta \tag{13.18}$$

which can be written as

$$\tau = (\eta^2 J_m + J_r)\ddot{q} + (\eta^2 B_m + B_r)\dot{q} \tag{13.19}$$

This equation can be further written as

$$\tau = J_e\ddot{q} + B_e\dot{q} \tag{13.20}$$

where J_e and B_e are the effective inertia and the effective damping factor, respectively, given by

$$J_e = \eta^2 J_m + J_r \tag{13.21}$$

and

$$B_e = \eta^2 B_m + B_r \tag{13.22}$$

Example 13.2 (Geared servo dynamics — RP manipulator): Referring to the RP robot in Figure 13.3, plot the torque experienced by the base motor with the following joint trajectories:

$$q_1 = \frac{\pi}{2}\sin(2t) \qquad d_2 = 0.1\cos(2t+1) \tag{13.23}$$

for time t such that $0 \geq t \geq 3$. The system parameters are:

$$\begin{aligned} l_1 &= 0.5 & m_1 &= 2 & m_2 &= 1 \\ J_m &= 1.0 \times 10^{-3} & B_m &= 0.1 & g &= 9.81 \end{aligned} \tag{13.24}$$

Assume point masses. Perform this task for gear ratios $\eta = 1$ and $\eta = 3$ for no load and 5 kg load cases.

Solution: The dynamic equation of the first joint, as seen at the end of the gear transmission, is

$$\tau_1 = J_r \ddot{q}_1 + B_r \dot{q}_1 + (m_1 l_1 + m_2 d_2)\, g \sin q_1 \tag{13.25}$$

where

$$J_r = m_1 l_1^2 + m_2 d_2^2 \qquad \text{and} \qquad B_r = 2 m_2 d_2 \dot{d}_2 \tag{13.26}$$

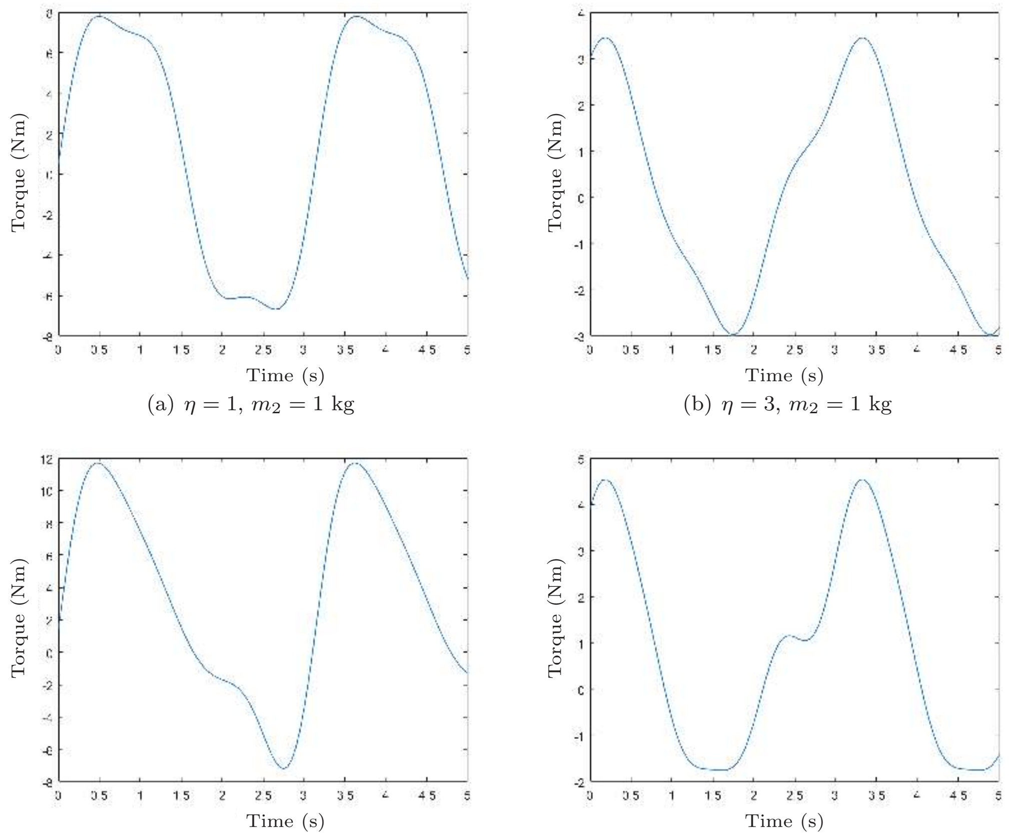
Using the servo dynamics equation (13.19), the torque exerted on the motor behind the gear transmission is

$$\tau = (\eta^2 J_m + J_r)\ddot{q} + (\eta^2 B_m + B_r)\dot{q} + \frac{1}{\eta}(m_1 l_1 + m_2 d_2)\, g \sin q_1 \tag{13.27}$$

To determine the torque profile, we need the joint velocities and accelerations. We can simply use the time differential of the proposed joint trajectories:

$$\begin{aligned} \dot{q}_1 &= \frac{3\pi}{2}\cos(3t) & \ddot{q}_1 &= -\frac{9\pi}{2}\sin(3t) \\ \dot{d}_2 &= 0.3\cos(3t+1) & \ddot{d}_2 &= -0.9\sin(3t+1) \end{aligned} \tag{13.28}$$

```
% Define symbolic variables
syms l_1 m_1 m_2 x_1 x_2 dx_1 dx_2 ddx_1 ddx_2 g real
vars = [x_1 dx_1 ddx_1 x_2 dx_2 ddx_2];
consts = [l_1 m_1 m_2 g];

% System parameters
n = 3;                          % Gear ratio
J_m = 1.0e-3;
B_m = 0.1;
J_r = m_1*l_1^2 + m_2*x_2^2;
B_r = 2*m_2*dx_1*x_2*dx_2;

% Convert symbolic to useable functions
J_r_f = matlabFunction(J_r,'Vars',{vars,consts});
B_r_f = matlabFunction(B_r,'Vars',{vars,consts});
D_f_ = matlabFunction((m_1*l_1 + m_2*x_2)*g*sin(x_1), ...
    'Vars',{vars,consts});

% Sub in values
l_1 = 0.5;
m_1 = 2;
m_2 = 1;        % Add 5 to this for 5kg load
g = 9.81;
consts = [l_1 m_1 m_2 g];

% Redefine functions with const values
J_e = @(x) n^2*J_m + J_r_f(x,consts);
B_e = @(x) n^2*B_m + B_r_f(x,consts);
D_f = @(x) D_f_(x,consts);

% Servo dynamics equation
t_1 = @(x,J,B) D_f(x)/n + B_e(x).*x(:,2) + J_e(x).*x(:,3);

% Generate time, velocity and acceleration vectors
t = linspace(0, 5, 101)';
s = 2;
x_1 = pi/2 * sin(s*t);
dx_1 = pi/2 * s * cos(s*t);
ddx_1 = pi/2 * -s^2 * sin(s*t);

x_2 = 0.1 * sin(s*t + 1);
dx_2 = 0.1 * s * cos(s*t + 1);
ddx_2 = 0.1 * -s^2 * sin(s*t + 1);

```



(a) $\eta = 1$, $m_2 = 1$ kg

(b) $\eta = 3$, $m_2 = 1$ kg

(c) $\eta = 1$, $m_2 = 6$ kg

(d) $\eta = 3$, $m_2 = 6$ kg

FIGURE 13.5
Servo motor torque profiles.

```
% Plot
x = [x_1 dx_1 ddx_1 x_2 dx_2 ddx_2];
plot(t,t_1(x))
xlabel('Time (s)')
ylabel('Torque (Nm)')
```

The resultant torques experienced by the servo motor are shown in Figure 13.5, which are generated by MATLAB script 13.1. We find that in a no-load scenario, a gear transmission with a ratio of $\eta = 3$ lowers driving torque from approximately 8 Nm to 3.5 Nm peak, or a reduction of around 44%. The load reduction is not quite $\frac{1}{3}$, due to the driving motor's dynamics caused by self-inertia and damping. However, with a 5 kg load (by adding 5 to m_2's mass), the gear transmission lowers driving torque from approximately 12 Nm to 4.5 Nm peak, or a reduction of around 37%. We see a greater reduction in torque due to the load having a greater effect on system dynamics than the servo motor.

Example 13.3 (Geared servo dynamics — RR manipulator): Consider an RR manipulator whose dynamic equation for the first joint is

$$\tau_1 = m_2 l_2^2 (\ddot{q}_1 + \ddot{q}_2) + m_2 l_1 l_2 \cos q_2 (2\ddot{q}_1 + \ddot{q}_2) + (m_1 + m_2) l_1^2 \ddot{q}_1 - m_2 l_1 l_2 \sin(q_2) \dot{q}_2^2 - 2 m_2 l_1 l_2 \sin(q_2) \dot{q}_1 \dot{q}_2 + m_2 l_2 g \cos(q_1 + q_2) + (m_1 + m_2) l_1 g \cos q_1 \quad (13.29)$$

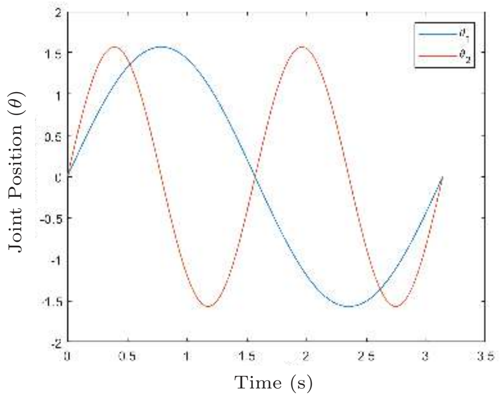


FIGURE 13.6
Joint trajectory for this example.

For the following trajectories over time $0 \leq t \leq 3.14$ seconds,

$$q_1 = \frac{\pi}{2}\sin(2t) \qquad \text{and} \qquad q_2 = \frac{\pi}{2}\sin(4t) \tag{13.30}$$

plot the servo motor load at the first joint for gear ratios $\eta = 1$ and 50. Note the peak torque and general characteristics of each profile. Assume the following servo parameters:

$$\begin{array}{llll} l_1 = 0.5 & l_2 = 0.3 & m_1 = 2 & m_2 = 1 \\ J_m = 1.0 \times 10^{-3} & B_m = 0.01 & g = 9.81 & \end{array} \tag{13.31}$$

where each mass is a point load.

Solution: For reference, Figure 13.6 shows the joint positions over the proposed time period.

First, determine inertia and friction coefficients of the robotic link load by collecting $\ddot{q}_1$ and $\dot{q}_1$ terms from the dynamic equation (13.29). We find that

$$J_r = l_1^2(m_1 + m_2) + l_2^2 m_2 + 2l_1 l_2 m_2 \cos q_2 \tag{13.32}$$

and

$$B_r = -2\dot{q}_2 l_1 l_2 m_2 \sin q_2 \tag{13.33}$$

Using the servo dynamics equation (13.19), the torque exerted on the motor behind the gear transmission for the first joint is

$$\tau = J_e\ddot{q} + B_e\dot{q} + \frac{1}{\eta}D_r \tag{13.34}$$

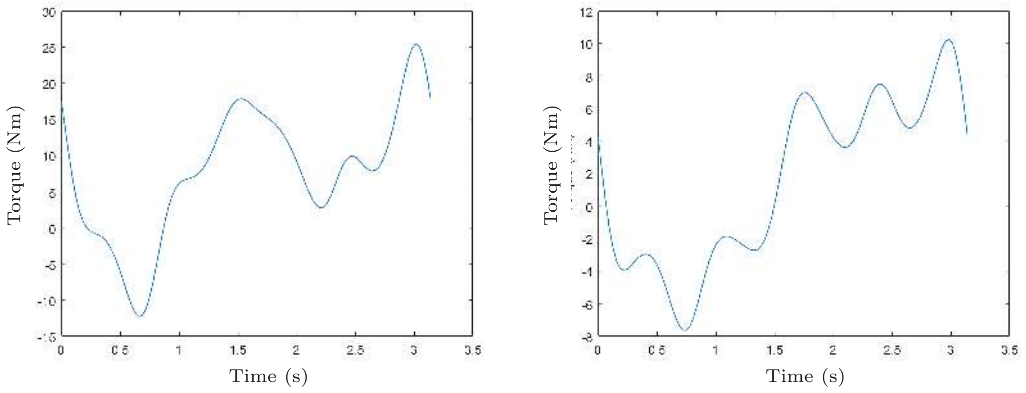
where

$$D_r = m_2 l_2^2 \ddot{q}_2 + m_2 l_2 g \cos(q_1 + q_2) + l_1 g (m_1 + m_2)\cos q_1 - m_2 l_2 \left(\dot{q}_2^2 \sin q_2 + \ddot{q}_2 \cos q_2\right) \tag{13.35}$$

Furthermore, the joint velocities and accelerations are

$$\begin{array}{ll} \dot{q}_1 = \pi\cos(2t) & \ddot{q}_1 = -2\pi\sin(2t) \\ \dot{q}_2 = 2\pi\cos(4t) & \ddot{q}_2 = -8\sin(2t) \end{array} \tag{13.36}$$

The servo motor torques as generated by MATLAB script 13.2 are shown in Figure 13.7.



(a) $\eta = 1$ (b) $\eta = 5$

FIGURE 13.7
Servo motor torque profiles.

```matlab
% NOTE: This script uses functions introduced in previous chapters!
% Define symbolic variables
syms l_1 l_2 m_1 m_2 x_1 x_2 dx_1 dx_2 ddx_1 ddx_2 g real
vars = [x_1 dx_1 ddx_1 x_2 dx_2 ddx_2];
consts = [l_1 l_2 m_1 m_2 g];

% Forward kinematics of RR
dht = [0 0 0 x_1; 0 l_1 0 x_2; 0 l_2 0 0];     % DH params
Tij = dh(dht);                                   % Find T matrices

% Newton-Euler dynamics to find torque equations
E = DynamicsNE(Tij,[dx_1 dx_2],[ddx_1 ddx_2],[l_1 0 0; l_2 0 0]',[m_1 m_2],[0 g 0]');
t_1 = expand(E(1));    % Torque of first joint

% Find link inertia, damping and other loads
J_r = simplify(subs(t_1-D_r, [dx_1 ddx_1], [0 1]));
B_r = simplify(subs(t_1-D_r, [dx_1 ddx_1], [1 0]));
D_r = simplify(subs(t_1,[dx_1 ddx_1], [0 0] ));

% Define function handles with constants
J_r_f = matlabFunction(J_r,'Vars',{vars,consts});
B_r_f = matlabFunction(B_r,'Vars',{vars,consts});
D_f_ = matlabFunction(D_r,'Vars',{vars,consts});

% Sub in values
n = 50;             % Gear ratio 1, 5, or 50
J_m = 1.0e-3;
B_m = 0.01;
l_1 = 0.5;
l_2 = 0.3;
m_1 = 2;
m_2 = 1;
g = 9.81;
consts = [l_1 l_2 m_1 m_2 g];

% Redefine functions with these consts
J_e = @(x) n^2*J_m + J_r_f(x,consts);
B_e = @(x) n^2*B_m + B_r_f(x,consts);
D_f = @(x) D_f_(x,consts);

% Servo dynamics equation for first joint
t_1 = @(x,J,B) D_f(x)/n + B_e(x).*x(:,2) + J_e(x).*x(:,3);

% Generate time, velocity and acceleration profiles
s = 2;
t = linspace(0, pi*s/2, 101)';
x_1 = pi/2 * sin(s*t);
dx_1 = pi/2 * s * cos(s*t);
ddx_1 = pi/2 * -s^2 * sin(s*t);

s = s*2;
x_2 = pi/2 * sin(s*t);
```

```
dx_2 = pi/2 * s * cos(s*t);
ddx_2 = pi/2 * -s^2 * sin(s*t);

% Plot
x = [x_1 dx_1 ddx_1 x_2 dx_2 ddx_2];
plot(t,t_1(x))
xlabel('Time (s)')
ylabel('Torque (Nm)')

figure
plot(t,x(:,[1 4]))
legend('\theta_1','\theta_2')
ylabel('Joint Position (rad)')
xlabel('Time (s)')
```

According to Figures 13.7(a) and 13.7(b), the gear reduction of 5 reduced peak torques from approximately 25 Nm to 10 Nm, or by a factor of 2.5. Again, the effect of gearing down is mitigated by the increased dynamic effects of the faster-spinning rotor, which cannot be mitigated behind the gear transmission.

13.3 Fixed Reference Tracking

In fixed reference tracking, we define a static reference point that defines the desired state in which we want the system to reach in finite time. In robot control, this is typically a joint position q_d to which we want the manipulator to move. This is known as a *step input* to the control system, which in the frequency domain is defined as $\dfrac{R}{s}$. There are many types of controllers that can achieve this, all varying in performance and complexity. In this section, we will analyse the output characteristics of three types of control systems whose servo dynamics are defined in (13.19): the proportional (P), proportional-derivative (PD), and proportional-integral-derivative (PID) controller. To simplify this analysis, we will use the frequency domain.

13.3.1 P Controller

One of the simplest forms of feedback controller is the proportional (P) gain controller. For a given input $U(s)$,

$$U(s) = K(q_d(s) - q(s)) + D(s) \tag{13.37}$$

where K is the proportional gain, q and q_d are the current and desired actuator positions, respectively, and $D(s)$ is some input disturbance due to the nonlinear effects of the manipulator. The open loop transfer function is

$$q(s) = \frac{Kq_d(s) + D(s)}{Js^2 + Bs} \tag{13.38}$$

and therefore, the closed-loop output transfer function is

$$q(s) = \frac{K}{p(s)}q_d + \frac{1}{p(s)}D(s) \tag{13.39}$$

where

$$p(s) = Js^2 + Bs + K \tag{13.40}$$

which is the closed-loop characteristic equation of the P controller. By analysing the coefficients of this polynomial, we can analyse the stability of the output (servo position) relative to different types of input, and determine the *steady-state* error (tracking performance) and *transient response* (disturbance rejection).

Stability Analysis

In order to have a stable closed-loop system, the closed-loop poles (the roots of the closed-loop characteristic polynomial) must be kept on the left-hand side of the s plane. That is, the real parts of the closed-loop poles must be negative. Assume that the closed-loop poles are $a + ib$ and $a - ib$, which are conjugate to each other. The closed-loop characteristic polynomial can be written in factorised form as

$$p(s) = J(s - a - ib)(s - a + ib) = Js^2 - 2Jas + J(a^2 + b^2) \tag{13.41}$$

To maintain stability, $a < 0$. Comparing (13.40) and (13.41), we find that

$$B = -2Ja \tag{13.42}$$

and upon rearranging,

$$a = -\frac{B}{2J} \tag{13.43}$$

Because a valid system must contain positive parameters, $J > 0$ and $B > 0$, we find that a will always be negative. That means the output of a P controller is *always* stable, regardless of the proportional gain value K!

Steady-State Error

To analyse the steady-state error, we use the *final value theorem*, which is determined by the equation

$$e_{ss} = \lim_{s \to 0} sE(s) \tag{13.44}$$

To analyse the steady-state error, we use the *final value theorem*, which is determined by the equation

$$E(s) = q_d(s) - q(s) \tag{13.45}$$

Referring to the output transfer function (13.39), substituting in (13.45) and rearranging for $E(s)$ we obtain the error transfer function

$$E(s) = \frac{Js^2 + Bs}{p(s)} q_d(s) - \frac{1}{p(s)} D(s) \tag{13.46}$$

noting the characteristic polynomial defined in (13.40). In a non-ideal system, for a given step input $U(s) = \dfrac{R}{s}$, we assume a disturbance step input is also generated $D(s) = \dfrac{D}{s}$. Finally, using (13.44), the steady-state error for a P controller is

$$\begin{aligned} e_{ss} = \lim_{s \to 0} sE(s) &= s\left[\frac{Js^2 + Bs}{p(s)} \frac{R}{s} - \frac{1}{p(s)} \frac{D}{s}\right] \\ &= R\frac{Js^2 + Bs}{p(s)} - \frac{D}{p(s)} \end{aligned}$$

$$= -\frac{D}{K} \tag{13.47}$$

which implies that steady-state error is nonzero for systems with input disturbance but can be reduced by increasing proportional gain K.

At this point, it seems that there are no implications of setting $K >> 0$ to minimise steady state error, as it does not affect the system's stability. However, the output transient response is also important to a well-functioning control system.

Transient Response

The P controller's characteristic equation (13.40) is of second-order. Therefore, we can analyse the system's response using the second-order equation

$$s^2 + 2\zeta\omega s + \omega^2 = 0 \tag{13.48}$$

where ω and ζ are the natural frequency and damping ratio, respectively. There are three types of stable transient responses governed by this characteristic equation.

- Overdamped response $\zeta > 1$, where the output response is slow to reach a steady state.
- Underdamped response $0 < \zeta < 1$, where the output reaches the reference point quickly, but with overshoot and oscillations.
- Critically damped response $\zeta = 1$, where the output reaches a steady state at the fastest speed without any of the features above.

In the control of robotic manipulators, a critically damped response is generally desired. Therefore, when gains are chosen, we like to set $\zeta = 1$. Comparing the P controller's characteristic equation (13.40) with the standard second-order characteristic equation (13.48), we find that

$$\frac{B}{J} = 2\zeta\omega \tag{13.49}$$

and

$$\frac{K}{J} = \omega^2 \tag{13.50}$$

Therefore, the relation of K on the damping ratio ζ is

$$\zeta = \frac{B}{2\sqrt{KJ}} \tag{13.51}$$

Here, we see the implications of increasing K in a proportional controller. Even though it decreases steady-state error, large values of K will likely drive the system into an underdamped response, which is highly undesirable in many robotic applications. To determine the maximum value of K that will yield up to a critically damped response, we solve for K after setting $\zeta \geq 1$ in equation (13.51). This yields

$$K \leq \frac{B^2}{4J} \tag{13.52}$$

where K must be less than this value to keep the system overdamped or equal to be critically damped. That means a P controller will always yield a high steady-state error if a critically damped response is still desired for systems where mechanical parameters are not favourable, such as for large inertial loads with low effective damping. Hence, a P controller would not be ideal for this scenario.

Example 13.4 (P controller): Consider a 0.3 m length rigid link of negligible mass. One end is attached to a revolute servo, which has a magnetic rotor field of 0.5 T, and the other end has a free-hanging mass of 2 kg. If the servo is at 0 rad when the rigid link is pointing downwards, determine the type of response to a step input of $\frac{\pi}{2}$ radians when $K = 0.1, 0.347$, and 3.47, and plot the first 10 seconds of the output. Assume there is no disturbance input.

Solution: The dynamic equation for the system is

$$\tau = ml^2\ddot{q} + b\dot{q} + mlg\sin q \tag{13.53}$$

where $m = 2$, $l = 0.3$, $b = 0.5$, and $g = -9.81$. Therefore, $J = ml^2 = 0.18$ and $B = 0.5$. Referring to (13.38), for a step input and no disturbance input, the open loop transfer function is

$$G(s) = \frac{K}{0.18s^2 + 0.5s} \tag{13.54}$$

The response characteristic is governed by Equation 13.51. We will use MATLAB to calculate ζ and generate the system response plots. The following script solves this problem.

```
l = 0.3;
m = 2;
B = 0.5;
J = m*l^2;
K = [0.1 0.347 3.47];
t = linspace(0, 10, 101);    % Time vector 10s over 101 samples
u = ones(1,101) * pi/2;      % Step input

for i = 1:3
    k = K(i);
    z(i) = B / (2*sqrt(k*J)); % Solve damping coeff, store in z
    % Display damping coefficient
    fprintf("For K = %5.3f, damping = %.3f.\n", k, z(i));
    w = sqrt(k/J);
    % Plot
    G = tf(k, [J B 0]);
    CLTF = feedback(G, 1);
    lsim(CLTF,u,t);
    hold on
end

% Generate legend
legend(arrayfun(@(x,y) ...
    sprintf('K = %5.3f, \\zeta = %.3f', x, y), K, z, 'Uni', 0))
```

```
For K = 0.100, damping = 1.863.
For K = 0.347, damping = 1.000.
For K = 3.470, damping = 0.316.
```

The output plot is shown in Figure 13.8.

We find that for $K = 0.1$, 0.347, and 3.47, the system results in an overdamped, critically damped, and underdamped response, respectively. Also note that with increasing K, the time to steady state does not significantly improve beyond $K = 0.347$, highlighting the limitations of P control in this scenario. We expect steady-state error to be zero due to a zero disturbance input.

13.3.2 PD Controller

A PD controller (Figure 13.9) utilises the sum of two signals to compensate for the error. The first one is proportional to the error utilised by the P controller, while the other is proportional to the derivative of the error. The ideal result is that this controller can compensate for current and *future error*. Consider the input $U(s)$ in the transfer function of the single joint; the input to the system due to a PD controller is given by

$$U(s) = (K_P + K_D s)(q_d(s) - q(s)) + D(s) \tag{13.55}$$

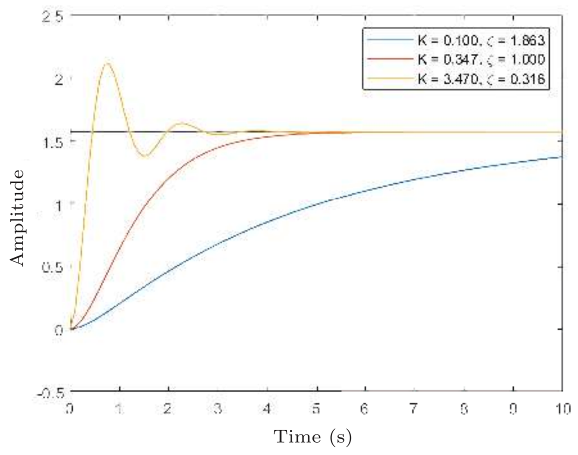


FIGURE 13.8
System response to $K = 0.1$, 0.347, and 3.47.

where K_P and K_D are the proportional gains and derivative gains, and $D(s)$ is the input disturbance due to the nonlinear effects of the manipulator. The closed-loop system with this PD controller is

$$q(s) = \frac{K_P + K_D s}{p(s)} q_d(s) + \frac{1}{p(s)} D(s) \tag{13.56}$$

where $p(s)$ is the closed-loop characteristic polynomial, given by

$$p(s) = Js^2 + (B + K_D)s + K_P \tag{13.57}$$

This is similar to the characteristic polynomial observed in the P controller, but notice now the dependence of differential gain K_D on the s coefficient. This allows us greater flexibility in controlling the output characteristics than a plain P controller. This becomes apparent by utilising the same methodology for the analysis of the P controller here.

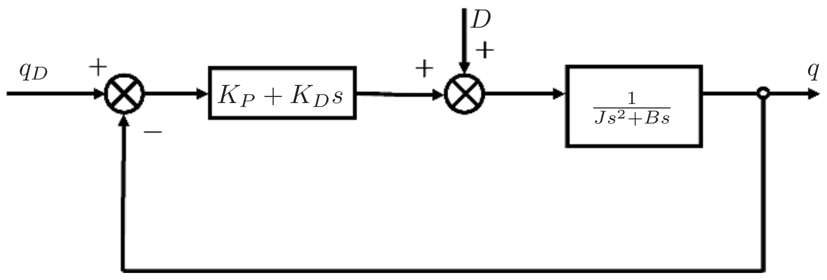


FIGURE 13.9
PD controller for a single joint.

Stability Analysis

Referencing (13.57) with the general factorised characteristic polynomial (13.41), we find two constraints that enforce output stability:

$$a = -\frac{B + K_D}{2J} \tag{13.58}$$

and

$$a^2 + b^2 = \frac{K_P}{J} \tag{13.59}$$

For stability, $a > 0$ for both constraints. In (13.58), applying this condition yields

$$\frac{B + K_D}{2J} > 0 \tag{13.60}$$

Since the inertia J and damping coefficient B of a physical system must be positive, we have a stable system as long as $K_D > 0$. Applying the same condition to the second constraint equation (13.59), and substituting in (13.58), solving for K_P we obtain

$$K_P \geq a^2 J = \frac{(B + K_D)^2}{4J} \tag{13.61}$$

The above two inequalities on K_D and K_P are the stability conditions for a PD controller on a single joint.

Stability via Routh Array

An alternative analysis for stability is to use Routh's stability criterion. Consider a general quadratic which represents the characteristic equation of a second-order control system

$$a_0 s^2 + a_1 s + a_2 = 0 \tag{13.62}$$

Then, the Routh array becomes

s^2	a_0	a_2
s^1	a_1	0
s^0	a_2	

According to the stability criterion, all coefficients on the first column of the array should be of the same sign. Because a_0 is generally positive, then for stability, coefficients a_1 and a_2 should also be positive. Referring to (13.57) where $a_0 = J$, $a_1 = B + K_D$, and $a_2 = K_P$, we find the gain conditions for stability are

$$K_D \geq -B \tag{13.63}$$

and

$$K_P \geq 0 \tag{13.64}$$

Steady-State Error

The closed-loop error transfer function, based on (13.56) is

$$E(s) = q_d(s) - q(s) = \frac{Js^2 + Bs}{p(s)} q_d(s) - \frac{1}{p(s)} D(s) \tag{13.65}$$

By applying a step $q_d(s) = \frac{R}{s}$ and a residual step disturbance input $D(s) = \frac{D}{s}$, we apply the final value theorem by using (13.44) on (13.65) yields

$$e_{ss} = \lim_{s\to 0} sE(s) = -\frac{D}{K_P} \tag{13.66}$$

Therefore, the PD controller has the same steady state error characteristics as a P controller, which is only caused by disturbance due to nonlinear actuator characteristics. This is of no improvement to the P controller, but is to be expected in this analysis due to the differential controller having no effect in steady-state conditions.

At this point, there seems to be no benefit in introducing a differential term on top of the P controller. However, as hinted in the characteristic equation of the PD controller (13.57), there is a dependence of K_D on the s-coefficient, which largely affects the transient response of the controller.

Transient Response

Comparing the characteristic equation of the PD controller (13.57) with the general form (13.48), we obtain two equations:

$$K_P = \omega^2 J \tag{13.67}$$

and

$$K_D = 2\zeta\omega J - B \tag{13.68}$$

Combining these two equations, we get

$$\zeta = \frac{K_D + B}{2\sqrt{K_P J}} \tag{13.69}$$

where for a critically damped system, we desire $\zeta = 1$. This equation indicates that the transient response is dependent on both gains, with K_D on the numerator and K_P at the denominator. This adds greater flexibility in tuning gains to achieve a low steady-state error whilst maintaining a critically damped system. For instance, if K_P is increased to reduce steady-state error, then K_D can be increased appropriately to bring the system back to a critically damped response.

Example 13.5 (PD controller design): Consider a servo with a rotor inertia of 1×10^{-3} kg.m^2, and a magnetic rotor field of 0.5 T. Assuming no disturbance input, then design a PD controller for this servo that achieves minimal settling time with a natural frequency of 500 rad/s. Plot the servo's response to a unit step input.

Solution: To minimise the settling time for a second-order system, we set $\zeta = 1$ and $\omega = 500$ for a natural frequency of 500 rad/s. With $J = 1\times10^{-3}$ and $B = 0.5$, according to (13.67),

$$K_P = 250 \text{ and } K_D = 0.5 \tag{13.70}$$

To plot the system, we need to find the closed-loop transfer function for Figure 13.9. Given that there is no disturbance input, the open loop transfer function G simplifies to

$$G(s) = \frac{K_P + K_D s}{Js^2 + Bs} \tag{13.71}$$

The closed-loop transfer function is then

$$\frac{q(s)}{q_d(s)} = \frac{G(s)}{1 + G(s)}$$

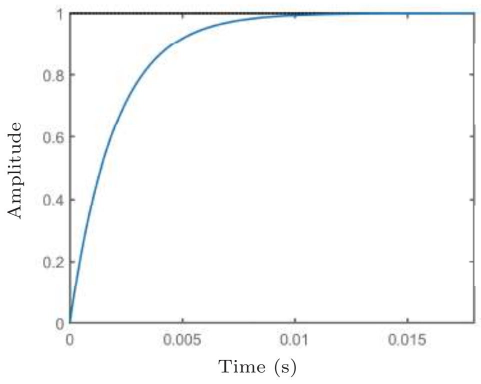


FIGURE 13.10
Servo response to a fixed reference input under PD control.

$$= \frac{K_D s + K_P}{Js^2 + (K_D + B)s + K_P}$$
$$= \frac{0.5s + 250}{0.001s^2 + s + 250} \tag{13.72}$$

We can then plot this transfer function in MATLAB using the following code, which represents the system response to a step (fixed reference) input (Figure 13.10).

```
    sys = tf([0.5 250], [0.001 1 250])
    step(sys)
```

Although the goal is to achieve a critically damped response, the output response will not reflect this due to the presence of a first-order zero.

Example 13.6 (Steady state error): Suppose that steady-state conditions for the same system are achieved for a unit step input. After $t = 0.025$ s, there is a 50-unit step disturbance input. Find the steady-state error and plot the output response under the same gain values to verify the answer.

Solution: The steady-state error from a disturbance input is found by evaluating (13.66), which is

$$-\frac{D}{K_P} = -\frac{50}{250} = -0.2 \tag{13.73}$$

which means that with disturbance, the output should exceed the input q_d by 0.2 units.

To plot the response, we utilise the closed-loop transfer function from (13.72), which is a multi-input single-output system. Note that the disturbance input portion of the CLTF is a simple second-order system with no zeros. Therefore, we expect a standard, critically-damped second-order response for the gains chosen in the previous example. We can then plot this transfer function in MATLAB using the following code representing the system response to a step disturbance input.

```
    CLTF = tf({[Kd Kp], 1},[J B+Kd Kp])    % Create two-input system
    t = linspace(0,0.05,100)               % Define time vector
    u = [ones(100,1) zeros([100 1])]       % Create input matrix
    % Input 1: theta_d
    % Input 2: disturbance
    u(t>0.025,2) = 50;                % Set input 2 = 50 after t>0.025
    lsim(CLTF,u,t)                    % Generate plot
    slim([0 1.2])                     % Adjust y-axis

```

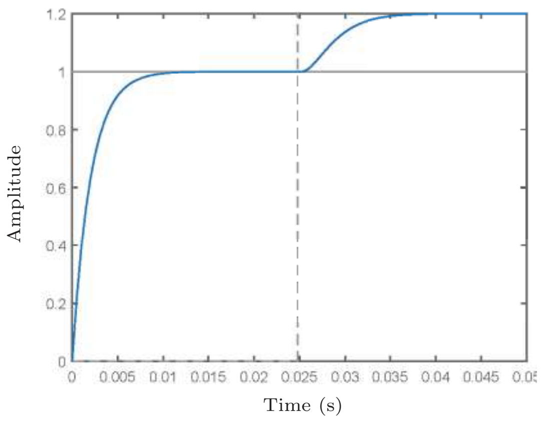


FIGURE 13.11
Servo response to a 50-unit step disturbance at $t = 0.025$ s at initial steady-state conditions.

From Figure 13.11, we can identify that the steady-state error is 0.2 units above the reference level once the final steady state is achieved. A nonzero steady-state error from a disturbance means that the PD controller has poor disturbance rejection.

13.3.3 PID Controller

As shown in (13.65) and from the examples, a PD controller has poor disturbance rejection due to a nonzero steady-state error to a nonzero disturbance. To reduce this error, we introduce the PID controller (Figure 13.12), which adds an integral signal to the PD controller. This signal is proportional to the integral of previous errors, which is expected to compensate for the historical accumulation of past errors. Due to its flexibility in controlling many types of signals, the PID controller finds many applications in the industry.

Consider the input $U(s)$ in the transfer function of the single joint; the input to the system due to a PID controller is given by

$$U(s) = (K_P + K_D s + \frac{K_I}{s})(q_d(s) - q(s)) + D(s) \tag{13.74}$$

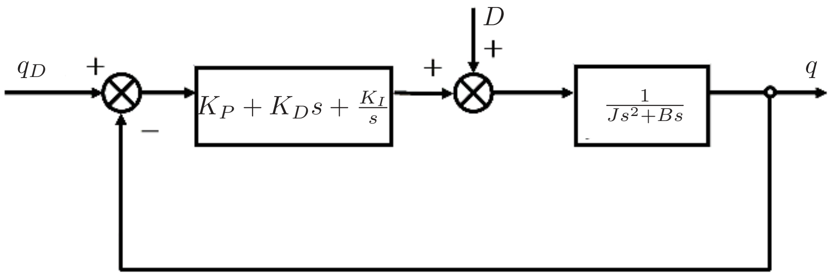


FIGURE 13.12
Block diagram of a PID controller.

where K_I is the newly introduced integral gain. The closed-loop system with this PID controller is shown in Figure 13.12. The output of this system can be readily derived as

$$q(s) = \frac{K_D s^2 + K_P s + K_I}{p(s)} q_d(s) + \frac{s}{p(s)} D(s) \tag{13.75}$$

where $p(s)$ is the closed-loop characteristic polynomial

$$p(s) = Js^3 + (B + K_D)s^2 + K_P s + K_I \tag{13.76}$$

Stability Analysis

To find the stability of this third-order closed-loop system without solving the closed-loop characteristic polynomial, we utilise Routh's stability criterion. The Routh array of (13.76) is shown below.

s^3	J	K_P
s^2	$B + K_D$	K_I
s^1	$K_P - J\dfrac{K_I}{B + K_D}$	
s^0	K_I	

According to the Routh criterion, the entries in the second column must have the same sign for a stable system, i.e., $K_D > -B$, $K_I > 0$, and $K_P > J\dfrac{K_I}{B + K_D}$. Typically, all gains should be positive, so the only remaining criterion that governs the stability of the PID controller is

$$K_I < \frac{(B + K_D)K_P}{J} \tag{13.77}$$

Steady-State Analysis

The steady-state error can be found in the frequency domain with respect to a step reference input $\dfrac{R}{s}$ and a step disturbance $\dfrac{D}{s}$ as the time approaches infinity.

$$\begin{aligned} e_{ss} &= \lim_{s \to 0} s\,(q_d(s) - q(s)) \\ &= \lim_{s \to 0} s \left(\frac{Js^3 + Bs^2}{p(s)} q_d(s) - \frac{s}{p(s)} D(s) \right) \\ &= \lim_{s \to 0} s \left(\frac{Js^3 + Bs^2}{p(s)} \frac{R}{s} - \frac{s}{p(s)} \frac{D}{s} \right) \\ &= \lim_{s \to 0} s \left(\frac{JRs^2 + BRs}{p(s)} - \frac{D}{p(s)} \right) = 0 \end{aligned} \tag{13.78}$$

We can now see that the PID can eliminate the error caused by a step disturbance. The conclusion can also be drawn from the fact that the control system shown in Figure 13.14 is *Type 1*, with one open-loop pole at zero. It is known that Type 1 systems have no steady-state error with respect to a step input.

Tuning Gains

Tuning PID gains can be difficult, as transient response and overshoot performance must be balanced while maintaining the stability of the system. A common strategy is to tune

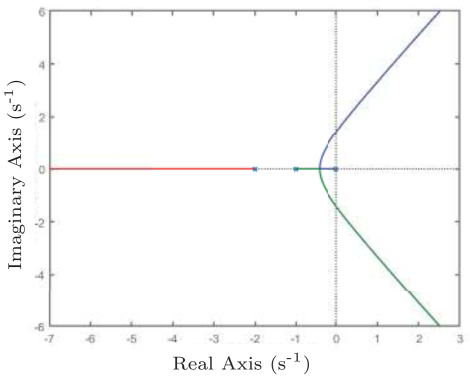


FIGURE 13.13
PID root locus.

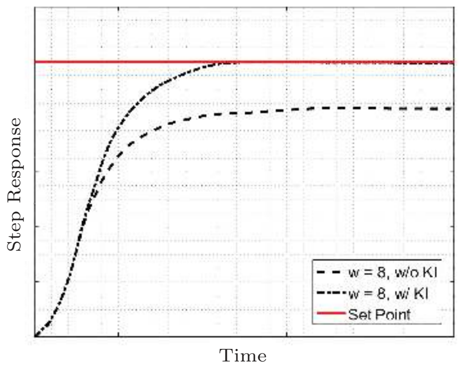


FIGURE 13.14
The response of the PID controlled single joint.

K_P and K_D to achieve the desired transient response first, then tune K_i so that it reduces or eliminates steady-state error without affecting stability. Due to its negative effects on transient response, $K_i = 0$ in many robot controllers, but if the integral term is absolutely needed to control steady-state error, then an *integrator anti-windup* could be used that limits the maximum integral error. The effects of K_I can be seen on the PID root locus plot in Figure 13.13, where $J = 1$ and gains are initially set to $K_D + B = 3$ and $K_P = 2$.

When $K_I = 0$, two roots lie on the negative real axis, and one root lies at zero, which indicates a fairly ideal transient response with no oscillations in the output. However, as K_I increases, the leftmost root heads towards negative infinity, while the other two roots move towards each other, eventually meeting and breaking away from the real axis. Once this happens, oscillations occur at the output that progressively worsens with increasing K_I. Once the imaginary roots cross to the right half of the plane, then the system becomes unstable.

Example 13.7 (PID tuning): Consider a single joint dynamics with J=1 and B=1. Find the PID controller gains to achieve the natural frequency of $\omega = 8$.

Solution: The best practice is to tune the PD portion of the controller before introducing I to remove the steady-state error. Based on Example 13.5, the desired PD gains to achieve ω=8 are K_P=64 and K_D=15. Then according to (13.77), to maintain stability, the condition for K_I must satisfy

$$K_I < \frac{(B + K_D)K_P}{J} = \frac{16 \times 64}{1} = 1024 \tag{13.79}$$

Therefore, we can safely select $K_I = 500$. The response of the PID-controlled single joint with respect to a unit step input is approximated in Figure 13.14.

13.4 Error Dynamics

So far, we have analysed the output signal properties of the P, PD, and PID controllers to a fixed reference (step) input. While we have derived the output transient response and steady-state error of each of these systems based on its second-order characteristic equation, the response is only relative to a constant input signal. In this section, we will generalise the discussion of output transient response and signal error to *time-variant* reference signals. The error within a control law is defined as

$$e(t) = q_d(t) - q(t) \tag{13.80}$$

where $e(t)$ is the difference between the desired input $q_d(t)$ and the actual output signal $q(t)$. The quantity q usually represents a servo angle, but this concept can be generalised to other signals. Furthermore, we can further extend this concept to the evolution or *change in error*, as described by a linear ordinary different equation of the form

$$a_p e^{(p)} + a_{p-1} e^{(p-1)} + ... + a_2 \ddot{e} + a_1 \dot{e} + a_0 e = c \tag{13.81}$$

where a are coefficients of the (p)-th order differential of the error q_e. This equation represents the *error dynamics* of a control law. In this section, we will study the error dynamics of a closed-loop control system for a single-DoF joint and up to the second order. We will base the error modelling on a linear mass-spring-damper system, where the second-order error can be written as

$$m\ddot{e}(t) + b\dot{e}(t) + ke(t) = 0 \tag{13.82}$$

where m, b, and k are the mass, damping coefficient, and spring constants, respectively.

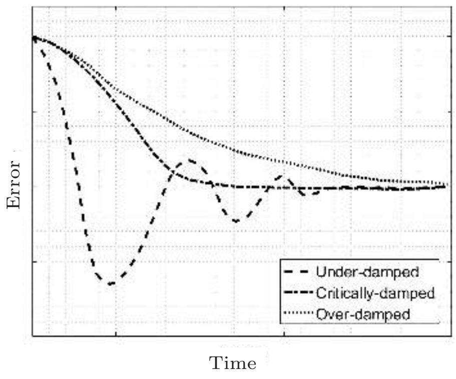


FIGURE 13.15
Over-damped, under-damped, and critical-damped error dynamics.

13.4.1 First-Order Error Dynamics

The first-order error dynamics, based on (13.82), can be written in the form

$$b\dot{e}(t) + ke(t) = 0 \tag{13.83}$$

which can be rearranged to

$$\dot{e}(t) + \frac{1}{T}e(t) = 0 \tag{13.84}$$

with $T = \frac{b}{k}$, and is the time constant of the first order ODE. It has a solution

$$e(t) = e(0)\exp\left(-\frac{t}{T}\right) \tag{13.85}$$

where $e(0)$ is the initial error of the system. The time constant T determines the time for error to decay 37% of initial error $e(0)$.

13.4.2 Second-Order Error Dynamics

The second-order dynamics

$$\ddot{e} + \frac{b}{m}\dot{e} + \frac{k}{m}e = 0 \tag{13.86}$$

can be written in the form

$$\ddot{e}(t) + 2\zeta\omega\dot{e}(t) + \omega^2 e(t) + d(t) = 0 \tag{13.87}$$

where ω and ζ are the natural frequency and damping ratio, respectively, and $d(t)$ is the disturbance in the time domain. The second-order error dynamics can be over-damped, under-damped, and critical-damped systems, as shown in Figure 13.15.

A critically damped system is desired in robotic applications ($\zeta = 0$). Therefore, we have

$$\ddot{e}(t) + 2\omega\dot{e}(t) + \omega^2 e(t) + d(t) = 0 \tag{13.88}$$

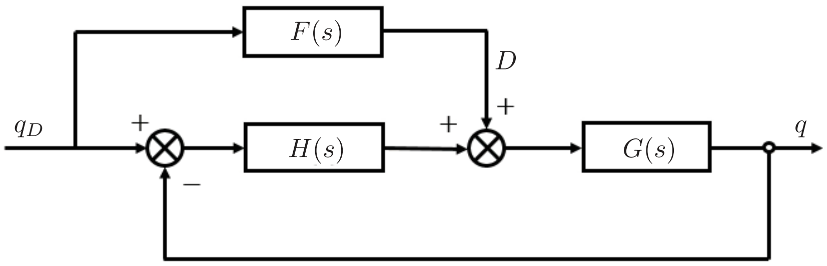


FIGURE 13.16
Feedforward control scheme.

With such critically damped error dynamics, the error is suppressed within a minimum time period without triggering oscillations. Without considering the disturbance, the above error dynamics can be written in the s domain as

$$E(s)(s^2 + 2\omega s + \omega^2) = 0 \tag{13.89}$$

Since $E(s) = q_d(s) - q(s)$, we have

$$\frac{q(s)}{q_d(s)} = 1 \tag{13.90}$$

which means that the desired closed-loop transfer function is 1. However, any control system has disturbances. One objective in designing a controller is to minimise the disturbance in the error dynamics. For example, the error dynamics of the PD controller shown in Figure 13.9 is given by (13.56), i.e.,

$$E(s)(Js^2 + (B + K_D)s + K_P) = (Js^2 + Bs)q_d(s) - D(s) \tag{13.91}$$

where the RHS appears as the disturbance including the modelled term $(Js^2 + Bs)q_d(s)$. One way to eliminate this term from the disturbance in (13.91) is to use the feedforward control scheme, as shown in Figure 13.16. The closed-loop transfer function is

$$T = \frac{q(s)}{q_d(s)} = \frac{G(s)F(s) + G(s)H(s)}{1 + G(s)H(s)} \tag{13.92}$$

where $F(s)$ is the feedforward transfer function. Using the condition of the unity transfer function, we have

$$F(s) = \frac{1}{G(s)} \tag{13.93}$$

This control law can be written as

$$V(s) = F(s)q_d + H(s)(q_d - q) \tag{13.94}$$

Apply the feedforward control to a single robotic joint as shown in Figure 13.17. The control law in time domain is given by

$$v(t) = J\ddot{q}_d + B\dot{q}_d + K_D(\dot{q}_d - \dot{q}) + K_p(q_d - q) \tag{13.95}$$

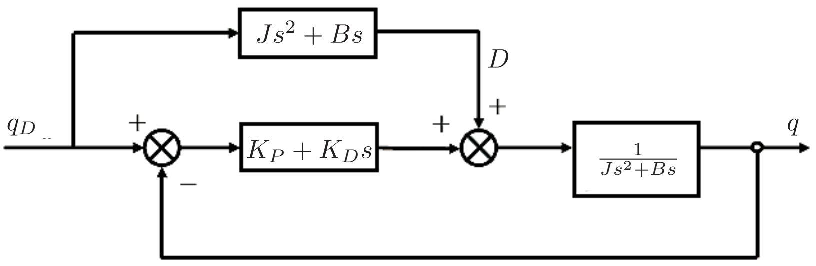


FIGURE 13.17
The PD and feedforward control on a single joint.

The closed-loop dynamics is derived as

$$J\ddot{q} + B\dot{q} = v(t) + d(t) \tag{13.96}$$

which yields the error dynamics:

$$J\ddot{e}(t) + (B + K_D)\dot{e}(t) + K_P e(t) + d(t) = 0 \tag{13.97}$$

where $e(t) = q_d(t) - q(t)$ and $d(t)$ is the disturbance. According to (13.88), we have

$$\frac{B + K_D}{J} = 2\omega, \quad \frac{K_P}{J} = \omega^2 \tag{13.98}$$

leading to

$$K_D = 2J\omega - B, \quad K_P = J\omega^2 \tag{13.99}$$

13.4.3 Velocity Control

We can extend the error dynamics to joint velocity control. Beginning with the desired error dynamics

$$\ddot{e} + 2\omega\dot{e} + \omega^2 e = 0 \tag{13.100}$$

we can define the error as

$$\ddot{e} = \ddot{q}_d - \ddot{q} \tag{13.101}$$

which yields

$$\ddot{q} = \ddot{q}_d + 2\omega\dot{e} + \omega^2 e \tag{13.102}$$

Taking the time integral yields the output equation

$$\dot{q} = \dot{q}_d + 2\omega e + \omega^2 \int e dt \tag{13.103}$$

where $\dot{q}_d$ is a feedforward signal, $K_P = 2\omega$, and $K_I = \omega^2$, which are feedback signals in a PI controller. Equation (13.103) will yield the desired error dynamics defined in (13.100).

If no feedforward signal is available ($q_d = 0$), then (13.103) becomes

$$\dot{q} = 2\omega e + \omega^2 \int e dt \tag{13.104}$$

in which

$$\ddot{q} = 2\omega\dot{e} + \omega^2 e \tag{13.105}$$

where after substituting the error definition (13.101), yields

$$\ddot{q}_d = \ddot{e} + 2\omega\dot{e} + \omega^2 e \tag{13.106}$$

13.5 Conclusion

In this chapter, we introduced PD and PID controllers that can be used to control the servos in a robotic system. Firstly, we derived the dynamics of a servo by utilising the common laws of physics to create a transfer function that relates servo output position with input voltage. With this transfer function, we are then able to derive a control system in a negative feedback loop to control this servo, and analyse it using control theory.

The PD controller compensates for errors between input and output servo positions by utilising the sum of proportional and differential errors. This control scheme seeks to minimise proportional error and suppress future error by measuring differential error. The resulting control scheme is of second order with two parameters to tune, which affects stability, steady-state error and transient response. For a fixed reference input (step servo position input), this control system can never reach a fixed input reference, as it has a constant, nonzero, steady-state error.

The PID controller utilises the integral of the measured error to compensate for the error, as well as proportional and differential error terms, as with the PD controller. Although this results in a third-order control system in which transient analysis is no longer trivial, stability analysis can be performed using the Routh array, and steady-state error can still be calculated in the Laplace domain. For a fixed reference input (step servo position input), the steady-state error for a PID controller is found to be zero for all stable cases. However, due to the system being of third order, PID tuning to achieve a stable output with a desired transient response can be difficult without experience or following a set procedure.

Although fixed reference (constant) inputs are easy to analyse, they are not practical, as desired trajectories in robotics are usually time-variant. In this scenario, we wish to track a changing reference input. To analyse system stability, and transient and steady-state errors of this type of input, we analyse the error dynamics as a result of time-varying inputs, generated by the control law. To ensure the control system can handle disturbances in a changing reference input, we model this as a second input into the control system and introduce a feedforward transfer function to eliminate this input from the final transfer function. This was demonstrated for the PD controller.

13.6 Exercises

Problem 1. Prove that the torque equation for the mechanism shown in Figure 13.18 is

$$T = \left[(J_a + J_{g1}) + \frac{(J_L + J_{g2})}{r^2}\right]\ddot{q}_m + \left[b_m + \frac{b}{r^2}\right]\dot{q}_m$$

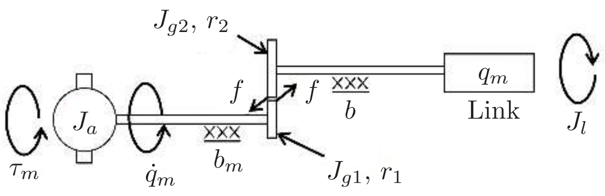


FIGURE 13.18
Mechanism for Problem 1.

where gear ratio $r = \frac{R_2}{R_1} = \frac{q_m}{q}$. What are the effective inertia and effecting damping terms of the system?

Problem 2. In a system like that shown in Figure 13.19, find the criteria for K_D and K_P of a PD controller such that the system is never unstable and never underdamped. Design the PD control to control τ (solve the torque balance equations for τ, rather than τ_m). The system possesses an un-modelled resonance due to an end-point stiffness K (on the non-motor side, not shown in the figure). Any damping terms in the system are negligible. Also, $\eta = \frac{R_2}{R_1} > 1$.

Problem 3. Design a trajectory-following controller for a system with dynamics given by

$$f = a(x+10)^2\ddot{x} + b\dot{x}^2 + c\dot{x} + d\sin(x)$$

where f and x are the input force and output displacement, respectively, while a, b, c, d are nonzero. The requirement is that errors are suppressed in a critically damped fashion over all configurations.

Problem 4. Consider a robotic system with the dynamic equation given by

$$\tau = M(q)\ddot{q} + V(q,\dot{q}) + G(q)$$

1. Construct the partitioned controller scheme to control this robot, based on its dynamic equation, so that the robot can be critically damped over the whole workspace. Here, we assume that all dynamic parameters are perfectly known.
2. Draw a block diagram showing this controller scheme. Show the corresponding equations inside the blocks of the block diagram.

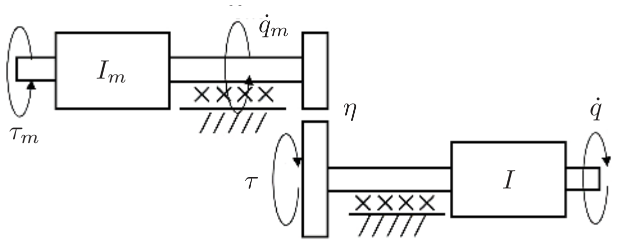


FIGURE 13.19
A geared system.

14

Computed Torque Control

In Chapter 13, we discussed the controllers to handle the computed torque control of a single joint. However, it is far from sufficient to assume linear system behaviour, especially when robot dynamics is factored in. This chapter aims to fill the gap by introducing the two-part inner-outer control loop to achieve stable non-linear computed torque control.

14.1 SISO Computed Torque Control

Consider a single-DoF system with open-loop dynamics given by

$$f = h_1(x)\ddot{x} + h_2(x, \dot{x}) \tag{14.1}$$

where f and x are the force input and displacement output, respectively, while $h_1(x)$ and $h_2(x, \dot{x})$ are nonlinear functions in general. We will not be able to use a linear controller to force this system to follow a given trajectory precisely, because the same gains cannot handle varying $h_1(x)$ and $h_2(x, \dot{x})$. One strategy is to linearise the system first by utilising the known dynamics of the system. We define the first control law as

$$f = h_1(x)f' + h_2(x, \dot{x}) \tag{14.2}$$

where f' is a new artificial input. With this control law, the original dynamics becomes

$$f' = \ddot{x} \tag{14.3}$$

which is the dynamics of a unit-mass-only system. The desired critically damped error dynamics is given by

$$\ddot{e} + K_D\dot{e} + K_P e = 0 \tag{14.4}$$

where error $e = x_d(t) - x(t)$, and $x_d(t)$ is the desired trajectory. Further, we have $K_D = 2\omega$ and $K_P = \omega^2$. Substituting the new dynamics (14.3) based on the first control law into (14.4), we have

$$f' = \ddot{x_d} + K_D\dot{e} + K_P e \tag{14.5}$$

which is the second control law to achieve the desired error dynamics. Combining two control laws together, we have the complete computed controller as

$$f = h_1(x)(\ddot{x_d} + K_D\dot{e} + K_P e) + h_2(x, \dot{x}) \tag{14.6}$$

The controller is shown in Figure 14.1.

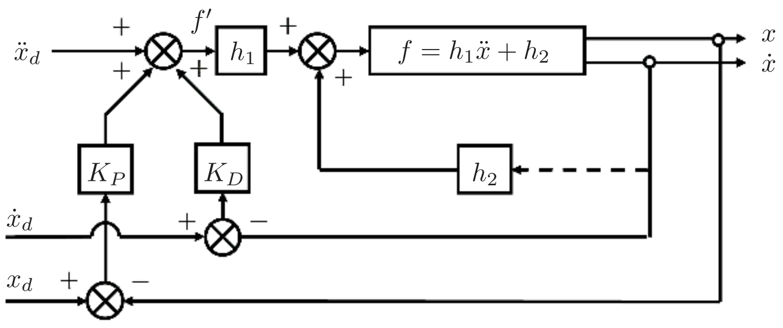


FIGURE 14.1
Computed torque controller for SISO.

Example 14.1 (Non-linear PD controller): Consider a system with the following dynamics

$$\tau_1 = (3 + d_2^2)\ddot{q}_1 + 2d_2\dot{d}_2\dot{q}_1 + (1 + d_2)g\sin q_1 \tag{14.7}$$

where τ_1, q_1, and d_2 are the first joint torque, the first joint angle, and the second joint distance, respectively. Design a PD controller to track a changing reference.
Solution: The first law is given by

$$\tau_1 = h_1\tau_1' + h_2 \tag{14.8}$$

where

$$h_1 = (3 + d_2^2) \tag{14.9}$$
$$h_2 = 2d_2\dot{d}_2\dot{q}_1 + (1 + d_2)g\sin q_1 \tag{14.10}$$

For a PD controller, the second law is given by

$$\tau_1' = \ddot{q}_{1d} + K_D\dot{e}_1 + K_P e_1 \tag{14.11}$$

Substituting τ_1' into the first law gives

$$\tau_1 = (3 + d_2^2)(\ddot{q}_{1d} + K_D\dot{e}_1 + K_P e_1) + \Big(2d_2\dot{d}_2\dot{q}_1 + (1 + d_2)g\sin q_1\Big) \tag{14.12}$$

14.2 MIMO Computed Torque Control

The dynamic equations of a robotic system are given by

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) \tag{14.13}$$

which can be written as

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{14.14}$$

where $\mathbf{u}$ and $\mathbf{q}$ are used to represent the control input vector $\boldsymbol{\tau}$ and general coordinate vector $\mathbf{q}$ of the control system. $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{B}(\mathbf{q})$ are the centrifugal and Coriolis matrix and damping coefficient matrix, respectively, which are highly nonlinear in general. We can use a similar law to the first law in the SISO controller (14.2) to linearise the dynamics:

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\,\mathbf{a} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{14.15}$$

where $\mathbf{a}$ is an artificially introduced new input. The linearised dynamics under this control law is given by

$$\mathbf{a} = \ddot{\mathbf{q}} \tag{14.16}$$

where all nonlinear dynamic terms are cancelled out if the model is *perfect*. In order to achieve the desired error dynamics, a PD controller is designed by

$$\mathbf{a} = \ddot{\mathbf{q}}_d + \mathbf{K}_D\,\dot{\mathbf{e}} + \mathbf{K}_P\,\mathbf{e} \tag{14.17}$$

with $\mathbf{e} = \dot{\mathbf{q}}_d - \mathbf{q}$, and $\dot{\mathbf{q}}_d$ is the desired joint trajectories. The resulting error dynamics under this control law is

$$\ddot{\mathbf{e}} + \mathbf{K}_D\,\dot{\mathbf{e}} + \mathbf{K}_P\,\mathbf{e} = \mathbf{0} \tag{14.18}$$

which is the final error dynamics. Since $\mathbf{K}_D$ and $\mathbf{K}_P$ are diagonal gain matrices, the error dynamics equations can be further decoupled as

$$\ddot{e}_i + K_{Di}\dot{e}_i + K_{Pi}e_i = 0 \tag{14.19}$$

which determines the performance of the system with respect to the initial errors and disturbances. Again, we prefer a critically-damped system by selecting $K_{Di} = 2\omega_i$ and $K_{Pi} = \omega_i^2$, where ω_i is the natural frequency that determines the speed of response or the rate of decay of tracking error. Hence, the gains become

$$\mathbf{K}_D = \begin{bmatrix} 2\omega_1 & 0 & \cdots & 0 \\ 0 & 2\omega_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2\omega_n \end{bmatrix}, \quad \mathbf{K}_P = \begin{bmatrix} \omega_1^2 & 0 & \cdots & 0 \\ 0 & \omega_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \omega_n^2 \end{bmatrix} \tag{14.20}$$

Control laws (14.15) and (14.17) are usually called the inner loop and the outer loop control, respectively, as illustrated in Figure 14.2. The total control law is then obtained by combining these two laws together:

$$\mathbf{u} = \mathbf{M}(\mathbf{q})(\ddot{\mathbf{q}}_d + \mathbf{K}_D\,\dot{\mathbf{e}} + \mathbf{K}_P\,\mathbf{e}) + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{14.21}$$

Example 14.2 (2R non-linear PD control): The system shown in Figure 14.3 has the following dynamics:

$$\begin{aligned}\tau_1 &= m_2l_2^2(\ddot{q}_1 + \ddot{q}_2) + m_2l_1l_2c_2(2\ddot{q}_1 + \ddot{q}_2) + (m_1+m_2)l_1^2\ddot{q}_1 \\ &\qquad - m_2l_1l_2\dot{q}_2^2s_2 - 2m_2l_1l_2\dot{q}_1\dot{q}_2s_2 + m_2l_2gc_{12} + (m_1+m_2)l_1gc_1 && (14.22)\\ \tau_2 &= m_2l_2\left[l_2(\ddot{q}_1+\ddot{q}_2) + l_1\ddot{q}_1c_2 + l_1\dot{q}_1^2s_2 + gc_{12}\right] && (14.23)\end{aligned}$$

where τ_1, q_1, and q_2 are the first joint torque, its angle, and the second joint angle, respectively. Design a PD controller for this manipulator to track a changing reference.
Solution: Rearrange the dynamics equations to find the mass matrix $\mathbf{M}$, centrifugal and Coriolis matrix $\mathbf{C}$, and gravity matrix $\mathbf{G}$:

$$\mathbf{M} = \begin{bmatrix} m_2l_2^2 + 2m_2l_1l_2c_2 + m_1l_1^2 & m_2l_2^2 + m_2l_1l_2c_2 + m_2l_1^2 \\ m_2l_2^2 + m_2l_1l_2c_2 & m_2l_2^2 \end{bmatrix} \tag{14.24}$$

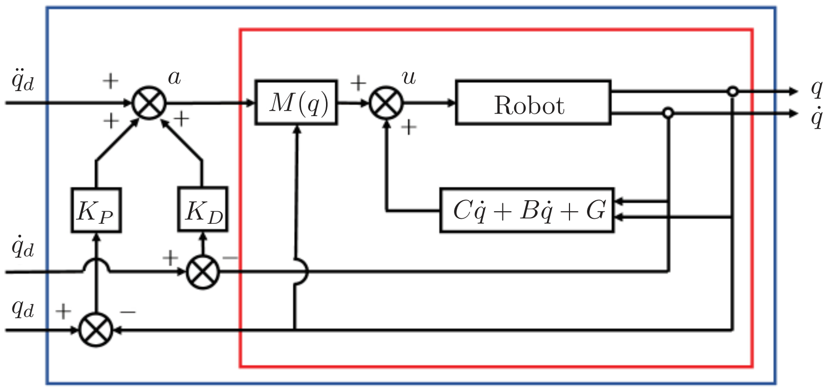


FIGURE 14.2
PD controller for a robotic manipulator.

$$\mathbf{C} = \begin{bmatrix} -2m_2l_1l_2s_2\dot{q}_2 & -m_2l_1l_2s_2\dot{q}_2 \\ m_2l_2l_1\dot{q}_1s_2 & 0 \end{bmatrix} \tag{14.25}$$

$$\mathbf{G} = \begin{bmatrix} (m_1+m_2)l_1gc_1 \\ m_2l_2gc_{12} \end{bmatrix} \tag{14.26}$$

Linearising the dynamics yields

$$\boldsymbol{\tau} = \mathbf{M}\ \mathbf{a} + \mathbf{C}\ \dot{\mathbf{q}} + \mathbf{G} \tag{14.27}$$

where for a PD controller

$$\mathbf{a} = \ddot{\mathbf{q}}_d + \mathbf{K}_D\ \dot{\mathbf{e}} + \mathbf{K}_P\ \mathbf{e} \tag{14.28}$$

Substituting a into the linearised dynamics gives

$$\boldsymbol{\tau} = \mathbf{M}(\ddot{\mathbf{q}}_d + \mathbf{K}_D\ \dot{\mathbf{e}} + \mathbf{K}_P\ \mathbf{e}) + \mathbf{C}\ \dot{\mathbf{q}} + \mathbf{G} \tag{14.29}$$

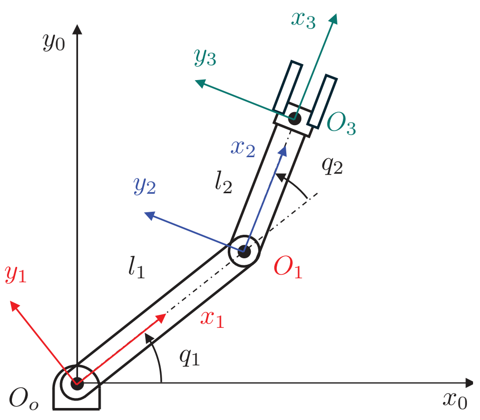


FIGURE 14.3
A 2R manipulator.

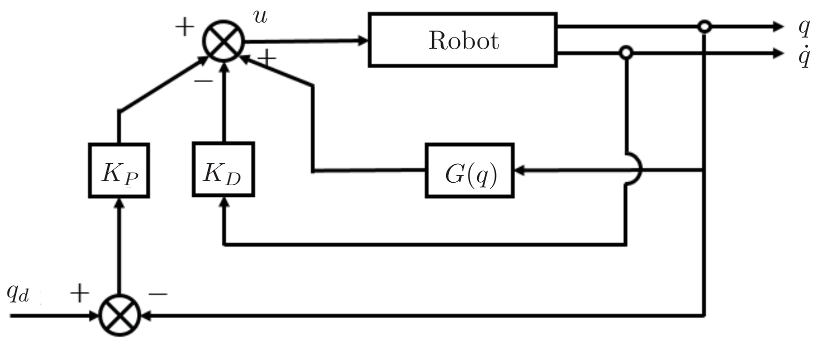


FIGURE 14.4
The control scheme of the PD controller with gravity compensation.

14.3 Controller with Gravity Compensation

The compensation of the complete nonlinear dynamics of a robotic manipulator requires accurate modelling, which is difficult or even impossible to achieve due to the lack of knowledge. Furthermore, the model requires high computation power to calculate in real time, which may not be possible in embedded systems with power constraints. The purpose of gravity compensation is to cancel the nonlinear effects of the gravity term in the original robotic dynamics, allowing the use of simpler PD or PID controllers.

A PD controller with gravity compensation for a changing reference is given by

$$\mathbf{u}(t) = \ddot{\mathbf{q}}_d + \mathbf{K}_P\,\mathbf{e} + \mathbf{K}_D\,\dot{\mathbf{e}} + \mathbf{G}(\mathbf{q}) \tag{14.30}$$

where $\mathbf{K}_P = \text{Diag}(K_{P1}, \ldots, K_{Pn})$ and $\mathbf{K}_D = \text{Diag}(K_{D1}, \ldots, K_{Dn})$. Here Diag stands for a diagonal matrix. If the reference is fixed, the controller becomes:

$$\mathbf{u}(t) = \mathbf{K}_P\,\mathbf{e} - \mathbf{K}_D\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{14.31}$$

which is shown in Figure 14.4. Further, a PID controller with gravity compensation for a changing reference can be defined similarly as

$$\mathbf{u}(t) = \ddot{\mathbf{q}}_d + \mathbf{K}_P\,\mathbf{e} + \mathbf{K}_D\,\dot{\mathbf{e}} + \mathbf{K}_I \int \mathbf{e} + \mathbf{G}(\mathbf{q}) \tag{14.32}$$

where $\mathbf{K}_I = \text{Diag}(K_{I1}, \ldots, K_{In})$. If the reference is fixed, the controller becomes

$$\mathbf{u}(t) = \mathbf{K}_P\,\mathbf{e} - \mathbf{K}_D\,\dot{\mathbf{q}} + \mathbf{K}_I \int \mathbf{e} + \mathbf{G}(\mathbf{q}) \tag{14.33}$$

Example 14.3 (PD controller with gravity compensation): Consider the system used in the previous Example 14.2. Design a PD controller with gravity compensation for position regulation.
Solution: The PD controller with gravitational compensation is given by

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \mathbf{K}_P\ \mathbf{e} - \mathbf{K}_D\ \dot{\mathbf{q}} + \mathbf{G} \tag{14.34}$$

$$= \begin{bmatrix}K_{P1} & 0\\ 0 & K_{P2}\end{bmatrix}\begin{bmatrix}q_{1d}-q_1\\ q_{2d}-q_2\end{bmatrix} - \begin{bmatrix}K_{D1} & 0\\ 0 & K_{D2}\end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix} + \begin{bmatrix}(m_1+m_2)l_1gc_1\\ m_2l_2gc_{12}\end{bmatrix} \tag{14.35}$$

$$= \begin{bmatrix}K_{P1}(q_{1d}-q_1) - K_{D1}\dot{q}_1 + (m_1+m_2)l_1gc_1\\ K_{P2}(q_{2d}-q_2) - K_{D2}\dot{q}_2 + m_2l_2gc_{12}\end{bmatrix} \tag{14.36}$$

14.4 Lyapunov Stability

To analyse the stability of the nonlinear controllers defined in Section 14.3, we can utilise the *Lyapunov stability criterion*, which is a powerful tool in control theory. Consider a nonlinear system that satisfies

$$\dot{x} = f(x) \qquad x(t_0) = x_0 \qquad x \in \mathbb{R}^n \tag{14.37}$$

where $f(x)$ is a *vector field* on $\mathbb{R}^n$, and suppose that $f(0) = 0$. Then, the origin in $\mathbb{R}^n$ is said to be an *equilibrium point* for $\dot{x} = f(x)$. A trivial solution to this nonlinear system is $x(t) = 0$, where the state of the system stays at the equilibrium point forever, which is also called the *null solution*.

14.4.1 Basic Definition

The stability of an equilibrium point of a dynamical system can be defined using the $\epsilon - \delta$ definition of limits.

Definition 14.1. *The null solution $x(t) = 0$ is* **stable** *if and only if, for any $\epsilon > 0$, there exists $\delta(\epsilon) > 0$ such that $\|x(t_0)\| < \delta$ implies $\|x(t)\| < \epsilon$ for all $t > t_0$. The solution $x(t) = 0$ is* **unstable** *if it is not stable.*

Definition 14.2. *The null solution $x(t) = 0$ is* **asymptotically stable** *if and only if 1) it is stable, and 2) there exists $\delta > 0$ such that $\|x(t_0)\| < \delta$ implies $\|x(t)\| \to 0$ as $t \to \infty$.*

Example 14.4 (Lyapunov stability analysis): Assume a mass-spring system with $m = 1$ and $k = 1$, as shown in Figure 14.5. Find the equilibrium point of this system and determine stability.
Solution: Its dynamics is $\ddot{y} = -y$. Define the states of the system as $x_1 = y$ and $x_2 = \dot{y}$. The state space equations can be written as

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}\dot{y}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}x_2\\ -x_1\end{bmatrix} \tag{14.38}$$

For an equilibrium point, we must have

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix}x_2\\ -x_1\end{bmatrix} = \mathbf{0} \tag{14.39}$$

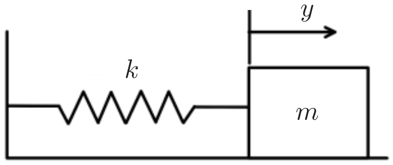


FIGURE 14.5
The mass-spring system.

which yields $x_1 = x_2 = 0$. Hence, the origin is the equilibrium point of the system. The time derivative of the state of this system is a vector field on $\mathcal{R}^2$ as shown in Figure 14.6, which is also called the *phase portrait* of the system.

To determine system stability, first define a circle at the origin with an arbitrarily small radius $\epsilon > 0$. Also, define a larger circle with a radius being $\delta(\epsilon) = 0.9\epsilon$. Within the larger circle, select an initial state $x(t_0)$, which yields a circular orbit with a radius r smaller than $\delta(\epsilon)$ under the governing dynamics. Since $\delta(\epsilon) < \epsilon$, we have $\|x(t)\| = r < \epsilon$ for all $t > t_0$. Therefore, the mass-spring system is stable. However, since $x(t)$ is always on the circular orbit with a constant radius of r, $\|x(t)\|$ is never zero. Therefore, this system is not asymptotically stable.

Example 14.5 (Lyapunov stability analysis — mass-spring-damper system): Consider a mass-spring-damper system with $m = 1$, $k = 1$, and $b = 1$, as shown in Figure 14.7. Use the Lyapunov stability criterion to determine the stability of the system.
Solution: Its dynamics is $\ddot{y} = -y - \dot{y}$. Define the states of the system as $x_1 = y$ and $x_2 = \dot{y}$. The state space equations can be written as

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}\dot{y}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}x_2\\ -x_1 - x_2\end{bmatrix} \tag{14.40}$$

Hence, we have $f(x) = \begin{bmatrix}x_2 & -x_1 - x_2\end{bmatrix}^T$. The time derivative of the state of this system is shown in Figure 14.8, from which we can observe that $x(t)$ is always approaching the origin and will fall

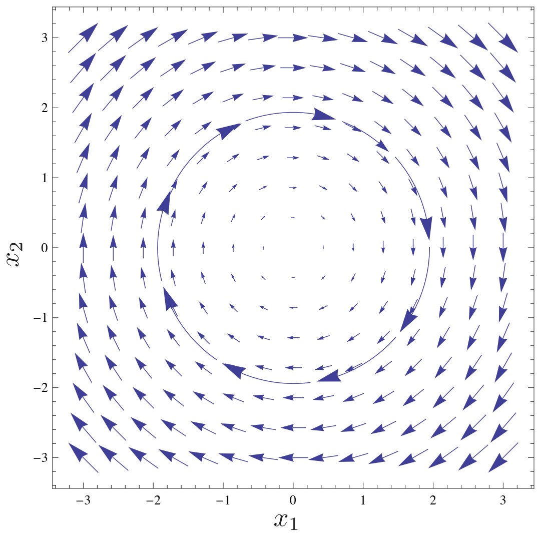


FIGURE 14.6
Phase portrait of the mass-spring system.

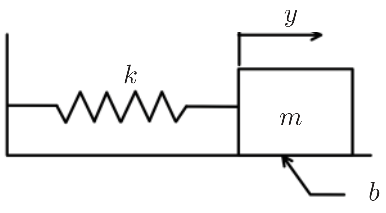


FIGURE 14.7
The mass-spring-damper system.

within a circle at the origin with an arbitrarily small radius, as long as sufficient time is given. Therefore, the mass-spring-damper system is asymptotically stable.

14.4.2 Lyapunov's Direct Method

Lyapunov's direct method, also known as the second method of Lyapunov, allows the determination of stability of a dynamical system without the need for integrating the system's differential equations (14.37). This is a generalised method that makes use of the *Lyapunov function*, $V(x)$, which, in a way, represents the system's potential energy. By studying the change in potential energy, we can ascertain the system's stability.

Definition 14.3. *Let $V(x)$ be a continuous function with continuous first partial derivatives in a neighbourhood of the origin in $\mathcal{R}^n$. Furthermore, suppose that $V(x)$ is positive definite, that is, $V(0) = 0$ and $V(x) > 0$ for $x \neq 0$. Then $V(x)$ is the Lyapunov function for the system given by $\dot{x} = f(x)$.*

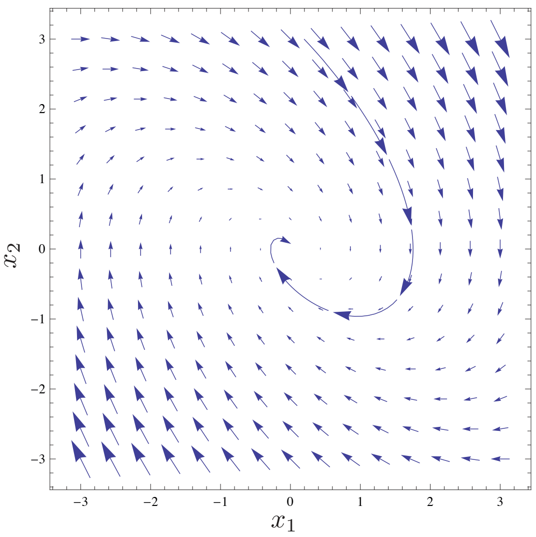


FIGURE 14.8
Phase portrait of the mass-spring-damper system.

Theorem 14.1. *The null solution $x(t) = 0$ of $\dot{x} = f(x)$ is stable, if there exists a Lyapunov function candidate V, such that $\dot{V}$ is negative semi-definite along solution trajectories of $\dot{x} = f(x)$, i.e.,*

$$\dot{V} = \frac{\partial V}{\partial x} f(x) \leq 0 \tag{14.41}$$

Theorem 14.2. *The null solution $x(t) = 0$ of $\dot{x} = f(x)$ is asymptotically stable, if there exists a Lyapunov function candidate V such that $\dot{V}$ is strictly negative along solution trajectories of $\dot{x} = f(x)$, i.e.,*

$$\dot{V} = \frac{\partial V}{\partial x} f(x) < 0 \tag{14.42}$$

Example 14.6 (Lyapunov's direct method — mass-spring system): Find the Lyapunov function for the mass-spring system in Example 14.4. Determine the system's stability using Lyapunov's direct method.
Solution: The Lyapunov function is $V = x_1^2 + x_2^2$, which is positive definite. The time derivative of the Lyapunov function is derived as

$$\begin{aligned} \dot{V} &= \frac{\partial V}{\partial x_1}\dot{x}_1 + \frac{\partial V}{\partial x_2}\dot{x}_2 = 2x_1\dot{x}_1 + 2x_2\dot{x}_2 \\ &= 2\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = 2\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} x_2 \\ -x_1 \end{bmatrix} = 0 \end{aligned} \tag{14.43}$$

According to Theorem 14.1, this system is stable, which agrees with the conclusion in Example 14.4. On the other hand, according to Theorem 14.2, this system is not asymptotically stable, which also agrees with the conclusion in Example 14.4.

Example 14.7 (Lyapunov's direct method — mass-spring-damper system): Find the Lyapunov function for the mass-spring-damper system in Example 14.5. Determine the system's stability using Lyapunov's direct method.
Solution: The Lyapunov function is $V = x_1^2 + x_2^2$ as well. The time derivative of this Lyapunov function is derived as

$$\begin{aligned} \dot{V} &= \frac{\partial V}{\partial x_1}\dot{x}_1 + \frac{\partial V}{\partial x_2}\dot{x}_2 = 2x_1\dot{x}_1 + 2x_2\dot{x}_2 \\ &= 2\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = 2\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} x_2 \\ -x_1 - x_2 \end{bmatrix} = -x_2^2 < 0 \end{aligned} \tag{14.44}$$

for any $x_2 \neq 0$. According to Theorem 14.2, this system is asymptotically stable, which agrees with the conclusion in Example 14.5.

14.5 Dynamic Relation

In order to utilise the Lyapunov criterion to analyse the nonlinear controllers with only gravity compensation, we need to understand a spherical dynamic relation in the robotic dynamics given by (14.14). From the point of view of the energy balance, the input power from the environment into the robot must equal the sum of the output power from the robot to the environment and the change rate of the total energy of the robot. The input power due to motors is given by

$$\mathbf{P}_i = \dot{\mathbf{q}}^T \ \mathbf{u} \tag{14.45}$$

The output power is the dissipated power due to friction, given by

$$\mathbf{P}_o = \dot{\mathbf{q}}^T \ \mathbf{B} \ \dot{\mathbf{q}} \tag{14.46}$$

where $\mathbf{B} \ \dot{\mathbf{q}}$ is the friction projected into the joint space. Since the total energy is the sum of the potential energy and the kinetic energy, the rate of change of the total energy is discussed accordingly. The rate of change of the potential energy equals the product of the velocity and the force overcoming the gravity, which is given by, in joint space,

$$\dot{\mathbf{E}}_p = \dot{\mathbf{q}}^T \ \mathbf{G}(\mathbf{q}) \tag{14.47}$$

The change rate of the kinetic energy is obtained by the time derivative of the kinetic energy as

$$\frac{d}{dt}(\frac{1}{2} \dot{\mathbf{q}}^T \ \mathbf{M}(\mathbf{q}) \ \dot{\mathbf{q}}) = \dot{\mathbf{q}}^T \ \mathbf{M}(\mathbf{q}) \ \ddot{\mathbf{q}} + \frac{1}{2} \dot{\mathbf{q}}^T \dot{\mathbf{M}}(\mathbf{q}) \ \dot{\mathbf{q}} \tag{14.48}$$

Hence, the energy balance yields

$$\dot{\mathbf{q}}^T \ \mathbf{u} - \dot{\mathbf{q}}^T \ \mathbf{B} \ \dot{\mathbf{q}} = \dot{\mathbf{q}}^T \ \mathbf{M}(\mathbf{q}) \ \ddot{\mathbf{q}} + \frac{1}{2} \dot{\mathbf{q}}^T \dot{\mathbf{M}}(\mathbf{q}) \ \dot{\mathbf{q}} + \dot{\mathbf{q}}^T \ \mathbf{G}(\mathbf{q}) \tag{14.49}$$

Substituting (14.14) into (14.49) gives

$$\dot{\mathbf{q}}^T \ \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \ \dot{\mathbf{q}} = \frac{1}{2} \dot{\mathbf{q}}^T \dot{\mathbf{M}}(\mathbf{q}) \ \dot{\mathbf{q}} \tag{14.50}$$

Example 14.8 (Lyapunov criterion — dynamic relation): Consider the RP robotic manipulator as shown in Figure 14.9. The dynamics is given by

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q}) \ \ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) \ \dot{\mathbf{q}} + \mathbf{B} \ \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{14.51}$$

where

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} m_1 l_1^2 + I_{y1} + I_{y2} + m_2 d_2^2 & 0 \\ 0 & M_2 \end{bmatrix} \tag{14.52}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \ \dot{\mathbf{q}} = \begin{bmatrix} 2 m_2 d_2 \dot{q}_1 \dot{d}_2 \\ -m_2 d_2 \dot{q_1}^2 \end{bmatrix} \tag{14.53}$$

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} (m_1 l_1 + m_2 d_2) g \sin q_1 \\ -m_2 g \cos q_1 \end{bmatrix} \tag{14.54}$$

Verify that

$$\dot{\mathbf{q}}^T \ \mathbf{C} \ \dot{\mathbf{q}} = \frac{1}{2} \dot{\mathbf{q}}^T \dot{\mathbf{M}} \ \dot{\mathbf{q}} \tag{14.55}$$

Solution:

$$\dot{\mathbf{q}}^T \ \mathbf{C} \ \dot{\mathbf{q}} = \begin{bmatrix} \dot{q}_1 & \dot{d}_2 \end{bmatrix} \begin{bmatrix} 2 m_2 d_2 \dot{q}_1 \dot{d}_2 \\ -m_2 d_2 \dot{q_1}^2 \end{bmatrix} \tag{14.56}$$

$$= 2 m_2 d_2 \dot{q_1}^2 \dot{d}_2 - m_2 d_2 \dot{q_1}^2 \dot{d}_2 \tag{14.57}$$

$$= m_2 d_2 \dot{q_1}^2 \dot{d}_2 \tag{14.58}$$

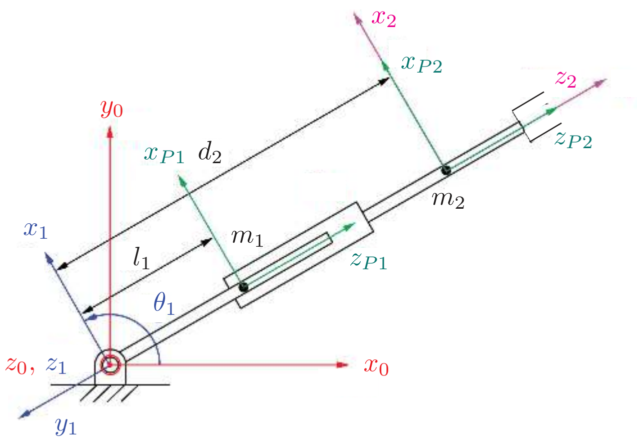


FIGURE 14.9
RP robotic manipulator.

$$\dot{\mathbf{M}} = \begin{bmatrix} 2m_2\dot{d}_2 d_2 & 0 \\ 0 & 0 \end{bmatrix} \tag{14.59}$$

$$\frac{1}{2}\,\dot{\mathbf{q}}^T \dot{\mathbf{M}}\,\dot{\mathbf{q}} = \frac{1}{2}\begin{bmatrix}\dot{q}_1 & \dot{d}_2\end{bmatrix}\begin{bmatrix} 2m_2\dot{d}_2 d_2 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix}\dot{q}_1 \\ \dot{d}_2\end{bmatrix} = m_2 d_2 \dot{q}_1{}^2 \dot{d}_2 \tag{14.60}$$

Thus, the equation is verified.

14.6 Stability Analysis of Nonlinear Robotic Controllers

Consider the PD controller with gravity compensation given by (14.31). Under this control scheme, the dynamics of this closed-loop system is obtained by substituting (14.31) into (14.14),

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} - \mathbf{K}_P(\dot{\mathbf{q}}_d - \mathbf{q}) + \mathbf{K}_D\,\dot{\mathbf{q}} = 0 \tag{14.61}$$

We can use the Lyapunov theorem to determine if this control scheme is stable. Firstly, define the Lyapunov function candidate V[1] as

$$V = \frac{1}{2}\,\dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}} + \frac{1}{2}\,\mathbf{e}^T\,\mathbf{K}_D\,\mathbf{e} \tag{14.62}$$

where $\mathbf{e} = \dot{\mathbf{q}}_d - \mathbf{q}$. The time derivative of V is

$$\begin{aligned} \dot{V} &= \dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \frac{1}{2}\,\dot{\mathbf{q}}^T\,\dot{\mathbf{M}}(\mathbf{q})\,\dot{\mathbf{q}} + \dot{\mathbf{e}}^T\,\mathbf{K}_P\,\mathbf{e} \\ &= \dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \frac{1}{2}\,\dot{\mathbf{q}}^T\,\dot{\mathbf{M}}(\mathbf{q})\,\dot{\mathbf{q}} - \dot{\mathbf{q}}^T\,\mathbf{K}_P\,\mathbf{e} \end{aligned} \tag{14.63}$$

[1]Finding a proper Lyapunov function candidate is a nontrivial task itself.

Substituting (14.14) into (14.63) yields

$$\begin{aligned}\dot{V} &= -\,\dot{\mathbf{q}}^T(\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} - \mathbf{K}_P\,\mathbf{e} + \mathbf{K}_D\,\dot{\mathbf{q}}) + \frac{1}{2}\,\dot{\mathbf{q}}^T\,\dot{\mathbf{M}}(\mathbf{q})\,\dot{\mathbf{q}} - \dot{\mathbf{q}}^T\,\mathbf{K}_P\,\mathbf{e} \\ &= -\,\dot{\mathbf{q}}^T\,\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} - \dot{\mathbf{q}}^T\,\mathbf{B}\,\dot{\mathbf{q}} - \dot{\mathbf{q}}^T\,\mathbf{K}_D\,\dot{\mathbf{q}} + \frac{1}{2}\,\dot{\mathbf{q}}^T\,\dot{\mathbf{M}}(\mathbf{q})\,\dot{\mathbf{q}} \qquad (14.64)\end{aligned}$$

In robotic dynamics, there is a relation between the Centrifugal/Coriolis matrix and the inertia matrix given by

$$\dot{\mathbf{q}}^T\,\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} = \frac{1}{2}\,\dot{\mathbf{q}}^T\,\dot{\mathbf{M}}(\mathbf{q})\,\dot{\mathbf{q}} \qquad (14.65)$$

Substituting (14.65) into (14.64) yields

$$\dot{V} = -\,\dot{\mathbf{q}}^T\,\mathbf{B}\,\dot{\mathbf{q}} - \dot{\mathbf{q}}^T K_D\,\dot{\mathbf{q}} \qquad (14.66)$$

Both $\mathbf{B}$ and $\mathbf{K}_D$ are positive-definite, therefore $\dot{V}$ is negative for any nonzero $\mathbf{q}$ and $\dot{\mathbf{q}}$. According to the theorem of Lyapunov stability, this closed-loop system is asymptotically stable. Therefore, PD control with gravity compensation is widely used in industrial robots.

Example 14.9 (Stability — PD control with gravity compensation): Consider a position-regulation system that (without loss of generality) attempts to maintain $q_d = 0$. Prove that the control law

$$\boldsymbol{\tau} = -\,\mathbf{K}_P\,\mathbf{q} - \mathbf{M}(\mathbf{q})\,\mathbf{K}_V\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \qquad (14.67)$$

yields an asymptotically stable nonlinear system. You may take K_V to be of the form $K_V = k_V I_n$ where k_V is a scalar, and I_n is the $n \times n$ identity matrix.
Solution: The closed-loop system is given by

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{V}_M(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = -\,\mathbf{K}_P\,\mathbf{q} - \mathbf{M}(\mathbf{q})K_V\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \qquad (14.68)$$

The Lyapunov equation and its time derivative are

$$\begin{aligned}V &= \frac{1}{2}\,\dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}} + \frac{1}{2}\,\mathbf{q}^T\,\mathbf{K}_P\,\mathbf{q} \qquad (14.69)\\ \dot{V} &= \frac{1}{2}\,\dot{\mathbf{q}}^T\dot{\mathbf{M}}(\mathbf{q})\,\dot{\mathbf{q}} + \dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \dot{\mathbf{q}}^T\,\mathbf{K}_P\,\mathbf{q} \\ &= \frac{1}{2}\,\dot{\mathbf{q}}^T\dot{\mathbf{M}}(\mathbf{q})\,\dot{\mathbf{q}} + \dot{\mathbf{q}}^T\big(-\,\mathbf{V}_M(\mathbf{q},\dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q}) - \mathbf{K}_P\,\mathbf{q} - \mathbf{M}(\mathbf{q})\,\mathbf{K}_V\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q})\big) \\ &\qquad + \dot{\mathbf{q}}^T\,\mathbf{K}_P\,\mathbf{q} \\ &= -\,\dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\mathbf{K}_V\,\dot{\mathbf{q}} \qquad (14.70)\end{aligned}$$

Because $\dot{V}$ is always negative for any nonzero $\dot{q}$, the system is asymptotically stable, according to Lyapunov's stability theorem.

14.7 Motion Control in Task Space

The discussion of motion control thus far has focused on joint space variables. This is convenient when considering joint-related constraints, such as range-of-motion or velocity limits. A trajectory described in the joint space also avoids issues like singularities and redundancy. However, there are situations where the end-effector is required to interact with the environment. In this case, it is more convenient to express manipulator motion in the environment frame, or *task space*.

14.7.1 Dynamics in Task Space

To regulate torque and force interaction of the end-effector with the environment, the dynamics of the robotic system should be intuitively expressed in the task space. The interactive torque and force between the robot and its environment, $\mathbf{F}_e$, can be written as

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}^T(\mathbf{q})\,\mathbf{F}_e \tag{14.71}$$

where $\mathbf{J}$ is the Jacobian matrix mapping the joint velocities to the velocity of the end-effector at the contact, i.e.,

$$\dot{\mathbf{x}} = \mathbf{J}\,\dot{\mathbf{q}} \tag{14.72}$$

The time derivative of (14.72) gives

$$\ddot{\mathbf{x}} = \mathbf{J}\,\ddot{\mathbf{q}} + \dot{\mathbf{J}}\,\dot{\mathbf{q}} \tag{14.73}$$

Hence, we have

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}\,\dot{\mathbf{x}} \tag{14.74}$$

and

$$\ddot{\mathbf{q}} = \mathbf{J}^{-1}\,\ddot{\mathbf{x}} - \mathbf{J}^{-1}\dot{\mathbf{J}}\,\mathbf{J}^{-1}\,\dot{\mathbf{x}} \tag{14.75}$$

Further, the joint motor torques and their projection onto the end-effector at the contact have the relation

$$\mathbf{u} = \mathbf{J}^T\,\mathbf{F}_t \tag{14.76}$$

Substituting (14.74), (14.75), and (14.76) into the dynamics in joint space (14.71) yields

$$\begin{aligned}\mathbf{J}^T\,\mathbf{F}_t = \mathbf{M}(\mathbf{q})(\mathbf{J}^{-1}\,\ddot{\mathbf{x}} - \mathbf{J}^{-1}\dot{\mathbf{J}}\,\mathbf{J}^{-1}\,\dot{\mathbf{x}}) + [\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{B}]\,\mathbf{J}^{-1}\,\dot{\mathbf{x}} +& \\ \mathbf{G}(\mathbf{q}) + \mathbf{J}^T\,\mathbf{F}_e&\end{aligned} \tag{14.77}$$

which is equivalent to

$$\begin{aligned}\mathbf{F}_t = (\mathbf{J}^T)^{-1}\,\mathbf{M}(\mathbf{q})(\mathbf{J}^{-1}\,\ddot{\mathbf{x}} - \mathbf{J}^{-1}\dot{\mathbf{J}}\,\mathbf{J}^{-1}\,\dot{\mathbf{x}}) + (\mathbf{J}^T)^{-1}(\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{B})\,\mathbf{J}^{-1}\,\dot{\mathbf{x}} +& \\ (\mathbf{J}^T)^{-1}\,\mathbf{G}(\mathbf{q}) + \mathbf{F}_e&\end{aligned} \tag{14.78}$$

and can further be written as

$$\mathbf{F}_t = \mathbf{M}_t\,\ddot{\mathbf{x}} + \mathbf{C}_t\,\dot{\mathbf{x}} + \mathbf{B}_t\,\dot{\mathbf{x}} + \mathbf{G}_t + \mathbf{F}_e \tag{14.79}$$

where $\mathbf{M}_t, \mathbf{C}_t, \mathbf{B}_t, \mathbf{G}_t$ are the equivalent mass matrix, centrifugal and Coriolis matrix, damping matrix, and gravity term, respectively. These quantities are defined as

$$\mathbf{F}_t = (\mathbf{J}^T)^{-1}\,\mathbf{u} \tag{14.80}$$

$$\mathbf{M}_t = (\mathbf{J}^T)^{-1}\,\mathbf{M}(\mathbf{q})\,\mathbf{J}^{-1} \tag{14.81}$$

$$\mathbf{C}_t = (\mathbf{J}^T)^{-1}\,\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\mathbf{J}^{-1} - \mathbf{M}_t\dot{\mathbf{J}}\,\mathbf{J}^{-1} \tag{14.82}$$

$$\mathbf{B}_t = (\mathbf{J}^T)^{-1}\,\mathbf{B}\,\mathbf{J}^{-1} \tag{14.83}$$

$$\mathbf{G}_t = (\mathbf{J}^T)^{-1}\,\mathbf{G}(\mathbf{q}) \tag{14.84}$$

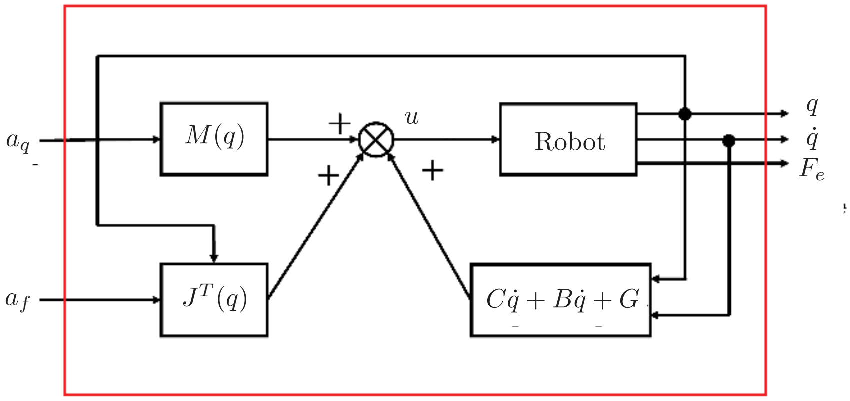


FIGURE 14.10
Block diagram for first inner law of system.

14.7.2 Task Space Dynamics Partitioning

The dynamics of the robot in task space allows us to observe the behaviour of a robot from the point of view of the end-effector. It can be further used to partition the nonlinear terms in task space, which is called dynamics partitioning in task space. One can use (14.79) to directly compensate for those nonlinear terms. However, it appears that the computational cost is too high. Since only the result dynamics in the task space is of our interests, the compensation can be first done in the joint space, i.e., define a control law given by

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\,\mathbf{a}_q + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}^T(\mathbf{q})\,\mathbf{F}_e \tag{14.85}$$

which is illustrated in Figure 14.10. Substituting this law into the dynamics given by (14.71) gives the closed-loop dynamics as

$$\mathbf{a}_q = \ddot{\mathbf{q}} \tag{14.86}$$

which can be represented in task space by using (14.75) as

$$\mathbf{a}_q = \mathbf{J}^{-1}\,\ddot{\mathbf{x}} - \mathbf{J}^{-1}\,\dot{\mathbf{J}}\,\dot{\mathbf{q}} \tag{14.87}$$

In order to linearise the above dynamics, the second control is defined as

$$\mathbf{a}_q = \mathbf{J}^{-1}\,\mathbf{a}_x - \mathbf{J}^{-1}\,\dot{\mathbf{J}}\,\dot{\mathbf{q}} \tag{14.88}$$

which yields the resulting closed-loop dynamics given by

$$\mathbf{a}_x = \ddot{\mathbf{x}} \tag{14.89}$$

The first and second partitioning laws are illustrated in Figure 14.11. The robotic manipulator is partitioned or decoupled into six independent linear systems along the six task-space axes, respectively.

14.7.3 Error Dynamics

To derive the error dynamics of motion control in the task space, again, we begin with the equation

$$\ddot{e}_x + 2\zeta\omega\dot{e}_x + \omega^2 e_x = 0 \tag{14.90}$$

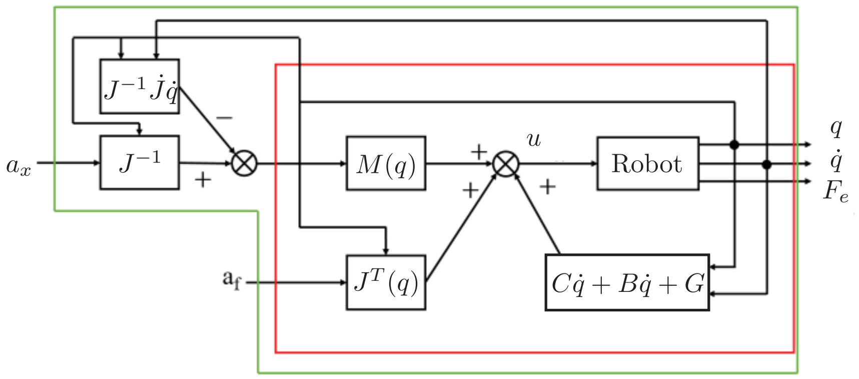


FIGURE 14.11
Block diagram for second inner law of system.

which represents the damped second-order error response. Furthermore, the error based on the closed-loop task space dynamics (14.89) is

$$\ddot{e} = \ddot{x}_d - \ddot{x} \tag{14.91}$$

Substituting into the error dynamics equation yields

$$\ddot{x} = \ddot{x}_d + 2\zeta\omega\dot{e}_x + \omega^2 e_x \tag{14.92}$$

We desire a critically-damped error response by setting $\zeta = 0$. Therefore, the control law that yields the error dynamics is

$$a = \ddot{x}_d + 2\omega\dot{e}_x + \omega^2 e_x \tag{14.93}$$

14.8 Conclusion

The computed torque control was introduced to fully linearise the robotic dynamics and achieve the desired stability and performance. By using the control laws, we defined an inner and outer control loop. The inner control loop encapsulates the robot dynamics, which are linearised under the assumption that the model is perfect. The outer control law then controls the inner loop, such that it drives a unit mass under a linear controller. The linear properties of the outer loop allow us to control the gains to achieve a critically damped system. In practice, relatively simple controllers with gravity compensation are typically used. Lyapunov's stability criterion was used to investigate both the stability and asymptotic stability of these nonlinear controllers.

We introduced task space motion planning, which provides an intuitive control strategy where the end-effector motion is required under environmental constraints. This has applications where the end-effector is required to interact with its environment. The dynamics of the robot were derived and expressed in the task space and partitioned into an inner control law and an outer control law. Finally, the error dynamics of motion control in the task space were derived.

14.9 Exercises

Problem 1. Derive the computer-torque-control (CTC) scheme and write the control law of a RP robot, whose dynamics is given by (assume point mass):

$$\tau = \begin{bmatrix} (l_1^2 m_1 + m_2 d_2^2)\ddot{\theta}_1 + 2m_2 d_2 \dot{\theta}_1 \dot{d}_2 + g(l_1 m_1 + m_2 d_2)\sin\theta_1 \\ m_2 \ddot{d}_2 - m_2 g \cos\theta_1 - m_2 d_2 \dot{\theta}_1^2 \end{bmatrix}$$

Specify the feedforward and feedback components of the CTC law and explain their roles.

Problem 2. Consider a linear control law (PD controller) based on the linearisation of the system about an equilibrium point is directly used for a robotic manipulator.

$$\tau = -K_v \dot{e} - K_p e$$

where K_v and K_p are positive definite matrices and $e = q - \dot{q}_d$.

Prove that if $\dot{q}_d$, the control law applied to the system renders the equilibrium point $\dot{q}_d = 0$ globally asymptotically stable.
The Lyapunov function is defined as:

$$V(q, \dot{q}) = \frac{1}{2}\dot{q}^T M \dot{q} + \frac{1}{2} q^T K_p q + P$$

where P is the potential energy.

Problem 3. Suppose a PID computed torque controller law is applied to eliminate nonzero steady-state error, and the error is defined as:

$$e = q_d - q$$

The error dynamics is given by

$$\ddot{e} + K_v \dot{e} + K_p e + K_i \varepsilon = 0$$

$$\dot{\varepsilon} = e$$

Draw the block diagram of the proposed PID-CTC law and derive the system dynamics equation with PID-CTC law, using the parameters below:

$$m_1 = m_2 = 1 \text{ kg}, \; l_1 = l_2 = 1 \text{ m}$$

The dynamics of the system Figure 14.12 is given by

$$\begin{aligned} \tau_1 =& \frac{1}{4}\left(2gc_1 \left(l_1 m_1 + 2l_1 m_2 + l_2 m_2 c_2\right) - 2l_2 m_2 \left(2l_1 + l_2 c_2\right) s_2 \dot{q}_1 \dot{q}_2 \right. \\ & \left. + \left(l_1^2 \left(m_1 + 4m_2\right) + l_2 m_2 c_2 \left(4l_1 + l_2 c_2\right)\right) \ddot{q}_1\right) \\ \tau_2 =& \frac{1}{4} l_2 m_2 \left(s_2 \left(-2gs_1 + \left(2l_1 + l_2 c_2\right)\dot{q}_1^2\right) + l_2 \ddot{q}_2\right) \end{aligned}$$

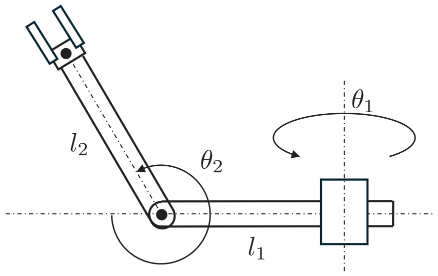


FIGURE 14.12
An RR manipulator.

Problem 4. Since we have proved that the PD control law is asymptotically stable at the equilibrium point $q = 0$, when $\dot{q}_d = 0$. Now, let us consider a modified version of the PD control law:

$$\boldsymbol{\tau} = M\ddot{q}_d + C\dot{q}_d + G - K_v\dot{e} - K_p e$$

We call it augmented PD control law. Prove that the control law applied to the system is asymptotically stable if, $K_v > 0,\ K_p > 0$, and explain how you would choose ϵ. The Lyapunov function candidate is chosen as:

$$V = \frac{1}{2}\dot{e}^T M\dot{e} + \frac{1}{2}e^T K_p e + \epsilon\dot{e}^T M\dot{e}$$

Problem 5. The dynamics of a robot in joint space is given by

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}^T(\mathbf{q})\,\mathbf{F}_e$$

where $\mathbf{J}$ is the Jacobian matrix. $\mathbf{u}$ and $\mathbf{F}_e$ are joint torques and force on the end-effector, respectively. Derive the dynamics of this robot in task space.

Problem 6. Design a PID controller with gravity compensation.

1. Define the fixed-reference equation $u(t)$.
2. Illustrate the complete block diagram of your control system assuming a fixed reference.
3. Indicate the feedforward and feedback components of the Computer Torque Control (CTC) law and explain their roles.

15

Force Control

Force control is a control strategy in which the robot applies a wrench (forces and torques) to the environment *without* controlling motion. This has some industrial applications such as grinding, deburring, or stamping where the motion of the end-effector is constrained in the direction of the applied *wrench*. Wrench is defined as

$$\mathcal{F} = \begin{bmatrix} f_x & f_y & f_z & \tau_x & \tau_y & \tau_z \end{bmatrix}^T \tag{15.1}$$

where each element of f is the force applied on each axis, and τ is each axis of torque about each axis.

Consider the manipulator dynamics equation

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{B}\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}(\mathbf{q})^T \mathcal{F} \tag{15.2}$$

where $\mathcal{F}$ is the wrench applied by the end-effector to the environment and $\mathbf{J}$ is the Jacobian defined in the same frame as $\mathcal{F}$. Since the motion of the end-effector is highly constrained in the direction of the wrench, joint velocity and acceleration can be ignored to get

$$\boldsymbol{\tau} = \mathbf{G}(\mathbf{q}) + \mathbf{J}(\mathbf{q})^T \mathcal{F} \tag{15.3}$$

In the absence of direct measurement of the wrench at the end-effector, joint-angle feedback is sufficient to implement the force control law

$$\boldsymbol{\tau} = \mathbf{G}'(\mathbf{q}) + \mathbf{J}(\mathbf{q})^T \mathcal{F}_d \tag{15.4}$$

where $\mathbf{G}'(\mathbf{q})$ is a model of joint torque caused by gravity, and $\mathcal{F}_d$ is the desired wrench. However, control strategy requires a highly accurate gravity compensation model and precise torque control at each joint. Alternatively, a six-axis force-torque sensor can be installed between the final link of the manipulator and the end-effector to measure the wrench. These are readily available and easy to install with very high precision, but can be very expensive.

15.1 Single Axis Control in Task Space

The simplest application of force control is the manipulator applying a force to the environment along a single axis in the task space. To develop a control law to implement this, we can represent the scenario as a generalised mass-spring system as shown in Figure 15.1, where the mass is according to the partitioned linear dynamics given by (14.86) while the spring is used to model the stiffness of the environment in contact.

The interaction dynamics of the system is given by

$$f + f_g = M\ddot{z} + K_e z + f_\delta \tag{15.5}$$

DOI: 10.1201/9781003614319-15

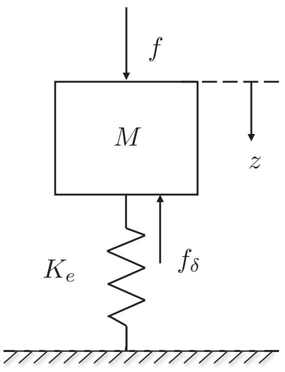


FIGURE 15.1
Mass-spring system.

where f, f_g, f_δ, and K_e are the driving force, the gravity of the object, the disturbance force, and the stiffness of the environment, respectively. The actual contact force is given by

$$f_e = K_e z \tag{15.6}$$

since the contact force f_e is to be controlled, we can rewrite equation (15.5) as

$$f + f_g = \tfrac{M}{K_e}\ddot{f}_e + f_e + f_\delta \tag{15.7}$$

The *control law I* can be proposed as

$$f + f_g = \tfrac{M}{K_e}f' + f_e \tag{15.8}$$

where f' is considered a virtual control input. Thus, the control objective now becomes designing a virtual control input f' to make sure the *system dynamics I* described by equation (15.7) is satisfied. Combining equation (15.7) with (15.8) yields the *system dynamics II*

$$\ddot{f}_e - f' + \tfrac{K_e}{M}f_\delta = 0 \tag{15.9}$$

The error dynamics of the contact force is given as

$$\ddot{e}_f + K_D\dot{e}_f + K_P e_f = \tfrac{K_e}{M}f_\delta \tag{15.10}$$

where $e_f = f_d - f_e$, K_D, and K_P are the derivative gain and proportional gain, respectively. Then, substituting the error dynamics into the *system dynamics II* yields the *control law II*

$$f' = \ddot{f}_d + K_D\dot{e}_f + K_P e_f \tag{15.11}$$

We can derive the final control input by combining the *control law I* and the *control law II*, described by equations (15.8) and (15.11). The final control input is given as

$$f = \tfrac{M}{K_e}(\ddot{f}_d + K_D\dot{e}_f + K_P e_f) + f_e - f_g \tag{15.12}$$

In practice, the desired contact force is typically required to be a constant (i.e., $\ddot{f}_d = 0$); therefore, the force controller can be simplified as

$$f = \tfrac{M}{K_e}(K_D\dot{e}_f + K_P e_f) + f_e - f_g \tag{15.13}$$

15.2 Hybrid Motion-Force Control

As typical with most industrial applications for robots, most tasks that require force control also require some control of motion. For instance, we previously mentioned grinding and deburring as common applications for force control. However, this task is quite limited if the tool remains stationary, and often, it is easier to manipulate the tool rather than to manipulate the entire workpiece. Hybrid motion-force control can be used to enable control of both force and motion. To fully illustrate the point, we need to define two types of constraints, natural and artificial, in two different types of spaces, motion-type and force-type.

15.2.1 Natural Constraints

There are six DoF of any rigid body in the 3D space, in which three DoF represent axes of translations, v_x, v_y, and v_z, and three DoF representing axes rotations, ω_x, ω_y, and ω_z. When concatenated, the resultant vector is

$$\mathcal{V} = \begin{bmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T \tag{15.14}$$

where $\mathcal{V}$ is known as *twist.*

In the motion-type space, contacts are specified by means of degeneration of mobility. If the motion in one direction is constrained, the corresponding item is assigned with zero, which is called the *natural constraint.* In force-type space, contacts are specified by the presence of a wrench as defined in (15.1). If no force is present in one direction, it implies no contact and the corresponding item is assigned with zero. We call this axis *naturally constrained* in the force-type space.

It is interesting to observe that any axis of an object must be constrained in either motion-type space or force-type space. The mathematical expression of this relation is given by

$$v_x f_x + v_y f_y + v_z f_z + \omega_x \tau_x + \omega_y \tau_y + \omega_z \tau_z = 0 \tag{15.15}$$

which is called the reciprocal relationship between the naturally constrained motion and the naturally constrained force. The form of the reciprocal relation is called the *Klein form*, and plays an important role in the *screw theory.*

Example 15.1 (Natural constraints — static workpiece): Find the natural constraints in both motion-type and force-type spaces of the contact for the scenario in Figure 15.2.

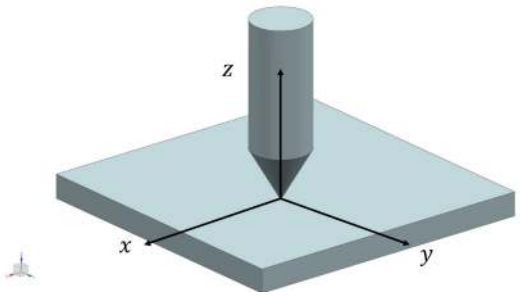


FIGURE 15.2
Tool interacting with a static workpiece.

Solution: The resulting natural constraints table is

M-Space	F-Space
v_x: U	f_x: 0
v_y: U	f_y: 0
v_z: 0	f_z: U
ω_x: U	τ_x: 0
ω_y: U	τ_y: 0
ω_z: U	τ_z: 0

$v_z = 0$ is the natural motion-type constraint preventing the tool from penetrating the workpiece; with rigid force assumption, the tool is also subjected to a non-zero interacting force, i.e., f_z becomes unconstrained.

15.2.2 Artificial Constraints

The unconstrained items in the motion space and the force space can be controlled since they are free. The artificial constraints are introduced to be complementary to the natural constraints in both spaces. The artificial constraints in the motion-type space are v_x^a, v_y^a, v_z^a, ω_x^a, ω_y^a, and ω_z^a, while the artificial constraints in the force-type space are f_x^a, f_y^a, f_z^a, τ_x^a, τ_y^a, and τ_z^a.

In the same type of space, any axis of an object in contact must be constrained either naturally or artificially. Neither under-constrained contact nor over-constrained contact is allowed in this type of contact.

Example 15.2 (Artificial constraints — static workpiece): Find the artificial constraints in both motion-type and force-type spaces of the contact for the scenario in Figure 15.2.
Solution:

M-Space	F-Space
v_x^a: v_x	f_x^a: U
v_y^a: v_y	f_y^a: U
v_z^a: U	f_z^a: f_z
ω_x^a: 0	τ_x^a: U
ω_y^a: 0	τ_y^a: U
ω_z^a: 0	τ_z^a: U

The presence of non-zero v_x^a and v_y^a indicates that the speed at which the tool sweeps over the workpiece surface is artificially controlled as per the task requirement. Meanwhile, the rotation of the tool is limited. In the force domain, the contact force f_z is artificially controlled to prevent damage to the tool and the workpiece.

Example 15.3 (Constraints — Lock and Key): Describe the steps required to open a lock mechanism using a key as shown in Figure 15.3 and show the artificial and natural constraints for each step.

TABLE 15.1
Insert the Key into the Lock

N-Constraints M-Space	N-Constraints F-Space	A-Constraints M-Space	A-Constraints F-Space
v_x: 0	f_x: U	v_x^a: U	f_x^a: 0
v_y: 0	f_y: U	v_y^a: U	f_y^a: 0
v_z: U	f_z: 0	v_z^a: v_{in}	f_z^a: U
ω_x: U	τ_x: 0	ω_x^a: 0	τ_x^a: U
ω_y: 0	τ_y: U	ω_y^a: U	τ_y^a: 0
ω_z: 0	τ_z: U	ω_z^a: U	τ_z^a: 0

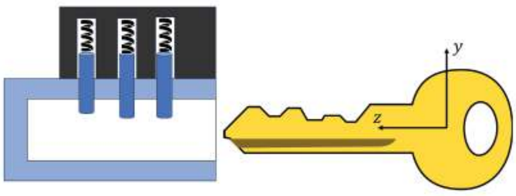


FIGURE 15.3
A key and lock mechanism.

Solution:

Step 1: Insert the key into the lock. In this process, the fit between the key and the keyhole prevents the key from linear movement along x and y axes, as well as rotation around y and z axes, hence the 0 natural motion-type constraints and unconstrained natural force-type constraints. On the other hand, the operator holding the key controls the linear velocity of the insertion v_{in} along the z axis and prevents key rotation around the x axis. Correspondingly, unconstrained force and moment on the corresponding axes are supplied to realise the motion. The constraints are summarised below in Table 15.1/15.2.

Step 2: Turn the key. With the key fully inserted, the keyhole prevents the key from movement along the x and the y axes, while allowing the key to be either extracted or turned, leading to 0 force-type natural constraints. On the operator side, turning the key requires the linear motion along the z axis to be zero to not have it extracted, and the rotation to be a non-zero ω_{turn}, as well as unconstrained force and moment supplied to achieve the desired motion, hence the artificial constraints presented. The constraints are summarised below in Table 15.2.

TABLE 15.2
Turn the Key

N-Constraints M-Space	N-Constraints F-Space	A-Constraints M-Space	A-Constraints F-Space
v_x: 0	f_x: U	v_x^a: U	f_x^a: 0
v_y: 0	f_y: U	v_y^a: U	f_y^a: 0
v_z: U	f_z: 0	v_z^a: 0	f_z^a: U
ω_x: 0	τ_x: U	ω_x^a: U	τ_x^a: 0
ω_y: 0	τ_y: U	ω_y^a: U	τ_y^a: 0
ω_z: U	τ_z: 0	ω_z^a: ω_{turn}	τ_z^a: U

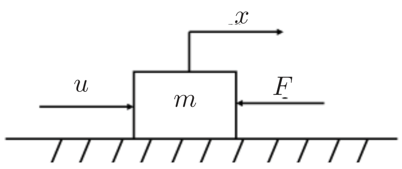


FIGURE 15.4
Single mass system.

15.3 Impedance Control

In ideal hybrid-motion control, we assume the environment to be relatively rigid. This demands two extremes of robot impedance. For motion control, we wish to achieve end-effector motion while rejecting external force disturbance, characterised by high robot impedance. On the other hand, force control demands low robot impedance, where we want to minimise the change in force due to motion disturbance. Realistically, there are limits to the levels of impedance that a robot can achieve.

In this section, we consider the problem of impedance control, where the robot end-effector is required to manipulate an object or its environment that has similar properties to a mass-spring-damper system. This has applications in haptic feedback, especially where there is consistent contact between a robotic end-effector and the environment, such as in a surgical environment.

15.3.1 Single Axis Control

Consider the single mass (M) system shown in Figure 15.4, where u, x, and F are the input force, displacement, and contact force, respectively. The idea here is to use a control law to modify the mass from the point of view of the contact interaction. The dynamics of the mass system is given by

$$u - F = M\ddot{x} \tag{15.16}$$

We can design a control law as $u = -mF$, where m is a positive scalar. The resulting dynamics is

$$-F = \frac{M}{1+m}\ddot{x} \tag{15.17}$$

where $\frac{M}{1+m}$ is called the *apparent inertia* of the system under the force feedback. If $m >> 1$, the inertia of the system is effectively removed from the contact dynamics. We can further change the stiffness and damping by a control law given by

$$u = M\,\ddot{\mathbf{x}}_d + B\dot{e} + Ke - mF \tag{15.18}$$

where $e = x_d - x$. The resulting dynamics is

$$\frac{M}{m+1}\ddot{e} + \frac{B}{m+1}\dot{e} + \frac{K}{m+1}e = -F \tag{15.19}$$

which yields the desired artificial compliance of the robot. If there is no contact force, this control simply performs motion tracking.

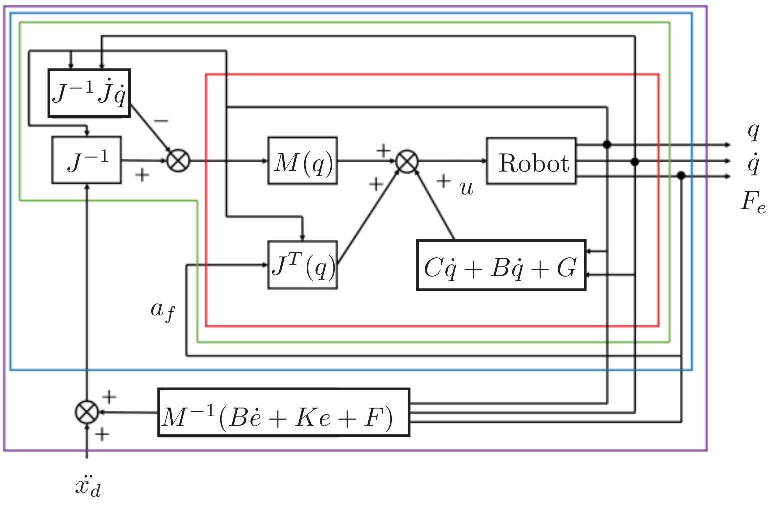


FIGURE 15.5
Impedance control in task space.

15.3.2 Task Space Control

A similar idea can be applied to a robotic system (Figure 15.5). The partitioned dynamics in task space is

$$\mathbf{a}_x = \ddot{\mathbf{x}} \tag{15.20}$$

Consider an outer control law given by

$$\mathbf{a}_x = \ddot{\mathbf{x}}_d + \mathbf{M}_d{}^{-1}(\mathbf{B}_d\, \dot{\mathbf{e}} + \mathbf{K}_d\, \mathbf{e} + \mathbf{F}\,) \tag{15.21}$$

The resulting system is

$$\mathbf{M}\; \ddot{\mathbf{e}} + \mathbf{B}\; \dot{\mathbf{e}} + \mathbf{K}\; \mathbf{e} = -\,\mathbf{F} \tag{15.22}$$

which will track the motion and regulate the contact forces simultaneously.

15.3.3 Control with Environmental Dynamics

When a robotic manipulator is interacting with the environment, as illustrated in Figure 15.6, it is modelled as a second-order system, given by

$$F = Z_r V \tag{15.23}$$

F and V denoting the force and velocity, respectively; the impedance Z_r is defined as

$$Z_r = M_r s + B_r + \frac{K_r}{s} \tag{15.24}$$

Similarly, we have for the environment:

$$F = Z_r V \tag{15.25}$$

where

$$Z_e = M_e s + B_e + \frac{K_e}{s} \tag{15.26}$$

The output transfer function with force feedback is defined as

$$\frac{F}{F_d} = \frac{\frac{Z_e}{Z_r}}{1 + \frac{Z_e}{Z_r}} = \frac{Z_e}{Z_r + Z_e} \tag{15.27}$$

whose error transfer function is

$$E = F_d - F = F_d - \frac{Z_e}{Z_r + Z_e} F_d = \frac{Z_r}{Z_r + Z_e} F_d \tag{15.28}$$

and the steady-state error

$$e_{ss} = \lim_{s \to 0} s \frac{Z_r}{Z_r + Z_e} \frac{f_d}{s} \tag{15.29}$$

Substituting Z_r and Z_e of Equations (15.24) and (15.26) into (15.29), and multiply both the numerator and the denominator by s yields

$$e_{ss} = \lim_{s \to 0} \frac{M_r s^2 + B_r s + K_r}{M_r s^2 + B_r s + K_r + M_e s^2 + B_e s + K_e} f_d = \frac{K_r}{K_r + K_e} f_d \tag{15.30}$$

or equivalently

$$e_{ss} = \frac{K_r / K_e}{1 + K_r / K_e} f_d \tag{15.31}$$

It can be readily seen that K_r/K_e should approach zero, i.e., robot stiffness much lower than the environment stiffness, for the steady state error to vanish.

15.3.4 Dynamic Relations

This analysis focuses on imposing a desired dynamic behaviour to the interaction between the end-effector and the environment, assuming the environment undergoes some small deflections (low contact forces) during the interaction. To do this, the generalised contact dynamics of a robot are used. The dynamics of a robot with the contact force in joint space are

$$\mathbf{M}_q\, \ddot{\mathbf{q}} + \mathbf{C}_q\, \dot{\mathbf{q}} + \mathbf{B}_q\, \dot{\mathbf{q}} + \mathbf{G}_q = \mathbf{u} - \mathbf{J}^T\, \mathbf{F}_e \tag{15.32}$$

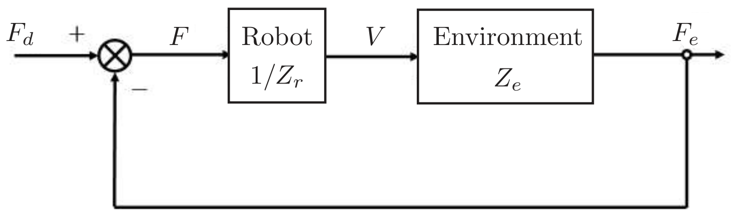


FIGURE 15.6
Block diagram of the system.

where $\mathbf{M}_q$, $\mathbf{C}_q$, $\mathbf{G}_q$ are the equivalent mass matrix, centrifugal and Coriolis matrix, and gravity term, respectively. $\mathbf{F}_e$ is an interaction between the end-effector and its environment, and the Jacobian of the system, $\mathbf{J}$ is used to project this force to the joint forces and torques. Additionally, the dynamics of a robot with contact force in task space is described in (14.79). By modelling the dynamics in the task space, the transformation of the end-effector forces is not required but instead requires the conversion of $\mathbf{M}$, $\mathbf{C}$, $\mathbf{G}$ matrices (14.79).

The design of the control law is completed in two steps: linearisation and decoupling of the system. Here we describe feedback linearisation in joint space, and is derived from Equations (14.85) and (14.88)

$$\mathbf{u} = \mathbf{M}_q\,\mathbf{a}_x + \mathbf{C}_q\,\dot{\mathbf{q}} + \mathbf{B}_q + \mathbf{G}_q + \mathbf{J}^T\,\mathbf{F}_e \tag{15.33}$$

where

$$\ddot{\mathbf{q}} = \mathbf{J}^{-1}\,\mathbf{a}_x - \mathbf{J}^{-1}\,\dot{\mathbf{J}}\,\dot{\mathbf{q}} \tag{15.34}$$

and

$$\mathbf{a}_x = \mathbf{J}\,\ddot{\mathbf{q}} + \mathbf{J}\,\dot{\mathbf{q}} = \ddot{\mathbf{x}} \tag{15.35}$$

The dynamic impedance model is described as

$$\mathbf{M}_m(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_d) + \mathbf{D}_m(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_d) + \mathbf{K}_m(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_d) = -\,\mathbf{F}_a \tag{15.36}$$

where $\mathbf{M}_m$, $\mathbf{D}_m$, $\mathbf{K}_m$ are the desired inertia, damping, and stiffness matrices of the system, respectively, y and $\mathbf{F}_a$ is the external forces from the environment. To apply this controller to our robotic system, substitute the term $\ddot{\mathbf{x}}$ with $\mathbf{a}_x$, which yields

$$\mathbf{a}_x = \ddot{\mathbf{x}}_d + \mathbf{M}_m{}^{-1}\{\mathbf{D}_m(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_d) + \mathbf{K}_m(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_d) + \mathbf{F}_a\} \tag{15.37}$$

where $\mathbf{x}_d$ is the desired motion, which typically interacts with a compliant environment, generating contact forces. Combining equations (15.33) and (15.37), we get our final impedance model (the desired error dynamics)

$$\begin{aligned}\mathbf{u} = \mathbf{M}_q\,\mathbf{J}^{-1}\left(\ddot{\mathbf{x}}_d + \mathbf{M}_m{}^{-1}\{\mathbf{D}_m\,(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_d) + \mathbf{K}_m(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_d) + \mathbf{F}_a\}\right)&\\ + \mathbf{C}_q\,\dot{\mathbf{q}} + \mathbf{B}_{\mathbf{q}} + \mathbf{G}_q + \mathbf{J}^T\,\mathbf{F}_e&\end{aligned} \tag{15.38}$$

which can be simplified to

$$\begin{aligned}\mathbf{u} = \mathbf{M}_q\,\mathbf{J}^{-1}\,\ddot{\mathbf{x}}_d + \mathbf{M}_q\,\mathbf{J}^{-1}\,\mathbf{M}_m{}^{-1}\{\mathbf{D}_m(\ddot{\mathbf{x}}_d - \ddot{\mathbf{x}}) + \mathbf{K}_m(\ddot{\mathbf{x}}_d - \ddot{\mathbf{x}})\}&\\ + \mathbf{M}_q\,\mathbf{J}^{-1}\,\mathbf{M}_m{}^{-1}\,\mathbf{F}_a - \mathbf{M}_q\,\mathbf{J}^{-1}\,\dot{\mathbf{J}}\,\dot{\mathbf{q}} + \mathbf{C}_q\,\dot{\mathbf{q}} + \mathbf{B}_{\mathbf{q}} + \mathbf{G}_q + \mathbf{J}^T\,\mathbf{F}_e&\end{aligned} \tag{15.39}$$

where $\mathbf{F}_a$ and $\mathbf{F}_e$ are defined as equal and opposite to each other, i.e., $\mathbf{F}_e = -\,\mathbf{F}_a$. To further reduce the final impedance model, a simplification to remove force feedback is made. To do this we desire the following

$$\mathbf{M}_q\,\mathbf{J}^{-1}\,\mathbf{M}_m^{-1} = \mathbf{J}^{-1} \tag{15.40}$$

where

$$\mathbf{J}^{-T}\,\mathbf{M}_q\,\mathbf{J}^{-1} = \mathbf{M}_m \tag{15.41}$$

and

$$\mathbf{M}_x = \mathbf{J}^{-T}\,\mathbf{M}_q\,\mathbf{J}^{-1} \tag{15.42}$$

Therefore we choose the apparent inertia ($\mathbf{M}_x$) to equal the natural inertia ($\mathbf{M}_m$) of the robot and Equation (15.39) simplifies to

$$\begin{aligned}\mathbf{u} = \mathbf{M}_q\,\mathbf{J}^{-1}\,\ddot{\mathbf{x}}_d + \mathbf{M}_q\,\mathbf{J}^{-1}\,\mathbf{M}_m{}^{-1}\{\mathbf{D}_m(\ddot{\mathbf{x}}_d - \ddot{\mathbf{x}}) + \mathbf{K}_m(\ddot{\mathbf{x}}_d - \ddot{\mathbf{x}})\}\\ - \mathbf{M}_q\,\mathbf{J}^{-1}\,\dot{\mathbf{J}}\,\dot{\mathbf{q}} + \mathbf{C}_q\,\dot{\mathbf{q}} + \mathbf{B}_{\dot{\mathbf{q}}} + \mathbf{G}_q\end{aligned} \tag{15.43}$$

where $\mathbf{F}_a$ and $\mathbf{F}_e$ are removed from the control law. Further combining equations (15.40) and (15.43) results in our final simplified impedance control law where a force and torque sensor at the end-effector are no longer needed:

$$\begin{aligned}\mathbf{u} = \mathbf{M}_q\,\mathbf{J}^{-1}\,\ddot{\mathbf{x}}_d + \mathbf{J}^T\,\mathbf{D}_m(\ddot{\mathbf{x}}_d - \ddot{\mathbf{x}}) + \mathbf{J}^T\,\mathbf{K}_m(\ddot{\mathbf{x}}_d - \ddot{\mathbf{x}})\\ - \mathbf{M}_q\,\mathbf{J}^{-1}\,\dot{\mathbf{J}}\,\dot{\mathbf{q}} + \mathbf{C}_q\,\dot{\mathbf{q}} + \mathbf{B}_{\dot{\mathbf{q}}} + \mathbf{G}_q\end{aligned} \tag{15.44}$$

15.4 Conclusion

In this chapter, we reviewed some basic force control concepts, which are applicable where a manipulator is required to interact with the environment through its end-effector. The most basic mode of force control handles control of the end-effector interaction with the environment, where the environment is assumed to be relatively stiff. However, this mode of control does not allow control of motion in the same direction of force application. The hybrid motion-force control introduces natural and artificial constraints that allow motion to be controlled in the unconstrained directions of the end-effector workspace. This has applications in manufacturing and assembly.

Impedance control allows force control of the end-effector in which the object or environment behaves like a mass-spring-damper system. This has applications in scenarios where the end-effector is in constant contact with the environment, such as in surgical scenarios.

15.5 Exercises

Problem 1. Given

$$^0\mathbf{T}_1 = \begin{bmatrix} 0.866 & -0.5 & 0 & 10 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If the wrench at the origin of $\{0\}$ is

$$^0\mathcal{F} = \begin{bmatrix} 0 & 2 & -3 & 0 & 0 & 4 \end{bmatrix}^T$$

Find the wrench with a reference point at the origin of $\{1\}$.

$$\begin{bmatrix} ^0\mathbf{f}_1 \\ ^0\mathbf{n}_1 \end{bmatrix} = \begin{bmatrix} ^0\mathbf{R}_1 & \mathbf{0} \\ ^0\mathbf{p}_1 \times {}^0\mathbf{R}_1 & ^0\mathbf{R}_1 \end{bmatrix} \begin{bmatrix} ^1\mathbf{f}_1 \\ ^1\mathbf{n}_1 \end{bmatrix}$$

Problem 2. Give the natural and artificial constraints for the motion of uncorking a bottle of wine using a corkscrew. Assume the corkscrew is already inserted and no friction or pressure forces present. Sketch the definition of the frame corresponding to the constraints.

Problem 3. Re-attempt the previous problem but without the simplistic force-domain assumptions, i.e., assume non-zero friction between the cork and the bottle, along with forces due to the pressure inside the bottle. Observe the difference between the resultant constraints and those of the previous problem.

Part III

Advanced Analysis and Case Studies

16 Mobility Analysis

The study of mobility or degree of freedom is often the very first step in designing a mechanism to fulfil specific tasks. In the previous parts of this book, we discussed serial robotic manipulators and planar kinematic chains, whose mobility can be readily identified through the Chebychev–Grübler–Kutzbach (CGK) criterion [1, 2, 3]. However, this method is associated with the following demerits:

- The analysis of parallel manipulators (a more in-depth discussion on parallel manipulators can be found in Chapter 20) is tedious due to the large number of links, multi-DoF joints, and closed-loop kinematic chains.
- The outcome of the analysis does not provide information on other critical aspects of the parallel manipulator of interest, e.g., the property of the motion and the actuation pattern.
- The method encounters limitations in handling special cases where the kinematic chain gains or loses mobility upon certain geometrical relationships or configurations.

We aim to provide more insights into solving the aforementioned limitations in this chapter. Specifically, an approach [4] based on the pattern of transformation matrix tackles the former two, and another method [5] featuring Taylor's theorem of constraint equations addresses the latter.

16.1 Mobility Analysis Based on the Pattern of Transformation Matrix

16.1.1 Pattern of Transformation Matrix

Based on the International Federation for the Promotion of Mechanism and Machine Science (IFToMM) definition [6], mobility is defined as *the number of independent coordinates needed to define the configuration of a kinematic chain or mechanism.* Following this definition, when joint variables are used to construct the array of generalised coordinates, unconstrained joint variables form its independent subset. As such, the mobility can be equivalently expressed as $n - c$, where n and c identify the numbers of joint variables and independent constraint equations, respectively.

The joint variables of actuated and passive joints of a robotic manipulator are embedded in the transformation matrices, which, in turn, describe the configurations of its end-effector. This feature provides a convenient pathway to study the mobility of a parallel manipulator based on the pattern of the transformation matrices.

Before diving specifically into parallel manipulators, it is essential to interpret the information provided by a transformation matrix in the general case. The positional entries are

DOI: 10.1201/9781003614319-16

generally unconstrained, leading to three independent entries. The orientational ones, i.e., those from the rotation matrix, belong to a special orthonormal subgroup $SO(3)$. Three of the nine entries are independent, as six constraints are derived from the orthogonal properties and normal conditions. On the other hand, the number of independent entries is often smaller than six (three positional + three orientational) for real-world kinematic chains, rendering it necessary to observe the pattern of the transformation matrix in the mobility analysis.

The pattern of a transformation matrix is defined based on the positions/indices (row-column) of non-zero entries with joint variables and the number of all independent entries [4]. Taking a generalised transformation matrix of an unconstrained body as an example:

$$\mathbf{T}\left(\theta\right) = \begin{bmatrix} t_{11} & t_{12} & t_{13} & p_x \\ t_{21} & t_{22} & t_{23} & p_y \\ t_{31} & t_{32} & t_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Its non-zero entries are (11,12,$\cdots$,34), and we know from the previous discussion that $SO(3)$ provides six constraints despite having 12 non-zero matrix entries, i.e., $k = 6$ matching the six DoF of a free-moving spatial body.

The patterns of commonly used motion subgroups are summarised in Table 16.1. The complete table exhausting all 12 motion subgroups can be found in our previous work [4].

For parallel manipulators, the loop-closure equation connects transformation matrices of the end-effector derived from individual legs, or

$$^{0}\mathbf{T}_{n1}\left(\mathbf{q}_1\right) = {}^{0}\mathbf{T}_{n2}\left(\mathbf{q}_2\right) = \cdots = {}^{0}\mathbf{T}_{nj}\left(\mathbf{q}_j\right) = \cdots = {}^{0}\mathbf{T}_{nj_{max}}\left(\mathbf{q}_{j_{max}}\right) \tag{16.1}$$

$^{0}\mathbf{T}_{nj}\left(\mathbf{q}_j\right)$ denoting the transformation matrix of the end-effector (body n) expressed using the joint variables $\mathbf{q}_j$ of leg j; j_{max} identifying the number of legs. The number of individual entries can then be determined by comparing the joint-variable-embedded entries of $^{0}\mathbf{T}_{nj}$ at matching positions.

In the ensuing subsections, we analyse example parallel mechanisms based on the constraints derived from Equation (16.1) and the transformation matrix patterns. We will focus on the following:

1. Mobility — what is the DoF of the manipulator;
2. Property of motion — which of the DoF are unconstrained and which are constrained;
3. Actuation pattern — what are the constraints on the assignment of actuators to the legs.

Moreover, we establish the criteria below for the actuation pattern:

1. Criterion 1 — The number of actuators must match the DoF to realise a fully-constrained system.
2. Criterion 2 — Independent joint variables must be distributed in such a way, that the number of dependent joint variables match that of the independent constraints in both the orientation and the position aspects.
3. Criterion 3 — For the selection of independent joint variables, the number of dependent joint variables shall match that of the independent constraints leg-wise.

TABLE 16.1
Transformation Matrix Patterns of Motion Subgroups

Subgroup	DoF or k	$\mathbf{T}$	Pattern
Zero motion group	0	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	n/a
Linear translation	1	$\begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	(14)
Single-axis rotation	1	$\begin{bmatrix} c\theta & -s\theta & 0 & 0 \\ s\theta & c\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	(11,12,21,22)
Planar translation	2	$\begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	(14,24)
Planar motion	3	$\begin{bmatrix} c\theta & -s\theta & 0 & p_x \\ s\theta & c\theta & 0 & p_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	(11,12,14,21,22,24)
Spatial translation	3	$\begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$	(14,24,34)
Spherical motion	3	$\begin{bmatrix} t_{11} & t_{12} & t_{13} & 0 \\ t_{21} & t_{22} & t_{23} & 0 \\ t_{31} & t_{32} & t_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	(11,12,$\cdots$,33)
Complete spatial motion	6	$\begin{bmatrix} t_{11} & t_{12} & t_{13} & p_x \\ t_{21} & t_{22} & t_{23} & p_y \\ t_{31} & t_{32} & t_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$	(11,12,$\cdots$,34)

16.1.2 Case Study—3-UPU Parallel Manipulator

With this case study, we demonstrate the process of mobility analysis based on simplified transformation matrices upon loop closure. A 3-UPU parallel manipulator [10] is shown in Figure 16.1, whose and the corresponding leg transformation matrix is (with leg 1 as the example):

$$ {}^{0}\mathbf{T}_{61} = \begin{bmatrix} c_1c_6c_{25} + s_1s_6 & -c_1c_6c_{25} + s_1c_6 & -c_1s_{25} & a_5c_1c_{25} + a_3\left(c_1c_2c_3 + s_1s_3\right) + a_1c_1 \\ s_1c_6c_{25} - c_1s_6 & -s_1s_6c_{25} - c_1c_6 & -s_1s_{25} & a_5s_1c_{25} + a_3\left(s_1c_2c_3 - c_1c_3\right) + a_1s_1 \\ -c_6s_{25} & s_6s_{25} & -c_{25} & -a_5s_{25} - a_3s_2c_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16.2} $$

with c and s representing the cosine and sine of the angles, respectively, and multi-digit subscripts identifying the summation of angles. a_1, a_3, and a_5 are DH parameters.

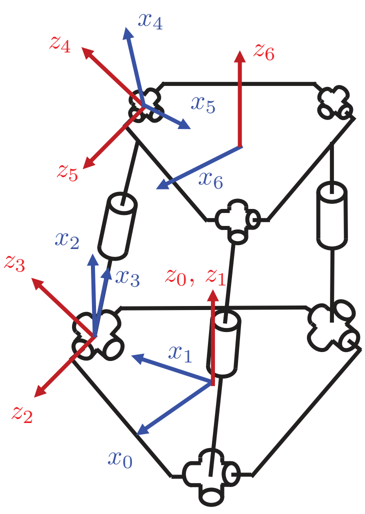


FIGURE 16.1
Schematics of the 3-UPU parallel manipulator.

Loop-closure based on Equation (16.1) leads to the following relation:

$$\begin{aligned} c_6 s_{25} &= c_6' s_{25}' = c_6'' s_{25}'' \\ s_6 s_{25} &= s_6' s_{25}' = s_6'' s_{25}'' \\ c_{25} &= c_{25}' = c_{25}'' \end{aligned} \tag{16.3}$$

It can be readily seen that $\theta_2 + \theta_5 = \theta_2' + \theta_5' = \theta_2'' + \theta_5''$. Moreover, since θ_6, θ_6', and θ_6'' are not mutually equal, $\theta_2 + \theta_5$, $\theta_2' + \theta_5'$, and $\theta_2'' + \theta_5''$ must be either 0 or π. Assuming π, back-substitution into the transformation matrix of Equation (16.2) yields

$$^{0}\mathbf{T}_{61} = \begin{bmatrix} -c_{16} & s_{16} & 0 & t_{14}\left(a_3, \theta_2, \theta_3\right) \\ s_{16} & -c_{16} & 0 & t_{24}\left(a_3, \theta_2, \theta_3\right) \\ 0 & 0 & 1 & t_{34}\left(a_3, \theta_2, \theta_3\right) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16.4}$$

Applying loop-closure for the second time further leads to $\theta_1 + \theta_6 = \theta_1' + \theta_6' = \theta_1'' + \theta_6''$. Furthermore, since these are all fixed angles dependent on the design of the base plate and the moving platform, while the base plate and the moving platform are rotational symmetry, we can conveniently rotate $\{1\}$ and $\{6\}$ around their z axes such that $\theta_1 + \theta_6 = \pi$, $\theta_1' + \theta_6' = \pi$, and $\theta_1'' + \theta_6'' = \pi$. From there, the second back-substitution further simplifies the transformation matrix into

$$^{0}\mathbf{T}_{61} = \begin{bmatrix} \mathbf{1} & \mathbf{p}\left(a_3, \theta_2, \theta_3\right) \\ \mathbf{0} & 1 \end{bmatrix} \tag{16.5}$$

$\mathbf{p}\left(a_3, \theta_2, \theta_3\right)$ is the position vector of leg 1, and $\mathbf{p}'$ and $\mathbf{p}''$ of legs 2 and 3 shares a similar form.

Upon simplification, we can base the mobility analysis upon (16.5).

Mobility

Six positional independent constraints can be obtained upon loop-closure: three scalar equations each from $\mathbf{p} = \mathbf{p}'$ and $\mathbf{p} = \mathbf{p}''$. With nine joint variables and six independent constraints, the 3-UPU manipulator is of three DoF.

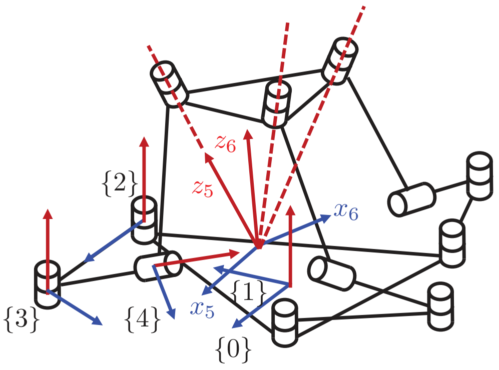


FIGURE 16.2
Schematics of the 3R2T mechanism.

Property of Motion

The 3-UPU is a translational parallel manipulator, as shown by the identity rotation matrix.

Actuation Pattern

Criteria 1 to 3 pose no limitation on the selection and allocation of actuated joints. Any combination of three joint variables will make the 3-UPU manipulator fully-actuated.

16.1.3 Case Study—3R2T Parallel Manipulator

With this case study, we move one step further to demonstrate that our method can be extended and remains valid without explicit expression of the transformation matrices, if the kinematic chain is not subject to special geometric conditions and the end-effector has no special motion property. This case study focuses on a 3R2T parallel manipulator [8] illustrated in Figure 16.2. It adopts a 3-RRTR architecture where the three legs all feature two proximal revolute joints, one prismatic joint and the distal revolute joint, from the base plate to the mobile platform. Frames $\{0\}$ and $\{1\}$ are on the base plate. The joint axes of the distal revolute joints intersect at a point offset from the mobile platform, where the origins of $\{5\}$ and $\{6\}$ are located.

The leg transformation matrix is given by:

$$ {}^{0}\mathbf{T}_{61} = \begin{bmatrix} t_{11}(\theta_2,\theta_3,\theta_4,\theta_5) & t_{12}(\theta_2,\theta_3,\theta_4,\theta_5) & t_{13}(\theta_2,\theta_3,\theta_4,\theta_5) & t_{14}(\theta_2,\theta_3,d_4) \\ t_{21}(\theta_2,\theta_3,\theta_4,\theta_5) & t_{22}(\theta_2,\theta_3,\theta_4,\theta_5) & t_{23}(\theta_2,\theta_3,\theta_4,\theta_5) & t_{24}(\theta_2,\theta_3,d_4) \\ t_{31}(\theta_4,\theta_5) & t_{32}(\theta_4,\theta_5) & t_{33}(\theta_4,\theta_5) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16.6} $$

While all three legs bear transformation matrices of the same form, we distinguish the entries and joint variables of legs 1 to 3 by $t_{pq}(\mathbf{q})$, $t'_{pq}(\mathbf{q}')$, and $t''_{pq}(\mathbf{q}'')$, respectively. This notation is adopted in all the ensuing case studies. The transformation matrices are fed into Equation (16.1) for further analysis. It should be noted that the detailed expressions of t_{pq} are not always required in this process.

Mobility

The comparison between ${}^0\mathbf{T}_{61}$-${}^0\mathbf{T}_{62}$ and ${}^0\mathbf{T}_{61}$-${}^0\mathbf{T}_{63}$ each yields five sets of independent constraints: two from non-zero position entries, and three from the orientation ones, rendering ten independent constraints. Note that while the comparison between orientation entries of two transformation matrices provides nine sets of constraints, only three are independent due to being $SO(3)$. With 15 joint variables, the mechanism is of five DoF.

Property of Motion

Position-wise, t_{14} and t_{24} are non-zero while t_{34} is zero, indicating two-DoF translation along the x- and y-axes.

Actuation Pattern

Based on Criterion 2, the independent joint variables must be selected from those embedded in the positional constraints, i.e., θ_2, θ_3, d_4, θ_2', θ_3', d_4', θ_2'', θ_3'', and d_4''. This way, the four dependent joint variables can be solved using the four positional independent constraints. Additionally, once all the positional joint variables are determined, there are six dependent joint variables (θ_4, θ_5, θ_4', θ_5', θ_4'', and θ_5'') and a matching number of independent constraints from the orientational side, i.e., all joint variables are solvable. Moreover, following Criterion 3, at least one joint variable from each leg must be independent, i.e., a 2-2-1 arrangement among the legs. In the opposite case where, for example, all five independent joint variables were selected from those of legs 1 and 2, there will be three dependent joint variables θ_2'', θ_3'', and d_4'' with two independent constraints $t_{14} = t_{14}''$, and $t_{24} = t_{24}''$, leaving the joint variables under-determined.

16.1.4 Case Study—3R1T Parallel Manipulator

The 2-URRH (universal-revolute-revolute-helical) parallel manipulator [11], depicted in Figure 16.3, is included as another example to showcase the motion analysis without explicit equations of transformation matrix entries. On the other hand, it also demonstrates the limitation in our method in revealing specific motion characteristics, which we will elaborate when discussing the property of motion.

The 3R1T mechanism comprises two identical legs of URRH kinematic chains. To clarify

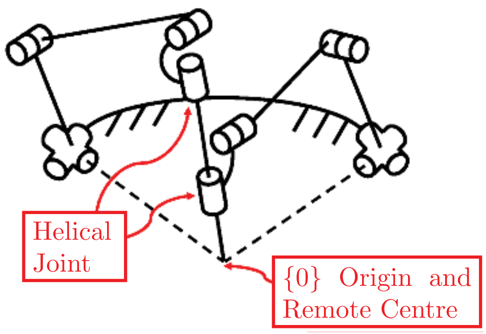


FIGURE 16.3
Schematics of the 2-URRH parallel manipulator.

the kinematics, we refer to the leg architecture as RRRRH, where the universal joint is replaced with two revolute ones. The leg transformation matrix is given by (leg 1 with subscript 1 as the example):

$$\begin{aligned} {}^{0}\mathbf{T}_{E1}\left(\alpha_1,\beta_1,x_1,z_1,\gamma_1\right) &= \mathbf{T}\left(\alpha_1\right)\mathbf{T}\left(\beta_1,x_1,z_1\right)\mathbf{T}\left(\gamma_1\right) \\ &= \begin{bmatrix} t_{11}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{12}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{13}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{14}\left(\beta_1,x_1,\gamma_1\right) \\ t_{21}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{22}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{23}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{24}\left(\alpha_1,\beta_1,z_1,\gamma_1\right) \\ t_{31}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{32}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{33}\left(\alpha_1,\beta_1,\gamma_1\right) & t_{34}\left(\alpha_1,\beta_1,z_1,\gamma_1\right) \end{bmatrix} \end{aligned} \tag{16.7}$$

subscript E identifying the frame attached to the end-effector. Of the three transformation matrices, $\mathbf{T}\left(\alpha_1\right)$ is resultant from the rotation of the first revolute joint, which defines the plane in which the leg moves; $\mathbf{T}\left(\beta_1,x_1,z_1\right)$ defines in-plane position and orientation of the end-effector; and $\mathbf{T}\left(\gamma_1\right)$ controls the axial rotation and longitudinal translation of the end-effector through a helical joint. Additionally, for the second transformation matrix $\mathbf{T}\left(\beta_1,x_1,z_1\right)$, we adopted two positional and one orientational joint variable as opposed to three joint angles. This is achievable since the second and the third revolute joints merely define the in-plane position of the fourth revolute joint. The transformation matrix of leg 2 shows a similar pattern, except joint variable x_1 is replaced by y_2, i.e., ${}^{0}\mathbf{T}_{E2}\left(\alpha_2,\beta_2,y_2,z_2,\gamma_2\right)$.

Property of Motion

In the leg rotation matrices, the 2-URRH mechanism has three independent orientational constraints and six variables (α_1, β_1, γ_1, α_2, β_2, and γ_2), which means it is of 3R. Once the orientational joint variables have been determined, the leg position vectors feature four unknown variables, x_1, z_1, y_2, and z_2, but three independent constraint equations. As such, there is one remaining translational DoF, i.e., the 2-URRH parallel manipulator is of 3R1T.

On another note, this mechanism is a remote center of motion mechanism where the end-effector pivots around and translates through a fixed point in the space. Such a characteristic can be readily observed, as the two planes in which the legs move always create an intersection, and the origin of the global frame, $[0,\ 0,\ 0]^T$, is a particular solution on the line of intersection. However, without their explicit expressions, such a characteristic is not directly visible from the pattern of leg transformation matrices.

Mobility

As described previously, the mechanism is of 3R1T.

Actuation Pattern

Criterion 2 demands at least one orientational joint variable to be selected independently; otherwise, the position will be over-constrained, but the orientation will be under-constrained. Additionally, criterion 3 requires both legs to have at least one independent joint variable so that the legs are not overly- or under-constrained. All other actuation patterns are valid. In [11], α_1, β_1, α_2, and β_2 are chosen for the convenience of actuation from the base.

16.1.5 Case Study—2-RPS-UPS Parallel Manipulator

The manipulator of interest here is the 2-RPS-UPS one displayed in Figure 16.4. We move one step further to demonstrate that our method may be extended to the cases where transformation matrices can be omitted.

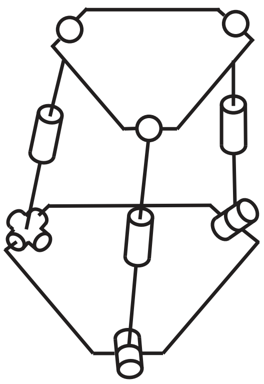


FIGURE 16.4
Schematics of the 2-RPS-UPS parallel manipulator.

It can be readily seen that the positions of the spherical joints are $\mathbf{p}\,(\theta_1, d_1)$, $\mathbf{p}'\,(\theta_2, d_2)$, and $\mathbf{p}''\,(\gamma_1, \gamma_2, d_3)$, respectively, for three legs. θ_1 and θ_2 are the angles of the revolute joints, γ_1 and γ_2 represent the rotation of the two revolute joints forming the universal joint, and the ds denote the positions of the prismatic joints. We take advantage of the distances between adjacent spherical joints to express three independent kinematic constraints:

$$\begin{aligned} g_1\,(\theta_1, d_1, \theta_2, d_2) &= ||\,\mathbf{p} - \mathbf{p}'|| - l_1^2 \\ g_2\,(\theta_2, d_2, \gamma_1, \gamma_2, d_3) &= ||\,\mathbf{p}' - \mathbf{p}''|| - l_2^2 \\ g_3\,(\theta_1, d_1, \gamma_1, \gamma_2, d_3) &= ||\,\mathbf{p} - \mathbf{p}''|| - l_3^2 \end{aligned} \tag{16.8}$$

where l_1, l_2, and l_3 are the distance between adjacent spherical joints, i.e., design parameters of the end-effector.

Mobility

With three independent constraints of (16.8) and seven joint variables, the 2-RPS-UPS manipulator is of four DoF.

Property of Motion

No special condition.

Actuation Pattern

Criterion 3 requires at least one joint variable of leg 3, i.e., one of ϕ_1, ϕ_2, or d_3, to be independent, otherwise leg 3 will become under-determined. All other actuation patterns are valid.

16.2 The Order of Mobility

The CGK criterion and our pattern-based method described previously focus on the global mobility of mechanisms. However, when special geometrical conditions are satisfied, a mechanism may gain or lose DoF. An example demonstrating such a phenomenon is the Bennett linkage [12]. By having four links (including ground) connected by four revolute joints, the Bennett linkage is, on paper, an over-constrained mechanism despite having one DoF in reality. In this section, we discuss a method that takes advantage of the Taylor expansion of constraint equations to reveal the local mobility of mechanisms. A novel spatial four-bar linkage with second-order local mobility is used as the case study.

16.2.1 Local Mobility

The kinematically-valid mechanism, described by array $\mathbf{q}$ of the generalised coordinates, must satisfy the following kinematic constraints:

$$\boldsymbol{\phi}\left(\mathbf{q}\right) = \mathbf{0} \tag{16.9}$$

where scalar constraint equations ϕ_j ($j \in [1,\ m]$) and entries q of the generalised coordinates are not necessarily independent. Suppose two distinct solutions exist that satisfy Equation (16.9), and we refer to one as $\mathbf{q}_1 = \mathbf{p}$ and the other $\mathbf{q}_2 = \mathbf{p} + \delta\mathbf{p}$, any scalar constraint equation ϕ_j of the latter can be expressed in terms of that of the former and Taylor expansion as

$$\begin{aligned}\phi_j\left(\mathbf{p}+\delta\mathbf{p}\right) =& \phi_j\left(\mathbf{p}\right) + \sum_{i=1}^{n} \delta p_i \frac{\partial \phi_j\left(\mathbf{q}\right)}{\partial q_i}\bigg|_{\mathbf{q}=\mathbf{p}} + \frac{1}{2!}\left(\sum_{i=1}^{n} \delta p_i \frac{\partial}{\partial q_i}\right)^2 \boldsymbol{\phi}\left(\mathbf{q}\right)\bigg|_{\mathbf{q}=\mathbf{p}} \\ &+ \cdots + \frac{1}{k!}\left(\sum_{i=1}^{n} \delta p_i \frac{\partial}{\partial q_i}\right)^k \boldsymbol{\phi}\left(\mathbf{q}\right)\bigg|_{\mathbf{q}=\mathbf{p}} + \cdots = 0, \quad j = 1, 2, \cdots, m\end{aligned} \tag{16.10}$$

q_i and δp_i identifying the generalised coordinates and the scalar displacements of $\delta\mathbf{p}$, respectively, and $\mathbf{q} \in \mathbb{R}^{n\times 1}$. Note that $\phi_j\left(\mathbf{p}\right)$ vanishes as per Equation (16.9).

In Equation (16.10), if $\delta\mathbf{p}$ is infinitesimal and all components with exponents greater than 1 are negligible, the equation can be re-expressed as

$${}^{1}\phi_j\left(\mathbf{p}+\delta\mathbf{p}\right) = \sum_{i=1}^{n} \delta p_i \frac{\partial \phi_j\left(\mathbf{q}\right)}{\partial q_i}\bigg|_{\mathbf{q}=\mathbf{p}} = 0, \quad j = 1, 2, \cdots, m \tag{16.11}$$

superscript 1 identifying the first-order approximation. Correspondingly, the *first-order local mobility* of a mechanism around configuration $\mathbf{p}$ is determined by the number of δp_i that can be selected freely and satisfy Equation (16.11). Said equation can also be written with Jacobian matrix $\mathbf{J}$ as

$${}^{1}\phi_j\left(\mathbf{p}+\delta\mathbf{p}\right) = \begin{bmatrix} \frac{\partial\phi_1(\mathbf{q})}{\partial q_1} & \cdots & \frac{\partial\phi_1(\mathbf{q})}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial\phi_m(\mathbf{q})}{\partial q_1} & \cdots & \frac{\partial\phi_m(\mathbf{q})}{\partial q_n} \end{bmatrix} \begin{bmatrix} \delta p_1 \\ \vdots \\ \delta p_n \end{bmatrix} = \mathbf{0} = \mathbf{J}\,\delta\mathbf{p} \tag{16.12}$$

Assuming $\mathbf{J}$ has a rank of r, the mobility of the mechanism is $n - r$, which agrees with the definition of mobility reported in [6].

Suppose all components with exponents greater than 2 are now neglected, Equation (16.10) becomes

$${}^{2}\phi_j\left(\mathbf{p}+\delta\mathbf{p}\right) = \sum_{i=1}^{n} \delta p_i \frac{\partial \phi_j\left(\mathbf{q}\right)}{\partial q_i}\bigg|_{\mathbf{q}=\mathbf{p}} + \frac{1}{2!}\left(\sum_{i=1}^{n} \delta p_i \frac{\partial}{\partial q_i}\right)^2 \boldsymbol{\phi}\left(\mathbf{q}\right)\bigg|_{\mathbf{q}=\mathbf{p}} = 0 \tag{16.13}$$

superscript 2 indicating the second-order approximation. Similar to that of Equation (16.12), we refer the number of δp_i that can be selected freely and satisfy Equation (16.13) to the *second-order local mobility* of the mechanism.

If $\delta \mathbf{p}$ is further increased until all components with exponents greater than s may be neglected, we obtain the s-th order approximation of Equation (16.10) and the resultant local mobility is of s-th order. Moreover, the ∞-order local mobility is equivalent to the global mobility. Notably, a higher-order local mobility must also be a lower-order one, and global mobility is any-order local mobility since the higher-order approximation of Equation (16.10) contains all the components of its lower-order counterparts.

For the first-order local mobility, the approximation displayed in Equation (16.11) forms a set of hyper-surfaces generating a common sub-manifold, who is, in turn, tangent to all the hyper-surfaces. A point representing the configuration of the mechanism can move freely within the sub-manifold, satisfying the kinematic constraints. Additionally, the dimension of the sub-manifold is the first-order mobility. Similarly, the second-order approximation of Equation (16.10) also produces a set of hyper-surfaces and generates a sub-manifold for the configuration point to move within. Moreover, a mechanism can move more smoothly near configuration $\mathbf{p}$ with the second-order local mobility than its counterpart with only the first-order.

16.2.2 Case Study—A Novel Spatial Four-Bar Linkage

We analyse the mobility of the special four-bar linkage as the case study [5]. As mentioned at the start of this section, a spatial four-bar linkage should feature zero DoF, i.e., it is a structure, not a mechanism. On the other hand, we will also demonstrate that our linkage has one-DoF local mobility.

To start with, the constraints for a generalised spatial four-bar linkage (Figure 16.5) are derived based on loop-closure equations. We borrow the concept of DH notation here, where $\mathbf{z}_i$ and $\mathbf{x}_i$ represent the joint axes and the common perpendicular lines, respectively, and DH parameters a_i and d_i are obtained correspondingly. The loop-closure equation is given by:

$$a_1\,\mathbf{x}_1 + d_1\,\mathbf{z}_1 + a_2\,\mathbf{x}_2 + d_2\,\mathbf{z}_2 + a_3\,\mathbf{x}_3 + d_3\,\mathbf{z}_3 + a_4\,\mathbf{x}_4 + d_4\,\mathbf{z}_4 = \mathbf{0} \tag{16.14}$$

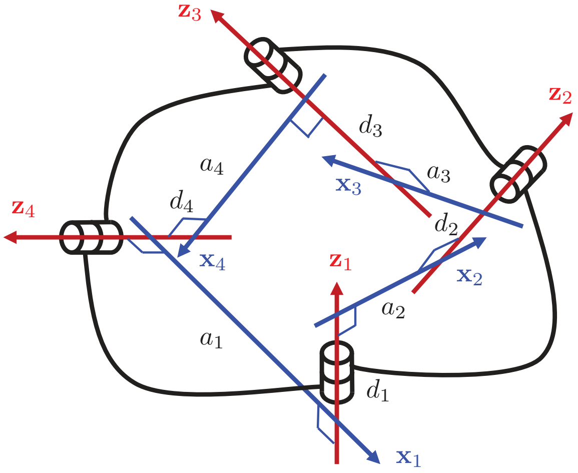


FIGURE 16.5
Schematics of the spatial four-bar linkage.

For simplicity, we assume all d are zero, rendering

$$a_1\,\mathbf{x}_1 + a_2\,\mathbf{x}_2 + a_3\,\mathbf{x}_3 + a_4\,\mathbf{x}_4 = \mathbf{0} \tag{16.15}$$

or equivalently

$$\begin{aligned} a_1\,\mathbf{x}_1 + a_2\,\mathbf{x}_2 &= -a_3\,\mathbf{x}_3 - a_4\,\mathbf{x}_4 \\ a_1\,\mathbf{x}_1 + a_4\,\mathbf{x}_4 &= -a_2\,\mathbf{x}_2 - a_3\,\mathbf{x}_3 \end{aligned} \tag{16.16}$$

Dot-producting the left-hand and right-hand sides of Equation (16.16) with themselves, respectively, leads to

$$\begin{aligned} a_1^2 + a_2^2 + 2a_1a_2\,\mathbf{x}_1^T\,\mathbf{x}_2 &= a_3^2 + a_4^2 + 2a_3a_4\,\mathbf{x}_3^T\,\mathbf{x}_4 \\ a_1^2 + a_4^2 + 2a_1a_4\,\mathbf{x}_1^T\,\mathbf{x}_2 &= a_2^2 + a_3^2 + 2a_2a_3\,\mathbf{x}_2^T\,\mathbf{x}_3 \end{aligned} \tag{16.17}$$

On the other hand, dot-producting Equation (16.15) with $\mathbf{z}_1$ to $\mathbf{z}_4$ and $\mathbf{x}_1$ to $\mathbf{x}_4$, respectively, yields

$$\begin{aligned} a_1\,\mathbf{z}_1^T\,\mathbf{x}_1 + a_2\,\mathbf{z}_1^T\,\mathbf{x}_2 + a_3\,\mathbf{z}_1^T\,\mathbf{x}_3 + a_4\,\mathbf{z}_1^T\,\mathbf{x}_4 &= 0 \\ a_1\,\mathbf{z}_2^T\,\mathbf{x}_1 + a_2\,\mathbf{z}_2^T\,\mathbf{x}_2 + a_3\,\mathbf{z}_2^T\,\mathbf{x}_3 + a_4\,\mathbf{z}_2^T\,\mathbf{x}_4 &= 0 \\ a_1\,\mathbf{z}_3^T\,\mathbf{x}_1 + a_2\,\mathbf{z}_3^T\,\mathbf{x}_2 + a_3\,\mathbf{z}_3^T\,\mathbf{x}_3 + a_4\,\mathbf{z}_3^T\,\mathbf{x}_4 &= 0 \\ a_1\,\mathbf{z}_4^T\,\mathbf{x}_1 + a_2\,\mathbf{z}_4^T\,\mathbf{x}_2 + a_3\,\mathbf{z}_4^T\,\mathbf{x}_3 + a_4\,\mathbf{z}_4^T\,\mathbf{x}_4 &= 0 \end{aligned} \tag{16.18}$$

and

$$\begin{aligned} a_1\,\mathbf{x}_1^T\,\mathbf{x}_1 + a_2\,\mathbf{x}_1^T\,\mathbf{x}_2 + a_3\,\mathbf{x}_1^T\,\mathbf{x}_3 + a_4\,\mathbf{x}_1^T\,\mathbf{x}_4 &= 0 \\ a_1\,\mathbf{x}_2^T\,\mathbf{x}_1 + a_2\,\mathbf{x}_2^T\,\mathbf{x}_2 + a_3\,\mathbf{x}_2^T\,\mathbf{x}_3 + a_4\,\mathbf{x}_2^T\,\mathbf{x}_4 &= 0 \\ a_1\,\mathbf{x}_3^T\,\mathbf{x}_1 + a_2\,\mathbf{x}_3^T\,\mathbf{x}_2 + a_3\,\mathbf{x}_3^T\,\mathbf{x}_3 + a_4\,\mathbf{x}_3^T\,\mathbf{x}_4 &= 0 \\ a_1\,\mathbf{x}_4^T\,\mathbf{x}_1 + a_2\,\mathbf{x}_4^T\,\mathbf{x}_2 + a_3\,\mathbf{x}_4^T\,\mathbf{x}_3 + a_4\,\mathbf{x}_4^T\,\mathbf{x}_4 &= 0 \end{aligned} \tag{16.19}$$

Equations (16.17), (16.18), and (16.19) combined serve as the complete set of scalar kinematic constraints derived from (16.14) [5]. Upon further simplification [5], said set becomes:

$$\begin{aligned} & a_1^2 + a_2^2 + 2a_1a_2\cos\theta_1 = a_3^2 + a_4^2 + 2a_3a_4\cos\theta_3 \\ & a_1^2 + a_4^2 + 2a_1a_4\cos\theta_4 = a_2^2 + a_3^2 + 2a_2a_3\cos\theta_2 \\ & a_3\sin\alpha_2\sin\theta_2 + a_4\sin\alpha_1\sin\theta_4 = 0 \\ & a_4\sin\alpha_4\sin\theta_4 + a_1\sin\alpha_2\sin\theta_1 = 0 \\ & a_1\sin\alpha_4\sin\theta_4 + a_2\sin\alpha_3\sin\theta_2 = 0 \\ & a_2\sin\alpha_1\sin\theta_1 + a_3\sin\alpha_4\sin\theta_3 = 0 \\ & a_1 + a_2\cos\theta_1 + a_3\left(\cos\theta_3\cos\theta_4 - \sin\theta_3\sin\theta_4\cos\alpha_4\right) + a_4\cos\theta_4 = 0 \\ & a_2 + a_3\cos\theta_2 + a_4\left(\cos\theta_4\cos\theta_1 - \sin\theta_4\sin\theta_1\cos\alpha_1\right) + a_1\cos\theta_1 = 0 \\ & a_3 + a_4\cos\theta_3 + a_1\left(\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2\cos\alpha_2\right) + a_2\cos\theta_2 = 0 \\ & a_4 + a_1\cos\theta_4 + a_2\left(\cos\theta_2\cos\theta_3 - \sin\theta_2\sin\theta_3\cos\alpha_3\right) + a_3\cos\theta_3 = 0 \end{aligned} \tag{16.20}$$

where θ_i and α_i are defined following the DH notation, and $\cos\theta_i$ and $\cos\alpha_i$ are the results of dot product between adjacent $\mathbf{x}_i$ and adjacent $\mathbf{z}_i$, respectively.

Our special four-bar linkage has the following geometrical features:

$$\begin{aligned} & a_1 = a_2 \\ & a_3 = a_4 \\ & a_2 \neq a_4 \\ & \alpha_1 = -\alpha_2 = \alpha_3 = -\alpha_4 \end{aligned} \tag{16.21}$$

Substituting these conditions into the generalised constraints of (16.20) and removing the duplicated scalar equations results in

$$a_2^2\left(1+\cos\theta_1\right)=a_4^2\left(1+\cos\theta_3\right) \tag{16.22}$$

$$\cos\theta_2=\cos\theta_4 \tag{16.23}$$

$$\sin\theta_2=\sin\theta_4 \tag{16.24}$$

$$a_2\sin\theta_1=a_4\sin\theta_3 \tag{16.25}$$

$$a_2+a_2\cos\theta_1+a_4\left(\cos\theta_3\cos\theta_4-\sin\theta_3\sin\theta_4\cos\alpha_4\right)+a_4\cos\theta_4=0 \tag{16.26}$$

$$a_2+a_4\cos\theta_2+a_4\left(\cos\theta_4\cos\theta_1-\sin\theta_4\sin\theta_1\cos\alpha_1\right)+a_2\cos\theta_1=0 \tag{16.27}$$

$$a_2+a_4\cos\theta_3+a_2\left(\cos\theta_1\cos\theta_2-\sin\theta_1\sin\theta_2\cos\alpha_4\right)+a_2\cos\theta_2=0 \tag{16.28}$$

$$a_2+a_2\cos\theta_4+a_2\left(\cos\theta_2\cos\theta_3-\sin\theta_2\sin\theta_3\cos\alpha_4\right)+a_4\cos\theta_3=0 \tag{16.29}$$

Equations (16.23) and (16.24) essentially indicate $\theta_2=\theta_4$. Substituting said condition into Equations (16.26) to (16.29) yields

$$a_2+a_2\cos\theta_1+a_4\left(\cos\theta_3\cos\theta_4-\sin\theta_3\sin\theta_4\cos\alpha_4\right)+a_4\cos\theta_4=0 \tag{16.30}$$

$$a_2+a_4\cos\theta_4+a_4\left(\cos\theta_4\cos\theta_1-\sin\theta_4\sin\theta_1\cos\alpha_4\right)+a_2\cos\theta_1=0 \tag{16.31}$$

$$a_4+a_4\cos\theta_3+a_2\left(\cos\theta_1\cos\theta_4-\sin\theta_1\sin\theta_4\cos\alpha_4\right)+a_4\cos\theta_4=0 \tag{16.32}$$

$$a_4+a_2\cos\theta_4+a_4\left(\cos\theta_4\cos\theta_3-\sin\theta_4\sin\theta_3\cos\alpha_4\right)+a_4\cos\theta_3=0 \tag{16.33}$$

Pre-multiplying Equation (16.30) and (16.33) with a_2 and a_4, respectively, and equaling the two equations returns (16.22). The same conclusion could be reached upon pre-multiplying Equation (16.31) and (16.32) with a_2 and a_4, respectively. Additionally, subtracting Equation (16.31) from (16.30), or equivalently Equation (16.32) from (16.33) gives identical expression of

$$\cos\theta_3\cos\theta_4-\sin\theta_3\sin\theta_4\cos\alpha_4=\cos\theta_4\cos\theta_1-\sin\theta_4\sin\theta_1\cos\alpha_4 \tag{16.34}$$

Therefore, the set of 10 scalar constraint equations can be reduced to one with four equations, namely [5],

$$a_2^2\left(1+\cos\theta_1\right)=a_4^2\left(1+\cos\theta_3\right) \tag{16.35}$$

$$a_2\sin\theta_1=a_4\sin\theta_3 \tag{16.36}$$

$$\theta_2=\theta_4 \tag{16.37}$$

$$\cos\theta_3\cos\theta_4-\sin\theta_3\sin\theta_4\cos\alpha_4=\cos\theta_4\cos\theta_1-\sin\theta_4\sin\theta_1\cos\alpha_4 \tag{16.38}$$

It can be readily observed that θ_1 and θ_3 are fully determined/determined upon solving Equations (16.35) and (16.36) and cannot be freely chosen. Furthermore, once θ_1 and θ_3 have been determined, θ_2 and θ_4 can be solved accordingly through Equations (16.37) and (16.38). As such, independent generalised coordinates cannot be chosen freely, and the special four-bar linkage has no global mobility. However, we will demonstrate below that it has local mobility based on Equations (16.35) and (16.36), and around a feasible solution $\theta_1=\theta_3=\pi$.

The first-order approximation of the constraints is derived based on Equation (16.11) as

$$\begin{aligned}{}^1\phi_1\left(\mathbf{p}+\delta\mathbf{p}\right)&=a_2\delta\theta_1-a_4\delta\theta_3=0\\{}^1\phi_2\left(\mathbf{p}+\delta\mathbf{p}\right)&=0\end{aligned} \tag{16.39}$$

where ϕ_1 and ϕ_2 represent the constraints in Equations (16.35) and (16.36), respectively. With two variables and one constraint, either θ_1 or θ_3 can be freely selected, and the special four-bar linkage has the first-order mobility of one around $\mathbf{p} = \begin{bmatrix}\pi & \pi\end{bmatrix}^T$.

The second-order approximation of the constraints is derived based on Equation (16.13) as

$$\begin{aligned} {}^2\phi_1\left(\mathbf{p}+\delta\mathbf{p}\right) &= a_2\delta\theta_1 - a_4\delta\theta_3 = 0 \\ {}^2\phi_2\left(\mathbf{p}+\delta\mathbf{p}\right) &= \frac{1}{2}\left(a_2\delta\theta_1 - a_4\delta\theta_3\right)\left(a_2\delta\theta_1 + a_4\delta\theta_3\right) = 0 \end{aligned} \tag{16.40}$$

which essentially points to $a_2\delta\theta_1 - a_4\delta\theta_3 = 0$. Again, with two variables and one constraint, the special four-bar linkage has the second-order mobility of one around $\mathbf{p} = \begin{bmatrix}\pi & \pi\end{bmatrix}^T$.

The third-order approximation of the constraints is given by

$$\begin{aligned} {}^3\phi_1\left(\mathbf{p}+\delta\mathbf{p}\right) &= a_2\delta\theta_1 - a_4\delta\theta_3 - \frac{1}{6}a_2\delta\theta_1^3 + \frac{1}{6}a_4\delta\theta_3^3 = 0 \\ {}^3\phi_2\left(\mathbf{p}+\delta\mathbf{p}\right) &= \frac{1}{2}a_2^2\delta\theta_1^2 - \frac{1}{2}a_4^2\delta\theta_3^2 = 0 \end{aligned} \tag{16.41}$$

upon simplification, ${}^3\phi_2\left(\mathbf{p}+\delta\mathbf{p}\right)$ gives two solutions $\delta\theta_1 = \pm a_4/a_2\delta\theta_3$. Substituting these solutions into ${}^3\phi_1\left(\mathbf{p}+\delta\mathbf{p}\right)$ returns

$$a_4\left(a_2 - a_4\right)\left(a_2 + a_4\right)^3\delta\theta_3^2 = 0 \tag{16.42}$$

for $\delta\theta_1 = a_4/a_2\delta\theta_3$, and

$$12a_2^2a_4\delta\theta_3 - a_4\left(a_2^2 + a_4^2\right)\delta\theta_3^2 = 0 \tag{16.43}$$

for $\delta\theta_1 = -a_4/a_2\delta\theta_3$. Either way, with condition $a_2 \neq a_4$, $\delta\theta_1$ and $\delta\theta_3$ must both be zero for the constraints to be valid, i.e., they cannot be freely chosen and our special four-bar linkage has no local mobility for third-order or above.

A prototype was constructed to demonstrate the mobility. Its design parameters are $a_1 = a_2 = 246$ mm; $a_3 = a_4 = 206$ mm, and $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = -60°$. As shown in Figure 16.6, the prototype was able to move smoothly between the folded and the deployed configurations, which is consistent with the theoretical prediction.

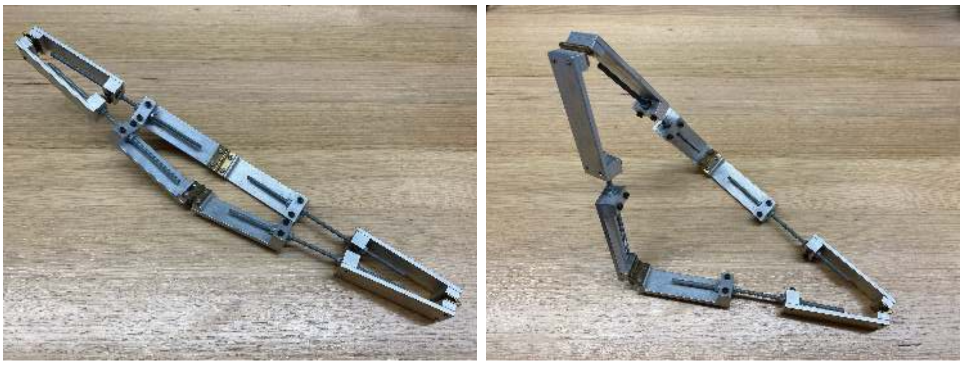


FIGURE 16.6
Prototype four-bar linkage with local mobility at folded (LHS) and deployed (RHS) configuration.

16.3 Conclusion

In this chapter, we introduced two methods for the mobility analysis of complex mechanisms. For parallel manipulators, analysis based on transformation matrices offers an easier-to-implement approach to determine the number of DoF of the system. More importantly, our method reveals the type of DoF, i.e., the property of motion, alongside constraints in the actuator arrangements. Furthermore, we demonstrated that the method could, in some cases, be applied without the explicit expressions of the transformation matrices. Another aspect we investigated is the analysis of local mobility, where we showcased the successful prediction of the conditions under which a structure gains local mobility and turns into a mechanism. The two methods discussed shall serve as powerful tools for the fast and thorough analysis of prescribed mechanisms in the design phase.

Bibliography

[1] P. L. Chebyshev, *Théorie des mécanismes connus sous le nom de parallélogrammes.* Imprimerie de l'Académie impériale des sciences, 1853.

[2] M. Grübler, *Allgemeine Eigenschaften der zwangläufigen ebenen kinematischen Ketten.* L. Simion, 1884.

[3] K. Kutzbach, "Mechanische leitungsverzweigung, ihre gesetze und anwendungen," *Maschinenbau*, vol. 8, no. 21, pp. 710–716, 1929.

[4] C. Chen, "Mobility analysis of parallel manipulators and pattern of transform matrix," *Journal of Mechanisms and Robotics*, vol. 2, no. 4, pp. 1–11, 2010.

[5] C. Chen, "The order of local mobility of mechanisms," *Mechanism and machine theory*, vol. 46, no. 9, pp. 1251–1264, 2011.

[6] T. G. Ionescu, P. Antonescu, I. Biro, G. Bögelsack, and A. K. Breteler, "Terminology for the mechanism and machine science: Chapter 0-13," *Mechanism and Machine Theory*, vol. 38, pp. 767–901, 2003.

[7] J. M. Hervé, "The lie group of rigid body displacements, a fundamental tool for mechanism design," *Mechanism and Machine theory*, vol. 34, no. 5, pp. 719–730, 1999.

[8] Q. Li, Z. Huang, and J. M. Hervé, "Type synthesis of 3r2t 5-dof parallel mechanisms using the lie group of displacements," *IEEE transactions on robotics and automation*, vol. 20, no. 2, pp. 173–180, 2004.

[9] A. Morozov and J. Angeles, "The mechanical design of a novel schönflies-motion generator," *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 1, pp. 82–93, 2007.

[10] L.-W. Tsai and S. Joshi, "Kinematics and optimization of a spatial 3-upu parallel manipulator," *J. Mech. Des.*, vol. 122, no. 4, pp. 439–446, 2000.

[11] G. Chen, J. Wang, H. Wang, C. Chen, V. Parenti-Castelli, and J. Angeles, "Design and validation of a spatial two-limb 3r1t parallel manipulator with remote center-of-motion," *Mechanism and Machine Theory*, vol. 149, p. 103807, 2020.

[12] G. T. Bennett, "A new mechanism," *Engineering*, vol. 76, p. 777, 1903.

17

Orientation Workspace

In the previous chapter, we discussed the analysis of the degree of freedom and the property of motion of parallel manipulators. In this chapter, we direct our focus on a similar study on three-DoF spherical motion robots, namely, the study on orientation workspace. The parameterisation of such workspace, e.g., by means of volume, is critical in the design process, as it allows the performance of different spherical motion robot designs to be quantified and compared. On the other hand, representing the orientation workspace in the Euclidean space is challenging. In the sub-sections below, we present our method of parameterisation based on quaternions and differential geometry [1].

17.1 Measurement Principles and Quaternions

Regardless of the method taken, the parameterisation of orientation workspace based on volume shall follow two principles of invariant [1]. Firstly, the performance index, i.e., the volume of an orientation workspace, must be invariant with respect to the selection of reference frames. Secondly, the volume of an orientation workspace must be invariant with respect to the description of the workspace. These principles render quaternion a suitable tool for the parameterisation of the volume of the orientation workspace, as explained below.

A unit quaternion is defined as

$$q_Q = a + bi + cj + dk \tag{17.1}$$

subject to

$$a^2 + b^2 + c^2 + d^2 = 1 \tag{17.2}$$

Moreover, the rotation matrix associated with q_Q is given by

$$\mathbf{R} = \begin{bmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{bmatrix} \tag{17.3}$$

More details on quaternion can be found in the Appendix for Chapter 5. The physical interpretation of the unit quaternion is a point $\mathbf{q}_Q = \begin{bmatrix} a & b & c & d \end{bmatrix}^T$ on a unit 3-sphere $\mathbb{S}^3$. The latter is, in turn, the analogy of a unit sphere in the four-dimensional space. Notably, the surface points of a sphere 1) each have their corresponding unique orientations, and 2) are equally weighted by having equal distance to the centre. As such, the volume corresponding to the collection of these workspace-representing surface points is our volume measurement of interest.

DOI: 10.1201/9781003614319-17

We adopt the spherical polar coordinates to parameterise the $\mathbb{S}^3$ and quantify its volume. For a 3-sphere, the quaternion can be expressed in the polar coordinates as

$$\begin{aligned} a &= \cos\psi \\ b &= \sin\psi\cos\phi \\ c &= \sin\psi\sin\phi\cos\theta \\ d &= \sin\psi\sin\phi\sin\theta \end{aligned} \tag{17.4}$$

where the angles are, in turn, within the following range to fully cover the 3-sphere:

$$\begin{aligned} &0 < \psi < \frac{\pi}{2} \\ &0 < \phi < \pi \\ &0 < \theta < 2\pi \end{aligned} \tag{17.5}$$

where the range of ψ is limited to $\pi/2$ from the original π, since only half of the $\mathbb{S}^3$ is needed to represent the orientations. Furthermore, $<$ replaces $\leq$ to enable the use of differential geometry. The added benefit of adopting an open set is that the singular positions of the coordinate systems at $\psi = 0$, π, and $\phi = 0$, π are now excluded.

The metric for volume measurement is a tensor [2, 5], ${}^{Q}\mathbf{g}$, given by

$${}^{Q}\mathbf{g} = \begin{bmatrix} \left(\frac{\partial \mathbf{q}_Q}{\partial\psi}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\psi} & \left(\frac{\partial \mathbf{q}_Q}{\partial\psi}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\phi} & \left(\frac{\partial \mathbf{q}_Q}{\partial\psi}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\theta} \\ \left(\frac{\partial \mathbf{q}_Q}{\partial\phi}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\psi} & \left(\frac{\partial \mathbf{q}_Q}{\partial\phi}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\phi} & \left(\frac{\partial \mathbf{q}_Q}{\partial\phi}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\theta} \\ \left(\frac{\partial \mathbf{q}_Q}{\partial\theta}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\psi} & \left(\frac{\partial \mathbf{q}_Q}{\partial\theta}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\phi} & \left(\frac{\partial \mathbf{q}_Q}{\partial\theta}\right)^T \frac{\partial \mathbf{q}_Q}{\partial\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sin^2\psi & 0 \\ 0 & 0 & \sin^2\psi\sin^2\phi \end{bmatrix} \tag{17.6}$$

and the corresponding weighted volume of a subset $\mathbf{U}$ of $\mathbb{S}^3$ is defined as

$${}^{Q}V = \int_U \sqrt{\det\left({}^{Q}\mathbf{g}\right)}d\psi d\phi d\theta = \int_U \sin^2\psi\sin\phi d\psi d\phi d\theta \tag{17.7}$$

with the ranges of angles given in Equation (17.5). Note that we term $\sqrt{\det\left({}^{Q}\mathbf{g}\right)}d\psi d\phi d\theta$ the volume element [2, 3] hereafter.

17.2 Generalised Volume of Orientation Workspace

The orientation workspace volume defined in Equation (17.7) can be generalised to suit other kinematic descriptions of a spherical motion generator. In this sub-section, we demonstrate that the parameterised volume derived from a generalised rotation matrix $\mathbf{R}$ is $1/16\sqrt{2}$ times of that derived from the quaternion $\mathbf{q}_Q$. To avoid confusion, we distinguish relevant terms, i.e., metric tensor g and the corresponding volume V, derived from a quaternion and a rotation matrix by superscripts Q and R, respectively.

Consider an arbitrary kinematic representation $\mathbf{q} = \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^T$. The entries of the metric tensors ${}^{Q}\mathbf{g}$ and ${}^{R}\mathbf{g}$ are given, correspondingly, by

$${}^{Q}\mathbf{g} = \begin{bmatrix} \left(\frac{\partial \mathbf{q}_Q}{\partial q_1}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_1} & \left(\frac{\partial \mathbf{q}_Q}{\partial q_1}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_2} & \left(\frac{\partial \mathbf{q}_Q}{\partial q_1}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_3} \\ \left(\frac{\partial \mathbf{q}_Q}{\partial q_2}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_1} & \left(\frac{\partial \mathbf{q}_Q}{\partial q_2}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_2} & \left(\frac{\partial \mathbf{q}_Q}{\partial q_2}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_3} \\ \left(\frac{\partial \mathbf{q}_Q}{\partial q_3}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_1} & \left(\frac{\partial \mathbf{q}_Q}{\partial q_3}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_2} & \left(\frac{\partial \mathbf{q}_Q}{\partial q_3}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_3} \end{bmatrix} \tag{17.8}$$

and

$$
{}^{R}\mathbf{g} = \begin{bmatrix} \left(\frac{\partial\,\mathbf{r}}{\partial q_1}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_1} & \left(\frac{\partial\,\mathbf{r}}{\partial q_1}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_2} & \left(\frac{\partial\,\mathbf{r}}{\partial q_1}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_3} \\ \left(\frac{\partial\,\mathbf{r}}{\partial q_2}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_1} & \left(\frac{\partial\,\mathbf{r}}{\partial q_2}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_2} & \left(\frac{\partial\,\mathbf{r}}{\partial q_2}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_3} \\ \left(\frac{\partial\,\mathbf{r}}{\partial q_3}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_1} & \left(\frac{\partial\,\mathbf{r}}{\partial q_3}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_2} & \left(\frac{\partial\,\mathbf{r}}{\partial q_3}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_3} \end{bmatrix} \tag{17.9}
$$

with $\mathbf{r}$ collecting the entries of rotation matrix $\mathbf{R}$

$$
\mathbf{r} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{21} & r_{22} & r_{23} & r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{17.10}
$$

the two digits of the subscripts identifying the rows and columns of the scalar entries, respectively. We'll show in the balance of the sub-section the following ratio of the volume elements:

$$
{}^{R}\mathbf{g} = 8\,{}^{Q}\mathbf{g} \tag{17.11}
$$

which further leads to the $1/16\sqrt{2}$ ratio of interest for the orientation workspace volume.

In Equation (17.9), scalar entry ${}^{R}g_{ij}$ can be expanded following the chain rule as

$$
{}^{R}g_{ij} = \left(\frac{\partial\,\mathbf{r}}{\partial q_i}\right)^T \frac{\partial\,\mathbf{r}}{\partial q_j} = \left(\frac{\partial \mathbf{q}_Q}{\partial q_i}\right)^T \left(\frac{\partial\,\mathbf{r}}{\partial \mathbf{q}_Q}\right)^T \frac{\partial\,\mathbf{r}}{\partial \mathbf{q}_Q}\frac{\partial \mathbf{q}_Q}{\partial q_j} \tag{17.12}
$$

The two terms in the middle, i.e., $\frac{\partial\,\mathbf{r}}{\partial \mathbf{q}_Q}$ and its transpose, in fact, satisfies

$$
\left(\frac{\partial\,\mathbf{r}}{\partial \mathbf{q}_Q}\right)^T \frac{\partial\,\mathbf{r}}{\partial \mathbf{q}_Q} = 8\,\mathbf{1} + 4\mathbf{q}_Q\mathbf{q}_Q^T \tag{17.13}
$$

$\mathbf{1}$ being the identity matrix. Upon substitution of (17.13), (17.12) becomes

$$
{}^{R}g_{ij} = 8\left(\frac{\partial \mathbf{q}_Q}{\partial q_i}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_j} + 4\left(\frac{\partial \mathbf{q}_Q}{\partial q_i}\right)^T \mathbf{q}_Q\mathbf{q}_Q^T\frac{\partial \mathbf{q}_Q}{\partial q_j} \tag{17.14}
$$

Additionally, we note that the partial derivatives of $\mathbf{q}_Q$ are essentially tangent vectors to the coordinate curves on their respective manifolds, rendering

$$
\mathbf{q}_Q^T\frac{\partial \mathbf{q}_Q}{\partial x_j} = 0 \tag{17.15}
$$

The latter term of summation of Equation (17.14) hence vanishes:

$$
{}^{R}g_{ij} = 8\left(\frac{\partial \mathbf{q}_Q}{\partial q_i}\right)^T \frac{\partial \mathbf{q}_Q}{\partial q_j} = 8\,{}^{Q}g_{ij} \tag{17.16}
$$

Equation (17.11) is thus proven. From there, the corresponding volume of the orientation workspace is calculated based on Equation (17.7) as

$$
\begin{aligned}
{}^{R}V &= \int_U \sqrt{\det({}^{R}\mathbf{g})}dq_1dq_2dq_3 \\
&= \int_U \sqrt{8^3\det({}^{Q}\mathbf{g})}dq_1dq_2dq_3 \\
&= 16\sqrt{2}\int_U \sqrt{\det({}^{Q}\mathbf{g})}dq_1dq_2dq_3 = 16\sqrt{2}\,{}^{Q}V
\end{aligned} \tag{17.17}
$$

The parameterisation of the orientation workspace volume based on an arbitrary kinematic description can thus be completed for comparison with others through the following steps:

1. Compute $\mathbf{r}$ according to Equation (17.10).
2. Structure ${}^{R}\mathbf{g}$, the metric tensor, by computing the dot products of the partial derivatives, according to Equation (17.6).
3. Apply the ratio of 8 based on the rotation-matrix tensor ${}^{R}\mathbf{g}$ to obtain the quaternion matrix tensor ${}^{Q}\mathbf{g}$, according to Equation (17.11).
4. An equivalent to the previous step is to compute the volume ${}^{R}V$ based on Equation (17.7), and apply the ratio of $16\sqrt{2}$ to obtain its quaternion counterpart ${}^{Q}V$.

17.3 Orientation Workspace Volume of Various Kinematic Descriptions

Given Equation (17.17) and the three-step procedure described previously, we can now show the derivation of orientation workspace volume based on different kinematic descriptions. Below, we will show the derivations of the Euler angles and the angle-axis representations used in this book.

17.3.1 Euler Angles

Step 1: Based on the rotation matrix under ZYX Euler angle, array $\mathbf{r}$ is defined following Equation (17.10) as

$$\mathbf{r} = \begin{bmatrix} \cos\alpha\cos\beta \\ \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma \\ \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta \\ \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma \\ \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta \\ \cos\beta\sin\gamma \\ \cos\beta\cos\gamma \end{bmatrix} \tag{17.18}$$

α, β, and γ denoting the rotation around z, y, and x, respectively.
Step 2: According to Equation (17.6), the expression of the metric tensor is:

$${}^{R}\mathbf{g} = \begin{bmatrix} 2 & 0 & -2\sin\beta \\ 0 & 2 & 0 \\ -2\sin\beta & 0 & 2 \end{bmatrix} \tag{17.19}$$

the resultant volume element is derived from Equation (17.7)

$$\sqrt{\det\left({}^{R}\mathbf{g}\right)} = 2\sqrt{2}\cos\beta d\alpha d\beta d\gamma \tag{17.20}$$

Step 3: From there, applying the coefficient of Equation (17.12) gives

$$\sqrt{\det\left({}^{Q}\mathbf{g}\right)} = \frac{1}{8}\cos\beta d\alpha d\beta d\gamma \tag{17.21}$$

Likewise, the orientation workspace volumes of other Euler angle representations can be obtained. For XYZ, XZY, YXZ, YZX, ZXY, and ZYX, the volume element is given by

$\frac{1}{8}\cos\beta d\alpha d\beta d\gamma$. For XYX, XZX, YXY, YZY, ZXZ, and ZYZ, the volume element is derived as $\frac{1}{8}\sin\beta d\alpha d\beta d\gamma$. The corresponding ranges of β for the two groups of Euler angles are $\left(-\frac{\pi}{2},\ \frac{\pi}{2}\right)$ and $(0,\ \pi)$, respectively.

17.3.2 Angle-Axis Representation

The four-element angle-axis representation can be expressed using the standard three-element spherical polar coordinates, where the axis of rotation becomes

$$q_A = \begin{bmatrix} q_{A1} \\ q_{A2} \\ q_{A3} \end{bmatrix} = \begin{bmatrix} \sin\psi_1\cos\psi_2 \\ \sin\psi_1\sin\psi_2 \\ \cos\psi_1 \end{bmatrix} \tag{17.22}$$

and the rotation angle $\theta = \psi_3$.

Step 1: The rotation matrix can be written according to Equation (23.8), and the resultant array $\mathbf{r}$ is given by

$$\mathbf{r} = \begin{bmatrix} \sin^2\psi_1\cos^2\psi_2\left(1-\cos\psi_3\right)+\cos\psi_3 \\ \sin^2\psi_2\cos\psi_2\sin\psi_2\left(1-\cos\psi_3\right)+\cos\psi_1\sin\psi_3 \\ \sin\psi_1\cos\psi_2\cos\psi_1\left(1-\cos\psi_3\right)-\sin\psi_1\sin\psi_2\sin\psi_3 \\ \sin^2\psi_1\cos\psi_2\sin\psi_2\left(1-\cos\psi_3\right)-\cos\psi_1\sin\psi_3 \\ \sin^2\psi_2\sin^2\psi_2\left(1-\cos\psi_3\right)+\cos\psi_3 \\ \sin\psi_1\sin\psi_2\cos\psi_1\left(1-\cos\psi_3\right)+\sin\psi_1\cos\psi_2\sin\psi_3 \\ \sin\psi_1\cos\psi_2\cos\psi_1\left(1-\cos\psi_3\right)+\sin\psi_1\sin\psi_2\sin\psi_3 \\ \sin\psi_1\sin\psi_2\cos\psi_1\left(1-\cos\psi_3\right)-\sin\psi_1\cos\psi_2\sin\psi_3 \\ \cos^2\psi_1\left(1-\cos\psi_3\right)+\cos\psi_3 \end{bmatrix} \tag{17.23}$$

Step 2: Based on Equation (17.6), the expression of the metric tensor is:

$$^{R}\mathbf{g} = \begin{bmatrix} \sin^2\frac{\psi_3}{2} & 0 & 0 \\ 0 & \sin^2\psi_1\sin^2\frac{\psi_3}{2} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} \tag{17.24}$$

and the corresponding volume element is

$$\sqrt{\det\left({}^{R}\mathbf{g}\right)}d\psi_1 d\psi_2 d\psi_3 = \frac{1}{2}\sin\psi_1\sin^2\left(\frac{\psi_3}{2}\right)d\psi_1 d\psi_2 d\psi_3 \tag{17.25}$$

Step 3: Applying the ratio of Equation (17.12) leads to

$$\sqrt{\det\left({}^{Q}\mathbf{g}\right)}d\psi_1 d\psi_2 d\psi_3 = \frac{1}{16}\sin\psi_1\sin^2\left(\frac{\psi_3}{2}\right)d\psi_1 d\psi_2 d\psi_3 \tag{17.26}$$

17.4 Case Study

In this section, we present the orientation workspace analysis of a spherical manipulator (Figure 17.1) [1]. With two different ground frames, {0} and {1}, and different parameterisation methods, we demonstrate that the weighted orientation workspace volume is an invariant as per required by the two principles. Additionally, the results are compared to those derived from the conventional methods for the demonstration of advantages.

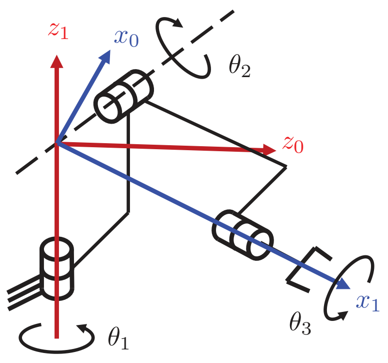


FIGURE 17.1
Schematic diagram of the spherical manipulator.

The spherical manipulator has three joints, identified by θ_1, θ_2, and θ_3, respectively. Their ranges are, correspondingly,

$$\begin{aligned} 0 \le \theta_1 &\le \frac{\pi}{6} \\ -\frac{\pi}{12} \le \theta_2 &\le \frac{\pi}{12} \\ 0 \le \theta_3 &\le \frac{\pi}{6} \end{aligned}$$

To demonstrate that the weight volume is invariant with respect to the ground frame, we consider two ground frames: the genuine ground frame $\{1\}$, and an artificially defined $\{0\}$ that is related to $\{1\}$ via ZYX Euler angles:

$$^0\mathbf{R}_1 = \begin{bmatrix} \cos\alpha'\cos\beta' & \cos\alpha'\sin\beta'\sin\gamma' - \sin\alpha'\cos\gamma' & \cos\alpha'\sin\beta'\cos\gamma' + \sin\alpha'\sin\gamma' \\ \sin\alpha'\cos\beta' & \sin\alpha'\sin\beta'\sin\gamma' + \cos\alpha'\cos\gamma' & \sin\alpha'\sin\beta'\cos\gamma' - \cos\alpha'\sin\gamma' \\ -\sin\beta' & \cos\beta'\sin\gamma' & \cos\beta'\sin\gamma' \end{bmatrix}$$

with $\alpha' = \frac{5\pi}{6}$, $\beta' = -\frac{\pi}{3}$, and $\gamma' = \frac{\pi}{4}$.

For the purpose of comparison, we also conducted the workspace space analysis based on the conventional method, which comprises the following steps:

1. Identify a kinematic description for the parameterisation of the orientational workspace.
2. Create an equispaced grid in the space of the corresponding parameters.
3. Identify the grid points that are within the workspace, taking into account the range, singularities, and mechanical interference.
4. Integrate those grip points within the workspace, then multiply their corresponding volume elements to compute the workspace volume.

Table 17.1 summarises the comparison between the orientational workspace volume determined by the conventional method and our proposed method based on the 3-sphere, and

TABLE 17.1
Comparison of workspace volumes.

Convention	Frame	Direct volume	Weighted volume
Quaternions	{1}	0.0182	0.0176
	{0}	0.0536	0.0176
ZYX Euler	{1}	0.1435	0.0177
	{0}	0.4179	0.0178
XZX Euler	{1}	0.6653	0.0177
	{0}	0.1553	0.0178
Angle-axis	{1}	2.4875	0.0178
	{0}	0.0198	0.0177

with {0} and {1} under different kinematic descriptions. The workspace volume computed through the two methods are labelled "direct volume" and "weighted volume", respectively.

It can be readily observed that the approach based on weighted volume yielded consistent measurement that is invariant with respect to the kinematic description and the frames, satisfying the two measurement principles; the small errors are the result of numerical computation. In contrast, the conventional method yielded significant variation in the measurement, both across different kinematic descriptions and between the two frames within the same parameterisation method.

17.5 Conclusion

The method presented in this chapter complements those in Chapter 16, to provide a uniform and cross-comparable way to evaluate the orientation workspace of mechanisms. The method takes advantage of the physical interpretation of quaternion for being a point on the surface of a 4D sphere to quantify the orientation workspace of a manipulator. To promote the compatibility of this method, we derived the mapping between other commonly used orientation parameterisation methods and quaternion, which significantly widens the application of the quaternion-based analysis.

Bibliography

[1] C. Chen and D. Jackson, "Parameterisation and evaluation of robotic orientation workspace: A geometric treatment," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 656–663, 2011.

[2] M. P. Do Carmo and J. Flaherty Francis, *Riemannian geometry*, vol. 6. Springer, 1992.

[3] J. Jost, *Riemannian geometry and geometric analysis*, vol. 42005. Springer.

[4] G. Yang, "A differential geometry approach for the workspace analysis of spherical parallel manipulators," *Proc. 11th World Cong. in Mechanism and Machine Science, Tianjin, 2004*, pp. 2060–2065, 2004.

[5] J. Angeles, *Fundamentals of robotic mechanical systems: theory, methods, and algorithms.* Springer, 2003.

[6] I. A. Bonev and C. M. Gosselin, "Analytical determination of the workspace of symmetrical spherical parallel mechanisms," *IEEE Transactions on Robotics*, vol. 22, no. 5, pp. 1011–1017, 2006.

[7] J. S. Dai, "An historical review of the theoretical development of rigid body displacements from rodrigues parameters to the finite twist," *Mechanism and Machine Theory*, vol. 41, no. 1, pp. 41–52, 2006.

[8] A. Cayley, "Sur quelques propriétés des déterminants gauches.," 1846.

[9] G. S. Chirikjian and A. B. Kyatkin, *Engineering applications of noncommutative harmonic analysis: with emphasis on rotation and motion groups.* CRC press, 2000.

[10] G. Yang, W. Lin, S. K. Mustafa, I.-m. Chen, and S. H. Yeo, "Numerical orientation workspace analysis with different parameterisation methods," in *2006 IEEE Conference on Robotics, Automation and Mechatronics*, pp. 1–6, IEEE, 2006.

[11] F. C. Park and B. Ravani, "Smooth invariant interpolation of rotations," *ACM Transactions on Graphics (TOG)*, vol. 16, no. 3, pp. 277–295, 1997.

18

Constraint Analysis for Underactuated Systems

In the previous parts, we limited the scope of our discussion to fully-actuated, rigid-link robotic manipulators. On the other hand, soft robots are increasingly being used in applications to replace their rigid-link counterparts. These robots are favoured for their inherent safety — the compliance allows them to deform upon contact with obstacles. Soft robots are a typical example of underactuated systems whose mechanical degrees of freedom (DoF) exceed the number of actuator inputs. Underactuated robotic grippers are another example of systems of this kind. Compared to their fully-actuated counterparts, underactuated grippers conform to the shape of the target object, rendering improved grasping capability.

Despite their favourable features, modelling the underactuated systems is a complex process. In static or quasi-static studies, the configuration of an underactuated system, as well as the internal and actuator input forces and moments, are determined by *minimum energy theory*. In addition, both the kinematic constraints and the force and moment equilibrium ("force constraints" hereafter) must be satisfied. Correspondingly, multibody systems can be analysed by solving a constrained minimisation problem. The objective function is related to the potential energy of the system, while the *kinetostatic constraints* are derived in both the kinematic and the force domains. Such derivation, involving two domains, is inherently tedious. Worse, the constraints can vary depending on the system's status. For example, as an underactuated gripper grasps the objects, the contacting finger segments can differ case by case with respect to the size of the object and the actuator inputs, leading to the necessity to re-derive the constraint equations corresponding to each contact case.

Constraint analysis with Lagrange multipliers can be exploited to tackle the issue and significantly simplify the modelling of underactuated systems. This approach takes advantage of the one-to-one correspondence among 1) a mechanical constraint of interest, 2) the kinematic constraint equation derived from said mechanical constraint, expressed in a unified format, 3) the Lagrange multiplier resulting from the kinematic constraint equation, and 4) the force and moment transmitted through the mechanical constraint. The force constraints can now be generated automatically from their kinematic counterparts. As such, only the kinematic-domain derivation is required, and the force-domain derivation is obviated. In addition, selection matrices can be used to identify the active kinematic constraint equations and Lagrange multipliers. In this case, a *master* set of kinetostatic constraints with kinematic constraints active can be expressed, and the re-derivation for individual cases is reduced into the derivation of corresponding selection matrices.

In this chapter, we present the modelling of planar underactuated systems based on the framework of constraint analysis, along with two case studies featuring a tendon-driven soft robot [2] and an underactuated prosthetic finger [10], respectively.

DOI: 10.1201/9781003614319-18

18.1 Constrained Minimisation

Assuming massless bodies, i.e., no gravitational potential energy, the constrained minimisation problem is defined as [1]:

$$\begin{aligned} \text{min} \quad & V(\mathbf{q}_u) \\ \text{subject to} \quad & \boldsymbol{\phi}_1(\mathbf{q}_u) = \mathbf{0} \\ & \boldsymbol{\phi}_2(\mathbf{q}_u) \leq \mathbf{0} \end{aligned} \tag{18.1}$$

$V(\mathbf{q}_u)$ denoting the objective function of the potential energy expressed in the unknowns $\mathbf{q}_u$ of the system, whose general-form expression is given by:

$$V(\mathbf{q}_u) = \sum_{v=1}^{v=v_{max}} \frac{1}{2} k_v \delta q_v{}^2 \tag{18.2}$$

subscript v identifying the compliant bodies with stiffness k_v and deformation δq_v. The constraint minimisation problem can be solved through software packages, e.g., MATLAB *fmincon*.

In Equation (18.1), $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ are the kinetostatic constraint equations and the constraint inequalities, respectively. In the ensuing subsection, we will discuss the derivation of these constraints based on the framework of planar constraint analysis.

18.2 Kinetostatic Constraints

18.2.1 Generalised Coordinates and Generalised External Forces

In the planar case, the configuration of a rigid body i can be fully described by three independent generalised coordinates: x_i and y_i representing the position, and θ_i or $\theta_{(i-1)i}$ for the orientation, where θ_i is the absolute angular position and $\theta_{(i-1)i}$ denotes the angular position of i with respect to the previous body $i-1$. Array $\mathbf{q}$ containing all the generalised coordinates shall fully represent the configuration of a system:

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_1^T & \mathbf{q}_2^T & \cdots & \mathbf{q}_{i_{max}}^T \end{bmatrix}^T \tag{18.3}$$

$\mathbf{q}_i$ ($i \in [1, i_{max}]$) identifying the sub-array collecting the generalised coordinates of body i.

Furthermore, in some cases, it is possible to neglect certain generalised coordinates to simplify the formulation. Once the free-floating bodies are assembled into kinematic chains, most of the generalised coordinates are dependent on the input of the system, and those relevant to mechanical constraints that are beyond the interest of the analysis may be expressed implicitly. We will demonstrate this type of simplification in the case study of the tendon-driven soft robot.

It should be noted that the *rigid body* herein, in fact, covers three types of bodies: the generic rigid bodies, the pseudo-rigid bodies, and the virtual bodies. Pseudo-rigid bodies [2] are used to model compliant continuum bodies, e.g., the backbone of a soft robot or a compliant link of an underactuated gripper. In these cases, a continuum body is treated as a sub-kinematic chain comprising multiple pseudo-rigid bodies and spring-loaded virtual joints. Virtual bodies can be regarded as those rigidly attached to prescribed locations along the kinematic chain, where an external force or moment is applied to a body without having an already-assigned generalised coordinate. Virtual bodies are introduced since $\mathbf{q}$ the generalised coordinates and $\mathbf{f}_e$ the generalised external forces, i.e., forces and moments, must be in a one-to-one correspondence:

$$\mathbf{f}_e = \begin{bmatrix} \mathbf{f}_{e1}^T & \mathbf{f}_{e2}^T & \cdots & \mathbf{f}_{ei_{max}}^T \end{bmatrix}^T \tag{18.4}$$

where positional and orientational q_i of (18.3) have matching scalar entries of forces and moments in $\mathbf{f}_e$, respectively, for the external forces and moments applied at the tip or along the kinematic chain of a system, as well as the actuator input forces and moments. For convenience, we refer to *moment* as the external or constraining moment and *torque* as the actuator input.

For underactuated systems, $\mathbf{f}_e$ also contains scalar entries for the forces and moments generated by the spring-loaded virtual joints of the pseudo-rigid bodies. Despite being internal in the physical system, these forces and moments can be interpreted as virtual actuator input forces and torques, whose magnitudes correspond to the displacements of the joints. The relationship is described through additional Hooke constitutive equations, discussed in the later subsection. Assuming prescribed external forces and moments, the unknown actuator input forces and torques of $\mathbf{f}_e$ form array $\mathbf{f}_{eu}$ of the unknown generalised external forces:

$$\mathbf{f}_{eu} = \begin{bmatrix} f_{ea1} & f_{ea2} & \cdots & f_{ea_{max}} & f_{ev1} & f_{ev2} & \cdots & f_{ev_{max}} \end{bmatrix}^T \tag{18.5}$$

subscripts a and v distinguishing the physical and virtual actuators, respectively.

18.2.2 General-Form Kinematic Constraint Equations and Lagrange Multipliers

In the framework of constraint analysis, kinematic constraint equations are expressed in a specific form to achieve the desired one-to-one correspondence between the Lagrange multipliers and the forces and moments transmitted through mechanical constraints. The general-form kinematic constraint equation $\boldsymbol{\phi}_k\left(\mathbf{q}_a, \mathbf{q}_b\right)$ between two connected bodies, a and b, is written as [3, 4, 5]:

$$\boldsymbol{\phi}_k\left(\mathbf{q}_a, \mathbf{q}_b\right) = \mathbf{p}\left(\mathbf{q}_a\right) - \mathbf{p}\left(\mathbf{q}_b\right) = \mathbf{0} \tag{18.6}$$

$\mathbf{p}$ representing the shared physical quantity of the two bodies, e.g., the position of a joint, and expressed using the generalised coordinates of the two bodies, respectively. It is noteworthy that (18.6) is directional, i.e., the resultant forces and moments are applied by body a on body b. The graphical representation of (18.6) is depicted in Figure 18.1. In the ensuing case studies, we discuss the kinematic constraint equations derived from position and

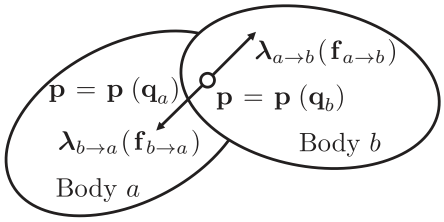

$\boldsymbol{\lambda}_{a\to b}(\mathbf{f}_{a\to b})$
$\mathbf{p} = \mathbf{p}\left(\mathbf{q}_a\right)$
$\mathbf{p} = \mathbf{p}\left(\mathbf{q}_b\right)$
$\boldsymbol{\lambda}_{b\to a}(\mathbf{f}_{b\to a})$
Body b
Body a

FIGURE 18.1
Graphical representation of constraint equation.

orientation – revolute joints; “fixed” joints of virtual bodies, attachment points of actuation tendons, contact points between a gripper link and the external object, and gear meshing.

The collection of kinematic constraint equations of a system is represented by array $\boldsymbol{\phi}_k$:

$$\boldsymbol{\phi}_k = \begin{bmatrix} \boldsymbol{\phi}_{k1}^T & \boldsymbol{\phi}_{k2}^T & \cdots & \boldsymbol{\phi}_{kj_{max}}^T \end{bmatrix}^T \tag{18.7}$$

$\boldsymbol{\phi}_{kj}$ ($j \in [1, j_{max}]$) identifies the sub-array of scalar kinematic constraint equations of the j-th mechanical constraint. The corresponding forces and moments are collected in array $\boldsymbol{\lambda}$ of the Lagrange multipliers:

$$\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_1^T & \boldsymbol{\lambda}_2^T & \cdots & \boldsymbol{\lambda}_{j_{max}}^T \end{bmatrix}^T \tag{18.8}$$

18.2.3 Force Constraint Equations

For a constrained system satisfying $\boldsymbol{\phi}_k(\mathbf{q}) = \mathbf{0}$, the governing force constraint equation is given by [6]:

$$\mathbf{f}_i = \mathbf{f}_c + \mathbf{f}_e \tag{18.9}$$

where $\mathbf{f}_i$, $\mathbf{f}_c$, and $\mathbf{f}_e$ are the generalised inertia forces, generalised constraint forces, and generalised external forces, respectively. Among them, $\mathbf{f}_i$ vanishes under the static/quasi-static assumption, and $\mathbf{f}_e$ is defined in (18.4). Furthermore, $\mathbf{f}_c$ can be related to $\mathbf{q}$, $\boldsymbol{\phi}_k$, and $\boldsymbol{\lambda}$ of (18.3), (18.7), and (18.8), respectively, as

$$\mathbf{f}_c = -\left(\frac{\partial\, \boldsymbol{\phi}_k}{\partial \mathbf{q}}\right)^T \boldsymbol{\lambda} \tag{18.10}$$

Equation (18.9) thus becomes:

$$\left(\frac{\partial\, \boldsymbol{\phi}_k}{\partial \mathbf{q}}\right)^T \boldsymbol{\lambda} - \mathbf{f}_e = \mathbf{0} \tag{18.11}$$

which is the force constraint of equilibrium of the system.

18.2.4 Hooke Constitutive Equations

As discussed previously, the Hooke constitutive equation relates the displacement of a pseudo-rigid joint and the input force or torque of this virtual actuator. Under the linear assumption, the relationship is defined as

$$\phi_{sv} = f_{ev} + k_v \delta q_v = 0 \tag{18.12}$$

f_{ev}, k_v, and δq_v identifying the input force or torque of joint v, the stiffness, and the deformation/joint displacement, respectively. Among them, f_{ev} is a scalar entry of (18.4). In addition, k_v and δq_v are also used to write (18.2), and δq_v is expressed using $\mathbf{q}$ of (18.3). The scalar Hooke constitutive equations ϕ_{sv} are collected in array $\boldsymbol{\phi}_s$ to form part of the kinetostatic constraints.

$$\boldsymbol{\phi}_s = \begin{bmatrix} \boldsymbol{\phi}_{s1}^T & \boldsymbol{\phi}_{s2}^T & \cdots & \boldsymbol{\phi}_{sv_{max}}^T \end{bmatrix}^T \tag{18.13}$$

18.2.5 Selection Matrices

In case some kinematic constraints of the system are inactive, e.g., the finger-object contact constraints of non-contacting phalanges in the analysis of underactuated grippers, relevant unknowns and constraints shall be removed from the master kinetostatic constraints. Such a process can be done efficiently through selection matrices, which identify relevant scalar entries from arrays of variables or constraint equations and inequalities. As such, the re-derivation of the kinetostatic constraints is reduced to the derivation of selection matrices, and the modelling process is simplified.

For an arbitrary array $\mathbf{q} \in \mathbb{R}^n$, its full-selection matrix $\mathbf{S}_q \in \mathbb{R}^{n\times n}$ is the identity matrix. Each column $\mathbf{s}_i$ of $\mathbf{S}_q$ is in a one-to-one correspondence with a scalar entry q_i of $\mathbf{q}$. Therefore, selection matrix $\mathbf{S}'_q \in \mathbb{R}^{n\times n'}$, containing n' columns, can be used to extract an n'-entry subset $\{\mathbf{q}'\} \subseteq \{\mathbf{q}\}$ with the corresponding scalar entries [1]:

$$\mathbf{q}' = \mathbf{S}'^{\,T}_q \mathbf{q} \tag{18.14}$$

where the columns of $\mathbf{S}'_q$ belong to a set $\{s'_q\}$ that satisfies

$$\begin{aligned} \{\mathbf{s}_q{}'\} \cup \{\mathbf{s}_q{}^*\} &= \{\mathbf{s}_q\} \\ \{\mathbf{s}_q{}'\} \cap \{\mathbf{s}_q{}^*\} &= \emptyset \end{aligned}$$

$\{\mathbf{s}_q\}$ is a subset collecting all $\mathbf{s}_i$ of $\mathbf{S}_q$, and entries of $\{\mathbf{s}_q{}^*\}$ correspond to $\{\mathbf{q}^*\}$, which is, in turn, defined as:

$$\begin{aligned} \{\mathbf{q}'\} \cup \{\mathbf{q}^*\} &= \{\mathbf{q}\} \\ \{\mathbf{q}'\} \cap \{\mathbf{q}^*\} &= \emptyset \end{aligned}$$

In our case studies, we derive $\mathbf{S}'_q$ by means of identifying and removing entries of $\{\mathbf{q}^*\}$. The use of selection matrices is detailed in the ensuing subsection.

18.2.6 Array of Unknowns

Array $\mathbf{q}_u$ of the unknown variables in the constrained minimisation problem (18.1) is defined as [1]:

$$\mathbf{q}_u = \left[\left(\mathbf{S}'^{\,T}_q \mathbf{q}\right)^T \quad \left(\mathbf{S}'^{\,T}_r \mathbf{q}_r\right)^T \quad \left(\mathbf{S}'^{\,T}_{eu}\, \mathbf{f}_{eu}\right)^T \quad \left(\mathbf{S}'^{\,T}_\lambda\, \boldsymbol{\lambda}\right)^T \right]^T \tag{18.15}$$

where array $\mathbf{q}$ of the generalised coordinates is presented in (18.3), array $\mathbf{f}_{eu}$ of the unknown generalised external forces is defined in (18.5), and array $\boldsymbol{\lambda}$ of the Lagrange multipliers is given in (18.8). In addition, $\mathbf{q}_r$ incorporates the unknown variables not already included in $\mathbf{q}$, e.g., those of the contact geometry. Selection matrices $\mathbf{S}'$ of the corresponding arrays identify the relevant active variables. Notably, a selection matrix is applied to $\mathbf{q}$ since some of the generalised coordinates can be prescribed, e.g., the actuator input displacement in the forward kinetostatic problem and the end-effector position and orientation in the inverse kinetostatic problem.

18.2.7 Kinetostatic Constraints

The equation part of the master kinetostatic constraints, i.e., $\boldsymbol{\phi}_1$ of (18.1), contains $\boldsymbol{\phi}_k$ of (18.7), $\boldsymbol{\phi}_f$ of (18.11), and $\boldsymbol{\phi}_s$ of (18.12). Factoring in the selection matrices, $\boldsymbol{\phi}_1$ is given by [1]:

$$\boldsymbol{\phi}_1 = \left[\left(\mathbf{S}'^{\,T}_\lambda\, \boldsymbol{\phi}_k\right)^T \quad \left(\left(\mathbf{S}'^{\,T}_\lambda \left(\frac{\partial\, \boldsymbol{\phi}_k}{\partial \mathbf{q}}\right)\right)^T \left(\mathbf{S}'^{\,T}_\lambda\, \boldsymbol{\lambda}\right) - \mathbf{f}_e \right)^T \quad \left(\mathbf{S}'^{\,T}_s\, \boldsymbol{\phi}_s\right)^T \right]^T = \mathbf{0} \tag{18.16}$$

where $\boldsymbol{\phi}_k$ and $\boldsymbol{\phi}_f$ both adopt $\mathbf{S}'_\lambda$ as the selection matrix, since $\boldsymbol{\phi}_k$ and $\boldsymbol{\lambda}$ are in a one-to-one correspondence.

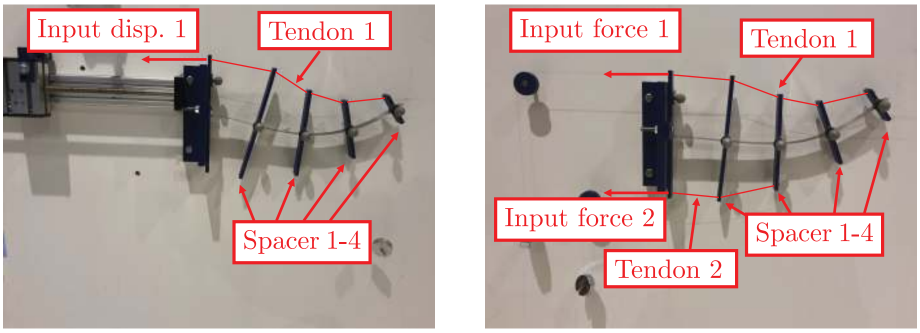


FIGURE 18.2
Tendon-driven soft robot under prescribed input displacement (left) and forces (right).

Similarly, array $\boldsymbol{\phi}_2$ of the inequalities of the master kinetostatic constraints is expressed as [1]:

$$\boldsymbol{\phi}_2 = \mathbf{S}_2'^{\,T}\, \boldsymbol{\phi}_2{}^* \leq \mathbf{0} \tag{18.17}$$

$\boldsymbol{\phi}_2{}^*$ denoting the full set of kinetostatic constraint inequalities. Examples of constraint inequalities include positive tendon force constraint for the actuation tendon of tendon-driven soft robots, and positive normal contact force constraint for finger-object contacts of underactuated grippers, which are relevant to the ensuing case studies.

18.3 Case Study — Soft Robots with Pseudo-Rigid Bodies

This case study analyses a planar tendon-driven soft robot [2]. In its most simplified form, a robot of this kind comprises a continuous compliant backbone and two actuation tendons routed through spacers that are, in turn, rigidly attached to the backbone. We aim to showcase with this case study the use of the pseudo-rigid body approach, where a continuum section is modelled as four pseudo-rigid links connected by three spring-loaded revolute joints (the 3R-PRB method [7]).

A prototype of the robot of interest is illustrated in Figure 18.2. This robot comprises two segments, each comprising two sub-segments, and is actuated by two tendons. The tendons are, in turn, connected to the distal and middle spacers, respectively, and routed through their corresponding proximal spacers. Additionally, the backbone of the robot is a piece of thin spring steel sheet of a rectangular cross-sectional shape. The design parameters of the robot are listed as follows.
Note that w the widths of the spacers are measured from the backbone to the tendon pass-through or attachment points.

On a two DoF robot, two actuator inputs or tip position/orientation parameters can be prescribed, leading to two cases in the forward kinetostatic problem and three in the inverse one:

- Forward problem 1 — prescribe actuator input displacements.
- Forward problem 2 — prescribe actuator input forces.
- Inverse problem 1 — prescribe tip x and y position.

- Inverse problem 2 — prescribe tip x position and tip orientation.
- Inverse problem 3 — prescribe tip y position and tip orientation.

This case study focuses on the forward kinetostatic problem with a prescribed input force. It should be noted that the unknowns and kinetostatic constraints of all five cases can be related to a master set of unknowns and constraints by means of pertinent selection matrices. However, we choose to direct our attention to the 3R-PRB approach herein, and selection matrices are detailed in the next case study.

18.3.1 3R-PRB Model of Backbone Sub-Segments

Each sub-segment of the continuous backbone is modelled via the 3R-PRB approach [7] as four pseudo-rigid links connected by three spring-loaded virtual revolute joints. The proximal and distal pseudo-rigid links are rigidly connected to the spacers that are, in turn, attached to the previous and the next sub-segment, respectively.

Within a sub-segment of length l_o, the lengths of the pseudo-rigid links are $0.125l_o$, $0.35l_o$, $0.388l_o$, and $0.136l_o$, respectively, from the proximal to the distal end, where the coefficients are defined in the 3R-PRB model. The stiffness of the proximal, middle, and distal spring-loaded virtual revolute joints are, correspondingly, $3.25EI/l_o$, $2.84EI/l_o$, and $2.95EI/l_o$, where I is the second moment of area of the cross-section calculated from design parameters h and b.

18.3.2 Generalised Coordinates and Generalised External Forces

According to (18.3), four types of bodies are incorporated to derive array $\mathbf{q}$ of the generalised coordinates:

- The rigid links of spacer disks.
- The rigid links to model tendon.
- The pseudo-rigid bodies of the continuum backbone, from the 3R-PRB model.
- Scalar variables to describe the actuator input displacements.

The inter-spacer tendon sections are treated as rigid bodies whose lengths can vary. They can be interpreted as sub-kinematic chains with embedded prismatic joints. Additional geometrical parameters are hence needed, representing the lengths of inter-space tendon sections and tendon elongation. The latter belong to $\mathbf{q}_r$ of (18.15). Kinematic constraint equations of overall tendon length and inequalities for zero/positive internal force, i.e., tension, are also required to make physical-meaningful tendons. Array $\mathbf{q}$ of the generalised coordinates is given by:

$$\mathbf{q} = \begin{bmatrix} d_1 & d_2 & \mathbf{q}_{b1}^T & \mathbf{q}_{b2}^T & \mathbf{q}_{b3}^T & \mathbf{q}_{b4}^T & \mathbf{q}_{t11}^T & \mathbf{q}_{t12}^T & \mathbf{q}_{t13}^T & \mathbf{q}_{t14}^T & \mathbf{q}_{t21}^T & \mathbf{q}_{t22}^T \end{bmatrix}^T \tag{18.18}$$

where d_1 and d_2 are the actuator input displacements of the actuation tendons. In addition, $\mathbf{q}_{bi}$ ($i \in [1,4]$) denoting the sub-array of generalised coordinates of backbone sub-segment i

$$\mathbf{q}_{bi} = \begin{bmatrix} \theta_{i1} & \theta_{i2} & \theta_{i3} & x_{i4} & y_{i4} & \theta_{i4} \end{bmatrix}^T$$

subscripts 1 to 4 correspond to the four pseudo-rigid bodies, respectively; among them, 4 is rigidly connected to and hence presents the configuration of the spacer plate attached to the distal end of the sub-segment. It is noteworthy that θ_{i1}, θ_{i2}, and θ_{i3} are joint displacements with respect to the previous pseudo-rigid link, while θ_{i4} is absolute orientation. The former is expressed in a relative manner for the convenience of writing the Hooke constitutive equations. Moreover, the positional generalised coordinates for the first three pseudo-rigid bodies are omitted since they can be embedded implicitly in the kinematic constraints, as shown in the ensuing sub-section. Similarly, sub-array $\mathbf{q}_{tai}$ ($a \in [1, 2]$) collects the generalised coordinates of the section of actuation tendon a at backbone sub-segment i

$$\mathbf{q}_{tai} = \begin{bmatrix}\theta_{tai} & d_{tai}\end{bmatrix}^T$$

θ_{tai} and d_{tai} indicating the orientation and the length of the tendon section ai, respectively. Again, positional generalised coordinates of the tendon sections are omitted.

Array $\mathbf{f}_e$ corresponding to the (18.18) is defined as:

$$\mathbf{f}_e = \begin{bmatrix}f_{e1} & f_{e2} & \mathbf{f}_{eb1}^T & \mathbf{f}_{eb2}^T & \mathbf{f}_{eb3}^T & \mathbf{f}_{eb4}^T & \mathbf{0}^T\end{bmatrix}^T \tag{18.19}$$

where f_{e1} and f_{e2} are the actuator input forces applied on the tendons. Non-zero sub-arrays $\mathbf{f}_{ebi}$ and the zero sub-array correspond to $\mathbf{q}_{bi}$ and $\mathbf{q}_{tai}$, respectively. The former is defined as

$$\mathbf{f}_{ebi} = \begin{bmatrix}\tau_{ei1} & \tau_{ei2} & \tau_{ei3} & f_{ei4x} & f_{ei4y} & 0\end{bmatrix}^T$$

the torques of the spring-loaded virtual revolute joints are represented by τ_{ei1}, τ_{ei2}, and τ_{ei3}. The prescribed external forces applied onto the spacer plates are incorporated through f_{ei4x} and f_{ei4y}, while the external moment is assumed zero. Notice that f_{e44x} and f_{e44y} are the tip force of the robot. The virtual actuator torques form array $\mathbf{f}_{eu}$ of the unknown generalised external forces:

$$\mathbf{f}_{eu} = \begin{bmatrix}\tau_{e11} & \tau_{e12} & \tau_{e13} & \cdots & \tau_{e41} & \tau_{e42} & \tau_{e43}\end{bmatrix}^T \tag{18.20}$$

18.3.3 Kinematic Constraint Equations and Lagrange Multipliers

We consider three types of kinematic constraints: those derived from the fixed joints between the spacer plates and sub-segments of the backbone, those describing the spacer-tendon connections, and those based on the overall length of the actuation tendon. We also demonstrate the implicit positional kinematic constraints.

The kinematic constraint of a fixed joint of spacer i is expressed as:

$$\boldsymbol{\phi}_{kpi} = \begin{bmatrix}x_{(i-1)4}\\ y_{(i-1)4}\\ \theta_{(i-1)4}\end{bmatrix} + \begin{bmatrix}\sum_{m=1}^{m=4} l_m \cos\left(\theta_{(i-1)4} + \sum_{n=1}^{n=m}\theta_{in}\right)\\ \sum_{m=1}^{m=4} l_m \sin\left(\theta_{(i-1)4} + \sum_{n=1}^{n=m}\theta_{in}\right)\\ \sum_{m=1}^{m=4}\theta_{im}\end{bmatrix} - \begin{bmatrix}x_{i4}\\ y_{i4}\\ \theta_{i4}\end{bmatrix} = \mathbf{0} \tag{18.21}$$

subscript m ($m \in [1, 4]$) identifying the proximal to distal pseudo-rigid links, respectively, and l_m representing the link length. In the case of the first backbone sub-segment, x_{04} and y_{04} are zero. Furthermore, the positional constraint relating the nodal position of two spacers is based on the propagation of pseudo-rigid links of the current backbone sub-segment. Four $\boldsymbol{\phi}_{kpi}$, or 12 scalar equations, can be derived from the four backbone-spacer connections.

On the other hand, the kinematic constraint for the spacer-tendon connections is derived from the position of the tendon pass-through or attachment points of spacer plates:

$$\begin{aligned}\boldsymbol{\phi}_{ktai} = &\left(\begin{bmatrix}x_{(i-1)4}\\ y_{(i-1)4}\end{bmatrix} + w_{(i-1)4}\begin{bmatrix}\cos\left(\theta_{(i-1)4} \pm \pi/2\right)\\ \sin\left(\theta_{(i-1)4} \pm \pi/2\right)\end{bmatrix} + d_{tai}\begin{bmatrix}\cos\theta_{tai}\\ \sin\theta_{tai}\end{bmatrix}\right)\\ &- \left(\begin{bmatrix}x_{i4}\\ y_{(i4}\end{bmatrix} + w_{i4}\begin{bmatrix}\cos\left(\theta_{i4} \pm \pi/2\right)\\ \sin\left(\theta_{i4} \pm \pi/2\right)\end{bmatrix}\right) = \mathbf{0}\end{aligned} \tag{18.22}$$

TABLE 18.1
Design parameters of the soft robot.

Parameter	Description	Value & Unit
l_o	Length of sub-segments	70 mm
E	Young modulus of the backbone	231.1 GPa
h	Cross-sectional height of the backbone	12.7 mm
b	Cross-sectional width of the backbone	0.3 mm
w_0	Width of the ground spacer	160 mm
w_1	Width of the first spacer	160 mm
w_2	Width of the second spacer	120 mm
w_3	Width of the third spacer	80 mm
w_4	Width of the fourth spacer	40 mm
k_c	Stiffness of the actuation tendon	185 N/mm/mm

where the sine and cosine terms have positive and negative signs for tendons 1 ($a = 1$) and 2 ($a = 2$), respectively. Design parameter w is as listed in Table 18.1. The expressions in the first and the second brackets are the position of tendon pass-through or attachment points of spacer i, represented using the generalised coordinates of the previous spacer plate $(i-1)$ and the current tendon section, and those of the current space plate i, respectively. The numbers of sub-arrays are four and two, correspondingly, for tendons 1 and 2.

Lastly, the overall lengths of two actuation tendons need to be constrained:

$$\phi_{kla} = (l_a + \delta l_a) - \left(u_a + \sum d_{tai}\right) = 0 \tag{18.23}$$

l_a and δl_a representing the overall length of the unstretched tendon and the tendon elongation. The expression in the first bracket is thus the overall length of the tendon under the actuator input force. The expression in the second bracket, on the other hand, is the same tendon length but written in terms of u_a the actuator input and the sum of sectional lengths d_{tai}. As well, $\boldsymbol{\phi}_{kla}$ introduces additional unknowns of tendon elongation:

$$\boldsymbol{\delta}\mathbf{l}_a = \begin{bmatrix}\delta l_1 & \delta l_2\end{bmatrix}^T \tag{18.24}$$

Array $\boldsymbol{\phi}_k$ of the kinematic constraint equations collects those derived based on (18.21), (18.22), and (18.23), and is given by:

$$\begin{aligned}\boldsymbol{\phi}_k = \big[&\boldsymbol{\phi}_{kp1}^T \quad \boldsymbol{\phi}_{kp2}^T \quad \boldsymbol{\phi}_{kp3}^T \quad \boldsymbol{\phi}_{kp4}^T \\ &\boldsymbol{\phi}_{kt11}^T \quad \boldsymbol{\phi}_{kt12}^T \quad \boldsymbol{\phi}_{kt13}^T \quad \boldsymbol{\phi}_{kt14}^T \quad \boldsymbol{\phi}_{kt21}^T \quad \boldsymbol{\phi}_{kt22}^T \quad \phi_{kl1} \quad \phi_{kl2}\big]^T\end{aligned} \tag{18.25}$$

Furthermore, according to (18.25), array $\boldsymbol{\lambda}$ of the Lagrange multipliers is defined as:

$$\begin{aligned}\boldsymbol{\lambda} = \big[&\boldsymbol{\lambda}_{kp1}^T \quad \boldsymbol{\lambda}_{kp2}^T \quad \boldsymbol{\lambda}_{kp3}^T \quad \boldsymbol{\lambda}_{kp4}^T \\ \big[&\boldsymbol{\lambda}_{kt11}^T \quad \boldsymbol{\lambda}_{kt12}^T \quad \boldsymbol{\lambda}_{kt13}^T \quad \boldsymbol{\lambda}_{kt14}^T \quad \boldsymbol{\lambda}_{kt21}^T \quad \boldsymbol{\lambda}_{kt22}^T \quad f_{kl1} \quad f_{kl2}\big]^T\end{aligned} \tag{18.26}$$

where its entries represent the constraining forces and moments transmitted through the fixed joints (for $\boldsymbol{\phi}_{kpi}$), the interaction forces between the tendon and the spacer plate expressed as x- and y- components (for $\boldsymbol{\phi}_{ktai}$), and the input force transmitted between the actuator and the tendon (for ϕ_{kli}).

18.3.4 Hooke Constitutive Equations

Hooke constitutive equations can be derived from two compliant elements: the virtual torsional springs from the 3R-PRB model and the elastic actuation tendon. Under the assumption of linear elasticity, the former can be written (18.12) as:

$$\phi_{sim} = f_{eim} + k_m \theta_{im} = 0 \tag{18.27}$$

for the torsional springs of backbone sub-segment i. Subscripts m ($m \in [1, 3]$) corresponds to the proximal, middle, and distal virtual revolute joints, respectively. The joint stiffness k_m is defined by the 3R-PRB model, the joint displacement θ_{im} is a generalised coordinate, and f_{eim} is the unknown input torque of the virtual actuator, contained in both $\mathbf{f}_e$ of (18.4) and $\mathbf{f}_{eu}$ of (18.5).

On the other hand, the tendon elongation is related to the input force $\mathbf{f}_{ea}$ as

$$\phi_{sa} = f_{ea} - k_c \delta l_a = 0 \tag{18.28}$$

for the elastic tendon. Tendon stiffness k_c is a design parameter listed in Table 18.1, f_{ea} the actuator input force of tendon a is presented in $\mathbf{f}_e$, and the tendon elongation from $\boldsymbol{\delta}\mathbf{l}_a$ of (18.24). Note that the (18.27) and (18.28) have opposite signs — f_{eim} is internal (virtual external) while f_{ea} is external.

Array $\boldsymbol{\phi}_s$ of the Hooke constitutive equation is hence given by

$$\boldsymbol{\phi}_s = \begin{bmatrix} \phi_{s11} & \cdots & \phi_{s43} & \phi_{s1} & \phi_{s2} \end{bmatrix}^T \tag{18.29}$$

which contains 12 equations of the virtual revolute joints and two equations of the tendons.

18.3.5 Constraint Inequalities

As discussed previously, the internal forces of the tendon rigid bodies must be zero or positive for physically meaningful tendons. Corresponding constraint inequalities are written as:

$$\phi_{kla} = 0 - f_{kla} \leq 0 \tag{18.30}$$

λ_{fla} is the tendon force involved in (18.25). The system requires two scalar inequalities for the actuation tendons.

18.3.6 Constrained Minimisation

According to (18.1), the constrained minimisation problem of the robot is formulated as:

$$\begin{array}{lr} \min & V(\mathbf{q}_u) \\ \text{subject to} & \boldsymbol{\phi}_1(\mathbf{q}_u) = \mathbf{0} \\ & \boldsymbol{\phi}_2(\mathbf{q}_u) \leq \mathbf{0} \end{array} \tag{18.31}$$

with

$$V(\mathbf{q}_u) = \sum_{i=1}^{i=4} \sum_{m=1}^{m=3} \frac{1}{2} k_m \theta_{im}^2 + \sum_{a=1}^{a=2} \frac{1}{2} k_c \delta l_a^2$$

as the objective function, and

$$\boldsymbol{\phi}_1(\mathbf{q}_u) = \begin{bmatrix} \boldsymbol{\phi}_k^T & \left(\left(\frac{\partial \, \boldsymbol{\phi}_k}{\partial \mathbf{q}} \right)^T \boldsymbol{\lambda} - \mathbf{f}_e \right)^T & \boldsymbol{\phi}_s^T \end{bmatrix}^T = \mathbf{0} \tag{18.32}$$

for the constraint equations. Arrays $\mathbf{q}$, $\mathbf{f}_e$, $\boldsymbol{\phi}_k$, $\boldsymbol{\lambda}$, and $\boldsymbol{\phi}_s$ are defined in (18.18), (18.19), (18.25), (18.26), and (18.29), respectively. No selection matrix is used, as all the constraints are valid at all times. Furthermore,

$$\boldsymbol{\phi}_2(\mathbf{q}_u) = \begin{bmatrix} \phi_{kl1} & \phi_{kl2} \end{bmatrix}^T \tag{18.33}$$

representing the constraint inequalities.

The unknowns, $\mathbf{q}_u$, are defined as following (18.15) as

$$\mathbf{q}_u = \begin{bmatrix} \mathbf{q}^T & \boldsymbol{\delta}\mathbf{l}_a^T & \mathbf{f}_{eu}^T & \boldsymbol{\lambda}^T \end{bmatrix}^T \tag{18.34}$$

where $\mathbf{q}$ and $\mathbf{f}_{eu}$ are displayed in (18.18) and (18.20), respectively. Array $\boldsymbol{\lambda}$ is defined in (18.26), and $\boldsymbol{\delta}\mathbf{l}_a$, i.e., array $\mathbf{q}_r$, of (18.24).

18.3.7 Experimental Verification

The experiment setup for verification is illustrated in Figure 18.2, where two weights were connected to the tendons to apply the prescribed input forces. The external force applied on the backbone was also varied. The following table summarises the prescribed input and external forces for each experiment case.

Config.	f_{e1} (N)	f_{e2} (N)	External force (N)
1	0.9830	0	0
2	0.7828	0	0
3	0.9830	0	f_{e34y} = -0.1962 (penultimate spacer)
4	0.9830	0.4905	0
5	0.9830	0.4905	f_{e34y} = -0.1962 (penultimate spacer)
6	0	0.1962	0
7	0	0.4905	0

Figure 18.3 illustrates the comparison between the experimental measurements of the spacer locations ("EXP") and their predictions ("CA"). A close alignment between the two results can be observed. Sources of errors include the friction between the tendons and the spacers at the pass-through holes, and the deviation in the tendon routing brought about by the clearance of these holes.

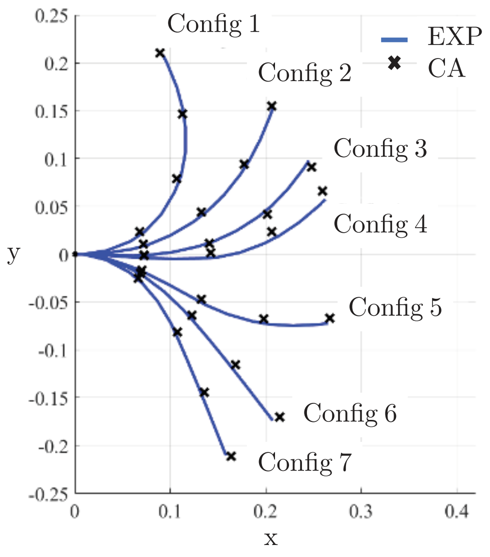


FIGURE 18.3
Comparison between experimental measurements and predictions.

18.4 Case Study — The Compliant Five-Link Epicyclic Finger

In this case study, we apply the framework to model a planar grasping case involving an underactuated prosthetic finger and circular objects [10]. The aim is to demonstrate:

- The use of selection matrices to derive pertinent kinetostatic constraints of a partial-contact grasping case, based on the master kinetostatic constraints of the full-contact case.
- The kinetostatic constraints of contacts, and the formulation of tangential and normal contact forces.

The prosthetic finger of interest is illustrated in Figure 18.4. It is an articulated finger, named the compliant five-bar epicyclic finger (the CFLE finger) [8, 9, 10]. The prosthetic finger adopts three phalanges, whose motion is constrained by an auxiliary link and a pair of epicyclic gears to mimic the freehand flexion trajectory of a human finger. Such an anthropomorphic motion is, in turn, beneficial in both increasing the chance of establishing successful grasping of objects and mitigating the psychological issue associated with hand loss. Furthermore, the CFLE finger offers grasping adaptability, where the spring steel rod embedded in the auxiliary link deforms to allow the finger to conform to the shape of the object.

The kinematic structure of the CFLE finger is now described, with nodes and key generalised coordinates highlighted in Figure 18.5. Links AB (red), BC (blue), and CD (red) are the proximal, middle, and distal phalanges. The sun gear is attached to the base at O, and the planet to the auxiliary link EF at E. Said auxiliary link comprises three sub-segments: the rigid proximal sub-segment EE_1 (solid blue), the rigid distal sub-segment F_1F (solid blue), and a compliant middle sub-segment E_1F_1 (dashed). Based on the 3R-PRB model, the compliant sub-segment is further modelled as four pseudo-rigid links E_1G, GH, HI, and IF_1, connected by spring-loaded virtual revolute joints G, H, and I. The design parameters of an adult-size CFLE index finger are listed below.

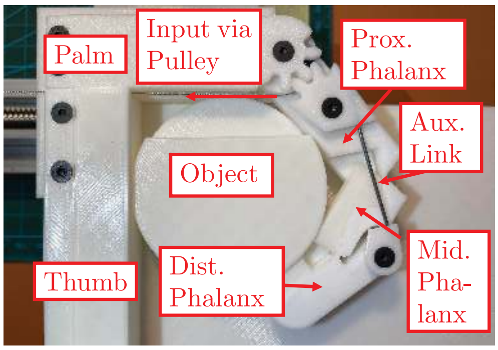
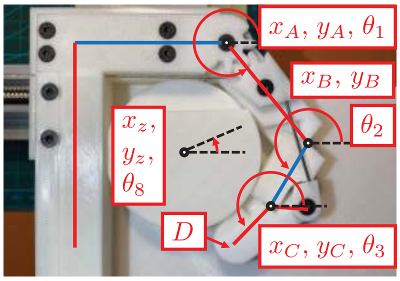


FIGURE 18.4
The finger-object system with a CFLE finger.



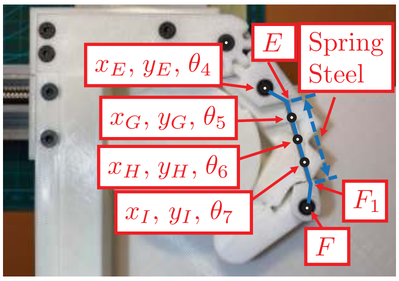


FIGURE 18.5
Nodes and key generalised coordinates of the finger-object system, on the main body of the body and the object (left) and the auxiliary link (right).

Parameter	Description	Value & Unit
l_{AB}	Length of proximal phalanx	44.7 mm
l_{BC}	Length of mid phalanx	26.1 mm
l_{CD}	Length of distal phalanx	17.7 mm
l_{CF}	Length of CF on distal phalanx	12.0 mm
α_3	$\angle DCF$	135.0 deg
r_0	Radius of the sun gear	8.6 mm
r_4	Radius of the planetary gear	11.4 mm
l_{EF}	Distance between joints E and F	43.0 mm
$l_{E_1F_1}$	Length of PRB-3R sub-kinematic chain	28.4 mm
l_{EE_1}	Length proximal extension of E_1F_1	8.6 mm
l_{F_1F}	Length of distal extension of E_1F_1	9.3 mm
θ_{40}	Angle between EF and x axis at home position	10.0 deg
α_4	$\angle FEE_1$	42.43 deg
α_5	$\angle EE_1G$	134.6 deg
α_7	$\angle IF_1F$	155.7 deg
E	Young modulus of compliant rod	202.9 GPa
d_4	Diameter of compliant rod	1.0 mm
k_j	Stiffness of joints B and C	0.0033 Nm/rad
l_{kn}	Thickness of finger segments	8.0 mm

According to the 3R-PRB model, the lengths of the pseudo-rigid links are $0.125l_{E_1F_1}$, $0.35l_{E_1F_1}$, $0.388l_{E_1F_1}$, and $0.136l_{E_1F_1}$, respectively, and the stiffness of the virtual revolute joints are, $3.25EI/l_{E_1F_1}$, $2.84EI/l_{E_1F_1}$, and $2.95EI/l_{E_1F_1}$, respectively, where I is calculated based on the d_4 the diameter of the spring steel rod.

18.4.1 Generalised Coordinates and Generalised External Forces

According to (18.3), three types of bodies are incorporated to derive array $\mathbf{q}$ of the generalised coordinates:

- The rigid object.
- The rigid links of the phalanges.
- The pseudo-rigid bodies of the auxiliary links.

The resultant array $\mathbf{q}$ of the generalised coordinates is thus written as:

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_Z^T & \mathbf{q}_A^T & \mathbf{q}_B^T & \mathbf{q}_C^T & \mathbf{q}_E^T & \mathbf{q}_G^T & \mathbf{q}_H^T & \mathbf{q}_I^T \end{bmatrix}^T \tag{18.35}$$

TABLE 18.2
Generalised coordinates of the system.

Positional q	Description	Orientational q	Description
x_Z, y_Z	Position of the object	θ_8	Orientation of the object
x_A, y_A	Position of AB	θ_1	Orientation of AB
x_B, y_B	Position of BC	θ_2	Orientation of AB
		θ_{12}	Angle of joint B
x_C, y_C	Position of CD	θ_3	Orientation of CD
		θ_{23}	Angle of joint C
x_E, y_E	Position of EE_1G	θ_4	Orientation of EE_1G
x_G, y_G	Position of GH	θ_5	Orientation of GH
		θ_{45}	Angle of joint G
x_H, y_H	Position of HI	θ_6	Orientation of HI
		θ_{56}	Angle of joint H
x_I, y_I	Position of IF_1F	θ_7	Orientation of IF_1F
		θ_{67}	Angle of joint I

sub-array $\mathbf{q}_Z$ containing the position and orientation of the object, and $\mathbf{q}_A$, $\mathbf{q}_B$, and $\mathbf{q}_C$ are those of three phalanges, respectively. Scalar entries of the generalised coordinates of the pseudo-rigid bodies are collected in $\mathbf{q}_E$ to $\mathbf{q}_I$. The rigid sections of the auxiliary link EE_1 and F_1F are modelled as proximal and distal extensions of the pseudo-rigid bodies E_1G and IF_1, respectively. The list of scalar generalised coordinates is shown in Table 18.2, along with a graphical representation in Figure 18.5. It is noteworthy that for the convenience of expressing Hooke constitutive equations, we introduced joint angles $\theta_{(i-1)i}$ for the physical (B and C) and virtual (G, H, and I) spring-loaded revolute joints in addition to the absolute joint orientation θ_i ($i \in [1, 8]$), which require additional orientational kinematic constraints.

For simplification, we take advantage of the *a priori* conditions to reduce the number of unknown generalised coordinates:

- The orientation θ_8 of the object is treated as zero since it is circular.
- The position x_A and y_A of the proximal phalanx at joint A are zero.
- The orientation θ_4 of the pseudo-rigid body EE_1G can be calculated based on θ_1 and the gear ratio.

Array $\mathbf{q}_{qu}$ of the unknown generalised coordinates thus becomes:

$$\mathbf{q}_{qu} = \begin{bmatrix} \mathbf{p}_Z^T & \theta_1 & \mathbf{q}_B^T & \mathbf{q}_C^T & \mathbf{p}_E^T & \mathbf{q}_G^T & \mathbf{q}_H^T & \mathbf{q}_I^T \end{bmatrix}^T \tag{18.36}$$

with sub-arrays $\mathbf{p}_Z$ and $\mathbf{p}_E$ identifying the position of the object and EE_1G, respectively.

Assuming no external load applied on the object, array $\mathbf{f}_e$ corresponding to the (18.18) is defined as:

$$\mathbf{f}_e = \begin{bmatrix} \mathbf{0}_2 & \tau_{eA} & \mathbf{0}_3 & \tau_{eB} & \mathbf{0}_3 & \tau_{eC} & \mathbf{0}_9 & \tau_{eG} & \mathbf{0}_3 & \tau_{eH} & \mathbf{0}_3 & \tau_{eI} \end{bmatrix}^T \tag{18.37}$$

among them, τ_{eA} is the prescribed actuator input torque at joint A. All other scalar τ_e are the actuator torques applied by the virtual spring-loaded revolute joints, forming array $\mathbf{f}_{eu}$ of the unknown generalised external forces:

$$\mathbf{f}_{eu} = \begin{bmatrix} \tau_{eB} & \tau_{eC} & \tau_{eG} & \tau_{eH} & \tau_{eI} \end{bmatrix}^T \tag{18.38}$$

18.4.2 Kinematic Constraint Equations and Lagrange Multipliers

We consider three types of kinematic constraints: those brought about by the joints, the gear meshing between the sun and the planet gears, and the contacts between the phalanges and the object. It is worth noting that for simplicity, we omit the fingertip contact and only consider palmar contact on the distal phalanx.

Kinematic constraint $\boldsymbol{\phi}_{kpi}$ derived from joint i is similar to that presented in (18.21) of the previous case study:

$$\boldsymbol{\phi}_{kpi} = \left(\begin{bmatrix} x_{i-1} \\ y_{i-1} \\ \theta_{i-1} \end{bmatrix} + \begin{bmatrix} l_{(i-1)i} \cos \theta_{i-1} \\ l_{(i-1)i} \sin \theta_{i-1} \\ \theta_{(i-1)i} \end{bmatrix} \right) - \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix} = \mathbf{0} \tag{18.39}$$

$l_{(i-1)i}$ denoting the length of the link listed. As well, angular offsets α from the table are also embedded. It should be noted that $\boldsymbol{\phi}_{kpA}$, $\boldsymbol{\phi}_{kpE}$, and $\boldsymbol{\phi}_{kpF}$ only comprise the positional scalar equations. The reason for the former two is that no joint angle is assigned to their corresponding links. As for $\boldsymbol{\phi}_{kpF}$, the orientational scalar equation is omitted since the joint is passive, providing no orientational constraint. Eight sets of joint constraint equations can be derived based on the joints of the CFLE finger.

The gear mesh constraint $\boldsymbol{\phi}_{km}$ is expressed in terms of the position of the meshing point that is always coincident with link AB:

$$\boldsymbol{\phi}_{km} = r_0 \begin{bmatrix} \cos \delta\theta_0 \\ \sin \delta\theta_0 \end{bmatrix} - \left(\begin{bmatrix} x_E \\ y_E \end{bmatrix} + r_4 \begin{bmatrix} \cos(\theta_4 + \delta\theta_4 + \pi) \\ \cos(\theta_4 + \delta\theta_4 + \pi) \end{bmatrix} \right) = \mathbf{0} \tag{18.40}$$

where $\delta\theta_0 = \theta_1$ and $\delta\theta_4 = \theta_{40} - (r_0/r_4)\,\theta_1$ represent the rotation of the sun and the planet gears, respectively. It is noteworthy that these conditions, along with those of the prescribed generalised coordinates, are only substituted into $\boldsymbol{\phi}_1$ of (18.1) once the partial derivative $\frac{\partial\, \boldsymbol{\phi}_k}{\partial \mathbf{q}}$ has been taken, in order to obtain the correct paths of internal force/moment transmission and the resultant Lagrange multipliers. The resultant Lagrange multipliers of (18.40) correspond to the force transmitted through the gear mesh.

Contact kinematic constraints $\boldsymbol{\phi}_{kk}$ are written in the form of positional constraint equations of the contact points. For a stable grasp to be established, we introduce two additional contact surfaces in addition to the phalanges: the palm and the thumb. Both are assumed stationary in our case study and thus represented by rigid links. For contact surface r ($r = [0, 4]$ for the palm, three phalanges, and the thumb), the expressions are given by:

$$\boldsymbol{\phi}_{kkr} = \left(\begin{bmatrix} x_r \\ y_r \end{bmatrix} + {}^0\mathbf{R}_r \begin{bmatrix} l_{ktr} \\ l_{knr} \end{bmatrix} \right) \left(\begin{bmatrix} x_Z \\ y_Z \end{bmatrix} + r_Z \begin{bmatrix} \cos(\theta_8 + \beta_r) \\ \sin(\theta_8 + \beta_r) \end{bmatrix} \right) = \mathbf{0} \tag{18.41}$$

design parameter l_{knr} representing the distance between the center line of link r to the contact surface, i.e., the thickness. In the case study, we adopt the same thickness, l_{kn}, for all the contacting links. Contact variable l_{ktr} identifies the distance from the origin of the contacting link to the contact point. They belong to the $\mathbf{q}_r$ of (18.15). For the convenience of applying selection matrices, we collect them into array $\mathbf{q}_{kk}$ of the unknown contact variables:

$$\mathbf{q}_{kk} = \begin{bmatrix} l_{kt0} & l_{kt1} & l_{kt2} & l_{kt3} & l_{kt4} \end{bmatrix}^T \tag{18.42}$$

The last contact variable β_r is a temporary one, introduced such that the expression in the second bracket of (18.41) can be written without using the generalised coordinates of the contacting body r. Essentially, β_r are given by:

$$\beta_r = \begin{cases} \pi/2 - \theta_8 & r = 0 \\ \theta_r + \pi/2 - \theta_8 & r = 1, 2, 3 \\ \pi - \theta_8 & r = 4 \end{cases}$$

which represents the angle of the vector directing from the centre of the object to the contact point, measured with respect to θ_8. Similar to other a priori conditions, β_r are not substituted into $\boldsymbol{\phi}_1$ for simplification until the partial derivative $\frac{\partial\, \phi_k}{\partial \mathbf{q}}$ has been taken. Lastly, ${}^0\mathbf{R}_r$ of (18.41) is the 2×2 z-axis rotation matrix of link r, written as:

$$
{}^0\mathbf{R}_r = \begin{cases} \mathbf{R}_z(0) & r = 0 \\ \mathbf{R}_z(\theta_r) & r = 1, 2, 3 \\ \mathbf{R}_z(-\pi/2) & r = 4 \end{cases}
$$

Array $\boldsymbol{\phi}_k$ of the kinematic constraints collects the constraint equations of the three types:

$$
\boldsymbol{\phi}_k = \begin{bmatrix} \boldsymbol{\phi}_{kp}^T & \boldsymbol{\phi}_{km}^T & \boldsymbol{\phi}_{kk}^T \end{bmatrix} \tag{18.43}
$$

each sub-array further comprising scalar kinematic constraint equations of their corresponding types. Array $\boldsymbol{\lambda}^*$ of the Lagrange multipliers is hence defined as:

$$
\boldsymbol{\lambda}^* = \begin{bmatrix} \boldsymbol{\lambda}_p^T & \boldsymbol{\lambda}_m^T & \boldsymbol{\lambda}_k^{*\,T} \end{bmatrix} \tag{18.44}
$$

Notably, $\boldsymbol{\lambda}_k^*$ represents the contact forces written as x- and y-components. On the other hand, it is more intuitive to express them as tangential and normal components, whose benefits are two-fold. Firstly, constraint inequalities are required to avoid negative normal contact force, making physically meaningful contacts. Secondly, dedicated tangential contact forces allow frictionless contact force equations or frictional contact force inequalities to be conveniently expressed. For this purpose, we introduce additional mapping between the x- and y-components of the contact forces and their tangential and normal counterparts:

$$
\begin{bmatrix} f_{krx} \\ f_{kry} \end{bmatrix} = {}^0\mathbf{R}_r{}' \begin{bmatrix} f_{krt} \\ f_{krn} \end{bmatrix} \tag{18.45}
$$

upon substitution, $\boldsymbol{\lambda}_k^*$ becomes $\boldsymbol{\lambda}_k$, comprising f_{krt} the scalar entry of tangential contact force and f_{krn} the normal contact force. The z-axis rotation matrix ${}^0\mathbf{R}_r{}'$ in $\boldsymbol{\lambda}_k$ is defined as

$$
{}^0\mathbf{R}_r{}' = \begin{cases} \mathbf{R}_z\,(\pi) & r = 0 \\ \mathbf{R}_z\,(\theta_r + \pi) & r = 1, 2, 3 \\ \mathbf{R}_z\,(-\pi/2) & r = 4 \end{cases} \tag{18.46}
$$

the rotation matrices are defined in such a way that the positive normal contact forces are in their respective positive y directions. Furthermore, array $\boldsymbol{\lambda}^*$ of the Lagrange multipliers is modified into:

$$
\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_p^T & \boldsymbol{\lambda}_m^T & \boldsymbol{\lambda}_k^T \end{bmatrix} \tag{18.47}
$$

For the convenience of expressing the array of unknown variables, we also assign $\boldsymbol{\lambda}_{ku}$ to collect the unknown contact forces:

$$
\boldsymbol{\lambda}_{ku} = \begin{bmatrix} f_{k0t} & f_{k0n} & f_{k1t} & f_{k1n} & f_{k2t} & f_{k2n} & f_{k3t} & f_{k3n} & f_{k4t} & f_{k4n} \end{bmatrix}^T \tag{18.48}
$$

18.4.3 Hooke Constitutive Equations

The Hooke constitutive equations of the CFLE finger are derived from the spring-loaded virtual revolute joints. The expression is similar to (18.27) of the previous case study:

$$
\phi_{sv} = \tau_{ev} + k_v \theta_{(v-1)v} = 0 \tag{18.49}
$$

where k_B and k_C the stiffness of the physical joints, and k_G, k_H, and k_I are determined based on the 3R-PRB model. Their corresponding joint angles, θ_{12}, θ_{23}, θ_{45}, θ_{56}, and θ_{67} are in $\mathbf{q}$. Array $\boldsymbol{\phi}_s$ collects all five scalar equations:

$$\boldsymbol{\phi}_s = \begin{bmatrix}\phi_{sB} & \phi_{sC} & \phi_{sG} & \phi_{sH} & \phi_{sI}\end{bmatrix}^T \tag{18.50}$$

18.4.4 Force Constraints of Contacts

As mentioned previously, additional force constraints are required to 1) ensure positive normal contact force and 2) relate tangential contact forces to the normal ones based on the friction conditions.

The force constraint inequality for normal contact force is defined below:

$$\phi_{fnr} = 0 - f_{krn} \leq 0 \tag{18.51}$$

In the frictionless case, the force constraint equation is written as:

$$\phi_{ftr} = f_{krt} - \mu f_{krn} = 0 \tag{18.52}$$

where μ the friction coefficient is zero. On the other hand, the frictional contact assumption leads to force constraint inequalities:

$$\phi_{ftr} = ||f_{krt}|| - \mu f_{krn} \leq 0 \tag{18.53}$$

18.4.5 Selection Matrices

Given five possible contact surfaces, there are 32 contact cases overall. Each contact case is identified by a set of five binary numbers, each digit corresponding to a contact surface. Furthermore, 0 and 1 refer to the no-contact condition and an established contact, respectively.

Selection matrices are applied to contact-related variables and constraints:

- Array $\mathbf{q}_{kk}$ of the contact variables of (18.42).
- Contact kinematic constraints $\boldsymbol{\phi}_{kk}$ of (18.43) and Lagrange multipliers $\boldsymbol{\lambda}_{ku}$ in (18.48).
- Normal contact force constraints $\boldsymbol{\phi}_{fn}$, collecting force constraint inequalities (18.51).
- Tangential contact force equations or inequalities $\boldsymbol{\phi}_{ft}$ with scalar entries of (18.52) or (18.53), respectively.

Notice that $\mathbf{q}_{kk} \in \mathbb{R}^5$ and the rest of the arrays are of $\mathbb{R}^{10}$ since each contact has two components, we only need two selection matrices. Their full-selection matrices are $\mathbf{S}_{k1} \in \mathbb{R}^{5\times5}$ and $\mathbf{S}_{k2} \in \mathbb{R}^{10\times10}$. The columns $\mathbf{s}$ corresponding to individual contacts are listed below.

Contact	Relevant $\mathbf{s}$ of $\mathbf{S}_{k1}$	Relevant $\mathbf{s}$ of $\mathbf{S}_{k2}$
Palm contact	$\mathbf{s}_1$	$\mathbf{s}_1$, $\mathbf{s}_2$
Prox. phalanx contact	$\mathbf{s}_2$	$\mathbf{s}_3$, $\mathbf{s}_4$
Middle phalanx contact	$\mathbf{s}_3$	$\mathbf{s}_5$, $\mathbf{s}_6$
Dist. phalanx contact	$\mathbf{s}_4$	$\mathbf{s}_7$, $\mathbf{s}_8$
Thumb contact	$\mathbf{s}_5$	$\mathbf{s}_9$, $\mathbf{s}_{10}$

The subscripts represent pertinent columns. As well, for columns of $\mathbf{S}_{k2}$, the odd- and even-number subscripts correspond to the tangential and the normal contacts, respectively. The arrays of variables or scalar equations/inequalities of a contact case can thus be determined based on $\mathbf{S}'_{k1}$ and $\mathbf{S}'_{k2}$ following (18.14).

18.4.6 Constrained Minimisation

According to (18.1), the constrained minimisation problem of the finger-object system is formulated as:

$$\begin{aligned} &\min && V(\mathbf{q}_u) \\ &\text{subject to} && \boldsymbol{\phi}_1(\mathbf{q}_u) = \mathbf{0} \\ & && \boldsymbol{\phi}_2(\mathbf{q}_u) \leq \mathbf{0} \end{aligned} \tag{18.54}$$

with

$$V(\mathbf{q}_u) = \sum_{v=1}^{v=5} \frac{1}{2} k_v \theta^2_{(v-1)v}$$

as the objective function describing the elastic potential energy of joints B, C, G, H, and I.

In a prescribed contact case, the kinetostatic constraint equations are given below for the frictionless and frictional contact conditions, respectively:

$$\boldsymbol{\phi}_1(\mathbf{q}_u) = \begin{bmatrix} {\boldsymbol{\phi}'_k}^T & \left(\left(\frac{\partial\, \boldsymbol{\phi}'_k}{\partial \mathbf{q}}\right)^T \boldsymbol{\lambda}' - \mathbf{f}_e\right)^T & \left({\mathbf{S}'_{k2}}^T\, \boldsymbol{\phi}_{ft}\right)^T & \boldsymbol{\phi}_s^T \end{bmatrix}^T = \mathbf{0} \tag{18.55}$$

and

$$\boldsymbol{\phi}_1(\mathbf{q}_u) = \begin{bmatrix} {\boldsymbol{\phi}'_k}^T & \left(\left(\frac{\partial\, \boldsymbol{\phi}'_k}{\partial \mathbf{q}}\right)^T \boldsymbol{\lambda}' - \mathbf{f}_e\right)^T & \boldsymbol{\phi}_s^T \end{bmatrix}^T = \mathbf{0} \tag{18.56}$$

arrays $\mathbf{q}$, $\mathbf{f}_e$, and $\boldsymbol{\phi}_s$ are defined in (18.36), (18.37), and (18.49), respectively. The tangential contact force equation $\boldsymbol{\phi}_{ft}$ contains scalar entries of (18.52), and $\mathbf{S}'_{k2}$ is formulated upon removal of relevant columns of $\mathbf{S}_{k2}$ from the previous sub-section. As well, arrays $\boldsymbol{\phi}_k{}'$ and $\boldsymbol{\lambda}'$ now have selection matrix $\mathbf{S}'_{k2}$ embedded:

$$\boldsymbol{\phi}_k{}' = \begin{bmatrix} \boldsymbol{\phi}_{kp}^T & \boldsymbol{\phi}_{km}^T & \left({\mathbf{S}'_{k2}}^T\, \boldsymbol{\phi}_{kk}\right)^T \end{bmatrix}^T$$

and

$$\boldsymbol{\lambda}' = \begin{bmatrix} \boldsymbol{\lambda}_p^T & \boldsymbol{\lambda}_m^T & \left({\mathbf{S}'_{k2}}^T\, \boldsymbol{\lambda}_k\right)^T \end{bmatrix}^T$$

relevant arrays are defined as follows: $\boldsymbol{\phi}_{kp}$, $\boldsymbol{\phi}_{km}$, and $\boldsymbol{\phi}_{kk}$ in (18.43); $\boldsymbol{\lambda}_p$, $\boldsymbol{\lambda}_m$, and $\boldsymbol{\lambda}_k$ in (18.47). Once $\boldsymbol{\phi}_1$ with the correct force constraint equations has been obtained, the a priori conditions can be substituted for simplification.

The kinetostatic constraint inequalities corresponding to the frictionless and frictional contact conditions, respectively, are defined as:

$$\boldsymbol{\phi}_2(\mathbf{q}_u) = \left({\mathbf{S}'_{k2}}^T\, \boldsymbol{\phi}_{fn}\right)^T \tag{18.57}$$

and

$$\boldsymbol{\phi}_2(\mathbf{q}_u) = \begin{bmatrix} \left({\mathbf{S}'_{k2}}^T\, \boldsymbol{\phi}_{ft}\right)^T & \left({\mathbf{S}'_{k2}}^T\, \boldsymbol{\phi}_{fn}\right)^T \end{bmatrix}^T \tag{18.58}$$

arrays $\boldsymbol{\phi}_{fn}$ and $\boldsymbol{\phi}_{ft}$ of the tangential and normal contact force inequalities contain scalar entries of (18.51) and (18.53), respectively.

The unknowns, $\mathbf{q}_u$, are defined as following (18.15) as

$$\mathbf{q}_u = \begin{bmatrix} \mathbf{q}_{qu}^T & \left({\mathbf{S}'_{k1}}^T \mathbf{q}_{kk}\right)^T & \mathbf{f}_{eu}^T & \boldsymbol{\lambda}_p^T & \boldsymbol{\lambda}_m^T & \left({\mathbf{S}'_{k2}}^T\, \boldsymbol{\lambda}_{ku}\right)^T \end{bmatrix}^T \tag{18.59}$$

where $\mathbf{q}_{qu}$, $\mathbf{f}_{eu}$ $\mathbf{q}_{kk}$, and $\boldsymbol{\lambda}_{ku}$ are defined in (18.36), (18.38), (18.42), and (18.48), respectively. In addition, $\mathbf{S}'_{k1}$ is derived by removing the relevant columns of $\mathbf{S}_{k1}$ of the previous sub-section, according to the contact case.

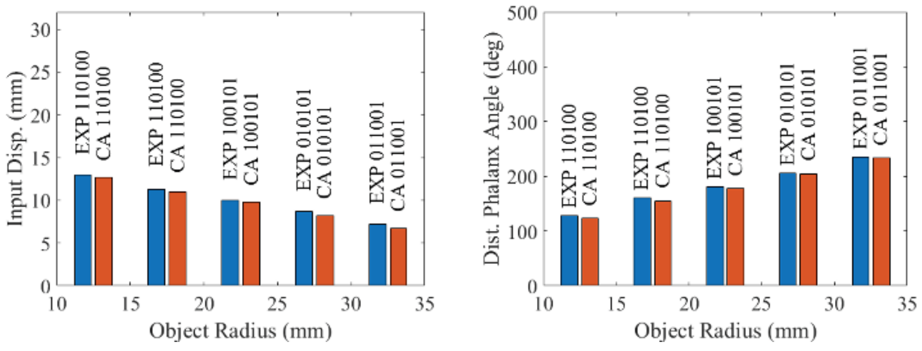


FIGURE 18.6
Input displacements (left) and distal phalanx angles (right) under small prescribed input forces.

Solving (18.54) reveals the configuration and force information for the prescribed contact case. On the other hand, while not all contact cases will converge to feasible solutions, it is possible to find multiple valid contact cases. In this case, the genuine solution V^* is the one corresponding to the lowest elastic potential energy of all n valid solutions:

$$V^*(\mathbf{q}_u) = \min \begin{pmatrix} V_1 & V_2 & \cdots & V_n \end{pmatrix} \tag{18.60}$$

18.4.7 Experimental Verification

The formulation was verified through grasping experiments based on the prototype displayed in Figure 18.4. We prescribed the input force by gradually increasing the input displacement (via a pulley with 10 mm diameter), until said force magnitude has been reached. The corresponding input displacement, orientation of the distal phalanx, and contact case are recorded for each combination of object diameter and prescribed input force. The circular objects tested were 25 mm to 65 mm in diameter, with a 10 mm interval. In addition, each object was tested under two prescribed input forces.

The results are illustrated in Figures 18.6 and 18.7, for small and large prescribed input forces, respectively. In the former, all objects adopted the same input force of 10 N. As for

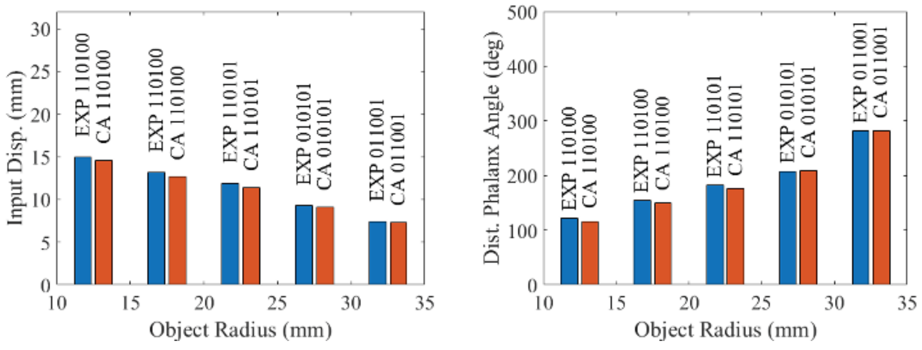


FIGURE 18.7
Input displacements (left) and distal phalanx angles (right) under large prescribed input forces.

the latter case, the 45 mm object used 50 N so that a change in the contact case could be observed, while 30 N was applied to the other objects. The legends, EXP and CA, identify the experimental measurements and the predictions, respectively. The horizontal axes mark the diameters of the object, whereas the vertical axes are the input displacements in mm or the angles of the distal phalanges. Also, the contact case is labelled with a five-digit binary number for each combination of the prescribed input force and object diameter.

These results show that our formulation based on constraint analysis was verified, as indicated by the consistency between the predictions and the experimental measurements:

- The maximum errors in the input displacement and distal phalanx angle are 6.9% and 4.3%, respectively.
- All predicted contact cases agreed with the experimental ground truth, i.e., our approach has captured the change in contact case with respect to the variation in both object size and input force.
- Internal friction, e.g., that from the joints and finger-object contact, is likely the major source of error since the experimental input displacements were constantly larger than the predictions.

18.5 Conclusion

In this chapter, we discussed the kinematic- and force-domain analysis of planar underactuated systems, which paves the way for the investigation of soft robotics. The proposed method takes advantage of Lagrange multipliers and pseudo-rigid bodies to bridge the gap between underactuated and ordinary fully actuated mechanisms. It allows the knowledge accumulated from the analysis of conventional rigid-body mechanisms to be exploited to conduct a unified kineto-static analysis of the soft and underactuated mechanisms, and shall provide opportunities for future in-depth investigations to advance the theoretical foundation of soft robotics. We completed the discussion with two case studies, where the proposed method was applied to analyse a tendon-driven robot and a finger prosthesis, respectively.

Bibliography

[1] S. Liu, C. Chen, and J. Angeles, "A novel framework for the analysis of underactuated fingers," *Mechanism and Machine Theory*, vol. 182, p. 105238, 2023.

[2] C. Troeung, S. Liu, and C. Chen, "Modelling of tendon-driven continuum robot based on constraint analysis and pseudo-rigid body model," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 989–996, 2023.

[3] C. Chen, "Power analysis of epicyclic transmissions based on constraints," *Journal of Mechanisms and Robotics*, vol. 4, no. 4, pp. 1–11, 2012.

[4] C. Chen and J. Chen, "Efficiency analysis of two degrees of freedom epicyclic gear transmission and experimental validation," *Mechanism and Machine Theory*, vol. 87, pp. 115–130, 2015.

[5] N. Hu, S. Liu, D. Zhao, and C. Chen, "Power analysis of epicyclic gear transmission for wind farm," in *Robotics and Mechatronics: Proceedings of the Fifth IFToMM International Symposium on Robotics & Mechatronics (ISRM 2017) 5*, pp. 251–262, Springer, 2019.

[6] A. A. Shabana, *Computational dynamics.* John Wiley & Sons, 2009.

[7] S. Huang, D. Meng, Y. She, X. Wang, B. Liang, and B. Yuan, "Statics of continuum space manipulators with nonconstant curvature via pseudorigid-body 3r model," *IEEE Access*, vol. 6, pp. 70854–70865, 2018.

[8] S. Liu, M. Van, Z. Chen, and C. Chen, "Theoretical joint load analysis of a novel prosthetic digit design," in *Advances in Mechanism and Machine Science: Proceedings of the 15th IFToMM World Congress on Mechanism and Machine Science 15*, pp. 113–122, Springer, 2019.

[9] S. Liu, M. Van, Z. Chen, J. Angeles, and C. Chen, "A novel prosthetic finger design with high load-carrying capacity," *Mechanism and Machine Theory*, vol. 156, p. 104121, 2021.

[10] S. Liu, J. Angeles, and C. Chen, "A novel adaptive prosthetic finger design with scalability," in *IFToMM World Congress on Mechanism and Machine Science*, pp. 85–95, Springer, 2023.

19

Concentric Tube Robot

Robotic manipulators are seeing ever-increasing applications in minimally invasive surgery (MIS) by virtue of their increased accuracy and repeatability, reduced size, the capability to restore lost manipulation degrees of freedom (DoF) inside the patient's body, and the potential for teleoperation. Minimally invasive surgery refers to surgical procedures in which the sites are accessed through small incision ports rather than big openings [1]. The advantages of adopting small incision ports include less damage to healthy tissues, reduced blood loss, decreased level of pain, and faster post-operative recovery and hospital stay. MIS procedures can be broadly categorised into two classes: the cannula type and the laparoscopic type. The cannula type often involves sending a surgical tool, e.g., a heart stent, through a cannula that navigates via blood vessels to reach the surgical site. In this chapter, we discuss concentric tube robots, a class of continuum robots particularly suitable for cannula applications.

19.1 Overview of Concentric Tube Robot

We have established that a cannula is required to deploy the surgical end-effector to the lesion in the surgical process of interest. A critical feature needed to achieve this objective is navigation, such that the cannula could advance along the correct body vessels. Furthermore, it is also desirable that the navigation is achieved actively, independent of the off-axis force provided upon contacting anatomical structures. Concentric tube robots (CTR) [2, 3], by virtue of their inherent compliance, mechanical simplicity, and active navigation capability, are highly suitable to be used as active cannulae for minimally invasive procedures.

A concentric tube robot comprises several super-elastic tubes arranged in a concentric manner. Each of these tubes features pre-curved sections of various curvatures at different locations along their arc lengths, whereby relative rotation and translation among the tubes dictates the final shape of the CTR upon static equilibrium, as shown in Figure 19.1. The relative rotation and translation of the tubes thus serve as the required actuator inputs. The simplified analogy of a CTR in the planar case is a system of two extension or compression springs, where the position of the connection point can be controlled by altering the lengths of the springs via translational inputs.

The benefits brought about by the unique mechanical structure and working principle of the CTR are now detailed. Firstly, by adopting super-elastic tubes as their bodies, CTR are compliant and will deform upon contact with anatomical structures, maximising patient safety during the procedures. Secondly, since the super-elastic tubes serve as both the backbone and the transmission while all the actuators are located at the proximal end, the diameter of a CTR could be small, allowing it to advance through body vessels not accessible by other soft robots or active cannulae. Lastly, the CTR inherently has a hollow centre

DOI: 10.1201/9781003614319-19

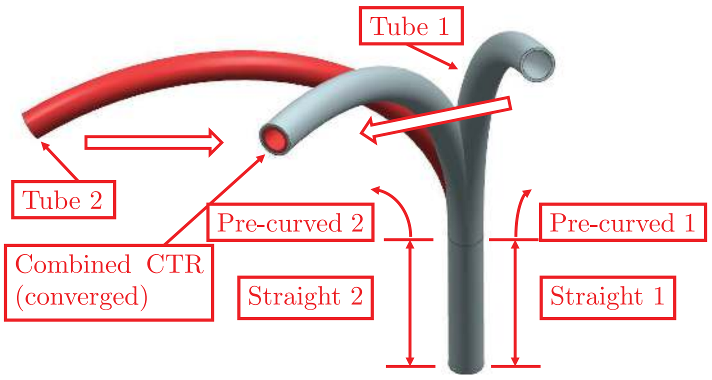


FIGURE 19.1
A CTR comprising two super-elastic tubes.

that may be used to deliver surgical devices, extract removed tissues, or run the actuation cable of a surgical end-effector through.

Despite the advantages, modelling a CTR remains a challenging task. Unlike the robotic manipulators described in earlier parts of the book, which are modelled through connected rigid or pseudo-rigid bodies, a CTR is formulated as a continuum body through the Cosserat rod theory. Following this method, the description of the configuration of a CTR can be interpreted by means of the trajectory of a moving frame travelling along the arc length of the CTR, from its proximal to the distal end. Herein, the shear and axial elongation, and the bending and twisting, are the equivalent of linear and angular velocities of the moving frame (with respect to the arc length), respectively. The formulation of CTR thus comprises two mappings: the *robot-independent* (RI) that integrates the velocity to obtain the position and orientation of the robot cross-sections and the *robot-dependent* (RD) mapping that computes the velocity or local curvatures based on static equilibrium and constitutive law. The two mappings are detailed in the two ensuing sections.

It is noteworthy that the formulation based on the Cosserat rod yields no closed-form solution, and propagation from one end of the robot is needed to compute its configuration. Moreover, the prescribed actuator inputs and external loads are at two ends of the CTR, rending a boundary value problem with split boundary conditions and the necessity to use the shooting method to obtain the solution. Therefore, the ways in which the governing equation, i.e., the combination of static equilibrium and constitutive law, is expressed in the RD mapping significantly impact the computational efficiency, and we will demonstrate such a variation in the corresponding section.

19.2 Robot-Independent Mapping

The robot-independent mapping relates the velocity of the moving frame and its trajectory and is, namely, irrelevant to the mechanical structure of the robot. Thanks to the similarity between a CTR and a strand, i.e., both being elongated structures that can bend and twist, we can take advantage of a strand model [4] to describe the robot-independent mapping. Its key equations are summarised below.

For an arbitrary continuum body, frame $\{s\}$ assigned to a cross-section s can be regarded as the instantaneous configurations of the moving frame. In Cosserat rod theory, the transformation matrix $\mathbf{T}$ between adjacent frames $\{s\}$ and $\{s+h\}$, h being the distance between the cross-sections, is defined as

$$^{s+h}\mathbf{T}_s \frac{d}{ds}\,{}^s\mathbf{T}_{s+h} = \begin{bmatrix} {}^s\mathbf{u}_s\times & {}^s\mathbf{v}_s \\ 0 & 0 \end{bmatrix} \tag{19.1}$$

$^s\mathbf{u}_s$ and $^s\mathbf{v}_s$ denote the angular and the linear velocities of frame $\{s\}$ with respect to the ground and expressed in $\{s\}$, respectively. Additionally, $^s\mathbf{u}_s\times$ is the cross-product of the velocity. In $^s\mathbf{u}_s$, the x- and y-components correspond to the scalar bending curvatures, while the z-component represents the torsional curvature. Similarly, the first two scalar entries of $^s\mathbf{v}_s$ are for shear, and the last one is the axial elongation.

Equation (19.1) is expressed in the discretised form, for numerical integration, as

$$^s\mathbf{T}_{s+h} = \begin{bmatrix} e^{\mathbf{u}_n\times h_n} & (\mathbf{1}^3 - e^{\mathbf{u}_n\times h_n})\,\mathbf{u}_n\times\mathbf{v}_n + \mathbf{u}_n\boldsymbol{u}_n^T\,\mathbf{v}_n h_n \\ 0 & 1 \end{bmatrix} \tag{19.2}$$

with

$$e^{\mathbf{u}_n\times h_n} = \mathbf{1}^3 + \mathbf{u}_n\times\sin h_n + \mathbf{u}_n\times\ \mathbf{u}_n\times(1-\cos h_n) \tag{19.3}$$

which is, in turn, written with normalised velocities and step size:

$$\begin{aligned} \mathbf{u}_n &= \frac{^s\boldsymbol{u}_s}{\|^s\boldsymbol{u}_s\|} \\ \mathbf{v}_n &= \frac{^s\boldsymbol{v}_s}{\|^s\boldsymbol{u}_s\|} \\ h_n &= h\|^s\boldsymbol{u}_s\| \end{aligned} \tag{19.4}$$

Provided that the external disturbances in the intended application of the CTR are small, we can neglect the shear ϵ_{xx} and ϵ_{yy} and the axial elongation ϵ_{zz}, leading to simplified linear velocity:

$$^s\mathbf{v}_s = \begin{bmatrix}\epsilon_{xx} & \epsilon_{yy} & 1+\epsilon_{zz}\end{bmatrix}^T = \begin{bmatrix}0 & 0 & 1\end{bmatrix}^T \tag{19.5}$$

On the other hand, the local curvatures $^s\mathbf{u}_s$ of the CTR are further described through two arrays: $^{0s}\mathbf{u}_{0s}$ of the CTR, and $^{is}\mathbf{u}_{is}$ of super-elastic tube i. The former contains exclusively scalar bending curvatures. The latter also carries a z-component representing the torsional curvature. Such a description finds its root in the mechanical structure of the CTR: while all the super-elastic tubes converge to a common bending shape, they also feature their respective axial twists. Said mechanical relationship can be described mathematically as

$$^{is}\mathbf{u}_{is} = {}^{is}\mathbf{R}_{0s}\,{}^{0s}\mathbf{u}_{0s} + {}^{is}\mathbf{u}_{isz} \tag{19.6}$$

$^{is}\mathbf{R}_{0s}$ is the 3×3 rotation matrix of the i-th super-elastic tube at cross-section s, essentially a z-axis rotation matrix of θ_{is}. The z-axis is, in turn, perpendicular to cross-section s or tangential to the CTR. $^{is}\mathbf{u}_{isz}$ is the array of torsional curvature, carrying zero x- and y-components and a non-zero z-component $^{is}u_{isz}$. Additionally, the scalar torsional curvature $^{is}u_{isz}$ is the spatial derivative of the axial rotation θ_{is} contained in $^{is}\mathbf{R}_{0s}$, i.e.,

$$\frac{d}{ds}\theta_{is} = {}^{is}u_{isz} \tag{19.7}$$

Based on the description of curvature, we assign $\{0s\}$ and $\{is\}$ to cross-section s of the CTR and the i-th super-elastic tube. Figure 19.2 depicts the graphical representation of the frames.

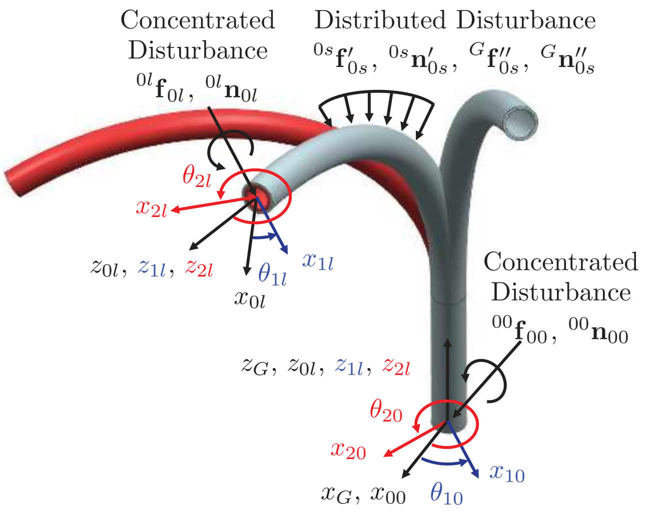

Concentrated Disturbance
$^{0l}\mathbf{f}_{0l}$, $^{0l}\mathbf{n}_{0l}$
Distributed Disturbance
$^{0s}\mathbf{f}'_{0s}$, $^{0s}\mathbf{n}'_{0s}$, $^{G}\mathbf{f}''_{0s}$, $^{G}\mathbf{n}''_{0s}$
θ_{2l}
x_{2l}
z_{0l}, z_{1l}, z_{2l}
θ_{1l}
x_{1l}
x_{0l}
Concentrated Disturbance
$^{00}\mathbf{f}_{00}$, $^{00}\mathbf{n}_{00}$
z_G, z_{0l}, z_{1l}, z_{2l}
θ_{20}
x_{20}
x_G, x_{00}
θ_{10}
x_{10}

FIGURE 19.2
Frames and external disturbances of a CTR.

19.3 Robot-Dependent Mapping

In the robot-dependent mapping , the angular velocity of the moving frame, i.e., the bending and torsional curvature, is computed by means of static equilibrium and constitutive equation based on the actuator inputs, the pre-defined curvatures of the super-elastic tubes, and the external disturbances.

In the formulation targeting minimally invasive procedures, we consider the following external disturbances distributed along the body of the CTR: the local frame force $\mathbf{f}'$ and moment $\mathbf{n}'$; and global frame force $\mathbf{f}''$ and moment $\mathbf{n}''$. On the other hand, the concentrated external disturbances, e.g., those applied by the actuators at the proximal end and the surgical forces and moments at the distal end, are incorporated in $\mathbf{f}$ and $\mathbf{n}$ at the cross-sections of the CTR. The aforementioned external forces and moments are also illustrated in Figure 19.2. We further assume all the external disturbances are known, requiring force sensing measurements to be deployed in the application, e.g., tip-mounted and actuator-attached force-torque sensors or an array of fibre Bragg grating sensors.

19.3.1 Constitutive Equation

Hooke's constitutive law incorporates the bending and torsional curvatures into the model by converting the curvatures to moments for the static equilibrium equations. To achieve this, the constitutive equation is expressed in the local frame $\{is\}$. Under the assumption of linear elasticity, we have at cross-section s:

$$^{is}\mathbf{n}_{is} = \mathbf{K}_{is}\left(^{is}\mathbf{u}_{is} - {}^{is}\mathbf{u}^{*}_{is}\right) \tag{19.8}$$

$^{is}\mathbf{u}^{*}_{is}$ denoting the prescribed curvature before deformation, and the stiffness matrix $\mathbf{K}_{is}$ is defined as

$$\mathbf{K}_{is} = \begin{bmatrix} E_{ix}I_{ixx} & 0 & 0 \\ 0 & E_{iy}I_{iyy} & 0 \\ 0 & 0 & G_{iz}I_{izz} \end{bmatrix}$$

E, G, and I identifying the Young's modulus, the shear modulus, and the second moment of area, respectively. Furthermore, since super-elastic tubes are often made homogeneously with uniform cross-sectional shapes along the arc length, we assume $E_{ix} = E_{iy} = E_i$, $I_{ixx} = I_{iyy} = I_i$, and $\mathbf{K}_{is} = \mathbf{K}_i$ for simplicity of derivation.

The moment associated with the deformation of the super-elastic tubes, i.e., Equation (19.8), is expressed in $\{is\}$ since the pre-defined curvatures are too represented in the local frame. However, it must be expressed in the CTR's frame $\{0s\}$ to be incorporated into the static equilibrium equation. Substituting Equation (19.6) into (19.8) yields the moment in $\{0s\}$:

$$ {}^{0s}\mathbf{n}_{is} = {}^{0s}\mathbf{R}_{is}\,\mathbf{K}_i\left({}^{is}\mathbf{R}_{0s}\,{}^{0s}\mathbf{u}_{0s} + {}^{is}\mathbf{u}_{isz} - {}^{is}\mathbf{u}^*_{is}\right) \tag{19.9} $$

As well, considering a CTR of m super-elastic tubes, the overall moment generated by the deformation of all super-elastic tubes is defined below:

$$ {}^{0s}\mathbf{n}_{0s} = \sum_{i=1}^{i=m} {}^{0s}\mathbf{R}_{is}\,\mathbf{K}_i\left({}^{is}\mathbf{R}_{0s}\,{}^{0s}\mathbf{u}_{0s} + {}^{is}\mathbf{u}_{isz} - {}^{is}\mathbf{u}^*_{is}\right) \tag{19.10} $$

19.3.2 Static Equilibrium

For a finite CTR section, the static equilibrium expressed in the global frame $\{G\}$ is given by:

$$ -\,{}^{G}\mathbf{R}_{0a}\,{}^{0a}\mathbf{f}_{0a} + {}^{G}\mathbf{R}_{0b}\,{}^{0b}\mathbf{f}_{0b} + \int_a^b {}^{G}\mathbf{R}_{0s}\,{}^{0s}\mathbf{f}'_{0s} + {}^{G}\mathbf{f}''_{0s}\,ds = \mathbf{0} \tag{19.11} $$

for the force equation, and

$$ \begin{aligned} &-\,{}^{G}\mathbf{p}_{GA} \times {}^{G}\mathbf{R}_{0a}\,{}^{0a}\mathbf{f}_{0a} + {}^{G}\mathbf{p}_{GB} \times {}^{G}\mathbf{R}_{0b}\,{}^{0b}\mathbf{f}_{0b} - {}^{G}\mathbf{R}_{0a}\,{}^{0a}\mathbf{n}_{0a} + {}^{G}\mathbf{R}_{0b}\,{}^{0b}\mathbf{n}_{0b} \\ &\quad + \int_a^b {}^{G}\mathbf{p}_{GS} \times \left({}^{G}\mathbf{R}_{0s}\,{}^{0s}\mathbf{f}'_{0s} + {}^{G}\mathbf{f}''_{0s}\right) ds + \int_a^b {}^{G}\mathbf{R}_{0s}\,{}^{0s}\mathbf{n}'_{0s} + {}^{G}\mathbf{n}''_{0s}\,ds = \mathbf{0} \end{aligned} \tag{19.12} $$

for the moment equation around the origin of $\{G\}$. ${}^{0a}\mathbf{n}_{0a}$ and ${}^{0b}\mathbf{n}_{0b}$ are the concentrated moments transmitted through the boundary cross-sections a and b, respectively. They can be related to the bending and torsional curvatures of their respective CTR finite sections following Equation (19.10). Their force counterparts, ${}^{0a}\mathbf{f}_{0a}$ and ${}^{0b}\mathbf{f}_{0b}$, are the collection of forces transmitted through individual super-elastic tubes, namely,

$$ {}^{0s}\mathbf{f}_{0s} = \sum_{i=1}^{i=m} {}^{0s}\mathbf{R}_{is}\,{}^{is}\mathbf{f}_{is} \tag{19.13} $$

In the static equilibrium equation, $\mathbf{R}$ are the 3×3 rotation matrices. Additionally, ${}^{G}\mathbf{p}_{GS}/{}^{G}\mathbf{p}_{GA}/{}^{G}\mathbf{p}_{GB}$ denote the position vectors directed from the origin of the global frame to cross-sections s, a, and b, respectively, and represented in the global frame. It is also noteworthy that the external disturbance may be given in either a global or local frame. For example, the interaction force between the surgical end-effector and the tissue could be a local frame concentrated force, while the weight of the CTR is expressed as a distributed force in the global frame. Regardless, all the external disturbances must be mapped into the same frame, in this case $\{G\}$, to be incorporated into the static equilibrium equation.

Equations (19.11) and (19.12) can be equivalently expressed into a unified equation as

$$ \int_a^b \frac{d}{ds}\left({}^{G}\mathbf{X}_{0a}\begin{bmatrix}{}^{0a}\mathbf{f}_{0a}\\ {}^{0a}\mathbf{n}_{0a}\end{bmatrix}\right) ds + \int_a^b {}^{G}\mathbf{X}_{0s}\begin{bmatrix}{}^{0s}\mathbf{f}'_{0s}\\ {}^{0s}\mathbf{n}'_{0s}\end{bmatrix} + \begin{bmatrix}{}^{G}\mathbf{f}''_{0s}\\ {}^{G}\mathbf{p}_{GS} \times {}^{G}\mathbf{f}''_{0s} + {}^{G}\mathbf{n}'_{0s}\end{bmatrix} ds = \mathbf{0} \tag{19.14} $$

where the 6×6 transformation matrix is discussed previously in Chapter 12:

$$ {}^{G}\mathbf{X}_{0s} = \begin{bmatrix} {}^{G}\mathbf{R}_{0s} & \mathbf{0} \\ {}^{G}\mathbf{R}_{0s}\,{}^{0s}\mathbf{p}_{GS}\times & {}^{G}\mathbf{R}_{0s} \end{bmatrix} = \begin{bmatrix} {}^{G}\mathbf{R}_{0s} & \mathbf{0} \\ {}^{G}\mathbf{p}_{GS}\times {}^{G}\mathbf{R}_{0s} & {}^{G}\mathbf{R}_{0s} \end{bmatrix} \tag{19.15} $$

Taking the derivative of (19.14) leads to the relation between the change in cross-sectional force and moment with respect to the external disturbances:

$$ \frac{d}{ds}\left({}^{G}\mathbf{X}_{0a}\begin{bmatrix} {}^{0a}\mathbf{f}_{0a} \\ {}^{0a}\mathbf{n}_{0a} \end{bmatrix}\right) + {}^{G}\mathbf{X}_{0a}\begin{bmatrix} {}^{0a}\mathbf{f}'_{0a} \\ {}^{0a}\mathbf{n}'_{0a} \end{bmatrix} + \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ {}^{G}\mathbf{p}_{GA}\times & \mathbf{1} \end{bmatrix}\begin{bmatrix} {}^{G}\mathbf{f}''_{0a} \\ {}^{G}\mathbf{n}''_{0a} \end{bmatrix} = \mathbf{0} \tag{19.16} $$

$\mathbf{1}$ being the 3×3 identity matrix. We can further express Equation (19.16) into the local frame $\{0a\}$ through the inverse transformation matrix ${}^{0a}\boldsymbol{X}_{G}$:

$$ \frac{d}{ds}\begin{bmatrix} {}^{0a}\mathbf{f}_{0a} \\ {}^{0a}\mathbf{n}_{0a} \end{bmatrix} = -\begin{bmatrix} {}^{0a}\mathbf{u}_{0a}\times & \mathbf{0} \\ \mathbf{v}\times & {}^{0a}\mathbf{u}_{0a}\times \end{bmatrix}\begin{bmatrix} {}^{0a}\mathbf{f}_{0a} \\ {}^{0a}\mathbf{n}_{0a} \end{bmatrix} - \begin{bmatrix} {}^{0a}\mathbf{f}'_{0a} \\ {}^{0a}\mathbf{n}'_{0a} \end{bmatrix} - \begin{bmatrix} {}^{0a}\mathbf{R}_{G} & \mathbf{0} \\ \mathbf{0} & {}^{0a}\mathbf{R}_{G} \end{bmatrix}\begin{bmatrix} {}^{G}\mathbf{f}''_{0a} \\ {}^{G}\mathbf{n}''_{0a} \end{bmatrix} \tag{19.17} $$

allowing the cross-sectional forces and moments of the CTR to be propagated. However, Equation (19.17) is written based on ${}^{0a}\mathbf{u}_{0a}$ for the combined CTR, and the torsional curvatures of individual super-elastic tubes are not captured. Additional z-axis equations are needed for this purpose.

For a super-elastic tube, the equivalent of the moment equilibrium of Equation (19.17) can be derived through a similar process as:

$$ \frac{d}{ds}\,{}^{ia}\mathbf{n}_{ia} = -\,\mathbf{v}\times{}^{ia}\mathbf{f}_{ia} - {}^{ia}\mathbf{u}_{ia}\times{}^{ia}\mathbf{n}_{ia} - {}^{ia}\mathbf{n}'_{ia} - {}^{ia}\mathbf{R}_{G}\,{}^{G}\mathbf{n}''_{ia} \tag{19.18} $$

where all the variables on the right-hand side of the equation, aside from ${}^{ia}\mathbf{n}_{ia}$, are unknown. However, since our interest is on the z-component, assumptions could be made according to the application in minimally invasive procedures, such that the equation is solvable. Firstly, the effect of ${}^{ia}\mathbf{f}_{ia}$ vanishes in the cross-product, as the x- and y-components of $\boldsymbol{v}$ are assumed zero. Secondly, the only source of z-axis local frame concentrated moment, ${}^{ia}n'_{iaz}$, is the inter-super-elastic-tube friction, which could be assumed zero. Finally, the z-component of global frame distributed moment, ${}^{G}n''_{iaz}$, could also be assumed zero since disturbances of this kind rarely present in the target application. The z-axis scalar Equation (19.18) can thus be simplified into:

$$ \frac{d}{ds}\,{}^{ia}n_{iaz} = -E_iI_i\left(-{}^{ia}u_{iax}\,{}^{ia}u^{*}_{iay} + {}^{ia}u_{iay}\,{}^{ia}u^{*}_{iax}\right), \tag{19.19} $$

${}^{ia}u_{iax}$ and ${}^{ia}u_{iay}$ denoting the x- and y-components of the bending curvature, their pre-defined counterparts identified by the asterisks. Additionally, Equation (19.9) could be substituted into (19.19) for direct propagation of the torsional curvature, namely,

$$ \frac{d}{ds}\,{}^{ia}u_{iaz} = -\frac{E_iI_i}{G_{iz}I_{izz}}\left(-{}^{ia}u_{iax}\,{}^{ia}u^{*}_{iay} + {}^{ia}u_{iay}\,{}^{ia}u^{*}_{iax}\right) \tag{19.20} $$

19.3.3 Variations of the Governing Equation

We have now established that the governing set of equations, comprising (19.10), (19.17), and (19.20), to solve for the local curvatures of the RD mapping. However, the governing equation may be expressed in different forms, leading to variation in the computational efficiency.

The first variation is the direct implementation of the governing equations, except that the constitutive equation is rearranged into the form below to provide the bending curvature explicitly:

$$ {}^{0a}\mathbf{u}_{0a}|_{xy} = \left(\sum_{i=1}^{i=m} \mathbf{K}_i \right)^{-1} \left({}^{0a}\mathbf{n}_{0a} + \sum_{i=1}^{i=m} {}^{0a}\mathbf{R}_{ia}\ \mathbf{K}_i\ {}^{ia}\mathbf{u}_{ia}^{*} \right)\Bigg|_{xy} \tag{19.21}$$

with subscripts xy denoting the scalar x and y equations, while the z-component of ${}^{0a}\mathbf{u}_{0a}$ is by definition zero. It is noteworthy that the inverse stiffness term is, in fact, ${}^{0a}\mathbf{R}_{ia}\ \mathbf{K}_i\ {}^{ia}\mathbf{R}_{0a}$, and the simplification to $\mathbf{K}_i$ is feasible due to the assumption of equal stiffness in x- and y-directions.

In the second variation, the number of scalar equations is reduced upon eliminating force scalar equations in (19.17). This is achieved by integrating the forces from cross-section a to the distal end of the CTR l and substituting into the moment scalar equations, namely,

$$\begin{aligned} \frac{d}{ds}\ {}^{0a}\mathbf{n}_{0a} &= -\ {}^{0a}\mathbf{u}_{0a} \times {}^{0a}\mathbf{n}_{0a} - {}^{0a}\mathbf{n}'_{0a} - {}^{0a}\mathbf{R}_G\ {}^{G}\mathbf{n}''_{0a} \\ &\quad - \mathbf{v} \times \left({}^{0a}\mathbf{R}_{0l}\ {}^{0l}\mathbf{f}_{0l} + {}^{0a}\mathbf{R}_G \int_a^l {}^{G}\mathbf{R}_{0s}\ {}^{0s}\mathbf{f}'_{0s} + {}^{G}\mathbf{f}''_{0s} ds \right) \end{aligned}$$

However, said simplification comes with a cost. Since rotation matrices of the farther cross-sections, ${}^{0a}\mathbf{R}_{0l}$ and ${}^{G}\mathbf{R}_{0s}$, are unknown, this equation is applicable when only the global frame distributed force ${}^{G}\mathbf{f}''_{0s}$ presents. In this case, said equation is essentially

$$\frac{d}{ds}\ {}^{0a}\mathbf{n}_{0a} = -\ {}^{0a}\mathbf{u}_{0a} \times {}^{0a}\mathbf{n}_{0a} - {}^{0a}\mathbf{n}'_{0a} - {}^{0a}\mathbf{R}_G\ {}^{G}\mathbf{n}''_{0a} - \mathbf{v} \times {}^{0a}\mathbf{R}_G \int_a^l {}^{G}\mathbf{f}''_{0s} ds \tag{19.22}$$

It is also noteworthy that the global frame concentrated external force ${}^{G}\mathbf{f}_{0l}$ applied at the distal end of the CTR can be incorporated into (19.22) by means of the Dirac Delta function, defined as:

$${}^{G}\mathbf{f}''_{0s} = \begin{cases} {}^{G}\mathbf{f}''_{0s} + {}^{G}\mathbf{f}''_{0l} & l-h \le s < l \\ {}^{G}\mathbf{f}''_{0s} & 0 \le s < l-h \end{cases}$$

i.e., the concentrated external force is treated as a distributed one applied at the last finite section of the CTR.

Alternatively, the reduction in the number of scalar equations can be achieved by making the constitutive law implicit, resulting in the third variation. The substitution of the Equation (19.10) into the moment component of (19.17) gives

$$\begin{aligned} \frac{d}{ds}\ {}^{0a}\mathbf{u}_{0a}|_{xy} = -\left(\sum_{i=1}^{i=m} \mathbf{K}_i \right)^{-1} &\times \left(\sum_{i=1}^{i=m} {}^{0a}\mathbf{R}_{ia} \left({}^{ia}\mathbf{u}_{ia} \times \mathbf{K}_i \right) \left({}^{ia}\mathbf{R}_{0a}\ {}^{0a}\mathbf{u}_{0a} - {}^{ia}\mathbf{u}_{ia}^{*} + {}^{ia}\mathbf{u}_{iaz} \right) \right. \\ &+ \sum_{i=1}^{i=m} \left(-\ \mathbf{K}_i {}^{0a}\boldsymbol{u}_{iaz} \times {}^{0a}\mathbf{u}_{0a} - {}^{0a}\mathbf{R}_{ia}\ \mathbf{K}_i \frac{d}{ds}\ {}^{ia}\mathbf{u}_{ia}^{*} + {}^{0a}\mathbf{R}_{ia}\ \mathbf{K}_i \frac{d}{ds}\ {}^{ia}\mathbf{u}_{iaz} \right) \\ &\left. +\ \mathbf{v} \times {}^{0a}\mathbf{f}_{0a} + {}^{0a}\mathbf{f}'_{0a} + {}^{0a}\mathbf{R}_G\ {}^{G}\mathbf{f}''_{0a} \right)\Bigg|_{xy} \end{aligned} \tag{19.23}$$

Again, the z-component of the rate of change of ${}^{0s}\mathbf{u}_{0s}$ is by definition zero. A limitation of this method is that it requires the pre-defined curvature ${}^{is}\boldsymbol{u}_{is}^{*}$ to be continuous. However, this is often not the case in the application. A piece-wise approach (within segments of continuous pre-defined curvatures) must be taken in the propagation.

The last variation has the least number of equations by adopting the simplification of the previous two, i.e., to eliminate the force part in the last row of Equation (19.23) through the integration of forces from cross-section a to l. The expression is given by:

$$\begin{aligned}\frac{d}{ds}\,{}^{0a}\mathbf{u}_{0a}|_{xy} = &-\left(\sum_{i=1}^{i=m}\mathbf{K}_i\right)^{-1} \times \left(\sum_{i=1}^{i=m}{}^{0a}\mathbf{R}_{ia}\left({}^{ia}\mathbf{u}_{ia}\times\mathbf{K}_i\right)\left({}^{ia}\mathbf{R}_{0a}\,{}^{0a}\mathbf{u}_{0a} - {}^{ia}\mathbf{u}^*_{ia} + {}^{ia}\mathbf{u}_{iaz}\right)\right.\\ &+\sum_{i=1}^{i=m}\left(-\mathbf{K}_i\,{}^{0a}\boldsymbol{u}_{iaz}\times{}^{0a}\mathbf{u}_{0a} - {}^{0a}\mathbf{R}_{ia}\,\mathbf{K}_i\frac{d}{ds}\,{}^{ia}\mathbf{u}^*_{ia} + {}^{0a}\mathbf{R}_{ia}\,\mathbf{K}_i\frac{d}{ds}\,{}^{ia}\mathbf{u}_{iaz}\right)\\ &+\mathbf{v}\times\left({}^{0a}\mathbf{R}_{0l}\,{}^{0l}\mathbf{f}_{0l} + {}^{0a}\mathbf{R}_G\int_a^l {}^{G}\mathbf{R}_{0s}\,{}^{0s}\mathbf{f}'_{0s} + {}^{G}\mathbf{f}''_{0s}\,ds\right)\\ &\left.\left.+\,{}^{0a}\mathbf{f}'_{0a} + {}^{0a}\mathbf{R}_G\,{}^{G}\mathbf{f}''_{0a}\right)\right|_{xy}\end{aligned} \tag{19.24}$$

The presence of the derivative of pre-defined curvatures requires the propagation based on Equation (19.24) to adopt a piece-wise method similar to that of (19.23).

In summary, the four variations of the governing equation in the RD mapping are:

- Variation 1 — Equations (19.17), (19.20), and (19.21), similar to [5]
- Variation 2 — Equations (19.10), (19.20), and (19.22), reported in the authors' work [6]
- Variation 3 — the force scalar equation of (19.17), along with Equations (19.20) and (19.23), reported in the authors' work [6]
- Variation 4 — Equations (19.20) and (19.24), similar to [7]

We will demonstrate the deviation in the computational efficiency through numerical case studies in the later section.

19.4 Iteration

As established in the previous sections, the kinetostatic formulation of the CTR is a boundary value problem with split boundary conditions. On the proximal end, the actuator input displacements are prescribed, which can be used to identify the pre-defined curvatures ${}^{is}\mathbf{u}^*_{is}$ at each cross-section s of the CTR. Additionally, the axial rotation, θ_{i0}, of the super-elastic tubes at the proximal end of the CTR are known. On the distal ends of the CTR and individual super-elastic tubes, the boundary conditions can be expressed as:

$$\begin{aligned}{}^{0l}\mathbf{f}_{0l} &= {}^{0l}\mathbf{f}^{\dagger}_{0l}\\ {}^{0l}\mathbf{n}_{0l}|_{xy} &= {}^{0l}\mathbf{n}^{\dagger}_{0l}|_{xy}\\ {}^{0m}n_{imz} &= {}^{0l}\mathbf{n}^{\dagger}_{0l}|_{z}\\ {}^{il'}u_{il'z} &= 0 \quad i = 1,\, 2,\, \cdots,\, m-1\end{aligned} \tag{19.25}$$

superscript † denoting the prescribed boundary conditions. Cross-sections l and l', correspondingly, identify the distal ends of the CTR and the rest of the super-elastic tubes. In Equation (19.25), the first and second conditions state that the force and bending moment equilibrium, respectively. The latter two conditions assume the external torsional moments at the distal end are all applied onto the innermost tube m, rendering zero torsional curvature on the rest of the super-elastic tubes.

The shooting method is used to solve the boundary value problem. Since ${}^{0a}\mathbf{R}_G$ presents the equations of RD mapping, the propagation starts from the proximal end of the CTR, and the initial guesses comprise the missing force-domain information and/or the curvatures. Furthermore, the RD mapping may require continuous tube stiffness and pre-defined curvature. In this case, the CTR is segmented for section-wise propagation, where terminals separating two CTR sections are placed according to the following rules: 1) at the junction where one of the super-elastic tubes has discontinuous pre-defined curvatures on two sides and 2) at the distal end of a super-elastic tube due to discontinuous stiffness. During the propagation, Equation (19.10) may be used at the start of a CTR section to recover the moment.

19.5 Case Studies for Computational Efficiency

In case studies, we considered three CTRs whose design parameters are listed as follows:

Design parameters	CTR 1 [8]		CTR 2 [7]		CTR 3 [9]	
	Tube 1	Tube 2	Tube 1	Tube 2	Tube 1	Tube 2
Outer diameter (mm)	2.77	2.41	2.37	1.75	2.29	1.60
Inner diameter (mm)	2.55	1.98	2.00	1.25	2.00	1.20
Length straight section (mm)	0	18	30.7	122.7	40	100
Length curved section (mm)	150	150	102.5	206.9	80	100
Radius of curved section (mm)	233	248	125	200	236	293
Direction of curvature	x	x	$-x$	$-x$	$-x$	$-x$

For a comprehensive evaluation of the computational efficiency, the CTRs were loaded with concentrated and distributed forces, respectively, and the average time taken to solve with respect to a set of prescribed actuator inputs was recorded for comparison. The external loads and actuator inputs adopted are summarised below, where the CTRs were subject to concentrated or distributed force.

Parameters	CTR1	CTR2	CTR3
Input rotation θ_{20} (deg)	0–315	0–315	0–315
Input rotation θ_{20} interval (deg)	45	45	45
Input translation d_{10} (mm)	0	-30.7	0
Input translation d_{20} (mm)	-18	-233.7–-141.7	-80–0
Input translation d_{20} interval (mm)	n/a	23	40
Tip force ${}^G\mathbf{f}_{0l}$ (N)	0–2	0–2	0–2
Tip force ${}^G\mathbf{f}_{0l}$ interval (N)	0.2	0.2	0.2
Tip force direction	$+y$	$+x/y/z$	$+x/y/z$
Distributed force ${}^G\,\mathbf{f}''_{0s}$ (N/m)	0–4	0–2.9	0–3
Distributed force ${}^G\,\mathbf{f}''_{0s}$ interval (N/m)	0.67	0.48	0.50
Distributed force direction	$+x$	$+x$	$+x$

As well, in the experiment, the relative translation and rotation between the super-elastic tubes were generated by means of actuating Tube 2, i.e., the inner tube.

TABLE 19.1
Average Computation Time (s) of CTR under Concentrated Tip Force

Variations	CTR1	CTR2	CTR3
1	22.6	37.9	19.5
2	16.7	30.8	13.8
3	28.7	49.7	24.3
4	21.3	36.5	17.2

Notably, the actuator input translation d_{10} and d_{20} at the proximal ends of the super-elastic tubes were measured from the proximal end of the combined CTR; hence, the magnitudes could be negative. Furthermore, a sole prescribed value refers to a fixed actuator position, whereas a range indicates the use of several discretised positions. Moreover, in the simulation, $+x/y/z$ of "tip force direction" means that the x-, y-, and z- components of the force are identical, and the norm of said force is prescribed.

To implement the shooting method for a solution, we adopted the Runge-Kutta 4 method in the iteration with a step size h of 1 mm and an accuracy of 1×10^{-5} (N or Nm). Additionally, the correction on the initial guesses is defined as:

$$\mathbf{q}_{j+1} = \mathbf{q}_j - 0.5\left(\mathbf{f} - \mathbf{f}^{\dagger}\right) \tag{19.26}$$

$\mathbf{q}_j$ and $\mathbf{q}_{j+1}$ denoting the initial guesses of iteration j and $j+1$, and the correction is made based on the difference between the iterated boundary condition $\mathbf{f}$ and its prescribed counterpart $\mathbf{f}^{\dagger}$. Such a relation is established since it is intuitive that excessive/insufficient actuator inputs will result in excessive/insufficient outputs on the distal end of the CTR.

The comparison in average iteration time is illustrated in Tables 19.1 and 19.2. It can be observed that the two variations, Equations (19.22) and (19.24), where the propagation of force is omitted, yield faster computation subject to concentrated tip force but are slower upon the presence of the distributed force. The degraded performance in the latter case is clearly due to the operation of integrating the distributed force from the cross-section s to the distal end of the CTR. On the other hand, in variations 1 and 4 where the propagation of force is retained, variation 1 seems to be the more computationally efficient despite having more scalar equations (explicit constitutive and moment equilibrium equations vs implicit constitutive equation and explicit moment equilibrium).

TABLE 19.2
Average Computation Time (s) of CTR under Distributed Tip Force

Variations	CTR1	CTR2	CTR3
1	12.2	22.9	9.03
2	16.2	65.9	12.6
3	15.6	41.0	11.6
4	20.5	51.2	15.2

19.6 Conclusion

The work described in this chapter concerns an alternative (vs. that in Chapter 18) method to model a soft robot, this time with continuum bodies that may be too long or too complex for the use of the pseudo-rigid body approach. The method splits the model into two parts: the robot-independent mapping universal to all soft continuum robots and the robot-dependent mapping specifically related to the mechanical designs and actuation schemes on individual robots — the latter centres around static equilibrium, with the kinematic-domain constraints embedded implicitly in the force-domain equations. Unlike the previous method that yields a closed-form analytical solution, the model described herein must be solved iteratively through numerical integration. As such, we further derive the variations of the force domain equations to investigate the deviation in computational efficiency.

Bibliography

[1] C.-H. Kuo, J. S. Dai, and P. Dasgupta, “Kinematic design considerations for minimally invasive surgical robots: an overview,” *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 8, no. 2, pp. 127–145, 2012.

[2] P. Sears and P. Dupont, “A steerable needle technology using curved concentric tubes,” in *2006 IEEE/RSJ international conference on intelligent robots and systems*, pp. 2850–2856, IEEE, 2006.

[3] R. J. Webster, A. M. Okamura, and N. J. Cowan, “Toward active cannulas: Miniature snake-like surgical robots,” in *2006 IEEE/RSJ international conference on intelligent robots and systems*, pp. 2857–2863, IEEE, 2006.

[4] D. K. Pai, “Strands: Interactive simulation of thin solids using Cosserat models,” *Computer graphics forum*, vol. 21, no. 3, pp. 347–352, 2002.

[5] P. E. Dupont, J. Lock, B. Itkowitz, and E. Butler, “Design and control of concentric-tube robots,” *IEEE Transactions on Robotics*, vol. 26, no. 2, pp. 209–225, 2009.

[6] S. T. Liu and C. Chen, “Framework of modelling concentric tube robot and comparison on computational efficiency,” *Meccanica*, vol. 52, pp. 2201–2217, 2017.

[7] D. C. Rucker, B. A. Jones, and R. J. Webster III, “A geometrically exact model for externally loaded concentric-tube continuum robots,” *IEEE transactions on robotics*, vol. 26, no. 5, pp. 769–780, 2010.

[8] J. Lock, G. Laing, M. Mahvash, and P. E. Dupont, , “Quasistatic modeling of concentric tube robots with external loads,” *2010 IEEE/RSJ international conference on intelligent robots and systems*, pp. 2325–2332, 2010.

[9] S. Bai, and C. Xing, “Shape modeling of a concentric-tube continuum robot,” *2012 IEEE international conference on robotics and biomimetics (ROBIO)*, pp. 116–121, 2012.

20

Path Planning of Parallel Manipulators

Our previous discussions are mostly limited to serial (single-chain) manipulators. That is, a robot consisting of a single chain of links and actuators with one end connected to a base, and the other manipulating an end effector. In this chapter, we will briefly introduce *parallel manipulators*. This type of manipulator consists of multiple serial manipulators, but connected in parallel to drive the same payload. There are many advantages to this topology, such as enhanced payload capacity, stiffness, and precision. However, they also bring about many challenges and unique characteristics that have fascinated kinematicians, robotics researchers, and even mathematicians over the last five decades. One of these unique characteristics is their complicated workspaces, which can be represented as complex surfaces or *manifolds*. Due to their parallel structure, they often feature complex singularity profiles that make path planning queries difficult, especially for longer paths in certain configuration spaces. This chapter will describe a generalised method for generating configuration spaces for parallel mechanisms, that can enhance the efficiency of path planning queries.

20.1 Parallel Manipulator Kinematics

The equations that describe the kinematics of a parallel mechanism are often known as *closure loop equations* because these are equations that describe the kinematic loop that is featured in parallel manipulators. We express this equation in the generalised form,

$$\mathbf{F}(\mathbf{x}, \mathbf{q}) = 0 \tag{20.1}$$

where $\mathbf{F}$ is a set of non-linear equations that describe the relationship between task space variables $\mathbf{x}$ and joint space variables $\mathbf{q}$. The path planning variables of parallel manipulators can consist of both joint and task space variables, depending on the manipulator's task and complexity. As these variables are used to fully define the configuration of the parallel manipulator, we can call this the $\mathcal{C}$-space. But while planning occurs in the $\mathcal{C}$-space, the control of a parallel manipulator still fundamentally occurs in the joint space. Therefore, we then require a *mapping* of the joint space variables to the $\mathcal{C}$-space in a one-to-one relation, in other words, a set of equations that can convert joint space variables to a *unique* point in $\mathcal{C}$-space where planning occurs. If a one-to-one mapping is guaranteed, then path planning in the $\mathcal{C}$-space can be greatly simplified.

Let $Q \subset \mathbb{R}^n$ be the set of all possible joint configurations for a parallel mechanism. Then we seek a set of functions $\mathbf{G}$ such that any joint configuration $\mathbf{q} \in Q$ can be mapped to a unique point in the $\mathcal{C}$-space, defined as set $C \subset \mathbb{R}^n$. Therefore

$$\mathbf{x} = \mathbf{G}(\mathbf{q}) \tag{20.2}$$

DOI: 10.1201/9781003614319-20

and

$$\mathbf{G} : Q \to C \tag{20.3}$$

in which $\mathbf{G}$ can be described as the forward kinematics of a parallel manipulator.

The set of functions $\mathbf{G}$, in general, is highly non-linear and very difficult to find closed-loop solutions, especially for parallel manipulators greater than 3 DoF such as the generalised Stewart-Gough platform [25]. A common technique to solve this problem is to represent the closed loop as a univariate polynomial, such that the roots of the polynomial represent the forward kinematic solution

$$P(u, \mathbf{q}) = 0 \tag{20.4}$$

where P is a univariate polynomial in $u \in \mathbb{C}$, whose coefficients are a function of joint space variables $\mathbf{q}$. Let $U \subset \mathbb{R}$ be the set of real roots that satisfy Equation (20.4). Then by applying kinematic mapping X, such that

$$X : U \to C \tag{20.5}$$

the roots of Equation (20.4) are mapped back into the $\mathcal{C}$-space, represented by the set C.

The univariate polynomial expression of the forward kinematics is very useful in that the real roots represent the forward kinematic solution. Numerical root-finding algorithms are quite established, and can find roots of polynomials of very high degree very efficiently. Furthermore, we find that this property can be exploited to significantly enhance $\mathcal{C}$-space generation speed, which is important for applications in higher dimensions.

20.1.1 Singularities

Given the general form of the kinematic constraints of a parallel robotic mechanism in (20.1), differentiating this with respect to time gives the relationship between the joint and end effector velocities

$$J_x(\mathbf{x})\dot{\mathbf{x}} = J_q(\mathbf{q})\dot{\mathbf{q}} \tag{20.6}$$

where J_x and J_q are parallel and serial Jacobians, respectively. Singularities occur when the determinant of either or both Jacobian matrices $J_\mathbf{x}$ and $J_\mathbf{x}$ equate to 0, and confine the mechanism to a single operating configuration where each configuration can be classed as a mode of operation. The types of singularities separating each configuration are inverse kinematic and forward kinematic singularities [3].

20.1.1.1 Type I Singularities: Working Modes

Under Type I parallel singularities [9], $\det J_x = 0$, where J_x becomes rank-deficient. Under this condition, the end-effector gains a degree of freedom (zero $\dot{\mathbf{q}}$ maps to a non-zero $\dot{\mathbf{x}}$), resulting in its loss of control. In terms of kinematics, multiple points in joint space relate to a single point in the task space under this condition. Under naïve path planning schemes that utilise only task space variables, type I singularities fragment the parallel manipulator's workspace into smaller segments called *working modes*. These working modes cannot be changed, unless starting conditions are changed, or direct drive of the actuators is utilised.

20.1.1.2 Type II Singularities: Assembly Modes

These exist as a result of multiple points in task space relating to a single point in joint space. In encountering the change in assembly mode boundary, $\det(J_\mathbf{x}) = 0$ *(type II direct* *kinematic singularity)*, a zero $\dot{\mathbf{q}}$ vector equates to a non-zero $\dot{\mathbf{x}}$. The loss in stiffness in the end effector is highly undesirable and so must be avoided when path planning. Again, under naïive path planning schemes, type II singularities fragment the workspace into smaller segments called *assembly modes*, meaning only a small portion of a parallel manipulator's workspace can be used unless there is user intervention (change in initial conditions), or advanced path planning methods are utilised.

This latter point was a hot topic of research for many roboticists, as workspace fragmentation of parallel manipulators results in limited useful workspace, which eventually restricts their usefulness to very niche applications. This problem was a significant challenge to overcome, but there were certain methods one could implement to enhance their workspace. This chapter introduces one such method.

20.1.2 Example: A 5R Parallel Manipulator

To better understand the problem, we begin with a simple parallel manipulator called the 5R, or five-bar manipulator, as shown in Figure 20.1. It features two kinematic chains, A_1B_1P and A_2B_2P, respectively, attached to a rigid base A_1A_2. Point A_i is an active (driven) joint for kinematic chain i, point B_i is a *passive* joint, and point P is the payload, which is also a passive joint. The output position of P is planar and is purely driven by A_1 and A_2, whose actuator position is defined as θ_1 and θ_2, relative to the y-axis. The bolded links L_1 and L_2 represent the links that are driven by these actuators respectively.

For the 5R manipulator, vectors q and x are

$$\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix} \tag{20.7}$$

and

$$\mathbf{q} = \begin{bmatrix} \theta_1 & \theta_2 \end{bmatrix} \tag{20.8}$$

Therefore, the kinematic constraint equations in the form of (20.1) are

$$(x - L_1 \cos\theta_1)^2 + (y - L_1 \sin\theta_1)^2 - {l_1}^2 = 0 \tag{20.9}$$

$$(x - d_x - L_2 \cos\theta_2)^2 + (y - d_y - L_2 \sin\theta_2)^2 - {l_2}^2 = 0 \tag{20.10}$$

Referring to our 5R parallel manipulator example, taking the time derivative of time-dependent variables x, y, θ_1, and θ_2 we obtain Jacobian matrices

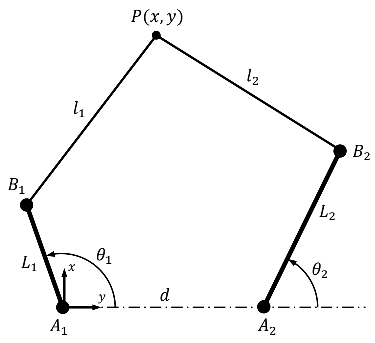


FIGURE 20.1
A 5R manipulator.

$$J_{\mathbf{x}} = \begin{bmatrix} x - L_1\cos\theta_1 & y - L_1\sin\theta_1 \\ x - d_x - L_2\cos\theta_2 & y - d_y - L_2\sin\theta_2 \end{bmatrix} \tag{20.11}$$

$$J_{\mathbf{q}} = \begin{bmatrix} L_1(x\sin\theta_1 - y\cos\theta_1) & 0 \\ 0 & L_2(\sin\theta_2(x - d_x) - \cos\theta_2(y - d_y)) \end{bmatrix} \tag{20.12}$$

20.1.2.1 Assembly Modes

These exist as a result of multiple points in task space relating to a single point in joint space (Figure 20.2(a)). The workspace boundary of an assembly mode is defined at $det(J_x) = 0$, in which a zero $\mathbf{q}$ vector equates to a non-zero $\mathbf{x}$. This indicates a loss of control at the end effector, and should be avoided when path planning. Because of this, the 5R mechanism cannot change assembly modes during operation without special actuation strategies, such as the use of dynamics or momentum to drive the end effector beyond its singularity region. However, we find later than more complex parallel manipulators can, in fact, change assembly modes.

Because the determinant of a configuration can be positive or negative, we can class a configuration as being in the positive or negative *aspect*, or $\oplus$ and $\ominus$ for short, respectively. A singularity-free path between configurations in differing aspects cannot exist because this will result in a sign change in $det(J_x)$, i.e., passing zero.

20.1.2.2 Working Modes

These exist as a result of multiple points in joint space relating to a single point in the task space (Figure 20.2(b)). The workspace boundary of a working mode is defined at $det(J_q) = 0$, in which case for some non-zero $\mathbf{q}$, results in a zero $\mathbf{x}$ vector, indicating loss of one or more degrees of freedom (DoF). Although the mechanism cannot be actuated by means of inverse kinematics, control can still be achieved using direct drive. Passing this kind of singularity in this control scheme results in the reconfiguration of the parallel mechanism.

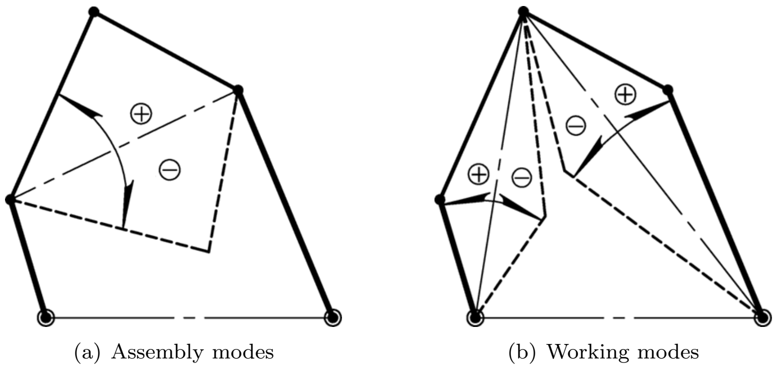


FIGURE 20.2
5R kinematic configurations.

20.2 The Path Planning Problem

Path planning between two configurations of a manipulator in the configuration space ($\mathcal{C}$-space) is a fundamental task in robotics, with varying degrees of difficulty depending on $\mathcal{C}$-space complexity [1]. The subject received much attention with advances in robot architecture such as parallel mechanisms, highly-redundant robots, and applications in mobile robotics, creating more complex and dynamically changing $\mathcal{C}$-space topologies [2]. Sampling-based planning methods such as rapidly exploring random trees (RRT) [4] and probabilistic road maps (PRM) [5] were developed as a solution to handle complex $\mathcal{C}$-spaces [6]. However, planners of this nature are probabilistically complete, meaning a valid solution is not guaranteed ahead of time. Highly complex or higher dimensional $\mathcal{C}$-space manifolds can contain narrow channels which typically reduce the chance of valid connectivity through that region. This can result in very high path planning times with little chance of a successful plan.

Path planning for parallel manipulators is a challenging task, due to non-linear constraints imposed by closed-loop kinematic chains [7]. This creates a $\mathcal{C}$-space manifold that is highly complex in nature due to the existence of singularities [8]. In addition, parallel manipulators have two types of singularities, called serial singularities, and parallel singularities. Serial singularities are analogous to those encountered earlier in this textbook. Parallel singularities, on the other hand, pose a significant threat to the controllability of a parallel manipulator, as encountering them results in total loss of control of the end effector. In some texts, this is called a Type II singularity [9], with serial singularities being Type I. This places further emphasis on planners to avoid Type II singularities at all costs, which is a non-trivial problem in $\mathcal{C}$-space planning for parallel manipulators.

Several works confront the Type II singularity problem in the $\mathcal{C}$-space, with the 3-R$\underline{\text{P}}$R parallel mechanism becoming the focus for many researchers for its unique $\mathcal{C}$-space manifold [10, 11, 12, 13, 14, 15]. These path planning strategies focus on unique features of the $\mathcal{C}$-space that allow singularity-free path planning in the joint space, which trivialises the Type I singularity problem [16]. While completely singularity-free path planning was validated in the joint space-generated $\mathcal{C}$-space, it required significant knowledge of specific features in the $\mathcal{C}$-space manifold called cusp points [17], which were the main focus and mechanism for singularity-free path planning in these works. Therefore, these path planning schemes cannot be generalised for all $\mathcal{C}$-spaces.

A few sampling-based strategies have been proposed for the general path planning of parallel manipulators [18, 19, 6, 20, 21], which verify the feasibility of this type of planner in complex and higher dimensional $\mathcal{C}$-spaces, but do not to account for any singularities. Other planners that consider the $\mathcal{C}$-space in task variables were proposed in [7, 22], where Type I singularities were assumed to be controllable. However, path planning in this $\mathcal{C}$-space becomes complicated in the presence of multiple inverse kinematic solutions, where the manipulator's working modes introduce extra layers and internal boundaries in the $\mathcal{C}$-space. There is no unified method of how to handle Type I singularities in this type of $\mathcal{C}$-space [23].

The method of $\mathcal{C}$-space decomposition plays a large role in the efficiency of path planning, especially when considering the applicability in online or offline multi-query planning. In [7, 15], the $\mathcal{C}$-space is constructed in its entirety and deterministically. This method of $\mathcal{C}$-space generation breaks the problem into smaller spaces called charts, where it is assumed the set of configurations it contains are connected. Chart interconnectivity is represented as a graph called an atlas, and global planning utilises the atlas to find a path between charts that contain the start and goal positions, creating a hierarchical path planning scheme. In addition, [7] does not require closed-form parametrisation of the $\mathcal{C}$-space, which can significantly reduce the computational intensity if the singularity profile is complex. While full $\mathcal{C}$-space construction allows the use of complete planners for deterministic path queries, these works suffer from higher dimensionality problems as the number of discretised elements grows exponentially. Other works [20, 21] randomly sample and grow parts of the $\mathcal{C}$-space using its tangent space to avoid its full reconstruction through RRT methods (known as AtlasRRT). While this method is suitable for higher dimensional cases, path planning is no longer deterministic, which can lower planning efficiency in highly complex higher dimensional $\mathcal{C}$-spaces such as that found in [24].

With these problems in mind, we introduce a novel $\mathcal{C}$-space construction method for general non-redundant parallel mechanisms that is fast and leaves a small memory footprint, even in higher-dimensional spaces. This $\mathcal{C}$-space allows for deterministic path planning, which carries significant advantages in speed and completeness. In this method, the $\mathcal{C}$-space is constructed with joint space variables to eliminate the complexities of handling Type I singularities. Points are randomly sampled in the $\mathcal{C}$-space and connected using Delaunay triangulation (DT), which will mitigate problems associated with planning in higher dimensions compared to grid discretisation. DT also ensures points are connected to their nearest neighbour, increasing the likelihood of connectivity between points in the $\mathcal{C}$-space. Singularities in the $\mathcal{C}$-space are found using root counting algorithms on the closed-form forward kinematic equations in univariate polynomial form, which is significantly faster than numerically solving the root of the determinant of a Jacobian matrix. Using a method similar to [15], the $\mathcal{C}$-space is broken down into charts based on the location of the singularity locus. Finally, using simple iterative connectivity algorithms, the charts are connected to form a chart, representing the atlas of the $\mathcal{C}$-space. With the atlas (graph) showing global connectivity amongst charts, global path planning queries are very efficient but, more importantly, deterministic so that no time is wasted waiting for an improbable path query to fail. Examples of $\mathcal{C}$-space construction using this method are provided for the three-DoF 3-RRR parallel manipulator.

20.3 Methodology

Because a parallel manipulator's workspace is so complex due to its complicated singularity profile, blindly applying a sample-based path planning algorithm will not yield a time-feasible result. An algorithm like RRT will perform many checks for path validity to construct its tree, and if its search resolution is not fine enough to capture parallel singularities, some paths may actually contain singularities, resulting in disastrous results for the robot when the path is executed. To completely eliminate this, we desire workspace definitions that are singularity-free. Thus, we propose the following methodology to ensure efficient and successful path planning for parallel manipulators:

Setup Phase

This phase only needs to occur once, with the goal of simplifying the path planning procedure.

1. Separate the manipulator's complicated workspace into unique, singularity-free workspace regions called *charts*
2. Find singularity-free paths between these charts called *links*
3. Represent the workspace in a connected chart called an *atlas*, where charts are represented as nodes with links representing singularity-free paths between them.

Path Planning Phase

For a path defining our initial and goal manipulator configurations, determine path feasibility and create a safe, singularity-free path for the end effector.

1. Determine which charts contain our initial and goal configurations.
2. If the two configurations exist but are on two separate charts, determine if they are connected on the atlas.
3. If they are connected, perform path planning *in each chart*, utilising the singularity-free links as defined in the atlas.
4. Once complete, a complete path plan consists of:
    (a) A path from the initial configuration to the link to the next chart;
    (b) From the link in the next chart, a path to the next link to the next chart, repeating this until the chart containing the goal configuration is found; and
    (c) A path from the last link to the goal configuration.

20.4 Generating Charts

The goal is to develop an efficient planning strategy to generate singularity-free paths for parallel manipulators. To ensure singularity-free paths, we should evaluate the mechanism's *singularity locus*. Once found, we can determine regions of the manipulator's reachable workspace that are fully connected and singularity-free, which we call a *graph*. These charts are critical for the efficiency of parallel manipulator path planning in higher dimensional spaces, as they allow the planner to determine path feasibility in advance.

20.4.1 Singularity Locus

Figure 20.3 shows a plot of $\det(J_x)$ of a parallel manipulator along a linear path between two points in the configuration space. To the left of the chart, the manipulator has four kinematic solutions (assembly modes). However, as we traverse along the path, we find that two of those solutions approach a parallel singularity, then eventually disappear. The point at which this happens is called a parallel singularity. However, we also observe that two assembly modes remain singularity-free along the path. Therefore, we refer to this point as a *singularity locus*, because a parallel singularity exists for certain configurations along the path, but not for others.

The formal definition of a singularity locus describes the *possible* location of a singularity in an n-dimensional $\mathcal{C}$-space. Because $\det J_x = 0$ is a projection onto the n-dimensional $\mathcal{C}$-space, it is reduced by one dimension, i.e., the singularity locus of a two-dimensional $\mathcal{C}$-space is represented by a line (Figure 20.4), or a surface when considering a three-dimensional $\mathcal{C}$-space.

20.4.2 Identifying a Chart

We define a *graph* as a region of workspace that is guaranteed to be singularity-free. Because the singularity locus defines regions of the workspace in which a singularity *may* exist, implicitly, a chart *will* be bounded by a singularity locus.

If we inspect further, a singularity locus also indicates where kinematic solutions (assembly modes) disappear or reappear. With this information, we can draw the conclusion that the singularity locus exists where the number of forward kinematic solutions changes. We can determine the number of kinematic solutions by counting the real roots of the forward kinematics univariate polynomial (20.4). Therefore, we can reduce this problem to a simple root-counting exercise, thus avoiding the computational costs of finding the roots of $\det J_x = 0$. Furthermore, we can avoid explicitly solving the roots by utilising fast root-counting algorithms, such as Sturm's theorem to count the real roots of a univariate polynomial within an interval. For example, using a bisection method can pinpoint exactly where a change in the number of real solutions occurs, finding the exact location of the singularity locus in the process.

Using Sturm's theorem with the bisection method for precise root counting, we can generate maps of kinematic solutions such as Figure 20.5. This shows the number of assembly modes for a 3-RRR parallel manipulator over a two-dimensional discretised grid (θ_2, θ_3), and with θ_1 held constant at 36°. We observe that there are clear continuous boundaries between regions of differing numbers of assembly modes, which we mark as the singularity locus. Again, we assume that within these boundaries, the workspace is singularity-free by

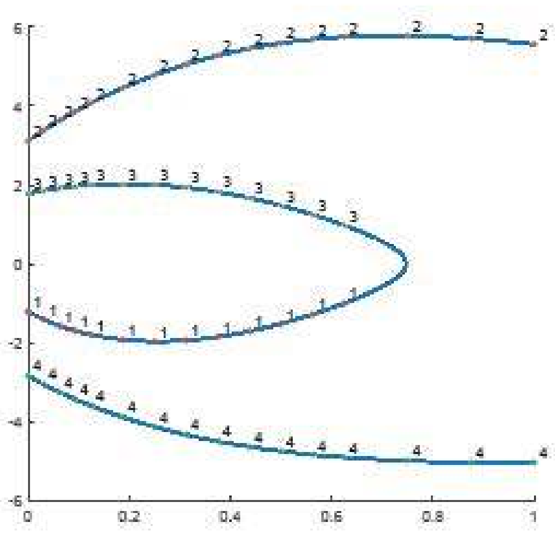


FIGURE 20.3
Plot of $\det J_x$ of a linear path between 4-solution and 2-solution vertices. The bisection method is used for finding the location of the singularity locus.

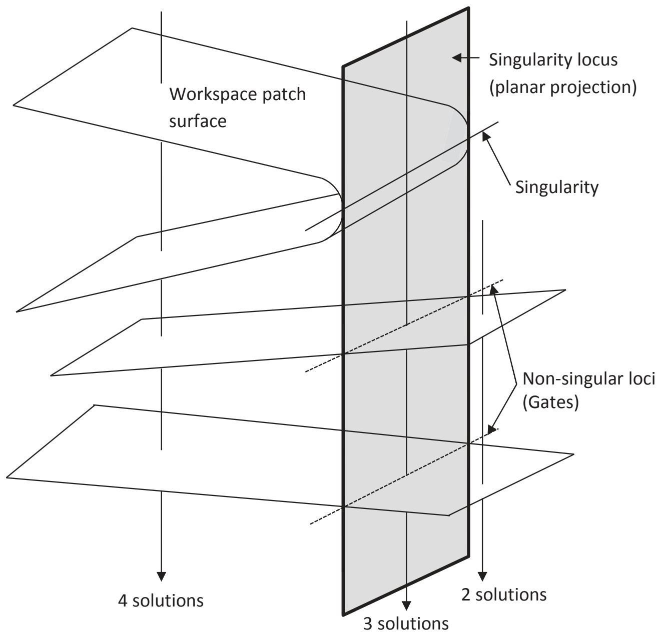


FIGURE 20.4
The $\mathcal{C}$-space manifolds that satisfy $\mathbf{F}(\mathbf{x}, \mathbf{q}) = 0$ of a two-dimensional joint space $\mathbf{q}$, with projection of the singularity locus.

the definition of an assembly mode.

20.4.3 Constructing a Chart's $\mathcal{C}$-space

There are two possible methods to build the connected region within a chart: grid discretisation and random sampling. We will utilise random sampling here, because parallel manipulators usually operate in higher dimensional workspaces, which makes grid discretisation highly inefficient.

Random-sampling the joint space variables used in this method of $\mathcal{C}$-space generation shares some similarities with generating a probabilistic roadmap [18], where the more samples are taken, the clearer the $\mathcal{C}$-space becomes in identifying features and obstacles (singularities). This paradigm allows this method of $\mathcal{C}$-space generation to remain viable in higher-dimensional spaces, as the number of samples in this space is user-defined, rather than exponentially increasing with each additional dimension. Increasing the number of samples of the joint space increases the resolution of the $\mathcal{C}$-space, at the expense of computation speed and memory footprint.

Each point sampled in the joint space is projected to a *single* in the $\mathcal{C}$-space is called a *vertex*, ignoring the presence of multiple assembly modes for generality for the time being. Then each vertex is initially connected via *edges* using Delaunay triangulation. This guarantees that all vertices are connected, and that a vertex is connected to at least its closest neighbour. Each edge in the $\mathcal{C}$-space represents a path between two points; a linear interpolation in the joint space coordinates, which will be checked later for path validity.

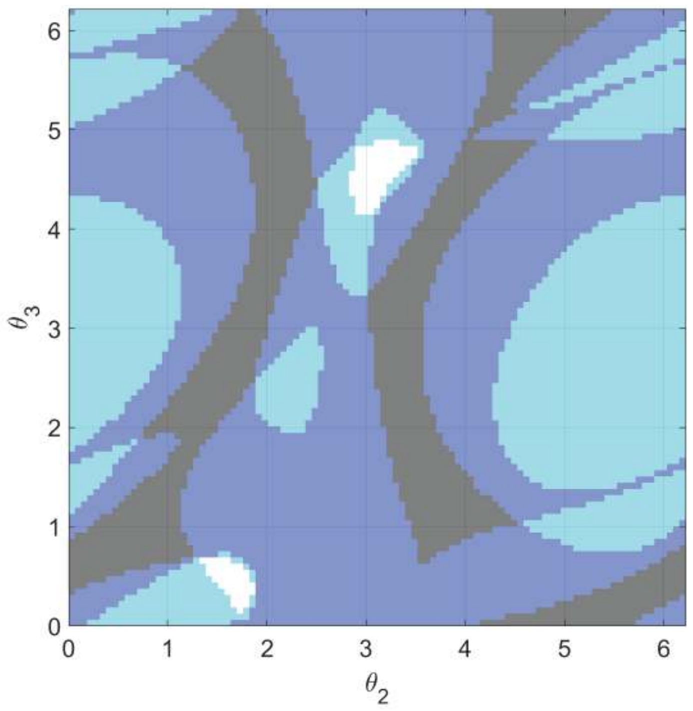


FIGURE 20.5
Number of assembly modes of a 3-RRR parallel manipulator in the configuration space, where θ_1 is held constant.

20.5 Constructing the Atlas

Recall that only some assembly modes will encounter singularity across a singularity locus. Using the continuity condition as defined in Section 20.6.2, we can verify which kinematic configurations allow passage across a singularity locus. This process can be visualised in Figure 20.8. The sampled points along the boundary of a chart (same coloured regions) are called gates. They are coloured dark blue for the cyan region (two kinematic solutions), cyan in the dark blue regions (four kinematic solutions), and yellow in the grey regions (six kinematic solutions). For each gate that is adjacent to another gate with *increasing* number of kinematic solutions, their paths are checked for singularity-free connectivity.

As we find connections between charts, we can represent this as a map of nodes and links, where a node is a chart, and links are the feasible paths between them. This map of connected charts is called an *atlas*, which represents the overall reachable workspace of a parallel manipulator as a chart of connected nodes. An example of this is shown in Figure 20.6, where each patch follows the naming convention

$$\boxed{\pm x.y.z} \tag{20.13}$$

where

- $\pm$ indicates the positive or negative aspect,
- x indicates the number of DK solutions (assembly modes) associated with this chart,

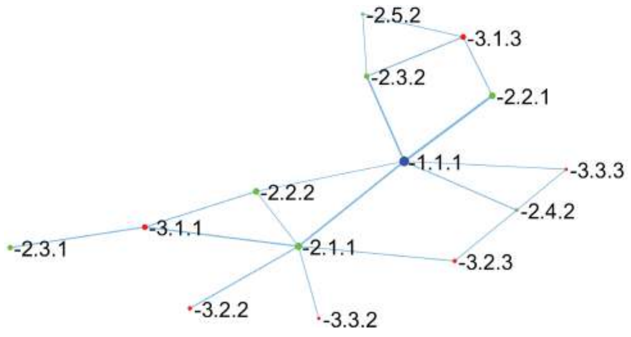


FIGURE 20.6
An example atlas in the *negative aspect only*.

- y indicates the unique chart ID for x, ordered descending of the number of vertices the chart contains, and
- z indicates the assembly mode ID for the particular chart associated with y.

If one were to determine whether a path between points in configuration space exists, we must perform the following steps:

1. Determine which charts contain starting and ending configurations.
2. Determine the location of the nodes in the atlas associated with these charts.
3. Perform a search to see if a path exists between the two nodes.
4. If a path is found in the atlas, we know a path between these configurations exists.

For example, we want to perform an assembly mode change with a starting configuration contained in chart $\boxed{-3.1.1}$. This means the ending configuration is contained in chart $\boxed{-3.1.3}$, as this refers to the same chart ID y, but in differing assembly mode ID z. Because these two charts are connected in the atlas, (e.g., via $\boxed{-2.2.2}$, $\boxed{-1.1.1}$, and $\boxed{-2.2.1}$), then we know a singularity-free path can exist. The advantage of utilising this atlas is that it determines path feasibility before computation can actually begin very efficiently. If a proposed path plan contained starting and ending configurations that were not connected by the atlas, the planning is immediately forfeited. This is incredibly useful for sample-based planners, as they will continuously search for a path that does not exist until a timeout condition is reached.

20.6 Case Study: A 3-RRR Parallel Manipulator

To demonstrate the efficiency of this planning method, we apply it to the 3-RRR parallel manipulator (Figure 20.7). The variable parameters that define the manipulator are:

$$OB = (OB_x, OB_y) \quad OC = (OC_x, OC_y) \quad MB = (MB_x, 0) \quad MC = (MC_x, MC_y) \tag{20.14}$$

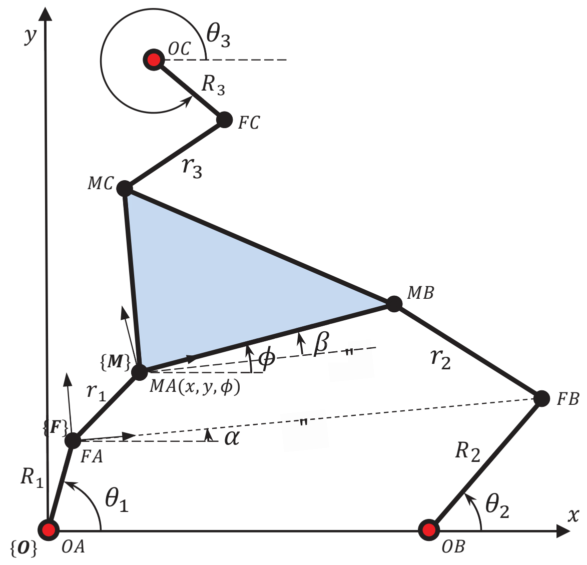


FIGURE 20.7
A 3-RRR parallel mechanism.

where the origin of the base frame $\{O\}$ is $OA = (0, 0)$. The orientation of the base frame is arbitrary, but it is usually aligned such that the parameter $OB_x = 0$. The dimensions of the moving platform are fully defined by MB and MC.

A secondary frame $\{F\}$ is defined at the passive joint of the first chain FA. This frame is located at position (x_{FA}, y_{FA}, α), relative to the base frame $\{O\}$. Frame $\{M\}$ is positioned on the moving platform such that $MB = (x_{MB}, 0)$ and $MC = (x_{MC}, y_{MC})$ in $\{M\}$. This frame is located at position (x_{MA}, x_{MA}, β), relative to frame $\{F\}$.

The end-effector is described by point ${}^{O}MA(x, y, \phi)$, where $\mathbf{x} = \mathbf{x}(x, y, \phi)$ gives the description of MA in planar position and rotation with respect to the base frame at whose origin is at OA.

Parameters R and r describe the active and passive link lengths, respectively.

20.6.1 Forward Kinematics

In general, there are up to six forward kinematic solutions for this class of manipulator [10], which is solvable in univariate polynomial form with the assistance of kinematic mapping [28]. Using kinematic mapping, we can avoid working with non-linear simultaneous equations generated by forward kinematics for each leg. Instead, the coefficients of the univariate polynomial can be generated straight from the co-ordinates of MB and MC in $\{M\}$ (which are constant), FB and FC in $\{F\}$ and $\mathbf{q} = (\theta_1, \theta_2, \theta_3)$. The univariate polynomial in the mapping variable X_3 [28] is given as

$$A_6X_3^6 + A_5X_3^5 + A_4X_3^4 + A_2X_3^2 + A_1X_3 + A_0 = 0 \tag{20.15}$$

where

$$A_6 = e_1^2 + a_6d_5^2 + d_1^2 \tag{20.16}$$

$$A_5 = 2d_1d_2 + 2a_6d_5d_6 + 2e_1e_2 \tag{20.17}$$

$$A_4 = a_6 d_5^2 + 2d_1 d_3 + e_2^2 + 2e_1 e_3 + d_2^2 + 2a_6 d_5 d_7 + a_6 d_6^2 \tag{20.18}$$
$$A_3 = 2d_1 d_4 + 2e_2 e_3 + 2a_6 d_6 d_7 + 2a_6 d_5 d_6 + 2d_2 d_3 + 2e_1 e_4 \tag{20.19}$$
$$A_2 = 2e_2 e_4 + e_3^2 + d_3^2 + a_6 d_7^2 + 2d_2 d_4 + 2a_6 d_5 d_7 + a_6 d_6^2 \tag{20.20}$$
$$A_1 = 2e_3 e_4 + 2d_3 d_4 + 2a_6 d_6 d_7 \tag{20.21}$$
$$A_0 = a_6 d_7^2 + e_4^2 + d_4^2 \tag{20.22}$$

$$K_a = -r_a^2 \tag{20.23}$$
$$K_b = {X_{FB}}^2 - r_b^2 \tag{20.24}$$
$$K_c = {X_{FC}}^2 + {Y_{FC}}^2 - r_c^2 \tag{20.25}$$

$$a_6 = \tfrac{1}{4} K_a \tag{20.26}$$
$$b_1 = -X_{FB} - X_{MB} \tag{20.27}$$
$$b_4 = -X_{FB} + X_{MB} \tag{20.28}$$
$$b_6 = \tfrac{1}{4} 2X_{FB} X_{MB} + K_b + {X_{MB}}^2 \tag{20.29}$$
$$b_7 = \tfrac{1}{4} {X_{MB}}^2 - 2X_{FB} X_{MB} + K_b \tag{20.30}$$
$$c_1 = -X_{FC} - X_{MC} \tag{20.31}$$
$$c_2 = -Y_{FC} - Y_{MC} \tag{20.32}$$
$$c_3 = -Y_{MC} + Y_{FC} \tag{20.33}$$
$$c_4 = -X_{FC} + X_{MC} \tag{20.34}$$
$$c_5 = X_{FC} Y_{MC} - Y_{FC} X_{MC} \tag{20.35}$$
$$c_6 = \tfrac{1}{4} \left(2Y_{FC} Y_{MC} + 2X_{FC} X_{MC} + K_c + {Y_{MC}}^2 + {X_{MC}}^2 \right) \tag{20.36}$$
$$c_7 = \tfrac{1}{4} \left({X_{MC}}^2 + {Y_{MC}}^2 - 2X_{FC} X_{MC} + K_c - 2Y_{FC} Y_{MC} \right) \tag{20.37}$$
$$d_1 = -c_2 b_6 + c_2 a_6 \tag{20.38}$$
$$d_2 = c_4 a_6 - c_4 b_6 + c_6 b_4 - a_6 b_4 \tag{20.39}$$
$$d_3 = c_5 b_4 - c_2 b_7 + c_2 a_6 \tag{20.40}$$
$$d_4 = -a_6 b_4 - c_4 b_7 + c_7 b_4 + c_4 a_6 \tag{20.41}$$
$$d_5 = c_2 b_1 \tag{20.42}$$
$$d_6 = -c_1 b_4 + c_4 b_1 \tag{20.43}$$
$$d_7 = -c_3 b_4 \tag{20.44}$$
$$e_1 = b_6 c_1 - b_1 c_6 + b_1 a_6 - a_6 c_1 \tag{20.45}$$
$$e_2 = b_6 c_3 - a_6 c_3 - b_1 c_5 \tag{20.46}$$
$$e_3 = -b_1 c_7 + b_1 a_6 - a_6 c_1 + b_7 c_1 \tag{20.47}$$
$$e_4 = b_7 c_3 - a_6 c_3 \tag{20.48}$$

Note that

$$MB = [X_{MB}, 0] \qquad MC = [X_{MC}, Y_{MC}] \tag{20.49}$$

are constant quantities with respect to frame $\{M\}$ while

$$FB = [X_{FB}, 0] \qquad FC = [X_{FC}, Y_{FC}] \tag{20.50}$$

are variables with respect to $\{F\}$. In order to find these quantities, some linear algebra work is required.

Let's define the following F coordinates with respect to the base frame:

$$^OFA = [^OX_{FA}, ^OY_{FA}] \qquad ^OFB = [^OX_{FB}, ^OY_{FB}] \qquad ^OFC = [^OX_{FC}, ^OY_{FC}] \tag{20.51}$$

Therefore ^OFA, ^OFB, and ^OFC are calculated by

$$^OX_{FA} = R_3 \cos\theta_1 \tag{20.52}$$
$$^OX_{FA} = R_3 \cos\theta_1 \tag{20.53}$$
$$^OX_{FB} = R_3 \cos\theta_2 + OB_x \tag{20.54}$$
$$^OY_{FB} = R_2 \sin\theta_2 + OB_y \tag{20.55}$$
$$^OX_{FC} = R_3 \cos\theta_3 + OC_x \tag{20.56}$$
$$^OY_{FC} = R_3 \sin\theta_3 + OC_y \tag{20.57}$$

The transformation matrix from frame $\{O\}$ to frame $\{F\}$ is

$$^O\mathbf{T}_F = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & ^OX_{FA} \\ \sin(\alpha) & \cos(\alpha) & ^OY_{FA} \\ 0 & 0 & 1 \end{bmatrix} \tag{20.58}$$

where

$$\alpha = \text{Atan2}\left(^OY_{FB} -^O Y_{FA}, ^O X_{FB} -^O X_{FA}\right) \tag{20.59}$$

We can now calculate the parameters in (20.50).

$$X_{FB} = \sqrt{(^OY_{FB} -^O Y_{FA})^2 + (^OX_{FB} -^O X_{FA})^2} \tag{20.60}$$

and

$$\begin{gathered} FC = {}^F\mathbf{T}_O \, {}^OFC \\ \therefore \begin{bmatrix} X_{FC} \\ Y_{FC} \\ 1 \end{bmatrix} = {}^O\mathbf{T}_F T^{-1} \begin{bmatrix} ^OX_{FC} \\ ^OY_{FC} \\ 1 \end{bmatrix} \end{gathered} \tag{20.61}$$

We now have all the parameters required to find the coefficients for the univariate polynomial (20.15). We will find up to six roots in the 6th-order polynomial for the variable X_3. The conversion of variable X_3 to output variables $(^FX_{MA}, ^FY_{MA}, \beta)$ in $\{F\}$ is performed by

$$\beta = 2 \text{ Atan2}(X_3, 1) \tag{20.62}$$
$$^FX_{MA} = 2\left(\frac{X_1X_3 + X_2}{X_2^3 + 1}\right) \tag{20.63}$$
$$^FY_{MA} = 2\left(\frac{X_2X_3 - X_1}{X_2^3 + 1}\right) \tag{20.64}$$

where

$$X_1 = \frac{d_1X_3^3 + d_2X_3^2 + d_3X_3 + d_4}{d_5X_3^2 + d_6X_3 + d_7} \tag{20.65}$$
$$X_2 = \frac{e_1X_3^3 + e_2X_3^2 + e_3X_3 + e_4}{d_5X_3^2 + d_6X_3 + d_7} \tag{20.66}$$

Hence

$$^{F}\mathbf{T}_M = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & ^{F}X_{MA} \\ \sin(\beta) & \cos(\beta) & ^{F}Y_{MA} \\ 0 & 0 & 1 \end{bmatrix} \tag{20.67}$$

Now that we have two transformation matrices $^{O}\mathbf{T}_F$ and $^{F}\mathbf{T}_M$ in Equations (20.58) and (20.67), respectively, we can find ^{O}MA and its given angle ϕ from the resulting transformation matrix (20.68)

$$^{O}\mathbf{T}_M = {}^{O}\mathbf{T}_F\,{}^{F}\mathbf{T}_M \tag{20.68}$$

The parameters for the 3-RRR used in this example are $OA(x, y) = (0, 0)$, $OB = (6, 0)$, $OC = (3, 2)$, $X_{MB} = 4$, $X_{MC} = 1$ and $Y_{MC} = 5$, where leg lengths $R = (3, 3, 3)$ and $r = (3, 4, 5)$ arbitrary unit lengths.

20.6.2 Continuity Conditions

Determining the continuity of adjacent vertex accurately in multiple-solution situations is very important in this path planning application. Not only is it necessary for chart construction, but it also determines which charts are continuous across a singularity locus, which forms the basis of atlas construction. Because there are multiple forward kinematics solutions, we must define a continuity condition that accurately selects the correct solution, given any number of DK solutions between postures.

Because the moving platform of the 3-RRR is defined by three passive joints defined in frame $\{M\}$, i.e., MA, MB, and MC, (Figure 20.7), due to their dependence on spatial coordinates only, we can set our continuity conditions based on the delta of passive joints M. That is, we can condense the continuity condition into a comparison of a single number between the original configuration O, and the next configuration n, where n denotes the n-th solution in the adjacent cell by the following equation:

$$\Delta_n = |MA_n - MA_O| + |MB_n - MB_O| + |MC_n - MC_O| \tag{20.69}$$

The continuity conditions between configurations O and n is therefore

$$\min(\Delta_n) \tag{20.70}$$

where n denotes the best forward kinematic solution in the next posture. An example of random sampling over a 3-RRR parallel manipulator $\mathcal{C}$-space with connected vertices is shown in Figure 20.8. The black circles indicate sampled vertices, black edges indicate connected paths within a chart, and solid dots indicate the precise location of the singularity locus using the bisection method.

20.6.3 Singularity Analysis

Researchers have shown that the 3-RRR manipulator is in singularity configuration when the vectors formed by passive links coincide in a single point (Figure 20.9)[12]. Knowing this, we can conduct singularity analysis with the geometric method. This can result in a closed-form solution that is simpler than the traditional Jacobian and determinant method in that it does not involve very complicated algebraic equations. Derived from the kinematic and straight-line equations, a parallel singularity occurs when the following equation is satisfied:

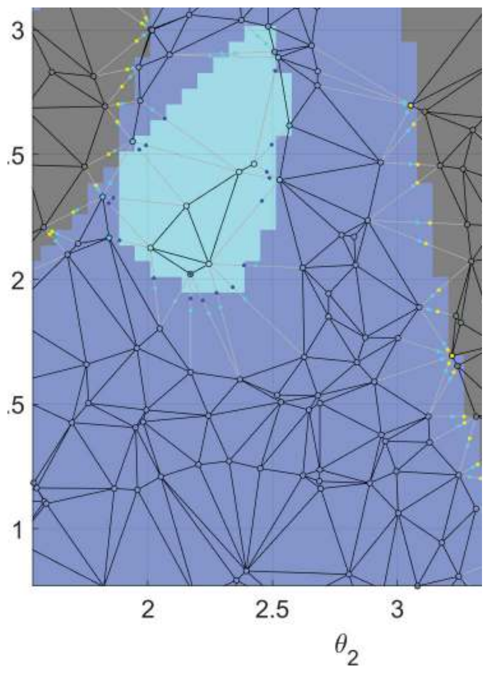


FIGURE 20.8
A small segment of 3-RRR two-dimensional $\mathcal{C}$-space with randomly sampled vertices and connected edges.

$$\begin{aligned}Y_{FB}\cos(\theta_2)\sin(\theta_1-\theta_3)+\cos(\theta_1)\sin(\theta_2)(Y_{FC}\cos(\theta_3)\\ +(X_{FB}-X_{FC})\sin(\theta_3))-\sin(\theta_1)(X_{FB}\cos(\theta_3)\sin(\theta_2)\\ +\cos(\theta_2)(Y_{FC}\cos(\theta_3)-X_{FC}\sin(\theta_3)))=0\end{aligned} \tag{20.71}$$

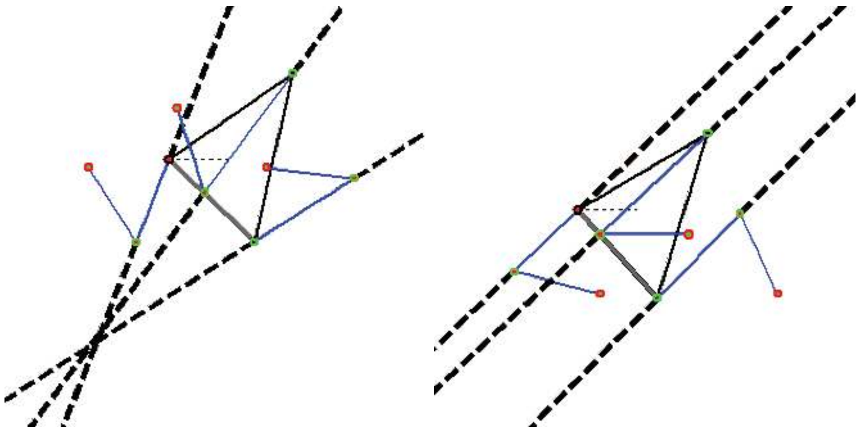

(a) Three axes coinciding. (b) Three axes parallel.

FIGURE 20.9
Geometric singularities of the 3RRR parallel manipulator.

We know that parallel singularities result in the loss of control of the end effector in a parallel manipulator. This has implications for the connectivity of its assembly modes. Figure 20.10 shows the six assembly modes for a particular joint space configuration. We observe that the sign of $\det(J_x)$ for each of these assembly modes defines its aspect, in which there are three in each aspect. While not mathematically proven, we observe that for an even number of assembly modes, there are always an even number of assembly modes in the positive and negative aspects.

20.6.4 Path Planning

We wish to plan a path for the 3-RRR manipulator, such that we achieve a change in the assembly mode. That is, a path in which the starting and ending joint configuration remains the same, but the platform pose (task space point) is different. In addition, we will constrain joint θ_1 to be constant, but the other joints are unconstrained and free to rotate continuously. The chosen joint configuration to start and end the path is

$$\begin{aligned}\mathbf{q} &= \begin{bmatrix}\theta_1 & \theta_2 & \theta_3\end{bmatrix}\\ &= \begin{bmatrix}0.6283 & 3.3671 & 3.3382\end{bmatrix}\end{aligned} \tag{20.72}$$

with the platform pose for assembly mode 1 defined as

$$\begin{bmatrix}MA_1\\ MB_1\\ MC_1\end{bmatrix} = \begin{bmatrix}2.9869 & -1.1839\\ 6.9641 & -1.6105\\ 4.5144 & 3.6809\end{bmatrix} \tag{20.73}$$

and assembly mode 2 defined as

$$\begin{bmatrix}MA_2\\ MB_2\\ MC_2\end{bmatrix} = \begin{bmatrix}2.3247 & -1.2349\\ 0.3156 & 2.2240\\ -2.5012 & -2.8815\end{bmatrix} \tag{20.74}$$

These two configurations are shown in Figure 20.10 as assembly modes 3 and 2, respectively.

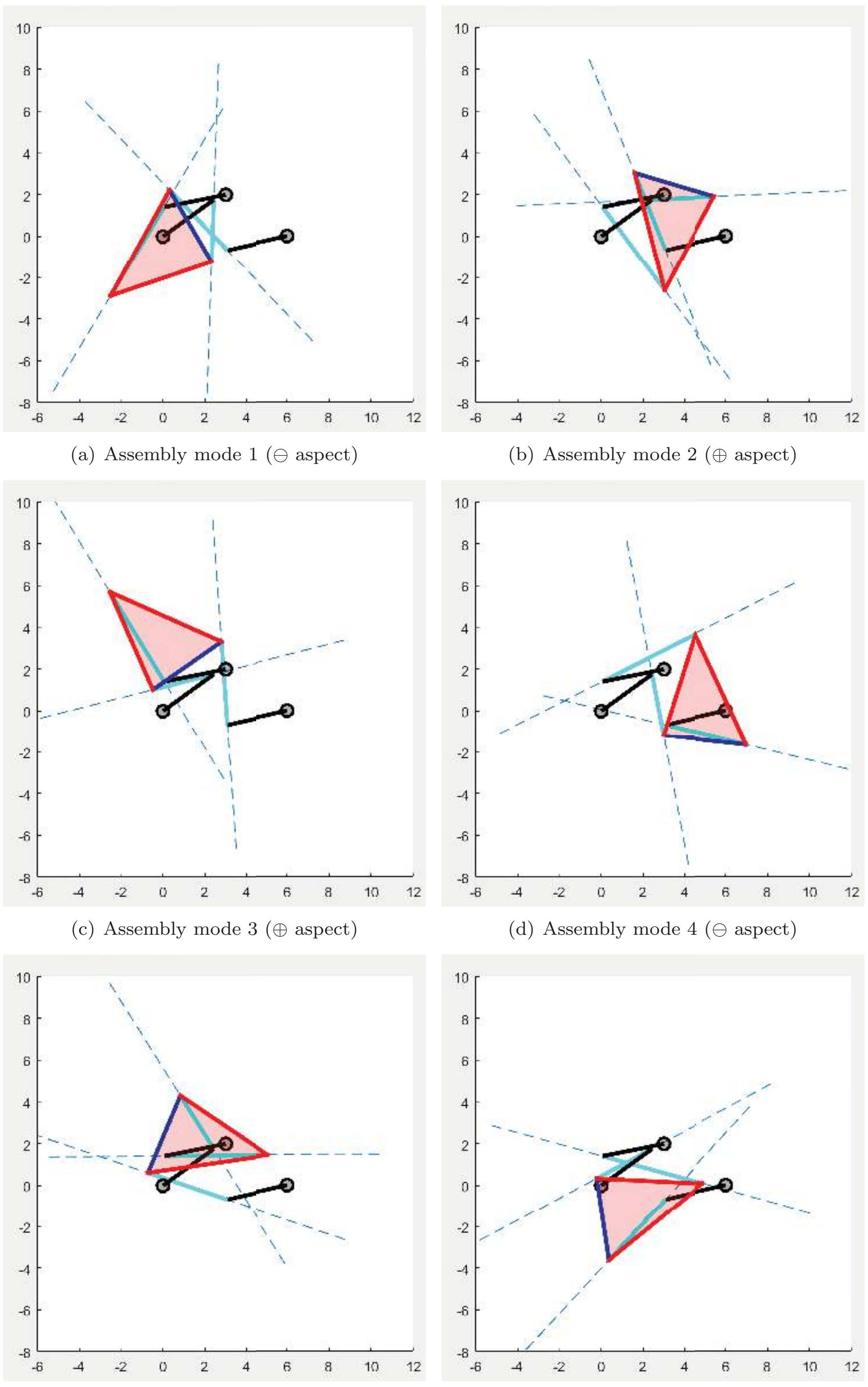
20.6.4.1 Charts and Atlas for $\theta_1 = 36°$

The randomly sampled joint space variables (as vertices), connected paths (as edges), links and gates are shown in Figure 20.11. Approximately 1000 points in the joint space were sampled. The colour of the map indicates the number of assembly modes (forward kinematic solutions) in that region, with white, cyan, blue, and grey indicating 0, 1, 2 and 3 forward kinematic solutions, respectively.

Figure 20.12 shows the 3-RRR atlas that make up the overall $\mathcal{C}$-space when θ_1 is constrained to 36°. The size of each chart (indicated by dots) represents how many connected sampled points it contains. This is indicative of chart size, but is not an accurate measure of the true size of the chart's reachable workspace. The thickness of each edge indicates how many valid paths connect to the adjacent chart. This is indicative of the *safety* of the transition between charts, as narrower paths tend to indicate closer proximity to a parallel singularity.

20.6.4.2 Path Plan

The first step is to identify the charts that contain configurations 1 and 2 that correspond to the starting pose $\mathbf{q}$. In this example, configuration 1 is contained in $\boxed{+3.2.2}$ and configuration 2 is contained in $\boxed{+3.2.1}$. According to the atlas in Figure 20.12, we can link the



(a) Assembly mode 1 ($\ominus$ aspect)

(b) Assembly mode 2 ($\oplus$ aspect)

(c) Assembly mode 3 ($\oplus$ aspect)

(d) Assembly mode 4 ($\ominus$ aspect)

(e) Assembly mode 5 ($\oplus$ aspect)

(f) Assembly mode 6 ($\ominus$ aspect)

FIGURE 20.10
Six assembly mode configurations for the same joint position.

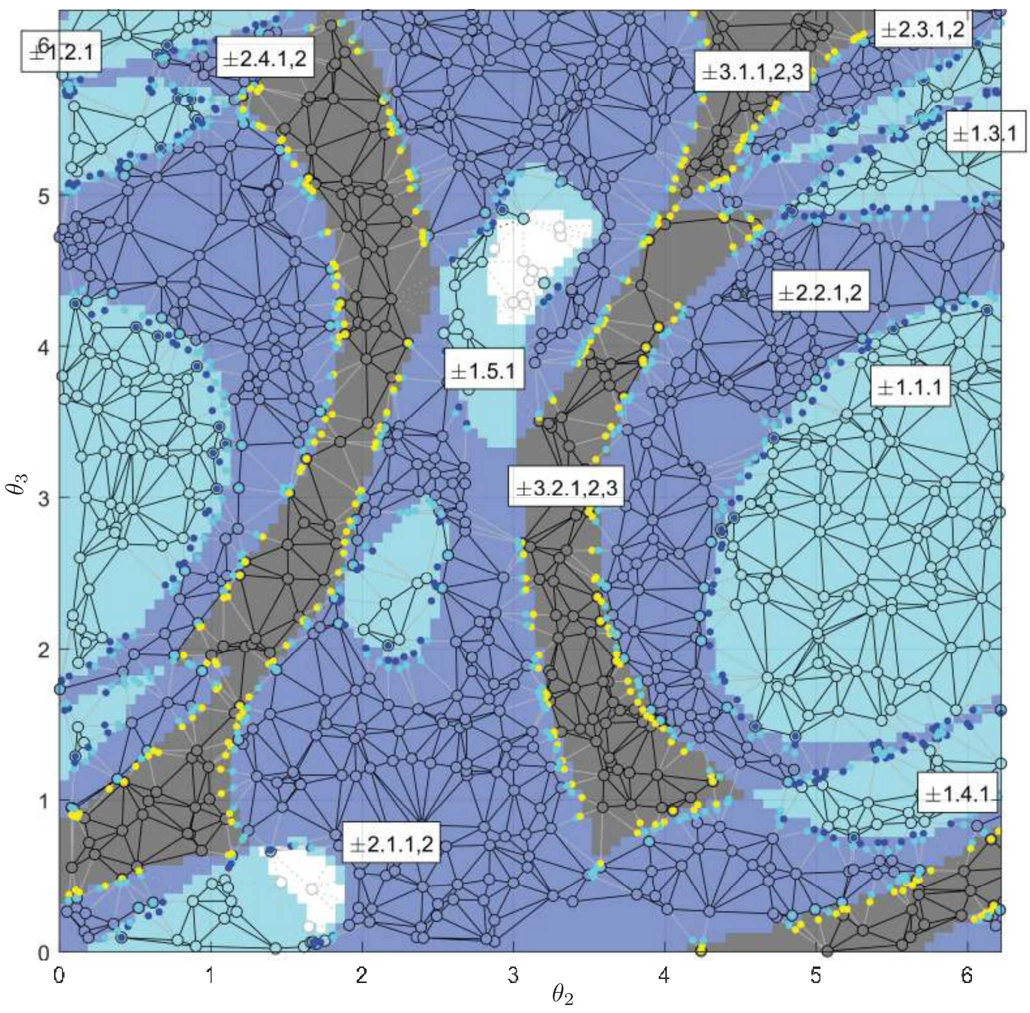


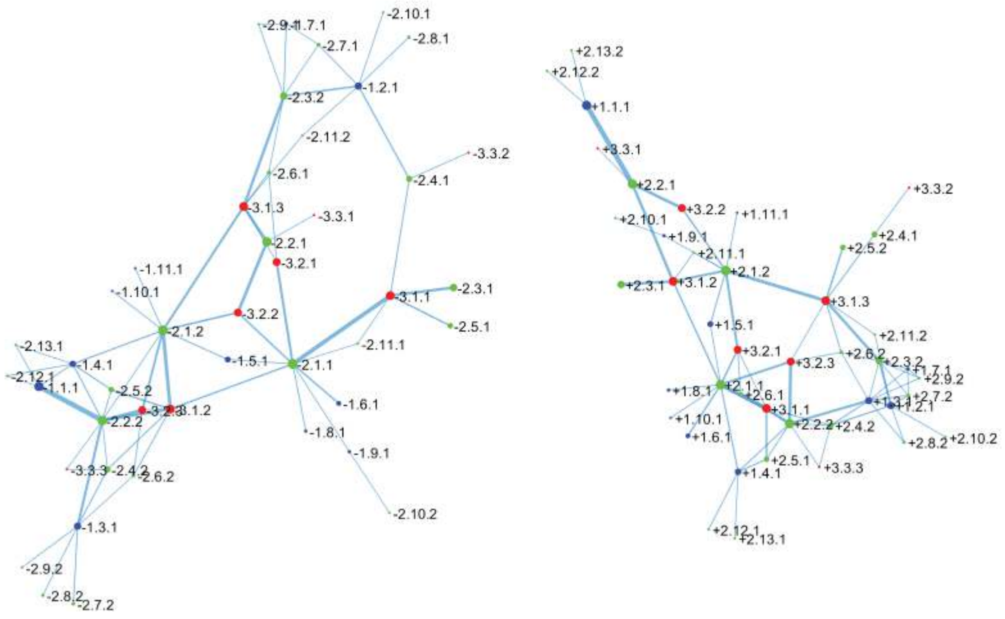
FIGURE 20.11
Chart visualisation in the 3-RRR in the 2D joint space.

two charts together with the following path:

$$\boxed{+3.2.2} \rightarrow \boxed{+2.1.2} \rightarrow \boxed{+1.5.1} \rightarrow \boxed{+2.1.1} \rightarrow \boxed{+3.2.1} \qquad (20.75)$$

With this information, localised paths in these charts can be planned, ensuring that the starting and ending points in each via chart correspond to the preceding and succeeding charts associated with their gates. Figure 20.13 shows the proposed path to be taken to perform the assembly mode change as represented as a joint space map, based on the chain of charts suggested by the atlas. Notice the starting and ending vertex are the same because assembly modes are configurations where multiple task space coordinates map to the same joint space coordinates. The colour of each region again represents the number of direct kinematic solutions (same as Figure 20.11). The IDs in the white boxes indicate the names of each chart, which can be considered as individual layers in the joint space map.

The path to be followed is in red and follows a loop in an anti-clockwise direction. The starting direction of the path is indicated by the orange arrow, and the return path is in



(a) ⊖ aspect. (b) ⊕ aspect.

FIGURE 20.12
Atlas of connected charts for the 3-RRR when $\theta_1 = 36°$.

yellow. We can see that this path takes the manipulator through the different layers in this map in a spiral, eventually finishing at its original position but in a different layer representing a different assembly mode. Traditional path planning algorithms may have difficulty handling this type of $\mathcal{C}$-space.

20.6.5 Performance

With approximately 1000 randomly sampled points in the $\mathcal{C}$-space, the atlas took approximately 30 seconds to generate on a system running MATLAB 2022b on Windows 10 with an Intel i7 2600K processor with 24 GB of RAM. Path planning queries are instantaneous with minimal memory footprint.

20.7 Conclusion

Path planning with the atlas and charts method was successfully demonstrated on the 3-RRR parallel manipulator. We showed that by randomly sampling the $\mathcal{C}$-space, memory footprint remains small, yet planning becomes probabilistically complete if additional resources are available. The atlas is highly efficient in guiding local path planning through connected charts, resulting in extremely quick and efficient planning queries. This plan-

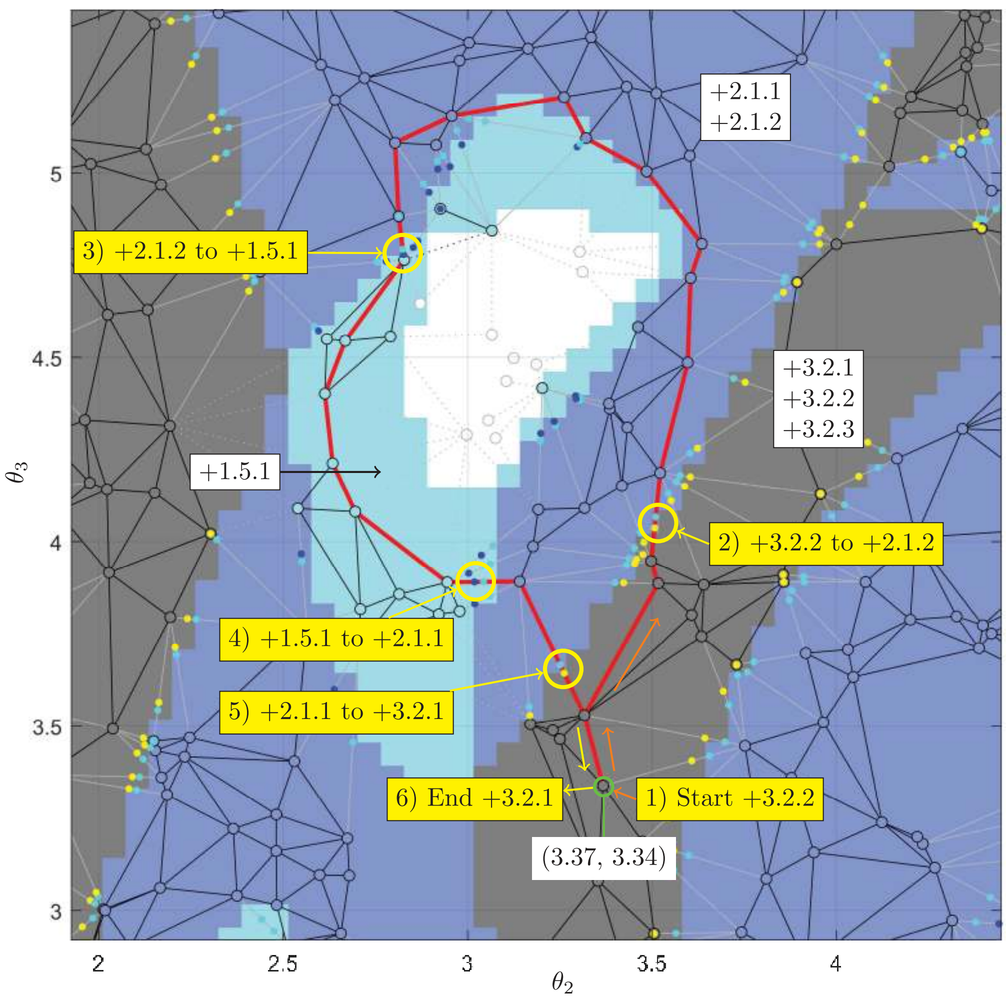


FIGURE 20.13
Proposed path for an assembly mode change as a joint space map. It follows an anti-clockwise direction, as indicated by the red edges.

ning method can handle complex path planning queries such as assembly mode changes, in which traditional sample-based planners may have trouble finding paths for due to complex parallel singularity profiles.

Bibliography

[1] J.-C. Latombe, *Robot Motion Planning.* Springer, 1991.

[2] S. M. LaValle, *Planning Algorithms.* Cambridge, 2006.

[3] L. Tsai, *Robot Analysis.* Wiley, 1999.

[4] I. A. Sucan and L. E. Kavraki, “A Sampling-Based Tree Planner for Systems with Complex Dynamics,” *IEEE Transactions on Robotics*, vol. 28, pp. 116–131, 2012.

[5] D. Hsu, J.-C. Latombe, and H. Kurniawati, “On the Probabilistic Foundations of Probabilistic Roadmap Planning,” *The International Journal of Robotics Research*, vol. 25, pp. 627–643, July 2006.

[6] L. Jaillet, J. Cortés, and T. Siméon, “Sampling-Based Path Planning on Configuration-Space Costmaps,” *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, 2010.

[7] O. Bohigas, M. E. Henderson, L. Ros, M. Manubens, and J. M. Porta, “Planning singularity-free paths on closed-chain manipulators,” *IEEE Transactions on Robotics*, vol. 29, no. 4, pp. 888–898, 2013.

[8] O. Bohigas, D. Zlatanov, L. Ros, M. Manubens, and J. M. Porta, “A General Method for the Numerical Computation of Manipulator Singularity Sets,” *IEEE Transactions on Robotics*, vol. 30, pp. 340–351, April 2014.

[9] C. Gosselin and J. Angeles, “Singularity Analysis of Closed-loop Kinematic Chains,” *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 281–290, June 1990.

[10] M. Hayes and M. Husty, “On the kinematic constraint surfaces of general three-legged planar robot platforms,” *Mechanism and Machine Theory*, vol. 38, pp. 379–394, 2003.

[11] M. Hayes, P. Zsombor-Murray, and C. Chen, “Unified Kinematic Analysis of General Planar Parallel Manipulators,” *Journal of Mechanical Design*, vol. 126, pp. 866–874, September 2004.

[12] M. Zein, P. Wenger, and D. Chablat, “Non-singular assembly-mode changing motions for 3-R$\underline{\text{P}}$R parallel manipulators,” *Mechanism and Machine Theory*, vol. 43, pp. 480–490, 2008.

[13] E. Macho, O. Alturazza, C. Pinto, and A. Hernandez, “Transitions between Multiple Solutions of the Direct Kinematic Problem,” in *Advances in Robot Kinematics: Analysis and Design* (J. Lenarčič and P. Wenger, eds.), pp. 301–310, Springer Science+Business Media B.V. 2008, 2008.

[14] M. Urízar, V. Petuya, O. Alturazza, E. Macho, and A. Hernández, “Assembly Mode Changing in the Cuspidal Analytic 3-RPR,” *IEEE Transactions on Robotics*, vol. 28, pp. 506–513, April 2012.

[15] W. Au, H. Chung, and C. Chen, “Generation of the Global Workspace Roadmap of the 3-R$\underline{\text{P}}$R using rotary disk search,” *Mechanism and Machine Theory*, vol. 78, pp. 248–262, 2014.

[16] E. Macho, O. Altuzarra, C. Pinto, and A. Hernandez, “Workspaces associated to assembly modes of the 5R planar parallel manipulator,” *Robotica*, vol. 26, pp. 395–403, 2008.

[17] G. Moroz, F. Rouiller, D. Chablat, and P. Wenger, “On the determination of cusp points of the 3-R$\underline{\text{P}}$R parallel manipulators,” *Mechanism and Machine Theory*, vol. 45, no. 11, pp. 1555–1567, 2010.

[18] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces,” *IEEE Transactions on Robotics and Autonomation*, vol. 12, pp. 566–580, August 1996.

[19] J. H. Yakey, S. M. LaValle, and L. E. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *IEEE Transactions on Robotics and Autonomation*, vol. 17, no. 6, pp. 951–958, 2001.

[20] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 105–117, 2013.

[21] B. Kim, T. T. Um, C. Suh, and F. C. Park, "Tangent bundle RRT: A randomized algorithm for constrained motion planning," *Robotica*, vol. 34, no. 1, pp. 202–225, 2016.

[22] W. Au, H. Chung, and C. Chen, "Path planning and assembly mode-changes of 6-DoF stewart-gough-type parallel manipulators," *Mechanism and Machine Theory*, vol. 106, pp. 30–49, 2016.

[23] A. Sintov, A. Borum, and T. Bretl, "Motion planning of fully actuated closed kinematic chains with revolute joints: A comparative analysis," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2886–2893, 2018.

[24] F. Pernkopf and M. L. Husty, "Workspace analysis of Stewart-Gough-type parallel manpulators," *Journal of Mechanical Engineering Science*, vol. 220, no. 7, pp. 1019–1032, 2006.

[25] M. L. Husty, "An algorithm for solving the direct kinematics of the general Stewart-Gough platforms," *Mechanism and Machine Theory*, vol. 31, no. 4, pp. 365–380, 1996.

[26] G. Alici, "Determination of singularity contours for five-bar planar parallel manipulators," *Robotica*, vol. 18, pp. 569–575, 2000.

[27] W. Au, H. Chung, and C. Chen, "Path Planning of Planar Parallel Mechanisms Using Global Workspace Road Maps," in *ASME 2012 International Design Engineering Technical Conferences and CIE*, (Chicago IL, USA), 2012.

[28] M. Hayes, P. Zsombor-Murray, and C. Chen, "Unified Kinematic Analysis of General Parallel Manipulators," *ASME Journal of Mechanical Design*, vol. 126, no. 5, pp. 866–874, 2004.

[29] M. Griffis and J. Duffy, "A forward displacement analysis of a class of stewart platforms," *Journal of Robotic Systems*, vol. 6, no. 6, pp. 703–720, 1989.

[30] C. Innocenti, "Forward Kinematics in Polynomial Form of the General Stewart Platform," *Transactions of the ASME*, vol. 123, pp. 254–260, June 2001.

21

Minimally-Invasive Surgical Robot with Remote Centre of Motion

In Chapter 19, we discussed a manipulator for cannula-type minimally invasive surgery (MIS). In this chapter, we direct our focus to the non-cannula type to discuss a remote centre of motion mechanism designed for minimally invasive surgical robots.

21.1 Minimally Invasive Surgery and MIS Robots

Laparoscopic minimally invasive surgery can be regarded as replications of their open-form counterparts. The surgical manipulations are executed via long and rigid surgical tools with end-effector mounted at their distal ends, while the visual guidance is provided through laparoscopic approaches. The most well-known example is the da Vinci prostatectomy, where the da Vinci surgical robot manoeuvres one laparoscope and up to three surgical tools to conduct surgical tasks on the patient's prostate. In this chapter, we limit the scope to the laparoscopic MIS robots.

Robotic manipulators address two fundamental issues encountered in the manual laparoscopic MIS process. Firstly, surgical tools must pivot around the incision port, or they could tear the surrounding skin during translational motion. Secondly, the DoF of the surgical tools is limited to four, as shown in Figure 21.1, while the translation of the end-effector due to the non-axial rotation (around axes 1 and 2) of the surgical tool is pivoted. The solution to the first issue is remote centre of motion (RCM), virtually or mechanically, and the second is readily addressable by means of an input-output mapping in the robots' control algorithms.

Remote centre of motion refers to the capability of a manipulator to rotate its output link around a fixed spatial point without having a physical joint at said point [1]. To maximise patient safety, surgical robots adopt RCM mechanisms to achieve mechanically constrained remote centres. Figures 21.2, 21.3, and 21.4 show examples of two-DoF RCM mechanisms. Furthermore, adding a mechanism for the axial translation and another for axial rotation onto the output link of the two-DoF RCM mechanisms achieves all four required DoF in the MIS surgical application.

It should be noted that among the three examples, the former two represent a typical way to synthesise a multi-DoF RCM mechanism: to connect multiple lower-DoF RCM mechanisms in a serial configuration, where their axes of rotation intersect to define the remote centre. In fact, the combination depicted in Figure 21.3 is adopted by da Vinci surgical robot. Following such synthesis, the parallelogram and its derivatives further represent the most widely used planar (1R) RCM mechanism, as seen on numerous medical robots within and beyond the scope of laparoscopic MIS.

DOI: 10.1201/9781003614319-21

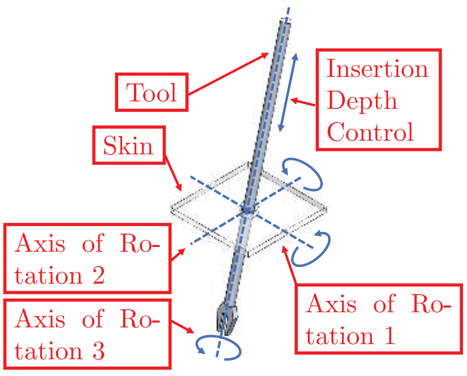


FIGURE 21.1
DoFs of a surgical tool through an incision port.

One limitation of the parallelogram-based RCM mechanisms is the device footprint, i.e., the space swept through by a robotic manipulator as it conducts surgical manoeuvres. In fact, it is not uncommon to observe inter-robotic-arm collisions of the da Vinci system in prostatectomy procedures. Having a large device footprint near the remote centre further prevents immediate access of human surgeons to the patient, which is, in turn, another critical aspect of ensuring patient safety in an emergency. To tackle this issue, we investigated a planar remote centre of motion mechanism, the dual-triangular linkage, and demonstrated its advantages through skeletal prototypes of MIS surgical robots [2, 3, 4].

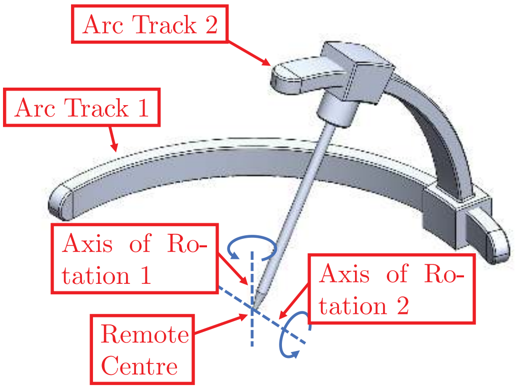


FIGURE 21.2
Two-DoF RCM mechanism based on two arc tracks.

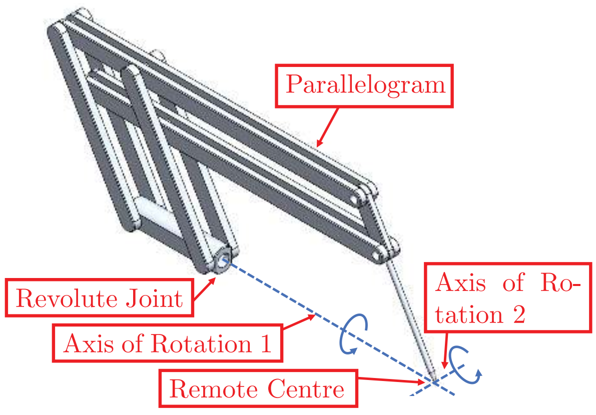


FIGURE 21.3
Two-DoF RCM mechanism based on revolute joint and parallelogram.

21.2 The Dual-Triangular Linkage

21.2.1 The Design

The schematic diagram of the dual-triangular linkage ("DT linkage" hereafter) is illustrated in Figure 21.5(a). Link AC is the input, and EH is the output. Additionally, a cantilever can be connected rigidly to EH as an extension, whose shape can be designed freely to suit surgical requirements. The DT linkage comprises two pairs of similar triangles (hence "dual-triangular"), OCA and OGF, and OCE and OGH, respectively, satisfying the following kinematic constraint [2]:

$$\begin{aligned} \frac{a_1 + a_2}{a_1} &= \frac{b_1 + b_2}{b_2} \\ a_1 &= b_1 \\ a_2 &= b_2 \end{aligned} \tag{21.1}$$

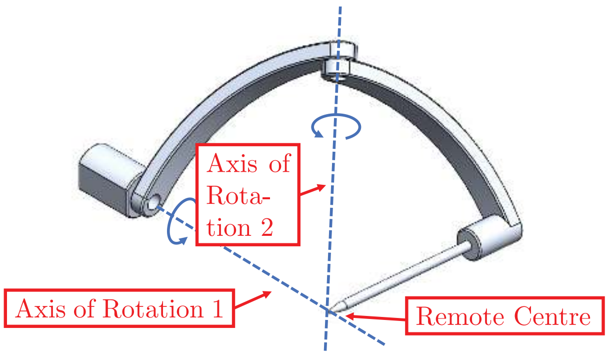
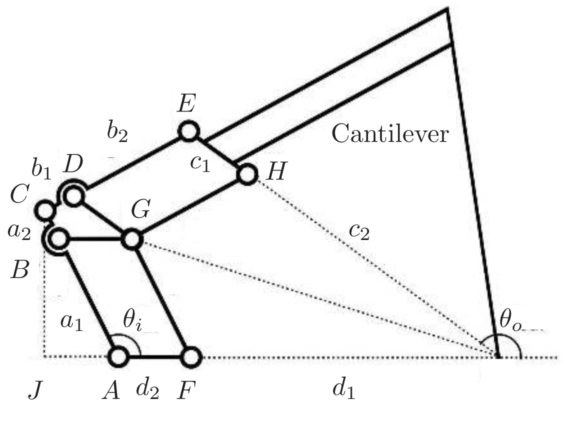
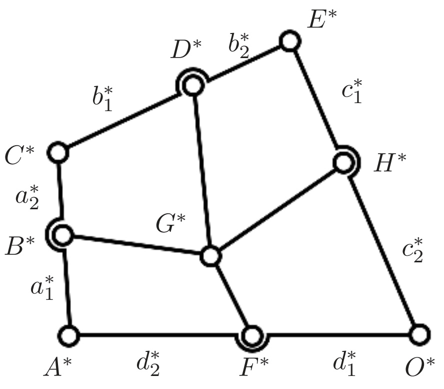


FIGURE 21.4
Two-DoF RCM mechanism based on spherical linkage.



(a) Schematics of the Dual-Triangular Linkage



(b) Schematics of the Kempe focal linkage

FIGURE 21.5
The comparison between dual-triangular linkage [4] and Kempe focal linkage [5].

a_1, a_2, b_1, and b_2 denoting the lengths of AB (GF), BC, DE (GH), and CD, respectively. Moreover, the input-output relation is defined as:

$$\theta_o = \pi - \tan^{-1}\frac{l_{CJ}}{l_{JO}} = \pi - 2\tan^{-1}\frac{l_{AC}\sin\theta_i}{l_{AO} - l_{AC}\sin\theta_i} \tag{21.2}$$

where θ_i and θ_o are the input and output angles, respectively.

The upper limit in the range of motion of the DT linkage corresponds to the configuration where AC and CE are collinear, and OCA and OCE become right-angle triangles. Starting from the home configuration, where all the links fold and overlap with the ground, the range of motion of the DT linkage is defined by

$$\theta_r = 2\sin^{-1}\frac{l_{AC}}{l_{AO}} \tag{21.3}$$

The DT linkage can also be interpreted as the combination of four-bar linkage $BCDG$ and parallelograms $ABGF$ and $DEHG$. It should be noted that the virtual four-bar linkage $ACEO$, corresponding to the outer shape of the DT linkage, forms a pair of similar parallelograms with $BCEG$. Four-bar linkage $ACEO$ adopts the 2K-SLLS architecture to maximise the clearance between the kinematic chain and the remote centre. 2K stands for symmetrical kite shape with two link lengths, and SLLS (short-long-long-short) refers to the link configuration, where the input (AC) and the connector (CE) are shorter than the ground (AO) and the output (EO). Such an interpretation allows the proof of remote centre of motion based on a comparison with the Kempe focal linkage, detailed in the subsection below.

21.2.2 Proof of Remote Centre of Motion

Figure 21.5(b) shows the schematic structure of a generalised Kempe focal linkage [5]. For convenience of comparison, we mark the matching nodes and link lengths on the Kempe linkage with starts. It can be readily seen that the major difference between the DT and the Kempe linkages is that FO and HO are virtual extensions, while F^*O^* and H^*O^* are physical links.

The mobility condition of the Kempe linkage is expressed as [6]

$$\begin{aligned}\frac{a_1^*}{a_2^*} &= \frac{c_2^*}{c_1^*}\\ \frac{b_1^*}{b_2^*} &= \frac{d_2^*}{d_1^*}\end{aligned} \tag{21.4}$$

Its equivalent of the DT linkage is given by

$$\begin{aligned}\frac{l_{AB}}{l_{BC}} &= \frac{l_{HO}}{l_{EH}}\\ \frac{l_{CD}}{l_{DE}} &= \frac{l_{AF}}{l_{FO}}\end{aligned} \tag{21.5}$$

On the other hand, (21.1) yields

$$\frac{l_{AF}}{l_{FO}} = \frac{l_{BG}}{l_{FO}} = \frac{l_{EH}}{l_{HO}} = \frac{l_{DG}}{l_{HO}} = \frac{l_{BC}}{l_{AB}} = \frac{l_{BC}}{l_{FG}} = \frac{l_{CD}}{l_{DE}} = \frac{l_{CD}}{l_{GH}} \tag{21.6}$$

comparing (21.5) and (21.6) shows that the DT linkage is a special case of the Kempe linkage, and that the EH will rotate around O despite O now being a virtual node, verifying the remote centre of motion.

21.2.3 The Gear-Constrained Dual-Triangular Linkage

The range of motion defined in (21.3) is between the overlap/home position and the maximum extension. The former is a singular configuration, where the singularity is brought about by both four-bar linkage $BCDG$ and parallelograms $ABGF$ and $DEHG$. Upon crossing the home position, the four-bar linkage could fall into an undesired overlap configuration, where the input and the connector remain overlapped, and the output stays collinear with the ground, i.e., the change in the input angle corresponds to no change in the output. On the other hand, the parallelogram also has two possible configurations upon crossing the home position: the desired parallelogram or the undesired crossed four-bar linkage.

In the singularity-free case, the DT linkage could move between the maximum extension positions above and below the ground, i.e., $\theta_r' = 2\theta_r$. Auxiliary mechanisms are introduced for the four-bar linkage and the parallelograms, respectively, for singularity removal and range of motion extension.

The auxiliary mechanism for the four-bar linkage, illustrated in Figure 21.6, comprises a sub-kinematic chain CIG and a gear train. Said sub-kinematic chain always aligns with the centre line CG of the four-bar linkage, and a prismatic joint I permits the distance between C and G to change with respect to the input angle. This sub-kinematic chain serves as the reference line for $\angle GCA$ and $\angle GCE$ to be synchronised via the gear train. The input gear 1 and the output gear 4 are rigidly attached to and rotate with AC and CE, respectively. Intermediate gears 2 and 3 rotate on CI and are symmetrical around the centre line (2 above and 3 below) to ensure gear 4 rotates in the reverse direction of gear 1. An additional gear, $2'$, rotates with 2 to mesh with 3. Notably, the auxiliary mechanism serves as a redundant constraint when the DT linkage is not in the singular position. As such, the DT linkage becomes fully constrained at the singular position and is over-constrained otherwise.

The auxiliary mechanism for the parallelograms is displayed in Figure 21.7. Two auxiliary parallelograms, $AA'F'F$ and $DD'G'G$, are rigidly attached to the main parallelograms, $ABGF$ and $DEHG$, respectively, with angular offsets. Said offset allows the main and the auxiliary parallelograms to not be in the singular position simultaneously, mutually constraining each other to ensure the integrated sub-kinematic chain is singularity-free.

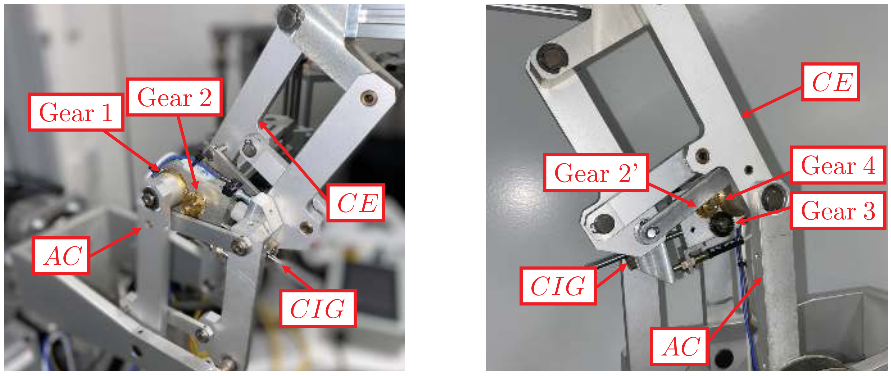


FIGURE 21.6
The auxiliary mechanism for four-bar linkage.

A prototype was constructed for demonstration based on the DT linkage. The prototype adopts a two-DoF remote centre of motion mechanism. A revolute joint provides the first DoF and defines the rotation plane for the DT linkage. The DT linkage, in turn, provides the second DoF through in-plane rotation by aligning its ground link AF with the revolute joint axis. The remote centre is thus created at the intersection of the revolute joint axis and the output axis of the DT linkage. In addition, a SCARA arm and a z-axis linear motion stage define the position of the remote centre in the free space. A photo of our prototype is shown in Figure 21.8.

21.2.4 The 2R1T RCM Mechanism Based on the DT Linkage

Another research interest of the DT linkage is the inclusion of the axial rotation and the longitudinal translation of the end-effector to achieve the 3R1T required for minimally invasive surgery. Figure 21.9 [7] depicts an example linkage of such kind.

The additional 1R1T is achieved by mounting two Peaucellier-Lipkin straight-line linkages (coloured blue and red in Figure 21.9, respectively) onto the output link of the DT linkage. The two straight-line linkages connect to the end-effector through their respective

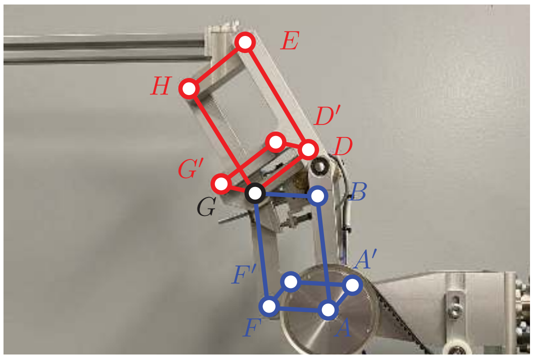


FIGURE 21.7
The auxiliary mechanisms for the two parallelograms.

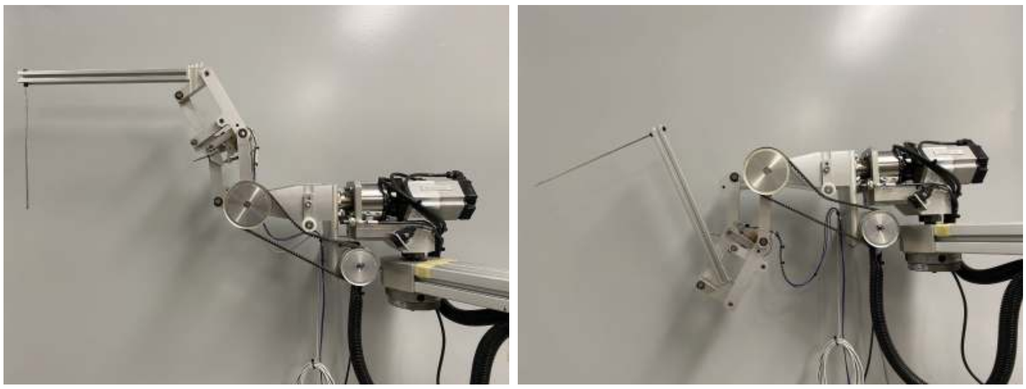


FIGURE 21.8
The two-DoF prototype at the upper (LHS) and lower (RHS) limits.

helical joints. These two joints, in turn, feature right-hand and left-hand screws, respectively. The trajectories of their output nodes are coincident, both passing through the remote centre of the DT linkage. The axial rotation of the end-effector is thus achieved by the relative translation between the output nodes of the straight-line linkages, whereas the simultaneous translation of the nodes realises the longitudinal translation.

21.3 The Cable-Constrained Linkage with Remote Centre of Motion

21.3.1 The Design

The design described previously has two areas for improvement. Firstly, the gear train introduces backlash. Secondly, the auxiliary parallelograms increase the thickness of the

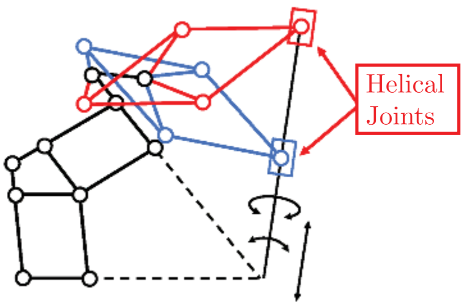


FIGURE 21.9
The 2R1T planar RCM mechanism based on DT linkage.

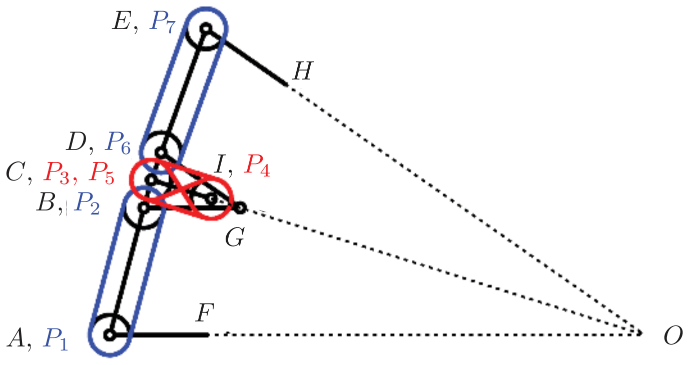


FIGURE 21.10
The schematic structure of the cable-constrained linkage.

mechanism in the transverse plane, enlarging the device footprint during surgical manipulation. To address these issues, a new design based on cable loops is proposed, illustrated in Figure 21.10.

On the cable-constrained dual-triangular linkage ("CC linkage" hereafter) [8], two cable loops replace parallelograms $ABGF$ and $DEHG$, respectively. Links GF and GH have hence been removed. Each cable loop comprises two pulleys: P_1 and P_2, and P_6 and P_7, respectively, where the pulleys, correspondingly, are rigidly connected to AF, BG, DG, and EH. As for the four-bar linkage $BCDG$, the gear train in the articulated design is replaced with a cable loop connecting three pulleys. Both the input (P_3) and the output (P_5) pulleys rotate around C, and are, correspondingly, connected to AC and CE. The intermediate/idler pulley P_4 is mounted on sub-kinematic chain CIG. As well, in the first cable section that retains tension while θ_i decreases (i.e., linkage moving towards the upper limit), the cable runs parallel to CIG from P_3 to P_4, then reaches P_5 in a crossed configuration. In the opposing cable section, the cable is parallel and crossed, between P_5 and P_4 and P_4 and P_3, respectively.

21.3.2 Proof of Functioning

Provided the proof of functioning of the DT linkage in the previous section, the CC linkage can too be proven an RCM mechanism, given that:

- The cable loop constraining four-bar linkage $BCDG$ does not obstruct the motion of the mechanism.
- The pertinent cable sections are in tension in the designed direction of motion.

The rest of the subsection focuses on demonstrating the conditions mentioned above based on the framework of constraint analysis discussed in Chapter 18.

The following generalised coordinates are used in the analysis:

Generalised coordinates	Definition	Reference
θ_1	Input angle, angle of $\overrightarrow{AC}$	$\overrightarrow{AF}$
θ_2	Angle of $\overrightarrow{BG}$ and P_2	$\overrightarrow{AC}$
θ_3	Angle of $\overrightarrow{CG}$	$\overrightarrow{AC}$
θ_4	Angle of P_4	$\overrightarrow{CG}$
θ_5	Angle of $\overrightarrow{CE}$	$\overrightarrow{GC}$
θ_6	Angle of $\overrightarrow{DG}$	$\overrightarrow{CE}$
θ_7	Angle of $\overrightarrow{EH}$	$\overrightarrow{CE}$
l_{CG}	Length of sub-kinematic chain CIG	n/a
x_H	x-position of H	A
y_H	y-position of H	A

which are collected in array $\mathbf{q}$ of the generalised coordinates

$$\mathbf{q} = \begin{bmatrix}\theta_1 & \theta_2 & \theta_3 & \theta_4 & \theta_5 & \theta_6 & \theta_7 & l_{CG} & x_H & y_H\end{bmatrix}^T \in \mathbb{R}^{10} \tag{21.7}$$

Furthermore, array $\mathbf{f}_e$ of the generalised external force is given by

$$\mathbf{f}_e = \begin{bmatrix}\tau_{e1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & f_{eHy}\end{bmatrix}^T \in \mathbb{R}^{10} \tag{21.8}$$

where τ_{e1} represents the required input torque when the mechanism is under a y-axis external force f_{eHy}.

The kinematic constraint equations of the CC linkage comprise those derived from joint positions and cable loops. For the former, three sets of constraint equations can be derived from the revolute joint G and the fixed joint connecting a virtual node E with the kinematic chain, as given below

$$\begin{aligned}\boldsymbol{\phi}_{G1} = &\left(l_1 \begin{bmatrix}\cos\theta_1\\ \sin\theta_1\end{bmatrix} + l_{CG}\begin{bmatrix}\cos(\theta_1+\theta_3)\\ \sin(\theta_1+\theta_3)\end{bmatrix}\right)\\ &- \left(l_2 \begin{bmatrix}\cos\theta_1\\ \sin\theta_1\end{bmatrix} + l_3\begin{bmatrix}\cos(\theta_1+\theta_2)\\ \sin(\theta_1+\theta_2)\end{bmatrix}\right) = \mathbf{0}\end{aligned} \tag{21.9}$$

for joint G following paths $A-C-G$ and $A-B-G$, respectively; and

$$\begin{aligned}\boldsymbol{\phi}_{G2} = &\left(l_1 \begin{bmatrix}\cos\theta_1\\ \sin\theta_1\end{bmatrix} + (l_1 - l_2)\begin{bmatrix}\cos(\theta_1+\theta_3+\pi+\theta_5)\\ \sin(\theta_1+\theta_3+\pi+\theta_5)\end{bmatrix} + l_3\begin{bmatrix}\cos(\theta_1+\theta_3+\pi+\theta_5+\theta_6)\\ \sin(\theta_1+\theta_3+\pi+\theta_5+\theta_6)\end{bmatrix}\right)\\ &- \left(l_2 \begin{bmatrix}\cos\theta_1\\ \sin\theta_1\end{bmatrix} + l_3\begin{bmatrix}\cos(\theta_1+\theta_2)\\ \sin(\theta_1+\theta_2)\end{bmatrix}\right) = \mathbf{0}\end{aligned} \tag{21.10}$$

for joint G following paths $A-C-D-G$ and $A-B-G$, respectively; and for the fixed joint E

$$\begin{aligned}\boldsymbol{\phi}_{E} = &\left(l_1 \begin{bmatrix}\cos\theta_1\\ \sin\theta_1\end{bmatrix} + l_1\begin{bmatrix}\cos(\theta_1+\theta_3+\pi+\theta_5)\\ \sin(\theta_1+\theta_3+\pi+\theta_5)\end{bmatrix} + l_3\begin{bmatrix}\cos(\theta_1+\theta_3+\pi+\theta_5+\theta_7)\\ \sin(\theta_1+\theta_3+\pi+\theta_5+\theta_7)\end{bmatrix}\right)\\ &- \begin{bmatrix}x_E\\ y_E\end{bmatrix} = \mathbf{0}\end{aligned} \tag{21.11}$$

the path described in the first bracket being $A - C - E - H$. Furthermore, for simplicity, the distances l_1, l_2, and l_3 between nodes are defined as:

$$\begin{aligned} l_1 &= l_{AC} = l_{CE} = \nu l_{AO} \\ l_2 &= l_{AB} = l_{DE} = \frac{l_{AC}}{\rho} = \frac{\nu l_{AO}}{\rho} \\ l_3 &= l_{AF} = l_{BG} = l_{DG} = l_{EH} = \left(1 - \frac{1}{\rho}\right) l_{AO} \end{aligned}$$

coefficients ν and ρ are design parameters of the CC linkage.

The kinematic constraints of cable loop relate the rotations of connected pulleys through prescribed (and constant) lengths of cable sections. We use the generalised expressions below to describe the length of cable wrapped on a pulley:

$$\begin{aligned} l_{cw1} &= r\left(\theta_{ji} - \theta_j\right) \\ l_{cw2} &= r\left(\theta_j - \theta_{jo}\right) \end{aligned} \tag{21.12}$$

$$\begin{aligned} l_{ccw1} &= r\left(\theta_j - \theta_{ji}\right) \\ l_{ccw2} &= r\left(\theta_{jo} - \theta_j\right) \end{aligned} \tag{21.13}$$

subscripts *cw* and *ccw* identify the directions of the cables, i.e., clockwise and counter-clockwise, respectively. Furthermore, subscripts 1 and 2 denote two halves of the cable loop, each tensioned in one direction of motion of the CC linkage. Angle θ_j ($j \in [1, 7]$) indicates the angle of pulley j. It can also be interpreted as the angle of the vector directed from the centre to the cable attachment point on the pulley. On the other hand, θ_{ji} and θ_{jo}, correspondingly, are the cable inlet and outlet angles.

For each direction of motion, four constraint equations can be derived following Equations (21.12) and (21.12), corresponding to the cable sections between P_1 and P_2, P_3 and P_4, P_4 and P_5, and P_6 and P_7, respectively. For the upward motion where θ_1 decreases, the corresponding equations are:

$$\begin{aligned} \phi_{c11} &= r\left(0 - \left(\theta_1 + \pi/2\right)\right) + r\left(\pi/2 - \theta_2\right) - l_{c11} = 0 \\ \phi_{c21} &= r\left(\left(\theta_3 - \pi/2\right) - 0\right) + r\left(\theta_4 - \left(0 - \pi/2\right)\right) - l_{c21} = 0 \\ \phi_{c31} &= r\left(\left(0 + \pi/2 + \beta\right) - \theta_4\right) + r\left(\left(0 + \pi/2 + \beta\right) - \theta_5\right) - l_{c31} = 0 \\ \phi_{c41} &= r\left(\theta_6 - \left(0 + \pi/2\right)\right) + r\left(\left(0 + \pi/2\right) - \theta_7\right) - l_{c41} = 0 \end{aligned} \tag{21.14}$$

where $r\beta$ describes the extra on-pulley cable length associated with the crossed cable configuration and is a constant. As well, l_{c11} to l_{c41} identify the prescribed lengths of the cable sections for the half cable loops. Moreover, the equivalent of (21.14) for the downward motion is given by

$$\begin{aligned} \phi_{c12} &= r\left(\left(\theta_1 - \pi/2\right) - 0\right) + r\left(\theta_2 - \left(0 - \pi/2\right)\right) - l_{c12} = 0 \\ \phi_{c22} &= r\left(0 - \left(\theta_3 + \pi/2\right)\right) + r\left(\left(0 + \pi/2\right) - \theta_4\right) - l_{c22} = 0 \\ \phi_{c32} &= r\left(\theta_4 - \left(0 - \pi/2 - \beta\right)\right) + r\left(\theta_5 - \left(0 + \pi/2 + \beta\right)\right) - l_{c32} = 0 \\ \phi_{c42} &= r\left(\left(0 - \pi/2\right) - \theta_6\right) + r\left(\theta_7 - \left(0 - \pi/2\right)\right) - l_{c42} = 0 \end{aligned} \tag{21.15}$$

similarly, l_{c12} to l_{c42} are the prescribed lengths of cable sections for the half cable loops.

In Equations (21.14) and (21.15), ϕ_{c21}, ϕ_{c31}, ϕ_{c22}, and ϕ_{c32} are derived from the cable loop constraining four-bar linkage $BCDG$. Summing ϕ_{c21} and ϕ_{c31} leads to

$$r\left(\pi + \theta_3 - \theta_4 + 2\beta\right) = l_{c21} + l_{c31}$$

As such, $\theta_3 = \theta_4$ is unconditionally valid. A similar conclusion can be drawn upon the summation of ϕ_{c22} and ϕ_{c32}. Therefore, $\angle OCA$ always equals $\angle OCE$, and four-bar linkage $BCDG$ is singularity-free at the overlap configuration. Moreover, the cable loop connecting pulleys P_3, P_4, and P_5 does not obstruct the motion of the four-bar linkage, and the first condition for RCM is satisfied.

Combining the kinematic constraint equations of joints Equations (21.9), (21.10), and (21.11) with cable constraints of the pertinent direction (21.14) or (21.15) yields the array of constraint equations for one direction of motion:

$$\boldsymbol{\phi}_1 = \begin{bmatrix}\boldsymbol{\phi}_{G1}^T & \boldsymbol{\phi}_{G2}^T & \boldsymbol{\phi}_H^T & \phi_{c11} & \phi_{c21} & \phi_{c31} & \phi_{c41}\end{bmatrix}^T \in \mathbb{R}^{10} \tag{21.16}$$

and

$$\boldsymbol{\phi}_2 = \begin{bmatrix}\boldsymbol{\phi}_{G1}^T & \boldsymbol{\phi}_{G2}^T & \boldsymbol{\phi}_H^T & \phi_{c12} & \phi_{c22} & \phi_{c32} & \phi_{c42}\end{bmatrix}^T \in \mathbb{R}^{10} \tag{21.17}$$

the former is for upward motion, the latter downward. Additionally, their corresponding array of Lagrange multipliers are:

$$\boldsymbol{\lambda}_1 = \begin{bmatrix}\boldsymbol{\lambda}_{G1}^T & \boldsymbol{\lambda}_{G2}^T & \boldsymbol{\lambda}_H^{T\,T} & \lambda_{c11} & \lambda_{c21} & \lambda_{c31} & \lambda_{c41}\end{bmatrix}^T \tag{21.18}$$

and

$$\boldsymbol{\lambda}_2 = \begin{bmatrix}\boldsymbol{\lambda}_{G1}^T & \boldsymbol{\lambda}_{G2}^T & \boldsymbol{\lambda}_H^{T\,T} & \lambda_{c12} & \lambda_{c22} & \lambda_{c32} & \lambda_{c42}\end{bmatrix}^T \tag{21.19}$$

where scalar Lagrange multipliers λ_c are the forces transmitted through the cable sections and must be zero or positive to satisfy the second condition for the CC linkage to be an RCM mechanism.

The static equilibrium equation of the CC linkage is expressed based on Equation (18.11). However, since the input torque τ_{e1} of $\mathbf{f}_e$, displayed in (21.8), is unknown, (18.11) is split into two equations by means of selection matrices, such that the Lagrange multipliers can be solved with the prescribed f_{eHy}. The two-component static equilibrium equation is defined as:

$$\left(\frac{\partial\,\boldsymbol{\phi}_k}{\partial \mathbf{q}}\,\mathbf{S}_1\right)^T \boldsymbol{\lambda}_k - \mathbf{S}_1^T\,\mathbf{f}_e = \mathbf{0} \tag{21.20}$$

and

$$\left(\frac{\partial\,\boldsymbol{\phi}_k}{\partial \mathbf{q}}\,\mathbf{S}_2\right)^T \boldsymbol{\lambda}_k - \mathbf{S}_2^T\,\mathbf{f}_e = \mathbf{0} \tag{21.21}$$

where $k = [1, 2]$, correspondingly, identifies the upward and downward motion. Arrays $\mathbf{q}$, $\mathbf{f}_e$, $\boldsymbol{\phi}$, and $\boldsymbol{\lambda}$ are defined in Equations (21.7), (21.8), (21.16), or (21.17), and (21.18) or (21.18), respectively. Moreover, $\mathbf{S}_1$ and $\mathbf{S}_2$ are the selection matrices of the input (θ_1) and other generalised coordinates, respectively, defined as:

$$\mathbf{S}_1 = \mathbf{s}_1 \tag{21.22}$$

and

$$\mathbf{S}_2 = \begin{bmatrix}\mathbf{s}_2 & \mathbf{s}_3 & \cdots & \mathbf{s}_9 & \mathbf{s}_{10}\end{bmatrix} \tag{21.23}$$

where

$$\mathbf{S} = \begin{bmatrix}\mathbf{S}_1 & \mathbf{S}_2\end{bmatrix} = \mathbf{1}_{10}$$

full-selection matrix $\mathbf{S}$ is the 10×10 identity matrix $\mathbf{1}_{10}$, and each column corresponds to one $\mathbf{s}_m$ ($m \in [1, 10]$).

In Equation (21.21), $\frac{\partial\,\boldsymbol{\phi}_k}{\partial \mathbf{q}}\,\mathbf{S}_2$ has the dimension of 10×9, i.e., a non-square constraint Jacobian of an over-constrained mechanism. It should be noted that at the singular pose, the x-component of $\boldsymbol{\phi}_{G2}$ vanishes, effectively rending a fully constrained system. The non-square constraint Jacobian can be solved by means of pseudo-inverse, or

$$\boldsymbol{\lambda}_k = \left(\left(\frac{\partial\, \boldsymbol{\phi}_k}{\partial \mathbf{q}}\, \mathbf{S}_2\right)^T\right)^\dagger \left(\mathbf{S}_2^T\, \mathbf{f}_e\right) \tag{21.24}$$

with † indicating a pseudo-inverse operation. A set of numerical cable forces is presented below in Figure 21.11 to verify the cable tension condition. The horizontal and vertical axes identify the cable forces and θ_1 the input angle, respectively. The corresponding design parameters are listed below.

Parameters	Value	Unit
l_{AO}	400	mm
ν	0.3	n/a
ρ	1.22	n/a
r	15	mm
f_{eHy}	∓ 8	N

For the external force f_{eHy}, the negative value is for the upward motion and its positive counterpart is for the downward.

The results are presented in Figure 21.11, where 12, 67, and 345 denote the cable sections between pulleys P_1 and P_2, P_6 and P_7, and among P_3 to P_5, respectively. Additionally, cable sections belonging to the same cable loop demonstrate identical tension during the upward and downward movements. The results show that all the cable sections are in tension in their designed range of motion. Therefore, the force condition for the CC linkage to be an RCM mechanism is verified.

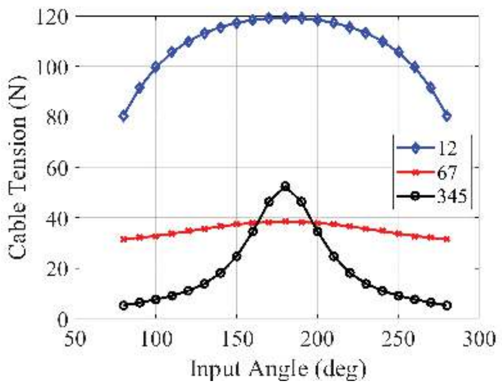


FIGURE 21.11
The cable tension during linkage motion.

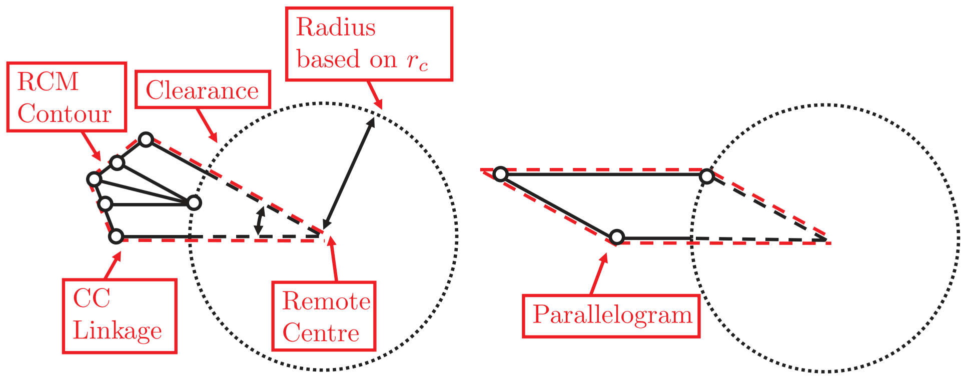


FIGURE 21.12
Footprint comparison of the planar RCM mechanisms.

21.4 Advantage of DT/CC Linkage over Parallelogram RCM Mechanism

To demonstrate the advantage, the device footprint of the manipulator based on the CC linkage is compared to its parallelogram-based counterpart. This device footprint is defined as the volume swept by the maneuver of the two-DoF RCM mechanism, and the difference is, in turn, dictated solely by the area bounded by the outer contour of the planar CC linkage or parallelogram.

Figure 21.12 illustrates the setup of the comparison. The clearance represented by the dashed circle refers to the space where the kinematic chains shall not enter, emulating the workspace that human surgeons can occupy to access the patient. The red dashed lines indicate the footprint of the planar mechanism. Two design parameters are considered: the dimensionless ratio r_c resultant from the clearance radius and the distance from the input joint to the remote centre, and the required range of motion θ_r. Also, with a prescribed set of design parameters, the device footprint is a normalised value obtained upon the summation of the bounded areas at two positions: the upper limit where output angle θ_o equals $\theta_r/2$ and the upper-mid position where θ_o is $\theta_r/4$. Notably, the positions where input angle $\theta_i \geq \pi$ are not considered, as both planar RCM mechanisms are symmetrical around the ground axis.

The percentage difference in the device footprint is presented in Figure 21.13, with the footprint of the parallelogram serving as the reference values. It has been demonstrated that the CC linkage has the advantage in cases where the required range of motion is small, and the clearance needs to be large. Moreover, Figure 21.14 compares the spatial footprints of RCM surgical manipulators based on the CC linkage and parallelogram under an emulated minimally invasive surgery setup. The volumes swept by the RCM mechanisms are coloured red; the clearances of the CC linkage and the parallelogram are represented by the blue and the orange spheres, respectively.

Four volumes of interest are highlighted in Figure 21.14:

- The volume swept by the output joint (E) of the CC linkage and a straight output link in a two-DoF manipulation, coloured in red in the LHS figure.

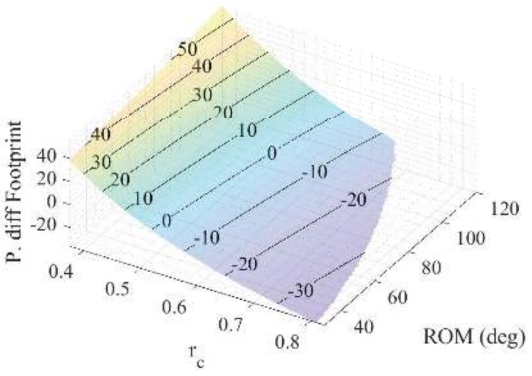


FIGURE 21.13
Percentage difference of CC linkage footprint over parallelogram.

- The volume swept by the output joint of a parallelogram and a straight output link in a two-DoF manipulation, coloured in red in the RHS figure.
- The clearance between the incision port and the kinematic chain that generates the RCM for the CC linkage, represented by the blue sphere.
- The clearance between the incision port and the kinematic chain that generates the RCM for the parallelogram, represented by the orange sphere.

It can be observed that clearance is larger for the CC linkage case, leading to additional room for easier access of human surgeons to the patient, benefiting the planning and execution of the minimally invasive surgical process.

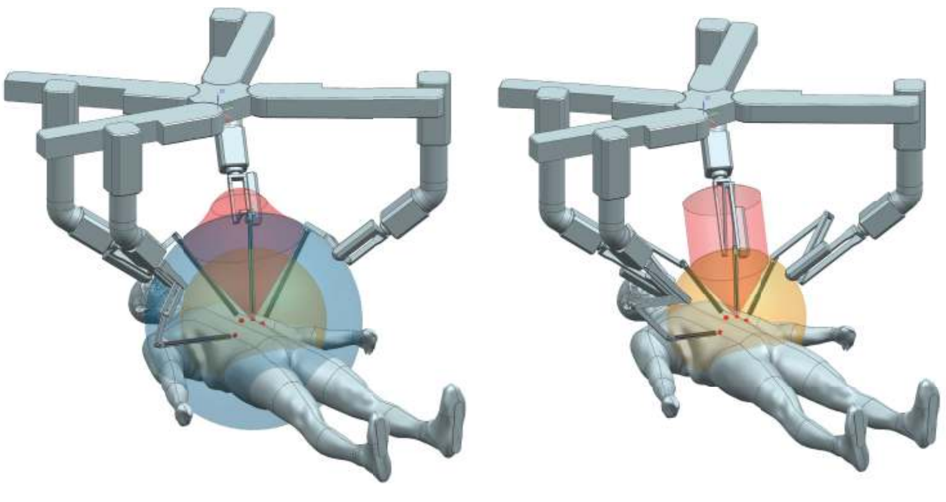


FIGURE 21.14
Comparison of surgical manipulator footprints for CC linkage (LHS) and parallelogram (RHS).

21.5 Conclusion

In this chapter, we presented our novel remote centre of motion mechanism, with the aim of tackling the bulkiness of its conventional parallelogram-based counterparts. Two design variations were introduced, and our focus is on the proof of the property of motion. For the base design that features a compound gear-linkage kinematic chain, we proved that it is indeed a remote centre of motion mechanism. From there, the improved design adopts a hybrid cable-linkage system, and our deviation showcased that the newly-introduced cable system could maintain necessary tension and hence enforce the kinematic constraints to achieve the remote centre of motion. For the completeness of the work, we conducted a comparison between the sweeping volumes of our invention and the parallelogram structure to quantify the advantage.

Bibliography

[1] C.-H. Kuo and J. S. Dai, "Robotics for minimally invasive surgery: a historical review from the perspective of kinematics," in *International Symposium on History of Machines and Mechanisms: Proceedings of HMM 2008*, pp. 337–354, Springer, 2009.

[2] C. Chen and M. Pamieta, "Novel linkage with remote centre of motion," in *IFToMM International Symposium on Robotics and Mechatronics 2013*, pp. 139–147, Research Publishing Services, 2013.

[3] S. Liu, B. Chen, S. Caro, S. Briot, and C. Chen, "Dual-triangular remote centre of motion mechanism with cable transmission," in *4th Joint International Conference on Multibody System Dynamics (IMSD2016)*, 2016.

[4] S. T. Liu, L. Harewood, B. Chen, and C. Chen, "A skeletal prototype of surgical arm based on dual-triangular mechanism," *Journal of Mechanisms and Robotics*, vol. 8, no. 4, p. 041015, 2016.

[5] A. B. Kempe, "On conjugate four-piece linkages," *Proceedings of the London Mathematical Society*, vol. 1, no. 1, pp. 133–149, 1877.

[6] J. E. Baker, and H. C. Yu, "Re-examination of a Kempe linkage," *Mechanism and Machine Theory*, vol. 18, no. 1, pp. 7–22, 1983.

[7] G. Chen, Y. Xun, Y. Chai, S. Yao, C. Chen, and H. Wang, "Design and validation of a novel planar 2r1t remote centre-of-motion mechanism composing of dual-triangular and straight-line linkages," *Journal of Mechanisms and Robotics*, pp. 1–11, 2021.

[8] S. Liu, B. Chen, S. Caro, S. Briot, L. Harewood, and C. Chen, "A cable linkage with remote centre of motion," *Mechanism and Machine Theory*, vol. 105, pp. 583–605, 2016.

22

MARS: The Monash Apple Retrieving System

The Monash Apple Retrieving System (MARS) is a selective apple harvesting platform, developed in A/Prof Chao Chen's Laboratory of Motion Generation and Analysis (Figure 22.1). It features a 6 degree of freedom (DoF) UR5 manipulator, a four-fingered soft gripper for grasping apples while minimising fruit and canopy damage, and a sensing suite consisting of a Livox Mid70 Lidar and an Intel RealSense D455 RGB-D camera. A one-stage instance segmentation neural network is used to identify apples in the canopy, obtained from the fused point cloud and RGB information from the sensor suite. The system runs on a ROS (Robot Operating System) backbone, hosted by an nVidia Jetson TX2, which integrates and manages the communication between all components using a standardised messaging protocol. An onboard WiFi router enabled wireless communications with the ROS master, allowing multiple computers that host task-specific nodes to connect to the robot to create a distributed computing system.

MARS is a culmination of many fields of robotics, some of which have been introduced in this textbook. It makes use of forward and inverse kinematics, path planning and trajectory generation, and aspects of mechanical and electrical engineering, soft robotics, and artificial intelligence. And last but not least, the software that integrates all components (ROS and programming), and the intelligent algorithms that control its decision-making processes.

Extensive trials were conducted over the 2021 and 2022 apple harvesting seasons in Australia, where with all these components integrated, MARS was able to generate gentle, collision-free harvesting trajectories in complex and unstructured canopies. It achieved a harvesting success rate of 62.8% at a cycle time of 9.18 seconds per apple, with negligible damage to the fruit. Performance increased to 70.8% success rate and 7.91 seconds of cycle time under ideal visibility conditions.

22.1 Background and Motivation

The use of robotics and intelligent machines in agriculture is touted as a long-term solution for the manual labour shortage experienced by the industry. Currently, selective harvesting is carried out mainly by humans on high-value crops such as apples, citrus, and tomatoes, and is regarded as the most labour-intensive and expensive agricultural task [1]. As such, there has been a significant drive towards the development of selective harvesting robots in the past two decades, which was made possible due to advances in machine vision and artificial intelligence in identifying fruit in the canopy [4, 5, 6] and different grasping technologies [7, 8].

DOI: 10.1201/9781003614319-22

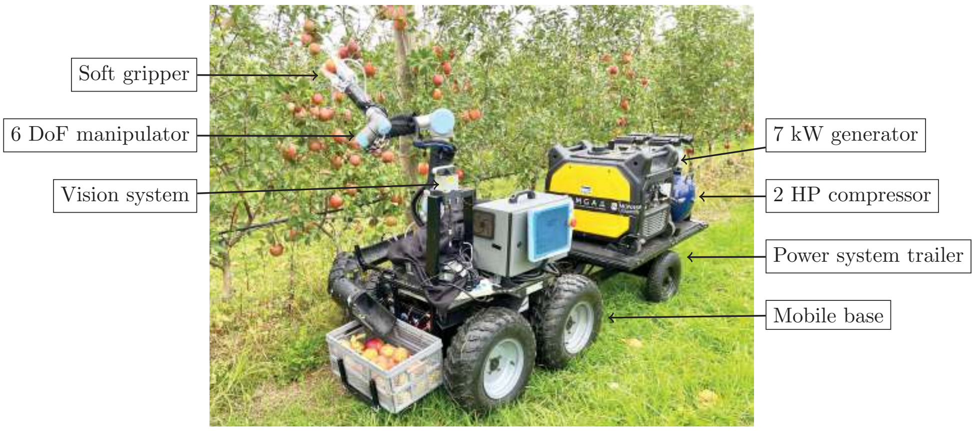


FIGURE 22.1
The Monash Apple Retrieving System.

Despite advancements in automated harvesting technologies, they have yet to achieve the level of productivity required to see widespread adoption [14] because of technical challenges encountered by robots in a highly dynamic and unpredictable canopy environment. A common theme amongst these challenges is the lack of accurate perception [3, 15] to handle such a complicated environment like an apple canopy. Other technical challenges include the lack of reliability, where precision servos and sensitive electronics are not designed for operation in harsh environments such as farms. Although many of these problems could be solved by simplifying the environment, farmers felt that delivering well-manicured canopies that are free of leaf and branch occlusions for minimally satisfactory robotic harvesting performance is an unrealistic expectation [16]. Without a solution to these problems, robotic fruit harvesting performance remains unsatisfactory which results in long cycle (harvesting) times, reduced harvest success rates, and increased potential to cause damage to fruit and canopies.

The MARS was developed with a focus on intelligent decision-making and planning algorithms, which was identified as one of the key fundamental issues holding automated selective fruit harvesters back from mass adoption. The centrepiece of this research was the planning algorithm, which enhances harvesting speed and success rate. This consists of a three-step process that utilises many topics introduced in this textbook so far.

1. *Virtual environment construction* with the vision sensor suite to identify apples and create a collision profile for collision-free planning,
2. *Optimisation* of the fruit's pose to enhance grasping success, and
3. *Path generation* for collision-free harvesting trajectories with kinematic Optimisation to reduce cycle time.

22.2 Virtual Environment Construction

All motion planning activities of MARS happen in the virtual canopy environment. Therefore, we require the following components to be known in the model:

- The kinematics model of the robot
- The position of all apples in the canopy
- Constraints in the configuration space, such as collisions
- The grasping pose of all reachable apples

22.2.1 Kinematics

Here, the frames of the robot, as well as forward and inverse kinematics, are defined.

22.2.1.1 Forward Kinematics

MARS consists of a UR5, which is a six-DoF robot in an articulated configuration with a non-spherical wrist. The modified Denavit-Hartenberg parameters of the UR5 are given in Table 22.1. Using forward kinematics, we can obtain the transformation matrices ${}^0_1T(\theta_1)$, ${}^1_2T(\theta_2)$, …, ${}^5_6T(\theta_6)$.

22.2.1.2 Inverse Kinematics

Due to the UR5's articulated structure with a non-spherical wrist, there are up to 8 inverse kinematic solutions. While this improves the UR5's ability to reach a target in a collision-free configuration [36], it also implies a complex singularity profile and configuration space, which negatively affects the robustness of numerical solutions involving inverse kinematics. Therefore, the inverse kinematics should be solved algebraically. To further improve the chances of successful grasping of an apple, the final end-effector twist can be treated as arbitrary about the arm's final z-axis (gripper's x-axis). This removes the dependence of θ_6 in the inverse-kinematic solution, i.e., we only need to find $\theta_1, \ldots, \theta_5$ for a given position ${}^0p_6$ and end-effector z-axis ${}^0\hat{Z}_6$. This also simplifies path planning queries, as there is one less DoF to handle.

First, we define a balanced equation position,

$$
{}^3\mathbf{T}_2 \; {}^2\mathbf{T}_1 \; {}^1\mathbf{T}_0 \; {}^0p_6 = {}^3\mathbf{T}_4 \; {}^4\mathbf{T}_5 \; {}^5\mathbf{T}_6 \; {}^6p_6 \tag{22.1}
$$

where ${}^0p_6 = \begin{bmatrix} p_x & p_y & p_z & 1 \end{bmatrix}^T$ and ${}^6p_6 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$. We also define a balanced equation to constrain the z-axis

$$
{}^3\mathbf{R}_2 \; {}^2\mathbf{R}_1 \; {}^1\mathbf{R}_0 \; {}^0\hat{Z}_6 = {}^3\mathbf{R}_4 \; {}^4\mathbf{R}_5 \; {}^5\mathbf{R}_6 \; {}^6\hat{Z}_6 \tag{22.2}
$$

TABLE 22.1
DH parameters of the UR5 robot

Link	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	$\frac{\pi}{2}$	0	0	θ_2
3	0	a_2	0	θ_3
4	0	a_3	d_4	θ_4
5	$\frac{\pi}{2}$	0	d_5	θ_5
6	$-\frac{\pi}{2}$	0	d_6	θ_6

where ${}^{0}\hat{Z}_6 = \begin{bmatrix} z_u & z_v & z_w & 1 \end{bmatrix}^T$ and ${}^{6}\hat{Z}_6 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$. Expanding Equations (22.1) and (22.2) yields six equations

$$p_z s_{23} - d_1 s_{23} - a_2 c_3 + p_x c_{23} c_1 + p_y c_{23} s_1 = a_3 + d_5 s_4 - d_6 c_4 s_5 \tag{22.3}$$
$$p_z c_{23} - d_1 c_{23} + a_2 s_3 - p_x s_{23} c_1 - p_y s_{23} s_1 = -d_5 c_4 - d_6 s_4 s_5 \tag{22.4}$$
$$p_x s_1 - p_y c_1 = d_4 + d_6 c_5 \tag{22.5}$$
$$z_w s_{23} + z_u c_{23} c_1 + z_v c_{23} s_1 = -c_4 s_5 \tag{22.6}$$
$$z_w c_{23} - z_u s_{23} c_1 - z_v s_{23} s_1 = -s_4 s_5 \tag{22.7}$$
$$z_u s_1 - z_v c_1 = c_5 \tag{22.8}$$

where $c_{ij} = \cos(\theta_i + \theta_j)$ and $s_{ij} = \sin(\theta_i + \theta_j)$.

We can solve for θ_1 by substituting (22.8) into (22.5) to eliminate c_5. Using tangent half-angle substitution,

$$\theta_1 = 2\arctan\left(B \pm \sqrt{\frac{B^2 - 2AC}{2A}}\right) \tag{22.9}$$

where

$$A = p_y - d_4 - d_6 z_v \tag{22.10}$$
$$B = 2p_x - 2d_6 z_u \tag{22.11}$$
$$C = d_6 z_v - p_y - d_4 \tag{22.12}$$

We can then solve θ_5 using (22.5), where

$$\theta_5 = \pi \pm \arccos\left(\frac{d_4 + p_y c_1 - p_x s_1}{d_6}\right) \tag{22.13}$$

With θ_1 and θ_5 known, we can find θ_3. We eliminate θ_4 from (22.3) and (22.4) by substituting (22.6) and (22.7) into them. The resultant equations can be expressed in the form

$$D s_{23} + E c_{23} = a_3 + a_2 c_3 \tag{22.14}$$
$$E c_{23} - D s_{23} = -a_2 s_3 \tag{22.15}$$

where

$$D = p_z - d_1 - d_6 z_w - \frac{d_5 (z_u c_1 + z_v s_1)}{s_5} \tag{22.16}$$
$$E = p_x c_1 - d_6 (z_u c_1 + z_v s_1) + p_y s_1 + \frac{d_5 z_w}{s_5} \tag{22.17}$$

By taking the squared sum of (22.14) and (22.15), we can solve for θ_3, such that

$$\theta_3 = \pm\arccos\left(\frac{D^2 + E^2 - {a_2}^2 - {a_3}^2}{2a_2 a_3}\right) \tag{22.18}$$

Joint angle θ_2 can be solved from (22.15) by via tangent half-angle substitution, such that

$$\theta_2 = 2\arctan\left(\frac{G - \sqrt{G^2 - FH}}{2F}\right) \tag{22.19}$$

where

$$F = \frac{1}{s_5}(d_5 z_w s_3 + d_1 c_3 s_5 - p_z c_3 s_5 + a_2 s_3 s_5 + p_x c_1 s_3 s_5 + p_y s_1 s_3 s_5 + d_5 z_u c_1 c_3 + d_5 c_3 s_1 + d_6 z_w c_3 s_5 - d_6 z_u c_1 s_3 s_5 - d_6 z_v s_1 s_3 s_5) \tag{22.20}$$

$$G = \frac{2}{s_5}(d_1 s_3 s_5 - d_5 z_w c_3 - p_z s_3 s_5 + d_5 z_v s_1 s_3 + d_6 z_w s_3 s_5 - p_x c_1 c_3 s_5 - p_y c_3 s_1 s_5 + d_5 z_u c_1 s_3 + d_6 z_u c_1 c_3 s_5 + d_6 z_v c_3 s_1 s_5) \tag{22.21}$$

$$H = -\frac{1}{s_5}(d_5 z_w s_3 + d_1 c_3 s_5 - p_z c_3 s_5 - a_2 s_3 s_5 + p_x c_1 s_3 s_5 + p_y s_1 s_3 s_5 + d_5 z_u c_1 c_3 + d_5 z_v c_3 s_1 + d_6 z_w c_3 s_5 - d_6 z_u c_1 s_3 s_5 - d_6 z_v s_1 s_3 s_5) \tag{22.22}$$

Finally, θ_4 can be solved from (22.4) using tangent half-angle substitution. We find

$$\theta_4 = -2 \arctan\left(\frac{J \pm \sqrt{J^2 - 4IK}}{2I}\right) \tag{22.23}$$

where

$$I = p_z c_{23} - d_1 c_{23} - d_5 + a_2 s_3 - p_x s_{23} c_1 - p_y s_{23} s_1 \tag{22.24}$$

$$J = 2 d_6 s_5 \tag{22.25}$$

$$K = d_5 - d_1 c_{23} + p_z c_{23} + a_2 s_3 - p_x s_{23} c_1 - p_y s_{23} s_1 \tag{22.26}$$

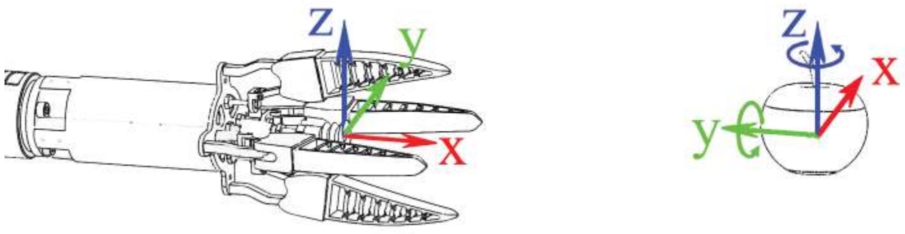
Due to each angle yielding two possible answers, mathematically there are up to 32 complete sets of solutions, based on this algebraic method. However, only eight of these sets of solutions are valid, which can be cross-checked using forward kinematics to ensure the end effector is in its intended pose. There are no performance implications in using an algebraic method for inverse kinematics on MARS, which is favoured due to its robustness compared to gradient-based numerical solutions.

22.2.2 Auxiliary Frame Assignments

The gripper frame $\{G\}$ is attached at a fixed distance to the robot's final frame and is defined in Figure 22.2(a), where its origin is coincident with the optimal position of the grasped object within the gripper. The x-axis represents the gripper's twisting axis, with the z-axis pointing upwards to the direction of gravity at zero-twist. The accompanying apple frame $\{A\}$ is defined in Figure 22.2(b). In general, $y - z$ axes represent the plane of the canopy, with the x-axis pointing inwards of the canopy. In an ideal grasping scenario, $\{G\}$ is coincident with $\{A\}$ before the fingers are closed, but these constraints can be relaxed to enhance the apple's reachability further.

22.3 Pose Optimisation

Pose Optimisation applies additional transformations to the fruit's initial frame such that the manipulator will approach the fruit at an optimised angle [29, 4]. Assuming that the vision system is capable of identifying the position of the apples in the canopy, a numerical optimisation approach is utilised to calculate the fruit approach and grasping angles. This



(a) Gripper frame $\{G\}$, with the x-axis parallel with the fingers in the direction of grasp.

(b) Apple frame $\{A\}$, with z-axis coincident with the stem and x-axis representing the front face of the apple.

FIGURE 22.2
Frame definitions. During a grasp, the origins of $\{G\}$ and $\{A\}$ are coincident, and $\mathbf{X}_G$ and $\mathbf{X}_A$ are parallel.

takes into account the robot's workspace, singularities, and occlusions or collisions with the canopy. An example of the final result is shown in Figure 22.3, where the apple's frame is adjusted such that the robot can grasp the apple at the manipulator's workspace boundary.

We begin with the standard optimisation problem

$$\underset{x}{\text{minimise}} \quad f(x) \tag{22.27}$$

$$\text{subject to} \quad g_i(x) \leq 0, \; i = 1, \ldots, m \tag{22.28}$$

$$h_j(x) = 0, \; j = 1, \ldots, n \tag{22.29}$$

Then for a single fruit, we define function $f(x)$ such that when it is minimised with constraints $g_i \leq 0$ and $h_j = 0$, the optimum approach angle for the apple is found, satisfying kinematic, path, and collision constraints.

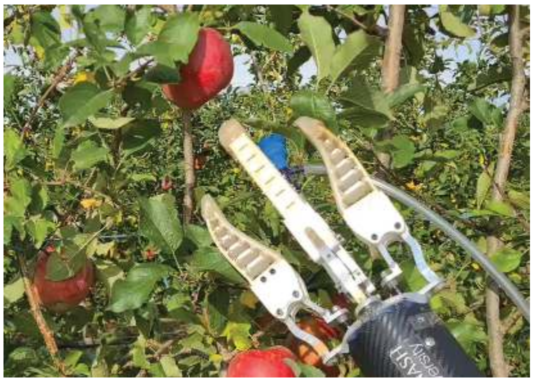


FIGURE 22.3
An optimised approach angle.

22.3.1 Objective Function

The goal of the objective function is to optimise the rotation of the fruit frame, such that only the minimum amount of frame rotation is required to ensure the fruit can be grasped whilst meeting other constraints, such as being collision-free and ensuring a valid path exists. In many cases, the final gripper twist position is arbitrary; hence, the optimisation problem can omit twist around $\mathbf{X}_G$, hence the objective function can be defined as

$$f(x) = \gamma^2 + \beta^2 \tag{22.30}$$

where $x = \begin{bmatrix}\gamma & \beta\end{bmatrix}^T$, and γ and β are the rotations about the fruit's z and y-axes respectively, applied in that order to the moving frame (Figure 22.2(b)).

22.3.2 Kinematic Constraints

Let ${}^{B}\mathbf{p}_{A_0}$ and ${}^{B}\mathbf{R}_{A_0}$ be the position and rotation of the *initial* apple frame $\{A_0\}$ relative to the robot base frame $\{B\}$. The Optimisation seeks to find the rotation

$${}^{A_0}\mathbf{R}_A = R(\gamma)R(\beta) \tag{22.31}$$

to apply on $\{A_0\}$ such that the grasp is optimised while satisfying kinematic and other constraints. Therefore the rotation matrix of the new apple grasping frame $\{A\}$ is

$${}^{B}\mathbf{R}_A = {}^{B}\mathbf{R}_{A_0} \; {}^{A_0}\mathbf{R}_A \tag{22.32}$$

The position of the end-effector, given $\{A_0\}$ and its applied rotation ${}^{A_0}\mathbf{R}_A$, is

$${}^{B}\mathbf{p}_E = {}^{B}\mathbf{R}_{A_0} \; {}^{A_0}\mathbf{R}_A \; {}^{A}\mathbf{p}_E \tag{22.33}$$

where ${}^{A}\mathbf{p}_E = {}^{G}\mathbf{p}_E$ in a grasping scenario. The position of the end-effector ${}^{B}\mathbf{p}_E$ will be used to constrain the kinematics. Assume that the inner and outer workspace boundaries of the robotic manipulator can be represented as spheres whose radii are r and R, respectively. Also, assume that the robot's workspace is relatively uniform such that if the robot's end effector frame $\{E\}$ is within the workspace boundary, then at least one inverse kinematic solution exists. Note that the end-effector frame $\{E\}$ is not coincident with the gripper frame $\{G\}$, hence the transformation ${}^{E}\mathbf{T}_G$ is non-identity. Therefore, we can write the inequality constraints

$$g_{(1,1)}(x) = |{}^{B}\mathbf{p}_E(x)| - R \tag{22.34}$$

$$g_{(1,2)}(x) = r - |{}^{B}\mathbf{p}_E(x)| \tag{22.35}$$

where a kinematic solution is guaranteed to exist if (22.27) is optimised, and (22.34) and (22.35) satisfy (22.28).

A grasping trajectory can follow a series of via frames

$$\mathbf{p}_{V_1} = \mathbf{p}_A \qquad \rightarrow \qquad \mathbf{p}_G = \mathbf{p}_A \qquad \rightarrow \qquad \mathbf{p}_{V_2} = \mathbf{p}_A \tag{22.36}$$

where $\mathbf{p}_{V_1}$ and $\mathbf{p}_{V_2}$ are via frames attached to the gripper relative to $\{G\}$. Then further constraints are defined as

$$g_{(2,2n-1)}(x) = |{}^{B}\mathbf{p}_{V_n}(x)| - R \tag{22.37}$$

$$g_{(2,2n)}(x) = r - |{}^{B}\mathbf{p}_{V_n}(x)| \tag{22.38}$$

such that n via points remain in the manipulator workspace when (22.27) is optimised, and (22.37) and (22.38) satisfy (22.28).

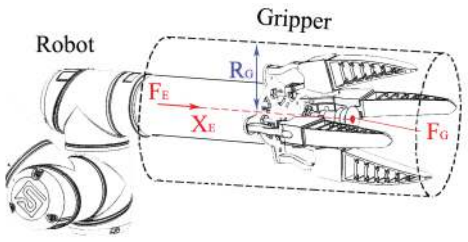


FIGURE 22.4
Gripper frames and collision cylinder (in dashed lines).

22.3.3 Collision Constraints

Collisions with the environment are considered *during* Optimisation, as poor pose estimation can cause the gripper to collide with the canopy, causing damage to both the environment and the gripper. A sensor providing cloud data for generating collision information is assumed to be available.

Let $c \in \mathbb{R}^3$ be a point defined in set C, known as the point cloud of the immediate environment. In addition, let set $C_B \subset C$ be the set of points such that it represents all inadmissible collision objects, such as trunks and branches as identified by some segmentation routine. All points in C_B will, therefore, be checked for collisions with the gripper. Let R_G be the collision radius surrounding the gripper. Finally, assume the gripper's base frame $\{E\}$ is at ${}^B\mathbf{p}_E$, and the gripper tip frame $\{G\}$ is at ${}^B\mathbf{p}_G$, hence the collision object for the gripper is a cylinder of radius R_G between these two points, along $\mathbf{X}_E$ (Figure 22.4).

Point $c \in C_B$, lies within the collision cylinder if

$$0 \leq r \leq 1 \tag{22.39}$$

where

$$r = \frac{(\mathbf{p}_E - \mathbf{p}_G) \cdot (\mathbf{p}_E - c)}{|\mathbf{p}_G - \mathbf{p}_E|} \tag{22.40}$$

and if

$$d \leq R_G \tag{22.41}$$

where

$$d = |(\mathbf{p}_G - \mathbf{p}_E)\, r + \mathbf{p}_E - c| \tag{22.42}$$

noting that $\mathbf{p}_G$ and $\mathbf{p}_E$ are all derived from x. We can now express (22.39) and (22.41) as inequality constraints,

$$g_{(3,3n-2)}(x) = -r(x) \tag{22.43}$$
$$g_{(3,3n-1)}(x) = r(x) - 1 \tag{22.44}$$
$$g_{(3,3n)}(x) = d(x) - R_G \tag{22.45}$$

for every n-th point $c \in C_B$. Therefore point c is within the collision cylinder (in collision with the gripper) if any of (22.43)–(22.45) do not satisfy the condition in (22.28). All points in C_B must be outside of the collision cylinder for the collision-free constraint to be satisfied.

22.3.4 Kinematic Configuration Constraint

We can generalise the configuration matching constraint for all non-redundant serial manipulators that exhibit more than one working mode. Let $S = \{1, 2, ...n\}$, which represents n-kinematic solutions of a robotic manipulator, where each integer represents a unique class of configuration of the robot (i.e., working mode [34, 35]). Then let $P = \{S_1, S_2, ..., S_m\}$, which represents a path for the gripper to follow along m points, and $S_i \in P, S_i \subseteq S$ represents the valid kinematic solutions at point i along path P. Then, the goal of manipulator planning is to maintain the same class of configuration through all points in P, such that

$$\left|\bigcap P\right| > 0 \tag{22.46}$$

In task space path planning such as that executed in grasping trajectories, (22.46) must be true in order to avoid serial singularities. Otherwise, the plan is immediately forfeited as infeasible. However, in joint space path planning, if (22.46) is true, then the shortest path in the joint space *can* exists in the absence of environmental and self-collisions. Otherwise if false, any path found along P will require the manipulator to change kinematic configurations, increasing cycle time.

22.4 Path Planning

Path planning involves generating the harvesting sequence for a single apple, which is an amalgamation of trajectories defined either in the task space or in the configuration space. Configuration space planning allows the full utilisation of the configuration space in motion generation, where planning algorithms generate collision-free trajectories in higher-dimensional spaces in a feasible time [36]. In the context of path planning for a non-redundant serial robot, the configuration space is identical to the joint space [35]. Task space planning generates motion relative to the manipulator's end-effector frame and is well-suited for trajectories that require linear motion, such as reaching, inspection, pulling and twisting trajectories.

22.4.1 Planning Algorithm

MARS utilises ROS's MoveIt motion planning framework to implement constrained task space motion and configuration space trajectories. The planning algorithm for configuration space trajectories uses a combination of RRT-Connect and PRM*. While RRT-Connect works very efficiently in configuration spaces where the collision object is complex or free space is limited [37], PRM* offers smoother or shorter configuration space trajectories where processing speed is non-critical [38].

In general, MoveIt works *out-of-the-box* for most serial manipulators utilising numerical kinematic solvers [23]. However, path planning performance can be significantly enhanced by directly utilising configuration space coordinates from an external inverse kinematics solver. This bypasses ROS (Melodic) MoveIt's limitations on linear trajectories by checking for serial singularities in advance, significantly improving the robustness of this mode of trajectory generation. The same advantage also applies to configuration space planning, where trajectories within the same working mode can reduce actuator travel, reducing execution times and speeding up the harvesting process.

Figure 22.5 shows an example of how RRT finds a path for the gripper to grasp a fruit while avoiding obstacles represented by the quadtree structure. The final path is shown in

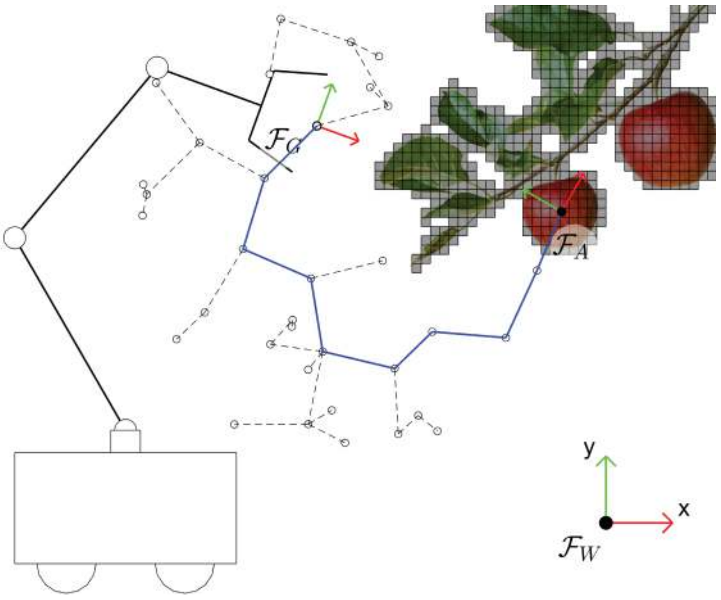


FIGURE 22.5
RRT finding a collision-free path between $\{G\}$ and $\{A\}$.

blue. While randomised forward propagation of the path results in a tree-like structure that can produce indirect paths and long branch spurs, the generation of this tree is extremely quick.

22.4.2 Harvesting Sequence

In Figure 22.6, two gripper frames are defined: $\{apple\_approach\}$ and $\{gripper\_grasp\}$, which are attached to the end effector frame $\{ee\_link\}$ at a static offset. A grasping trajectory can be defined by first manipulating the gripper such that $\{apple\_approach\}$ is coincident with the apple frame $\{apple\_0\}$ ($\{apple\}$ for brevity). Once this is achieved, the robot can then grasp the apple by manipulating the gripper such that $\{gripper\_grasp\}$ is now coincident with $\{apple\}$. This concept is carried through the entire harvesting sequence for a single apple.

MARS utilises a five-step harvesting sequence that combines both configuration space and task space trajectories. This sequence was determined through experiments and field trials, and was found to be the most effective based on the current gripper hardware. To ensure a path for the harvesting sequence exists, and the effects of singularities are minimised, the apple frame's orientation can be optimised, as discussed in Section 22.3. This is essential for apples located at the boundary of the manipulator's workspace, as it ensures the existence of at least one kinematic solution for all via points in the harvesting sequence. Details of the harvesting sequence are as follows.

1. Approach

The robot approaches the apple by manipulating the end effector such that $\{apple\_approach\}$ is coincident with $\{apple\}$, where $\{apple\_approach\}$ is attached to $\{ee\_link\}$ at a constant offset (Figure 22.7). This path is planned in the configuration space from the robot's initial state to the apple approach pose as determined by inverse

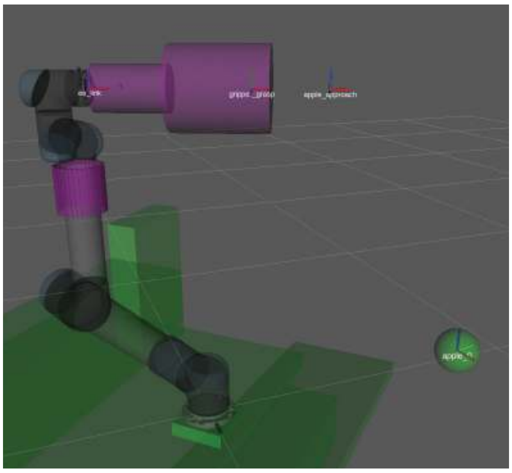


FIGURE 22.6
$\{gripper\_home\}$ coincident with $\{apple\}$.

kinematics. This step determines the robot's working mode for the rest of the harvesting sequence.

2. Reach

The robot reaches for the apple by manipulating $\{gripper\_grasp\}$ to be coincident with $\{apple\}$ (Figure 22.8. A linear trajectory is generated in the task space and executed with the gripper's fingers in the open position. Once the frames are coincident, the fingers will close to envelop the apple. The manipulator working mode between the initial and final pose must be identical to minimise the chance of encountering singularities during motion.

3. Twist

The robot applies a twisting motion to detach the apple from the canopy using a task space trajectory. There are two modes of twisting that can be implemented based on the

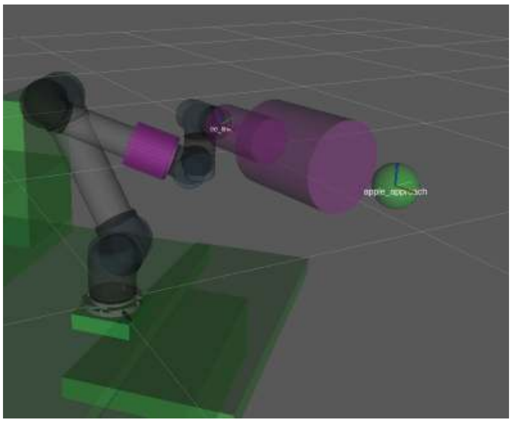


FIGURE 22.7
$\{gripper\_approach\}$ coincident with $\{apple\}$.

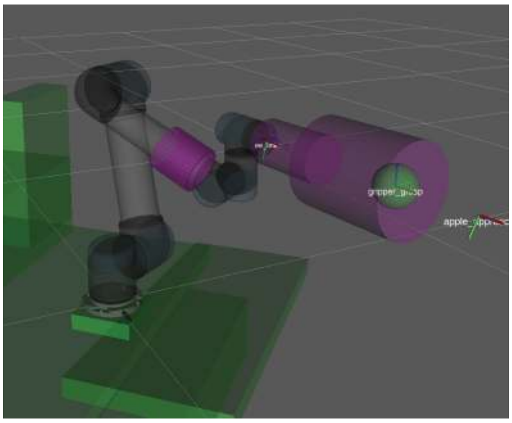
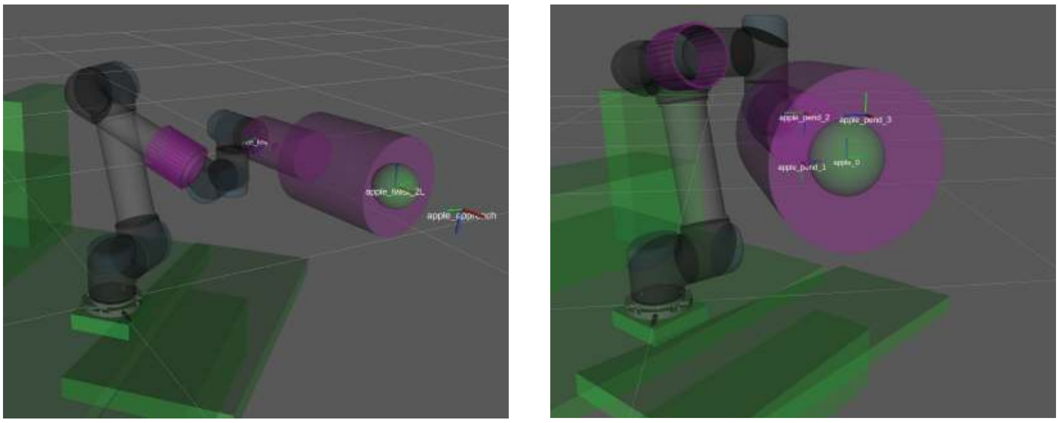


FIGURE 22.8
{*gripper_grasp*} coincident with {*apple*}.

gripper hardware. By default, a purely twisting motion of approximately 270° is applied to the apple, which is sufficient to detach most apples. This is achieved when {*gripper_twist*} coincident with {*apple*} (Figure 22.9(a)). An alternate method is to apply a pendulum motion, where the trajectory is defined as via points attached to the {*ee_link*} (Figure 22.9(b)). This motion allows for the detachment of apples at lower forces. However, it increases translational movement, which can be problematic for apples that lie in clusters or are heavily occluded by branches.

4. Extract

The robot extracts the apple from the canopy with a linear motion generated in the task space, and is mostly a pulling motion with a slight twist. This allows the apple to detach from the canopy if the previous twisting motion was insufficient to break the stem. In this pose, it is assumed that the apple is inside the gripper's fingers. Therefore {*apple_withdraw*}



(a) {*gripper_twist*} coincident with {*apple*}

(b) Ordered pendulum via-frames to apply pendulum motion

FIGURE 22.9
Twisting motions applied to the apple.

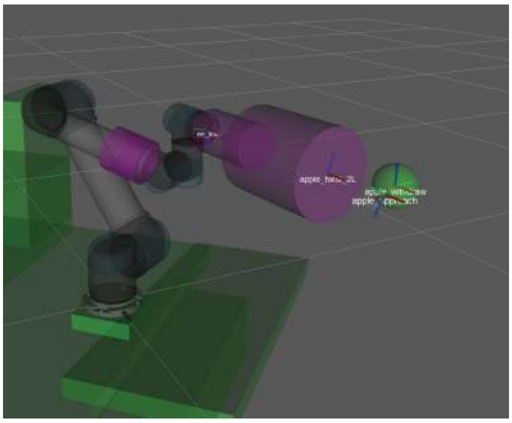


FIGURE 22.10
{*apple_withdraw*} coincident with {*apple*}.

is coincident with the original location of {*apple*} (Figure 22.10). The manipulator working mode between the initial and final pose, again must be identical to avoid problems with singularities.

5. Deposit

The robot moves to the deposit pose by aligning the gripper's {*gripper_tip*} with the {*apple_deposit*}, located above the apple's depositing location (Figure 22.11). This path is generated in the configuration space to utilise the maximal workspace, reducing the chance of collisions with the canopy. In this step, the working mode is unconstrained, allowing the robot to utilise the full configuration space to avoid collisions as necessary. However, if the same working mode is maintained from extraction to deposit, the overall depositing trajectory is optimised, reducing overall cycle time.

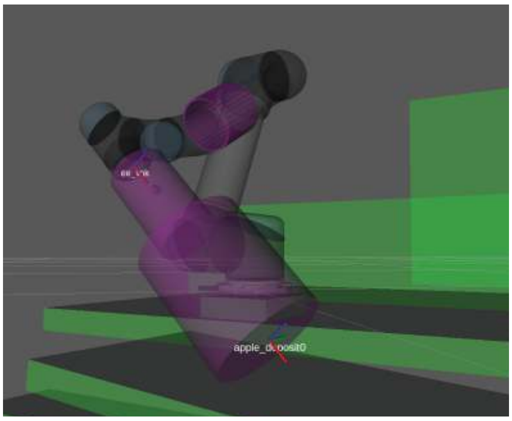


FIGURE 22.11
{*gripper_tip*} coincident with {*apple_deposit*}.

22.5 Results

The goal of the trials was to obtain a *realistic* representation of harvesting performance in typical canopy settings. For all experiments, the only assistance given to MARS was the removal of some leaves to allow the vision system to see completely occluded apples and remove apples from the harvesting cycle that would be dangerous for the robot to pick, such as those behind trellis wires. Given that this assistance was a minimal effort for the robot's operator, MARS' performance values, as shown in Tables 22.2 and 22.3, offer a reasonable insight into selective robotic harvesting performance for *minimally-modified* Tatura-trellis (V) canopies, if farmers had utilised this technology as state-of-the-art in 2022.

Overall, MARS achieved a harvest success rate of 62.8% at a median speed of 9.18 seconds per apple (approximately 6.5 apples per minute) at up to 80% of the manipulator's rated speed. In optimal settings, MARS was capable of higher successful grasping rates, 70.8% when not considering occluded vision, and achieved a faster cycle time where 25% of apples were harvested in less than 8 seconds.

22.5.1 Harvest Success Rate

Table 22.2 shows the unsuccessful apple extraction data for all trials combined. Each mode of failure can be classed into five categories.

- **User** failure mode indicates apples that were incorrectly identified by the user as feasible for harvest, but extraction was still attempted. Unsighted occlusions such as branches and trellis wires were common problems. In addition, system stop problems usually occurred when the gripper's fingers interacted forcibly with unseen branches, or due to pneumatic cable entanglements.

TABLE 22.2
Overall Unsuccessful Apple Extractions in the 2021–2022 Trials

			Failure Mode				
	Apples Attempted	Dropped due to overripeness	User fault	Apple shift	Vision inaccuracy	Grasp interference	Extraction
	806	3	31	39	48	40	160
Visible	803	Excluded	3.86 %	4.86 %	5.98 %	4.98 %	19.93 %
			39.60 % (60.40 % success)				
Reachable	772	Excluded		5.05 %	6.22 %	5.18 %	20.73 %
				37.18 % (62.82 % success)			
Graspable	685	Excluded				5.84 %	23.36 %
						29.20 % (70.80 % success)	

TABLE 22.3
MARS Timing Data for Each Task During Harvesting for Each Apple

Task		Quartile Time (s)		
		Qtr 1	Qtr 2	Qtr 3
Plan	Optimisation	0.63	0.67	0.75
	Grasp Sequence	1.06	1.14	1.20
	Approach	0.10	0.20	0.30
	Grasp	0.30	0.30	0.30
	Deposit	0.20	0.30	0.30
Execution	Approach	2.28	3.13	3.84
	Grasp	1.08	1.08	1.09
	Deposit	3.19	3.65	4.03
	Gripper	0.94	1.01	1.11
	Cycle Time	7.91	9.18	10.31

- **Apple shift** failures occur when the target apple's position shifts in the canopy. MARS currently assumes the positions of the apples stay relatively static through the harvesting process, and will not update its map even if apples change positions due to the robot's interaction with the canopy. During harvest, the robot can accidentally knock an adjacent apple off its branch, or shift its parent branch, causing all other child apples to move in the process. Over-compliant branches are also problematic, as extraction causes them to oscillate for an extended period of time.

- **Vision inaccuracy** problems cause the robot to partially grasp or completely miss the target fruit. Due to the dependence of the point cloud in the localisation process, any stray leaves and branches in front of a targeted fruit can affect its depth perception. Other sources of problems include clustering and extreme lighting conditions. They cause localisation errors that affect translational accuracy, and are classed as "Planar" inaccuracies.

- **Grasp interference** failures occur when the gripper's accuracy is ideal, but the grasping process is interfered with, resulting in a failed grasp. This usually involves physical interference during the grasping process, such as from branches, other apples, or even parts of the gripper hardware.

- **Extraction** failures occur where upon successful grasp, the gripper trajectory executed by the manipulator fails to detach the apple from the canopy. There are many causes of extraction failure, involving both hardware and software aspects.

Further to these categories, apples can be grouped under differing assumptions that allow further evaluation of MARS' harvest success rates. These are displayed as three main rows in Table 22.2, which are labelled as *Visible*, *Reachable*, and *Graspable* apples.

- **Visible apples** represents all apples that were detected by MARS that were selected to be harvested by the robot, excluding the apples that were unsuccessfully harvested due to apples prematurely detaching from the canopy naturalistically as part of their ripening process. This resulted in MARS having a 60.4% harvest success rate, when including all sources of unsuccessful grasps.

- **Reachable apples** represent the same sample as visible apples, but exclude all sources of unsuccessful grasps that were caused by the user. During the manual selection process, the user may unintentionally allow an apple that lies behind a trellis wire or situated next to a hidden branch to be harvested. In these scenarios, finger interference is highly likely, resulting in an unsuccessful grasp. Other reasons include attempts at grasping apples that are too small for the designed gripper, or if the robot halts unexpectedly due to software faults or protective stops, regardless of whether an apple was successfully grasped or not. The resultant calculated harvest success rate in this category represents MARS' entire system performance, which is 62.82% out of 772 potentially reachable apples.

- **Graspable apples** represent the same apples as the reachable set, but further exclude all sources of unsuccessful grasps caused by the vision system, such as localisation errors and unexpected shifting of apples caused by the robot interacting with the canopy. These results best indicate MARS' fruit-picking performance on the assumption that the vision system and user operation are perfect. The resultant harvest success rate under this assumption is 70.8%.

22.5.2 Cycle Time

In addition to the logged unsuccessful grasps, a further number of apples had their cycle times thoroughly analysed. The definition of cycle time is the time it takes for the robot to approach an apple from any resting position, grasp, extract, and then deposit the apple at a designated position. It is *not* assumed that the depositing condition is simply dropping the apple after extracting it from the canopy. This cycle can be grouped into two categories for timing purposes: planning and execution. Planning largely occurs while the robot is in transit (during the approach and deposit phases); hence, planning tasks have minimal impact on the overall cycle time. Therefore, the summed time of execution tasks should be approximately equal to the cycle time, with any time discrepancies caused by the communication delays between ROS and the robot's hardware.

A summary of time spent on tasks during the harvesting process is shown in Table 22.3. The times are expressed as lower (Qtr 1) and upper (Qtr 3) quartiles (25th and 75th percentile), and middle quartile (Qtr 2), which represents median time. Based on this, MARS achieved a median harvesting cycle time of 9.18 seconds for Tatura-structured canopies with *very minimal* leaf or branch thinning to assist in robotic harvesting, with the manipulator operating at up to 80% of the rated maximum joint speed.

22.6 Conclusion

The significance of these results is that it is a realistic representation of selective apple harvesting performance under typical apple canopy conditions, where minimal attempts were made to modify the existing canopy to assist in robotic harvesting. These results are promising, even at the prototyping stage, and show that the planning methods implemented are a step in the right direction for enhancing the reliability of selective apple harvesting robots in challenging canopy environments. In application, this will reduce the labour burden on farmers who are willing to adopt robotic harvesting, but may not have the resources to prepare idealistic canopy conditions for a robot to operate efficiently.

MARS was shown to be particularly gentle in harvesting apples from the canopy with negligible fruit and canopy damage. This sort of finesse is typical of a robot that operates under the paradigm of accuracy and efficiency that is not commonly observed in agricultural robots, but is a trait that is demanded by farmers from their workers. By focusing on this paradigm for selective harvesting robots and enhancing their intelligence and reliability in typical canopy conditions, it is hoped that farmers will begin to change their perception of these robots, from being expensive farming machinery that requires constant maintenance and attention, to a platform they can rely on to supplement their harvesting capacity.

Bibliography

[1] S. Fountas, N. Mylonas, I. Malounas, E. Rodias, C. Hellmann Santos, and E. Pekkeriet, “Agricultural robotics for field operations,” *Sensors*, vol. 20, no. 9, p. 2672, 2020.

[2] T. L. Robinson, “Apple-orchard planting systems.,” in *Apples: Botany, Production and Uses*, pp. 345–407, CABI publishing Wallingford UK, 2003.

[3] G. Kootstra, X. Wang, P. M. Blok, J. Hemming, and E. Van Henten, “Selective harvesting robotics: current research, trends, and future directions,” *Current Robotics Reports*, pp. 1–10, 2021.

[4] H. Kang and C. Chen, “Fruit detection and segmentation for apple harvesting using visual sensor in orchards,” *Sensors*, vol. 19, no. 20, p. 4599, 2019.

[5] H. Kang and C. Chen, “Fast implementation of real-time fruit detection in apple orchards using deep learning,” *Computers and Electronics in Agriculture*, vol. 168, p. 105108, 2020.

[6] H. Kang, H. Zhou, and C. Chen, “Visual perception and modeling for autonomous apple harvesting,” *IEEE Access*, vol. 8, pp. 62151–62163, 2020.

[7] X. Wang, H. Zhou, H. Kang, W. Au, and C. Chen, “Bio-inspired soft bistable actuator with dual actuations,” *Smart Materials and Structures*, vol. 30, no. 12, p. 125001, 2021.

[8] X. Wang, H. Kang, H. Zhou, W. Au, and C. Chen, “Geometry-aware fruit grasping estimation for robotic harvesting in apple orchards,” *Computers and Electronics in Agriculture*, vol. 193, p. 106716, 2022.

[9] J. Baeten, K. Donné, S. Boedrij, W. Beckers, and E. Claesen, “Autonomous fruit picking machine: A robotic apple harvester,” in *Field and service robotics*, pp. 531–539, Springer, 2008.

[10] Z. De-An, L. Jidong, J. Wei, Z. Ying, and C. Yu, “Design and control of an apple harvesting robot,” *Biosystems engineering*, vol. 110, no. 2, pp. 112–122, 2011.

[11] J. R. Davidson, A. Silwal, C. J. Hohimer, M. Karkee, C. Mo, and Q. Zhang, “Proof-of-concept of a robotic apple harvester,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 634–639, IEEE, 2016.

[12] A. Silwal, J. R. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis, “Design, integration, and field evaluation of a robotic apple harvester,” *Journal of Field Robotics*, vol. 34, no. 6, pp. 1140–1159, 2017.

[13] E. A. Smith, R. L. Bettinger, C. A. Bishop, V. Blundell, E. Cashdan, M. J. Casimir, A. L. Christenson, B. Cox, R. Dyson-Hudson, B. Hayden, *et al.*, “Anthropological applications of optimal foraging theory: a critical review [and comments and reply],” *Current Anthropology*, vol. 24, no. 5, pp. 625–651, 1983.

[14] H. Zhou, X. Wang, W. Au, H. Kang, and C. Chen, "Intelligent robots for fruit harvesting: Recent developments and future challenges," *Precision Agriculture*, pp. 1–52, 2022.

[15] L. F. Oliveira, A. P. Moreira, and M. F. Silva, "Advances in agriculture robotics: A state-of-the-art review and challenges ahead," *Robotics*, vol. 10, no. 2, p. 52, 2021.

[16] K. Legun and K. Burch, "Robot-ready: How apple producers are assembling in anticipation of new ai robotics," *Journal of Rural Studies*, vol. 82, pp. 380–390, 2021.

[17] S. K. Devitt, "Cognitive factors that affect the adoption of autonomous agriculture," *arXiv preprint arXiv:2111.14092*, 2021.

[18] R. R Shamshiri, C. Weltzien, I. A. Hameed, I. J Yule, T. E Grift, S. K. Balasundram, L. Pitonakova, D. Ahmad, and G. Chowdhary, "Research and development in agricultural robotics: A perspective of digital farming," *International Journal of Agricultural and Biological Engineering*, 2018.

[19] K. Zhang, K. Lammers, P. Chu, Z. Li, and R. Lu, "System design and control of an apple harvesting robot," *Mechatronics*, vol. 79, p. 102644, 2021.

[20] X. Yu, Z. Fan, X. Wang, H. Wan, P. Wang, X. Zeng, and F. Jia, "A lab-customized autonomous humanoid apple harvesting robot," *Computers & Electrical Engineering*, vol. 96, p. 107459, 2021.

[21] Z. Zhang, P. H. Heinemann, J. Liu, T. A. Baugher, and J. R. Schupp, "The development of mechanical apple harvesting technology: A review," *Transactions of the ASABE*, vol. 59, no. 5, pp. 1165–1180, 2016.

[22] S. M. LaValle, *Planning algorithms.* Cambridge university press, 2006.

[23] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[24] B. Chen, "Design and simulation of path research for fruit picking robot based on data mining," *Agronomia*, vol. 37, no. 1, 2020.

[25] Y. Edan, T. Flash, U. M. Peiper, I. Shmulevich, and Y. Sarig, "Near-minimum-time task planning for fruit-picking robots," *IEEE transactions on robotics and automation*, vol. 7, no. 1, pp. 48–56, 1991.

[26] P. Kurtser and Y. Edan, "Planning the sequence of tasks for harvesting robots," *Robotics and Autonomous Systems*, vol. 131, p. 103591, 2020.

[27] M. Leighton and D. R. Leighton, "The relationship of size of feeding aggregate to size of food patch: howler monkeys (alouatta palliata) feeding in trichilia cipo fruit trees on barro colorado island," *Biotropica*, pp. 81–90, 1982.

[28] J. F. Aristizabal, S. Negrete-Yankelevich, R. Macías-Ordóñez, C. A. Chapman, and J. C. Serio-Silva, "Spatial aggregation of fruits explains food selection in a neotropical primate (alouatta pigra)," *Scientific reports*, vol. 9, no. 1, pp. 1–13, 2019.

[29] A. Koirala, K. B. Walsh, Z. Wang, and C. McCarthy, "Deep learning–method overview and review of use for fruit detection and yield estimation," *Computers and electronics in agriculture*, vol. 162, pp. 219–234, 2019.

[30] H. Kang, H. Zhou, X. Wang, and C. Chen, "Real-time fruit recognition and grasping estimation for robotic apple harvesting," *Sensors*, vol. 20, no. 19, p. 5670, 2020.

[31] P. Eizentals and K. Oka, "3D pose estimation of green pepper fruit for automated harvesting," *Computers and Electronics in Agriculture*, vol. 128, pp. 127–140, 2016.

[32] C. W. Bac, J. Hemming, B. Van Tuijl, R. Barth, E. Wais, and E. J. van Henten, "Performance evaluation of a harvesting robot for sweet pepper," *Journal of Field Robotics*, vol. 34, no. 6, pp. 1123–1139, 2017.

[33] Y. Xiong, Y. Ge, L. Grimstad, and P. J. From, "An autonomous strawberry-harvesting robot: Design, development, integration, and field evaluation," *Journal of Field Robotics*, vol. 37, no. 2, pp. 202–224, 2020.

[34] K. P. Hawkins, "Analytic inverse kinematics for the Universal Robots UR5/UR10 arms," tech. rep., Georgia Institute of Technology, 2013.

[35] W. Au, H. Chung, and C. Chen, "Generation of the global workspace roadmap of the 3-RPR using rotary disk search," *Mechanism and Machine Theory*, vol. 78, pp. 248–262, 2014.

[36] W. Au, H. Chung, and C. Chen, "Path planning and assembly mode-changes of 6-DoF stewart-gough-type parallel manipulators," *Mechanism and Machine Theory*, vol. 106, pp. 30–49, 2016.

[37] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.

[38] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, vol. 1, pp. 521–528, IEEE, 2000.

[39] R. Pentreath, "AP13035 Final Report - Apple and pear industry data collection," tech. rep., Horticulture Innovation Australia, 2015.

[40] H. Zhou, J. Xiao, H. Kang, X. Wang, W. Au, and C. Chen, "Learning-based slip detection for robotic fruit grasping and manipulation under leaf interference," *Sensors*, vol. 22, no. 15, p. 5483, 2022.

Part IV

Appendix

23

Appendices for Chapter 5

23.1 Angle-Axis

It is known that any two arbitrary orientations can be linked by a rotation around a fixed axis with an angle. Such a phenomenon yields the representation of orientation by using the fixed axis and angle, which is called the angle-axis representation. Again, we will discuss the direct and inverse problems separately.

23.1.1 Forward Problem

Consider a rotation axis being a unit vector $\mathbf{k} = \left[k_x, k_y, k_z\right]^T$ and the rotation angle being θ. The rotation matrix is given by

$$\mathbf{R}_k = \mathbf{k}\mathbf{k}^T + \cos\theta(\mathbf{1} - \mathbf{k}\mathbf{k}^T) + \sin\theta\mathbf{K} \tag{23.1}$$

where $\mathbf{K}$ is the *cross-product matrix* of $\mathbf{k}$, such that

$$\mathbf{K} = [\mathbf{k}\times] = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix} \tag{23.2}$$

Proof: consider a vector $\mathbf{p}$ and $\mathbf{p}'$ that are the same vector before and after rotation, as shown in Figure 23.1(a). Using the second meaning of rotation matrix, we have

$$\mathbf{p}' = \mathbf{R}_k\,\mathbf{p} \tag{23.3}$$

where $\mathbf{R}_k$ is the matrix we are seeking. According to Figure 23.1(a), we have the following relations:

$$\mathbf{p}' = \mathbf{l} + \mathbf{R}' = (\mathbf{k}^T\mathbf{p})\mathbf{k} + \mathbf{R}' = \mathbf{k}\mathbf{k}^T\mathbf{p} + \mathbf{R}' \tag{23.4}$$

$$\mathbf{R} = \mathbf{p} - \mathbf{l} = (\mathbf{1} - \mathbf{k}\mathbf{k}^T)\mathbf{p} \tag{23.5}$$

where $\mathbf{1}$ is a 3×3 identity matrix. On the other hand, $\mathbf{R}'$ can be represented in terms of $\mathbf{R}$ and $\mathbf{R}''$, as shown in Figure 23.1(b), such that

$$\mathbf{R}' = c\theta\,\mathbf{R} + s\theta\,\mathbf{R}'' \tag{23.6}$$

where $\mathbf{R}'' = \mathbf{k} \times \mathbf{p}$. Combining the above equations together yields

$$\mathbf{p}' = \mathbf{k}\mathbf{k}^T\mathbf{p} + \cos\theta(\mathbf{1} - \mathbf{k}\mathbf{k}^T)\mathbf{p} + \sin\theta\mathbf{k} \times \mathbf{p} \tag{23.7}$$

which is called Rodrigues's rotation formula. Comparing (23.3) and (23.7) yields (23.1).

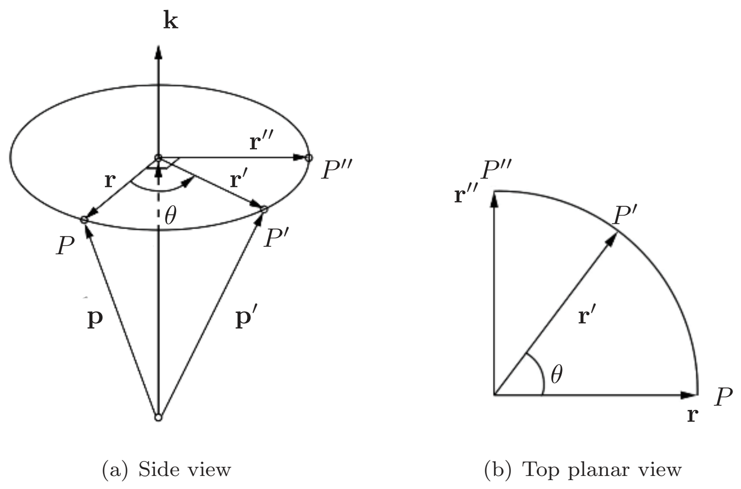


FIGURE 23.1
A vector is rotated from its original position to the final position.

□

Expanding (23.1) gives

$$\mathbf{R}_k = \begin{bmatrix} k_xk_x(1-c\theta)+c\theta & k_xk_y(1-c\theta)-k_zs\theta & k_xk_z(1-c\theta)+k_ys\theta \\ k_xk_y(1-c\theta)+k_zs\theta & k_yk_y(1-c\theta)+c\theta & k_yk_z(1-c\theta)-k_xs\theta \\ k_xk_z(1-c\theta)-k_ys\theta & k_yk_z(1-c\theta)+k_xs\theta & k_zk_z(1-c\theta)+c\theta \end{bmatrix} \tag{23.8}$$

In linear algebra, we have

$$\mathbf{kk}^T = \mathbf{1} + \mathbf{K}^2 \tag{23.9}$$

where **K** is the cross product of matrix k. By means of (23.9), Rodrigues' formula (23.1) can also be written as

$$\begin{aligned} \mathbf{R_k} &= \mathbf{kk}^T + \cos\theta(\mathbf{1} - \mathbf{kk}^T) + \sin\theta\ \mathbf{K} \\ &= \mathbf{1} + \mathbf{K}^2 - \cos\theta\ \mathbf{K}^2 + \sin\theta\ \mathbf{K} \\ &= \mathbf{1} + (1-\cos\theta)\mathbf{K}^2 - \sin\theta\ \mathbf{K} \end{aligned} \tag{23.10}$$

For further details on deriving (23.10), please refer to Appendix 23.3.

23.1.2 Inverse Problem

Inverse problem is again to the rotation axis and the angle of rotation upon a given matrix,

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{23.11}$$

The solution can be readily obtained as

$$\begin{aligned} \mathbf{k} &= \frac{1}{2\sin\theta}\begin{bmatrix} r_{32}-r_{23} \\ r_{13}-r_{31} \\ r_{21}-r_{12} \end{bmatrix} \\ \theta &= \arccos\frac{r_{11}+r_{22}+r_{33}-1}{2} \end{aligned} \tag{23.12}$$

where **k** can be obtained by normalising the vector

$$\begin{bmatrix} r_{32} - r_{23} & r_{13} - r_{31} & r_{21} - r_{12} \end{bmatrix}^T \tag{23.13}$$

without acquiring the information of θ. Note that the solution is valid for $\theta \in (0°, 180°)$. When $\theta = 0°$, the rigid body remains stationary, and any axis is possible. When $\theta = 180°$, the axis has two opposite choices.

Example 23.1 (Angle-axis rotation): Find out the axis and angle of rotation of the rotation matrix below.

$$^0\mathbf{R}_1 = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{23.14}$$

Solution: According to (23.12), we have

$$\mathbf{k} = \frac{1}{\| \begin{bmatrix} -1 & 1 & -1 \end{bmatrix}^T \|} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \frac{\sqrt{3}}{3} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

$$\theta = \arccos \frac{-1}{2} = 120°$$

23.2 Quaternion

An orientation can be best represented by a point on a *3-sphere* in $\mathbb{R}^4$,

$$a^2 + b^2 + c^2 + d^2 = 1 \tag{23.15}$$

where a, b, c, d are called the quaternion. The corresponding rotation matrix is given by

$$\mathbf{R} = \begin{bmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2bd + 2ac \\ 2bc + 2ad & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2cd + 2ab & a^2 - b^2 - c^2 + d^2 \end{bmatrix} \tag{23.16}$$

The inverse problem involves finding the quaternion based on a given rotation matrix

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{23.17}$$

where the solution can be readily found as

$$a = \frac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1} \tag{23.18}$$

$$b = \frac{r_{32} - r_{23}}{4a} \tag{23.19}$$

$$c = \frac{r_{13} - r_{31}}{4a} \tag{23.20}$$

$$d = \frac{r_{21} - r_{12}}{4a} \tag{23.21}$$

The quaternion is a powerful representation of the orientation, because two rotations can be combined as a linear operation defined by one quaternion on the other quaternion. Consider two rotations represented by two quaternions, $\mathbf{q}_1 = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \end{bmatrix}^T$ and $\mathbf{q}_2 = \begin{bmatrix} a_2 & b_2 & c_2 & d_2 \end{bmatrix}^T$, respectively. The combined rotation can be represented by a new quaternion $\mathbf{q}_{12}$ given by

$$\mathbf{q}_{12} = \mathbf{q}_1 \circ \mathbf{q}_2 = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ -b_1 & a_1 & -d_1 & c_1 \\ -c_1 & d_1 & a_1 & -b_1 \\ -d_1 & -c_1 & b_1 & a_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} \tag{23.22}$$

which requires minimum computations and storage of the orientations. The relation between the quaternion and the angle-axis representation is given by

$$a = \cos(\theta/2) \tag{23.23}$$
$$b = k_x \sin(\theta/2) \tag{23.24}$$
$$c = k_y \sin(\theta/2) \tag{23.25}$$
$$d = k_z \sin(\theta/2) \tag{23.26}$$

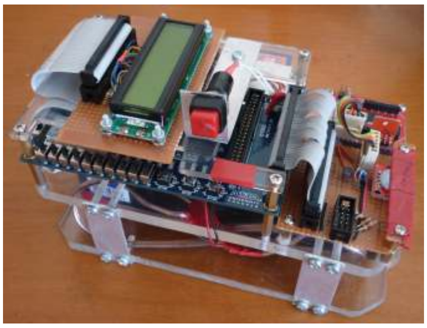
which are called the Euler parameters as well. This relation can be used to visualise the rotation by converting a given quaternion into the angle and axis of the rotation.

Example 23.2 (Quaternion rotations): An inertial measurement unit (IMU) shown in Figure 23.2(a) is an electronic device that measures and reports on its velocity, orientation, and gravitational forces, using a combination of accelerometers and gyroscopes. These information can be used to assist a SCUBA diver to map underwater caves, as shown in Figure 23.2(b). In one experiment, the angular velocities $\omega_x(t_i), \omega_y(t_i), \omega_z(t_i)$, around the on-board x, y, and z axes, are recorded, respectively, for $i = 0, \ldots, n$. The sampling time Δt is constant and known. Find the trajectory of the orientation of IMU with respect to its initial orientation $\mathbf{R}(t_0)$.

Solution: Let

$$\omega(t_i) = \sqrt{\omega_x(t_i)^2 + \omega_y(t_i)^2 + \omega_z(t_i)^2} \tag{23.27}$$

where ω is the magnitude of angular velocity, and $\omega_x(t_i)$, $\omega_y(t_i)$, and $\omega_z(t_i)$ refer to the x, y, and z components of angular velocity, respectively, at time t_i. According to the representation of


(a) An IMU

(b) Mapping an underwater cave

FIGURE 23.2
A SCUBA driver mapping a underwater cave.

angle-axis (Section 23.1), $\mathbf{k} = \begin{bmatrix} k_x & k_y & k_z \end{bmatrix}^T$. Its cross product matrix $\mathbf{K}$ is given by

$$\mathbf{K}(t_i) = \frac{1}{\omega(t_i)} \begin{bmatrix} 0 & -\omega_z(t_i) & \omega_y(t_i) \\ \omega_z(t_i) & 0 & -\omega_x(t_i) \\ -\omega_y(t_i) & \omega_x(t_i) & 0 \end{bmatrix} \tag{23.28}$$

where δt is the time step. We can then apply Rodrigues' formula (23.10) to calculate the rotation matrix between the initial orientation and orientation at the next time step t_{i+1}, ${}^{i-1}_{i}R$

$$^{i-1}\mathbf{R}_i = \mathbf{1} + (1 - \cos(\omega(t_i)\delta t))\,\mathbf{K}^2 - \mathbf{K}(t)\sin(\omega(t_i)\delta t) \tag{23.29}$$

Finally, we can calculate the rotation matrix between initial orientation and orientation at time t_i by

$$\begin{aligned} {}^0\mathbf{R}_i &= {}^0\mathbf{R}_1 \ \ldots \ {}^{i-1}\mathbf{R}_i \\ &= {}^0\mathbf{R}_{i-1} \ {}^{i-1}\mathbf{R}_i \end{aligned} \tag{23.30}$$

23.3 Alternate Expression of Rodrigues' Formula

$$\begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix} \begin{bmatrix} k_x & k_y & k_z \end{bmatrix} = \begin{bmatrix} k_x{}^2 & k_x k_y & k_x k_z \\ k_x k_y & k_y{}^2 & k_y k_z \\ k_x k_z & k_y k_z & k_z{}^2 \end{bmatrix} = \mathbf{k}\ \mathbf{k}^T \tag{23.31}$$

$$\begin{aligned} \mathbf{K}^2 &= \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix} \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix} \\ &= \begin{bmatrix} -k_z{}^2 - k_y{}^2 & k_x k_y & k_x k_z \\ k_x k_y & -k_z{}^2 - k_x{}^2 & k_y k_z \\ k_x k_z & k_y k_z & -k_y{}^2 - k_x{}^2 \end{bmatrix} \end{aligned} \tag{23.32}$$

$$\mathbf{k}\ \mathbf{k}^T - \mathbf{K}^2 = \begin{bmatrix} k_x{}^2 + k_y{}^2 + k_z{}^2 & 0 & 0 \\ 0 & k_x{}^2 + k_y{}^2 + k_z{}^2 & 0 \\ 0 & 0 & k_x{}^2 + k_y{}^2 + k_z{}^2 \end{bmatrix} = \mathbf{1} \tag{23.33}$$

$$\therefore \mathbf{k}\ \mathbf{k}^T = \mathbf{1} + \mathbf{K}^2 \tag{23.34}$$

24

Appendices for Chapter 8

24.1 Proof of $\mathbf{Q}\ \mathbf{Q}^T$ Being Screw-Symmetric

$\mathbf{Q}\ \mathbf{Q}^T = \mathbf{1} \Rightarrow d(\mathbf{Q}\ \mathbf{Q}^T)/dt = \mathbf{0} \Rightarrow \dot{\mathbf{Q}}\ \mathbf{Q}^T + \mathbf{Q}\ \dot{\mathbf{Q}}^T = \mathbf{0} \Rightarrow \boldsymbol{\Omega} + \boldsymbol{\Omega}^T = \mathbf{0} \quad \square$

24.2 Proof of a Screw-Symmetric Matrix Being the Crossproduct Matrix of a Vector

Assume $\boldsymbol{\Omega}$ given below being scew-symmetric.

$$\boldsymbol{\Omega} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

According to the skew-symmetric property, we have

$$\boldsymbol{\Omega} + \boldsymbol{\Omega}^T = \begin{bmatrix} 2w_{11} & w_{12} + w_{21} & w_{13} + w_{31} \\ w_{21} + w_{12} & 2w_{22} & w_{23} + w_{32} \\ w_{31} + w_{13} & w_{32} + w_{23} & 2w_{33} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

which yields zero diagonals and $w_{12} = -w_{21}, \quad w_{13} = -w_{31}, \quad w_{23} = -w_{32}$. Therefore, $\boldsymbol{\Omega}$ can be further written as

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 & -w_{21} & w_{13} \\ w_{21} & 0 & -w_{32} \\ -w_{13} & w_{32} & 0 \end{bmatrix} \tag{24.1}$$

(24.1) is called crossproduct matrix of $\boldsymbol{\omega} = \begin{bmatrix} w_{32} & w_{13} & w_{21} \end{bmatrix}^T$, because of the relation below:

$$\boldsymbol{\Omega}\ \mathbf{q} = \boldsymbol{\omega} \times \mathbf{q}$$

where $\mathbf{q}$ is an arbitrary 3D vector. This relation can be readily verified.

Finally, define $\omega_x = w_{32}, \quad \omega_y = w_{13}, \quad \omega_z = w_{21}$ such that $\boldsymbol{\omega} = \begin{bmatrix} w_x & w_y & w_z \end{bmatrix}^T$

$\square$

DOI: 10.1201/9781003614319-24

24.3 Proof of $\boldsymbol{\omega}$

We will show $\boldsymbol{\omega} = \begin{bmatrix}\delta\gamma/\delta t & \delta\beta/\delta t & \delta\alpha/\delta t\end{bmatrix}^T$. Without loss of generality, we can assume that the instantaneous orientation of the end-effector is $\mathbf{R}_0 = \mathbf{1}$, where $\mathbf{1}$ is an identity matrix. The orientation caused by the small derivations of the fixed angles was shown previously as

$$\mathbf{R}_1 = \begin{bmatrix} 1 & -\delta\alpha & \delta\beta \\ \delta\alpha & 1 & -\delta\gamma \\ -\delta\beta & \delta\gamma & 1 \end{bmatrix}$$

Hence, we have the crossproduct matrix of the angular velocity as

$$\boldsymbol{\Omega} = \dot{\mathbf{R}}_0 \, \mathbf{R}_0{}^T = \frac{\mathbf{R}_1 - \mathbf{R}_0}{\delta t} \, \mathbf{R}_0{}^T = \begin{bmatrix} 0 & -\delta\alpha/\delta t & \delta\beta/\delta t \\ \delta\alpha/\delta t & 0 & -\delta\gamma/\delta t \\ -\delta\beta/\delta t & \delta\gamma/\delta t & 0 \end{bmatrix}$$

Therefore, the angular velocity is obtained as

$$\boldsymbol{\omega} = \begin{bmatrix} \delta\gamma/\delta t \\ \delta\beta/\delta t \\ \delta\alpha/\delta t \end{bmatrix}$$

25

Appendices for Chapter 11

ppendices for Chapter 11]Appendices for Chapter 11

25.1 Proof of the Orthogonality of Eigenvectors of an Inertia Tensor

According to (11.9), the inertia tensor is symmetric, i.e., $\mathbf{I}^T = \mathbf{I}$.

Assume two eigenvectors $\mathbf{e}_1$ and $\mathbf{e}_2$ of $\mathbf{I}$ are associated with two distinct eigenvalues λ_1 and λ_2, respectively. We want to show $\mathbf{e}_1^T\ \mathbf{e}_2 = 0$. Start with

$$\lambda_1\ \mathbf{e}_1^T\ \mathbf{e}_2 = (\lambda_1\ \mathbf{e}_1)^T\ \mathbf{e}_2 = (\mathbf{I}\ \mathbf{e}_1)^T\ \mathbf{e}_2 = \mathbf{e}_1^T\ \mathbf{I}^T\ \mathbf{e}_2$$

Since $\mathbf{I}$ is symmetric, we have

$$\lambda_1\ \mathbf{e}_1^T\ \mathbf{e}_2 = \mathbf{e}_1^T\ \mathbf{I}\ \mathbf{e}_2 = \lambda_2\ \mathbf{e}_1^T\ \mathbf{e}_2$$

which yields

$$(\lambda_1 - \lambda_2)\ \mathbf{e}_1^T\ \mathbf{e}_2 = 0$$

Since λ_1 and λ_2 are distinct, we have $\mathbf{e}_1^T\ \mathbf{e}_2 = 0$.

□

25.2 Proof of Euler's Law

The proof starts from the following relation

$$^F\mathbf{n} = {}^F\dot{\mathbf{h}}_c$$

where $\mathbf{n}$ and $\mathbf{h}_c$ are the moment on a rigid body and the angular momentum around its mass center, respectively. Both quantities are measured in the ground frame $\{F\}$. Since $\mathbf{h}_c = {}^F\mathbf{I}_c\ {}^F\boldsymbol{\omega}$, we have

$$^F\mathbf{n} = {}^F\dot{\mathbf{I}}_c\,\boldsymbol{\omega} + {}^F\mathbf{I}_c\,{}^F\dot{\boldsymbol{\omega}}$$

According to the similarity transformation of the inertia tensor, $^F\mathbf{I}_c = {}^F\mathbf{R}_M\ {}^M\mathbf{I}_c\ {}^F\mathbf{R}_M{}^T$, where $\{M\}$ is attached to the body and $^M\mathbf{I}_c$ is invariant. Therefore,

$$\begin{aligned}
{}^F\mathbf{n} &= [{}^F\dot{\mathbf{R}}_M\, {}^M\mathbf{I}_c\, {}^F\mathbf{R}_M{}^T + {}^F\mathbf{R}_M\, {}^M\mathbf{I}_c\, {}^F\dot{\mathbf{R}}_M{}^T]\, {}^F\boldsymbol{\omega} + {}^F\mathbf{I}_c\, {}^F\dot{\boldsymbol{\omega}} \\
&= [{}^F\boldsymbol{\omega} \times {}^F\mathbf{I}_c + {}^F\mathbf{I}_c(-\, {}^F\boldsymbol{\omega}\times)]\, {}^F\boldsymbol{\omega} + {}^F\mathbf{I}_c\, {}^F\dot{\boldsymbol{\omega}} \\
&= {}^F\mathbf{I}_c\, {}^F\dot{\boldsymbol{\omega}} + {}^F\boldsymbol{\omega} \times {}^F\mathbf{I}_c\, {}^F\boldsymbol{\omega} \qquad (25.1)
\end{aligned}$$

Further, (25.1) is valid in any measure frame. Hence, it can be written as

$$\mathbf{n} = \mathbf{I}_c\, \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}_c\, \boldsymbol{\omega} \qquad (25.2)$$

□

25.3 Proof of Parallel-Axis Theorem

Two parallel frames, {0} and {1}, are shown in Figure 25.1, where {1} sits at the CoM of the rigid body. The position vector **r**, **c**, and **d** shown in Figure 25.1 have the geometric relation: $\mathbf{r} = \mathbf{c} + \mathbf{d}$. According to Formula (e6-1-7), we have

$$ {}^0\mathbf{I} = -\int [\mathbf{r}]^2 dm $$

$$ {}^1\mathbf{I} = -\int [\mathbf{d}]^2 dm $$

Substituting the geometric relation into ${}^0\mathbf{I}$ yields

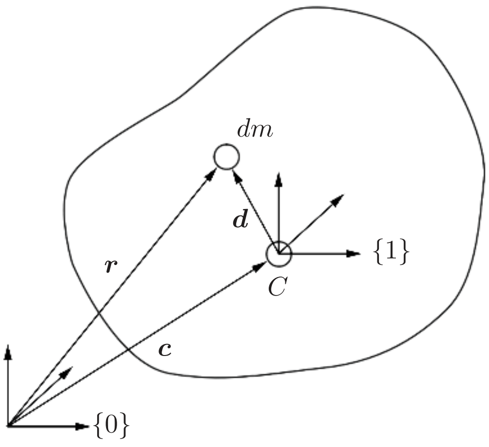


FIGURE 25.1
Two parallel frames associated with a rigid body.

$$\begin{aligned}
{}^0\mathbf{I} &= -\int [\mathbf{c} + \mathbf{d}]^2 dm \\
&= -\int ([\mathbf{c}] + [\mathbf{d}])^2 dm \\
&= -\int ([\mathbf{c}]^2 + [\mathbf{d}]^2 + [\mathbf{c}][\mathbf{d}] + [\mathbf{d}][\mathbf{c}]) dm \\
&= -[\mathbf{c}]^2 \int dm - \int [\mathbf{d}]^2 dm - [\mathbf{c}] \int [\mathbf{d}] dm - \int [\mathbf{d}] dm \; [\mathbf{c}]
\end{aligned} \tag{25.3}$$

Since $\{1\}$ sits at the CoM of the body, we have $\int [\mathbf{d}] dm = \mathbf{0}$. Hence,

$$\begin{aligned}
{}^0\mathbf{I} &= -[\mathbf{c}]^2 M - \int [\mathbf{d}]^2 dm \\
&= -[\mathbf{c}]^2 M + \ {}^C\mathbf{I}
\end{aligned} \tag{25.4}$$

One can readily show $[\mathbf{c}]^2 = -\mathbf{c}^T\ \mathbf{c}\ \mathbf{1}_3 + \mathbf{c}\ \mathbf{c}^T$ by simplifying and expanding this formula. Hence,

$${}^0\mathbf{I} = (\mathbf{c}^T\ \mathbf{c}\ \mathbf{1}_3 - \mathbf{c}\ \mathbf{c}^T) M + \ {}^1\mathbf{I} \tag{25.5}$$

$\square$

Index

3-RRR robot, 374
3-sphere, 324
3R-PRB model, 338

A* (A star) algorithm, 179
    MATLAB function, 180
Acceleration
    angular, 229
    linear, 229
Actuation pattern, 310
Additive manufacturing, 9
Amplifier, 252
Angle axis rotation, 423
    forward problem, 423
    inverse problem, 424
Angle-axis representation, 328
Angular acceleration, 229
Angular velocity, 28, 137, 228
Articulated robot, 12
Assembly mode, 365
Atlas (path planning), 373

Balancing equations, 112
Bang-bang controller, 57
Body
    pseudo-rigid, 333
    rigid, 333
    virtual, 333
Boundary value problem, 360
    shooting method, 361
    split boundary conditions, 360
Breadth first search, 169
    MATLAB function, 169

Cable loop, 394
Cartesian robot, 9
Centre of mass, 201
Centrifugal force, 211
Chart (path planning), 370
Chebychev–Grübler–Kutzbach (CGK) criterion, 309
Closed-loop control, 57
Compensator, 252
Complete planning algorithm, 169
Computed torque control, 278
    multi-input/multi-output (MIMO), 279
    non-linear, 278
    single-input/single-output (SISO), 278
Concentric tube robot, 353
Configuration, 159
Configuration space, 159
Connectivity, 163
Constrained minimisation, 333
    MATLAB function, 333
Constraint
    cable length, 340, 396
    cable loop, 396
    equation, 334
    force, 335
    gear mesh, 346
    inequality, 337
    joint position, 339, 395
    kinematic, 317, 334, 395
    kinetostatic, 336
    pulley, 396
    tendon-spacer, 339
Constraint (control)
    artificial, 298
    force-type, 297
    motion-type, 297
    natural, 297
Constraint analysis, 332, 394
Contact force
    normal, 347
    tangential, 347
Control
    disturbance rejection, 252
    error, 272
    force, 295
    hybrid motion-force, 297
    set point, 56

steady-state error, 262
task space (force), 295
task space (motion), 289
transient response, 262
Control law
decoupling, 291
error, 272
error dynamics, 272
first, 278
first partitioning, 291
inner loop, 280
outer loop, 280
partitioning, 291
second, 278
second partitioning, 291
Controller, 252
bang-bang, 57
closed-loop, 57
derivative, 61
feedback, 57
feedforward, 56
gravity compensation, 282
impedance, 300
integral, 61
linearisation, 278, 303
non-feedback, 56
open-loop, 56
proportional (P), 59, 261
proportional-derivative (PD), 62, 264
proportional-integral (PI), 62
proportional-integral-derivative (PID), 64, 269
pulse width modulation (PWM), 65
Coriolis force, 211
Cosserat rod theory, 354
Cubic splines, 36
Cylindrical robot, 9

da Vinci surgical robot, 387
Decoupling (control), 291
Degrees of freedom, 6, 309
Depth first search, 172
MATLAB function, 172
Device footprint, 388
DH notation, 91, 105
joint angle, 93
link length, 93
link offset, 93
link twist, 93
MATLAB function, 102
parameter, 92
prismatic joint, 97
Dialytic method, 121
Dijkstra's algorithm, 175
MATLAB function, 176
Discretisation
grid, 163
quadtree, 410
random sampling, 166, 372
Disturbance, 252
Disturbance rejection, 252
Dynamics, 201
centrifugal force, 211
Coriolis force, 211
friction term, 218
gravity term, 211, 218
Lagrangian, 207
load term, 219
mass matrix, 217
Newton-Euler, 222

Effective damping factor, 256
Effective inertia, 255
Effort, 144
Eigenvalue, 204
Eigenvector, 204
End effector, 6
Error dynamics, 272
first-order, 273
joint velocity, 275
second-order, 273
task space, 291
Euler angle rotation, 77, 327
Z-Y-X, 77
forward problem, 78
inverse problem, 79
Euler parameters, 426
Euler's equation, *see* Euler's second law
Euler's second law, 222

First search algorithm, 169
Five-bar manipulator, 366
Fixed angle rotation, 75
X-Y-Z, 75
forward problem, 75
inverse problem, 76
Fixed reference tracking, 261
Force control, 295
Forward kinematics, 91
of a 3-RRR, 375
of a 3R, 21, 95

- of a cylindrical robot, 104
- of a five-bar manipulator, 366
- of a PUMA, 100
- of a spherical wrist, 99
- of a UR5, 404
- of an RP manipulator, 23
- of an RPR, 98, 102
- planar, 21

Four-bar linkage, 390
Four-dimensional space, 324
Frame
- base, 17
- measuring, 137
- reference, 17, 137
- world, 17

Function handle, 190

Gear ratio, 256
Gear train, 391
Gear transmission, 255
Geared motor, 256
Geared servo, 256
Generalised coordinates, 317, 333, 395
Generalised external force, 334, 395
Geometric method, 24
Gravity compensation, 282
Grid discretisation, 163
Gripper
- MARS, 402
- underactuated, 332

Hooke constitutive equation, 335, 356
Hybrid motion-force control, 297

Impedance control, 300
- contact dynamics, 302
- impedance, 301
- single-axis, 300
- task-space, 301

IMU, 426
Industry 4.0, 4
Inertia tensor, 202
- eigenvalue, 204
- eigenvector, 204
- parallel-axis theorem, 206
- principal axes, 204
- principal axis, 204
- principal frame, 204
- principal moment, 204
- similarity transformation, 205

Intermediate axis theorem, 225
Inverse kinematics, 110
- analytical method, 112
- geometric method, 24
- of a 2R, 24, 25, 112, 122
- of a 4R, 114
- of a 6R, 121
- of a cylindrical robot, 119
- of a UR5, 404
- of PUMA, 115
- planar, 24

Inward propagation, 231

Jacobian, 29, 132, 317
- MATLAB function, 150
- non-square, 397
- parallel, 365
- serial, 7

Joint angle, 93
Joint space, 6, 20

Kinematic chain, 6
Kinematics
- forward, 91
- inverse, 110
- parallel manipulator, 364
- planar, 20

Kinetic energy, 208
Klein form, 297

Lagrange equation, 207
Lagrange multiplier, 335, 397
Lagrangian, 207
Lagrangian dynamics, 207
- centrifugal force, 211
- Coriolis force, 211
- friction term, 218
- gravity term, 211, 218
- kinetic energy, 208
- load term, 219
- mass matrix, 217
- potential energy, 209

Linear velocity, 28, 136, 227
Linearisation (control), 278, 303
Liner acceleration, 229
Link length, 93
Link offset, 93
Link twist, 93
Linkage
- arc track, 388
- Bennett, 317
- cable-constrained, 394

dual-triangular, 389
four-bar, 390
Kempe focal, 390
parallelogram, 389
Peaucellier-Lipkin straight-line, 392
spatial four-bar, 318
spherical, 389
Links, 6
Local mobility, 317
first-order, 317
second-order, 318
Loop-closure equation, 310
Lyapunov
direct method, 285
second method, 285
Lyapunov function, 285
Lyapunov stability, 283
asymptotically stable, 283
criterion, 283
equilibrium point, 283
null solution, 283
phase portrait, 284
stable, 283
vector field, 283

Mapping
robot-dependent, 356
robot-independent, 354
Mass matrix, 217
Mass moment of inertia, 203
Mass product of inertia, 203
MATLAB function
A* (A star) algorithm, 180
breadth first search, 169
constrained minimisation, 333
depth first search, 172
DH notation, 102
Dijkstra's algorithm, 176
Jacobian, 150
Newton-Euler dynamics, 241
Minimally Invasive Surgery, 353
cannula type, 353
laparoscopic type, 353, 387
robot, 353, 387
tool, 387
Minimum energy theory, 332
Mobility, 309
actuation pattern, 310
global, 316
independent constraint equation, 309
joint variables, 309
local, 317
property of motion, 310
Monash Apple Retrieving System, 402
Motion subgroups, 310
Motor
armature circuit, 252
armature current, 252
armature wire resistance, 252
back-EMF voltage, 252
coil inductance, 252
DC, 252
disturbance torque, 254
EMF constant, 253
torque constant, 253
voltage loop equation, 252

Newton's second law, 201
Newton-Euler dynamics, 222
acceleration propagation, 229
Euler's equation, 222
inward propagation, 231
MATLAB function, 241
outward propagation, 227
prismatic joint, 229
velocity propagation, 229
Non-linear computed torque control, 278
Numerical integration, 355
discretised form, 355

Open-loop control, 56
Orientation, 73
Orientation workspace, 324
principles of invariant, 324
tensor, 325
volume, 324
volume element, 325
Outward propagation, 227

Parabolic blends, 42
Parallel axis theorem, 206
Parallel robot, 6, 309, 364
Parallelogram, 389
Partitioning (control), 291
Path planning, 159
atlas, 373
chart, 370
complete algorithms, 169
completeness, 168
complexity, 168
potential fields, 189

sample-based algorithms, 184
trajectory generation, 36
Plücker coordinates, 143
Planning algorithm
RRT-Connect, 410
Planning algorithms
A* (A star), 179
breadth first search, 169
complete algorithms, 169
depth first search, 172
Dijkstra, 175
first search, 169
potential fields, 189
probabilistic road map (PRM), 184
Rapidly-exploring random tree (RRT), 186
sample-based algorithms, 184
Plant, 252
Polynomial simplification, *see* Dialytic method
Pose, 6, 17
Position and orientation, 16, 71
Position vector, 137, 227
Potential energy, 209, 333
Potential fields, 189
Principal axis, 204
Principal frame, 204
Principal moment, 204
Principle of virtual work, 145, 148
Prismatic joint, 6
Probabilistic road map (PRM), 184
Programming
function handle, 190
ROS, 191
symbolic, 190
Property of motion, 310
Proportional (P) controller, 59, 261
Proportional-derivative (PD) controller, 62, 264
gravity compensation, 282
Proportional-integral (PI) controller, 62
Proportional-integral-derivative (PID) controller, 64, 269
gravity compensation, 282
Prosthesis, 343
Pseudo inverse, 398
Pseudo-rigid body, 333
Pulse width modulation (PWM) controller, 65
PUMA, 3

Quadtree, 410
Quaternion, 324, 425

Random sampling, 166, 372
Rapidly-exploring random tree (RRT), 186
Reference
fixed, 56, 261
time-variant, 272
Remote centre, 387
Remote centre of motion, 387
Revolute joint, 6
Rigid body, 333
Rigid body dynamics, 201
Robot
2-RPS-UPS, 315
2-URRH, 314
3-RRR, 374
3-RRTR, 313
3-UPU, 311
articulated, 12
Cartesian, 9
concentric tube, 353
cylindrical, 9
five-bar manipulator, 366
MARS, 402
PUMA, 3
SCARA, 11
sphere, 11
spherical, 328
tendon-driven, 337
Unimate, 3
UR5, 85, 404
Robot architectures, 5
Robot operating system, *see* ROS
ROS, 191
action service, 195
master, 193
message, 194
MoveIt, 410
node, 193
publisher, 194
service, 194
subscriber, 194
topic, 194
Rotation matrix, 71
RRT-Connect, 410
Runge-Kutta 4, 362

Sample-based path planning, 184
SCARA, 11

Screw theory, 143, 297
Selection matrix, 336, 397
Sensor, 252
Serial robot, 6
Servo, 252
Set point, 56
Shooting method, 361
Similarity transformation, 205
Single-input/single-output (SISO) control system, 252
Singularity, 30, 148
  four-bar linkage, 391
  locus, 370
  parallelogram, 391
  Type I parallel, 365
  Type II parallel, 365
Singularity analysis
  of a 3-RRR, 378
  of a 5R, 366
Singularity locus, 370
Skew-symmetric matrix, 137
Soft robot, 332
  concentric tube, 353
  tendon-driven, 337
Special orthonormal subgroup $SO(3)$, 72, 310
Spherical polar coordinates, 325, 328
Spherical robot, 11, 328
Stability, 252
Static equilibrium, 357
Statics, 31, 144
Steady-state error, 262
Step input, 261
Strand model, 354
Surgical robot, 353, 387
Symbolic function, 190
Synthesise, 387
System
  fully-constrained, 398
  over-constrained, 317, 391
  underactuated, 332

Tangent half-angle substitution, *see* Univariate polynomial
Task space, 6, 20
Task space control
  dynamics, 290
  motion, 289
  single-axis force, 295
Taylor expansion, 317
Tennis racquet theorem, *see* Intermediate axis theorem
Tracking, 252
Trajectory generation, 36
  cubic splines, 36
  parabolic blends, 42
Transcendental equation, solving, 110
Transformation, 80
  inverse matrix, 82
  matrix, 80
  of twist and wrench, 236, 358
  pattern, 310
  planar, 17
  tree, 19
Transient response, 262
Twist, 219, 297
  inverse transformation, 237
  transformation, 236
Two-dimensional space, 17

Unimate, 3
Univariate polynomial, 119, 365
UR5, 85, 404

Variable
  joint space, 133
  task space, 133
Velocity, 26, 136
  angular, 28, 137, 228
  linear, 28, 136, 227
  skew-symmetric matrix, 137
Velocity propagation, 140
  prismatic joint, 143
Virtual body, 333
Virtual work, 145, 148

Weierstrauss substitution, *see* Univariate polynomial
Working mode, 365
Workspace, 32, 148
  boundary, 149
  configuration space, 159
  joint space, 20
  task space, 20
  volume, 9
Wrench, 219, 295
  transformation, 238
Wrist, 14