

Bruno Siciliano
Editor

Robotics Goes MOOC

Knowledge



MOREMEDIA



Springer

Robotics Goes MOOC

Bruno Siciliano

Editor

Robotics Goes MOOC

Knowledge



Springer

Editor

Bruno Siciliano
Department of Electrical Engineering
and Information Technology
University of Naples Federico II
Naples, Italy

ISBN 978-3-319-74094-2 ISBN 978-3-319-74095-9 (eBook)

<https://doi.org/10.1007/978-3-319-74095-9>

© Springer Nature Switzerland AG 2025

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Cover illustration: The discovery of the gravity has been an important milestone for human kind. The apple that fell is an iconic picture that represent the moment humans gain that knowledge. Fabrizio Bosco 2022

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

If disposing of this product, please recycle the paper.

Preface

In effetti l'uomo si dimostra essere cosa divina perché dove la natura finisce di produrre le sue spetie l'uomo quivi comincia colle cose naturali a fare coll'aiutorio d'essa natura infinite spetie. (Leonardo da Vinci)

Technique is the essence of the human being. With the technique, human beings can obtain for themselves what they once asked of the Gods. A push to overcome one's own limits that is projected into the ambition to create artifacts in one's own image and likeness: this is how the automaton was born, a machine that moved by itself. Through the course of centuries, there were mechanical inventions of various types: self-propelled fountains, clocks which moved by the force of water, heat, or a particular mechanism, emulating human beings up to the recent zoomorphic or anthropomorphic creatures. They marked a long history from the Christian and Arab Middle Ages to the Italian Renaissance, with the ingenious designs of Leonardo da Vinci, up to the flourishing in Europe and Asia of creations such as the Jaquet-Droz family of androids and the *karakuri-ningyo* mechanical dolls of the eighteenth century. The playful dimension of the first automatons was rooted in the frontier technical knowledge of the time. In the industrial period, with the prevalence of the concept of machine utility over any other function, the wonders of these automata are to be considered as the parents of modern robots.

The term robot, of Slav origin and synonym of subordinate work, comes from the pages of the Czech writer Karel Čapek in the drama *Rossum's Universal Robots* (R.U.R.) from 1920 to indicate an anthropomorphic machine designed and built with organic material to relieve the fatigue of humans. Twenty years later, in 1940, the image of the robot changed becoming a mechanical artifact with the Russian writer Isaac Asimov, yet at the level of science fiction.

In parallel to the development of robotics as a new science, we observed the coexistence of myths and legends with the anthropomorphization of machines which often interferes with what robotic technologies really are. To get to understand the technical meaning of the term robot, we can refer to the definition of robotics of the 1980s as the science that studies the intelligent connection between perception and action. The action is offered by a mechanical system equipped with a locomotion apparatus to move (wheels, tracks, limbs, crawlers) and a manipulation apparatus

to act on the objects in the surrounding environment (mechanical arms, artificial hands, end-effectors, tools). Perception is entrusted to a sensory system capable of acquiring information on the mechanical system and on the environment (position and speed sensors, cameras, distance and proximity sensors, force and tactile sensors). The intelligent connection is entrusted to a control system that governs motion in relation to what happens in the environment, according to the same principle of feedback that regulates the functions of the human body. It is therefore understood how robots cannot ignore physical reality—not only minds and sensors as in artificial Intelligence (AI), but also mechanical bodies.

In the 1990s, research was stimulated by the need to use robots to address the problem of human safety in hazardous environments, to increase the skills of operators, and to reduce their fatigue or by the desire to develop products for potentially large markets and designed to improve the quality of life. A common denominator of these application scenarios is the challenge of operating in poorly structured contexts, whose geometric or physical characteristics not completely known *a priori* require increased skills and a higher level of autonomy.

At the turn of the new millennium, robotics unveiled a wide spectrum of applications involving different scientific disciplines such as biomechanics, haptic perception, neuroscience, virtual simulation, animation, machine learning, and sensor networks. In perspective, its further development can be declined by themes and visions on the basis of four paradigms: **knowledge, design, interaction, and impact**.

The **Robotics Goes MOOC** project is organized into **four volumes**, each devoted to one of the above four paradigms. The contents go beyond those to be found in a standard textbook in robotics by enlightening the common threads on the advantages of resorting to robotic systems and their inherent potential of innovation. On the other hand, the topics are dealt with a lighter level of deepening on the research challenges, such as in the widely known Springer *Handbook of Robotics*, with the clear intent of an outreach toward new users and new communities of robotics science and technology in disparate applications. In return, the challenges of the new emerging areas are proving an abundant source of stimulation and insights for the field of robotics. It is indeed at the intersection of disciplines that the most striking advances are expected to happen.

The eight chapters of each book contain useful lab cases, applications, and examples. The main novelty of the project over the previous literature is the associated **MOOC course “Robotics & Robots”** available at ► sn.pub/NNHqHS. This unique complement features video lectures by some of the chapters’ authors and supporting slides containing exercises, schemes, and tables summarizing the most important concepts explained in the online videos, with the goal of facilitating the comprehension of the various concepts.

In the following, the four paradigms are explained and contextualized, and the contents of each book are presented.

Knowledge

The robot “concept” was clearly established by those many creative historical realizations, such as those recalled above. Nonetheless, the emergence of the “physical” robot had to await the advent of its underlying technologies of mechanics, controls, computers, electronics, and sensors—in one word, mechatronics—during the course of the twentieth century. As always, new designs motivate new research and discoveries which, in turn, lead to enhanced solutions and thus to novel concepts. This virtuous circle over time produced that **knowledge** and understanding which gave birth to the field of robotics, properly referred to as the science and technology of robots.

To make robots and intelligent machines useful to humans, it is necessary to have a broad and tight intersection between robotics and AI. Sophisticated mathematical models are needed that enable the robot from a physical point of view, as well as intelligent algorithms capable of correlating all the information coming from the use of technologically advanced sensors with the data available from experience. It is expected that the synergy of model-based techniques with data-driven approaches will contribute to increasing the level of autonomy of robots and intelligent machines in the near future.

The first book of the Robotics Goes MOOC project starts with the journey of robotics in the introductory chapter by Khatib, who has pioneered our field of robotics and has ferried it to the third millennium. Sensing is crucial for the development of intelligent and autonomous robots, as covered in the second chapter by Nüchter et al. Model-based control is dealt with in the third chapter by Kröeger et al. along with motion planning, as well as in the fourth chapter by Villani and the fifth chapter by Chaumette to handle force and visual feedback, respectively, when interacting with the environment. Resorting to AI techniques is the focus of the last part of the book, namely, the sixth chapter by Peters et al. on learning, the seventh chapter by Beetz et al. on knowledge representation and reasoning, and the eighth chapter by Burgard et al. on graph-based SLAM.

Design

A robot’s appearance and its way of interacting with humans are of fundamental importance. Until a few years ago there was a clear asymmetry between the typically excellent performance of industrial robots and their ugly and disharmonious bodies, with crude ways and potentially very dangerous movements for the human environment. A modern artifact can be as harmonious and beautiful as a complex biological machine or a work of plastic art, and thus it should be clear how **design** plays a key role in robot technology to become a part of our everyday life and change it essentially in a responsible and beneficial manner. It is designers who shape the interface between humans and machines and, as such, they will contribute to make robots customizable and intuitively useful to inexperienced users according to a plug-and-play mode.

The new concept of robotronics as the mechatronics approach to designing advanced robots is the focus of the first chapter of the second book of the Robotics Goes MOOC project by Asfour et al. The main issues for robot manipulator design

are covered in the subsequent material, namely redundant robots in the second chapter by Maciejewsky et al. and parallel robots in the third chapter by Müller, where widely adopted kinematic solutions are presented. Then, the adoption of flexibility, as opposed to the rigid mechanics paradigm, is discussed in the fourth chapter by Malzahn et al. with reference to elastic robots and in the fifth chapter by Laschi focused on soft robotics. Somewhat speculating on the previous two design solutions comes the sixth chapter by Cutkosky dealing with bioinspired robots. The last part of the book is devoted to robot locomotion, namely, the seventh chapter by Vendittelli on wheeled robots and the eighth chapter by Harada on biped humanoids.

Interaction

With the massive and pervasive diffusion of robotics technology in our society, we are heading toward a new type of AI, which we call “physical AI” at the intersection of robotics with AI, which is the science of robots and intelligent machines performing a physical action to help humans in their jobs of daily lives. Physical assistance to disabled or elderly people; reduction of risks and fatigue at work; improvement of production processes of material goods and their sustainability; safety, efficiency, and reduction of environmental impact in transportation of people and goods; and progress of diagnostic and surgical techniques are all examples of scenarios where the new InterAction Technology (IAT) is indispensable.

The **interaction** between robots and humans must be managed in a safe and reliable manner. The robot becomes an ideal assistant, like the tool used by a surgeon, a craftsman, or a skilled worker. The new generation of robots will co-exist—the cobots—with humans not only in the workplace but, gradually, in homes and communities, providing support in services, entertainment, education, health, manufacturing, and care.

As discussed above, interaction plays a crucial role in the development of modern robotic systems. Grasping, manipulation, and cooperative manipulators are covered in the first part of the third book of the Robotics Goes MOOC project, respectively, in the first chapter by Prattichizzo et al., the second chapter by Kao et al., and the third chapter by Caccavale. Specific interaction issues along with the development of digital and physical interfaces are dealt with in the fourth chapter by Marchal et al. and in the fifth chapter by Croft et al., respectively. Interaction between robot and humans also means that a robot can be worn by a human as presented in the sixth chapter by Vitiello et al. A different type of interaction at a cognitive and planning level is the focus of the seventh chapter by Lima devoted to multi-robot systems and the eighth chapter by Song et al. on networked, cloud, and fog robotics, respectively.

Impact

It is often read in the media that AI and robotics are the primary cause of technology unemployment. AI and machine learning techniques are expected to take over lower-level tasks, while humans can spend more time with higher-level tasks. In perspective, it can be said that jobs requiring boring cognitive tasks or repeatable and dangerous physical tasks will be considerably shredded by automation thanks to the wide adoption of AI and robotics technology to replace humans, while jobs requiring challenging cognitive tasks or unstructured physical tasks will be suitably

re-engineered with the progressive introduction of AI and robotics technology to assist humans.

From the discussion above, it should be clear that in a world populated by humans and robots, issues arise that go beyond engineering and technology due to the **impact** resulting from the use of robots in various application scenarios. The anthropization of robots cannot ignore the resolution of those ethical, legal, social, and economic (ELSE) problems that have so far slowed their spread in our society.

The final book of the Robotics Goes MOOC project enlightens the impact of using robotic technology in the main fields of application, namely, industrial robots as in the first chapter by Bischoff et al., medical robotics as in the second chapter by Dario et al., aerial robots as in the third chapter by Ollero et al., orbital robotics as in the fourth chapter by Lampariello, underwater robots as in the fifth chapter by Antonelli, and rescue robots as in the sixth chapter by Murphy. The last part is devoted to the open dilemma of using and accepting robots in human co-habited environments which is addressed in the seventh chapter on social robotics by Pandey and the very final chapter by Tamburrini on the important issues raised by roboethics.

Some 100 years after the entry of the word robot into our lexicon, the challenge and at the same time the opportunity that the world of research will have to represent is related to future scenarios in which robotics will become an interactive means to help improve conditions of life. In this vision, the robot revolution can help us reaffirm the least artificial feature of our world: our humanity. Will human-friendly robots become commonplace, as so often imagined by science fiction? Only time will tell.

Acknowledgments

I am deeply thankful to the authors for their intellectual contributions and careful cross-reviews of the material contained in these four books, as well as for the video lectures and annexed material they produced for the MOOC course. I am indebted to Francesca Bonadei, Executive Editor at Springer Science+Business Media for MOOCs, and her staff in Milan. I wish to acknowledge the continuous support of Ilaria Merciai, PR and Media Manager at federica.eu, who has been committed to the project with her staff to produce the MOOC course at the University of Naples Federico II. Mario Selvaggio and Daniela Passariello from my team have also helped with the technical and communications contents of the MOOC course, respectively. Finally, a warm note of thanks goes to Fabrizio Bosco, the artist who has created the images on the covers to metaphorically illustrate the four paradigms

of robotics; namely, hand trying to catch apple (knowledge), hand firmly grasping apple (design), hand dexterously manipulating apple (interaction), and hand holding bitten apple (impact).

Naples, Italy
January 2025

Bruno Siciliano

About “Robotics Goes MOOC” Books

Each chapter of the “*Robotics Goes MOOC*” books [1–4] contains citations and cross-references to content in other chapters belonging to the four books. The reader can find the cross-references as listed in the scheme below:

Book title	Chapter title	Cross-references
KNOWLEDGE [1] Siciliano B. (Ed) Robotics Goes MOOC – Knowledge, ISBN 978-33-197-4094-2, Springer, 2025		
KNO1	The Journey of Robotics	
KNO2	Sensing and Estimation	KNO5, KNO8, DES7
KNO3	Control and Motion Planning	KNO4, KNO5, KNO6, DES2, INT5, IMP1, IMP2, IMP3, IMP4, IMP5, IMP6
KNO4	Force Control	KNO2, KNO3, DES4, INT1, INT2, INT3, INT6, IMP2
KNO5	Visual Control	KNO2, KNO3
KNO6	Learning	KNO2, KNO3, KNO7
KNO7	Knowledge Representation and Reasoning	KNO6, INT5, INT7, INT8, IMP7
KNO8	Graph-Based SLAM	KNO2, KNO3
DESIGN [2] Siciliano B. (Ed) Robotics Goes MOOC – Design, ISBN 978-33-197-5822-0, Springer, 2025		
DES1	Robotronics: Robot Mechatronics	KNO2, KNO3, DES5, DES8, INT1, INT2

(continued)

Book title	Chapter title	Cross-references
DES2	Redundant Robots	KNO3, DES3, INT1
DES3	Parallel Robots	KNO3, DES2, INT2, IMP1, IMP2
DES4	Elastic Robots	KNO3, KNO4, KNO6, DES5, DES6, DES8, INT2, INT3, INT5, INT6
DES5	Soft Robotics	KNO6, DES6, INT6, IMP2, IMP5
DES6	Bioinspired Robots	DES5, DES8, INT1, IMP3, IMP5, IMP7
DES7	Wheeled Robots	KNO2, KNO3, KNO8
DES8	Humanoids	DES1, DES2, DES3, DES6
INTERACTION [3] Siciliano B. (Ed.) Robotics Goes MOOC – Interaction, ISBN 978-33-197-7269-1, Springer, 2025		
INT1	Grasping	KNO2, KNO4, DES5, INT2, INT3
INT2	Manipulation	KNO3, KNO4, DES6, INT1, INT5
INT3	Cooperative Manipulators	KNO3, KNO4, INT1, INT2
INT4	Virtual Reality and Haptics	DES1, DES5, INT1, INT5, INT6
INT5	Human–Robot Interaction	DES5, DES8, INT1, INT3, INT4, INT6, IMP6, IMP7, IMP8
INT6	Wearable Robotics	KNO4, DES1, DES4, INT4, INT5, IMP2
INT7	Multi-robot Systems	DES6, INT7, INT8
INT8	Networked, Cloud and Fog Robotics	KNO6, INT5, INT7
IMPACT [4] Siciliano B. (Ed.) Robotics Goes MOOC – Impact, ISBN 978-33-197-7266-0, Springer, 2025		
IMP1	Robotics Applications	KNO2, KNO3, KNO4, KNO5, KNO6, KNO7, KNO8, DES1, DES2, DES3, DES4, INT1, INT2, INT3, INT5, INT8, IMP2, IMP8
IMP2	Medical Robotics	KNO2, KNO3, KNO4, KNO5, KNO6, DES3, DES4, INT1, INT4, INT5, INT6, INT8, IMP7, IMP8

(continued)

Book title	Chapter title	Cross-references
IMP3	Aerial Robots	KNO2, KNO3, DES6, INT2, INT5, IMP5
IMP4	Orbital Robotics	KNO2, KNO3, KNO5, KNO6
IMP5	Underwater Robots	KNO2, KNO3, IMP3
IMP6	Rescue Robots	KNO2, KNO3, KNO8, DES5, DES7, INT5, INT7, INT8, IMP2, IMP3, IMP5, IMP8
IMP7	Social Robotics	KNO2, KNO3, KNO6, KNO7, DES6, DES8, INT1, INT2, INT3, INT4, INT5
IMP8	Roboethics	KNO6, KNO7, DES8, INT5, IMP1, IMP2, IMP5

Naples, Italy

Bruno Siciliano

References

1. B. Siciliano (ed.), Robotics Goes MOOC – Knowledge (Springer, 2025), ISBN 978-33-197-4094-2
2. B. Siciliano (ed.), Robotics Goes MOOC – Design (Springer, 2025), ISBN 978-33-197-5822-0
3. B. Siciliano (ed.), Robotics Goes MOOC – Interaction (Springer, 2025), ISBN 978-33-197-7269-1
4. B. Siciliano (ed.), Robotics Goes MOOC – Impact (Springer, 2025), ISBN 978-33-197-7266-0

About MOOC&BOOK

The *MOOC&BOOK* offers a unique opportunity to access a book complementary to an online course and thereby combine the quality of an academic/scholarly/research essay with the communicative power of an online educational product.

As a matter of fact, in addition to the classic book formats and other texts, we all have started to learn with many different multimedia formats—this applies to students and lecturers as well as to all those who are undergoing professional training. In order to meet this need for learning, we have integrated multimedia formats into our books and also developed an app to access the digital content in print books.

The Springer Nature More Media app (iOS, Android) offers the possibility to play videos and soon also other digital content in addition to the printed book content.

The content published here is linked to a series of MOOCs on Robotics specifically created and hosted by Federica Web Learning. You can access the related content via our app: download the SN More Media app for free, scan the link, and access directly to the online courses on your smartphone or tablet: ► sn.pub/9je5v2.

Contents

The Journey of Robotics	1
Oussama Khatib	
1 The Concept of the Artificial	1
2 The Physical Robot.....	2
3 The Twentieth-Century Robots	5
4 Robotics in the New Millennium	6
5 The Robotics Community.....	9
References	11
Sensing and Estimation	13
Dorit Borrmann and Andreas Nüchter	
1 Introduction.....	13
2 Motion Sensors	14
2.1 Odometry	14
2.2 Inertial Navigation Systems	18
3 Optics and Cameras	19
3.1 Propagation of Light	19
3.2 The Pinhole Camera Model	20
3.3 Thin Lenses.....	21
3.4 Aberrations	22
4 Laser Scanning	25
5 Triangulation	31
6 Incremental Registration	38
6.1 Registration of Images	39
6.2 Registration of 3D Point Cloud Data.....	43
7 State Estimation with Bayes Filters	48
7.1 Basics	48
7.2 Bayes Filter.....	50
7.3 Kalman Filter and Extended Kalman Filter.....	52
8 Cross-References	57
References	57

Control and Motion Planning	61
Lars Berscheid and Torsten Kröger	
1 Introduction.....	61
2 Kinematics and Dynamics	62
2.1 Motion in Time and Space.....	63
2.2 Forward and Inverse Kinematics	64
2.3 Rigid-Body Equations of Motion	65
3 Robot Control	67
3.1 Motor Control	67
3.2 Joint Position Control.....	67
3.3 Computed Torque Control	70
3.4 Task Space Control	71
3.5 Force Control	71
4 Robot Motion Planning	72
4.1 Trajectory Planning	72
4.2 Graph-Based Planning	78
4.3 Sampling-Based Planning	80
4.4 Potential Fields	82
4.5 Optimization Algorithms	84
5 Software and Robot Application	86
6 Further Reading	89
7 Exercises.....	90
8 Cross-References	91
References	91
Force Control	95
Luigi Villani	
1 Introduction.....	95
2 Control of Robot Physical Interaction.....	97
3 Impedance Control	99
3.1 Coupled Stability.....	102
3.2 Basic Impedance Control	105
3.3 Impedance Control with Force Feedback	109
3.4 Admittance Control	111
3.5 Multivariable Systems	113
4 Hybrid Force/Motion Control	117
4.1 Constrained Motion	117
4.2 Hybrid Force/Motion Control with Rigid Environment.....	122
4.3 Hybrid Force/Motion Control with Non-rigid Environment	124
5 Cross-References	130
6 Recommended Readings	130
References	132

Visual Control	135
François Chaumette	
1 Introduction	135
2 Basic Theory	136
2.1 The Features Jacobian	137
3 Modeling Visual Features	139
3.1 Image Point	139
3.2 Other Visual Features	141
3.3 Exercises	143
4 Control	147
4.1 Basics	147
4.2 A Typical Example	149
4.3 General Case	156
4.4 Exercises	157
5 Applications	161
6 Summary and Future Directions	162
References	162
Learning	165
Joe Watson, Julien Urain, Joao Carvalho, Niklas Funk, and Jan Peters	
1 Introduction	165
1.1 What Is Robot Learning?	166
1.2 Structure of the Chapter	168
2 Optimal Control from Robot Forward Models	168
2.1 Markov Decision Processes	169
2.2 The Linear Quadratic Regulator	171
2.3 Nonlinear Approximate Optimal Control	172
3 Supervised Machine Learning: Acquiring Models from Data	174
3.1 Bayesian Linear Regression and Nonparametric Gaussian Processes	175
3.2 Automatic Differentiation for Nonlinear Function Approximation ..	178
4 Model Learning for Control	181
4.1 Model Learning: System Identification Meets Machine Learning ...	181
4.2 Optimal Control for Approximate Dynamics Models	182
5 Policy Representations for Robotics	184
5.1 From Torque Control to Motion Generation	184
5.2 Types of Movement Primitives	185
6 Reinforcement Learning	187
6.1 Value Function Methods	188
6.2 Policy Search Methods	192
6.3 Episodic Policy Search	193
6.4 Step-Based Policy Search	195
6.5 Episodic- or Step-Based Policy Search?	197
7 Imitation Learning	197
7.1 Learning Policies by Imitation	197

7.2	Learning Rewards from Data: Inverse Reinforcement Learning.....	199
8	Outlook	201
9	Cross References	203
	References	203
	Knowledge Representation and Reasoning	209
	Michael Beetz and Daniel Nyga	
1	Introduction	209
2	The Rational Robot Agent Model	211
3	Knowledge-Enabled Agent Architecture	214
3.1	Modelling States	215
3.2	Modelling Actions	219
3.3	Modelling Environment Interaction and Evolution.....	220
4	Logic-Based Knowledge Representation and Reasoning	222
4.1	Logic-Based Knowledge Representation	224
4.2	Logic-Based Action Planning	228
4.3	Inferring a Popcorn Making Plan.....	231
4.4	Complexity of Logical Reasoning.....	234
4.5	Implementations of Planning Systems	235
4.6	Discussing Logic-Based Planning of Popcorn Making	236
5	KR&R Within Perception-Action Loops.....	238
5.1	Linking Representation to Perception and Motion	239
5.2	Reasoning for Object Manipulation.....	244
6	Conclusion.....	247
7	Exercises	248
	References	251
	An Updated Tutorial on Graph-Based SLAM	255
	Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard	
1	Introduction	256
2	Probabilistic Formulation of SLAM.....	258
3	Graph-Based SLAM	261
3.1	Error Minimization via Iterative Local Linearizations	263
3.2	Considerations About the Structure of the Linearized System.....	265
3.3	Least Squares on a Manifold	267
3.4	Dealing with Outliers	270
4	Practical Applications	272
5	Further Reading	275
6	Conclusions	279
	References	281

About the Editor



Bruno Siciliano is professor of Control and Robotics at the University of Naples Federico II, Chair of the Scientific Council of the ICAROS Center, and Director of the PRISMA Lab at the Department of Electrical Engineering and Information Technology. He is also Honorary Professor at the University of Óbuda where he holds the Rudolf Kálmán chair. His research interests include robot manipulation and control, human–robot cooperation, and service robotics. He has co-authored/co-edited 24 books, more than 140 journal papers, and more than 310 conference papers/book chapters; his book *Robotics: Modelling, Planning and Control* is one of the most widely adopted textbooks worldwide and has been translated into Chinese, Greek, and Italian. He has delivered more than 30 keynotes and more than 150 invited lectures and seminars at institutions worldwide, and he has been the recipient of several awards, including the recent IEEE/RAS Pioneer Award in Robotics and Automation (2024). He is a Fellow of IEEE, ASME, IFAC, AAIA, and AIIA. He is Co-Editor of the Springer Tracts in Advanced Robotics (STAR) series and the Springer Proceedings in Advanced Robotics (SPAR) series, and he has served on the Editorial Boards of several journals as well as Chair or Co-Chair for numerous international conferences.

He co-edited the Springer *Handbook of Robotics*, which received the AAP PROSE Award for Excellence in Physical Sciences & Mathematics and was also the winner in the category Engineering & Technology (2009). His group has been granted more than 25 European projects, including an Advanced Grant and a

Synergy Grant from the European Research Council. He has served the IEEE Robotics and Automation Society as President, as Vice-President for Technical Activities and Vice-President for Publications, as a member of the AdCom, and as a Distinguished Lecturer. He has been a Board Director of the European Robotics Association. Professor Siciliano is currently an IFAC Pavel J. Nowacki Distinguished Lecturer, a member of the International Foundation of Robotics Research Board, a member of the Nature Italy Board of Trustees, a member of the Advisory Board of Rovial Space, and chair of the Science and Innovation Committee of Enchanted Tools.

The Journey of Robotics



Oussama Khatib

Abstract Robots today are making significant inroads into many aspects of modern life, from their initial roots in industrial manufacturing moving now into healthcare, transportation, logistics, and exploration of deep space and the oceans. This chapter sketches the fascination of this field and its evolution from ancient to modern times. The journey takes us through numerous milestones: from the first automated mechanical artifact (1400 BC) through the establishment of the robot concept in the 1920s, the realization of the first industrial robots in the 1960s, the definition of robotics science and the birth of an active research community in the 1980s, and the expansion into the challenges of our modern twenty-first-century world.

1 The Concept of the Artificial

Robotics has deep cultural roots. Over the course of centuries, we humans have strived continuously to develop substitutes that would be able to mimic our behavior in the various ways we interact among ourselves and with our environment. Many motivations have inspired this search bearing on philosophical, economic, social, and scientific concepts.

The dream of creating artifacts demonstrating skill and intelligence has been part of our quest from the beginning of recorded history. Strikingly, it is a dream that is—at last—becoming part of our reality. Since the earliest civilizations, one of man's greatest pretensions have been the creating of anthropomorphic partners. The Greek mythology legends of the Titan Prometheus, who molded humankind from clay, or

O. Khatib (✉)

Artificial Intelligence Laboratory, Computer Science Department, Stanford University, Stanford, CA, USA

e-mail: khatib@cs.stanford.edu

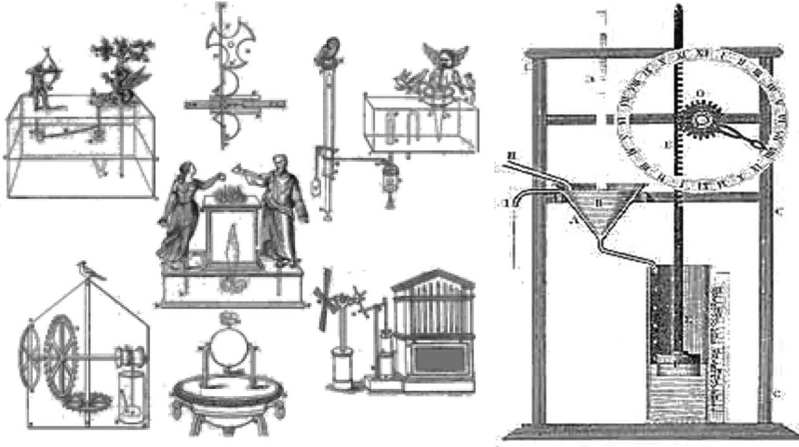


Fig. 1 Egyptian oracle statues and Babylonian clepsydra water clock

that of the giant Talus, the bronze slave forged by Hephaestus (3500 BC), testify to this fascination with moving action and intelligence beyond ourselves.

Egyptian oracle statues with priests secreted inside (2500 BC) were simple early expressions of our modern desire to demonstrate intellectual machines. The search for mechanical assistants gave us the Babylonian clepsydra water clock (Fig. 1), introduced about 1400 BC and one of the first successful mechanisms of automation.

In the following centuries, human creativity gave rise to a host of devices such as the automaton theatre of Hero of Alexandria (100 AD), the hydro-powered water-raising and humanoid machines of Al-Jazari (1200), and Leonardo da Vinci's multitude of ingenious designs (1500) (Fig. 2).

Among the earliest attempts at prosthetics were the artificial limbs created by the French military surgeon Ambroise Paré (1579) [1] to be fitted on soldier amputees (Fig. 3).

Development of automata continued to flourish into the eighteenth century in Europe and Asia with creations such as the Swiss Jacquet-Droz's family of androids (drawer, musician, and writer) and the Japanese Karakuri-Ningyo mechanical dolls (tea server and archer) (Fig. 4).

2 The Physical Robot

The robot "concept" was clearly established by these and other single-purpose historical creations. However, the emergence of functional "physical" robots had to await the arrival of the enabling underlying technologies of the twentieth century.

Literature played its role in bringing the common man into an awareness of the possibilities. The beginning of the twentieth century brought us the term robot—

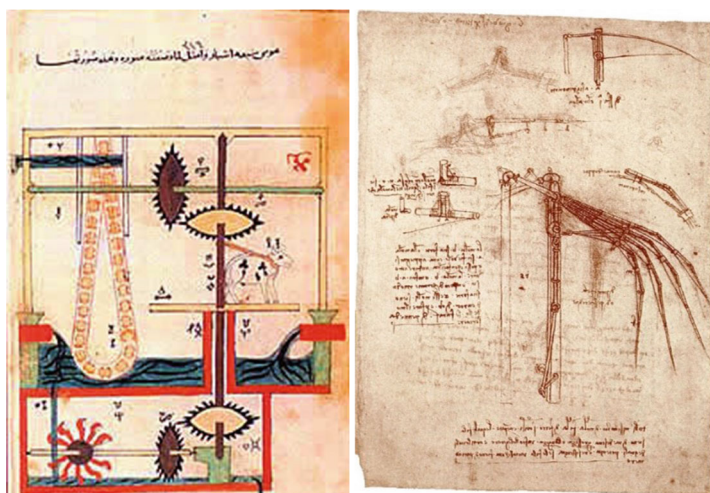


Fig. 2 Al-Jazari's machine and Leonardo da Vinci's designs

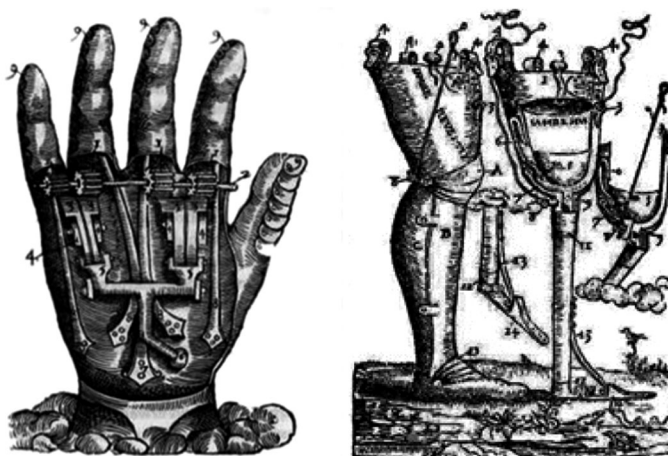


Fig. 3 Paré's artificial limbs designed and built to help those who had been maimed

derived from the Slavic term “robota,” meaning *subordinate labor*—by the Czech playwright Karel Čapek in his play *R.U.R.*, Rossum's Universal Robots (Fig. 5). In 1940, the ethics of the interaction between robots and humans was envisioned to be governed by The Three Laws established by Isaac Asimov,¹ the American

¹ A robot shall not harm a human, or by inaction allow a human to come to harm; a robot shall obey orders given by a human except where such orders would conflict with the First Law; a robot must protect its own existence as long as such protection does not conflict with the First or Second Law.



Fig. 4 Jacquet-Droz's family of androids and Karakuri-Ningyo mechanical dolls

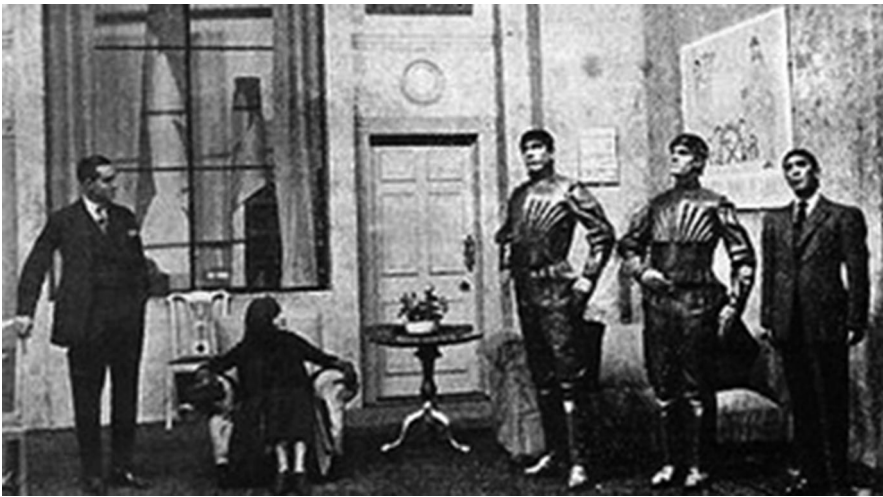


Fig. 5 A scene from Karel Čapek's play R.U.R.

science-fiction writer, in his robot book series. These stories gave us hope for a future where machines could replace manual labor, and provided some reassurance that they would not take on powers beyond human control.

Throughout history, the human body and mind have inspired artists, engineers, and scientists, using media as diverse as cave paintings, sculpture, mechanical toys, photographs, and computer animation. Robots serve as a powerful new medium enabling creation of artifacts that operate within the real world, exhibiting human form and behavior.

3 The Twentieth-Century Robots

The middle of the twentieth century brought the first explorations of the connection between human intelligence and machines, marking the beginning of an era of fertile research in the field of artificial intelligence (AI). Benefitting from advances in the varied technologies of mechanics, controls, computers, and electronics, the first functioning robots came to be realized about this time. As always, new design motivates new research and discovery, which in turn leads to enhanced understanding, new solutions, and the creation of further novel concepts. Over time, this virtuous circle birthed the field of *robotics*, perhaps most appropriately considered as the science and technology of robots.

The early robots of the 1960s grew from the confluence of two technologies: numerical control machines for precise manufacturing, and teleoperators for remote handling of radioactive material. These were robot arms, designed to duplicate joint-for-joint the mechanics of the human arm. They had rudimentary control and little sensing. The mid-to-late twentieth century saw the development of integrated circuits, digital computers, and miniaturized components, enabling the design and programming of computer-controlled robots equipped with the ability to sense their environment and effect change under sophisticated feedback.

These devices, termed industrial robots, became essential components in the automation of flexible manufacturing from the late 1970s. Beyond their initial wide application in the automotive industry, industrial robots were successfully employed in general industry, such as in metal, chemical, electronics, and food production. More recently, robots have found applications outside the factory in areas such as cleaning, search and rescue, underwater and space operations, and medical procedures.

In the 1980s, robotics was defined as development of intelligence in the connection between perception and action. Considering this, the action of a robotic system is given to a locomotion apparatus for movement in the environment (wheels, treads, legs, propellers) possibly coupled with a manipulation apparatus for operating on presented artifacts (arms, end effectors, artificial hands), with suitable actuators and sensors animating and reporting on the evolving state of the system's mechanical components. Perception is the extraction of understanding from sensors that deliver information about the robot's state and that of its surrounding environment—with properties such as position, speed, and location—typically using such measurements as force, touch, and sight. The “intelligence” in this connection is assigned to a programming, planning, and control architecture that builds on perceptual information and models of the robot and its environment, and this is provided or acquired through design or methods involving learning or skill acquisition.

In the 1990s, research was boosted by safety concerns to employ robots as substitutes for human physical presence in endangering environments (field robotics) and by the societal desire to develop commercial products covering diverse markets aimed at improving the quality of life (service robotics). A common denominator of such application scenarios was the requirement of operating in a relatively

unstructured environment, which ultimately demands increased sensing, improved environmental understanding, and a high degree of autonomy. The video “Robots—A 50 Year Journey” (2000)² shows the development of robotics through these first five decades.

4 Robotics in the New Millennium

By the beginning of the new millennium, robotics had undergone a major transformation in both scope and dimension. This expansion was brought about by the increasing maturity of the hardware, software, and algorithms of the field and by advances in its related sensor, actuator, and electronics technologies. Beginning with a largely industrial focus, robotics has been expanding into the challenges of the human world with life-like and people-safe robotics, bringing increasing impact on the economy, production, and culture across our global society.

Unlike robots in manufacturing, this new generation is being conceived to explore, work, and interact with humans in more-complex, unstructured, and human-populated environments including — in homes, workplaces, new production systems, and challenging field domains. The emerging robots are being brought in for increased support in mining, underwater, and related hostile environment operations, and in more open domestic, health, industry, and service applications. Combining the experience and cognitive abilities of the human with the strength, dependability, reach, and endurance of robots will fuel a wide range of new robotic applications extending throughout all aspects of our lives.

Cross-disciplinary, the field of Robotics (Fig. 6) is benefitting from a remarkable convergence of supporting technologies. Transformative advances have been made in robot design from innovations across actuators and sensors, mechanisms and mechatronics, material and fabrication methods, and communications and computation. Decades of research and development have brought the field maturity in its core concepts and methodologies including sensing and perception, manipulation and navigation, skill and strategy acquisition, and planning and reasoning. These advances have been brought together through more-recently developed advanced control architectures and the accelerating impact of machine learning methodologies.

Deploying robots into unstructured environments brings special challenges — in design, sensing, perception, planning, and control. The conventional approach with industrial robots — preprogramming — has proved markedly insufficient for unstructured environments where dynamic changes and incomplete and evolving knowledge make action perception ineffective. Instead, a sensor-based reactive approach employing robust and responsive autonomous *skills* and organized within

² <https://vimeo.com/137042620>

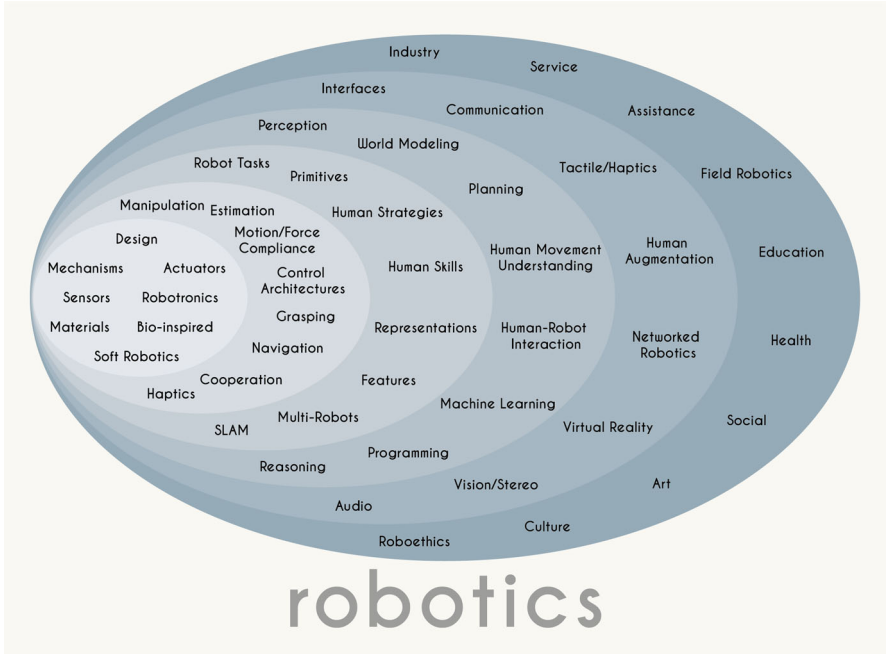


Fig. 6 Robotics cross-disciplinary nature

a *hierarchical architecture* (Fig. 7) integrating human-level cognitive and problem-solving abilities at multiple levels of abstraction to bridge the connections between perception and action. This approach is based on timely understanding of the system’s physical environment and means for accommodating to uncertainties impacting its required dynamic behavior. Effective exercise of these skills requires sensing, perception, and tactile and force feedback, the latter depending on compliant torque-controlled actuation. With this compliance, robot arms, mobile manipulators, and humanoid robots can be sufficiently responsive to interact safely with each other and with partnering humans.

As robots gain in mobility, versatility, and autonomy, they are being tasked with taking on ever more complex challenges in widely varying environments, each requiring negotiating and overcoming a plethora of physical, intellectual, and social obstacles. Much of this development is being driven by understanding the skills and strategies that living systems employ. This not only inspires robot design and control from nano to macro scales, but accelerates multiple issues including our understanding of life, the potential for robots to interact with humans in augmenting our capabilities, and potential means for affecting this that can lead to their becoming fluid co-participants as we employ them among us in our home, community, and workplace environments.

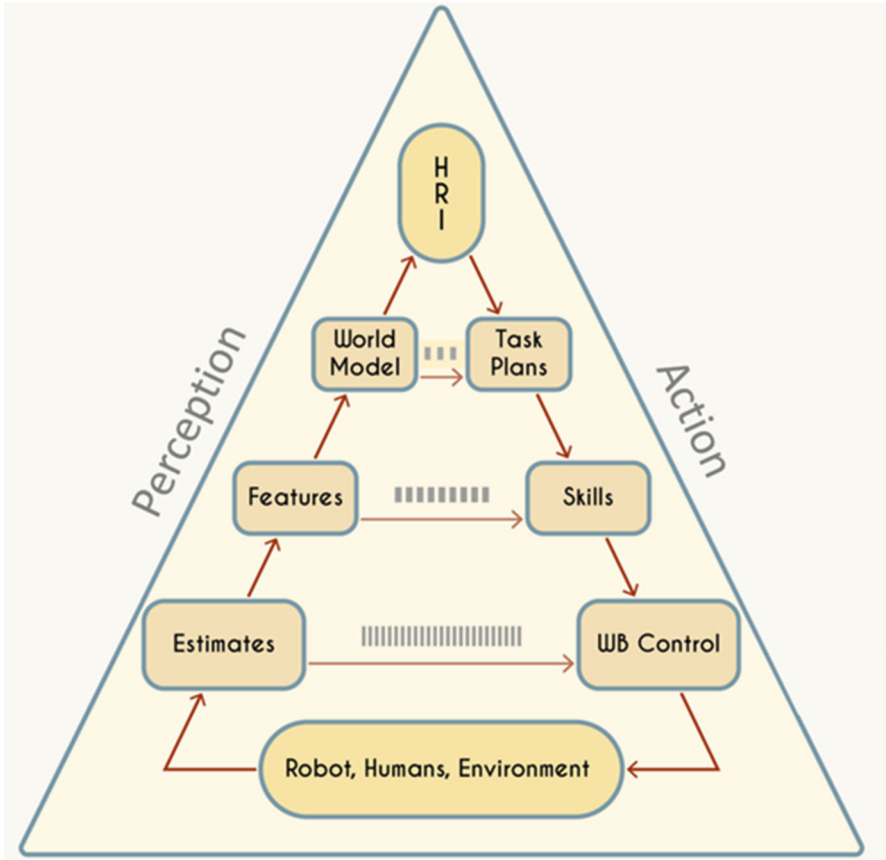


Fig. 7 Robotics multi-layered connection between perception and action

Beyond its impact on physical control, the body of knowledge robotics has produced is revealing a much wider range of applications reaching across diverse research areas and scientific disciplines such as biomechanics, haptics, neurosciences, virtual simulation, animation, surgery, and sensor networks. In return, the challenges of these new emerging areas are proving an abundant source of stimulation and insight for the field of robotics. Indeed, it is at the intersection of disciplines that we expect to see the most striking of advances.

The video “Robots — The Journey Continues”³ (2016) shows the rapid evolution of robotics through the first decade and a half of the new millennium.

³ <https://vimeo.com/173394878>

5 The Robotics Community

The dissemination of research results and findings in archival publications and conference presentations has played a significant role in the advancement of robotics over the past decades. The extent of scientific activities in robotics has led to the establishment of professional societies and research networks devoted to the field, and these have ultimately contributed to the building of an international robotics community.

In the early 1980s, robotics was entering a new era in its development following the success of early industrial applications. With much energy and excitement, the pioneers of robotics launched initiatives which came to shape our field. The first robotics textbook “Robot Manipulators: Mathematics, Programming, and Control” appeared in 1981. A year later, the first journal entirely devoted to robotics — the International Journal of Robotics Research (IJRR) — was established. This was followed by the organization in 1983 of the first International Symposium of Robotics Research (ISRR) with the intent of casting robotics as a distinct scientific field and creating a community around it. Furthering pursuit of these aims, the International Foundation of Robotics Research (IFRR) was founded in 1986 to promote the development of robotics as a science, establishing the theoretical foundations and technology basis for its expanding applications, with emphasis on its potential role in bringing benefit to humans. The mid 80s saw a host of other major developments: establishment of the IEEE Robotics and Automation Council (RAC), organizing of the first IEEE Conference on Robotics (which became the IEEE International Conference on Robotics and Automation—ICRA), launching of the IEEE Journal of Robotics and Automation and the RAC Newsletter. The end of the decade brought transformations: the RAC into the IEEE Robotics and Automation Society (RAS); the IEEE Journal of Robotics and Automation into IEEE Transactions on Robotics and Automation (subsequently splitting into the Transactions on Robotics and Transactions on Automation Science and Engineering); the RAS Newsletter into the IEEE Robotics and Automation Magazine (1994), and the launching of the IEEE Robotics and Automation Letters (RA-L) in 2015.

Over the years, numerous other activities have been initiated. New conferences such as IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1983), International Conference on Advanced Robotics (ICAR 1983), International Symposium on Experimental Robotics (ISER 1989), Robotics: Science and Systems (RSS 2005), as well as new journals such as Robotica (1983), Journal of Robotic Systems (1984, subsequently Journal of Field Robotics 2006), Robotics and Autonomous Systems (1985), Advanced Robotics (1986), Journal of Intelligent and Robotic Systems (1988), Autonomous Robots (1994), have all contributed to furthering the substantial growth of the robotics community.

The 1990s saw an important novelty enter the ICRA flagship conference series with the creation of video proceedings, attaining an important role for rapid

dissemination of research across the community. Over the subsequent years these media have become an integral part of archival publications.

Equally, the international growth of academic robotics graduate programs over these decades shows the field achieving scientific maturity, with subsequent texts establishing the body of rigor needed for mastering the technical foundations: “Introduction to Robotics: Mechanics and Control” [2], “Robot Analysis and Control” [3], “Robotics: Control, Sensing, Vision, and Intelligence” [4], “Robot Dynamics and Control” [5], “Foundation of Robotics” [6], “A Mathematical Introduction to Robotic Manipulation” [7], and “Modelling and Control of Robot Manipulators” [8].

Trans-national engagement with students is essential for the future of the field. Through a joint initiative between IEEE RAS and IFRR, the School of Robotics Science was founded in 2004, with summer schools rapidly expanding in scope and frequency providing students and young researchers an opportunity for training and connections at the highest scientific level. This has served to unite efforts across the globe and facilitated the forming of personal and professional relationships that are the basis for community.

Both IEEE RAS and IFRR promote research, education, dissemination, and industrial realizations through their forums and through collaborations with other organizations worldwide, including European Robotics Research Network (EURON which later merged into euRobotics, the European Association of Robotics), the International Federation of Robotics (IFR), and the Robotics Society of Japan (RSJ).

A seminal contribution to learning and communication in the field came in 2008 with the publication of the Springer Handbook of Robotics, followed in 2016 by its second edition.⁴ This unique award-winning reference for robotics collects the most significant achievements of the international community and covers the foundations of the field, its various growing research areas, and emerging applications. Topics include the fundamental principles and methods for modeling, design, and control of robotic systems, with a chapter devoted to each of kinematics, dynamics, mechanisms and actuation, sensing and estimation, model identification, motion planning, motion control, force control, redundant robots, robots with flexible elements, robotic systems architectures and programming, behavior-based systems, AI reasoning methods for robotics, and robot learning.

The second edition is an up-to-date authoritative reference for robotics researchers, newcomers to the field, and scholars from related disciplines with contents structured to achieve four main objectives: enlargement of foundational topics for robotics, enlightenment in the design of various types of robotic systems, extension of the treatment on robots moving in the environment, and enrichment of advanced robotics applications. Further to an extensive update, additional chapters introduce emerging topics, and links are presented to videos showcasing a comprehensive collection of multimedia reference material.

⁴ B. Siciliano, O. Khatib, “Robotics and the Handbook”, Springer Handbook of Robotics, 2nd Edition, Springer, 2016, <http://www.springer.com/us/book/9783319325507>

As our field expands, embracing the challenges of operating in the modern human world, it is opening into an ever-widening range of applications engaging a broadening diversity of scientific disciplines including biomechanics, neuroscience, haptics, animation, surgery, and sensor networks. New communities of researchers and developers are forming, with growing connections to the core of robotics research. A strategic goal for our community is one of outreach and scientific cooperation across regions and organizations. Our combined efforts to reach this goal carries the promise of exciting advances toward new frontiers.

We are coming to realize the dreams of our forebearers and are developing intelligent machines that promise to reshape humans' relationship with their world. The responsibilities are huge, the potential benefits enormous, and the transformational impact immeasurable. We welcome these new frontiers of human progress.

References

1. P. Hernigou, Ambroise Paré IV: The early history of artificial limbs (from robotic to prostheses). *Int. Orthop.* **37**(6), 1195–1197 (2013). <https://doi.org/10.1007/s00264-013-1884-7>
2. J.J. Craig, *Introduction to Robotics Mechanics and Control*, 1st edn. (Addison-Wesley Pub. Co., Reading, MA, 1985)
3. H. Asada, J.-J.E. Slotine, *Robot Analysis and Control* (Wiley, New York, 1986)
4. K.S. Fu, R.C. Gonzalez, C.S.G. Lee, S.D. Bencomo, *Robotics: Control, Sensing, Vision and Intelligence* (McGraw-Hill, Madrid, 1987)
5. M.W. Spong, M. Vidyasagar, *Robot Dynamics and Control*, 1st edn. (Wiley, New York, 1989)
6. T. Yoshikawa, *Foundations of Robotics Analysis and Control* (The MIT Press, Cambridge, MA, 1990)
7. R.M. Murray, S. Zexiang Li, S. Sastry, *A Mathematical Introduction to Robotic Manipulation* (CRC Press, Boca Raton, 1994)
8. L. Sciavicco, B. Siciliano, *Modeling and Control of Robot Manipulators*, 1st edn. (McGraw-Hill, New York, 1995)

Sensing and Estimation



Dorit Borrmann and Andreas Nüchter

Abstract “Nothing is in the intellect that was not first in the senses.”—A famous saying which is attributed to John Locke applies to the knowledge of robots as well. Most information processed by robots is perceived by sensors. All sensor measurements are affected by noise and may be faulty. A crucial challenge is therefore how to perceive the environment and how to handle sensor data.

This chapter discusses important sensors for mobile and industrial robots: motion sensors, cameras, structured-light and time-of-flight sensors, laser range finders, and stereo-cameras. Associated algorithms are needed to process such sensor data. Most of the algorithms use a probabilistic model and/or optimization with a specific target function.

1 Introduction

Due to the many application areas the types of sensors used on robots are manifold. However, two objectives hold true for any intelligent robot, the need to perceive the environment and its own pose (position and orientation) within the environment. Thus, this chapter focusses on presenting sensors for these purposes. First, motion sensors are introduced (Sect. 2), followed by cameras (Sect. 3) and sensors for depth perception (Sects. 4, 5). Afterwards, methods to join sensor data are presented. On the one hand, joining sensor data from different positions yields a full map of the environment (Sect. 6), on the other hand, fusing data from different sensors decreases uncertainty (Sect. 7).

D. Borrmann · A. Nüchter (✉)

Julius Maximilian University of Würzburg, Robotics and Telematics, Würzburg, Germany

e-mail: andreas@nuechter.de

2 Motion Sensors

A crucial task for a robot is to determine its movement within the environment. One way of doing so is to observe its own actuators and to estimate how their movement will affect the motion of the entire robot. As wheels make up a large percentage of mobile robot locomotion, Sect. 2.1 explains how to estimate the robot movement from wheel rotations. Another way to determine the robot pose makes use of acceleration and velocity sensors as explained in Sect. 2.2.

2.1 Odometry

Calculating the trajectory of a moving object by continuously determining the orientation, velocity and time is called *dead reckoning*. If the first two parameters are calculated from the motion of the wheels, one talks about *odometry*. The underlying measurement of the wheel turns is performed by analysing the rotation angles using wheel encoders. Such an approach is a kinematic one. Using only kinematics is a simplification as one does not consider related physical effects such as friction, acceleration of masses, moments and other forces. In general forward kinematics refers to the use of the kinematic equations of a robot to compute the pose of the robot itself or its end effector based on internal parameters. The following introduces first the differential drive, which is commonly used in robotics, followed by a more general model. For more details on wheeled robot motion including kinematic constraints, please refer to Chap. DES7.

2.1.1 Differential Drive

Consider a system with two active wheels that are connected by an axis but can rotate at different speed. Let θ be the orientation of the system in radian ($1 \text{ rad} = 180/\pi$), where $\theta = 0 \text{ rad}$ specifies the x -direction, i.e., the initial orientation or viewing direction of the system, y is towards the left. Let b be the length of the mechanically powered axis and v_r, v_l the constant velocities of the right and left wheel (cf. Fig. 1). Skid-steered systems or robots with tracks can be treated like differential drive systems with small modifications. The assumption of constant velocities is not a limitation: if the measuring frequency is sufficiently high, the function can be regarded as piecewise constant. Furthermore, let Δt be the time difference between two measurements, which is also assumed to be constant. P_n corresponds to the pose at the time point t_n , with $t_n = t_{n-1} + \Delta t$ and $t_0 = 0 \text{ sec}$.

First we consider the orientation of the mobile system and we assume the robot as rigid. The orientation is usually defined with respect to a reference point on the

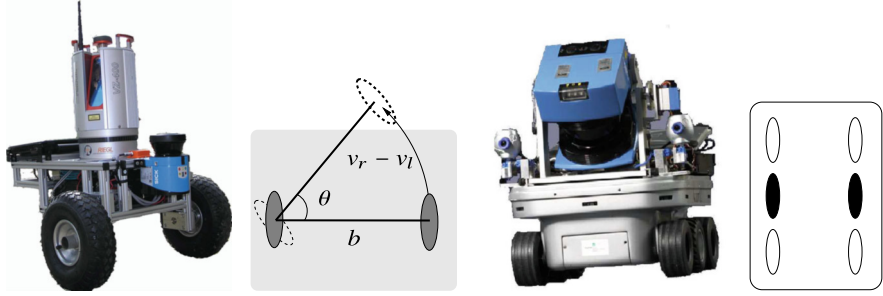


Fig. 1 Two mobile systems. Left: the Irma3D (intelligent robot for mapping applications in 3D) system with two motorized wheels and a passive castor. Change of the pose when turning. Right: the skid-steered robot Kurt3D which can also be treated as a differential drive system by assuming the front and rear wheels as passive. Scheme of the differential drive of such a skid-steered system

system, commonly the center point between the two motorized wheels. The change of the orientation over time is given by:

$$\frac{d\theta}{dt} = \frac{v_r - v_l}{b} . \quad (1)$$

This becomes immediately clear from Fig. 1, by considering not the velocities v_l , v_r , but the distance that the right wheel has driven. The length of the circular arc, in the figure denoted as $v_r - v_l$, is calculated as the product of the distance b and the angle θ in rad. Integration of Eq. (1) using a starting value θ_0 yields the calculation of the orientation θ_t as a function of time t :

$$\theta_t = \theta_0 + \frac{(v_r - v_l)t}{b} . \quad (2)$$

If the reference point is the center of the system, the overall velocity is just the average of the velocity of the left and right wheel and with Eq. (2) follows:

$$\frac{dx}{dt} = \left[\frac{v_l + v_r}{2} \right] \cos \theta_t , \quad \frac{dy}{dt} = \left[\frac{v_l + v_r}{2} \right] \sin \theta_t . \quad (3)$$

Similarly to Eq. (2) we yield the analytical values for x_t , y_t by integration:

$$x_t = \frac{v_l + v_r}{2} \int \cos \theta_t dt , \quad y_t = \frac{v_l + v_r}{2} \int \sin \theta_t dt . \quad (4)$$

The integration creates the term $1/v_l - v_r$. Thus, straight movement, i.e., $v_r = v_l$, must be treated separately. Furthermore, the division becomes numerically instable for $v_r \approx v_l$. To cope with these problems, one performs, instead of the correct integration of Eq. (4), the following approximation: we assume constant velocities

within a measuring interval as usually cycle times are in the order of 100 Hz, i.e., Δt is in the order of $1/100$ s. Using $\Delta\theta = (v_l - v_r)/b$ and $v = (v_l + v_r)/2$ the pose P_n of the system at time t_n is approximated by:

$$P_n \equiv \begin{pmatrix} x_n \\ y_n \\ \theta_n \end{pmatrix} = \begin{pmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{pmatrix} + \begin{pmatrix} v \cdot \Delta t \cdot \cos(\theta_{n-1} + \frac{1}{2}\Delta\theta\Delta t) \\ v \cdot \Delta t \cdot \sin(\theta_{n-1} + \frac{1}{2}\Delta\theta\Delta t) \\ \Delta\theta \cdot \Delta t \end{pmatrix}. \quad (5)$$

Using a fixed time difference between two measurements given by the frequency of the velocity measurements it is possible to compute a trajectory of the system $\langle P_1, P_2, P_3, \dots \rangle$. Faster measurements, i.e., smaller Δt , yield a more precise calculation.

2.1.2 Steering Gear

A more general kinematic model considers one or two steering axes. This describes a system with lateral motion, i.e., a robot that can set the steer angle of the front axis independently from the rear axis (cf. Fig. 2, left). The Ackermann steering, which is the well-known car-like steering with rear drive, is a special case of the lateral motion. The computation of the kinematics uses a reduced model, where the front and the rear wheels are combined and a virtual middle axis is created (cf. Fig. 2, right). Here θ denotes the orientation relative to the x -axis, β the slip angle, i.e., the angle between the yaw axis and the actual motion direction. Furthermore, let δ_r the angularity of the rear wheel, δ_f of the front wheel, $L = l_f + l_r$ the distance of the axes, where l_r is the distance of the rear wheel (A) to the center C, and l_f correspondingly for the front wheel (B). The instantaneous center of curvature (ICC) or instantaneous center of rotation (ICR) O is defined using the intersection of the center of the wheels from orthogonal to the yaw axis of the wheels.

The following model assumes that the vectors of the wheels motion are oriented along the orientation of the wheels (δ_f, δ_r , in relation to the yaw axis of the robot).

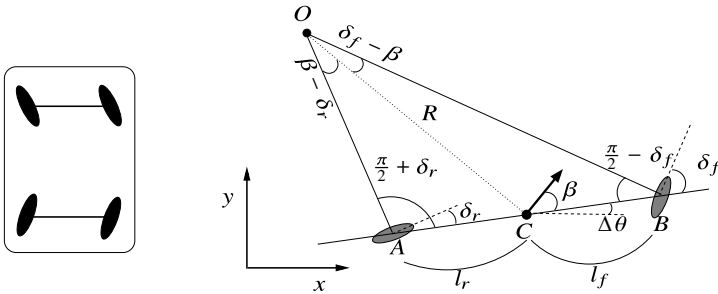


Fig. 2 Steering kinematics. Left: general model. Right: reduction to a virtual middle axis

This assumption is not true for systems driving at high velocities. In Fig. 2 (right) one sees that for the triangles OCA respectively OCB the following conditions hold:

$$\frac{\sin(\delta_f - \beta)}{l_f} = \frac{\sin\left(\frac{\pi}{2} - \delta_f\right)}{R}, \quad \frac{\sin(\beta - \delta_r)}{l_r} = \frac{\sin\left(\frac{\pi}{2} + \delta_r\right)}{R}, \quad (6)$$

which can be rewritten as:

$$(\tan \delta_f - \tan \delta_r) \cos \beta = \frac{l_f + l_r}{R}. \quad (7)$$

Under the assumption of small velocities for the change in orientation per time unit $\Delta\theta = v/R$ holds. We obtain:

$$\Delta\theta = \frac{v \cos \beta}{l_f + l_r} (\tan \delta_f - \tan \delta_r) \quad (8)$$

with the slip angle β and the overall velocity v of the system in the reference point C , calculated using the velocities v_f, v_r of the front and rear wheel:

$$\beta = \arctan\left(\frac{l_f \tan \delta_r + l_r \tan \delta_f}{l_f + l_r}\right), \quad (9)$$

$$v = \frac{v_f \cos \delta_f + v_r \cos \delta_r}{2 \cos \beta}. \quad (10)$$

We obtain now a model of the system with lateral motion:

$$P_n = \begin{pmatrix} x_n \\ y_n \\ \theta_n \end{pmatrix} = \begin{pmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{pmatrix} + \begin{pmatrix} v \cdot \Delta t \cdot \cos(\theta_{n-1} + \beta + \Delta\theta \Delta t) \\ v \cdot \Delta t \cdot \sin(\theta_{n-1} + \beta + \Delta\theta \Delta t) \\ \Delta\theta \cdot \Delta t \end{pmatrix}. \quad (11)$$

Since the Ackermann steering (cf. Fig. 3) is a special case of the above model, Eq. (11) remains valid with $\delta_r = 0$. Rotations on the spot are obviously possible using the general lateral kinematics by setting the wheels the opposite $\pm 90^\circ$. For the Ackermann steering δ_r is fixed and therefore this motion is not possible.

A further aspect is illustrated in Fig. 3: The defined reduction to one middle axis is just an approximation for four wheel vehicles, since the radii of the trajectories of the two wheels differ. Let δ_i the angle of the inner and δ_o the angle of the outer wheel and b the distance. If the radius R of the system is large compared to the distance of the axes L ($R = \overline{CO} \gg L$), then the angles can be estimated as:

$$\delta_o = \frac{L}{R + \frac{b}{2}}, \quad \delta_i = \frac{L}{R - \frac{b}{2}}. \quad (12)$$

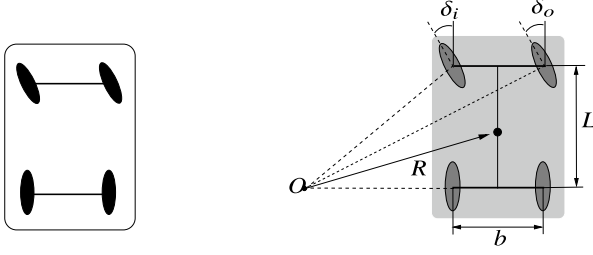


Fig. 3 Left: Ackermann steering. Right: relevant quantities for calculating the odometry

The averaged steering angle is approximated by:

$$\delta = \frac{\delta_o + \delta_i}{2} \approx \frac{L}{R} \quad (13)$$

and therefore, we have for the difference of the inner and outer wheel orientation:

$$\delta_i - \delta_o = \frac{L}{R^2} b = \delta^2 \frac{b}{L} . \quad (14)$$

Thus, the model Eq. (11) can be applied to both sides in a separate fashion to reduce approximation errors.

2.2 Inertial Navigation Systems

An alternative solution to estimating the self-motion of a system by odometry is by use of an inertial measurement unit (IMU), i.e., a combination of several inertial sensors such as accelerometers and gyroscopes. An IMU is the measurement component of an Inertial Navigation System (INS), which is part of many robotic systems. The IMU outputs raw data of gyroscopes, accelerometers, and magnetometers. The Attitude and Heading Reference System (AHRS) embeds an Extended Kalman Filter (EKF) to deliver orientation data, and the Inertial Navigation System (INS) additionally embeds a GNSS receiver for a filtered position in real-time.

For estimating all 6 kinematic degrees of freedom, namely rotation $\mathbf{R}$ and translation $\mathbf{t}$, an IMU consists of three perpendicular accelerometers for measuring the translational motion in x -, y - and z -direction and three perpendicular gyroscopes to measure the rotational motions around the x -, y - and z -axis. An IMU yields three linear acceleration values for the translational motion and three rotational velocities as rotational rates. An INS uses the IMU measurements. After compensating the linear acceleration caused by gravity, integration yields the linear velocity. A second integration step determines the position in space with respect to a reference point. The integration of the rotational rates yield the orientation of the system.

For estimating the integration constants, for improving the accuracy and correcting for drift of the sensors, INSs integrate additionally magnetometers, barometers, and Global Navigation Satellite Systems (GNSS). For fusing measurements and estimation of the output values, the Kalman filter is often used.

3 Optics and Cameras

The previously described methods are used to determine the pose of the robot itself, mostly independent of the surrounding environment. To be able to interact with the environment, however, it is important to perceive what is around the robot. To quickly observe their surroundings humans often use the sense of sight. The technological equivalent, a camera, is therefore commonly used as a robotic sensor, for tele-operation as well as for autonomous tasks. For details on controlling a robot using visual sensors, please refer to Chap. [KNO5](#)). Cameras are optical systems that rely on the propagation of light. To understand the benefits and limitations of such a system it is essential to understand the underlying physical concepts. Thus, this section gives a brief introduction into the propagation of light, followed by the description of camera models.

3.1 Propagation of Light

Most luminous sources emit light due to their high temperature. The heat corresponds to the atoms being in a state of agitation. The effects of these disturbances are transmitted into all directions as vibrations or waves that radiate outwards at high speed. At low temperatures these disturbances can be felt by the sense of touch in the nearer surroundings. Increasing temperatures lead to faster waves that are visible to the human eye. As an example, a cold dark metal emits no light. When heated up, first the temperature is detectable before it glows red, yellow and later white. The exact nature of the wave motions is not known but as a model commonly compared to the ripples occurring when dropping a stone into water. The characteristics of these waves are the velocity v at which they travel outwards, the distance between the wave crests, known as wavelength λ and the frequency f , i.e., the rate of rise and fall. These are related by:

$$v = \lambda f . \quad (15)$$

All optical waves in a vacuum share the common velocity $c = 299.792458 \cdot 10^6$ m/s, the speed of light, and vary only in frequency and wavelength. The example with the metal leads to the assumption that heat has a lower frequency than red light which has again a lower frequency than yellow and white light. From Eq. (15) follows that the wavelength is inversely proportional to frequency. Based on these characteristics

the waves are classified into categories. The exact subdivision is not consistent in literature. The visible light covers only a very small part of the spectrum. It should be noted that what is perceived as white light is actually constituted of all colors equally. This was discovered by Newton. When focusing a narrow beam on a glass prism the white light is dispersed into its constituent colors, all the colors of the rainbow.

The wave motion is described by Huygen's principle. In a homogeneous isotropic optical media light spreads uniformly at the same speed in all direction. Assuming a point light source the light spreads in the form of a sphere at all times, the so called wave front. Due to the uniform spherical extension of the wave fronts each point of the wave travels in a straight line, the ray. Considering each point on the surface of the wave front as a new light source the light spreads again spherically in so called wavelets until the next wave front. The concept of wavelets explains the change of direction occurring when light enters a different medium or is reflected at a surface. Neither rays nor wavelets actually exist but they suffice as model to explain the behavior of light.

When light from a point source passes through a small opening, in optics, especially photography, commonly called aperture, the small bundle of rays that pass through are diverging. There are no sharp edges, instead the waves bend slightly around the edges into the space behind the obstacle causing so called diffractions. If light from a larger light source passes through the small opening, each point from the light source causes a small bundle of rays which form together a beam. Light that enters a new medium usually changes its direction. This process is called refraction. The refractive index that describes the amount of change is based on the change of speed at which the light travels through the old and the new medium.

3.2 The Pinhole Camera Model

Cameras are based on the idea to capture the light that passes through a small opening, from now on called aperture. A pinhole camera supports the aforementioned model where light travels in straight lines. Given a small opening, as depicted in Fig. 4 (left), each point of an illuminated object produces a bundle of rays that passes through the aperture and leaves a small patch of light on the image plane,

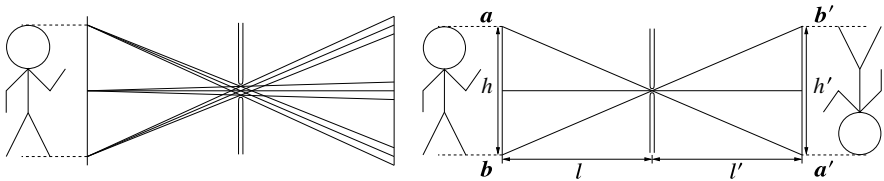


Fig. 4 Pinhole camera model. Left: extended hole. Right: pinhole. Reproduced based on [14]

a plane parallel to the plane of the opening. Each patch has the shape of the aperture resulting in a blurry image of the object that is inverted because of the straight lines. With a very small aperture (Fig. 4, right), a pinhole, the overlap between the patches becomes so small that a fairly sharp image appears. The remaining blur is caused mainly by diffraction.

The mathematics of similar triangles in Fig. 4 (right) show that the size of the projection depends on the distances of both the object l and the image plane l' to the center of projection:

$$\frac{h'}{h} = \frac{l'}{l} . \quad (16)$$

The pinhole camera has no distortions and a long depth of focus, i.e., at greatly varying distances the sharpness remains the same. However, the small hole causes very low illumination as opposed to the larger aperture of lenses that allow more light to pass through.

3.3 Thin Lenses

Figure 5 explains the geometry of image formation using a positive thin lens. The aperture is assumed to be limited to the paraxial region, i.e., the rays that form small angles with and have a small distance to the optical axis (paraxial assumption). The optical axis is the symmetry axis in an optical system. Angles and heights in all figures are exaggerated to point out the effects. Given an object located at a distance of l from the lens, three points, the optical center o of the lens and the two

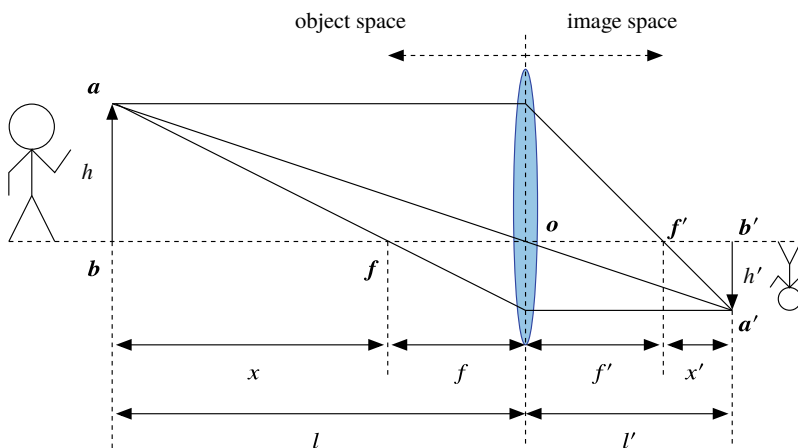


Fig. 5 Thin lens model. Reproduced based on [14]

principal foci f and f' and three rays emerging from point a and passing through these points are used to calculate the image distance l' . A ray parallel to the optical axis is refracted to cross the optical axis at the second principal focus. A ray passing through the first principal focus is refracted parallel to the optical axis. At the optical center the two surfaces of the lens are parallel and perpendicular to the optical axis. For a ray crossing the lens at this point, the lens appears as two parallel planes, thus due to the thinness assumption the ray passes without deviation. The intersection of these three rays gives the distance of the image plane l' . Two rays are sufficient to determine the distance, the third ray is only used for verification. With an ideal lens, all rays coming from point a intersect at point a' and form a sharp image and all points from point b intersect at point b' and form a sharp image. As for the pinhole camera model, the image is again mirrored. Using similar triangles the relations:

$$m = \frac{h'}{h} = \frac{l'}{l} = \frac{x'}{f'} = \frac{f}{x} \quad (17)$$

are derived, where m is the magnification. For a negative lens, divergence is added to the light so that a virtual image of the object is formed. The two principal foci switch sides, i.e., f' is closer to the object than f . A virtual image appears to be in the object space. However, the image space is still to the right of the lens and the rays only exist there.

The thin lens equations are approximations. Actual camera systems use lenses where the thickness d cannot be ignored. However, with a few considerations the equations developed for the thin lens assumption can still be used for thick lenses or even systems of lenses. Refractions at several distant lens surfaces can be simplified to a single refraction at an imagined virtual plane, the principal plane, resulting in the equivalent focal length f_E . The degree to which a lens converges or diverges light is quantified by the focal power. Converging lenses have positive focal power while diverging lenses have negative power.

3.4 Aberrations

All equations derived under the paraxial assumption are only approximations of the true underlying mathematics of optics. All derivations from these ideal conditions are called aberrations, that fall under two main categories [18]. Chromatic aberrations appear due to the fact that the refractive index, is indeed a function of frequency or color, while monochromatic aberrations also occur with quasi-monochromatic light. Under the latter fall the five primary aberrations that can be grouped in two subgroups, aberrations that blur the image, such as spherical aberrations, coma and astigmatism as well as aberrations that deform the image, such as Petzval field curvature and distortions. Many of the aberrations caused by using spherical surfaces with finite aperture outside of the paraxial region can be minimized by configuring a system of lenses and manipulating the power, shape, thickness, glass

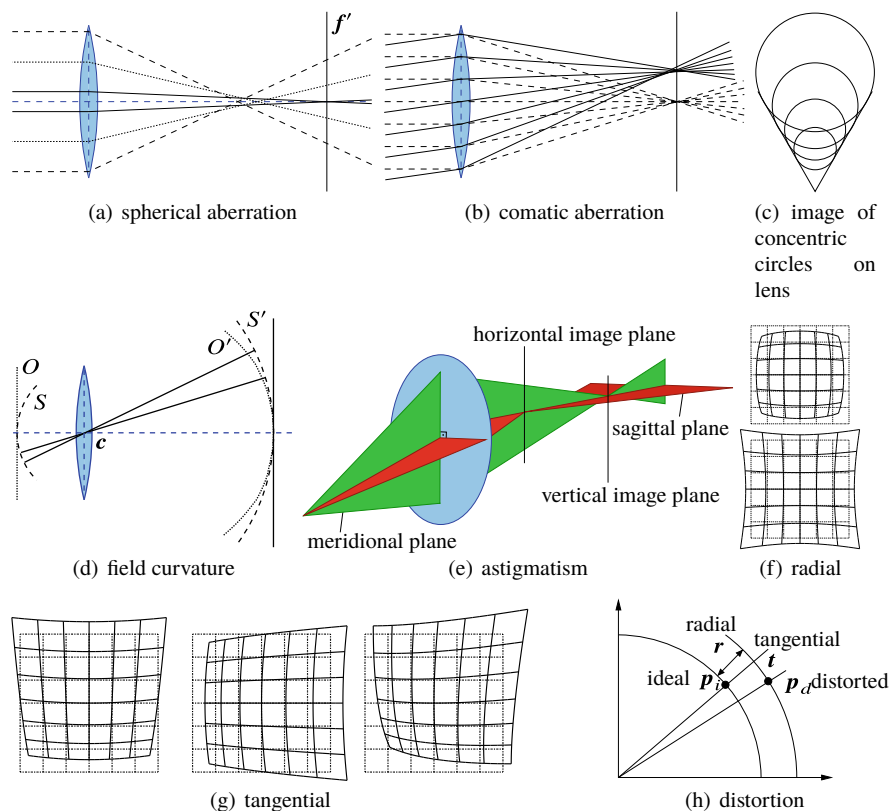


Fig. 6 The five primary aberrations. Reproduced based on [18] (a), (d), (f) [14] (b), (c), [22] (e) and [35] (h)

type, arrangement of lenses and locations of the stops. An aperture stop is either a lens or a metal plate with an aperture that imposes the most limitations to the amount of light within the lens system. The purpose of an aperture stop is to let the most amount of light through without sacrificing image quality.

The five primary aberrations are depicted in Fig. 6 and will briefly be discussed in the following. Spherical aberrations (Fig. 6a) are a direct cause from the curvature of the surface. For a converging lens marginal rays are bent too much and are focused in front of the paraxial rays. With growing distance to the optical axis the distance to the principal focus grows as well. Comatic aberrations appear already for object points that are a small distance away from the axis. In the off-paraxial region the principal planes are rather curved surfaces leading to the effect visualized in Fig. 6b. Considering a lens that is free of spherical aberrations a parallel bundle of rays (dashed lines) is refracted to focus in one point on the image plane. When rotating these rays slightly so that the bundle of rays becomes oblique, the coma is evident. The effective focal length and magnification differs for rays traversing the off-axis

region of the lens. As a consequence concentric circles on the lens are imaged in conic coma flair as depicted in Fig. 6c with a comet-like tail to which it owes its name. For a positive coma the distance of the comatic circle from the axis grows with the size of the circle on the lens. A large positive coma appears when using convex lenses while concave lenses cause a negative coma [14, 18].

Given a lens without spherical and comatic aberrations one can assume a one to one relationship between object points and image points. In the paraxial region the image of a planar object that is perpendicular to the image axis is planar. For finite apertures, however, this assumption does not hold true anymore because the distance between object point and image point clearly grows for off-axis points, resulting in a curved surface. Given a spherical object S (cf. Fig. 6d) with the center of curvature at c given its image S' is again a spherical object with the center at c because off-axis points have a longer distance to cover. Moving each object point now towards the plane O means that also each image point is moved towards the lens resulting in the paraboloidal Petzval surface O' . For a positive refracting lens the Petzval surface is curved inward while for negative lenses the Petzval surface is curved outward. An appropriate combination of positive and negative lenses negates the field curvature.

Closely related to field curvature is the astigmatism that is based on the fact that oblique rays for off-axis points converge to different points in the tangential or meridional and equatorial or sagittal plane causing two paraboloid image surfaces (cf. Fig. 6e). This effect can be reduced through lens design. The meridional plane is spanned by the optical axis and the principal ray that goes through the center of the aperture. The sagittal plane intersects the meridional plane perpendicular at the principal ray. It is actually not a single plane but changes slope when the principal ray changes direction at the surfaces of the lens system. For axial object points the cone of light is symmetrical. However, with growing angle between the optical axis and the principal ray, the difference between the focal lengths of a parallel bundle of rays increases. The focal lengths of the meridional rays decrease compared to the sagittal rays and thus the rays are tilted more with respect to the lens. This is demonstrated in Fig. 6e. Somewhere between the horizontal and the vertical image planes there is a point where the cross section of the beam is circular. This point is the so-called circle of least confusion and the ideal image plane. The astigmatism leads to two paraboloidal image surfaces on the same side of the Petzval surface.

The last primary aberration is distortion. In lens systems imperfections in the manufacturing process lead to distortions that have both radial and tangential components. When deriving the magnification it was assumed to be identical over the entire image. In real systems, however, it is often a function of the off-axis image distance. This is due to the fact that different areas of a thick lens can have different focal length. This leads to distortions of the image at whole while each individual point is in focus. This concept is called radial distortion. The two types of radial distortion from the dashed original are depicted in Fig. 6f. In the positive or pincushion distortion the off-axis magnification is greater than the optical axis magnification causing each point to move radially outward with growing distance. The negative distortion where the outer points move inward is also called barrel distortion. The direction of the distortion depends on the sign of the focal power, i.e.,

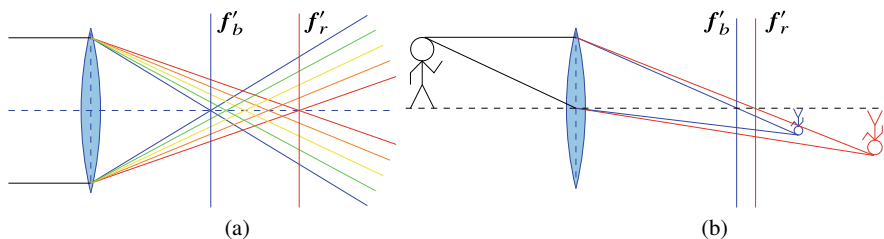


Fig. 7 Chromatic aberration. The different focal lengths for varying wavelength are exemplarily shown for blue (f'_b) and red (f'_r). Reproduced based on [18]. (a) Axial. (b) Lateral

a positive lens suffers from positive radial distortion while a negative lens suffers from negative radial distortion, respectively. Apart from radial distortions tangential distortions occur in decentered lenses where the optical center of the lens is not identical to the mechanical axis. Tangential distortions are most easily envisioned by imagining an image plane that is slightly tilted with respect to the optical axis. The effects of tangential distortion are shown in Fig. 6g, with examples of rotations around each of the axes individually and around both axes simultaneously. The combination of decentering effects as well as tilted elements within the lens system and the varying magnification leads to a point position p_d with distortion that differs from the ideal point position p_i as illustrated in Fig. 6h where both the radial distortion r and the tangential distortion t are indicated.

The five primary aberrations have an effect on the image quality even for monochromatic light. The last type of aberrations describes the effect of a lens on polychromatic light. Up to now the refractive index was considered constant for each lens. More precisely, however, the refractive index of a material is a function of the wavelength $i(\lambda)$. Thus rays of different wavelength, e.g., for a camera this corresponds to different color, are refracted differently and have different focal lengths as depicted in Fig. 7. Depending on the position of the image the edges of the blurred image appear in different colors. This effect is called axial chromatic aberration. For off-axis points the lateral chromatic aberration comes into effect. The different focal lengths based on the varying wavelength also cause a difference in transverse magnification. The different constituents of white light form thus a continuum of overlapping images that vary in size and color. As the human eye is most sensitive to yellow-green light the focus should be on the image of this color.

4 Laser Scanning

By projecting the 3D world onto a 2D image plane, photographic imaging loses information, particularly the depth. Two classes of optical measurement devices overcome this issue. Laser range imaging uses directed light and the time-of-flight principle to acquire range measurements while the triangulation principle helps to recover depth information from images based on the reflection of rays (Sect. 5).

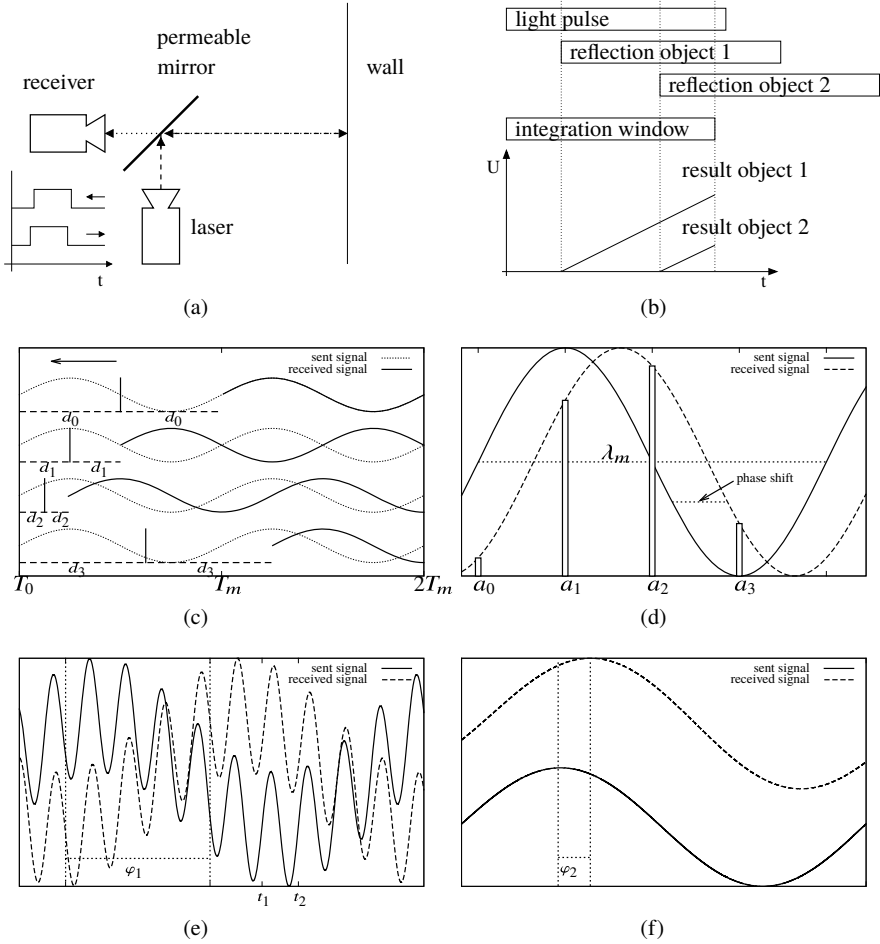


Fig. 8 Time of flight measurement principles. Reproduced from [8]. (a) direct time measurements. (b) pulse integration. (c) phase shift for changing distances. (d) phase shift calculation. (e) two frequency phase shift ω_1 . (f) two frequency phase shift ω_2

The consistency at which light spreads and the independence of the surrounding medium enable light to be used for distance measurements. Three basic methods are commonly used for time-of-flight based distance measurements [8, 21]. The concepts are outlined in Fig. 8.

First, for direct time measurements a light pulse or a sequence of pulses is sent out. A receiver waits for the return of the signal after it is reflected by an obstacle. The time between sending and receiving the light is measured and multiplied with $c/2$ to calculate the distance to the obstacle.

Second, impulse integration is often used in image based technology. Simultaneously with sending out the light pulse the sensor starts integrating the received

light for a fixed exposure time. With increasing distance to the object the time before receiving the reflected signal increases thus decreasing the amount of light returned before the end of the integration window. In Fig. 8b the concept is sketched without the influence of the ambient light. For practical applications the procedure is repeated several times to increase the accuracy and to compensate for ambient light. By taking the difference of two images, one without and one with light impulses, the remaining intensity is proportional to the distance of the reflecting objects.

Third, in contrast to the two discontinuous methods measurements with modulated light are a means for continuous distance measurements. Instead of sending out individual pulses a continuous wave is sent out, i.e., the light source is modulated with a sinusoidal signal. The modulation is either a frequency modulation or an amplitude modulation. The frequency modulated continuous wave method allows to measure several targets in the same direction simultaneously but as the signal processing is complex, it is not commonly used, instead the amplitude is modulated. Assuming the signal travels through air the relation between the modulation wave λ_m and the period time T_m is given by $\lambda_m = c \cdot T_m$. The sensor receives the returned signal shifted by the runtime resulting in a phase shift $\Delta\varphi$ between the sent and the received signal. The distance d to the object is given by:

$$d = \frac{\Delta\varphi}{2\pi} \cdot \frac{\lambda_m}{2} + n \cdot \frac{\lambda_m}{2}, \quad n = 0, 1, 2, \dots, \quad (18)$$

which means that $\Delta\varphi$ is repeated periodically with the distance to an object, thus yielding unambiguity only between 0 and 2π , i.e., up to a distance of $\lambda_m/2$. Figure 8c illustrates both the sent and the received signal for varying distances d_i , where $d_0 = \lambda_m/2$, $d_1 = \lambda_m/4$, $d_2 = \lambda_m/8$ and $d_3 = 5 \cdot \lambda_m/8$, respectively. At time T_0 the signal modulation starts. For d_0 the signal reaches the obstacle from where it is reflected after $T_m/2$, thus the return signal is received at T_m , leading to a phase shift of λ_m , i.e., a full wavelength. For decreasing distances to the object the phase shift also decreases. When the distance to the object exceeds $\lambda_m/2$, as is the case for d_3 , the phase shift is ambiguous, in the example, the phase shift for d_3 and d_2 is periodically identical.

To calculate the phase shift from the reflected signal the discrete Fourier transform is used. Four equally spaced samples are sufficient to reconstruct the return signal as demonstrated in Fig. 8d. Let $t_0 = 0$, $t_1 = T_m/4$, $t_2 = T_m/2$, $t_3 = 3T_m/4$ and a_i the amplitude of the signal received at t_i , then the period φ of the phase is calculated as:

$$\varphi = \arctan \left(\frac{a_0 - a_2}{a_1 - a_3} \right). \quad (19)$$

The modulation frequency determines directly the unambiguous range and thus the maximum range of the measurement device. With increasing wavelength the range measurement accuracy decreases. To obtain high accuracy at a long range the sent sinusoidal signal P_s is concurrently intensity modulated with two different

frequencies ω_1 and ω_2 and the return signal P_r contains the two phase shifts φ_1 and φ_2 for both modulation frequencies:

$$P_s(t) = P_{s0} \cdot (\cos(\omega_1 t) + \cos(\omega_2 t)) , \quad (20)$$

$$P_r(t) = P_{r0} \cdot (\cos(\omega_1 t + \varphi_1) + \cos(\omega_2 t + \varphi_2)) . \quad (21)$$

Figures 8e, f show the concept. The component φ_1 is unambiguous over the entire predefined measurement range and is used for the coarse quantization to determine a range window where the fine quantization component φ_2 is unique again while φ_2 is otherwise ambiguous over the effective range. Frequency selective combination of the two phase shift components yields precise and unique measurements over an increased measurement range.

Two of these measurement techniques are used in laser scanning technology. Pulsed laser scanners use the concept of direct time measurements by sending out a focused beam of light. A receiver waits for the return of the reflected light and measures the time between sending and receiving the light. Multiplied with the speed of light c this yields the distance to an obstacle. Second, phase-shift systems send out continuous waves and calculate the phase shift of the return signal to determine the distance of an obstacle. The two measurement techniques feature different advantages and disadvantages. While phase-shift systems enable high data rates due to the continuous measurements, pulsed laser scanners have to wait for each return signal, thus increasing the measurement time and decreasing the data rate. To increase the data rate modern pulsed systems such as the Riegl VZ-400i offer the option to have two pulses in the air simultaneously where ambiguities are resolved in post processing. The advantage of pulsed systems is their long range. While current phase-shift technology is limited in distance, pulsed laser scanners operate up to distances of several kilometers. A further advantage of pulsed laser scanners is the ability to perform full wave analysis. This is demonstrated in Fig. 9. Due to the beam divergence the beam splits up when hitting an edge. Part of the

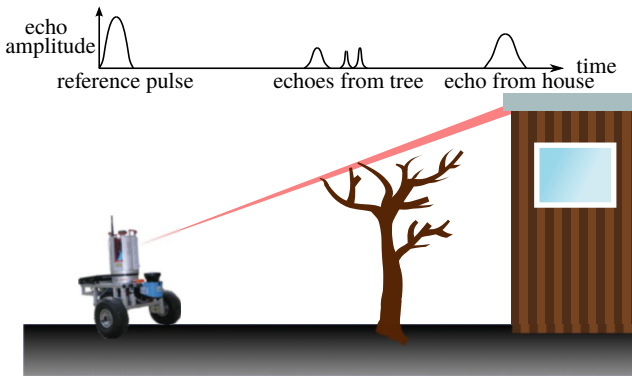


Fig. 9 The principle of full wave analysis. Multiple echoes are returned when the beam is split up at edges

beam is directly reflected towards the receiver while the other part continues until it hits another target from where it is reflected. This results in several smaller peaks in the return signal and the possibility to measure multiple points per beam. Analyzing the full wave form allows for reasoning about the perceived environment, e.g., detection of vegetation [9].

Apart from range values most laser scanners also measure the reflectivity of the target. By calculating the proportion between intensity of the measured scattered light and the sent signal the reflectivity of the object is directly derivable. The reflectivity depends on the material and the color of the object, but the intensity is also influenced by the distance and the incidence angle at which the laser beam hits the surface. To allow for a 2D laser line scan, 2D laser scanners contain a rotating mirror that directs the sent out laser beam into all directions enabling a field of view of up to 360° at fast rates. The pulsed SICK LMS100 laser scanner that is mounted at the front of Irma3D operates at a rate of 50 Hz and measures the distance to obstacles within its 270° field of view at a resolution of 0.5° . The mobile robot Kurt3D is equipped with a SICK LMS200 laser scanner that features an opening angle of 180° with a resolution between 1.0° and 0.25° . The systematic error is denoted with 15 mm and the statistical error with 5 mm.

The Riegl VZ-400 (Fig. 10, left) measures calibrated reflectivity for all targets in dB. The intensity information is the relative reflectance in dB which corresponds to the ratio of the actual amplitude of the return signal to the amplitude of a white flat target at the same distance aligned orthogonally to the laser beam. Thus, such a white target has 0 dB as intensity. The reference for 0 dB is a diffuse target. A relative reflectance higher than 0 dB results from targets reflecting with a directivity different from that of a Lambertian reflector. Such targets may be reflecting foils, corner-cube arrays, license plates, traffic signs or mirror-like objects as, e.g. window panes or blank metal aligned perpendicular to the measurement beam. This is used to create special targets from reflective foil that are identifiable in the scan based on their reflectivity. Placing them in the environment allows for georeferencing the point clouds. To capture color information Riegl offers a camera mount through which certain cameras can be controlled via the scanner to capture images after the point acquisition [27, 29].

Fig. 10 Riegl offers a camera mount for the VZ-400 (left) [28] while the Faro Focus3D (right) has an integrated camera to capture color information [10]. In a post-processing step the color information is assigned to the points



The Faro Focus3D (Fig. 10, right) is a phase-based laser scanner that captures up to 976,000 points per second with a vertical field of view of 320° at a maximum range of 120 m. Apart from intensity values the scanner is capable of capturing color information for each point with an integrated camera. The images are captured after the point cloud acquisition [10].

Figure 11 gives an example of data collected with a pulsed laser scanner as a range image (above) or a point cloud (below). The reflectance values give a sense of texture. How to obtain true color and depth information simultaneously is explained in the following.

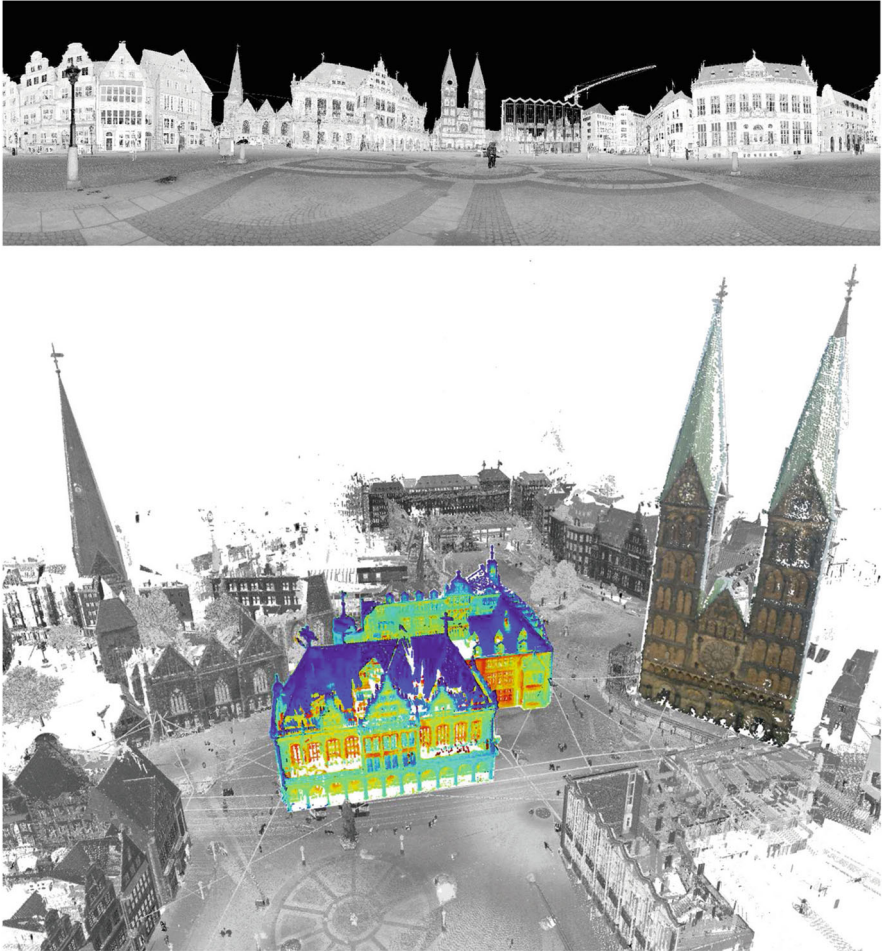


Fig. 11 3D point cloud acquired in the city center of Bremen, Germany with a pulsed time-of-flight laser scanner. Above: 3D scan in equirectangular projection. Below: the city hall has been colored with data of a coregistered thermal camera, the cathedral with data from a colored point cloud obtained from structure-from-motion

5 Triangulation

As an alternative to time of flight measurements a second class of optical distance measurement devices use the concept of triangulation (cf. [8] and [19]). Triangulation is based on the reflection of rays and the principle is sketched in Fig. 12. Consider a small bundle of rays that is emitted from a sender. At a distance of d from the principal plane of the optical lens of the sender it is reflected at a surface. A receiver is positioned at a distance B from the optical center of the sender. For simplicity the principal planes of the sender and receiver optic are coplanar and the detection plane of the receiver is parallel to this plane at a distance F . Now the ray that is reflected at the object passes through the optical center of the receiver lens and reaches the detection plane with an offset of x , called disparity. From similar triangles the relation:

$$\tan \theta = \frac{B}{d} = \frac{x}{F} \quad (22)$$

is derived, where θ is the angle between the incident and the reflected ray. When the object moves further away, the angle decreases and the offset becomes smaller according to the relation. Measuring x with a position sensitive detector allows for direct calculation of the object distance d given that F and B are known:

$$d = \frac{B \cdot F}{x} . \quad (23)$$

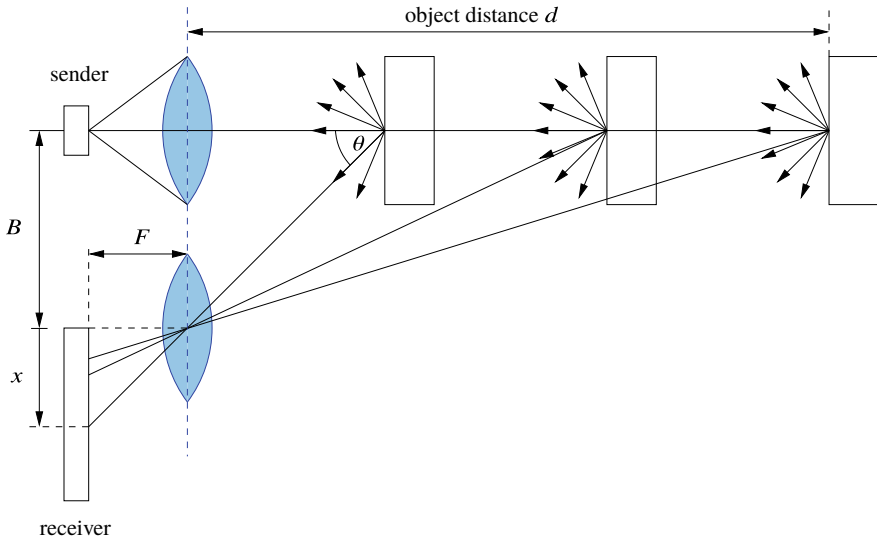


Fig. 12 Principle of triangulation. Reproduced based on [8]

B is called the baseline and influences the measurable range. The larger B the larger is the measurement range for the object distances. The importance of the baseline is also evident when looking at the sensitivity of the distance measurements, i.e., how much the distance changes depending on the changes of the offset x and subsequently how precise distance measurements can be made [19]. By differentiating Eq. (23) and substituting x results the quadratic sensitivity:

$$\frac{dd}{dx} = \frac{-d^2}{B \cdot F} . \quad (24)$$

From this it automatically follows that the precision of a sensor changes quadratically with distance. If the resolution of the offset measurement x allows for a precision of 1 mm at a distance of 1 m the precision grows to 4 mm at a distance of 2 m. At a distance of 10 m the precision reaches only 10 cm. Increasing the baseline or the distance F between optic and detection plane with a factor Δ , thus decreasing the field of view, improves the precision by a factor $\frac{1}{\Delta}$. This shows that triangulation measurements have limitations for long distances. In configurations with a laser diode typical devices have ranges between 20 and 5000 mm [8].

The concept of triangulation is not limited to single point measurements. Two main categories exist, namely stereo cameras and structured light sensors. Instead of using a sender and a receiver stereo cameras use two cameras that are fixed with a baseline. Their field of view changes slightly and consequently the resulting images differ. To determine the disparities correspondences in both images are established by detecting and matching features such as corners. Stereo cameras suffer from two problems. First, in regions with little texture no features are detected leading to a disability to obtain distance measurements. Second, features are typically not individual pixels but small patches, which decreases the resolution of the resulting depth measurements [19].

In the second category one of the cameras is replaced by a projector. Here a distinction can be made based on the types of projectors. The first type uses line laser projectors to project lines or grids on the scene while the second type uses a full image projector to project a known pattern onto the scene. Figure 13 explains the principle of a line laser. Assuming a camera is set up above the scene with the optical axis perpendicular to the ground plane. For a line laser the laser ray is diverged using a cylindrical lens to form a laser plane, which when hitting a surface appears as a line. If the line laser is placed at a distance B from the camera and projects a line onto the scene with the incident angle α a change in the height h within the scene leads to an offset b in the image. For different slope types the shape of the line changes as visualized in Fig. 13. The general relation between the offset and the height is given by:

$$h = b \cdot \tan \alpha . \quad (25)$$

Detecting the laser line in the camera image is fairly simple as long as the light is not absorbed by the object. Using a static camera one image is taken with and one

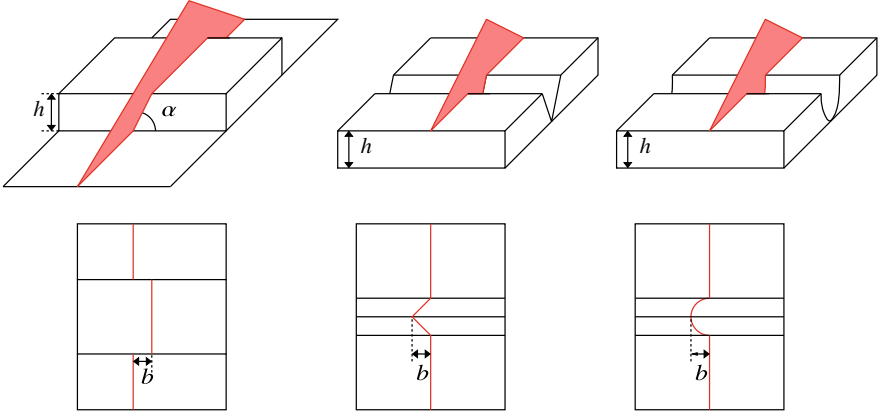


Fig. 13 Principle of line scanning. Reproduced based on [8]

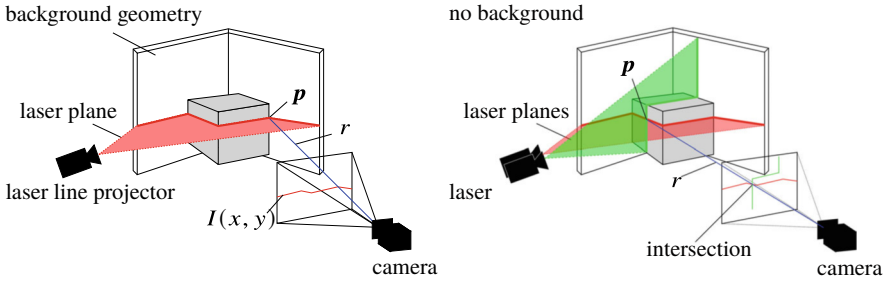


Fig. 14 Laser line projector systems using one (left) or two (right) laser planes

without the laser line. The difference image $I_d = I - I_l$ leaves only the pixels that are changed due to the laser line. However, to get a full model, the laser needs to be swiped over the scene and the relative transformation between the camera and the line laser need to be known to determine the angle α . A common solution to this problem is to use a fixed camera and to attach markers on the back plane. In [37] a reference double-frame is placed as a calibration target around the object. The laser is calibrated using the four intersections of the laser plane with the double-frame. The system proposed by Winkelbach et al. [36] is more general as it is independent of a precisely designed double frame. It is similar to the one depicted in Fig. 14a. The main advantage is the increased number of points for calculating the laser plane. The idea is to use the known geometry of the background, e.g., a corner with known angle, for calibrating the laser line.

Consider a static calibrated camera in a scene with known background geometry [36]. For the line scanner both intrinsic and extrinsic calibrations are needed. This means that not only is it known how a point in world coordinates is projected onto the image but also the position in world coordinates is known, i.e., the relative translation between the optical center of the camera and the background geometry.

Using a corner like the one depicted in Fig. 14 (left), this is achieved by attaching calibration patterns to each of the back planes. When directing the line laser at the object the laser plane intersects with the unknown object surface as well as with the known background geometry. The background points are used to calculate the parameters of the laser plane. From the difference image I_d the line pixels $Y(x)$ are calculated. Assuming a horizontal line, a single y coordinate is determined by weighted averaging over the points in column x . By intersecting the ray through the pixel $Y(x)$ with the background geometry a 3D point candidate is generated for each line pixel. A RANSAC [12] procedure filters the object points from the background points. Three linearly independent background points suffice to determine the laser plane. The remaining points are used to evaluate the guess. For the remaining object points the intersection of the ray with the laser plane gives the 3D point coordinates. Repeating this procedure while the line laser sweeps over the object yields the full visible surface. With an average or median filter multiple distance estimates for individual pixels obtained from different images are incorporated to get a better final result. A full 3D model of the object requires scans from different angles.

An alternative solution that works without a special background configuration uses a cross laser [5, 15]. Here the setup consists of a fixed calibrated camera and two line lasers that produce orthogonal laser planes as sketched in Fig. 14 (right). Each image shows two laser curves, giving $2 \cdot N$ lines in N images. The intersections of these lines from multiple images are used to determine the plane parameters for each laser plane up to a scale. Care has to be taken that laser planes with only colinear intersection points are omitted. Furthermore a geometric consistency check helps to reject inaccurately estimated laser planes. The calculation of the plane parameters is a two-step procedure. First solving a linear equation system using the coplanarity constraint, i.e., using the intersection points, yields the plane parameters up to a 4-DOF indeterminacy. The exact plane is then calculated by considering the orthogonality constraint between the laser pair from each image. The intersection of the thus calculated laser planes with the image rays yield the 3D point coordinates as in the single line laser case.

Both solutions, using one or two lasers have one major disadvantage. Each individual image captures only a small number of points. To acquire a full 3D model many images are necessary. An image projector overcomes this issue. The optics of a projector are similar to those of a camera. Instead of detecting the light it emits light rays. Instead of projecting one or two lines into the scene it projects a known pattern that covers the entire scene. Typical patterns are black and white stripes. The depth can be decoded at each pixel or at small patches of pixels, especially at edges. To improve resolution, detectivity and 3D data quality often several images with varying patterns are projected. Gray codes and sinusoidal phase-shifted patterns are popular [19]. Figure 15 shows examples of four scenes with a one line, a two line and a stripe pattern projector. For comparison the scenes are identical, even though only the single line projector requires a known background. The images reveal some drawbacks of the structured light methods. The detection is highly dependent on the lighting conditions. The black and white pattern from the projector is sensitive to high contrast, i.e., changes in lighting and in background color. In dark areas, such



Fig. 15 Structured light in various scenes using a line projector (left), a cross-line projector (center) and a projected light pattern (right). Last row, center: the accumulated lines of 26 images. As the camera and the scene remain stable the intersection detection is performed with all images

as the shorts and the shirt of the dwarf the lines disappear as well as on white parts of the polar bear and the mug. The best results are achieved on the uniformly colored body of the mannequin. A further challenge is to focus the pattern correctly on the desired distance while keeping the projector calibrated and achieving sharp edges even with the limited resolution of the projector. In this setup both the projector and the camera are static. With the requirement of a baseline this limits the field of view further as areas that are visible in the camera are occluded for the projector, an effect which is more present in crowded scenes but is also visible on the side of the mannequin. With the line projectors this effect can be eliminated by moving the projector as demonstrated for the mannequin with the cross line setup. However, with 26 images the body is still far from being completely covered hinting to the amount of images and thus amount of time needed to cover the model completely. The focused light of the line projectors makes the lines more distinguishable even in changing lighting conditions but also more vulnerable to errors due to reflections. Depending on the surface color, the colors of the laser lines change, as demonstrated

on the dwarf and on the tennis ball. When using the cross setup this causes problems when trying to distinguish between the two lines. The sensitivity to light makes structured light systems difficult to use outdoors.

What the previous methods have in common is that they require a static scene where at least one of the hardware components remains in the same position over the entire acquisition time. A second similarity is the usage of visible light which might be disturbing in an environment with humans. The PrimeSense technology [16] overcomes both these issues by combining an infrared projector with an infrared camera in one device. The most prominent device using this technology is the Microsoft Kinect that was developed for motion tracking on the Microsoft Xbox video game console. It is convenient to use an alternative that is especially popular due to its ability to draw power and communicate through USB, the Asus Xtion Pro Live RGB-D camera. Both cameras have similar specifications and consist of an infrared projector, an infrared camera and an additional RGB camera.

In the general concept a beam of light in the near IR range is sent through a diffusor. The interference among the different components of the diffused beam creates a speckle pattern that is projected into a scene. Several reference images for different surface distances are stored on the device. When capturing an image of a surface in the measurement area, this image is compared to the reference images to determine the distance. This is done by correlating a block of 19×19 pixels to the reference images along a horizontal line [19]. Locations with local peaks for the cross-correlation indicate the displacement of the points on the object surface, i.e., the disparity that corresponds to the angle with respect to the projector and thus the change in distance compared to the reference image as given by the triangulation equations. The method requires the speckle pattern to be relatively invariant over a range of distances and unique over a horizontal area leading to strong cross-correlation values only in the range of the 3D measurement [16, 19].

The Asus Xtion Pro Live RGB-D camera operates best at a distance of 0.8 to 3.5 m at an opening angle of $58^\circ \times 45^\circ$. The depth image has a resolution of 640×480 pixels, the RGB image a resolution of 1280×1024 pixels with 30 frames per second. The baseline between the IR projector and the IR camera is approximately 7.5 cm. The RGB camera is between the two at a distance of approximately 2.5 cm from the IR camera. With a depth precision of 11 bits the actual depth decision is limited by the resolution. Due to the smearing effects of the block correlation the spatial precision is lower than suggested by the resolution of the depth image. The pattern expands and contracts based on temperature leading to disparities along the vertical axis. This variation is compensated based on measurements with a temperature sensor but it is visible when examining subsequent images captured by the IR camera closely. [16, 19]

Figure 16 shows the speckle pattern as seen by the IR camera of the Asus Xtion Pro Live RGB-D camera. The first image shows the speckle pattern on a white wall. At a closer look the pattern appears to be arranged into nine smaller rectangles each of which contains one brighter white circle. In the second image a soccer ball and a board are placed in front of the wall. The change in the speckle pattern is apparent. The reflective characteristics of the ball let the pattern appear much darker

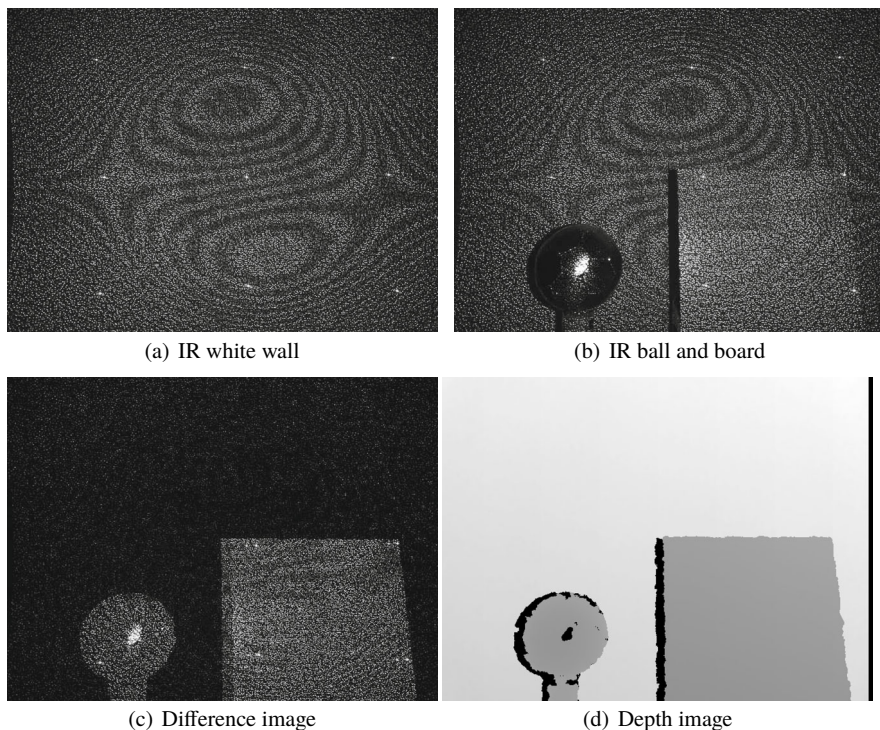


Fig. 16 Images taken with the Asus Xtion Pro Live RGB-D camera. **(a)** The infrared image captured by the infrared camera from a white wall. **(b)** The infrared image with a ball and a board in front of the wall. **(c)** The difference image between **(a)** and **(b)**. The depth image corresponding to **(b)**

while in the center specular reflectance causes a white spot. More evident is the change when looking at the difference image. In the parts where the wall is still visible in the second image, there are small differences caused by noise and by the temperature variation, but the ball and the board clearly stick out. The depth image demonstrates that the specular reflections on the ball prevent the device from taking depth measurements. The individual data for a more complex scene is depicted in Fig. 17. The speckle pattern is clearly visible in the image. Again the reflectance on the ball but also the fabric of the two animals cause problems for the depth determination and holes in the depth image and consequently in the point cloud. Where the moose is sitting on the ball reflections are visible on the ball surface that correspond to the large hole in the depth image. On the black squares of the chessboard pattern it is hard to detect the speckles with the human eye but the depth image shows that the device succeeded. Nevertheless, materials that absorb the infrared light are a further source for dropouts in the data.

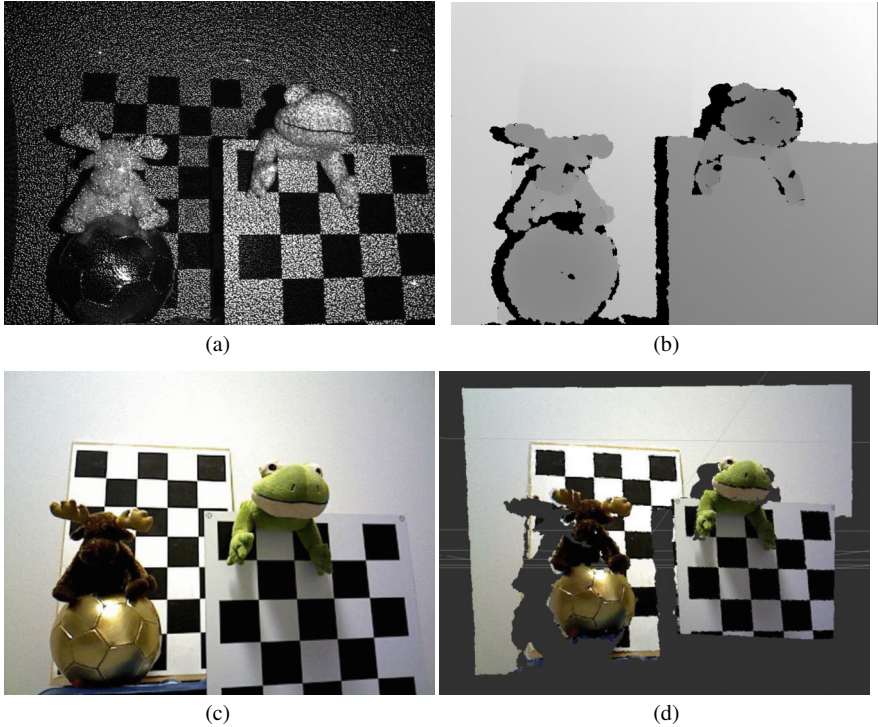


Fig. 17 A scene captured with the Asus Xtion Pro Live RGB-D camera. (a) IR image. (b) Depth image. (c) RGB image. (d) The point cloud colored with the RGB data

6 Incremental Registration

The previous sections explained the working principle of sensors often used on robotic systems to perceive the environment. However, each sensor measurement captures only a small part of the scene. To create a full map, the data acquired at different positions needs to be jointed into one common coordinate system, a process often referred to as image or scan registration or matching. This section focuses on how the information from the camera (Sect. 6.1) or 3D laser scanner (Sect. 6.2) is used to create a map of the environment that the robot traverses. 3D environment mapping using 3D scanners on mobile robots is subject to intensive research [23]. Given a map, the sensor data allows for localizing the robot in it. When a robot explores an unknown environment it has no access to such a map. However, given the perfect localization of the robot in a reference coordinate system the perceived sensor data can immediately be transformed into this reference frame. Unfortunately, robot localization is prone to errors. Accordingly, a robot in an unknown environment faces an optimization problem, the problem of simultaneously having to create a map and to localize itself in this map given erroneous input information.

This problem is known as the SLAM (Simultaneous Localization and Mapping) problem. It is closely related to bundle adjustment known from Photogrammetry. For details, please refer to Chap. [KNO8](#).

6.1 Registration of Images

In order to reconstruct the 3D environment from 2D cameras the properties influencing the image formation, that is the parameters of the camera model, need to be known. Additionally, the knowledge about the pose is essential. The following explains the process of determining the geometric camera properties through calibration before outlining methods to estimate the motion between two images.

6.1.1 Camera Calibration

Combining several sensor components within one system requires knowledge of their geometric properties. This is achieved by geometric calibration. The calibration consists of two parts, namely intrinsic and extrinsic calibration.

Intrinsic calibration for a camera describes the process of determining the internal parameters of the camera that define how a point in world coordinates is transformed into image coordinates, i.e. projected onto the image plane. The standard camera model consists of two parameters for the focal length, two coordinates for the camera center, three radial distortion coefficients and two tangential distortion coefficients. The standard method to determine them is to take images of a pattern with known features. The image coordinates of the features and their known relation in world coordinates are used to formulate a non-linear least squares problem that is solved using the Levenberg-Marquardt algorithm [38].

Extrinsic calibration refers to the process of determining the relative pose (orientation and translation) between two components. The prerequisite for extrinsic calibration is the intrinsic calibration of the devices.

To reference several cameras to the coordinate system of a laser scanner, for example, we need the specific properties of both coordinate systems. Each camera has unique parameters that define how a point $\mathbf{m} = (x, y, z)^T$ in world coordinates is projected onto the image plane. These parameters are calculated through a process known as geometric camera calibration. Given the focal length (f_x, f_y) of the camera and the camera center (c_x, c_y) based on the pinhole camera model (cf. Sect. 3.2) the image coordinates $\bar{\mathbf{m}} = (\bar{x}, \bar{y})$ are calculated as:

$$\begin{pmatrix} \bar{x} \\ \bar{y} \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_A \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix}. \quad (26)$$

$\mathbf{A}$ is called the intrinsic or camera matrix. Given the radial distortion coefficients k_1, k_2, k_3 and the tangential distortion coefficients p_1, p_2 and $r = \sqrt{\bar{x}^2 + \bar{y}^2}$ the image points (x_d, y_d) in the distorted image are calculated as:

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \begin{pmatrix} \bar{x}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 \bar{x} \bar{y} + p_2(r^2 + 2\bar{x}^2) \\ \bar{y}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2\bar{y}^2) + 2p_2 \bar{x} \bar{y} \end{pmatrix}. \quad (27)$$

To determine the parameters of optical cameras, chessboard patterns are commonly used because the corners are reliably detectable in the images. A number of images N showing a chessboard pattern with known number M and size of squares are recorded. Assuming $\hat{z} = 1$, $\hat{x}$ and $\hat{y}$ the coordinates of the pattern features given through their known distance, and the rotation $\mathbf{R}$ and translation $\mathbf{t}$ that relate the pattern to the camera coordinate system the projection is defined up to a scale factor s by:

$$s \begin{pmatrix} \bar{x} \\ \bar{y} \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}}_{(\mathbf{R}, \mathbf{t})} \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \\ 1 \end{pmatrix}. \quad (28)$$

Detecting the internal corners in each image allows to formulate Eqs. (28) and (27) as a non-linear least squares problem and to solve for the calibration parameters [6, 38]:

$$\sum_{i=1}^N \sum_{j=1}^M \|\bar{\mathbf{m}}_{ij} - \hat{\mathbf{m}}(\mathbf{A}, \mathbf{D}, \mathbf{R}_i, \mathbf{t}_i, \mathring{\mathbf{m}}_j)\|^2. \quad (29)$$

$\bar{\mathbf{m}}_{ij}$ are the detected coordinates of corner j in image i . $\hat{\mathbf{m}}(\mathbf{A}, \mathbf{D}, \mathbf{R}_i, \mathbf{t}_i, \mathring{\mathbf{m}}_j)$ is the projection function of point $\mathring{\mathbf{m}}_j$ in image i , according to Eqs. (28) and (27) using the camera matrix $\mathbf{A}$ and the distortion coefficients $\mathbf{D}$. The rotation matrix $\mathbf{R}_i$ and the translation vector $\mathbf{t}_i$ describe the position and orientation of image i in relation to the camera.

6.1.2 Two View Estimation

Estimating the motion between two images equals the stereo geometry problem. In principle, a sequence of image registrations can be performed by considering two consecutive images as a stereo camera. Given an image point in one image, in two view geometry one can restrict the position of the corresponding image point in the other image. The relationship of the two views is estimated by a set of corresponding points.

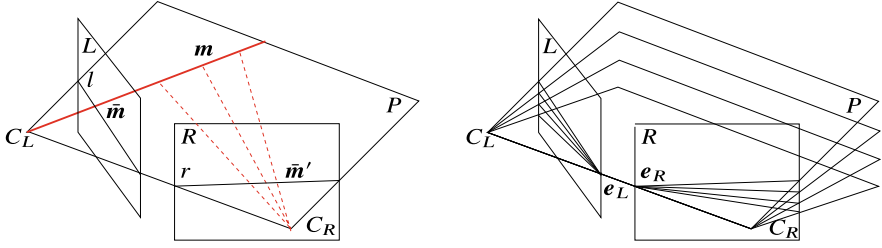


Fig. 18 Two view geometry. Left: correspondence problem. Given a 3D point $\mathbf{m}$ and its image in the left camera C_L $\tilde{\mathbf{m}}$, the search for the corresponding point $\tilde{\mathbf{m}}'$ can be restricted to searching along a line as $\mathbf{m}$, $\tilde{\mathbf{m}}$, $\tilde{\mathbf{m}}'$ from a plane. Right: General epipolar geometry. The line connecting C_L and C_R defines a set of planes and the epipoles $\mathbf{e}_L$ and $\mathbf{e}_R$

Given an unknown 3D point $\mathbf{m}$ it is bound to be on the line of sight of the corresponding image point $\tilde{\mathbf{m}}$. The line of sight is projected to the other image and thus, the search for a corresponding point $\tilde{\mathbf{m}}'$ is restricted to search along this line (cf. Fig. 18, left).

Following the notation of [26] every plane passing through both centers of projections C_L and C_R results in such a set of corresponding epipolar lines, as given in Fig. 18 (right). All these lines pass through two specific points $\mathbf{e}_L$ and $\mathbf{e}_R$, the epipoles, and they are the projection of the center of projection in the opposite image.

As $\tilde{\mathbf{m}}$ is on a line l it holds: $l^T \tilde{\mathbf{m}} = 0$. The line passing through $\tilde{\mathbf{m}}$ and the epipole $\mathbf{e}_L$ is:

$$l \sim [\mathbf{e}_L]_{\times} \tilde{\mathbf{m}} \quad (30)$$

where $[\mathbf{e}_L]_{\times}$ is the 3×3 matrix representing the vectorial product with $\mathbf{e}_L$:

$$[\mathbf{e}_L]_{\times} = \begin{pmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{pmatrix}. \quad (31)$$

The plane Π corresponding to l is easily obtained as $\Pi \sim \mathbf{P}^T l$ and similarly $\Pi \sim \mathbf{P}'^T r$. Putting these equations together results in:

$$r \sim \left(\mathbf{P}'^T \right)^{\dagger} \mathbf{P}^T l \equiv \mathbf{H}^{-T} l \quad (32)$$

where $\dagger$ indicates the pseudo-inverse. Substituting Eq. (30) into (32) yields

$$r \sim \mathbf{H}^{-T} [\mathbf{e}_L]_{\times} \tilde{\mathbf{m}}. \quad (33)$$

Defining $\mathbf{F} = \mathbf{H}^{-T} [\mathbf{e}_L]_{\times}$, we obtain:

$$\mathbf{r} \sim \mathbf{F} \bar{\mathbf{m}} \quad \text{and thus} \quad \bar{\mathbf{m}}' \mathbf{F} \bar{\mathbf{m}} = 0. \quad (34)$$

$\mathbf{F}$ is called the fundamental matrix, which was first introduced by Faugeras [11] and Hartley [17]. Having the calibration of a stereo camera system, $\mathbf{F}$ can be computed and a constraint is obtained for corresponding points.

When the calibration, i.e., the rotation and translation $(\mathbf{R}, \mathbf{t})$ between the images, is not known Eq. (34) is used to compute the fundamental matrix $\mathbf{F}$. Thus, it is possible to recover the motion between two images, given corresponding points. Every pair of corresponding points gives one constraint on $\mathbf{F}$. Since $\mathbf{F}$ is a 3×3 matrix which is only determined up to scale, it has $3 \cdot 3 - 1$ unknowns. Therefore, 8 pairs of corresponding points suffice for computing $\mathbf{F}$ with a linear algorithm. Many methods exist to obtain the fundamental matrix $\mathbf{F}$ from two uncalibrated images [34, 39]. The 8-point algorithm, is one of these methods.

The 8-Point Algorithm

The fundamental matrix $\mathbf{F}$ is a 3×3 matrix determined up to an arbitrary scale factor, 8 equations are required to obtain a unique solution. The simplest way to compute the fundamental matrix consists of using Eq. (34). This equation can be rewritten in the following form:

$$\begin{bmatrix} xx' & yx' & x'x & xy' & yy' & y'y & x & y & 1 \end{bmatrix} \mathbf{f} = 0 \quad (35)$$

where $\bar{\mathbf{m}} = [x \ y \ 1]^T$, $\bar{\mathbf{m}}' = [x' \ y' \ 1]^T$ and $\mathbf{f} = [F_{11} \ F_{12} \ F_{13} \ F_{21} \ F_{22} \ F_{23} \ F_{31} \ F_{32} \ F_{33}]^T$ is a vector containing the elements of the fundamental matrix $\mathbf{F}$. By stacking eight of these equations in a matrix $\mathbf{A}$ the following equation is derived:

$$\mathbf{A} \mathbf{f} = 0. \quad (36)$$

This system of equations is usually solved by Singular Value Decomposition (SVD). Applying SVD to $\mathbf{A}$ yields the decomposition $\mathbf{U} \mathbf{S} \mathbf{V}^T$ where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices and $\mathbf{S}$ is a diagonal matrix containing the singular values. These singular values σ_i are positive and in decreasing order. Therefore in our case σ_9 is guaranteed to be identically zero (8 equations for 9 unknowns) and thus the last column of $\mathbf{V}$ is the correct solution [26].

The fundamental matrix $\mathbf{F}$ is reconstructed from the solution vector $\mathbf{f}$. In presence of noise, the rank-2 constraint will not be satisfied. Thus, there will be no real epipoles through which all epipolar lines pass, but these will be “smeared out” to a small region. A solution to this problem is to obtain $\mathbf{F}$ as the closest rank-2 approximation of the solution coming out of the linear equations [26]. Since

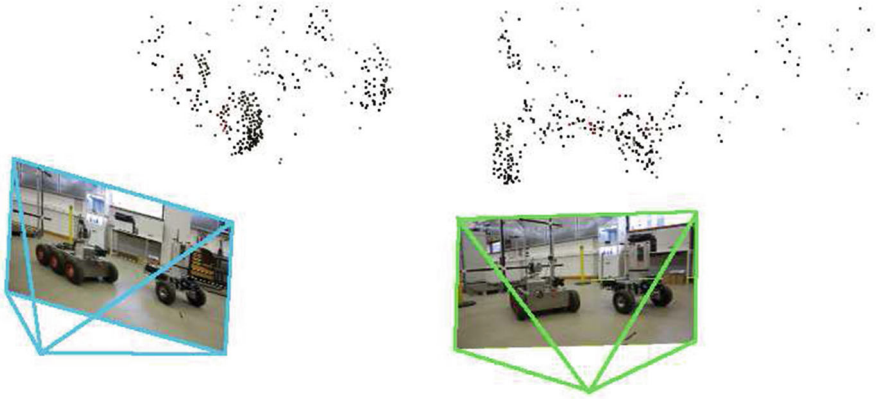


Fig. 19 Two photos registered for stereo matching. A sparse 3D point cloud is shown in the back

F contains besides the intrinsic camera parameters A the rotation and translation (R, t) between the views, i.e.,

$$F = A_R^{-T} [t]_{\times} R A_L^{-T}, \quad (37)$$

one can also search for the closest valid rotation matrix. Figure 19 shows two registered camera images. The image orientation has been calculated from matching image features.

6.2 Registration of 3D Point Cloud Data

While image registration has to recover 3D object positions from 2D images to determine the relative pose between two images, 3D point cloud data does already represent the 3D geometry of the scene correctly. Thus, three precise point correspondences would be sufficient to compute the transformation between two sets of points acquired at different poses, but to compensate for noise practical applications typically use significantly more point pairs. Finding these corresponding points is the most challenging part in 3D point cloud registration. In the following the general concept is first explained using a method that works on the raw point data before outlining an alternative that relies on detecting features.

6.2.1 The ICP Algorithm

The ICP algorithm (iterative closest point) is one of the most common algorithms for registering two point clouds. First proposed in 1992 [4], it has hence developed into a state of the art for optimizing the alignment of two point clouds. Its

major drawback is the necessity to have pose estimates available. In robotics, this requirement is fulfilled if the robot is equipped with sensors to measure odometry. For other data sets, GPS measurements or feature-based approaches (see Sect. 6.2.2) provide these.

Starting from such initial pose estimates the algorithm calculates effectively the correct transformation between two point clouds by means of minimizing distances between corresponding points. Corresponding points are chosen based on Euclidean distances. This is described in Algorithm 1.

Algorithm 1 The ICP algorithm

Require: point clouds M and D

- 1: find point correspondences
- 2: minimize for rotation $\mathbf{R}$ and translation $\mathbf{t}$

$$E_{\text{ICP}}(\mathbf{R}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^N \|\mathbf{m}_i - (\mathbf{R}\mathbf{d}_i + \mathbf{t})\|^2 \quad (38)$$

- 3: **return** pose $(\mathbf{R}, \mathbf{t})$
 - 4: iterate 1 and 2
-

Given an already registered point cloud M and a point cloud D with an initial pose estimate the ICP first tries to find for each point $\mathbf{d}_i$ from D the point $\mathbf{m}_i$ in M that is closest to $\mathbf{d}_i$. N is the number of point correspondences. Then one needs to solve for a transformation $(\mathbf{R}, \mathbf{t})$ (translation $\mathbf{R}$ and orientation $\mathbf{t}$) that minimizes the error function E_{ICP} , i.e., the distance between all point pairs. The transformation is applied to the point cloud D and the two steps are iterated to find the best final transformation between the two point clouds. For best results a threshold t_{dist} is introduced and all point pairs with a distance larger than t_{dist} are discarded from the calculations. Without the threshold t_{dist} the ICP is proven to converge monotonously towards a local minimum E_{min} as in each iteration the error is reduced [4]. In the next iteration the point correspondences are either the same as in the previous iteration or they have a smaller distance. When introducing the threshold the number of point correspondences N is not constant anymore. Point correspondences may be removed when the distance between the points increases or more point correspondences may be found when points are moved within the range of the threshold thus increasing the actual error value. In practice this procedure improves the matching results especially in areas with large occlusions as wrong correspondences from occluded areas have less influence on the registration.

For multiple scans the pairwise scan matching is adopted as follows: using the first scanning position as reference the n th scan is always registered against the $(n - 1)^{\text{th}}$ scan. This way all scans are sequentially transformed into the same coordinate system.

To minimize the error function different methods have been presented in literature. These method can briefly be divided into two categories, namely direct

and indirect methods. Indirect methods as Gradient Descent, the Gauss-Newton method or the Levenberg-Marquardt algorithm evaluate the error function (38) in various positions thus needing more time. Nüchter et al. describe several closed form solutions to the ICP problem [24]. Here the method using singular value decomposition (SVD) will be explained. The difficulty of minimizing the error function lies in the preservation of the orthonormality constraint of the rotation matrix $\mathbf{R}$. To enforce orthonormality the minimization is split into two parts, calculating the rotation first and deriving the translation afterwards from the results. To compute the rotation, the point sets are shifted such that the centroid lies on the origin of the coordinate system:

$$\mathbf{M}' = \{\mathbf{m}'_i = \mathbf{m}_i - \mathbf{c}_m\}_{1,\dots,N}, \quad \mathbf{D}' = \{\mathbf{d}'_i = \mathbf{d}_i - \mathbf{c}_d\}_{1,\dots,N} \quad (39)$$

by subtracting the mean:

$$\mathbf{c}_m = \frac{1}{N} \sum_{i=1}^N \mathbf{m}_i, \quad \mathbf{c}_d = \frac{1}{N} \sum_{i=1}^N \mathbf{d}_i. \quad (40)$$

The error function is adapted introducing the points $\mathbf{m}'_i$ and $\mathbf{d}'_i$ from the translated point clouds $\mathbf{M}'$ and $\mathbf{D}'$:

$$E_{\text{ICP}}(\mathbf{R}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^N \|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i - \underbrace{(\mathbf{t} - \mathbf{c}_m + \mathbf{R}\mathbf{c}_d)}_{=\tilde{\mathbf{t}}}\|^2 \quad (41)$$

$$= \frac{1}{N} \sum_{i=1}^N \|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i\|^2 - \frac{2}{N} \tilde{\mathbf{t}} \sum_{i=1}^N (\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i) + \frac{1}{N} \sum_{i=1}^N \|\tilde{\mathbf{t}}\|^2 i. \quad (42)$$

Minimizing Eq. (42) requires minimization of each of the three individual terms. The second sum computes to zero, as it is essentially the difference of the centroids of the new point clouds $\mathbf{M}'$ and $\mathbf{D}'$ which are designed to be zero. The third part has a minimum at $\tilde{\mathbf{t}} = 0$ which dissolves to:

$$\mathbf{t} = \mathbf{c}_m - \mathbf{R}\mathbf{c}_d. \quad (43)$$

Accordingly the minimization reduces to calculating the rotation to minimize the first term from Eq. (42):

$$E_{\text{ICP}}(\mathbf{R}, \mathbf{t}) \propto \sum_{i=1}^N \|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i\|^2 \quad (44)$$

and computing the translation according to Eq. (43).

Calculating the optimal rotation using SVD was introduced by Arun, Huang and Blostein [2]. Using matrix representation, the rotation $\mathbf{R}$ has to be an orthonormal 3×3 matrix. To find the optimal $\mathbf{R}$ the cross-correlation Matrix $\mathbf{H}$ is constructed from the point pairs as:

$$\mathbf{H} = \sum_{i=1}^N \mathbf{m}'_i \mathbf{d}'_i{}^T = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix} \quad (45)$$

where:

$$S_{xx} = \sum_{i=1}^N m'_{x,i} d'_{x,i}, \quad S_{xy} = \sum_{i=1}^N m'_{x,i} d'_{y,i}, \dots \quad (46)$$

Deriving the matrices $\mathbf{V}$ and $\mathbf{U}$ by singular value decomposition $\mathbf{H} = \mathbf{U} \mathbf{\Lambda} \mathbf{V}^T$ the rotation is calculated as $\mathbf{R} = \mathbf{V} \mathbf{U}^T$.

Figure 20 shows the ICP-based registration of two 3D scans. The main drawback of the ICP algorithm is its necessity to have initial pose estimates available. For small pose changes the estimates are negligible, however, given the long range and the scan acquisition times of 3D laser scanners the assumption of small changes usually does not hold true. Thus, additional methods are used to estimate the pose of the next scan. Global navigation satellite systems (GNSS) are restricted to outdoor scenarios and are limited in accuracy.

The alternative for the initial guess is to use natural features, this is described in Sect. 6.2.2. In the robotics context the method of choice is to let the robot track and estimate the path while it moves from one scanning position to the next. Odometry is often applied but not very accurate. Laser scanners are among the most accurate measurement devices on robots. Thus, an often applied method is to use fast 2D laser scanners to track the robot position over time and create a map simultaneously.

6.2.2 Feature-Based Registration

ICP does not rely on features—however it still needs structure—, instead it correlates points with the closest Euclidean distance. To be successful, it needs a good starting estimate. Feature based methods establish correspondences by comparing descriptions of points of interest, i.e., features. Existing automatic methods use either structural 3D features or image features. Well-known structural features are planar patches [25], THRIFT [13], or point feature histograms [32, 33]. Image features can also be obtained from images generated from 3D scans. In addition to measuring distances, current laser scanners are able to determine the amount of light returning to the scanner. On such images, SIFT [20], SURF [3], FAST [30], BRIEF [7], ORB [31], or CenSurE [1] can be computed.

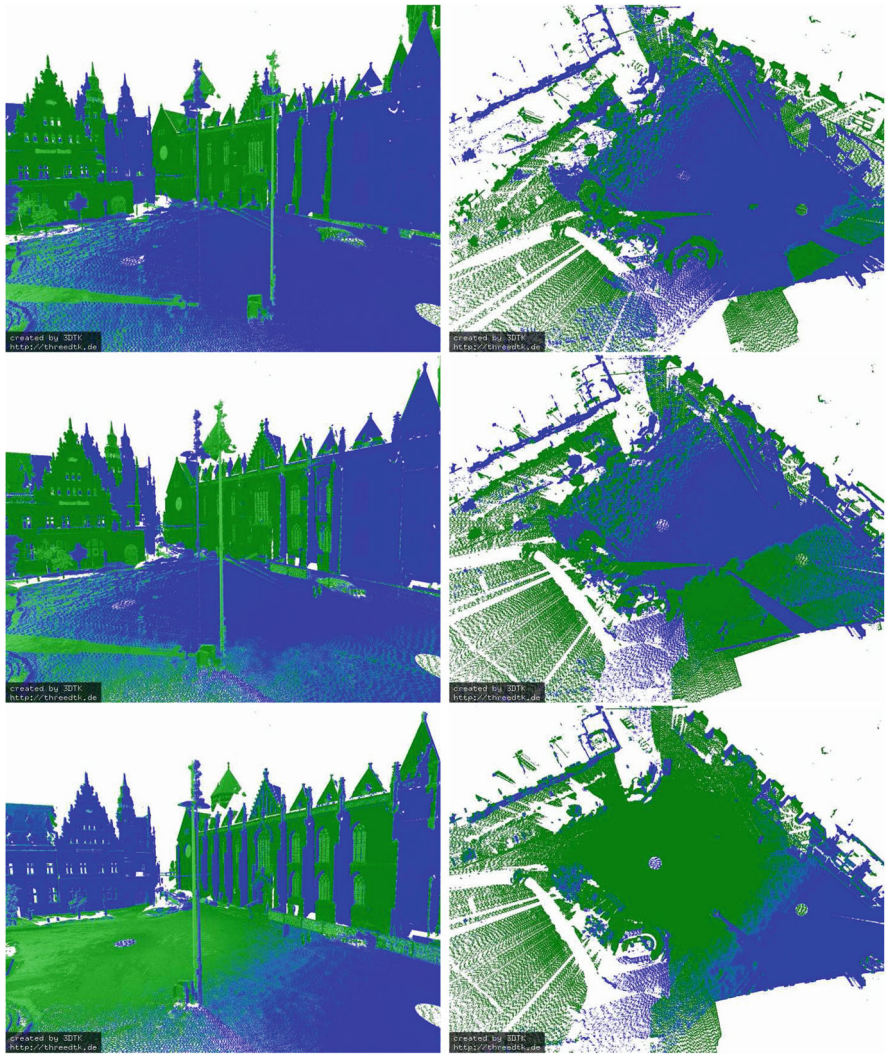


Fig. 20 Registration of 3D point clouds. The scene has been captured in the city center of Bremen. Left: 3D point cloud view. Right: bird-eye view. Above: Initial orientation based on rough estimates. Middle: 5 ICP iterations. Below: final ICP result

Figure 21 shows two 3D laser scans in Pannini projection [19] with extracted SIFT features. Matching the corresponding SIFT descriptors yields several correct, but also incorrect matches. Thus, the RANSAC algorithm [12] is commonly used to filter for the correct ones. Equation (38) and the minimization using the SVD is used to compute the registration parameters. Fine registration is usually performed by ICP, afterwards.

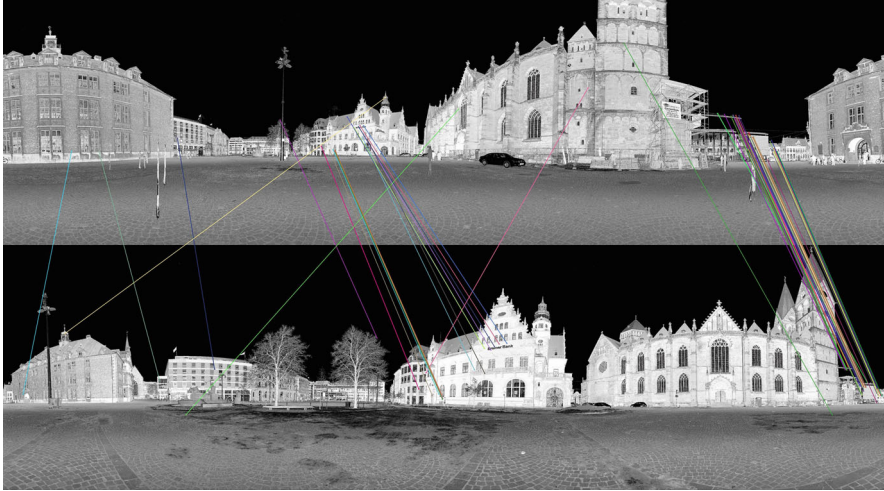


Fig. 21 Feature based matching of 3D point clouds using SIFT features and descriptor on Pannini projections of two scans acquired in Bremen city

7 State Estimation with Bayes Filters

Up to now, we use unimodal deterministic calculations for pose estimation based on odometry or on incremental registration. Real systems however, have to cope with imprecise measurements and actions. These inaccuracies or errors can be of systematic or stochastic nature and lead to system and environment states that are of statistic nature as well. If it is possible to estimate the quality of the measurement by some variance or co-variance we have the basis for *fusing* independently acquired information and to reduce the overall uncertainty. Obviously one wants to weight more reliable measurements to a higher degree. The methods presented in the following lead to a statistically better solution as the original separate solutions. The filters are also used in many other fields, whenever the task is to fuse sensor data of time discrete measurements.

7.1 Basics

If two facts are related by cause and effect, then it is often only possible to observe the effect. This *diagnostic* dependency, i.e., the probability distribution $P(\text{causation}|\text{effect})$ is often hard to obtain. On the other hand, the *causal* dependency $P(\text{effect}|\text{causation})$ is relatively easy to obtain, e.g., by counting

occurrences. If the causal dependency is known, then Bayes theorem gives the conversion:

$$P(x | y) = \frac{P(y | x)P(x)}{P(y)} . \quad (47)$$

Furthermore the following notation:

$$P(x | y) = \eta P(y | x)P(x) \quad (48)$$

with the normalization factor $\eta = P(y)^{-1} = (\sum_v P(y | v)P(v))^{-1}$ is common. In case of many quantities the formula generalises as:

$$P(x | v_1, \dots, v_n) = \frac{P(v_n | x, v_1, \dots, v_{n-1})P(x | v_1, \dots, v_{n-1})}{P(v_n | v_1, \dots, v_{n-1})} . \quad (49)$$

If the v_i are state-independent variables, i.e., an action or a measurement, then the above formula can be simplified using the Markov assumption. The assumption postulates that a next state only depends on the previous state and the last action, but not on previous actions or states. Therefore, we yield:

$$\begin{aligned} P(x | v_1, \dots, v_n) &= \frac{P(v_n | x, v_1, \dots, v_{n-1})P(x | v_1, \dots, v_{n-1})}{P(v_n | v_1, \dots, v_{n-1})} \\ &= \frac{P(v_n | x)P(x | v_1, \dots, v_{n-1})}{P(v_n | v_1, \dots, v_{n-1})} \\ &= \eta P(v_n | x)P(x | v_1, \dots, v_{n-1}) \\ &= \eta_{1, \dots, n} \prod_{i=1, \dots, n} P(v_i | x)P(x) . \end{aligned} \quad (50)$$

The Markov assumption simplifies the calculations but is often violated in practice, which can be seen in the following example: assume localization of a robotic system in a given map. Theoretically, in an ideal environment, the sensor values only depend on the pose of the system (state) such that the system uses the predicted and measured sensor values for calculations. If the environment is dynamic, e.g., humans walk by, doors open and close, etc., then the measurements are systematically corrupted. This could be detected by analyzing the history of measurements, but not from a single state and the given map.

7.2 Bayes Filter

We search for a function Γ , that calculates the probability of a next state x_{t+1} , depending on the previous state as well as a sequence of *evidences* e_i , where every e_i is either a measurement z or an action u :

$$P(x_{t+1} | e_{1:t+1}) = \Gamma(e_{1:t+1}, P(x_t | e_{1:t})) . \quad (51)$$

The notation $x_{a:b}$ denoted the sequence $(x_a, \dots, x_b)_{a \leq b}$. An ansatz for such a function Γ is given by the conversion:

$$\begin{aligned} P(x_{t+1} | e_{1:t+1}) &= P(x_{t+1} | e_{1:t}, e_{t+1}) \\ &= \eta P(e_{t+1} | x_{t+1}, e_{1:t}) P(x_{t+1} | e_{1:t}) \\ &= \eta \underbrace{P(e_{t+1} | x_{t+1})}_{\text{filter}} \underbrace{P(x_{t+1} | e_{1:t})}_{\text{prediction}} \end{aligned} \quad (52)$$

exploiting Bayes rule and the Markov assumption. The prediction can be further simplified to:

$$\begin{aligned} P(x_{t+1} | e_{1:t}) &= \int_{x_t} P(x_{t+1} | x_t, e_{1:t}) P(x_t | e_{1:t}) \\ &= \int_{x_t} \underbrace{P(x_{t+1} | x_t)}_{\text{transition model}} \underbrace{P(x_t | e_{1:t})}_{\text{curr. state}} . \end{aligned} \quad (53)$$

Consolidating the above, yields the recursive formula:

$$P(x_{t+1} | e_{1:t+1}) = \eta P(e_{t+1} | x_{t+1}) \int_{x_t} P(x_{t+1} | x_t) P(x_t | e_{1:t}) . \quad (54)$$

An alternative approach is given next: up to now, we have calculated the state—more precisely its probability—based on an evidence (action and measurement) and the previous state. Now, we are looking for the probability of a state, given a sequence of alternating actions u_i and measurements z_i . Furthermore, we have a sensor model $P(z | x)$ which represents the uncertainties of the measurements and an action model $P(x | u, x')$ as well as an initial a-priori probability of the system state $P(x)$ (cf. Fig. 22).

Using the Markov assumption, we have:

$$P(z_{t+1} | x_{0:t+1}, z_{0:t}, u_{1:t+1}) = P(z_{t+1} | x_{t+1}), \quad (55)$$

$$P(x_{t+1} | x_{0:t}, z_{0:t}, u_{1:t+1}) = P(x_{t+1} | x_t, u_{t+1}) . \quad (56)$$

By the Markov assumption, we simplify the problem implicitly: first of all, a static world is assumed, i.e., there are no changes except of the actions. Furthermore,

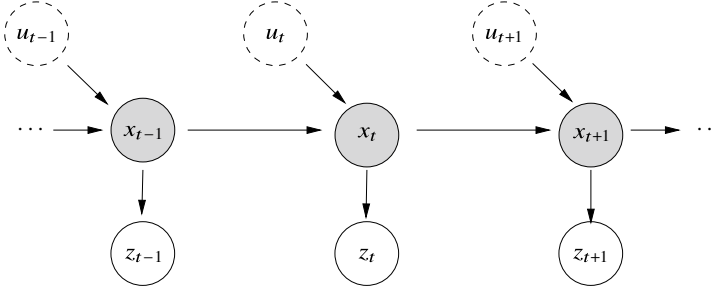


Fig. 22 Basic principle for a stream of states x_i , u_i actions and z_i measurements in a Bayes filter

we assume independent noise and perfect models. A violation of the Markov assumption does however only imply that we approximate the corresponding probabilities.

We want to have a statistical estimation of the state x of a dynamic system, given all previous measurements and actions. Both measurements and actions *alternate* to fulfill the requirement of independence. The a-posteriori estimation of the new state x_{t+1} is also called belief $Bel(x_{t+1})$ and is defined as:

$$Bel(x_{t+1}) = P(x_{t+1} | z_0, u_1, z_1, \dots, u_{t+1}, z_{t+1}) . \quad (57)$$

It is calculated as:

$$\begin{aligned}
 Bel(x_{t+1}) &= P(x_{t+1} | z_0, u_1, z_1, \dots, u_{t+1}, z_{t+1}) \\
 &= \eta P(z_{t+1} | x_{t+1}, z_0, u_1, z_1, \dots, u_{t+1}) \cdot \\
 &\quad P(x_{t+1} | z_0, u_1, z_1, \dots, u_{t+1}) \\
 &= \eta P(z_{t+1} | x_{t+1}) P(x_{t+1} | z_0, u_1, z_1, \dots, u_{t+1}) \\
 &= \eta P(z_{t+1} | x_{t+1}) \int P(x_{t+1} | z_0, u_1, z_1, \dots, u_{t+1}, x_t) \cdot \\
 &\quad P(x_t | z_0, u_1, z_1, \dots, u_{t+1}) dx_t \\
 &= \eta P(z_{t+1} | x_{t+1}) \int P(x_{t+1} | u_{t+1}, x_t) \cdot \\
 &\quad P(x_t | z_0, u_1, z_1, \dots, u_t, z_t, u_{t+1}) dx_t \\
 &= \eta P(z_{t+1} | x_{t+1}) \int P(x_{t+1} | u_{t+1}, x_t) \cdot \\
 &\quad P(x_t | z_0, u_1, z_1, \dots, u_t, z_t) dx_t \\
 &= \eta P(z_{t+1} | x_{t+1}) \underbrace{\int P(x_{t+1} | u_{t+1}, x_t) Bel(x_t) dx_t}_{= \overline{Bel}(x_{t+1})} , \quad (58)
 \end{aligned}$$

exploiting the definition, the Bayes rule, the Markov assumption, the law of total probability, and again the Markov assumption.

An algorithm corresponding to formula (58) is presented as Algorithm 2. A new action leads to the calculation of an a-priori estimate $\overline{Bel}$ by integration over all successor states of x_t according to the action model u_{t+1} . After a measurement the a-priori estimate is converted to the a-posteriori-estimate Bel .

Algorithm 2 Bayes filter. Input are an action u and a measurement z . The action leads to an a-priori estimate $\overline{Bel}(x_{t+1})$. Using the measurement z this is updated into an a-posteriori estimate $Bel(x_{t+1})$.

Require: Current belief $Bel(x_t)$ and the pair (u, z) .

Ensure: Updated Belief $Bel(x_{t+1})$

```

1: for all states  $x_t$  do {Action update}
2:    $\overline{Bel}(x_{t+1}) = \sum_{x_t} P(x_{t+1} | u, x_t) Bel(x_t)$ 
3: end for
4:  $\eta = 0$ 
5: for all states  $x_{t+1}$  do {measurement update}
6:    $Bel(x_{t+1}) = P(z | x_{t+1}) \overline{Bel}(x_{t+1})$ 
7:    $\eta = \eta + Bel(x_{t+1})$ 
8: end for
9: for all states  $x_{t+1}$  do {normalization}
10:   $Bel(x_{t+1}) = \eta^{-1} Bel(x_{t+1})$ 
11: end for
12: return  $Bel(x_{t+1})$ 
```

7.3 Kalman Filter and Extended Kalman Filter

The Kalman filter is an instantiation of the Bayes filter and is commonly used to fuse two independent sources of information. Robotic systems use it, for example, to fuse odometry data with external measurements. We will continue to refer to this as actions and measurements.

Given a multi-dimensional state estimation. The goals of the filter are to yield an estimate of the state of the system and to integrate different sensor measurements. In addition, the error models should be updated in an optimal way. These goals are reached by the following two steps:

1. **Prediction:** Predict the following state that will be obtained by executing the action/transition. The uncertainty of the state will be updated according to the action-error model (cf. Fig. 23). The action is executed.
2. **Filtering:** Measure the resulting state. After one of several measurements (which might contradict each other) calculate the most likely state based on the measurement and the prediction and error models. The error model is updated as well. Goto step 1.

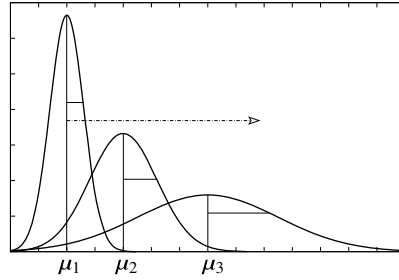


Fig. 23 Propagation of the state estimate without fusing of an independent measurement leads to a changed state (modified μ) according to the action, and an increase of the uncertainty (increasing σ)

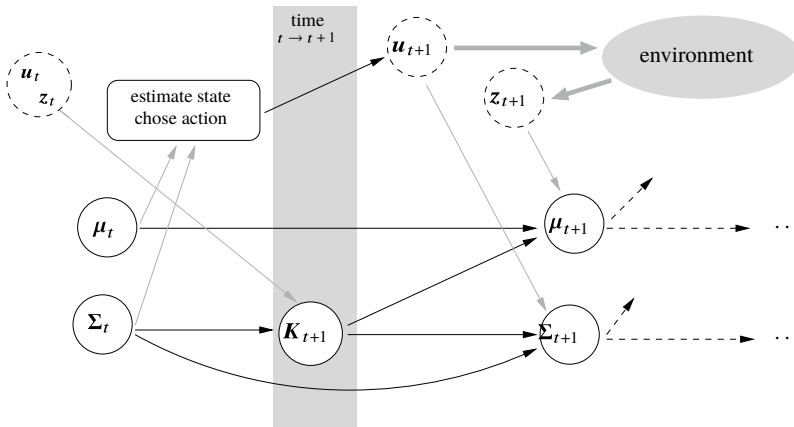


Fig. 24 Principle of the Kalman filter: estimation of the probability distribution $\mathcal{N}(\mu, \Sigma)$ over time with given actions u and measurements z

The combination of the steps is sketched in Fig. 24. Figure 25 gives an intuition of the process. The Kalman filter is subject to the following two issues:

- The state distributions are Gaussian distributions.
- The error model are Gaussian distributions.

By this, we can write a state in the following as $x \sim \mathcal{N}(\mu, \sigma)$ with a mean value μ of its estimation: according to the data μ the best estimation for the state x with an uncertainty σ . The same holds true for the n -dimensional case, i.e., $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$.

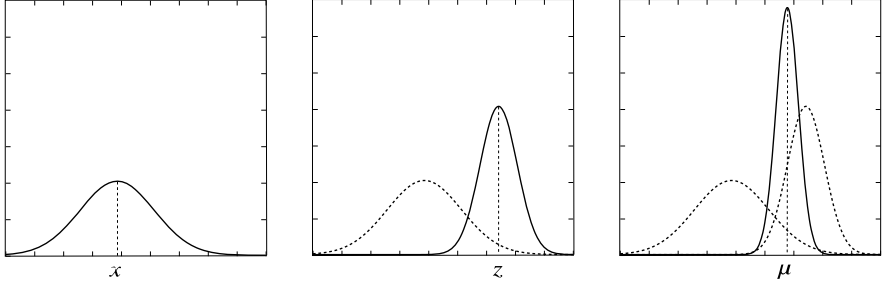


Fig. 25 Kalman filter with Gaussian distributions. Left: uncertainty in the current state x . Middle: an independent measurement z yields competing information (mean and variance). Right: fusion of both data yields a weighted average and a resulting variance with higher certainty in the filtered state

7.3.1 The n -Dimensional Kalman Filter

A multivariate Gaussian distribution is defined as

$$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}) = \alpha e^{-\frac{1}{2} \left((\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right)} . \quad (59)$$

The state $\mathbf{x}$ and the mean value $\boldsymbol{\mu}$ are n -dimensional vectors, $\boldsymbol{\Sigma}$ a $(n \times n)$ -dimensional covariance matrix. The probability of a successor state given the current state and the action, and the probability of a measurement in the current time step are given as well by their covariance matrices. The action, i.e., transition model and the sensor model are given as:

Action/Transition Model

$$P(\mathbf{x}_{t+1} | \mathbf{u}_t, \mathbf{x}_t) = \mathcal{N}(\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t, \boldsymbol{\Sigma}_u)(\mathbf{x}_{t+1}) , \quad (60)$$

Sensor Model

$$P(\mathbf{z}_t | \mathbf{x}_t) = \mathcal{N}(\mathbf{H}\mathbf{x}_t, \boldsymbol{\Sigma}_z)(\mathbf{z}_t) . \quad (61)$$

In general the state, the action and the measurement vector $(\mathbf{x}, \mathbf{u}, \mathbf{z})$ have different dimensionality. Let n be the dimension of the state, m the one of the action, and l the dimension of the measurement vector. Then we have the following quantities:

- A** transition model, $(n \times n)$ -matrix. $\mathbf{A}\mathbf{x}_t$ describes the transition from the state $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$, independent of an action, i.e., it describes what changes between two time steps. As it is independent of the action, it is often $\mathbf{A} = \mathbb{1}$.
- B** action model, $(n \times m)$ -matrix. $\mathbf{B}\mathbf{u}_t$ converts the (local) action $\mathbf{u}_t$ into the state space.

Σ_u covariance matrix for the action model, $(n \times n)$ matrix.

H sensor model, $(l \times n)$ -matrix, that converts the state into the space of measurements, such that the prediction becomes comparable to the measurement.

Σ_z covariance matrix for the sensor model, $(l \times l)$ -matrix.

mean und covariance for the state are initially $\mathbf{0}$ and given an action they are updated as:

$$\mu_{t+1} = A\mu_t + Bu_t + K_{t+1}(z_{t+1} - H(A\mu_t + Bu_t)) , \quad (62)$$

$$\Sigma_{t+1} = (\mathbb{I} - K_{t+1}H)(A\Sigma_t A^T + \Sigma_u) . \quad (63)$$

The Kalman gain is the $(n \times l)$ matrix K , that describes how an error made by the prediction is going into the new state:

$$K_{t+1} = (A\Sigma_t A^T + \Sigma_u)H^T (H(A\Sigma_t A^T + \Sigma_u)H^T + \Sigma_z)^{-1} . \quad (64)$$

The Eqs. (62)–(64) implement the Kalman filter. The terms are

$\mathbb{I}$	identity matrix, $(n \times n)$ -dimensional
$A\Sigma_t A^T + \Sigma_u$	a-priori prediction of the state covariance
$A\mu_t + Bu_t$	a-priori prediction of the mean/state
$H(A\mu_t + Bu_t)$	predicted sensor measurement in $(t + 1)$
$z_{t+1} - H(A\mu_t + Bu_t)$	difference between the sensor prediction and the actual measurement. This difference is often called <i>innovation</i> .

Please note, that the measurement z is not compared with the concurrent action u , but only with the predicted state that is the result of the previous state and the action. If the measurement consists of incremental information, the measurements have to be integrated/accumulated or related with the last state. Algorithm 3 implements the components of the Kalman filter. The current state x is the same as the mean μ . In this two-step implementation it is clear that the Kalman filter is a Bayes filter and we have the intuitive interpretation of the two phases:

Step 1, the update of the state over time, consists of a prediction of the a-priori state and the uncertainty $\bar{x}$, $\bar{\Sigma}$ based on the motion model. Step 2 corresponds to the update with a new measurement: The a-posteriori estimate K , x , Σ is corrected using the prediction from step 1. The Kalman gain K minimizes an a-posteriori error covariance matrix Σ . The a-posteriori estimate x is given as a linear combination of the a-priori estimate $\bar{x}$ and a weighted difference between the measurement z and the predicted measurement $\bar{z} = H\bar{x}$ and has the error covariance Σ . Continuous iteration of the equations given an initial estimate of the systems state x_0 and the Gaussian distributed estimate of the error Σ_0 in x_0 leads to optimal estimates of the successor states.

Algorithm 3 Kalman filter. Input is an action $\mathbf{u}$ as well as a percept $\mathbf{z}$. The action leads to an a-priori estimate $\overline{Bel}(\mathbf{x}_{t+1})$. Using the measurement $\mathbf{z}$ the a-posteriori estimate $Bel(\mathbf{x}_{t+1})$ is updated

Require: $\mathbf{x}_t$ current state, Σ_t current error model and the evidence pair $(\mathbf{u}, \mathbf{z})$

Ensure: the updated $\mathbf{x}_{t+1}$, Σ_{t+1}

- 1: *Prediction using the action $\mathbf{u}$*
 - 2: $\bar{\mathbf{x}}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}$
 - 3: $\bar{\Sigma}_{t+1} = \mathbf{A}\Sigma_t\mathbf{A}^T + \Sigma_u$
 - 4: *Correction using the measurement $\mathbf{z}$*
 - 5: $\mathbf{K}_{t+1} = \bar{\Sigma}_{t+1}\mathbf{H}^T(\mathbf{H}\bar{\Sigma}_{t+1}\mathbf{H}^T + \Sigma_z)^{-1}$
 - 6: $\mathbf{x}_{t+1} = \bar{\mathbf{x}}_{t+1} + \mathbf{K}_{t+1}(\mathbf{z} - \mathbf{H}\bar{\mathbf{x}}_{t+1})$ {state $\mathbf{x}_i \hat{=}$ mean μ_i }
 - 7: $\Sigma_{t+1} = (\mathbb{I} - \mathbf{K}_{t+1}\mathbf{H})\bar{\Sigma}_{t+1}$
 - 8: **return** $\mathbf{x}_{t+1}$, Σ_{t+1}
-

7.3.2 The Extended Kalman Filter (EKF)

As already noted, the Kalman filter is still applicable, if the optimality conditions are not met. For instance, if the errors are not Gaussian distributed. The calculations are then not provable optimal anymore, but still make sense. But, what happens in the case, where state transitions or dependencies are not linear anymore and cannot be described by matrices? For this case, there exists the extended Kalman filter, which approximates the non-linear models by Taylor approximations of first order (Jacobi matrices). For small sampling intervals this is quasi-optimal. The matrix multiplications are now substituted by the non-linear functions f and h (cf. Eqs. (65) and (66)). This allows general modeling of action and sensor information, i.e., non-linear transformation of the data.

To regard the non-linearities the prediction of the state is replaced by:

$$\bar{\mathbf{x}}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \quad (65)$$

and the predicted measurement by $h(\bar{\mathbf{x}}_t)$, i.e., the correction by:

$$\mathbf{x}_{t+1} = \bar{\mathbf{x}}_{t+1} + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - h(\bar{\mathbf{x}}_{t+1})) . \quad (66)$$

The above formulas and the algorithm are still valid, only the constant matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{H}$ are replaced by time-dependent versions $\mathbf{F}_t$, $\mathbf{H}_t$: These are the Jacobi matrices of the partial derivatives of f and g with respect to $\mathbf{x}$. Let $F^{[i,j]}$ the (i, j) -th entry of the vector $\mathbf{x}$ of the multi-dimensional function f (same for $\mathbf{H}$ and $\mathbf{h}$), then it holds:

$$\begin{aligned}
\mathbf{F}_{t+1} &= \left(F_{t+1}^{[i,j]} \right)_{i,j} = \frac{\partial f^{[i]}}{\partial x^{[j]}}(\mathbf{x}_t, \mathbf{u}_t) = \begin{pmatrix} \frac{\partial f^{[1]}}{\partial x^{[1]}} & \cdots & \frac{\partial f^{[1]}}{\partial x^{[n]}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f^{[n]}}{\partial x^{[1]}} & \cdots & \frac{\partial f^{[n]}}{\partial x^{[n]}} \end{pmatrix}, \\
\mathbf{H}_{t+1} &= \left(H_{t+1}^{[i,j]} \right)_{i,j} = \frac{\partial h^{[i]}}{\partial x^{[j]}}(\tilde{\mathbf{x}}_{t+1}) = \begin{pmatrix} \frac{\partial h^{[1]}}{\partial x^{[1]}} & \cdots & \frac{\partial h^{[1]}}{\partial x^{[n]}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h^{[n]}}{\partial x^{[1]}} & \cdots & \frac{\partial h^{[n]}}{\partial x^{[n]}} \end{pmatrix}.
\end{aligned} \tag{67}$$

Algorithm 4 summarizes the EKF analog to Algorithm 3 for the n -dimensional linear Kalman filter.

Algorithm 4 Extended Kalman filter

Require: $\mathbf{x}_t$ current state, Σ_t current error model, and the evience pair $(\mathbf{u}, \mathbf{z})$

Ensure: The updated $\mathbf{x}_{t+1}, \Sigma_{t+1}$

- 1: *Prediction using the action $\mathbf{u}$*
 - 2: $\tilde{\mathbf{x}}_{t+1} = f(\mathbf{x}_t, \mathbf{u})$
 - 3: $\tilde{\Sigma}_{t+1} = \mathbf{F}_{t+1} \Sigma_t \mathbf{F}_{t+1}^T + \Sigma_u$
 - 4: *Correction using the measurement $\mathbf{z}$*
 - 5: $\mathbf{K}_{t+1} = \tilde{\Sigma}_{t+1} \mathbf{H}_{t+1}^T (\mathbf{H}_{t+1} \tilde{\Sigma}_{t+1} \mathbf{H}_{t+1}^T + \Sigma_z)^{-1}$
 - 6: $\mathbf{x}_{t+1} = \tilde{\mathbf{x}}_{t+1} + \mathbf{K}_{t+1} (\mathbf{z} - h(\tilde{\mathbf{x}}_{t+1}))$ {state $\mathbf{x}_i \hat{=}_{\text{mea}} \mu_i$ }
 - 7: $\Sigma_{t+1} = (\mathbb{I} - \mathbf{K}_{t+1} \mathbf{H}_{t+1}) \tilde{\Sigma}_{t+1}$
 - 8: **return** $\mathbf{x}_{t+1}, \Sigma_{t+1}$ {unchanged, if $\mathbf{e} = \mathbf{u}$ }
-

8 Cross-References

This chapter contains cross-references to content published in other chapters belonging to the four “Robotic Goes MOOC” books: KNO5, KNO8, DES7. The full list of chapter abbreviations is available in the Preface.

References

1. M. Agrawal, K. Konolige, M.R. Blas, CenSurE: center surround extremas for realtime feature detection and matching, in *Proceedings of the European Conference on Computer Vision (ECCV'08)*. Lecture Notes in Computer Science (2008), pp. 102–115
2. K.S. Arun, T.S. Huang, S.D. Blostein, Least square fitting of two 3-D point sets. *IEEE Trans. Pattern Anal. Mach. Intell.* **9**(5), 698–700 (1987)
3. H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, Speeded-up robust features (surf). *Comput. Vis. Image Understand.* **110**(3), 346–359 (2008)
4. P. Besl, N. McKay, A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* **14**(2), 239–256 (1992)

5. M. Bleier, A. Nüchter, Low-cost 3D laser scanning in air or water using self-calibrating structured light, in *Proceedings of the 7th ISPRS International Workshop 3D-ARCH 2017: "3D Virtual Reconstruction and Visualization of Complex Architectures"*. ISPRS Archives Photogrammetry and Remote Sensing Spatial Information Sciences, Nafplio, vol. XLII/W3 (2017), pp. 105–112
6. G. Bradski, A. Kaehler, *Learning OpenCV, Computer Vision with OpenCV Library*, 1st edn. (O'Reilly Media, Sebastopol, 2008)
7. M. Calonder, V. Lepetit, C. Strecha, P. Fua, Brief: binary robust independent elementary features, in *Proceedings of the 11th European Conference on Computer Vision (ECCV'10)* (2010), pp. 778–792
8. H. Ekbart, G. Schönfelder (eds.), *Sensoren in Wissenschaft und Technik* (Vieweg + Teubner Verlag, Wiesbaden, 2012)
9. J. Elseberg, D. Borrmann, A. Nüchter, Full wave analysis in 3D laser scans for vegetation detection in urban environments, in *Proceedings of the XXIII International Symposium on Information, Communication and Automation Technologies (ICAT'11), Sarajevo* (IEEE, Piscataway, 2011)
10. FARO Technologies, Inc., <http://www.faro.com> (2017)
11. O. Faugeras, What can be seen in three dimensions with an uncalibrated stereo rig, in *Proceedings of the IEEE European Conference on Computer Vision (ECCV '92)*. Lecture Notes in Computer Science, vol. 588 (1992), pp. 563–578
12. M.A. Fischler, R.C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* **24**, 381–395 (1981)
13. A. Flint, A. Dick, A.J. van den Hengel, Thrift: local 3D structure recognition, in *Proceedings of the 9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications (DICTA'07)* (2007), pp. 182–188
14. M.H. Freeman, C.C. Hull, *Optics*, 11th edn. (Buterworth-Heinemann, Edinburgh, 2003)
15. R. Furukawa, H. Kawasaki, Laser range scanner based on self-calibration techniques using coplanarities and metric constraints. *Comput. Vis. Image Understand.* **113**(11), 1118–1129 (2009)
16. J. Garcia, Z. Zalevsky, Range mapping using speckle decorrelation. US Patent 7,433,024, October 7 (2008)
17. R. Hartley, Estimation of relative camera positions for uncalibrated cameras, in *Proceedings of the IEEE European Conference on Computer Vision (ECCV'92)*. Lecture Notes in Computer Science, vol. 588 (1992), pp. 579–587
18. E. Hecht, *Optics*, 5th edn. (Pearson, Boston, 2017)
19. K. Konolige, A. Nüchter, Range sensors, in *Springer Handbook of Robotics*, ed. by O. Khatib, B. Siciliano, chap. 31, 2nd edn. (Springer, Berlin, 2016), pp. 783–810
20. D.G. Lowe, Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.* **60**, 91–110 (2004). <https://doi.org/10.1023/B:VISI.0000029664.99615.94>
21. M. Mettenleiter, F. Härtl, S. Kresser, C. Fröhlich, *Laserscanning*. Die Bibliothek der Technik, vol. 371 (verlag moderne industrie, Munich, 2015)
22. C.R. Nave, Hyperphysics (2017). <http://hyperphysics.phy-astr.gsu.edu>
23. A. Nüchter, *3D Robotic Mapping: The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*. Springer Tracts in Advanced Robotics, vol. 52 (Springer, Berlin, 2009)
24. A. Nüchter, J. Elseberg, P. Schneider, D. Paulus, Study of parameterizations for the rigid body transformations of the scan registration problem. *J. Comput. Vis. Image Understand.* **114**(8), 963–980 (2010)
25. K. Pathak, D. Borrmann, J. Elseberg, N. Vaskevicius, A. Birk, A. Nüchter, Evaluation of the robustness of planar-patches based 3D-registration using marker-based ground-truth in an outdoor urban scenario, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10) Taipei* (2010), pp. 5725–5730
26. M. Pollefeys, Visual 3d modeling from images. Technical Report, University of North Carolina, Chapel Hill (2004)

27. Riegl Laser Measurement Systems, RiSCAN PRO. Software Description & User's Instruction, Version 1-5-0sp1 (2009)
28. Riegl Laser Measurement Systems (2017). <http://riegl.com/nc/products/terrestrial-scanning/>
29. Riegl Laser Measurement Systems, Data sheet, rieg1 VZ-400 (2017). <http://riegl.com/nc/products/terrestrial-scanning/>
30. E. Rosten, T. Drummond, Machine learning for high-speed corner detection, in *Proceedings of the European Conference on Computer Vision (ECCV'06)* (2006), pp. 430–443
31. E. Rublee, V. Rabaud, K. Konolige, G. Bradski, *ORB: An Efficient Alternative to SIFT or SURF*. (IEEE, Piscataway, 2011), pp. 2564–2571.
32. R.B. Rusu, N. Blodow, M. Beetz, Fast point feature histograms (FPFH) for 3D registration, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'09)*, Kobe (2009), pp. 1848–1853
33. R.B. Rusu, N. Blodow, Z.C. Marton, M. Beetz, Aligning point cloud views using persistent feature histograms, in *Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'09)*, Nice (2009), pp. 3384–3391
34. P. Torr, P. Beardsley, D. Murray, Robust vision, in *Proceedings of the British Machine Vision Conference (BMVC'94)* (1994)
35. J. Weng, P. Cohen, M. Herniou, Camera calibration with distortion models and accuracy evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.* **14**(10), 965–980 (1992)
36. S. Winkelbach, S. Molkenstruck, F.M. Wahl, Low-cost laser range scanner and fast surface registration approach, in *Proceedings of Pattern Recognition: 28th DAGM Symposium, Berlin, September 12–14* (Springer, Berlin, 2006), pp. 718–728
37. L. Zagorchev, A. Goshtasby, A paintbrush laser range scanner. *Comput. Vis. Image Underst.* **101**(2), 65–86 (2006)
38. Z. Zhang, A flexible new technique for camera calibration. *Trans. Pattern Anal. Mach. Intell.* **22**(11), 1330–1334 (2000)
39. Z. Zhang, R. Deriche, O. Faugeras, Q.T. Luong, A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artif. Intell. J.* **78**, 87–119 (1995)

Control and Motion Planning



Lars Berscheid and Torsten Kröger

Abstract Concepts and algorithms for robot motion *control* are based on feedback control schemes that keep robots on their desired position or force trajectories. Going beyond foundational concepts like PID control in joint space, we explain how to design algorithmic concepts that are able to compensate for robot model errors and environmental uncertainties. For applications requiring high precision, high velocities, high stiffness, and high repeatability, position control methods are optimized for various use cases that we will discuss.

Using application-specific input signals, robot path and trajectory *planning* algorithms provide set points for underlying motion controllers. We will learn how to design real-time path and trajectory planning algorithms that can be used in hybrid switched-system controllers or for human-robot collaboration applications. With the goal of planning more complex paths and trajectories, sampling-based methods, optimal planning concepts, and reinforcement learning algorithms are explained.

1 Introduction

In this chapter, we consider two tightly related topics: *robot control* and *robot motion planning*. Both topics are very foundational areas that are highly relevant for all robotic application domains: The *Impact* Chapters 25–30 describe examples including industrial robots, medical robots, aerial robots, space robots, underwater robots, rescue robots, and many more. They are also the foundation for robot *force control* (Chap. 4) and *visual control* (Chap. 5). With the goal of providing

L. Berscheid
Jacobi Robotics Inc., Emeryville, CA, USA
e-mail: berscheid@jacobirobotics.com

T. Kröger (✉)
Intrinsic Innovation LLC, Mountain View, CA, USA
e-mail: t@kroe.org

generalizable concepts that can be adapted and deployed in all applications domains, we will exemplarily focus on control and motion planning for *robot arms*.

We take the definition of the term “robot” specified by the *International Organisation for Standardisation* [1] as the basis: “An automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications.” This definition is also accepted by the *International Federation of Robotics (IFR)* [2] and by the *European Robotics Research Network (euRobotics aisbl)* [3].

Still, even industrial robot arms solve a great variety of tasks: For example, they pick-and-place objects in logistic applications, weld in industrial production lines, support heavy-duty tasks in human-robot collaborations, and expand into flexible assembly, disassembly and recycling. Unsurprisingly, robots solve these tasks in the real world by planning and executing *motions*. While numerous control and motion planning algorithms have been developed in the last decades, we introduce the most fundamental concepts furthermore. In short, this chapter is about letting robots *move*.

2 Kinematics and Dynamics

Before diving into the core concepts of robot control and motion planning, let us give a brief overview of the kinematics and dynamics of robot motions first. The geometry of the 3D *Cartesian space* is the basis for this. Here, a *pose* is given by a *position and orientation*. In homogeneous coordinates, a pose $\mathbf{T} \in SE(3)$ can be expressed by a 4×4 matrix

$$\mathbf{T}_a^b = \begin{pmatrix} \mathbf{R}_a^b & \vec{t} \\ 0 & 1 \end{pmatrix} \quad (1)$$

with the 3D rotation matrix $\mathbf{R}_a^b$ and a translation vector $\vec{t} = (x, y, z)^T$. Using a simple matrix multiplication, the pose (1) transforms a point $\mathbf{p}^a \in \mathbb{R}^3$ expressed in the coordinate frame a to a frame b according to $\mathbf{p}^b = \mathbf{T}_a^b \mathbf{p}^a$. Despite the 12 free parameters of the homogeneous matrix, a general pose $\mathbf{T}$ has six degrees of freedom (DoF): Three for the spatial translation and rotation each. A more compact parametrization of orientations are three-angle representations, as arbitrary rotations can be expressed as successive rotations around different coordinate axis (known as the Euler’s rotation theorem). These *Euler angles* $\vec{r} = (\alpha, \beta, \gamma)^T$ can be applied to a number of different coordinate axis combinations, so it is important to keep in mind which convention is used exactly. In the domain of robot arms, Euler angles are usually given in the *XYZ* convention representing a successive rotation around the Z , Y , and X coordinate axes. Mobile robots commonly use the *ZYX* convention, also known as roll-pitch-yaw angles.

To represent the six DoFs of a homogeneous transformation $\mathbf{T}$ as a vector with position $\vec{r}$ and orientation $\vec{r}$, we write

$$\vec{x} = \left(\vec{r}^T, \vec{r}^T \right)^T, \quad (2)$$

whereas $\vec{r}$ may also be represented by Cartesian, cylindrical, or spherical coordinates, and $\vec{r}$ may be represented by Euler angles or quaternions.

2.1 Motion in Time and Space

A *path* is a geometric representation that connects a start and a goal pose $\vec{x}_{start}$ and $\vec{x}_{goal}$. Let $s \in [0, 1]$ be the relative position (also known as the path parameter) on a path $\vec{f}$ with

$$\vec{f}(s) : [0, 1] \mapsto SE(3) \quad \text{and} \quad \vec{f}(s = 0) = \vec{x}_{start}, \quad \vec{f}(s = 1) = \vec{x}_{goal}.$$

Of course, a path $\vec{f}(s)$ needs to be continuous to allow a real-world motion. In most cases, there is an infinite number of paths between the start and end pose. Then an *optimal* path should be found minimizing a pre-defined metric. Commonly, the path length L

$$L = \int_0^1 \sqrt{\left(\frac{\partial \vec{f}}{\partial s} \right)^2} ds$$

is minimized by a straight line path $\vec{f}(s) = \vec{x}_{start} + s(\vec{x}_{goal} - \vec{x}_{start})$. However, the boundaries as well as obstacles of the robot's workspace might constraint possible paths. In fact, many algorithms in motion planning deal with the subtask of finding a collision-free path among a collection of static and dynamic obstacles.

A *trajectory* is more than a path: The trajectory $\vec{g}(t)$ depends on the time t and therefore also includes timing information along a path $\vec{f}$:

$$\vec{g}(t) = \vec{f}(s(t)) \quad \text{with} \quad t \in [0, T] \quad (3)$$

throughout the trajectory duration T . The mapping $s(t)$ is called the time parametrization of the path. Implicitly, the trajectory $\vec{g}(t)$ includes the time derivatives velocity $\vec{v}$, acceleration $\vec{a}$, and jerk $\vec{j}$

$$\vec{v}(t) = \frac{\partial \vec{g}(t)}{\partial t}, \quad \vec{a}(t) = \frac{\partial \vec{v}(t)}{\partial t}, \quad \text{and} \quad \vec{j}(t) = \frac{\partial \vec{a}(t)}{\partial t}.$$

The trajectory position and its three derivatives are summarized as the *kinematic state* at a given point of time. While higher derivatives of course exist, they are rarely relevant. Every real-world trajectory is limited by kinematic or dynamic constraints of the robot, its environment, or the task. A common task is to find the time parametrization $s(t)$ for a priori specified paths, which again optimize for a certain criterion (e.g., a minimum trajectory duration T). For other problems however, the trajectory $\vec{g}(t)$ can be optimized directly without calculating an intermediate path representation first.

2.2 Forward and Inverse Kinematics

A robot arm can be mathematically represented by serial links of n joints. Each joint applies a transformation $\mathbf{T}_i^{i+1}$ on the prior link i . These transformations are partially parameterizable to represent the robot's actuation, but also partially fixed to account for the robot's mechanical structure. A common convention are the (modified) Denavit-Hartenberg parameters leading to the transformation

$$\begin{aligned} \mathbf{T}_{i-1}^i &= \mathbf{T}_{i-1}^i(\alpha_{i-1}, a_{i-1}, \theta_i, d_i) \\ &= \begin{pmatrix} \cos \theta_i & -\sin \theta_i & 0 & a_{i-1} \\ \sin \theta_i \cos \alpha_{i-1} & \cos \theta_i \cos \alpha_{i-1} & -\sin \alpha_{i-1} & -d_i \sin \alpha_{i-1} \\ \sin \theta_i \sin \alpha_{i-1} & \cos \theta_i \sin \alpha_{i-1} & \cos \alpha_{i-1} & d_i \cos \alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

We consider robot arms with revolute joints furthermore: Here, the joint angle θ_i is actuated, while the distance d_i , the offset a_{i-1} , and angle α_{i-1} are inherent to the robot. We refer to the i -th actuated parameter as the joint position q_i . The vector of all joint positions $\vec{q} = (q_1, q_2, \dots, q_n)^T \in \mathbb{R}^n$ is element of the so-called *joint space*. Note that the joint position takes all DoFs into account; in contrast, the Cartesian position does only include the three translational DOFs.

To calculate the pose of the end-effector $\mathbf{T}_{ee}$ (commonly referred to as the robot's pose), all transformations can be applied iteratively

$$\mathbf{T}_{ee} = \prod_{i=1}^n \mathbf{T}_i^{i+1}$$

relative to a base frame $\mathbf{T}_0$. Furthermore, the six dimensional representation $\vec{x}_{ee}$ is extracted from the transformation $\mathbf{T}_{ee}$ following Eq. (2). The function

$$\vec{x}_{ee} = \vec{h}(\vec{q}) \quad (4)$$

mapping the joint coordinates to the robot's pose is called the *forward kinematics* and can be calculated easily for serial kinematic chains. The *Jacobian* $\mathbf{J} \in \mathbb{R}^{6 \times n}$ is the derivative of the end-effector pose with respect to the joint coordinates

$$\mathbf{J}(\vec{q}) = \frac{\partial \vec{x}_{ee}}{\partial \vec{q}}. \quad (5)$$

In particular, the Jacobian transforms the joint velocity $\dot{\vec{q}}$ to the Cartesian velocity (twist) via $\dot{\vec{x}} = \mathbf{J}(\vec{q})\dot{\vec{q}}$. The *inverse kinematics*

$$\vec{q} = \vec{h}^{-1}(\vec{x}_{ee}),$$

however is much more complex than Eq. (4), as it calculates a joint configuration $\vec{q}$ reaching a known end-effector pose $\vec{x}_{ee}$. For example, there might be multiple solutions or no solution at all. For $n \leq 6$, there are up to 16 distinct solutions $\vec{q}$ which can be calculated by a closed-form analytical form [4]. While the derivation can become complex for the general case, certain conditions simplify the resulting equations significantly. For example, the last three rotational axis of industrial robots often intercept in a single point (known as a *spherical wrist*). For $n > 6$, an infinite number of solutions might exist. Here, a numerical approach is preferred that minimizes the difference

$$\vec{q}^* = \arg \min_{\vec{q}} \|\vec{h}(\vec{q}) - \vec{x}_{ee}\| \quad (6)$$

between the desired end-effector pose and the forward kinematics of the resulting solution $\vec{q}^*$. An iterative approach converges towards the solution by stepping

$$\vec{q}^{j+1} = \vec{q}^j - \lambda \mathbf{J}^{-1}(\vec{q}^j) \left(\vec{h}(\vec{q}^j) - \vec{x}_{ee} \right)$$

along the difference's gradient with step-size λ . As the inverse of the Jacobian, as defined by Eq. (5), does not exist in general for $n \neq 6$, a *pseudo-inverse* is required instead. This is usually caused by additional DoFs $n > 6$ that allow to move the robot's joints while keeping the end-effector pose fixed and fulfilling Eq. (6). This subspace of the joint space is called the *null space* of the robot. Numerical approaches allow to utilize this null space more flexibly, for example by integrating secondary optimization criteria on top of Eq. (6). Chapter 10 introduces these *redundant robots* in detail.

2.3 Rigid-Body Equations of Motion

From a mechanical point of view, a robot (as a serial link of joints) can be analyzed as a multi-body system. The Lagrangian formulation of mechanics is well-suited to

study the *dynamics* of such systems. Let the Lagrangian $L = T - U$ be a scalar function with the total kinematic energy T and potential energy U of the system. The joint positions $\vec{q}$ of the robot arm's kinematics correspond to the generalized coordinates. Then, Lagrange's equation for each coordinate gives

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau_i$$

with the (generalized) torques $\vec{\tau}$. For a complete derivation of the dynamics equations, please refer to [5] or [6]. The resulting equation can be written as

$$\sum_{j=1}^n M_{ij} \ddot{q}_j + \sum_{j=1}^n \sum_{k=1}^n C_{ijk} \dot{q}_j \dot{q}_k + \tau_{gi} = \tau_i$$

with the so-called Christoffel symbols of the first type given by

$$C_{ijk} = \frac{1}{2} \left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} + \frac{\partial M_{jk}}{\partial q_i} \right)$$

and the torques caused by gravity $\vec{\tau}_g$. This can be expressed in matrix-notation as

$$\mathbf{M}(\vec{q}) \ddot{\vec{q}} + \mathbf{C}(\vec{q}, \dot{\vec{q}}) \dot{\vec{q}} + \vec{G}(\vec{q}) = \vec{\tau} \quad (7)$$

with the inertia matrix $\mathbf{M}(\vec{q})$, the Coriolis matrix $\mathbf{C}(\vec{q}, \dot{\vec{q}})$ (including centrifugal effects), the joint torques generated by the gravitational force $\vec{G}(\vec{q})$, and the torques $\vec{\tau}$. The latter torques $\vec{\tau} = \vec{\tau}_m + \vec{\tau}_{ext}$ are composed of internal motor torques $\vec{\tau}_m$ and external torques $\vec{\tau}_{ext}$ acting on the robot. External torques can be linked to Cartesian forces $\vec{F}$ via

$$\vec{\tau}_{ext} = \mathbf{J}(\vec{q})^T \vec{F}_{ext},$$

the transposed Jacobian. To be precise, the so-called *wrench* $\vec{F}$ combines translational forces $\vec{f}$ and rotary moments (torques) $\vec{m}$

$$\vec{F} = \left(\vec{f}^T, \vec{m}^T \right)^T$$

along the coordinate axis defined in Eq. (2).

Equation (7) is a general *dynamical model* of a robot describing the development of the joint position $\vec{q}$ over time. This is an ideal and simplified model of a robot; a hard-to-model friction term $\mathbf{B}(\vec{q}) \dot{\vec{q}}$ can improve the accuracy significantly.

3 Robot Control

We will first focus on the robot's lower-level control that will allow it to follow a desired motion robustly even under disturbances. In this regard, the vast majority of deployed robot arms are position controlled industrial robots with six revolute joints (cf. Chap. 25). As an emerging and fast-growing field, torque-controlled robot arms with joint torque sensors also play an important role, especially for applications that require physical interactions between humans and robots (cf. Chap. 21). Hence, we will address the underlying concepts of robots with (1) *position* controlled joints and (2) *torque* controlled joints. In both cases, we assume that the joint actuators are electrical motors (e.g., DC motors, synchronous AC motors, or asynchronous AC motors). Conceptually, the here-presented methods can also be applied to redundant robots (cf. Chap. 10), robots with prismatic joints, and robots with different actuators (e.g., hydraulic or pneumatic actuators).

3.1 Motor Control

Depending on the hardware, robotic actuators take a desired motor torque $\bar{\tau}_{m,d}$, a desired motor velocity $\bar{\dot{q}}_{m,d}$, a desired motor position $\bar{q}_{m,d}$, or a combination of these three (e.g., motor velocity with feed-forward torque control) as input. The standard CiA 402 [7] defines all physically possible motor control modes and profiles. CiA 402 is supported by all major real-time field bus systems (e.g., EtherCAT, Ethernet POWERLINK, ProfiNet, EtherNet/IP [8–11] and servo drive vendors, so that this standard serves as a basis for defining software interfaces for motor control.

In most cases, gears are used between the motor axes and the joint axes, so that the gear ratio is taken into account to transform between motor coordinates and joint coordinates. Sometimes, there is a coupling between two or more joints; for example two motors may drive two joints. For the dynamic model (Eq. (7)), gears usually introduce *friction* that is challenging to model (cf. $\mathbf{B}(\vec{q})\dot{\vec{q}}$).

3.2 Joint Position Control

Let us consider a robot arm with n joints as sketched in Fig. 1. The arm's state of motion at time t is measured by joint position sensors like encoders or resolvers and is represented by the *actual* joint position vector

$$\vec{q}_a(t) = (q_{a,1}(t), q_{a,2}(t), \dots, q_{a,n}(t))^T. \quad (8)$$

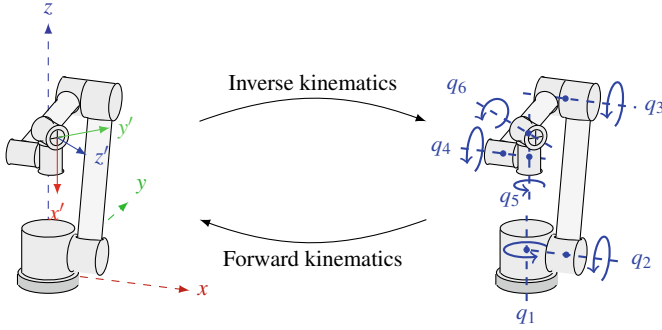


Fig. 1 A robot with $n = 6$ revolute joints. The end-effector pose is given relative to a base frame in Cartesian space (left), while the joint space spans the position of all joint actuators (right). The forward and inverse kinematics calculate in between the Cartesian and joint space

The motion of the robot arm is defined by a *desired* joint trajectory $\vec{q}_d(t)$ with overall duration T . The task of robot joint position controllers is to execute and follow a given desired joint trajectory $\vec{q}_d(t)$ while minimizing the joint position control error

$$\vec{q}_e(t) = |\vec{q}_d(t) - \vec{q}(t)|. \quad (9)$$

The controller calculates motor commands such as desired motor torques $\vec{\tau}_{m,d}$. In particular, the robot arm acts within an environment that can influence the arm's motion in a (generally) unpredictable way. Additionally, initial deviations, model uncertainties, or even numerical drift in the control software necessitate an error feedback to maintain the desired trajectory $\vec{q}_d(t)$. Figure 2 shows an overview of this basic joint position control scheme. The feedback loop and new motor commands are calculated at a high frequency called the *control rate*, typically in the range of 100–2000 Hz.

A *PID controller* is a simple yet very common feedback controller: It calculates the desired torques based on a proportional, integral, and derivative term of the joint position control error $\vec{q}_e(t)$.

$$\vec{\tau}_{m,d} = \mathbf{K}_p \vec{q}_e + \mathbf{K}_i \int \vec{q}_e(t) dt + \mathbf{K}_d \dot{\vec{q}}_e \quad (10)$$

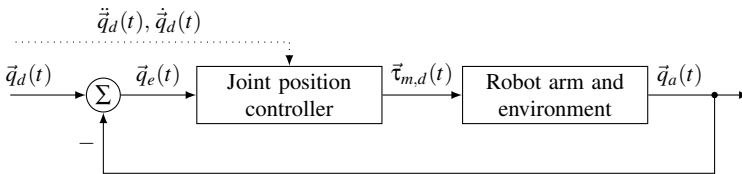


Fig. 2 Basic structure of a joint position control scheme for robot arms

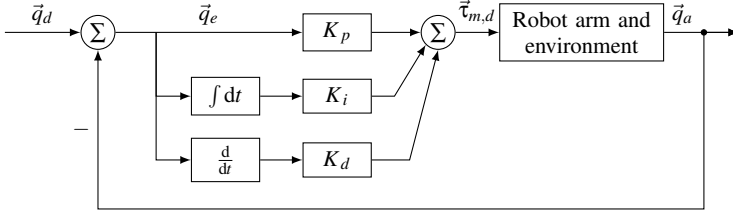


Fig. 3 A PID control scheme for basic robot torque control

By tuning the control gains $\mathbf{K}_p$, $\mathbf{K}_i$, and $\mathbf{K}_d$, the error can be minimized effectively over time. In short, the proportional term pushes the control towards a zero state equilibrium, the derivative term dampens the change of control regarding overshooting, and the integral term eliminates steady-state errors. We refer to standard control engineering literature for fundamentals and general stability analysis of PID controllers [5, 12, 13]. For the application within the robotics domain, Fig. 3 shows the block diagram of a robot controlled via a PID feedback. We want to briefly discuss the stability of a PD controlled robot. Inserting a PD control equation (10) into the general motion equation (7) yields

$$\mathbf{M}(\vec{q}) \ddot{\vec{q}} + \mathbf{C}(\vec{q}, \dot{\vec{q}}) \dot{\vec{q}} + \vec{G}(\vec{q}) = \mathbf{K}_p \vec{q}_e + \mathbf{K}_d \dot{\vec{q}}_e$$

with $\mathbf{K}_i = 0$. Furthermore, we simplify the system to a single DoF in a horizontal plane without gravity $\vec{G}(\vec{q}) = 0$ and around a set-point with zero desired velocity and acceleration. Then, $\dot{q}_d = 0$ and $\ddot{q}_d = 0$ result in $\dot{q} = -\dot{q}_e$ and $\ddot{q} = -\ddot{q}_e$.

$$\ddot{q}_e + \frac{(C + K_d)}{M} \dot{q}_e + \frac{K_p}{M} q_e = 0$$

This differential equation describes a common mass-spring-damper system. Converting into the standard second-order form yields $\ddot{q}_e + 2\zeta\omega_0\dot{q}_e + \omega_0^2 q_e = 0$ with the natural frequency ω_0 and the damping ratio ζ

$$\omega_0 = \sqrt{\frac{K_p}{M}} \quad \text{and} \quad \zeta = \frac{C + K_d}{2\sqrt{K_p M}}.$$

Without further input, the system would oscillate around the desired position q_d in a frequency of ω_0 . Over time, the damping decreases the amplitude until the robot comes to rest and the error q_e vanishes. Classical control theory states that this system is stable if $C + K_d > 0$ and $K_p > 0$. In general, a high K_p is preferred for a fast response and error correction. Then, K_d should be chosen according to $\zeta = 1$, so that the damping is time-optimal. For very large K_p and K_d however, torque violations of the motors or mechanical vibrations due to a limited control rate might occur.

3.3 Computed Torque Control

Feedback controllers need to wait for errors in the first place to detect and correct them afterwards. Instead of this passive approach, the controller can use the robot's dynamic model to calculate the required torques in a *foresighted* manner. This results in smaller tracking errors, in particular for dynamic motions. Combining the equation of motion (7), the error definition (9), and a PID controller yields

$$\vec{\tau} = \mathbf{M}(\vec{q}) \left(\ddot{\vec{q}}_d + \mathbf{K}_p \vec{q}_e + \mathbf{K}_i \int \vec{q}_e(t) dt + \mathbf{K}_d \dot{\vec{q}}_e \right) + \mathbf{C}(\vec{q}, \dot{\vec{q}}) \dot{\vec{q}} + \vec{G}(\vec{q}). \quad (11)$$

Note that this PID controller acts on the error of the joint acceleration

$$\ddot{\vec{q}}_e + \mathbf{K}_p \vec{q}_e + \mathbf{K}_i \int \vec{q}_e(t) dt + \mathbf{K}_d \dot{\vec{q}}_e = 0$$

instead of the torques as previously. The feedback controller outputs a joint acceleration $\ddot{\vec{q}}_{fb}$. The desired joint acceleration $\ddot{\vec{q}}_d$ generates *feed-forward* torques, while the PID controller adds a *feedback* term on top. If the robot model would be perfect, so that Eq. (7) would hold true irrevocably, a feedback controller would not be necessary. However due to inevitable modelling errors, feedback is required to keep the error $\vec{q}_e$ small. Figure 4 shows the block diagram of this computed torque controller.

As introduced, variables such as the joint error $\vec{q}_e$ are vectors corresponding to multiple DoFs. In particular, it leads to the gains $\mathbf{K}$ being matrices. Diagonal matrices are usually a good tradeoff between parameter tuning and controller stability. Moreover, the calculation of the inertia matrix $\mathbf{M}(\vec{q})$ as well as the Coriolis matrix $\mathbf{C}(\vec{q}, \dot{\vec{q}})$ can become too computationally expensive for calculating at very high control rates. Then, the model might be simplified even further for small desired velocities and accelerations. In the limit, the model is approximated by the robot's gravitational forces alone:

$$\vec{\tau} = \ddot{\vec{q}}_d + \mathbf{K}_p \vec{q}_e + \mathbf{K}_i \int \vec{q}_e(t) dt + \mathbf{K}_d \dot{\vec{q}}_e + \vec{G}(\vec{q}).$$

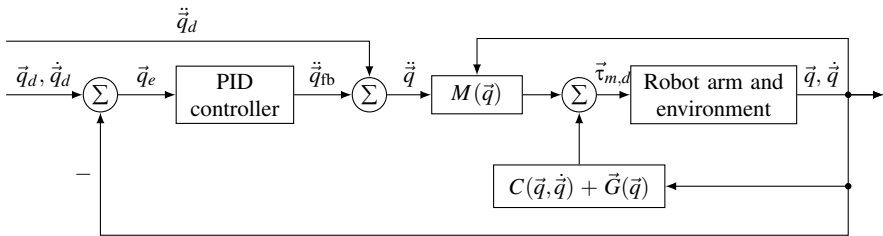


Fig. 4 Computed torque control. Basic structure of a joint position control scheme for robot arms

This controller is commonly used as a starting point for practical implementations, as it easily allows for gravity compensation.

3.4 Task Space Control

So far, the proposed control algorithms have been derived in the joint space of the robot. For many applications however, it is beneficial to reformulate the equations in the task (or operational) Cartesian space. Based on the equation of motion from Eq. (7), we replace the joint by their respective task coordinates

$$\dot{\vec{x}} = \mathbf{J}(\vec{q})\dot{\vec{q}} \quad \text{and} \quad \ddot{\vec{x}} = \dot{\mathbf{J}}(\vec{q})\dot{\vec{q}} + \mathbf{J}(\vec{q})\ddot{\vec{q}}$$

using the Jacobian for transformation. The Cartesian equations of motion

$$\Lambda(\vec{x})\ddot{\vec{x}} + \mu(\vec{x}, \dot{\vec{x}})\dot{\vec{x}} + \vec{p}(\vec{x}) = \vec{F}$$

can be derived with the inertia matrix Λ , Coriolis matrix μ , gravity vector $\vec{p}$, and the external wrench $\vec{F}$. The Cartesian control parameters are related to their counterparts in joint space by

$$\begin{aligned} \Lambda(\vec{x}) &= \mathbf{J}(\vec{q})^{-T} \mathbf{M}(\vec{q}) \mathbf{J}(\vec{q})^{-1}, \\ \mu(\vec{x}, \dot{\vec{x}}) &= \mathbf{J}(\vec{q})^{-T} \mathbf{C}(\vec{q}, \dot{\vec{q}}) - \Lambda(\vec{q}) \dot{\mathbf{J}}(\vec{q}) \dot{\vec{q}}, \\ \vec{p}(\vec{x}) &= \mathbf{J}(\vec{q})^{-T} \vec{G}(\vec{q}). \end{aligned}$$

Cartesian control algorithms can be derived correspondingly to the joint space. As the joint space is quite non-intuitive for human operators, controlling the robot in Cartesian space allows for more predictable control reactions. Moreover, it allows to have different control strategies along meaningful DoFs (like global coordinate axes). For example, it allows to have a different control strategy for the z -axis (e.g., for pressing something on the table) than in the xy -plane (e.g., for the wiping motion). Task space control requires the Jacobian $\mathbf{J}(\vec{q})$ to be invertible, which is not the case at robotic *singularities*.

3.5 Force Control

For many applications, a desired trajectory is not enough to solve a given task. Instead of only controlling the pose, the robot needs to control the force it is applying to the environment as well. Chapter 4 is devoted exclusively to the area of *force control*, including control strategies like impedance and admittance control.

4 Robot Motion Planning

So far, we have outlined how a robot is able to follow a *desired* trajectory as closely as possible. In comparison, *finding* a trajectory in the first place is very much task-specific: Besides moving from pose $\mathbf{T}_A$ to $\mathbf{T}_B$ directly, this might include avoiding known obstacles, reacting to dynamic obstacles along the way, following a path with a fixed velocity, complying with dynamic constraints, or even applying a force onto the environment. Due to this variety of tasks, a wide range of *interfaces* for letting a human operator define the robot's motion and intended behaviour have emerged. Nonetheless, the most essential problem is moving a robot from a start configuration to a goal subject to:

1. **Kinematic constraints** of position, velocity, acceleration, and optionally jerk.
2. **Dynamic constraints** due to the physical limits of the joint actuators (e.g., maximum motor torques).
3. **Obstacle avoidance** to ensure a safe robotic motion. This includes preventing robotic self-collisions between different links.

In many applications, the resulting trajectory is optimized for minimum duration. A guarantee for time-optimality can usually only be given for simplified problem definitions.

In particular, we differentiate between *online* vs. *offline* motion planning algorithms: Offline approaches plan the trajectory beforehand, that is, before the robot starts executing the trajectory. In comparison, online algorithms are either designed (1) for low-latency sense-plan-act application or (2) for meeting deterministic time and/or result requirements (i.e., real-time capable algorithms). The latter allow (at best) to be run within the same control loop of the robot controller (cf. Sect. 3) and enables instantaneous adaption at run time, for example, for improvement of accuracy, better utilization of currently available dynamics, reaction to and interaction with dynamic environments, or reaction to sensor events. Moreover, online approaches should handle arbitrary initial conditions to be used anytime, or offer a fallback strategy at least.

4.1 Trajectory Planning

Let us first consider a *waypoint-based* interface that defines a single or multiple kinematic states a robot should reach subsequently. Here, the first waypoint refers to the start, while the last waypoint refers to the goal of a path or path segment. Commonly, the first and last states are fully defined by their position, velocity, and acceleration. Intermediate waypoints however might only be defined by their position alone. It is the task of the trajectory planning algorithm to find suitable velocities and accelerations.

Let $\vec{q}(t)$ be a trajectory of duration T with fully-defined initial and goal states

$$\begin{aligned}\vec{q}(0) &= \vec{q}_0, & \dot{\vec{q}}(0) &= \dot{\vec{q}}_0, & \ddot{\vec{q}}(0) &= \ddot{\vec{q}}_0, \\ \vec{q}(T) &= \vec{q}_T, & \dot{\vec{q}}(T) &= \dot{\vec{q}}_T, & \ddot{\vec{q}}(T) &= \ddot{\vec{q}}_T,\end{aligned}$$

and an optional number of m via points defined only by their position

$$\vec{q}(T_j) = \vec{q}_j \quad \text{with} \quad 1 \leq j \leq m \quad \text{and} \quad 0 \leq T_j \leq T_{j+1} \leq T.$$

We use this notation for joint space trajectories and Cartesian space trajectories. In general, we want to find the trajectory

$$\vec{q}^*(t) = \arg \min_{\vec{q}(t)} T$$

minimizing the overall duration of the motion. Simple approaches of trajectory planning usually neglect dynamic constraints and consider kinematic constraints alone. In this regard, obstacle avoidance is also outsourced, as we assume the waypoints to be suitable for a collision-free trajectory.

4.1.1 Motion Profiles

When no intermediate via points are given, the robot moves from a fully-defined initial to a fully-defined target state. While this only allows simple trajectories, these point-to-point motions are very essential building blocks for real-world applications. In particular, *motion profiles* allow a time-optimal trajectory by utilizing the full kinematic capability of the robot: The underlying idea is to always accelerate (or decelerate) along the maximum kinematic limit to reach the highest possible velocity. This approach results in a so-called *bang-bang* profile. Depending on the initial and target state, different limits might be reached, leading to a finite set of possible profile types.

Let us first consider the case of constrained velocity v and acceleration a for a single DoF. Then, the trajectory should fulfill

$$\forall t \in [0, T], \forall i \in (0, \dots, n) : v_i^{\min} \leq \dot{q}_i(t) \leq v_i^{\max} \text{ and } a_i^{\min} \leq \ddot{q}_i(t) \leq a_i^{\max}. \quad (12)$$

All elements of $\vec{v}^{\min}$ and $\vec{a}^{\min}$ are negative and unequal to zero. For a single DoF i , we want to exemplarily analyze the case with initial and goal velocities of zero as well as $q_{T,i} > q_{0,i}$. Then, a time-optimal profile has three steps: First, the maximum acceleration $a_i^{\max}$ is applied until the maximum velocity $v_i^{\max}$ is reached. To stop at the desired position $q_{T,i}$, the profile needs to decelerate with minimum acceleration $a_i^{\min}$ beforehand. Due to its velocity graph over time, this is called a *trapezoidal*

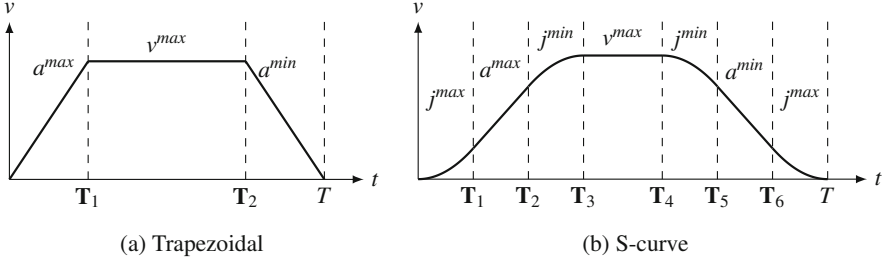


Fig. 5 Motion profiles of constrained velocity, acceleration (a), and jerk (b)

profile (cf. Fig. 5a). The respective durations $t_{j,i}$ of each step j and each DoF i are given by

$$t_{1,i} = \frac{v_i^{max}}{a_i^{max}}, \quad t_{2,i} = \frac{q_{T,i} - q_{0,i}}{v_i^{max}} - \frac{v_i^{max}}{a_i^{max}}, \quad t_{3,i} = \frac{v_i^{max}}{a_i^{max}}.$$

However, the maximum velocity v_i^{max} might not be reached if the goal position is too close to the initial position. In this case, the time for step 2, $t_{2,i}$, would become negative. Neglecting $t_{2,i}$ yields

$$t_{1,i} = \sqrt{\frac{q_{T,i} - q_{0,i}}{a_i^{max}}}, \quad t_{3,i} = \sqrt{\frac{q_{T,i} - q_{0,i}}{a_i^{max}}} \quad \text{if} \quad q_{T,i} - q_{0,i} < \frac{(v_i^{max})^2}{a_i^{max}}.$$

The trapezoidal profile generates infinite jerk at the switching points between maximum, zero, and minimum acceleration. This results in vibrations of the mechanical system that could lead to wear-and-tear or even to damage of the robot. To generate *smoother* motions, a constraint of the jerk $j_i^{min} \leq \ddot{q}_i(t) \leq j_i^{max}$ is usually added besides Eq. (12). By definition, j_i^{min} is negative, and j_i^{max} is positive for all $i \in (0, \dots, n)$.

Then, the time-optimal profile uses steps of constant jerk instead of constant acceleration: The so-called *S-curve profile* has seven steps of maximum, minimum, or zero jerk (cf. Fig. 5b). First, it accelerates with maximum jerk until the maximum acceleration is reached. Before reaching the maximum velocity however, the profile decelerates with minimum jerk to ensure zero acceleration for constant maximum velocity. There are three possible constraints, leading to eight cases that must be taken into account.

In the same manner, motion profiles can be extended to work with arbitrary non-zero initial and goal states. When dealing with multiple DoFs, and in particular with non-zero goal velocities, all DoFs have to be synchronized to reach the goal state at the same point of time. In most cases, all but one DoF need be slowed down for the constraining DoF, which is the task of *time scaling*. However, in some cases duration intervals without possible trajectory exists, which need to be avoided for the synchronized duration. For jerk-constrained trajectories, time scaling can become quite a costly calculation, as roots of polynomials of up to sixth degree needs to be

solved. To allow online trajectory generation, this needs to be implemented with real-time capable approaches like bisection methods. The correct motion profile type can be found by using decision trees [14] or by calculating and rejecting all but the optimal one [15].

4.1.2 Via Point Trajectories

Via points allow to define more complex motions, for example, to avoid obstacles or to follow a route. If the complete state including intermediate velocities and accelerations is known, we can simply concatenate time-optimal motion profiles. However, if only intermediate positions are defined, corresponding velocities and accelerations need to be calculated first. Therefore, generating a time-optimal trajectory breaks down into finding two scalar values per via point and DoF. In general, this is a non-convex optimization problem that has a variety of approaches. For example, Ruckig [15] uses a gradient-based optimization of the intermediate motion profiles to minimize the overall trajectory duration.

It is much more common to split the motion planning problem into a *path planning* and a subsequent *time scaling* (also *time parameterization*) step. Analogous to Eq. (3), the time scaling $s(t)$ maps each time t to an abstract path position $s : [0, T] \mapsto [0, 1]$ yielding

$$\vec{q}(t) = \vec{f}(s(t)) \quad \text{with} \quad s \in [0, 1].$$

To allow a continuous and smooth motion with non-zero velocity, the path $\vec{f}$ itself needs to be twice or three times differentiable. Paths either interpolate or approximate via points: The former reaches them exactly, while the latter only follows them roughly within a certain distance.

Figure 6 shows different path algorithms for a 2D case. *Cubic splines* interpolate a path with cubic polynomials per intermediate section. In contrast, *B-splines* only guarantee to be within the convex hull of the via points. In robotics, *linear segments with polynomial blends* (LSPB) or circular blends are widespread, as the resulting path is easily predictable. A parameter setting the maximal distance to a linear interpolation adjusts the smoothness of the path approximation.

4.1.3 Time Parametrization

Given a path $\vec{f}(s)$, we now want to find the time-optimal scaling $s(t)$ that fulfills constraints on velocity and acceleration. Depending on the path parameter s , they are given for each DoF i by

$$-v_i^{max} \leq f'_i(s)\dot{s} \leq v_i^{max}, \quad (13)$$

$$-a_i^{max} \leq f'_i(s)\ddot{s} + f''_i(s)\dot{s}^2 \leq a_i^{max}, \quad (14)$$

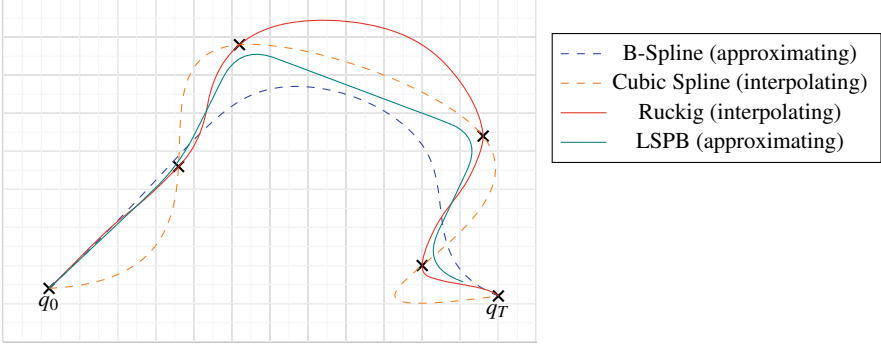


Fig. 6 Path planning for via points in a 2D example: B-Spline approximation (red), Spline interpolation (orange), linear segments with polynomial blending (green), and Ruckig with joint path and trajectory generation (red)

similar to the constraint Eq. (12) for motion profiles. As the time scaling can be arbitrarily complex (or even non-expressible in a closed form), time parametrization usually refers to a numeric approach. Here, a number of reference points are selected and analyzed. For example, they can be chosen at a fixed distance with interpolation to ensure a smooth control output in between. In the following, we assume a numerical integration with the control rate. Time parametrization is usually computational expensive and not real-time capable, although performance improvements were achieved in recent years [16, 17].

The path position is monotonically increasing, so that the path velocity $\dot{s}(t) \geq 0$ is non-negative for all times t . There exists a maximum allowed path velocity called the velocity *limit curve* $V(s)$. Amongst others, Eq. (13) constraints the path velocity to be below

$$\dot{s}^{vel}(s) = \min_i \frac{v_i^{max}}{|f'_i(s)|} \quad (15)$$

Similarly, the path acceleration $\ddot{s}$ is constrained by both an upper and lower acceleration limit curve $L(s, \dot{s}) \leq \ddot{s} \leq U(s, \dot{s})$. Transposing Eq. (14) to get $\ddot{s}$ (here only for $f'(s) \neq 0$) gives

$$L(s, \dot{s}) = \max_i \left(\frac{-a_i^{max}}{|f'_i(s)|} - \frac{f''_i(s)\dot{s}^2}{f'_i(s)} \right), \quad (16)$$

$$U(s, \dot{s}) = \min_i \left(\frac{a_i^{max}}{|f'_i(s)|} - \frac{f''_i(s)\dot{s}^2}{f'_i(s)} \right). \quad (17)$$

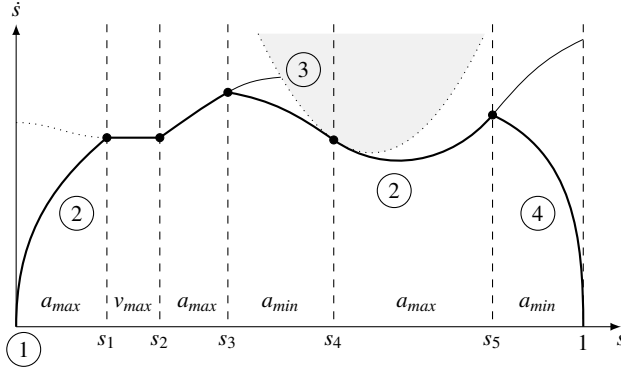


Fig. 7 Calculating a time-optimal time parametrization in the $(s, \dot{s})$ phase plane. The trajectory (thick) switches between maximum, zero, or minimum acceleration in one DoF (dashed), to comply with the velocity limit curve (dotted)

Of course, a feasible motion requires $L(s, \dot{s}) \leq U(s, \dot{s})$. Solving Eqs. (16) and (17) for $\dot{s}$ yields the maximum allowed path velocity

$$\dot{s}^{acc}(s) = \min_{i,j} \sqrt{\frac{\frac{a_i^{max}}{|f'_i(s)|} + \frac{a_j^{max}}{|f'_j(s)|}}{\left| \frac{f''_i(s)}{f'_i(s)} - \frac{f''_j(s)}{f'_j(s)} \right|}}. \quad (18)$$

The velocity limit curve is the minimum $V(s) = \min\{\dot{s}^{vel}(s), \dot{s}^{acc}(s)\}$. The $(s, \dot{s})$ phase plane is a common visualization for the time parameterization problem (cf. Fig. 7). A simplified algorithm to introduce the underlying idea is as following:

1. Start at $(s, \dot{s}) = (0, 0)$.
2. Integrate forward with maximum acceleration $U(s, \dot{s})$ until either: (a) $s > 1$, then set $(s, \dot{s}) = (1, 0)$ and go to Step 4. (b) the velocity limit curve is hit with $\dot{s} > V(s)$, then go to Step 3.
3. Find the first point s_{tan} on the velocity limit curve so that the lower acceleration curve $L(s_{tan}, \dot{s}_{tan})$ does not penetrate the limit curve. In particular, the necessary condition

$$\frac{d}{ds} V(s) = \frac{L(s, V(s))}{\dot{s}}$$

can be derived based on Eqs. (15)–(18). Starting from $(s_{tan}, \dot{s}_{tan})$, integrate backwards with minimum acceleration until the start trajectory from Step 2 is hit. The intersection is a switching point from maximum to minimum acceleration. In comparison, the tangent point $(s_{tan}, \dot{s}_{tan})$ is a switching point from minimum

to maximum acceleration. Then, continue the forward integration from there according to Step 2.

4. Integrate backwards with minimum acceleration $L(s, \dot{s})$ until the trajectory is crossed. The intersection is a switching point from maximum to minimum acceleration.

Similar to trapezoidal profiles in Sect. 4.1.1, this time-parameterization algorithm generates infinite jerks at the acceleration switching points. Generalizing this approach to jerk-constraint motions has proven to be difficult: Usually, the result is smoothed after a time-optimal trajectory regarding velocity and acceleration constraints has been found.

4.2 Graph-Based Planning

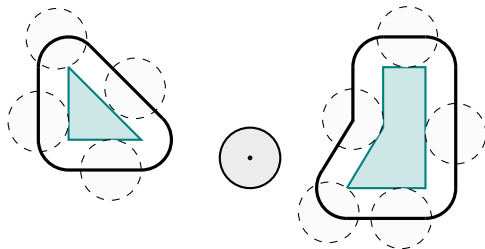
The robot is now able to follow a pre-defined path or move along a list of via points. These approaches allow human operators to control the robot as desired for a wide range of tasks. If, however, the robot is to behave more autonomously, then a new higher-level interface becomes necessary: A core problem of motion planning is finding these paths or via points based on a start and a goal pose as well as a *model of the environment*.

4.2.1 Configuration Space Obstacles

Section 2.1 defines a path to be within the real-world Cartesian space, while the forward and inverse kinematics allow to switch between the joint and Cartesian space flexibly. As the joint parameters fully determine the robot's configuration (or shape), the joint space is also known as the *configuration space* [18]. While moving within its physical environment, the robot needs to avoid collisions between obstacles and its own extent. In the configuration space however, the robot can be seen as a single point that is, at every arbitrary position, either in collision or not (Fig. 8). The complete configuration space $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$ is therefore the sum of a free subspace and obstacles. Joint or workspace limits can also be mapped to obstacles in the configuration space, defining a basis for motion planning problems. Let the sub-space *in collision* consist of a number of compact obstacles $\mathcal{C}_{obs} = \bigcup_i \mathcal{C}_{obs,i}$. In particular, we are interested in a distance metric d between the robot $\vec{q}$ and an obstacle i with

$$\begin{aligned} d(\vec{q}, \mathcal{C}_{obs,i}) &> 0 && \text{is contact-free,} \\ d(\vec{q}, \mathcal{C}_{obs,i}) &= 0 && \text{is in contact,} \\ d(\vec{q}, \mathcal{C}_{obs,i}) &< 0 && \text{is penetrating.} \end{aligned}$$

Fig. 8 A 2D projection of a robot's end-effector (gray) and environment obstacles (blue). The configuration space obstacles (black) correspond to the real-world obstacles dilated by the robot's shape



Detecting collisions based on a 3D model of the robot and its environment is nontrivial, and a range of algorithms and libraries exist. Predominantly, obstacles are approximated in a conservative manner or subdivided into convex objects to allow for efficient calculation [19].

Discretizations of the configuration space can be represented by a weighted graph in the following manner. The robot is able to move between two nodes of the graph, resulting in an optimization cost corresponding to the weight of the travelled edge. The total cost of the path is the sum of all travelled edges. Each node might be an admissible state of the robot, or a simple position in case of path planning. The structure of a weighted graph allows to reduce the number of possible paths to a calculable number of solutions, so that optimality guarantees can be given.

4.2.2 A* Algorithm

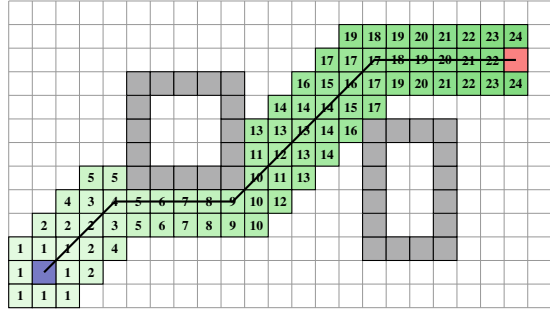
A* is a popular algorithm for graph search, as it allows to efficiently find a path on a weighted graph. The foundational idea is to build up a *tree of paths* from the start node until the goal node is reached. Let $c(s, n)$ be the cost of the path from the start s to a current node n , and $h(n, g)$ be a heuristic for estimating the cost from the current node n to the goal g . The algorithm extends its tree iteratively by adding the node n'

$$n' = \arg \min_n \{c(s, n) + h(n, g)\} \quad (19)$$

that minimizes the estimated total cost of the path. If the heuristic h underestimates the cost, then A* is guaranteed to return an optimal path. In the domain of path planning, the straight-line distance is a lower-bound of the distance between two nodes. We briefly introduce an actual implementation of the A* algorithm furthermore. Let

- OPEN be a set of nodes, initialized with the start node s ,
- PARENT a map linking a node to its precursor node on the shortest path,
- COST be the minimum cost $c(s, n)$ for each node n that is currently known, and
- EST TOTAL COST the current guess of the total cost $c(s, n) + h(n, g)$ of a path from start to goal through n .

Fig. 9 The A^* algorithm on a 2D grid with obstacles (gray) and the shortest path from the start (blue) to goal (red) node. Each visited node shows its cost (green)



At each iterative step, the node within OPEN with the lowest EST TOTAL COST is referred to as the **current node**. If the OPEN set is empty, there exists no path between the start and the goal. If the **current node** is the goal, the algorithm has finished. Else, remove **current** from OPEN. For each of its adjunctive **neighbor** nodes, calculate the sum of the weight of the corresponding edge and the COST of the **current node**. If the neighbor was not visited yet, or the calculate sum is less then the known COST, we have found a new path to the **neighbor** node. In this case, update the COST and EST TOTAL COST of the **neighbor** and add it to the OPEN set. Also, save this path by setting the **current node** as the PARENT of the **neighbor**. At the end, the optimal path is reconstructed by following the PARENT nodes from the goal back to the start node.

A grid is a special form of a graph, and one of the most simple discretization of a continuous configuration space. Each cell of the grid has a fixed number of neighbours, and the edge weight is usually constant. For 2D grids, there are typically 4 or 8 neighbors (depending on whether diagonal transitions are allowed). Figure 9 shows a path found by the A^* algorithm in a 2D grid with obstacles. For high dimensional or large problems however, A^* becomes exponentially more expensive, in particular as the cost of all visited nodes needs to be hold in memory.

4.3 Sampling-Based Planning

In comparison to graph-based planning algorithms, sampling-based approaches work directly in the continuous configuration space that robots operate in. Instead of using a fixed set, the planner *samples* nodes to explore the configuration space iteratively. Most sampling-based planners (and in particular those introduced below) are probabilistically complete: With increasing number and density of nodes, the probability of finding one (or an optimal) solution converges to 100%. Sampling based approaches scale well to high-dimensional configuration spaces: The number of required samples is usually much smaller in comparison to the cells of grid planner scaling exponentially with the DoFs.

4.3.1 RRT Algorithm

A *rapidly-exploring random tree* is a structure on which a family of popular sampling algorithms is based on [20]. Similar to the A^* algorithm, RRT builds up a tree T to find a path from the start node q_{start} . The goal state is defined by a set Q_{goal} , usually as a small environment around the true goal pose q_{goal} . In contrast to A^* , RRT is able to add nodes to its underlying graph, removing the primary limitation of the graph discretization. The basic RRT algorithm can be summarized as:

1. Initialize the tree T with the start node q_{start} .
2. Sample a node q_{rand} from the free configuration space $\mathcal{C}_{free}$.
3. Find the nearest node q_{near} within the tree T to q_{rand} minimizing a distance metric $d(q_{near}, q_{rand})$.
4. Create a new node q_{new} from q_{near} in the direction of q_{rand} . Usually, q_{new} is clipped at a maximum distance Δq of q_{near} .
5. If the new node q_{new} and the corresponding edge (q_{near}, q_{new}) are collision-free, add them to the tree T .
6. If $q_{new} \in Q_{goal}$, a path has been found. Else, repeat with Step 2.

Step 2 explores the free configuration space, and does so completely assuming an unlimited number of iterations and an appropriate sampling strategy. Usually, nodes are sampled uniform randomly except of a small probability of choosing the goal state. Despite the algorithm's name of a *random tree*, this step can also be deterministic as long as the samples become dense in the configuration space. Step 4 restricts the exploration to be in small steps and pulls the tree into unvisited territory. In general, this increases the probability that the path in between is obstacle free. Still, many implementations of step 4 depend on the actual use-case. In simple cases, q_{new} lies on a straight line between q_{near} and q_{rand} . For more complex cases however, a local planner might be used to integrate additional path constraints and limit q_{new} accordingly. With each iteration step, the tree T expands and fills into the free space until a node within the goal set has been found (cf. Fig. 10). Then, the final path is generated by traversing the tree backwards.

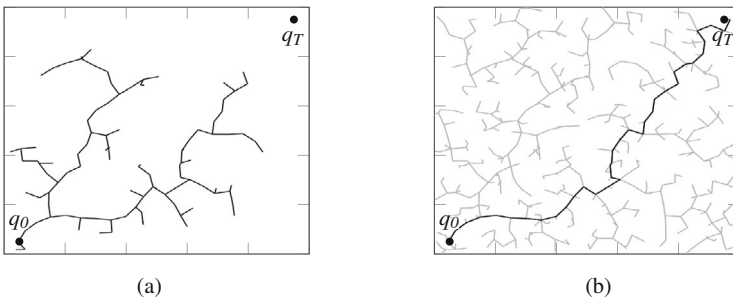


Fig. 10 The exploration of a free space by the RRT algorithm. (a) Initial exploration (100 steps). (b) Solution found (500 steps)

In contrast, the *Bidirectional RRT* algorithm is able to find an exact goal state. It explores the environment alternately using a distinct forward and backward tree. The primary task is then to connect both trees; which is oftentimes faster than finding a single node within a goal set.

4.3.2 RRT*

The introduced RRT algorithm stops and returns as soon as a path is found. *RRT** allows to refine a path until a termination criteria is met [21]. For example, *RRT** returns the path with minimal cost found after a pre-defined calculation duration. Therefore, it adapts Step (5) of the original RRT algorithm: First, let Q_{near} be a set of nodes around q_{new} . Instead of the nearest node q_{near} , the edge (q', q_{new}) is added with node $q' \in Q_{near}$ that minimizes the total cost of the path all the way from q_{start} to q_{new} . Second, *RRT** checks if a node $q \in Q_{near}$ can be reached through q_{new} at a lower cost. If so, the nodes are rearranged accordingly in the tree T . This way, *RRT** combines the cost optimization of the A* algorithms with the sample-based RRT algorithm.

4.4 Potential Fields

Potential fields are a class of algorithms that allow real-time capable motion planning and generation. While previous algorithms like A* or RRT return a complete path from start to goal, potential fields consider only the current state within the environment. The underlying idea is strikingly simple: The robot moves through its configuration space inspired by a physical particle in a potential field. While commonly used equations resemble the *elastic potential* (e.g., found in springs), the idea can be imagined easily like a ball rolling through the gravitational field of a hilly landscape until it comes to rest at the lowest point.

The trajectory is shaped by a potential $P(\vec{q})$ with its minimum at $\vec{q}_{goal}$. It is created by a quadratic potential K for the goal-targeted behavior, superimposed by a number of obstacle potentials

$$P(\vec{q}) = \frac{1}{2}(\vec{q} - \vec{q}_{goal})^T K (\vec{q} - \vec{q}_{goal}) + \sum_i P_{O_i}(\vec{q}).$$

Each obstacle P_O defines a positive potential that repels the robot based on the distance $d(\vec{q}, O)$ in between. A typical obstacle potential is of the form

$$P_O(\vec{q}) = \frac{k}{2d(\vec{q}, O)^2}$$

with a scaling factor k . Analog to a physical potential, the robot is controlled by the resulting force of the potential. This force $\vec{F}(\vec{q})$ can be calculated by the negative gradient of the potential

$$\vec{F}(\vec{q}) = -\frac{\partial P(\vec{q})}{\partial \vec{q}} = K(\vec{q}_{goal} - \vec{q}) + \sum_i \frac{k}{d(\vec{q}, O_i)^3} \frac{\partial d(\vec{q}, O_i)}{\partial \vec{q}}.$$

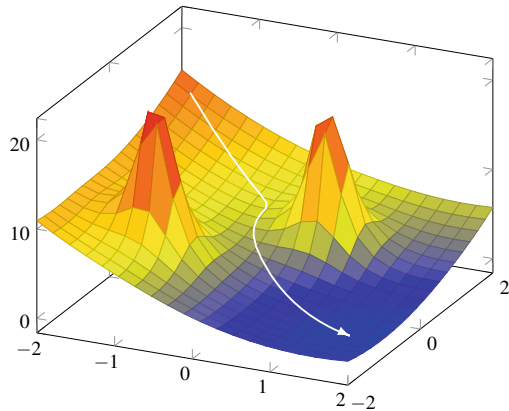
The robot can then be controlled via Newton's law $\ddot{\vec{q}} = \frac{\vec{F}(\vec{q})}{m}$ with a virtual mass m . As this however leads to overshooting the goal state $\vec{q}_{goal}$, a damping term B is introduced. This results in the typical control law

$$\ddot{\vec{q}} = \frac{\vec{F}(\vec{q})}{m} - B\dot{\vec{q}}$$

corresponding to a ball moving in a potential field with rolling friction. The overall approach shares many similarities to minimizing the potential $P(\vec{q})$ via gradient descent with step size m . In contrast, the complete *trajectory* is of interest here (Fig. 11).

There are two major limitations of potential fields: First, local minima might cause traps preventing the robot from reaching its goal. Research has found guarantees for potential fields without local minima, restricting the obstacles (or their potentials) to certain geometries known as a *sphere* or *star world*. Second, potential fields do not consider explicit kinematic constraints. Obstacles with high scaling factor k might induce an acceleration exceeding the robotics limits. Still, k has a lower limit as the obstacle potential needs to repulse a robot even moving with maximum velocity, so that a collision-free path is guaranteed for any initial state.

Fig. 11 The potential field of a motion planning problem with two obstacles and the calculated trajectory (white)



4.5 Optimization Algorithms

All the approaches so far have tried to make the motion planning problem tractable. From an abstract point of view, trajectory planning is an optimization problem that can, to a certain degree, be solved by mathematical solvers. In general, we want to find the optimal trajectory $\vec{q}^*$ minimizing a loss function J

$$\begin{aligned}\vec{q}^*(t) &= \arg \min_{\vec{q}(t), T} J(\vec{q}(t), T), \\ \vec{q}(t) &\in \mathcal{C}_{free}, \quad \forall t \in [0, T], \\ \vec{q}(0) &= \vec{q}_{start}, \quad \vec{q}(T) = \vec{q}_{goal}.\end{aligned}$$

This allows for very flexible constraints that couldn't be considered so far, in particular *dynamic constraints*. Arbitrary equality and inequality constraints $\vec{h}$ and $\vec{g}$

$$\vec{h}(\vec{q}(t)) = 0 \quad \text{and} \quad \vec{g}(\vec{q}(t)) \leq 0, \quad \forall t \in [0, T]$$

can be appended to the problem definition. Similar to the approaches introduced before, this notation holds true only when each DoF is controllable. As solvers are limited to optimize numeric values in form of a parameter vector $\vec{x}$, a suitable trajectory representation $\vec{x} \mapsto \vec{q}(t)$ needs to be found first.

4.5.1 Sequential Quadratic Programming

Quadratic Programming (QP) is the basis of many optimization algorithms, as very efficient mathematical solvers exist. Here, the loss function and the constraints are given by

$$\arg \min_{\vec{x}} \quad \vec{x}^T \mathbf{Q} \vec{x} + \vec{c}^T \vec{x} \quad \text{subject to} \quad \mathbf{H} \vec{x} = \vec{a}, \quad \mathbf{G} \vec{x} < \vec{b}$$

with respective quadratic or linear transformations of the parameters $\vec{x}$. We want to discuss an exemplary use-cases for optimization in motion planning: Finding the shortest collision-free path between two poses $\vec{q}_{start}$ and $\vec{q}_{end}$. Here, the path $\vec{q}(s)$ is defined by a linear interpolation of a number of intermediate waypoints $\vec{q}_i$. The parametrisation

$$\vec{x} = \left[(\vec{q}_1 - \vec{q}_0)^T, (\vec{q}_2 - \vec{q}_1)^T, \dots, (\vec{q}_n - \vec{q}_{n-1})^T \right]^T$$

allows to define the (quadratic) path length objective as simple as

$$\sum_i \|\vec{q}_{i+1} - \vec{q}_i\|^2 = \vec{x}^T \vec{x}.$$

The path should keep a minimal distance $d(\vec{q}(s_j), \mathcal{C}_{obs}) > d_{min}$ at a number of test points $\vec{q}(s_j)$ to obstacles $\mathcal{C}_{obs}$. However, distance constraints, in particular involving the forward kinematics, or other general constraints are non-linear. Therefore, the constraints are linearised at the current solution $\vec{x}_t$ to iteratively solve the QP to calculate an updated solution $\vec{x}_{t+1}$. This method is called *sequential quadratic programming*. Due to the linearised constraints, the updates should be kept small, for example by trust region approaches [22]. While trajectory optimization is usually able to converge to a solution quickly, it is easily susceptible by the initial solution $\vec{x}_0$. In contrast to sampling-based planners, trajectory optimization acts locally and is not able to find a global optimum.

4.5.2 Reinforcement Learning

In recent years, *Reinforcement Learning* (RL) has gained traction for problems regarding trajectory generation and motion planning. While Chap. 6 will focus solely on robot *learning*, we will give a short introduction to solving motion [23]. First, let a Markov Decision Process be defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, p_0, r)$ with a state $s \in \mathcal{S}$, an action $a \in \mathcal{A}$, a transition model $\mathcal{T}$, the initial configuration p_0 , and a reward function r . At a discrete time step t , the transition model T maps an action a_t in a given state s_t to a new state probability $p(s_{t+1})$. The pre-defined reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ maps an action in a given state to a single scalar. The goal of an agent is to maximize the discounted sum of collected rewards over time

$$R = \sum_i^N \gamma^i r(s_i, a_i)$$

with the (possibly infinite) episode length N and a so-called discount factor $\gamma < 1$. The solution of a RL problem is a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ mapping a state s_t to an action a_t at every time step t . This policy is learnt in a data-driven manner: The agent needs to interact and explore the environment (given in form of the state space and transition model) with the reward as its only feedback. In this regard, RL is a very general optimization algorithm, however without any model information. Unsurprisingly, RL requires a lot of interaction between the agent and its environment, making the training process slow and costly.

Simulation allows to scale and accelerate the training significantly in comparison to real-world interaction. The *transition model* is then given by a physics simulation of a pre-defined environmental and robotic model. Due to model uncertainties and algorithmic as well as numerical simplifications, a *sim-to-real* gap between simulation and reality remains in particular for the robot's dynamics.

For motion planning, the *state space* is usually the common configuration space. Current approaches however differ in their *action spaces*: Torques or accelerations within the control loop allow the most low-level real-time control of the robot. This flexibility comes at the cost of more required training samples. Moreover, kinematic

or dynamic constraints needs to be considered explicitly [24]. Alternatively, an action can be defined by waypoints approached with time-optimal motion profiles. This allows to decouple the higher-level waypoint calculation with RL into a non-real-time loop with lower frequency.

5 Software and Robot Application

In the previous sections, several algorithms for different aspects and subsystems for controlling a robot have been introduced. *Integrating* all these algorithms and their interfaces into a single robot is just as important. To highlight a programmer's point of view furthermore, we focus on an exemplary robotic task in the context of human-robot collaboration: A robot should move its end-effector within a shared environment with a human (cf. Fig. 12). The human's position and motion needs to be observed throughout to let the robot react in an online manner, while still trying to accomplish the underlying task. In this example, the robot should move from *start* to *goal* along two defined via points. First, a collision-free motion without human interruption should be generated. This motion is a best-case scenario and can be prepared offline.

1. The goal state of the motion including velocity and acceleration needs to be defined. In this example, we assume an initial and goal state at rest (with zero velocity and acceleration). While the initial pose is determined by the current robot pose, the goal pose is usually task-specific, for example, a grasp pose or welding spot. Given this Cartesian pose in Euler angles, the inverse kinematics calculates a corresponding joint position (cf. Sect. 2.2). Software libraries like *KDL* or *ikfast* take the URDF file or Denavit-Hartenberg parameters as input. We assume a robot with seven DoFs having a null-space with a single DoF. To

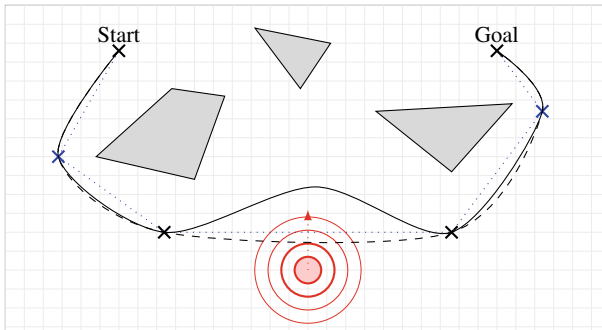
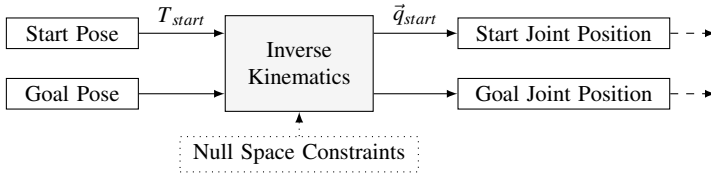
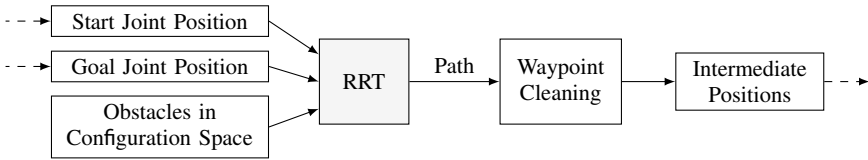


Fig. 12 A bird's eye view on the human-robot collaboration task. The robot should move from *Start* to *Goal*, avoiding the obstacles (gray), and react to a human (red) entering its path. The task is specified by the four waypoints (black)

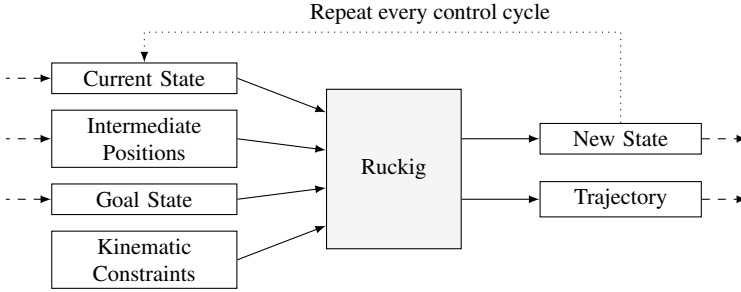
maximize the distance of the goal joint position to the robot's joint limits, we use a numerical inverse kinematics solver to specify additional optimization criteria.



2. The real-world obstacles needs to be transferred into an *environment model* within the robot's configuration space. For many industrial tasks, the model comes for free from the workspace design. For simpler cases, the obstacles can be approximated with geometric primitives like polygons. For static yet unknown obstacles, sensors can provide occupancy data, for example, for a volumetric voxel grid.
3. The *RRT* algorithm (cf. Sect. 4.3.1) finds a collision-free path between two poses. It is applied consecutively to all waypoints to find the entire path from *start* to *goal*. RRT requires a set of hyperparameters, including the probability of sampling the final state or the step size for new nodes.

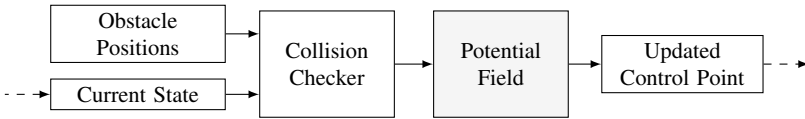


4. The RRT algorithm as described in Sect. 4.3.1 returns a path with linear interpolation between nodes. Oftentimes, these nodes needs to be filtered to ensure a smooth and fast motion towards the goal. For example, the filter algorithm might neglect as many nodes as possible while keeping a safety distance to obstacles.
5. The *Ruckig* library generates a motion considering the kinematic capabilities of the robot, taking the start and goal state as well as the filtered via points as input. Ruckig is capable of *online* trajectory generation: Then, a new state is returned after a fixed calculation time so that it can be used within the real-time control loop. The interface will take the updated current state as input for the next iteration, so that it can detect unexpected changes and plan accordingly. For now, the resulting trajectory is the preliminary result of the offline planning and will be executed by the robot initially.



Along the way, the robot should avoid humans to ensure a safe behavior. These evasive motions need to be calculated online, as they depend on the current distance between the robot and human.

6. The positions of key-points of nearby humans are detected by visual sensors. While this is a complex task in practice, we assume a given method that returns a list of Cartesian points. Alternatively, a voxel-based occupancy grid allow simpler algorithms for sensor evaluation. As long as the distance is below a pre-defined threshold, the original offline trajectory is continued.



7. Otherwise, the online adaption takes over: The general idea is to use a *potential field* (cf. Sec. 4.4) to move the robot around the human to the next (not yet reached) via point. Each key-point of nearby humans is added as an obstacle to the potential field. In particular, the potential field does not control the robot directly, but only a *control point* instead. Then, the robot calculates a time-optimal motion profile to the control point using the online capability of the Ruckig library. This update is fed back to the potential field planner to comply with the kinematic limits of the robot. This approach is inspired by Flacco et al. [25].
8. Finally, the low-level robot controller follows and executes the desired trajectory. In most cases, the software interface of the robot will accept *positions* within the real-time control loop. Oftentimes, a finely tuned controller by the robot's manufacturer based on Computed Torque Control (cf. Sect. 3.3) is used.

Robots are versatile tools for solving numerous tasks. However, they require various, and oftentimes application-specific, control and planning algorithms. In this regard, we have introduced only the most fundamental and general building blocks while focusing on robot arms. With this application example, we have highlighted the interfaces required for integrating different algorithms into a complete robotic setup. Of course, there are countless other algorithms that we could not cover in the

brevity of this chapter. If interested, please refer to our recommended literature in the following section.

6 Further Reading

This chapter was inspired by other influential textbooks, papers, and conversations with colleagues. While this chapter is specifically written for the “*Robotics Goes MOOC*” series, we would like to acknowledge that many textbooks already exist. They have served many students around the world very well. From a historic perspective, we would like to start by referring to the edited book by Brady et al. from 1982 [26]. The articles in this book have defined the foundations of robotics as we know them today. Some parts of this chapter are also based on chapters of the *Handbook of Robotics* [27]. The Handbook was written for an academic audience, researchers, and practitioners; it is not necessarily well-suited for teaching robotics foundations from scratch.

Robot Control

One of the early and most influential textbooks that has shaped many university courses and lectures is the introductory textbook by Craig in its first edition from 1986 [6]. It covers spatial descriptions, robot kinematics, robot dynamics, basic motion planning algorithms, design, and control of robot arms. Murray et al.’s textbook from 1994 [28] offers a great introduction to Lie algebra applied in the domain of robotics. Mason’s textbook from 2001 [12] focuses on kinematics, dynamics, and control for physical robot manipulation. It explains the importance of force and torque control with respect to a task frame in a comprehensive manner. Spong et al.’s textbook from 2006 [13] addresses two distinct parts: (1) foundations of spatial descriptions and basics about control and motion planning; and (2) robot dynamics and sensor-based control. It is one of the most comprehensive introductory books. Offering a solid mathematical foundation, Siciliano et al.’s book from 2008 [29] covers simple and advanced robot control concepts as well as model-based motion planning algorithms. The most recent, most accessible, and from my subjective perspective the new gold standard for introductory robotics textbooks, Lynch et al.’s textbook from 2008 [5] provides very comprehensive, deep, and broad overview to the most important topics: spatial descriptions using Lie algebra (often based on [28]), kinematics, dynamics, path and trajectory planning, and control. All chapters are accompanied by online content and exercises. The two most relevant chapters from the *Handbook of Robotics* [27] are “*Motion Control*” by Chung et al. [30] and “*Motion for Manipulation Tasks*” by Kuffner et al. [31].

Motion Planning

The works of Latombe have influenced the domain of robot motion planning since the early 1980s. His text book from 1991 [32] offers a bread and deep overview of the most foundational robot motion planning concepts. Offering more modern approaches, including probabilistic algorithms, the text book of Choset et al. from

2005 [33] is used in many advanced robotics courses. LaValle's textbook from 2006 [34] provides an abstract foundation to various motion planning problems and solutions. It delivers a strong mathematical background that is relevant for many derived works. With a focus on trajectory planning for robotics and machine tools, the book of Biagiotti et al. from 2008 [35] introduce a large set of motion planning algorithms. This book is particularly valuable for practitioners as the described algorithms can often be directly implemented for real-world solutions. The two most relevant chapter from the *Handbook of Robotics* [27] is "Motion Planning" by Kavraki et al. [36].

7 Exercises

Exercise 1 Solve the analytical forward and inverse kinematics for a planar three-link robot arm. The joint angles q_1 , q_2 , and q_3 are defined as shown in Fig. 13. First, calculate the forward kinematics (x, y, θ) for a given joint configuration $\vec{q}$. Then, solve for the inverse kinematics to move the robot to a specified planar coordinate. How many solutions exist in general?

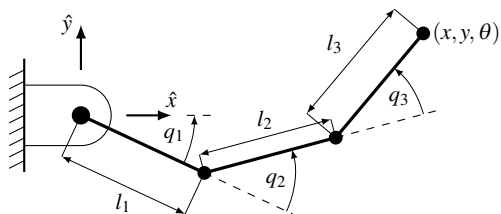
Exercise 2 In Sec. 3.2, the stability of a PD joint controller was briefly discussed. Consider now the stability of a complete PID controller. To calculate the set-point dynamics of the controlled system, take the derivative before analyzing the roots of the characteristic equation first. Note that all roots require a negative real part to ensure stability and convergence to zero error. What are the constraints for the control gains K_p , K_i , and K_d ?

Exercise 3 Consider a trapezoidal motion profile as introduced in Sect. 4.1.1. When synchronizing multiple DoFs, they should reach their target position at the same point of time T . Therefore, some DoFs need to be slowed down. Calculate the intermediate durations $\mathbf{T}_1$ to $\mathbf{T}_3$ for a given duration T so that the required energy for the motion is minimized. Transfer this solution to a jerk-limited S-curve profile using a computer algebra system.

Exercise 4 Polynomial trajectories offer a simpler alternative to time-optimal motion profiles. Here, the trajectory is parametrized by a polynomial

$$q(t) = a_n t^n + a_{n-1} t^{n-1} + \dots + a_1 t + a_0$$

Fig. 13 A planar, three-link robot arm. The rotary joints i position the links of length l_i at a joint angle q_i . Let (x, y) be the end-effector position and θ its orientation



of degree n . Let an intermediate section between two via points q_0 and q_1 be completely defined with velocities and accelerations. Which degree does a polynomial need to ensure an acceleration-constrained or jerk-constrained trajectory? Find the mapping g that calculates the parameters of the polynomial trajectory $\vec{a} = g(q_0, \dot{q}_0, \ddot{q}_0, q_1, \dot{q}_1, \ddot{q}_1)$ for both cases.

Exercise 5 Implement the RRT algorithm in a programming language of your choice. Consider a 2D environment corresponding to a rectangle between the coordinates $(0, 0)$ and $(1, 1)$.

1. First, start with the basic RRT algorithm that returns as soon as a solution is found. The exact path can be found by saving the parent of each node and walking backwards from the found goal node.
2. Your algorithm should find a collision free path around obstacles. They are given as a list of closed polygons; each line is a boundary that must not be crossed by the robot. What happens if no valid solution exists?
3. Finally, extend your motion planner to the RRT* algorithm. Give a brief overview about the qualitative influence of each hyperparameter.

8 Cross-References

This chapter contains cross-references to content published in other chapters belonging to the four “Robotic Goes MOOC” books: KNO4, KNO5, KNO6, DES2, INT5, IMP1, IMP2, IMP3, IMP4, IMP5, IMP6. The full list of chapter abbreviations is available in the Preface.

References

1. International Organisation for Standardisation. ISO 8373: manipulating industrial robots — Vocabulary. International Standard (2012)
2. International Federation of Robotics (IFR) (2020). Homepage. <http://www.ifr.org>. Accessed 21 Dec 2020
3. euRobotics aisbl (2020). Homepage. <https://www.eu-robotics.net>. Accessed 21 Dec 2020
4. M. Raghavan, B. Roth, Inverse kinematics of the general 6r manipulator and related linkages. *J. Mech. Design* **115**(3), 502–508 (1993)
5. K.M. Lynch, F.C. Park, *Modern Robotics: Mechanics, Planning, and Control*, 1st edn. (Cambridge University Press, Cambridge, 2017). <http://www.modernrobotics.org>
6. J.J. Craig, *Introduction to Robotics: Mechanics and Control*, 4th edn. (Pearson, London, 2017)
7. CAN in Automation e.V., Kontumazgarten 3, D-90429 Nuremberg, Germany. CiA 402 series: CANopen device profile for drives and motion control (2020). <https://www.can-cia.org/can-knowledge/canopen/cia402/>. Accessed: 28 Dec 2020
8. EtherCAT Technology Group, Ostendstraße 196, D-90482 Nuremberg, Germany (2020). <http://www.ethercat.org>. Accessed 28 Dec 2020
9. Ethernet POWERLINK Standardization Group, c/o B&R Industrie-Automation AG, Langfeldstraße 90, 8500 Frauenfeld, Switzerland (2020). <https://www.ethernet-powerlink.org/>. Accessed 28 Dec 2020

10. PROFIBUS Nutzerorganisation e.V. (PNO), Haid-und-Neu-Straße 7, 76131 Karlsruhe, Germany (2020). <https://www.profibus.com/technology/profinet>. Accessed 28 Dec 2020
11. EtherNet/IP, ODVA, Inc., Ann Arbor, Michigan USA (2020). <https://www.odva.org/technology-standards/key-technologies/ethernet-ip/>. Accessed 28 Dec 2020
12. M.T. Mason, *Mechanics of Robotic Manipulation* (MIT Press, Cambridge, 2001)
13. M.W. Spong, S.A. Hutchinson, M. Vidyasagar, *Robot Modeling and Control* (Wiley, Hoboken, 2006)
14. T. Kröger, F.M. Wahl, On-line trajectory generation: basic concepts for instantaneous reactions to unforeseen events. *IEEE Trans. Robot.* **26**(1), 94–111 (2010)
15. L. Berscheid, T. Kröger, Jerk-limited real-time trajectory generation with arbitrary target states, in *Robotics: Science and Systems XVII* (2021)
16. T. Kunz, M. Stilman, Time-optimal trajectory generation for path following with bounded acceleration and velocity, in *Robotics: Science and Systems VIII* (2012), pp. 1–8
17. H. Pham, Q.C. Pham, A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Trans. Robot.* **34**(3), 645–659 (2018)
18. T. Lozano-Perez, Spatial planning: a configuration space approach, in *Autonomous Robot Vehicles* (Springer, Berlin, 1990), pp. 259–271
19. E.G. Gilbert, D.W. Johnson, S.S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE J. Robot. Autom.* **4**(2), 193–203 (1988)
20. S.M. LaValle, et al., Rapidly-exploring random trees: a new tool for path planning. *Computer Science Dept.* (1998)
21. S. Karaman, Incremental sampling-based algorithms for optimal motion planning. *Robot. Sci. Syst.* **VI** **104**(2), (2010). <https://www.roboticsproceedings.org/rss06/p34.html>. <https://www.roboticsproceedings.org/rss06/index.html>
22. J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, P. Abbeel, Motion planning with sequential convex optimization and convex collision checking. *Int. J. Robot. Res.* **33**(9), 1251–1270 (2014)
23. R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, 2018)
24. J.C. Kiemel, T. Kröger, Learning robot trajectories subject to kinematic joint constraints, in *2021 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, Piscataway, 2021), pp. 4799–4805
25. F. Flacco, T. Kröger, A. De Luca, O. Khatib, A depth space approach to human-robot collision avoidance, in *Proceedings of the IEEE International Conference on Robotics and Automation, Saint Paul* (2012), pp. 338–345
26. M. Brady, J.M. Hollerbach, T.L. Johnson, T. Lozano-Pérez, M.T. Mason, (eds.), *Robot Motion: Planning and Control* (MIT Press, Cambridge, 1982)
27. B. Siciliano, O. Khatib, (eds.), *Springer Handbook of Robotics*, 2nd edn. (Springer, Berlin, 2016)
28. R.M. Murray, Z. Li, S.S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, 1st edn. (CRC Press, Boca Raton, 1994)
29. B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing (Springer, Berlin, 2008)
30. W. Chung, L.C. Fu, T. Kröger, Motion control, in *Springer Handbook of Robotics*, chap. 8, 2nd edn. (Springer, Berlin, 2016), pp. 163–194
31. J. Kuffner, J. Xiao, Motion for manipulation tasks, in *Springer Handbook of Robotics*, ed. by B. Siciliano, O. Khatib (eds.), chap. 36, 2nd edn. (Springer, Berlin, 2016), pp. 897–930
32. J.C. Latombe, *Robot Motion Planning* (Kluwer, Boston, 1991)
33. H. Choset, K.M. Lynch, S. Hutchinson, G.A. Kantor, W. Burgard, L.E. Kavraki, S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, 1st edn. (MIT Press, Chicago, 2005)

34. S.M. LaValle, *Planning Algorithms* (Cambridge University Press, Cambridge, 2006)
35. L. Biagiotti, C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots*, 1st edn. (Springer, Berlin, 2008)
36. L.E. Kavraki, S.M. LaValle, Motion planning, in *Springer Handbook of Robotics*, ed. by B. Siciliano, O. Khatib (eds.), chap. 7, 2nd edn. (Springer, Berlin, 2016), pp. 139–162

Force Control



Luigi Villani

Abstract Controlling the force exchanged between a robot manipulator and the environment during the physical interaction is crucial for the successful execution of a number of practical tasks both in industrial and service robotics. The chapter focuses on basic control problems related to interaction control of robot manipulators with static/dynamic environments or with humans. It starts from the analysis of impedance control, conceived to keep the contact forces limited by ensuring a suitable compliant behavior to the robot without requiring an accurate model of the environment. The coupled stability and performance of the interacting systems is analyzed by using the concept of passivity, starting from simple linear examples. Then, the problem of interaction tasks modeling is considered, in the case of rigid environments of known geometry imposing constraints to the robot motion, as well as in the case of compliant environments of elastic type. This formulation leads to the synthesis of hybrid force/motion control schemes.

1 Introduction

Robots are typically used to accomplish both operations which require pure motion and operations involving physical interaction, with the exchange of forces with the environment. For instance, in assembly lines of automotive factories, industrial manipulators are used to perform pure motion tasks as spray painting or glue deposition, but also to execute operations involving physical interaction where forces are exchanged through a tool mounted on the robot end effector, e.g., surface polishing, removing material by carving or grinding, assembling mechanical parts.

Many service robots are designed to work in close proximity with humans, or for direct human-robot interaction, also physical, within a shared space. These kinds of robots, known as collaborative robots or *cobots*, are also being used in industrial

L. Villani (✉)

Department of Electrical Engineering and Information Technologies, Università di Napoli Federico II, Napoli, Italy

e-mail: luigi.villani@unina.it

applications, to automate unergonomic tasks such as helping workers moving heavy parts, or machine feeding, or assembling.

To ensure safety during physical interaction, a fundamental requirement is *compliance*. Compliance allows to keep limited the forces exchanged during the interaction and can be achieved both in passive and in active way.

Passive compliance can be obtained, for example, by using soft materials and elastic interfaces. An example of passive compliance device used in industrial assembling operations is the so called Remote Center of Compliance, which allows to achieve a selective behaviour, e.g. different but fixed compliance behaviours along or about different directions.

The alternative to passive compliance is that of achieving a selective compliant behaviour actively, using control. *Active compliance* allows to set and modify the compliance by software, according to the task requirements. *Force control* refers to the control approaches that allows the robot to achieve active compliance or, more in general, are designed to be effective in case of physical interaction of the robot with the environment.

Force control strategies are adopted in many robotic fields: they are used in grasping, physical human-robot interaction, cooperative robotics, tele-manipulation, locomotion, just to mention a few.

Many but not all force control strategies make use of the feedback of the measure of the interaction force through suitable force/torque sensors.

A typical force sensor in industrial robotics is the wrist sensor, mounted at the basis of the tool, that allows to measure the forces and moments due to the interaction of the tool with the environment.

Contact forces and moments can be also estimated by using torque sensors at the robot joints, if available, or using the measures of the currents of the joint actuators, that are proportional to the torques. These are especially useful if the interaction occurs not at the end effector but on the robot body, and for this reason they are often present in collaborative robots to enhance safety.

Other force sensors can be also mounted on the robotic fingers, and they usually are tactile, in the sense that they allow to measure both the force (usual the contact force) and to locate the area of contact. These sensors are used for grasp control.

To fully analyse the physical interaction of a robot with a surface, an object or a human (in the following, the two interacting systems will be referred as *robot* and *environment*) there are three different situations to be considered:

- free motion, when the robot is not in contact with the environment;
- transition from free motion to contact and viceversa;
- the robot and the environment are in contact.

The problem of controlling the robot before contact, or after that contact is lost, is a motion control problem, but it is closely related to interaction control during the transition phases, when the robot is making or breaking contact. The control of transitions is very challenging and will not be considered here. In the following analysis, it is assumed that the robot is already in contact with the environment and that the contact is not lost.

2 Control of Robot Physical Interaction

During physical interaction, that may occur through a contact point or a distributed surface, there is an exchange of mechanical power between the robot and the environment. This power can be computed as the scalar product of a velocity $\mathbf{v}$ and a force $\mathbf{f}$, defining an *interaction port*, as follows:

$$P = \mathbf{v}^T \mathbf{f}, \quad (1)$$

where the power P is a scalar, while $\mathbf{v}$ and $\mathbf{f}$ are usually vectors (twists, wrenches or suitable vectors of generalised coordinates). Therefore, in order to have a controlled physical interaction, the two variables that are candidate to be controlled are the velocity $\mathbf{v}$ and the force $\mathbf{f}$ of the robot at the interaction port.

Using physical intuition, it is clear that it is not possible to control the velocity and the force at the same time, because if one system (the controlled robot) imposes velocity, the corresponding force depends on the velocity itself and on the dynamics of the other system, which is not controlled, and viceversa. Therefore, three options are available:

1. controlling the velocity (or the position) of the robot;
2. controlling the force applied by the robot to the environment;
3. controlling the dynamic relationship between force and velocity.

The first two options are more “usual”, in the sense that, usually, controlling a system means selecting the variables to control (velocity, force). The third option, proposed by Neville Hogan in mid 80s, is the so-called impedance control.

Which is the most suitable control strategy? It depends on the particular task. Let us consider some examples.

When we have an *inertial* environment (e.g., a robot manipulator lifting an object) it makes more sense to control the velocity (or the position) of the robot, because it is more important to follow a given path in a reasonable time rather than imposing a given force to the object. In this case, the interaction force depends on the weight and inertia of the object. The inertial component, which depends on the acceleration, can be reasonably kept limited by limiting the acceleration imposed to the object. For the design of the motion controller, this force could be considered as an external disturbance, or a parametric uncertainty, or taken into account into the robot dynamics.

Similarly, if the interaction force is of *damping* type (viscous friction or damping proportional to the velocity, e.g., when a mobile robot pushes a box with small inertia sliding on the floor), the force can be kept limited by limiting the controlled velocity.

On the other hand, when we have an environment of *elastic* type (i.e., which behaves like a spring), or in the case that the elasticity is the dominant property of the physical behaviour of the environment (like when we push against a deformable surface that tends to recover the original shape), we can still control the velocity (or

position), taking into account that the interaction force will be proportional to the deformation and to the stiffness coefficient. In this case, however, there could be problems if a pure motion control strategy is adopted, especially when the stiffness coefficient is high. In fact, uncertainty in the knowledge of the stiffness coefficient and on the position of the surface may produce large forces in case that the motion controlled robot attempts to reach a position that produces a large deformation. The force may overcome the threshold that causes damages in the physical systems or saturation of the robot actuators (stability is at risk!). In this case, a control strategy aimed at imposing a given force and not a given motion (velocity, position) is more advisable.

In view of the considered examples, we can conclude that:

- *motion control* is suitable to manage the interaction with environments of inertial or damping type or, as limit case, when no interaction occurs, when the force (as well as the exchanged mechanical power) is null for any velocity;
- pure *force control* is suitable in the presence of high interaction forces in response to relatively small robot displacements, namely, when the environment is of elastic type and the stiffness coefficient is high. The limit case is when the environment is rigid (infinite stiffness): in this case, along the directions where interaction occurs, the velocity (as well as the exchanged mechanical power) is always null and the contact force can be arbitrarily imposed;
- in the intermediate situations, a possible alternative is to use *impedance control*, which is not aimed at imposing to the robot a given motion or a given interaction force, but at achieving for the controlled robot a desired dynamic behaviour between velocity (or position) and force at the interaction port.

It is important to emphasise that a real situation of physical interaction is often a distributed phenomenon in which the local geometric and physical characteristics of the contact surfaces as well as the global dynamics of the robot and of the environment are involved. Moreover, there are tasks where the environment imposes kinematic constraints to the end-effector motion. In these situations the interaction forces include the constraint forces, i.e., the reaction forces that arise when the end effector tends to violate the constraints. This happens, for example, in the case of a robot sliding a rigid tool on a rigid surface, or a robot turning a crank.

An accurate mathematical model which allows to catch all these phenomena is often difficult to obtain. Therefore, interaction control laws are usually designed on the basis of simplifying assumptions. The effectiveness in real situations, where some of the ideal assumptions are relaxed, is entrusted to the robustness of the closed-loop control.

The literature on force control is very rich and different categorisations of force control laws have been proposed. In this chapter a complete categorisation is not presented, but two main control approaches are illustrated. The first is based on *impedance control*, which is usually adopted when the model of the environment is not available or uncertain, and the interaction is modelled as an exchange of mechanical power with a passive system, i.e., a system which cannot generate energy. The second is based on *hybrid force/motion control*, which is useful

especially in the presence of kinematic constraints and consists in controlling the motion along the unconstrained task directions and the force along the constrained task directions.

3 Impedance Control

Impedance control is based on the concepts of mechanical *impedance* and *admittance* at an interaction port, which are the analogous of an electrical impedance/admittance of a bipolar circuit. The mechanical power flowing through the port depends on the velocity, which is also denoted as flow variable and is the analogous of a current in an electric circuit, and on the force, which is denoted as effort variable, and can be thought as the analogous of a voltage.

A mechanical (electrical) impedance (usually denoted as Z) at an interaction port is the dynamic operator that determines an output force (voltage) in response to an input velocity (current) at the same port.

Using the same electrical analogy:

A mechanical (electrical) admittance (usually denoted as Y) at an interaction port can be defined as the dynamic operator that determines an output velocity (current) in response to an input force (voltage) at the same port.

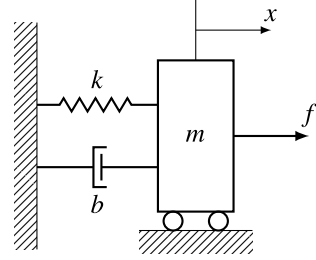
In the linear scalar case, impedance and admittance can be represented as transfer functions in the Laplace domain and the corresponding transfer functions are reciprocal, i.e., $Z(s) = 1/Y(s)$.

A typical example of linear mechanical impedance is analysed in the following.

Example 1 (Translational Mass-Spring-Damper System) Consider a mass m attached to a spring of stiffness k , in the presence of viscous friction represented by the damper b , which translates along a single direction subject to an external force f (see Fig. 1). The equation of motion can be written in the form:

$$f = m\dot{v} + bv + kx,$$

Fig. 1 Mass-spring-damper system



where the position x is the integral of the velocity:

$$x(t) = \int_0^t v(\tau) d\tau.$$

The Laplace-transformed equation can be easily computed as:

$$F(s) = \left(ms + b + \frac{k}{s} \right) V(s) = Z(s) V(s),$$

where

$$Z(s) = ms + b + \frac{k}{s} \quad (2)$$

is the mechanical impedance of the system. The admittance can be computed as $Y(s) = 1/Z(s)$

In a more general case, the mechanical impedance equation can be nonlinear and multivariable. For example, using a state-space representation of system dynamics, a mechanical admittance of a Lagrangian system can be described by a state equation and an output equation relating input generalized force $\mathbf{f}(t) \in R^m$ to output velocity $\mathbf{v}(t) \in R^m$ as follows:

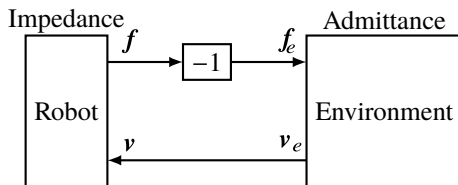
$$\dot{\mathbf{x}} = \mathbf{Y}_x(\mathbf{x}, \mathbf{f})$$

$$\mathbf{v} = \mathbf{Y}_o(\mathbf{x}, \mathbf{f})$$

where $\mathbf{x}(t) \in R^n$ is the state vector of the system, and $\mathbf{Y}_x$ and $\mathbf{Y}_o$ are, in general, nonlinear functions. The mechanical impedance can be defined by inverting the *causality*, i.e., exchanging the role of the input and of the output.

The physical interaction of the robot with the environment can be modelled as an exchange of mechanical power at an interaction port of two physical systems modelled as impedance or admittance.

Fig. 2 Connected impedance/admittance with common velocities and opposite forces at the interaction port



In general, the robot and the environment, taken separately, can be represented both as impedance and admittance, i.e., their causality can be inverted. However, the appropriate causality of each system often depends on the mechanical features of the environment and of the robot, as well as on the way they are coupled. When they interact, the causality of the robot and of the environment must be reciprocal, because the output of one system must be the input of the other and vice-versa.

For example, if the environment is a rigid surface, the velocity along the direction orthogonal to the surface is constrained to be zero (kinematic constraint). Therefore, the input variable of the environment at the interaction port must be the force and thus the environment must be modelled as an admittance (which is zero in the direction orthogonal to the surface). On the other hand, if the robot is driven by actuators that use transmissions with poor back-drivability, then it is more convenient to model the robot as an admittance which imposes a velocity to the environment.

Figure 2 shows one of the two possible causal interconnections between the robot and the environment, where the robot is modelled as an impedance and the environment is modelled as an admittance. Notice that, at the interaction port, the velocities of the two systems are the same ($v_e = v$), while the forces acting on the systems have opposite signs ($f_e = -f$), according to the Newton's third law of action and reaction.

It is important to notice that:

The velocity v and force f at the interaction port depends on the dynamics of both the robot and of the environment, and cannot be predicted in the absence of a complete characterisation of both the systems. On the other hand, the impedance and the admittance of each system are independent from each other.

Therefore, the impedance or admittance of the robot can be modified by using feedback control, regardless the environment. The impedance (admittance) of a controlled robot at the interaction port is usually denoted as “apparent” impedance (admittance).

This is where the idea of impedance control comes from:

Design of a control law that allows the robot to behave like a desired impedance or a desired admittance at the interaction port.

According to this view, a successful controller has not the goal of ensuring tracking or regulation of one or more physical variables of the system, but must satisfy the following goals:

- coupled stability regardless of the environment, or for certain class of environments;
- satisfactory performance, i.e., minimal deviation of the apparent impedance (or admittance) from the desired one.

The above goals guarantee a good behaviour during the transient and at steady state, for both force and velocity.

For what concerns stability, it is important to notice that a system that is stable in isolation can become unstable when coupled to another system that is itself stable. Therefore, the issue of coupled stability is of the utmost importance.

3.1 *Coupled Stability*

To gain insight into the problem of coupled stability, it is useful to start from the case of linear Single Input Single Output (SISO) linear time-invariant systems. The achieved results can be (partially) extended to the multivariable and nonlinear case.

The concept of stability of systems that exchange power at an interaction port can be related to the concept of *passivity* of a system.

Passivity quantifies the notion that the energy that a system can deliver, for any time period, at its port of interaction, cannot be greater than the energy that, in total, was input into the system from the same port for all time.

A passive system cannot generate energy, the power injected into the system can be only dissipated or stored. A stronger property of passivity is that of dissipativity or *strict passivity*.

For SISO time-invariant linear systems passivity can be translated into properties of the transfer functions. In particular, passive systems have *positive real* (PR) transfer functions, while strictly passive systems have *strictly positive real* (SPR) transfer functions.

While passivity and PR transfer function are equivalent properties, there are slightly different definitions of strict passivity related to different SPR conditions (see, e.g., [1] for details). For our purposes, the following definitions are adopted.

Definition 1 A transfer function $H(s)$ is *positive real* (PR) if and only if:

1. $H(s)$ has no poles in the right half plane;
2. the imaginaries poles of $H(s)$, if any, are simple and have positive real residues (i.e., a positive coefficient after partial fraction expansion);
3. the real part of the complex function $H(j\omega)$ is non-negative, i.e., $\text{Re}(H(j\omega)) \geq 0$ for any $\omega \in \mathbb{R}$ for which $j\omega$ is not a pole of $H(j\omega)$.

Definition 2 A PR transfer function $H(s)$ is said to be *strictly positive real* (SPR) if satisfies the following conditions:

1. $H(s)$ has no poles in the closed right half-plane.
2. $\text{Re}(H(j\omega)) > 0$ for all $\omega \in \mathbb{R}$.
- 3a. $\lim_{\omega^2 \rightarrow \infty} \omega^2 \text{Re}(H(j\omega)) > 0$ when $r = 1$, or
- 3b. $\lim_{\omega^2 \rightarrow \infty} \text{Re}(H(j\omega)) > 0$, $\lim_{|\omega| \rightarrow \infty} \frac{H(j\omega)}{j\omega} > 0$ when $r = -1$,

where $r = n - m$ is the relative degree of $H(s)$, being m and n the degrees of its numerator and denominator, respectively.

It is easy to verify that the (strictly) positive real condition $\text{Re}(H(j\omega)) \geq 0$ (> 0) is equivalent to:

$$P(\omega) = \text{Re}(N(j\omega))\text{Re}(D(j\omega)) + \text{Im}(N(j\omega))\text{Im}(D(j\omega)) \geq 0 \quad (> 0) \quad (3)$$

for all $\omega \in \mathbb{R}$ for which $j\omega$ is not a pole of $H(j\omega)$, being $N(j\omega)$ and $D(j\omega)$ the numerator and denominator of $H(j\omega)$, respectively.

A transfer function $H(s)$ needs not be proper to be PR or SPR. Also, it can be proven that, if $H(s)$ is PR (SPR), also $1/H(s)$ is PR (SPR). This means that if impedance $Z(s)$ is PR (SPR), also the admittance $Y(s) = 1/Z(s)$ is PR (SPR).

The following are necessary conditions for a transfer function to be PR (SPR):

- $H(s)$ is (strictly) minimum phase, i.e. all poles and zeros have real parts ≥ 0 if PR or > 0 if SPR.
- The relative degree of $H(s)$ is either $r = 0$ or $r = \pm 1$.
- The Nyquist contour of $H(j\omega)$ lies wholly within the (open) right half-plane; equivalently, $H(j\omega)$ has phase in the interval $[-\pi/2, +\pi/2]$ if PR or in the interval $] -\pi/2, +\pi/2[$ if SPR.

The interacting impedance-admittance system can be seen as a closed loop system and the stability can be analysed by using the Nyquist stability criterion. In detail, if the environment is represented by a passive system and the controlled robot is also a passive system, then the phase of the “open-loop” transfer function is always in the interval $[-\pi, \pi]$, because the phase of each of the two transfer functions is between $-\pi/2$ and $+\pi/2$, and the two are summed. Moreover, the number of poles of the open-loop transfer function in the right half-plane is zero. Therefore, in view of the Nyquist stability criterion, the closed loop system is never unstable, but marginally stable at worst. Moreover, if one of the two systems is strictly passive, then the phase of the open-loop transfer function is in the interval $] -\pi, \pi[$ and therefore the closed-loop system is asymptotically stable.

In view of the above considerations, it is possible to assess that:

If a robot is controlled so that it behaves like a passive system (impedance or admittance) at the interaction port, then coupled stability is ensured for any passive environment.

This is an important result of practical application in interaction control although there are limitations:

- The passivity of the controlled robot is only a sufficient condition: even though the controlled robot is not passive, the closed loop system can be asymptotically stable.
- Although passivity is very useful to guarantee stability when the model of the environment is not available or uncertain (this is a very frequent situation), it is often too conservative and produces poor transient performance: in some applications it is intentionally violated.
- Although the stability is guaranteed for any passive environment, the transient behaviour of the coupled system depends on the actual dynamics of the environment and of the robot. Hence, a model of the environment is required for parameters tuning.

- In many applications the environment is not passive, for example, in the case of interaction with a human being. In fact, the human user can exhibit non-passive dynamical behaviour. However, from everyday experience, we know that the interaction of humans with passive objects is stable, unless the human operator does not intentionally try to destabilize the system.

A useful property is that the elementary mechanical elements (translational/rotational mass, spring, damping) are passive and any power continuous connection of these elements is also passive.

The above results, to some extent, can be generalised to non-linear and multi-variable systems. In particular, there are definitions of passivity and of strict passivity of nonlinear systems described in the state space. Moreover, it is possible to prove that the feedback interconnection of two passive systems is still passive, and that the strict passivity of one of the systems, under some additional assumptions on the other system, is a sufficient condition for stability.

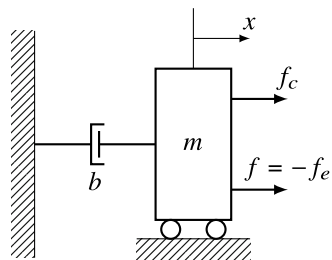
3.2 Basic Impedance Control

Consider the simplified model of a rigid robot represented in Fig. 3. This model can be interpreted as the translational mechanical analogous of a single-degree-of freedom robot composed by a rotary actuator which generates mechanical power by the supply of electrical power through a controlled current. Such an actuator is commonly an electrical motor, which imposes a torque on the mechanics of the robot, composed of a mechanical transmission with constant $k_r > 1$, and a robotic link. The mechanical transmission amplifies the torque from the motor side to the link side of a factor k_r and scales the angular velocity by a factor $1/k_r$.

In the translational analogy, all the quantities are reflected to the link side. In particular:

- $m > 0$ is the total mass, which is the analogous of the total inertia, sum of the inertia of the link and of the inertia of the motor reflected to the link side (notice that the reflected inertia is multiplied by a factor k_r^2).

Fig. 3 Simplified model of a rigid robot



- $b > 0$ is the total viscous friction coefficient reflected to the link side; friction can be usually neglected in backdrivable robots, while it is often dominant in non backdrivable robots.
- x and $v = \dot{x}$ are the position and velocity of the link (and of the environment), which are the translational analogous of the link angular position and velocity.
- f is the external force analogous to the external torque applied by the environment on the robot's link, which is opposite to force f_e applied by the robot on the environment.
- f_c is the control force analogous to the control torque.

Usually the mechanical transmissions, especially those with high k_r , introduce also non linear phenomena like Coulomb friction, stiction and backlash, that are difficult to model and are not considered here, but that further reduce backdrivability.

The equation of motion can be written in the form:

$$m\ddot{x} + b\dot{x} = f_c - f_e. \quad (4)$$

Let us choose the control force $f_c = f_{PD}$ where

$$f_{PD} = k_d(x_d - x) + b_d(\dot{x}_d - \dot{x}), \quad (5)$$

corresponding to a position control law of PD type, that can be interpreted as the force produced by a parallel spring and damper, with coefficients $k_d \geq 0$ and $b_d \geq 0$ respectively. The quantity x_d is a desired position, also known as “virtual” position, corresponding to the rest position of the spring. Deviation of the virtual position from the actual position of the robot produces a force according to Eq. (5), which “pushes” or “pulls” the robot. In the absence of interaction with the environment (i.e., $f_e = 0$), the robot's position tends to follow the virtual position. If interaction occurs, the robot position deviates from the virtual position, which could be not physically realisable (for example, the environment is a wall and the virtual position is a point beyond the wall). This is the reason of the name “virtual”. The actual motion of the robot as well as the value of the interaction force will depend, of course, on the virtual position and velocity, but also on the parameters of the two interacting systems (controlled robot and environment) at the interaction port.

The interpretation of the interacting controlled robot-environment system in terms of impedance/admittance in this simple example can be achieved in the Laplace domain by modelling the environment as an admittance, and the controlled robot as an impedance, or viceversa. In detail, control law (5) can be computed in the Laplace domain as

$$F_{PD} = Z_c(s)(V_d - V), \quad (6)$$

where V (V_d) is the Laplace transform of the velocity $\dot{x}$ ($\dot{x}_d$), and

$$Z_c(s) = b_d + \frac{k_d}{s} \quad (7)$$

is the transfer function of the controller, corresponding to a PI control on the velocity. Similarly, computing the Laplace transform of the robot gives:

$$Z_r(s)V = F_{PD} - F_e, \quad (8)$$

where:

$$Z_r(s) = ms + b \quad (9)$$

is the impedance of the uncontrolled robot. Using Eqs. (8) and (6), the force F_e can be computed as:

$$F_e = -Z(s)V + Z_c(s)V_d, \quad (10)$$

where:

$$Z(s) = Z_c(s) + Z_r(s) = ms + b + b_d + \frac{k_d}{s} \quad (11)$$

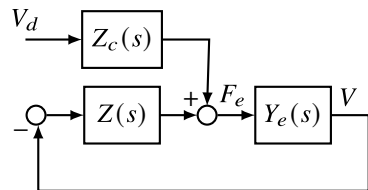
is the apparent impedance. The overall system can be represented as in Fig. 4, where $Y_e(s)$ is the admittance of the environment. One can easily verify (see Exercise 1) that the impedance $Z(s)$ in (11) is passive, for $b_d \geq 0$ and $k_d \geq 0$, and strictly passive if $b_d > 0$ and $k_d = 0$. Then the coupled stability is guaranteed in case of interaction with any kind of passive environment. The term depending on the desired velocity V_d is a feedforward contribution which is irrelevant for stability.

In view of the expression (11) of the apparent impedance it can be observed that:

- the mass m cannot be modified;
- since b is uncertain, the quantity b_d must always set non negative to guarantee passivity, and then the overall damping cannot be lower than that of the uncontrolled robot.

In sum, to ensure coupled stability, the apparent impedance $Z(s)$ can only be “higher” than the impedance of the uncontrolled robot and a desired impedance cannot be achieved, in general. This is a drawback especially for high-impedance robots, namely, heavy robots or robots with non backdrivable actuators.

Fig. 4 Block scheme in the Laplace domain of the controlled robot (impedance $Z(s)$) interacting with the environment (admittance $Y_e(s)$)



Exercises

Exercise 1 Analyse the passivity of impedance (11).

Exercise 2 Prove that the block scheme of Fig. 4, with $Z(s)$ in (11) and that of Fig. 5 are equivalent.

Exercise 3 The simplified model of a robot with elastic transmission is represented in Fig. 6, where m_m is the mass of the motor and m is the mass of the link. The control force f_c acts on the motor while the force f applied by the environment on the robot acts on the link. The parameter k is the stiffness coefficient of the transmission. Prove that the PD control law:

$$f_c = k_d(x_d - x_m) + b_d(\dot{x}_d - \dot{x}_m)$$

based on feedback of the motor position and velocity, define a passive impedance (or admittance) between the link (and environment) velocity $v = \dot{x}$ and the interaction force f . Assume $b_m = 0$ and consider only the case $k_d > 0$.

Fig. 5 Block scheme of the robot/environment system (in gray) with basic impedance control (in orange)

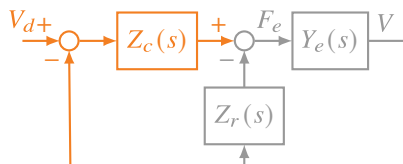
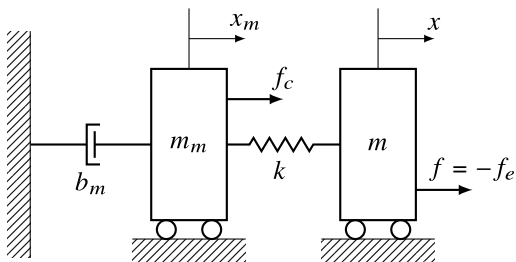


Fig. 6 Simplified model of a robot with elastic transmission



Solution The impedance of the controlled robot is:

$$Z(s) = \frac{mm_ms^4 + mb_ds^3 + (mk + m_mk + mk_d)s^2 + kb_ds + kk_d}{s(m_ms^2 + b_ds + k + k_d)},$$

with $b_m = 0$. Notice that the pole in $s = 0$ is present only if $k_d > 0$. In this case, the transfer function $Z(s)$ can be, at most, PR. If, in addition, $b_d \geq 0$, the other two poles of $Z(s)$ are not in the open right half plane, and thus Condition 1 of Definition 1 is satisfied. Condition 2 is satisfied as well, because the residue of the (simple) pole $s = 0$ is:

$$R_0 = sZ(s)|_{s=0} = \frac{kk_d}{k + k_d} > 0,$$

being $k > 0$. Also, in the case $b_d = 0$, there are two imaginary poles in $s_{1,2} = \pm j\sqrt{(k + k_d)/m_m}$ and the residues of such poles are

$$R_{1,2} = (s - s_{1,2})Z(s)|_{s=s_{1,2}} = \frac{k^2}{2(k + k_d)} > 0.$$

Condition 3 can be checked by computing the polynomial $P(\omega)$ in (3):

$$P(\omega) = kb_d\omega^2 \geq 0, \quad \forall \omega \in \mathbb{R} - \{0\},$$

if $b_d \geq 0$, being $j\omega = 0$ a pole of $Z(j\omega)$. Thus $Z(s)$ is PR and passive if $k_d > 0$ and $b_d \geq 0$.

3.3 Impedance Control with Force Feedback

The performance of impedance control can be improved by using a force sensor that allows to measure the force f_e and a suitable proportional force control law, namely:

$$f_c = k_F(f_{PD} - f_e) + f_e, \quad (12)$$

where f_{PD} is the PD position control law given in (5). The last term in (12), that allows to (ideally) cancel out the interaction force f_e , could be also omitted with minor changes to the analysis.

By replacing the above control law into the equation of the robot (4), the computation of the force F_e from the velocity V in the Laplace domain yields to the same Eq. (10), with apparent impedance:

$$Z(s) = \frac{1}{k_F} Z_r(s) + Z_c(s) = \frac{m}{k_F} s + \frac{b}{k_F} + b_d + \frac{k_d}{s}, \quad (13)$$

with $Z_c(s)$ in (7) and $Z_r(s)$ in (9). The controlled robot in interaction with the environment can be represented by the same block scheme of Fig. 4, with the apparent robot's impedance (13). It is worth observing that:

- control law (12) includes: (i) a inner force control loop, which allows to reshape the robot apparent impedance, and, in particular, to achieve a desired mass by setting $k_F = m/m_d$, and (ii) an outer PD position control loop, which allows achieving a desired damping and stiffness;
- in the presence of intrinsic structural compliance of the robot (i.e., flexibility in the links or in the joints, as it is often the case in collaborative robot)- it can be found that the gain k_F must be set in the interval $[0, 1]$ to ensure passivity (see Exercise 5).

Exercises

Exercise 4 Sketch a block scheme for the impedance control law with force feedback (12), so that the robot impedance, the environment admittance and the controller can be separated as in Fig. 5.

Exercise 5 For a robot with elastic transmission, assuming $b_m = 0$, show that control law (12) with PD control on the motor variables x_m and $\dot{x}_m$ and control gains $k_d > 0$ and $b_d > 0$, defines a passive impedance (or admittance) between the link (and environment) velocity $v = \dot{x}$ and the interaction force $f = -f_e$, provided that $k_F \in [0, 1]$.

Solution The impedance of the controlled robot is: haty

$$Z(s) = \frac{mm_ms^4 + mb_dk_Fs^3 + (mk + m_mk + mk_dk_F)s^2 + kb_dk_Fs + kk_dk_F}{s(m_ms^2 + b_dk_Fs + (k + k_d)k_F)},$$

which, in the case $k_F = 1$, coincides with the impedance of Exercise 3, with the same results.

(continued)

In the case $k_F = 0$ the transfer function $Z(s)$ reduces to

$$Z(s) = \frac{mm_ms^2 + mk + m_mk}{m_ms},$$

thus $Z(s)$ has a pole in $s = 0$ and can be at most PR, if conditions of Definition 1 are satisfied. Conditions 1. and 2. can be easily checked and are met. Condition 3. is met as well because $\text{Re}(Z(j\omega)) = 0$ for all $\omega \in \mathbb{R}$. Hence, if $k_F = 0$, the impedance is passive. In the case $k_F \neq 0$, $Z(s)$ keeps the pole in $s = 0$, thus it can be at most PR. Condition 1. of Definition 1 requires that $k_F > 0$. The residue R_0 of the pole in $s = 0$ is the same as in Exercise 3, thus is strictly positive being $k_d > 0$ and $k > 0$, so Condition 2. is satisfied as well. Condition 3. can be verified by computing function $P(\omega)$ in (3), which gives

$$P(\omega) = mkb_dk_F(1 - k_F)\omega^4 + k^2b_dk_F^2\omega^2 \geq 0, \quad \forall \omega \in \mathbb{R} - \{0\},$$

if $k_F \leq 1$, being $j\omega = 0$ a pole of $Z(j\omega)$. Thus, transfer function $Z(s)$ is PR and passive if $k_F \in [0, 1]$.

3.4 Admittance Control

The idea of admittance control is that of exploiting the motion control loop of the robot in free space, designed to make the robot as fast as possible while rejecting friction and all external disturbances (including interaction forces). Therefore, the robot is controlled to behave like a desired admittance imposing a velocity to the environment, modelled as an impedance. The desired force-velocity relation (i.e., the desired admittance) at the interaction port is achieved by using an inner motion control loop and an outer force control loop, as in Fig. 7, where $C_f(s)$ is the transfer function of the force controller, $C_v(s)$ is the transfer function of the velocity

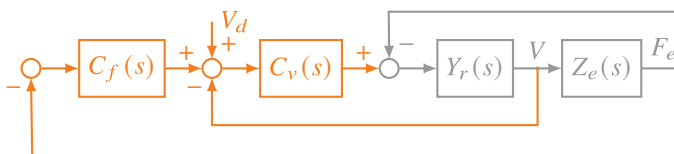


Fig. 7 Block scheme of the robot/environment system (in gray) with admittance control (in orange)

controller, $Y_r(s) = 1/Z_r(s)$ is the impedance of the robot and $Z_e(s) = 1/Y_e(s)$ is the impedance of the environment.

In the absence of interaction ($F_e = 0$), the outer force control loop is inactive and the robot follows the desired velocity V_d , i.e., $V \simeq V_d$. When interaction occurs, to study the coupled stability, it is necessary to compute the transfer function from the force $F = -F_e$ to the velocity V , i.e., the admittance of the controlled robot, assuming $V_d = 0$, being V_d irrelevant for stability. Namely:

$$V = -\frac{C_v(s)Y_r(s)}{1 + C_v(s)Y_r(s)}C_f(s)F_e - \frac{Y_r(s)}{1 + C_v(s)Y_r(s)}F_e. \quad (14)$$

In case of high gain velocity controller, the second term of the sum (14) can be neglected. This term could also be canceled by adding a force feedback to the control input that (ideally) cancels the interaction force F_e as in the control law (12). Moreover, the following approximation holds:

$$V \simeq -C_f(s)F_e = C_f(s)F, \quad (15)$$

and therefore, by setting $C_f(s) = Y_d(s)$ any kind of passive desired admittance can be achieved, for example, that corresponding to a mass-spring-damper system:

$$Y_d(s) = \frac{1}{m_d s + b_d + k_d/s}, \quad (16)$$

with all the possible combination of null and non-null (strictly positive) parameters. For example, a pure stiffness, a pure damping as well as a pure mass behaviour can be rendered, and this is not possible with impedance control.

However, the above hypotheses are approximated and the dynamics of the inner motion control loop must be taken into account for more accurate results. In general, the apparent impedance can be made close to the desired impedance within the bandwidth of the velocity control loop, where the approximation (15) is more accurate. Passivity of the apparent admittance, however, is not always guaranteed.

Exercises

Exercise 6 Consider a PI velocity controller

$$C_v(s) = k_p + \frac{k_i}{s},$$

and assume that the second term of (14) is null, i.e., the contact force is canceled by controller. Set $k_p = \alpha_v m$ and $k_i = \alpha_v b$, with $\alpha_v > 0$ so that

(continued)

Exercise 6 (continued)

the zero in $s = -b/m$ of the velocity controller cancels out the pole of $Y_r(s)$ and Eq. (14) becomes:

$$V = -\frac{\alpha_v}{\alpha_v + s} C_f F_e = Y(s) F.$$

Show that, by setting $C_f(s) = Y_d(s)$, the passivity of the apparent admittance $Y(s)$ is not always guaranteed. In particular, prove that the PR condition fails in case of pure inertia, i.e., if $Y_d(s) = 1/m_d s$ and for any other desired admittance containing an inertial contribution. Moreover, prove that the apparent admittance is SPR if $k_d = 0$ and $b_d > 0$ and PR if $k_d \geq 0$ and $b_d \geq 0$.

Exercise 7 Prove that a PR or SPR apparent admittance in the case $m_d > 0$ can be achieved considering the same controller $C_v(s)$ of Exercise 6 and setting:

$$C_f(s) = \frac{s + \alpha_v}{\alpha_v} Y_d(s).$$

3.5 Multivariable Systems

The extension of the results analysed for 1-DOF linear systems to a robot manipulator with n joint variables grouped into a $n \times 1$ joint vector $\mathbf{q}$ is not straightforward, due to the nonlinearity of the robot model and to the fact that the interaction force exerted by the manipulator on the environment is, in general, a 6×1 wrench vector $\mathbf{h}_e = [\mathbf{f}_e^T \mathbf{m}_e^T]^T$, where $\mathbf{f}_e$ and $\mathbf{m}_e$ are the 3×1 vectors of force and moment respectively.

The interaction of the robot with the environment may happen in any part of the robot's body. In this chapter, the focus will be on the case that the interaction occurs at the end effector of the manipulator and that the robot is not redundant, i.e., $n = 6$. In this case, the end-effector velocity is denoted by the 6×1 twist vector $\mathbf{v} = [\dot{\mathbf{p}}^T \boldsymbol{\omega}^T]^T$ being $\dot{\mathbf{p}}$ the translational velocity and $\boldsymbol{\omega}$ the angular velocity, and can be computed from the 6×1 joint velocity vector $\dot{\mathbf{q}}$ using the linear mapping

$$\mathbf{v} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}, \quad (17)$$

while the joint torques τ_e corresponding to h_e can be computed as:

$$\tau_e = J^T(q) f_e. \quad (18)$$

Matrix J is the 6×6 end-effector geometric Jacobian. For simplicity, the case of non-singular manipulators is considered; therefore, the Jacobian is a square and non-singular matrix.

Moreover, it is useful to consider the operational space formulation of the dynamic model of a rigid robot manipulator in contact with the environment

$$\Lambda(q)\dot{v} + \Gamma(q, \dot{q})v + \eta(q) = h_c - h_e, \quad (19)$$

where $\Lambda(q)$ is the 6×6 equivalent inertia matrix seen by the end-effector, $\Gamma(q, \dot{q})$ is a wrench including centrifugal and Coriolis effects, $\eta(q)$ is the wrench of the gravitational effects, and $h_c = J^{-T}(q)\tau_c$ is the equivalent end-effector wrench corresponding to the input joint torques τ_c . Eq. (19) can be seen as a representation of the Newton's Second Law of Motion where all the generalized forces acting on the joints of the robot are reported at the end-effector.

Exercises

Exercise 8 Derive the operational space formulation (19) of the dynamic model of a non redundant manipulator in contact with the environment, starting from the dynamic model in the joint space:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau_c - J^T(q)h_e.$$

3.5.1 Basic Impedance Control

The basic impedance control law (5) can be extended to the multivariable case by choosing $h_c = h_{PD+}$, with:

$$h_{PD+} = h_{PD} + \eta(q), \quad h_{PD} = K_d \Delta x_d + B_d(v_d - v) \quad (20)$$

where K_d and B_d are suitable matrix gains, and Δx is a suitable error between a desired and the actual end effector position and orientation. The position error component of Δx_d can be simply chosen as $p_d - p$, being p the position of the origin of a frame attached to the end effector. Concerning the orientation error component, different choices are possible, which are not all equivalent, but this issue is outside the scope of this chapter.

The control law (20), also known as *PD plus* position control in the operational space, corresponds to a wrench (force and moment) applied to the end effector, which includes a gravity compensation term $\eta(\mathbf{q})$, a viscous damping term $\mathbf{K}_d \mathbf{v}$ and an elastic wrench provided by a virtual spring with stiffness matrix $\mathbf{K}_d$ connecting the end effector frame with a virtual frame of desired position and orientation.

This control law is also denoted as *stiffness control* or *compliance control*, when the desired position and orientation of the virtual frame are set constant, i.e., $\mathbf{v}_d = 0$. In this case, at steady-state, the robot's end effector has the desired elastic behaviour

$$\mathbf{K}_d \Delta \mathbf{x}_d = \mathbf{h}_e,$$

and the coefficient of the stiffness matrix $\mathbf{K}_d$ can be selected by tacking into account the geometry and the estimated mechanical features of the environment. For example, a higher value of the stiffness coefficient along a direction means a higher accuracy of the position control at expense of higher interaction forces. Hence, the end-effector stiffness in that direction should be made low to ensure low interaction forces. Conversely, along the directions where the environment is expected to be compliant, or where there is not interaction, the end-effector stiffness should be made high so as to follow closely the desired position.

The closed-loop system (21) with (20) can be written as

$$\Lambda(\mathbf{q})\dot{\mathbf{v}} + \mu(\mathbf{q}, \dot{\mathbf{q}})\mathbf{v} - \mathbf{B}_d(\mathbf{v}_d - \mathbf{v}) - \mathbf{K}_d \Delta \mathbf{x}_d = -\mathbf{h}_e, \quad (21)$$

corresponding to a 6-DOF nonlinear and configuration dependent mechanical impedance of mass-spring-damper type, with inertia (mass) matrix $\Lambda(\mathbf{q})$, adjustable damping $\mathbf{B}_d$ and stiffness $\mathbf{K}_d$, producing the external wrench $\mathbf{h}_e$.

Using passivity, it is possible to prove the coupled stability in the case of interaction with a passive environment, which can be unknown or uncertain. However, to ensure satisfactory performance during the interaction, the elements of the matrices $\mathbf{B}_d$ and $\mathbf{K}_d$ should be properly set, considering that it is not possible to modify the natural inertia of the robot (seen from the end effector) nor the friction torques that have not been considered here but that can be easily included into the term $\mu(\mathbf{q}, \dot{\mathbf{q}})\mathbf{v}$. Hence, the performance is poor in the case that natural robot's inertia and friction are dominant with respect to the desired damping and stiffness. This happens, for example, when trying to impose a low impedance to a bulky industrial robot.

3.5.2 Impedance Control with Force Feedback

A partial solution to the aforementioned problem can be that of using an end effector torque/force sensor which allows to introduce force feedback in the control. The simplest generalisation of (12) is the control law:

$$\mathbf{h}_c = k_F(\mathbf{h}_{PD} - \mathbf{h}_e) + \mathbf{h}_e + \eta(\mathbf{q}), \quad (22)$$

where $\mathbf{h}_{PD}$ is the PD position control law given in (20) and $k_F > 0$ is a suitable control gain. This yields the closed-loop impedance equation:

$$k_F^{-1}(\Lambda(\mathbf{q})\dot{\mathbf{v}} + \boldsymbol{\mu}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{v}) - \mathbf{B}_d(\mathbf{v}_d - \mathbf{v}) - \mathbf{K}_d\Delta\mathbf{x}_d = -\mathbf{h}_e, \quad (23)$$

where the inertia and Coriolis terms can be scaled by the factor k_F .

A more effective solution can be that of using an inverse dynamics control law with force feedback:

$$\mathbf{h}_c = \Lambda(\mathbf{q})\boldsymbol{\alpha} + \boldsymbol{\mu}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{v} + \boldsymbol{\eta}(\mathbf{q}) + \mathbf{h}_e, \quad (24)$$

which produces the linear and decoupled system:

$$\dot{\mathbf{v}} = \boldsymbol{\alpha} \quad (25)$$

where $\boldsymbol{\alpha}$ is a new control input. With the choice:

$$\boldsymbol{\alpha} = \dot{\mathbf{v}}_d + \mathbf{M}_d^{-1}(\mathbf{B}_d(\mathbf{v}_d - \mathbf{v}) + \mathbf{K}_d\Delta\mathbf{x}_d - \mathbf{h}_e). \quad (26)$$

the following expression can be found for the closed-loop system:

$$\mathbf{M}_d(\dot{\mathbf{v}}_d - \dot{\mathbf{v}}) + \mathbf{B}_d(\mathbf{v}_d - \mathbf{v}) + \mathbf{K}_d\Delta\mathbf{x}_d = \mathbf{h}_e, \quad (27)$$

which can be analysed as in the single degree of freedom case. In detail, by ignoring the feedforward terms, and choosing $\mathbf{B}_d$ and $\mathbf{K}_d$ as diagonal positive semidefinite matrices, and $\mathbf{M}_d$ as a diagonal positive definite matrix, Eq. (27) defines a passive impedance from the velocity $\mathbf{v}$ to the force $\mathbf{h} = -\mathbf{h}_e$.¹ Therefore, coupled stability is guaranteed in case of interaction with any passive environment. Notice that, however, this ideal behaviour can be obtained only in the hypothesis that the friction forces, which can be included into the term $\boldsymbol{\mu}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{v}$, are negligible or can be completely cancelled out using (24), which is unlikely to happen. In any case, the desired mass cannot be null because matrix $\mathbf{M}_d$ must be invertible.

3.5.3 Admittance Control

The admittance control scheme of Fig. 7 can be easily generalised to multiple joint robots by adopting an high gain inner motion control loop to cope with friction and to ensure tracking of a reference motion trajectory. In case that the inner motion control loop is based on the inverse dynamics control law (24), leading to the

¹ Particular attention must be payed to the orientation part of Eq. (27) due to the nonlinearity of the orientation error.

decoupled linear system (25), the velocity controller and the outer force controller can be designed as illustrated for the single component. The robot admittance for each velocity component is a simple integrator, i.e., $Y_r(s) = 1/s$.

4 Hybrid Force/Motion Control

Impedance control ensures stability of the coupled robot-environment system without requiring a model of the environment, provided that both the controlled robot and the environment are passive. However, there are situations where specifying the dynamic relationship between force and motion is not enough and the successful execution of the task requires the capability of controlling both the end-effector motion and the interaction force along different directions.

A typical example is that of a robot manipulator cleaning a glass window. In this case, the manipulator should be controlled so that the cleaning tool moves on the glass surface while keeping contact and applying a suitable force.

This kind of contact situation can be modelled by doing simplifying assumptions, which avoid to take into account the dynamics of the environment, or, at least, allow to consider simple contact models. Namely, the following two cases are considered:

- both the robot and the environment are rigid and pure kinematic or *holonomic* constraints are imposed by the environment; reaction forces and moments arise when the robot tends to violate the constraints;
- the robot is rigid and the interaction forces produce a local deformation of the environment which returns to its original shape when the forces are removed; these forces are approximated by a linear elastic model.

In both cases, frictionless contact is assumed. It is obvious that the above assumptions are only ideal. However, the robustness of the controller should be able to cope with situations where some of the assumptions are not fully satisfied.

4.1 Constrained Motion

The kinematic constraints imposed by the environment to the robot's end effector motion reduce the dimension of the space of the feasible end effector velocities and of the contact forces and moments.

In the presence of m independent constraints ($m < 6$), the end effector twist belongs to a subspace of dimension $6 - m$ while the end effector wrench belongs to a subspace of dimension m , and can be expressed in the form

$$\mathbf{v} = \mathbf{S}_v(\mathbf{q})\boldsymbol{\sigma}, \quad \mathbf{h}_e = \mathbf{S}_f(\mathbf{q})\boldsymbol{\lambda} \quad (28)$$

where σ is a suitable $(6-m) \times 1$ vector and λ is a suitable $m \times 1$ vector. Matrices S_v and S_f are known as *selection matrices*. These matrices are not uniquely defined. However, it is convenient choosing S_v so that its columns are a set of $6-m$ linearly independent twists and S_f so that its columns are a set of m linearly independent wrenches. In this way σ and λ are dimensionless vectors.

The subspaces of forces and velocity are *reciprocal*, i.e.

$$h_e^T v = 0, \quad S_f^T(q) S_v(q) = 0.$$

The concept of reciprocity expresses the physical fact that, in the hypothesis of rigid and frictionless contact, the wrench does not cause any work against the twist.

An interaction task can be assigned in terms of a desired end effector twist v_d and wrench h_d that are computed as

$$v_d = S_v(q) \sigma_d, \quad h_d = S_f(q) \lambda_d, \quad (29)$$

by specifying vectors λ_d and σ_d .

4.1.1 Geometric Approach

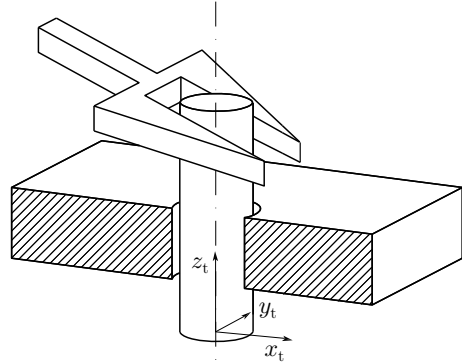
In many robotic tasks it is possible to set an orthogonal reference frame, usually referred as *task frame*, in which the matrices S_v and S_f are constant. The task frame can be chosen either attached to the end effector or to the environment. Moreover, the interaction task is specified by assigning a desired force/torque or a desired linear/angular velocity along/about each of the frame axes.

The desired forces and velocities are known as *artificial constraints* because they are imposed by the controller; these constraints concern only those variables that are not subject to the constraints imposed by the environment, known as *natural constraints*.

An example of task frame definition and task specification is given below.

Example 2 (Peg-in-Hole) The goal of this task is to push the peg into the hole while avoiding wedging and jamming. The peg has two degrees of motion freedom, hence the dimension of the velocity controlled subspace is $6-m=2$, while the

Fig. 8 Insertion of a cylindrical peg in a hole



dimension of the force controlled subspace is $m = 4$. The task frame can be chosen as in the Fig. 8 and the task can be achieved by assigning the following desired forces and torques:

- zero forces along the x_t and y_t axes;
- zero torques about x_t and y_t axes;

and the following desired velocities:

- a non-zero linear velocity along the z_t axis;
- an arbitrary angular velocity about the z_t axis.

The task continues until a “large” reaction force in the z_t direction is measured, indicating that the peg has hit the bottom of the hole, not represented in the figure. Hence, matrices S_f and S_v can be chosen as

$$S_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad S_v = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

The task frame can be chosen either attached to the end effector or to the environment.

Exercises

Exercise 9 Consider the task of sliding a prismatic block over a planar surface while pushing with a given force against the planar surface. Choose

(continued)

Exercise 9 (continued)

the task frame attached to the environment, as in the Fig. 9. Show that matrices S_f and S_v can be chosen as

$$S_f = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad S_v = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Exercise 10 Consider the task of turning a crank with idle handle. The task frame can be assumed as in the Fig. 10, attached to the crank. The handle has two degrees of motion freedom, corresponding to the rotation about the z_t axis and to the rotation about the rotation axis of the crank. Show that matrices S_f and S_v can be chosen as

$$S_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad S_v = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

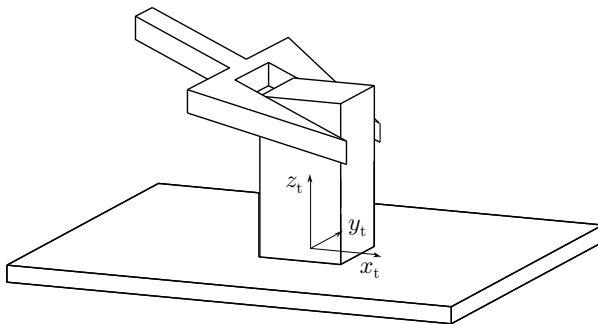
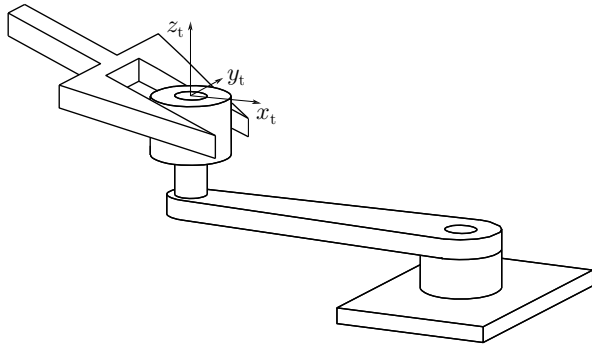


Fig. 9 Sliding of a prismatic object on a planar elastic surface

Fig. 10 Turning a crank with idle handle



4.1.2 Analytic Approach

In more complex and general situations, the matrices S_v and S_f can be found analytically, starting from the analytic expression of the *holonomic constraints* in the joint space

$$\phi(q) = 0, \quad (30)$$

where ϕ is an $m \times 1$ twice differentiable vector function, whose components are linearly independent at least locally in a neighborhood of the operating point.

Differentiation of (30) yields

$$J_\phi(q)\dot{q} = 0, \quad (31)$$

where $J_\phi(q)$ is the $m \times 6$ *constraint Jacobian*. In the absence of friction, the generalized interaction forces are represented by a reaction wrench that tends to violate the constraints. This end-effector wrench produces reaction torques at the joints that can be computed using the principle of virtual work as

$$\tau_e = J_\phi^T(q)\lambda,$$

where λ is an $m \times 1$ vector of *Lagrange multipliers*. The end-effector wrench corresponding to τ_e can be computed as

$$h_e = J^{-T}(q)\tau_e = S_f(q)\lambda, \quad (32)$$

where

$$S_f = J^{-T}(q)J_\phi^T(q). \quad (33)$$

Using (17) and (33), the equality (31) can be rewritten in the form

$$\mathbf{J}_\phi(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q})\mathbf{v} = \mathbf{S}_f^T \mathbf{v} = \mathbf{0}, \quad (34)$$

which, by virtue of (32), is equivalent to the reciprocity condition $\mathbf{h}_e^T \mathbf{v}$. At this point, matrix $\mathbf{S}_v$ can be computed from the equality $\mathbf{S}_f^T(\mathbf{q})\mathbf{S}_v(\mathbf{q}) = \mathbf{0}$.

4.2 Hybrid Force/Motion Control with Rigid Environment

The reciprocity of the velocity and force subspaces naturally leads to a control approach, known as *hybrid force/motion control*, aimed at controlling simultaneously both the contact force and the end effector motion in two reciprocal subspaces. In the following, the matrices $\mathbf{S}_v$ and $\mathbf{S}_f$ are assumed to be constant, but the results can be extended to the general case.

In the presence of holonomic constraints, the external wrench can be written in the form $\mathbf{h}_e = \mathbf{S}_f \boldsymbol{\lambda}$ and, starting from (19), it is possible to compute the dynamics of the constrained system in terms of $6 - m$ second-order equations

$$\boldsymbol{\Lambda}_v(\mathbf{q})\dot{\boldsymbol{\sigma}} = \mathbf{S}_v^T [\mathbf{h}_c - \boldsymbol{\beta}(\mathbf{q}, \dot{\mathbf{q}})], \quad (35)$$

where $\boldsymbol{\Lambda}_v = \mathbf{S}_v^T \boldsymbol{\Lambda} \mathbf{S}_v$ and $\boldsymbol{\beta}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\mu}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{v} + \boldsymbol{\eta}(\mathbf{q})$. Moreover, the vector $\boldsymbol{\lambda}$ can be computed as

$$\boldsymbol{\lambda} = \mathbf{S}_f^\dagger(\mathbf{q})[\mathbf{h}_c - \boldsymbol{\beta}(\mathbf{q}, \dot{\mathbf{q}})], \quad (36)$$

being $\mathbf{S}_f^\dagger(\mathbf{q}) = (\mathbf{S}_f^T \boldsymbol{\Lambda}^{-1}(\mathbf{q})\mathbf{S}_f)^{-1} \mathbf{S}_f^T \boldsymbol{\Lambda}^{-1}(\mathbf{q})$ a weighted pseudoinverse of matrix $\mathbf{S}_f$. This equation reveals that the contact force is a constraint force which instantaneously depends on the applied input wrench $\mathbf{h}_c$.

An inverse dynamics inner control loop can be designed by choosing the control wrench $\mathbf{h}_c$ as

$$\mathbf{h}_c = \boldsymbol{\Lambda}(\mathbf{q})\mathbf{S}_v \boldsymbol{\alpha}_\sigma + \mathbf{S}_f \mathbf{f}_\lambda + \boldsymbol{\beta}(\mathbf{q}, \dot{\mathbf{q}}), \quad (37)$$

where $\boldsymbol{\alpha}_\sigma$ and $\mathbf{f}_\lambda$ are properly designed control inputs. By replacing (37) into (35) and (36), the following linear equations can be found:

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{\alpha}_\sigma, \quad \boldsymbol{\lambda} = \mathbf{f}_\lambda, \quad (38)$$

showing a complete decoupling between motion control and force control.

Then, the desired force $\lambda_d(t)$ can be achieved by setting

$$f_\lambda = \lambda_d(t),$$

but this choice is very sensitive to disturbance forces, since it contains no force feedback. Alternative choices are

$$f_\lambda = \lambda_d(t) + \mathbf{K}_{P\lambda}[\lambda_d(t) - \lambda(t)],$$

or

$$f_\lambda = \lambda_d(t) + \mathbf{K}_{I\lambda} \int_0^t [\lambda_d(\tau) - \lambda(\tau)] d\tau,$$

where $\mathbf{K}_{P\lambda}$ and $\mathbf{K}_{I\lambda}$ are suitable positive-definite matrix gains. The proportional feedback is able to reduce the force error due to disturbance forces, while the integral action is able to compensate for constant bias disturbances.

Motion control is achieved by setting

$$\alpha_\sigma = \dot{\sigma}_d(t) + \mathbf{K}_{P\sigma}[\sigma_d(t) - \sigma(t)] + \mathbf{K}_{I\sigma} \int_0^t [\sigma_d(\tau) - \sigma(\tau)] d\tau, \quad (39)$$

where $\mathbf{K}_{P\sigma}$ and $\mathbf{K}_{I\sigma}$ are suitable matrix gains. It is straightforward to show that asymptotic tracking of $\sigma_d(t)$ and $\dot{\sigma}_d(t)$ is ensured with exponential convergence for any choice of positive definite matrices $\mathbf{K}_{P\sigma}$ and $\mathbf{K}_{I\sigma}$.

A block scheme of the hybrid force/velocity control is shown in Fig. 11, where $C_f(s)$ and $C_v(s)$ are the transfer matrices of the linear part of the force and velocity controllers.

Notice that the implementation of force feedback requires the computation of vector λ from the measurement of the end effector wrench h_e as $S_f^\dagger h_e$. Analogously, vector σ can be computed from v as $S_v^\dagger v$. Matrix $S_f^\dagger$ ($S_v^\dagger$) is any weighted pseudoinverse of matrix S_f (S_v).

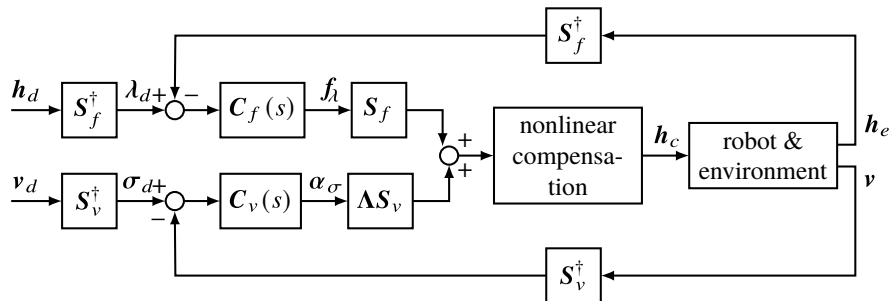


Fig. 11 Block scheme of the hybrid force/velocity control for rigid environments

Exercises

Exercise 11 Derive Eqs. (35) and (36).

Exercise 12 Derive the equations in (38).

Exercise 13 Show that, in the case that matrices S_f and S_v are not constant, Eq. (35) can be rewritten as:

$$\Lambda_v(q)\dot{\sigma} = S_v^T [h_c - \beta(q, \dot{q}) - \Lambda(q)\dot{S}_v\sigma],$$

in view of the equality $\dot{v} = S_v\dot{\sigma} + \dot{S}_v\sigma$. Moreover, in view of the identity $\dot{S}_f^T S_v = -S_f^T \dot{S}_v$, Eq. (35) can be rewritten as:

$$\lambda = S_f^\dagger(q) [h_c - \beta(q, \dot{q}) - \Lambda(q)\dot{S}_v\sigma].$$

Exercise 14 Using the results of Exercise 13, show that, in the case that matrices S_f and S_v are not constant, the equations in (38) can be achieved using the control law:

$$h_c = \Lambda(q)S_v\alpha_\sigma + S_f f_\lambda + \beta(q, \dot{q}) + \Lambda(q)\dot{S}_v\sigma.$$

4.3 Hybrid Force/Motion Control with Non-rigid Environment

The hypothesis of rigid contact can be removed, and this implies that along some directions both motion and force are allowed, although they are not independent. Hybrid force/motion control schemes can be defined also in this case.

In a general situation, the contact between the manipulator and the environment may be compliant along some directions, and rigid along some other directions. Therefore, the force control subspace can be decomposed into two distinct subspaces, one corresponding to the reaction forces and the other corresponding to

the active forces, which depend on the velocities. For simplicity, the following assumptions are made:

All the interaction forces are active and of elastic type, so that the velocity in the force control subspace can be computed as $\mathbf{v} = \mathbf{C}\dot{\mathbf{h}}_e = \mathbf{C}\mathbf{S}_f\dot{\boldsymbol{\lambda}}$. Matrix $\mathbf{C}$, which physically is a *compliance*, i.e. the inverse of a *stiffness*, is a 6×6 *constant* positive semidefinite matrix of rank equal to the dimension of the force control subspace. Moreover, it is $\mathbf{S}_v^T\mathbf{C} = \mathbf{0}$ while matrix $\mathbf{S}_f^T\mathbf{C}\mathbf{S}_f$ is full rank.

Assuming constant $\mathbf{S}_f$ and $\mathbf{S}_v$, the following decomposition for the end-effector acceleration holds:

$$\dot{\mathbf{v}} = \mathbf{S}_v\dot{\boldsymbol{\sigma}} + \mathbf{C}\mathbf{S}_f\ddot{\boldsymbol{\lambda}}. \quad (40)$$

By adopting the inverse dynamics control law (24), the linear and decoupled system (25) is achieved. Then, on the basis of the decomposition (40), with the choice

$$\boldsymbol{\alpha} = \mathbf{S}_v\boldsymbol{\alpha}_\sigma + \mathbf{C}\mathbf{S}_f\mathbf{f}_\lambda, \quad (41)$$

a complete decoupling between force control and velocity control can be achieved. In fact, replacing (40) and (41) into (25) and premultiplying both sides of the resulting equation once by $\mathbf{S}_v^T$ and once by $\mathbf{S}_f^T$, being $\mathbf{S}_v^T\mathbf{S}_v$ and $\mathbf{S}_f^T\mathbf{C}\mathbf{S}_f$ non-singular, the following decoupled equations are obtained:

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{\alpha}_\sigma, \quad \ddot{\boldsymbol{\lambda}} = \mathbf{f}_\lambda. \quad (42)$$

Notice that, if only an estimate $\widehat{\mathbf{C}}$ of the compliance of the environment is available, but contact geometry is known, so that $\mathbf{S}_v^T\widehat{\mathbf{C}} = \mathbf{0}$, control law (41) can be rewritten in the form

$$\boldsymbol{\alpha} = \mathbf{S}_v\boldsymbol{\alpha}_\sigma + \widehat{\mathbf{C}}\mathbf{S}_f\mathbf{f}_\lambda. \quad (43)$$

In this case, the closed-loop equations (42) are replaced by the equations

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{\alpha}_\sigma \quad \ddot{\boldsymbol{\lambda}} = (\mathbf{S}_f^T\mathbf{C}\mathbf{S}_f)^{-1}\mathbf{S}_f^T\widehat{\mathbf{C}}\mathbf{S}_f\mathbf{f}_\lambda, \quad (44)$$

showing that the force and velocity control subspaces remain decoupled.

As for the rigid environment case, the task can be assigned specifying a desired force, in terms of vector $\boldsymbol{\lambda}_d(t)$, and a desired velocity, in terms of vector $\boldsymbol{\sigma}_d(t)$. The overall control scheme can be represented as in Fig. 12.

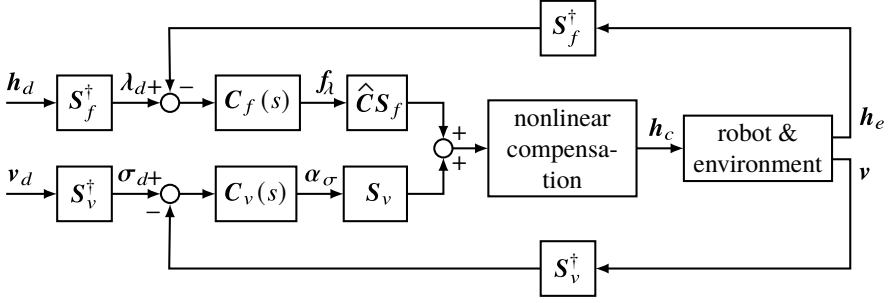


Fig. 12 Block scheme of the hybrid force/velocity control for compliant environments

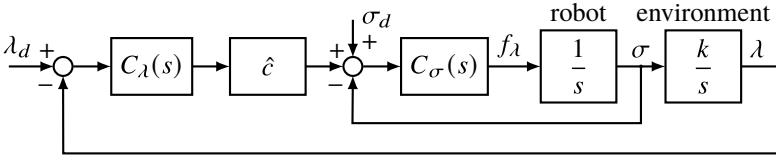


Fig. 13 Block scheme of the linear force and velocity control for a single force and velocity component in the case of elastic environment

The velocity control input α_σ can be selected as in the rigid contact case, by using (39), while the force control input f_λ can be set by taking into account that force and position are not independent, but are related by the compliance of the environment.

In detail, considering for sake of simplicity the control of a single force component, the time derivative of the force λ applied by the robot to the environment is related to the end-effector velocity along the same direction σ by the stiffness $k = c^{-1}$, i.e., $\dot{\lambda} = k\sigma$. This allows to avoid feedback of the force derivative, that can be replaced by a velocity feedback, as long as an estimate $\hat{c}$ of the compliance of the environment is available. Hence, the overall force control scheme consists of an inner velocity controller with transfer function $C_\sigma(s)$ and an outer force controller with transfer function $C_\lambda(s)$, to be suitably designed. The resulting block scheme, reported in Fig. 13, resembles the admittance control of Fig. 7, where the robot admittance is an integrator $Y_r(s) = 1/s$, the environment impedance is $Z_e(s) = k/s$ and a desired force λ_d is specified, while $\sigma_d = 0$. In the case $\sigma_d \neq 0$ the scheme is known as parallel force/position control.

Different choices are possible for the linear controllers $C_\sigma(s)$ and $C_\lambda(s)$. For example:

- P (Proportional) control on both velocity and force:

$$C_\sigma(s) = k_P, \quad C_\lambda(s) = k_F \quad (45)$$

- PI (Proportional Integral) control on velocity and P control on force:

$$C_\sigma(s) = k_P + \frac{k_I}{s}, \quad C_\lambda(s) = k_F \quad (46)$$

- PI control on both velocity and force:

$$C_\sigma(s) = k_P + \frac{k_I}{s}, \quad C_\lambda(s) = k_{PF} + \frac{k_{IF}}{s}. \quad (47)$$

All the choices ensure that the contact force is equal to the desired force λ_d at steady state. Moreover, if a good estimate of the stiffness of the environment is available, i.e., $\hat{c} = k^{-1}$, the stability and the transient performance are independent of k .

With control laws (45), the admittance of the controlled robot from the force $-\lambda$ to the velocity σ is strictly passive for any choice of $k_P > 0$ and $k_F > 0$. Then the coupled stability is ensured in case of interaction with any passive environment, not necessarily of elastic type.

With control laws (46), the integral action on the inner velocity control loop guarantees a better rejection of joint friction, especially in robots with mechanical transmissions of high transmission ratio k_r . In this case, passivity is guaranteed if the gains $k_P > 0$ and $k_I > 0$ are set so that $k_I < k_P^2$, for all values of $k_F > 0$.

With the control laws (47), the integral action in the outer force control loop guarantees a null force error at steady state, also in the presence of constant bias disturbances. However, passivity is not always guaranteed and, in the case of elastic environment, coupled stability is guaranteed if k_I and k_{IF} are set sufficiently small with respect to k_P and k_{PF} . The control design can be simplified by using a modified force controller with transfer function:

$$C'_\lambda(s) = \frac{s^2 + k_P s + k_I}{(k_P s + k_I)(1 + s\tau_F)} C_\lambda(s) \quad (48)$$

with $C_\lambda(s)$ set as in (47) and the pole $s = -1/\tau_F$, with $\tau_F > 0$ ensures that $C'_\lambda(s)$ is a proper transfer function. Using this controller, the admittance of the controlled robot can be made passive independently on the gains k_P and k_P of the velocity control loop, if the gains of the force control loop are strictly positive and satisfy condition $k_{PF} \geq k_{IF}\tau_F$.

Notice also that, in this case, the inner velocity control law is the same adopted in the velocity control subspace (39), without acceleration feedforward. This allows to use the same inner control law, possibly with different gains, for the velocity components along all the directions of the task frame, while the outer force control loop operates only for the directions in the force control subspace. This approach is known as *parallel force/position control*.

Exercises

Exercise 15 Derive the equations in (42).

Exercise 16 Compute the admittance of the controlled robot from the scheme (Fig. 13) with $C_\lambda(s)$ and $C_\sigma(s)$ as in (45) and analyse passivity. Analyse stability in the case of elastic environment, both with known and unknown environment.

Solution The admittance of the controlled robot is:

$$Y(s) = \frac{k_F k_P \hat{c}}{s + k_P}$$

which is SPR for strictly positive gains k_F and k_P . Therefore coupled stability can be guaranteed in case of interaction with any passive environment. For elastic environment, the transfer function from λ_d to λ is

$$F(s) = \frac{\hat{c} k k_F k_P}{s^2 + k_P s + \hat{c} k k_F k_P},$$

confirming that the coupled system is asymptotically stable for strictly positive gains k_F and k_P , even in case of uncertain stiffness of the environment. Moreover, being $F(0) = 1$, the contact force λ asymptotically approaches a constant set point λ_d .

Exercise 17 Compute the admittance of the controlled robot from the scheme (Fig. 13) with $C_\lambda(s)$ and $C_\sigma(s)$ as in (46) and analyse passivity. Analyse stability in the case of elastic environment, both with known and uncertain stiffness.

Solution The admittance of the controlled robot is:

$$Y(s) = \hat{c}k_F \frac{k_P s + k_I}{s^2 + k_P s + k_I}$$

which is SPR for strictly positive gains k_F, k_P and $0 < k_I < k_P^2$. This latter condition can be found computing function $P(\omega)$ in (3). Therefore coupled asymptotic stability can be guaranteed in case of interaction with any passive environment. For elastic environment, the transfer function from λ_d to λ is

$$F(s) = \frac{\hat{c}k k_F (k_P s + k_I)}{s^3 + k_P s^2 + (\hat{c}k k_F k_P + k_I)s + \hat{c}k k_F k_I},$$

showing that the coupled system is asymptotically stable if the condition

$$k_P k_I > \hat{c}k k_F (k_I - k_P^2)$$

is verified. Notice that the above inequality is always satisfied if the control gains are set so as the admittance $Y(s)$ is strictly passive, confirming that strict passivity of the robot implies asymptotic stability of the coupled system. Moreover, being $F(0) = 1$, the contact force λ asymptotically approaches a constant set point λ_d .

Exercise 18 Compute the admittance of the controlled robot from the scheme (Fig. 13) with $C_\lambda(s)$ and $C_\sigma(s)$ as in (47), but using the modified force controller (48) and analyse passivity. Analyse stability in the case of elastic environment, both with known and unknown stiffness.

Solution The admittance of the controlled robot is:

$$Y(s) = \hat{c} \frac{k_{PF}s + k_{IF}}{s(1 + s\tau_F)}$$

which is PR for strictly positive gains k_{IF}, τ_F and if $k_{PF} > k_{IF}\tau_F$. Therefore coupled stability can be guaranteed in case of interaction with any passive

(continued)

environment. For elastic environment, the transfer function from λ_d to λ is

$$F(s) = \frac{\hat{c}k(k_P s + k_{IF})}{\tau_F s^3 + s^2 + \hat{c}k k_P s + \hat{c}k k_{IF}},$$

showing that the coupled system is asymptotically stable with the same control gains for which the admittance $Y(s)$ is strictly passive. Moreover, being $F(0) = 1$, the contact force λ asymptotically approaches a constant set point λ_d .

Exercise 19 Design a hybrid force/motion control for a 2-link planar robot in contact with a frictionless elastic plane, in the case of point contact. The end effector is free to move in the direction parallel to the plane, while the elastic force is $\lambda = k\sigma$ along the direction orthogonal to the plane, being $k = c^{-1}$ the stiffness coefficient of the environment.

5 Cross-References

This chapter contains cross-references to content published in other chapters belonging to the four “Robotic Goes MOOC” books: KNO2, KNO3, DES4, INT1, INT2, INT3, INT6, IMP2. The full list of chapter abbreviations is available in the Preface.

6 Recommended Readings

This chapter has summarized the main approaches to force control in a unifying perspective. However, there are many aspects that have not been considered and that must be carefully taken into account when dealing with interaction robotic tasks. The performance of a force-controlled robotic system or, in general, of interaction control, depends on the dynamics of the environment which is very difficult to model and identify accurately. Hence, the standard performance indices used to evaluate a control system, i.e., stability, bandwidth, accuracy, and robustness, cannot be defined by considering the robotic system alone, as for the case of robot motion control, but must be always referred to the particular contact situation at hand. For this reason, although force control in industrial applications can be considered as a mature technology, standard design methodologies are not yet available. Force control techniques are employed in medical robotics, haptic systems, telerobotics,

humanoid robotics, micro and nano robotics. An interesting field of application is related to human-centered robotics, where control plays a key role to achieve adaptability, reaction capability and safety. Robots and biomechatronic systems based on series elastic actuators or on novel variable impedance actuators, with physically adjustable compliance and damping, capable to react softly when touching the environment, necessitates the design of specific force control laws.

Due to the inherent complexity of the force control problem, a large number of research papers on this topic have been published in the past three decades. A description of the state of the art of the first decade is provided in [30], whereas the progress of the second decade is surveyed in [29] and [11], as well as in the monographs [13, 26]. A more recent survey, including also robots with elastic joints and with series elastic actuators, can be found in [4].

The two major paradigms of force control (impedance and hybrid force/motion control) presented in this chapter have a number of different implementations, deriving from the specific needs of the applications or exploiting more advanced control methods. An extensive treatment of this topic with related bibliography can be found in [28] and in [27].

The original idea of a mechanical impedance model used for controlling the interaction between the manipulator and the environment was presented in [14], while the use of controlled stiffness or compliance was introduced in [25]. The stability of impedance and admittance control based on passivity for rigid and elastic joint robots is discussed in [7, 15, 16, 20, 21]. Properties of passive systems can be found in the monograph [1].

For spatial rotations and translations, the specification of task-relevant impedance (and especially stiffness) is particularly challenging. This problem is addressed, e.g., in [3], as well as in the monograph [19]. The interaction control problem of redundant robots in a multi-priority framework is considered in [22, 24].

The original idea of the hybrid force and position control was introduced in [23], based on the concepts of natural and artificial constraints proposed in [17]. The inclusion on constraints in the manipulator dynamics was considered in [18, 31], while the task frame formalism was systematically developed in [2, 9, 12]. A systematic approach to modeling the interaction with a dynamic environment was developed in [8]. Approaches designed to cope with uncertainties in the environment geometry are the force control with feedforward motion scheme proposed in [10] and the parallel force/position control [5], based on the concept of dominance of force control on motion control, thanks to the use of an integral action on the force error. A parallel force/position regulator was developed in [6].

References

1. B. Brogliato, R. Lozano, B. Maschke, O. Ekeland, *Dissipative Systems Analysis and Control* (Springer, Cham, 2020)
2. H. Bruyninckx, J. De Schutter, Specification of force-controlled actions in the task frame formalism – a synthesis. *IEEE Trans. Robot. Autom.* **12**, 581–589 (1996)
3. F. Caccavale, C. Natale, B. Siciliano, L. Villani, Six-DOF impedance control based on angle/axis representations. *IEEE Trans. Robot. Autom.* **15**, 289–300 (1999)
4. A. Calanca, R. Muradore, P. Fiorini, A review of algorithms for compliant control of stiff and fixed-compliance robots. *IEEE/ASME Trans. Mech.* **21**, 613–624 (2016)
5. S. Chiaverini, L. Sciavicco, The parallel approach to force/position control of robotic manipulators. *IEEE Trans. Robot. Autom.* **9**, 361–373 (1993)
6. S. Chiaverini, B. Siciliano, L. Villani, Force/position regulation of compliant robot manipulators. *IEEE Trans. Autom. Contr.* **39**, 647–652 (1994)
7. E. Colgate, N. Hogan, The interaction of robots with passive environments: application to force feedback control, in *Advanced Robotics*, ed. by K.J. Waldron (Springer, Berlin, 1989), pp. 465–474
8. A. De Luca, C. Manes, Modeling robots in contact with a dynamic environment. *IEEE Trans. Robot. Autom.* **10**, 542–548 (1994)
9. J. De Schutter, H. Van Brussel, Compliant robot motion I. A formalism for specifying compliant motion tasks. *Int. J. Robot. Res.* **7**(4), 3–17 (1988)
10. J. De Schutter, H. Van Brussel, Compliant robot motion II. A control approach based on external control loops. *Int. J. Robot. Res.* **7**(4), 18–33 (1988)
11. J. De Schutter, H. Bruyninckx, W.H. Zhu, M.W. Spong, Force control: a bird’s eye view, in *Control Problems in Robotics and Automation*, ed. by K.P. Valavanis, B. Siciliano (Springer, Berlin, 1998), pp. 1–17
12. J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aerbeliën, K. Claes, H. Bruyninckx, Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *Int. J. Robot. Res.* **26**(5), 433–455 (2007)
13. D.M. Gorinevski, A.M. Formal'sky, A.Yu. Schneider, *Force Control of Robotics Systems* (CRC Press, Boca Raton 1997)
14. N. Hogan, Impedance control: an approach to manipulation: parts I–III. *ASME J. Dyn. Syst. Meas. Contr.* **107**, 1–24 (1985)
15. N. Hogan, S.P. Buerger, Impedance and interaction control, in *Robotics and Automation Handbook* (CRC Press, Boca Raton, 2004), pp. 19.1–19.24
16. A. Keemink, H. Kooij, A. Herman, Admittance control for physical human–robot interaction. *Int. J. Robot. Res.* **37**, 1421–1444 (2018)
17. M.T. Mason, Compliance and force control for computer controlled manipulators. *IEEE Trans. Syst. Man Cybern.* **11**, 418–432 (1981)
18. N.H. McClamroch, D. Wang, Feedback stabilization and tracking of constrained robots. *IEEE Trans. Autom. Contr.* **33**, 419–426 (1988)
19. C. Natale, *Interaction Control of Robot Manipulators: Six Degrees-of-Freedom Tasks*. Springer Tracts in Advanced Robotics (STAR), vol. 3 (Springer, Heidelberg, 2003)
20. W.S. Newman, Stability and performance limits of interaction controllers. *ASME J. Dyn. Syst. Meas. Control* **114**, 563–570 (1992)
21. C. Ott, A. Albu-Schaeffer, A. Kugi, G. Hirzinger, On the passivity based impedance control of flexible joint robots. *IEEE Trans. Robot.* **24**, 416–429 (2008)
22. C. Ott, A. Dietrich, A. Albu-Schäffer, Prioritized multi-task compliance control of redundant manipulators. *Automatica* **53**, 416–423 (2015)
23. M.H. Raibert, J.J. Craig, Hybrid position/force control of manipulators. *ASME J. Dyn. Syst. Meas. Contr.* **103**, 126–133 (1981)
24. H. Sadeghian, L. Villani, M. Keshmiri, B. Siciliano, Task-space control of robot manipulators with null-space compliance. *IEEE Trans. Robot. Autom.* **30**, 493–506 (2014)

25. J.K. Salisbury, Active stiffness control of a manipulator in Cartesian coordinates, in *19th IEEE Conference on Decision and Control* (1980), pp. 95–100
26. B. Siciliano, L. Villani, *Robot Force Control* (Kluwer Academic, Boston 1999)
27. B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, *Robotics: Modelling, Planning and Control* (Springer, London, 2009)
28. L. Villani, J. De Schutter, Force control, in *Springer Handbook of Robotics*, ed. by B. Siciliano, O. Khatib (Springer, Berlin, 2008), pp. 161–185
29. M. Vukobratović, Y. Nakamura, Force and contact control in robotic systems, in *Tutorial IEEE International Conference on Robotics and Automation, Atlanta (1993)*
30. D.E. Whitney, Historical perspective and state of the art in robot force control. *Int. J. Robot. Res.* **6**(1), 3–14 (1987)
31. T. Yoshikawa, Dynamic hybrid position/force control of robot manipulators – description of hand constraints and calculation of joint driving force. *IEEE J. Robot. Autom.* **3**, 386–392 (1987)



François Chaumette

Abstract This chapter describes the basic concepts of robot visual control, that is, the use of visual data to control the motions of a robotic system. It describes the modeling steps allowing the design of kinematics control schemes. Applications are also described.

1 Introduction

Visual control, also named visual servoing, refers to the use of computer vision data as input of real time closed loop control schemes to control the motion of a dynamic system, a robot typically [7, 23]. It can be seen as sensor-based control from a vision sensor and relies on techniques from image processing, computer vision, and control theory.

An iteration of the control scheme (see Fig. 1) consists of the following steps:

- acquire an image;
- extract some useful image measurements;
- compute the current value of the visual features used as inputs of the control scheme;
- compute the error between the current and the desired values of the visual features;
- update the control outputs, which is usually the robot velocity, to regulate that error to zero, i.e., to minimize its norm.

For instance, for the first example depicted on Fig. 2, the image processing part consists in extracting and tracking the center of gravity of the moving people, the visual features are composed of the two Cartesian coordinates of this center of gravity, and the control law computes the pan and tilt camera velocities so that the

F. Chaumette (✉)

Inria, Université de Rennes, CNRS, IRISA, Rennes, France

e-mail: Francois.Chaumette@inria.fr

Fig. 1 Visual control is nothing but closed loop sensor-based control from visual data

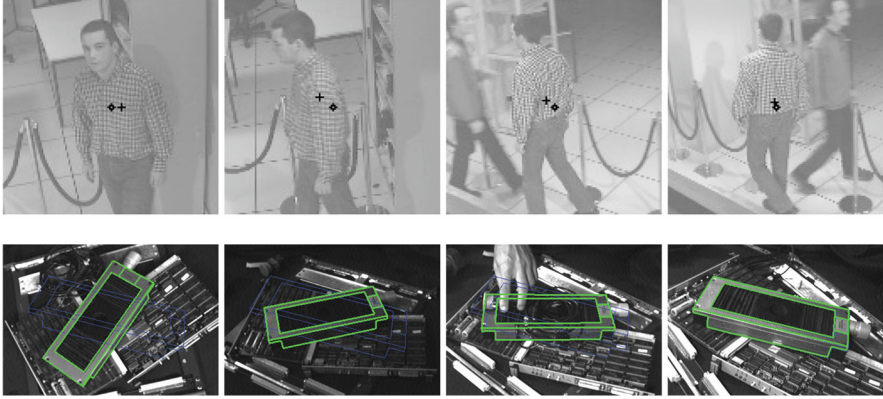
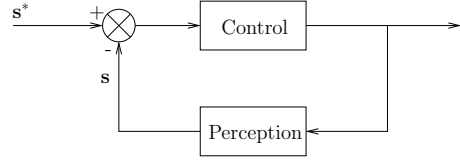


Fig. 2 Few images acquired during two visual servoing tasks: on the top, pedestrian tracking using a pan-tilt camera; on the bottom, controlling the 6 degrees of freedom of an eye-in-hand system so that an object appears at a particular position in the image (shown in blue)

center of gravity is as near as possible of the image center despite the unknown motion of the people. In the second example where a camera mounted on a six degrees of freedom robot arm is considered, the image measurements are a set of segments that are tracked in the image sequence. From these measurements and the knowledge of the 3D object model, the pose from the camera to the object is estimated and used as visual features. The control law now computes the six components of the robot velocity so that this pose reaches a particular desired value corresponding to the object position depicted in blue on the images.

2 Basic Theory

Main if not all visual servoing tasks can be expressed as the regulation to zero of an error $\mathbf{e}(t)$ which is defined by

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(\mathbf{r}(t)), \mathbf{a}) - \mathbf{s}^*(t). \quad (1)$$

The parameters in (1) are defined as follows [7]. The vector $\mathbf{m}(\mathbf{r}(t))$ is a set of image measurements (e.g., the image coordinates of points of interest, or the area, the center of gravity and other geometric characteristics of an object). These image measurements depend on the pose $\mathbf{r}(t)$ between the camera and the environment. They are used to compute a vector $\mathbf{s}(\mathbf{m}(\mathbf{r}(t)), \mathbf{a})$ of visual features, in which $\mathbf{a}$ is

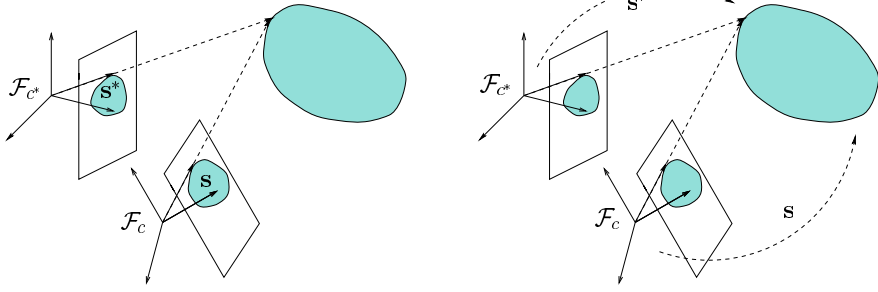


Fig. 3 If the goal is to move the camera from frame $\mathcal{F}_c$ to the desired frame $\mathcal{F}_{c^*}$, two main approaches are possible: IBVS on the left, where the features s and s^* are expressed in the image, and PBVS on the right, where the features s and s^* are related to the pose between the camera and the observed object

a set of parameters that represent potential additional knowledge about the system (e.g., coarse camera intrinsic parameters or 3D model of objects). The vector $s^*(t)$ contains the desired value of the features, which can be either constant in the case of a fixed goal, or varying if the task consists in following a specified trajectory.

Visual servoing schemes mainly differ in the way that the visual features are designed. As represented on Fig. 3, the two most classical approaches are named image-based visual servoing (IBVS), in which s consists of a set of 2D parameters that are directly expressed in the image [16, 50], and pose-based visual servoing (PBVS), in which s consists of a set of 3D parameters related to the pose between the camera and the target [48, 50, 52]. In that case, the 3D parameters have to be estimated from the image measurements either through a pose estimation process using the knowledge of the 3D target model [33] or through a triangulation process if a stereovision system is considered. Inside IBVS and PBVS approaches, many possibilities exist depending on the choice of the features. Each choice will induce a particular behavior of the system. There also exist hybrid approaches, named 2 1/2D visual servoing, which combine 2D and 3D parameters in s in order to benefit from the advantages of IBVS and PBVS while avoiding their respective drawbacks [31].

2.1 The Features Jacobian

The design of the control scheme is based on the link between the time variation of the features and the robot control inputs, which are usually the velocity of the robot joints $\mathbf{q}$. From the definition $s(\mathbf{m}(\mathbf{r}(t)), \mathbf{a})$ of the visual features, the time variation $\dot{s}$ is given by

$$\dot{s} = \frac{ds}{dt} = \frac{\partial s}{\partial \mathbf{r}} \frac{d\mathbf{r}}{dt} \quad (2)$$

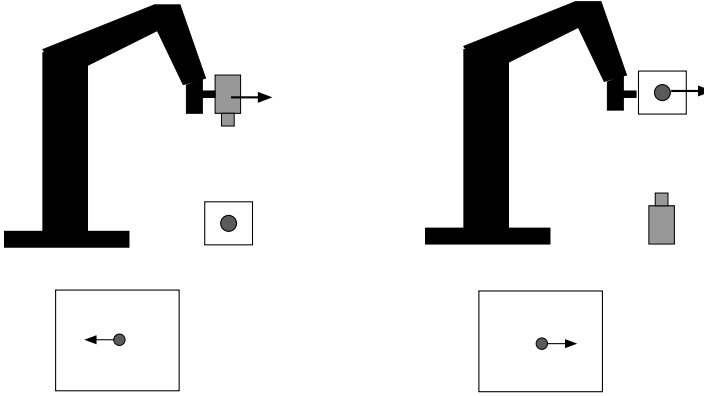


Fig. 4 In visual servoing, the vision sensor can either be mounted on the robot (eye-in-hand configuration) or observing it (eye-to-hand configuration). For the same robot motion, the motion produced in the image will be opposite from one configuration to the other

The time variation of the relative pose $\mathbf{r}$ may be due to the motion of the camera and/or the object it observes. Since only a single part of this motion is due to the robot motion, another equivalent form for defining the visual features is given by $\mathbf{s}(t) = \mathbf{s}(\mathbf{q}(t), t)$, from which we deduce

$$\dot{\mathbf{s}} = \frac{\partial \mathbf{s}}{\partial \mathbf{q}} \frac{d\mathbf{q}}{dt} + \frac{\partial \mathbf{s}}{\partial t} = \mathbf{J}_s \dot{\mathbf{q}} + \frac{\partial \mathbf{s}}{\partial t} \quad (3)$$

where $\mathbf{J}_s$ is the features Jacobian matrix, defined from the equation above similarly as the classical robot Jacobian. For an eye-in-hand system (see the left part of Fig. 4), the term $\frac{\partial \mathbf{s}}{\partial t}$ represents the time variation of $\mathbf{s}$ due to a potential object motion, while for an eye-to-hand system (see the right part of Fig. 4) it represents the time variation of $\mathbf{s}$ due to a potential sensor motion.

As for the features Jacobian, in the eye-in-hand configuration, it can be decomposed as [7]

$$\mathbf{J}_s = \mathbf{L}_s {}^c\mathbf{V}_e \mathbf{J}(\mathbf{q}) \quad (4)$$

where

- $\mathbf{L}_s$ is the **interaction matrix** of $\mathbf{s}$ defined such that

$$\dot{\mathbf{s}} = \mathbf{L}_s \mathbf{v} \quad (5)$$

where $\mathbf{v} \in se_3$ is the relative velocity between the camera and the environment expressed in the camera frame. More details on how to determine this matrix are given below.

- ${}^c\mathbf{V}_e$ is the spatial motion transform matrix from the vision sensor to the end-effector. It is given by [27].

$${}^c\mathbf{V}_e = \begin{bmatrix} {}^c\mathbf{R}_e & [{}^c\mathbf{t}_e]_{\times} & {}^c\mathbf{R}_e \\ \mathbf{0} & & {}^c\mathbf{R}_e \end{bmatrix} \quad (6)$$

where ${}^c\mathbf{R}_e$ and ${}^c\mathbf{t}_e$ are respectively the rotation matrix and the translation vector between the sensor frame and the end-effector frame, and where $[{}^c\mathbf{t}_e]_{\times}$ is the skew symmetric matrix associated to ${}^c\mathbf{t}_e$. Matrix ${}^c\mathbf{V}_e$ is constant when the vision sensor is rigidly attached to the end-effector, which is usually the case. Thanks to the robustness of closed-loop control schemes with respect to calibration errors, a coarse approximation of ${}^c\mathbf{R}_e$ and ${}^c\mathbf{t}_e$ is generally sufficient in practice to serve as a satisfactory estimation of ${}^c\mathbf{V}_e$ to be injected in the control law. If needed, an accurate estimation is possible through classical hand-eye calibration methods [49].

- $\mathbf{J}(\mathbf{q})$ is the robot Jacobian such that $\mathbf{v}_e = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$ where $\mathbf{v}_e$ is the robot end-effector velocity.

In the eye-to-hand configuration, the features Jacobian $\mathbf{J}_s$ is composed of

$$\mathbf{J}_s = -\mathbf{L}_s {}^c\mathbf{V}_f {}^f\mathbf{V}_e \mathbf{J}(\mathbf{q}) \quad (7)$$

where

- ${}^f\mathbf{V}_e$ is the spatial motion transform matrix from the robot reference frame to the end-effector frame. It is known from the robot kinematics model.
- ${}^c\mathbf{V}_f$ is the spatial motion transform matrix from the camera frame to the reference frame. It is constant as long as the camera does not move. In that case, similarly as for the eye-in-hand configuration, a coarse approximation of ${}^c\mathbf{R}_f$ and ${}^c\mathbf{t}_f$ is usually sufficient.

3 Modeling Visual Features

A lot of works have concerned the modeling of the visual features and the determination of the analytical form of their interaction matrix. We consider the most classical ones in the following.

3.1 Image Point

Let us consider a static 3D point M with Cartesian coordinates (X, Y, Z) in the camera frame (see Fig. 5). Its perspective projection in the image plane results in a point m with normalized Cartesian coordinates $\mathbf{x} = (x, y)$ such that

$$x = X/Z, \quad y = Y/Z \quad (8)$$

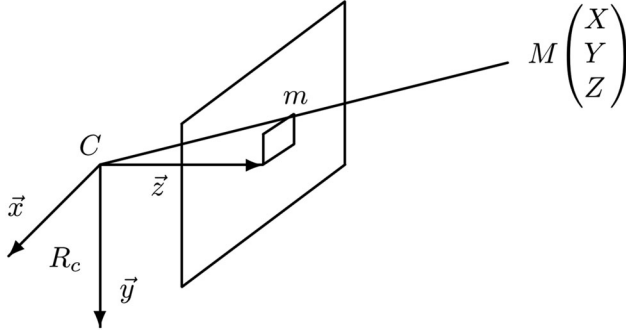


Fig. 5 Camera model

These normalized coordinates can be obtained from image measurements expressed in pixel units after a calibration step, or just using coarse approximations from the knowledge of some camera parameters (focal length and pixel size typically).

The velocity $\dot{\mathbf{x}}$ of the image point can be expressed with respect to the velocity $\mathbf{v}$ through the interaction matrix $\mathbf{L}_x$ given by [29] (see Exercise 3.3.1)

$$\mathbf{L}_x = \begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix} \quad (9)$$

where the three first columns contain the elements related to the three components $\mathbf{v} = (v_x, v_y, v_z)$ of the translational velocity, and where the three last columns contain the elements related to the three components $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$ of the rotational velocity. Denoting ${}^o\mathbf{R}_c$ the rotation matrix from the camera frame to any object frame, we recall that $\boldsymbol{\omega}$ is defined from the relation ${}^o\dot{\mathbf{R}}_c = {}^o\mathbf{R}_c [\boldsymbol{\omega}]_\times$, or similarly

$$[\boldsymbol{\omega}]_\times = {}^o\mathbf{R}_c^T {}^o\dot{\mathbf{R}}_c = -{}^o\dot{\mathbf{R}}_c^T {}^o\mathbf{R}_c \quad (10)$$

that comes from ${}^o\mathbf{R}_c^T {}^o\mathbf{R}_c = \mathbf{I}_3$ (from which we deduce ${}^o\mathbf{R}_c^T {}^o\dot{\mathbf{R}}_c + {}^o\dot{\mathbf{R}}_c^T {}^o\mathbf{R}_c = 0$, meaning ${}^o\mathbf{R}_c^T {}^o\dot{\mathbf{R}}_c$ is indeed a skew symmetric matrix).

By just changing the parameters representing the same image point, that is, by using the cylindrical coordinates defined by $\boldsymbol{\gamma} = (\rho, \theta)$ with $\rho = \sqrt{x^2 + y^2}$ and $\theta = \text{Arctan}(y/x)$, the interaction matrix of these parameters has a completely different form [24] (see Exercise 3.3.2):

$$\mathbf{L}_\gamma = \begin{bmatrix} -\cos \theta / Z & -\sin \theta / Z & \rho / Z (1 + \rho^2) \sin \theta & -(1 + \rho^2) \cos \theta & 0 \\ \sin \theta / (\rho Z) & -\cos \theta / (\rho Z) & 0 & \cos \theta / \rho & \sin \theta / \rho & -1 \end{bmatrix} \quad (11)$$

This implies that using the Cartesian coordinates or the cylindrical coordinates as visual features will induce a different behavior, that is, a different trajectory of the point in the image and, consequently, a different robot trajectory. The main objective in designing a visual servoing control scheme is thus to select the best set of visual features in terms of stability, global behavior (adequate trajectories both in the image plane and 3D space), and robustness to noise and to modeling and calibration errors from the task to be achieved, the environment observed, and the available image measurements. All these aspects can be studied from the interaction matrix of the potential visual features.

3.2 Other Visual Features

Currently, the analytical form of the interaction matrix is available for most basic features resulting from the perspective projection of simple geometrical primitives such as segments, straight lines, circles, spheres, and cylinders [16]. For instance, in the case of a straight line defined by

$$\begin{cases} A_1 X + B_1 Y + C_1 Z = 0 \\ A_2 X + B_2 Y + C_2 Z + D_2 = 0 \end{cases} \quad (12)$$

its projection in the image plane is a straight line $\mathcal{D}$ that can be parameterized by $\delta = (\rho, \theta)$ such that

$$g(x, y, \rho, \theta) = x \cos \theta + y \sin \theta - \rho = 0, \forall (x, y) \in \mathcal{D} \quad (13)$$

for which we have (see Exercise 3.3.3):

$$\mathbf{L}_\delta = \begin{bmatrix} \lambda_\rho \cos \theta & \lambda_\rho \sin \theta & -\lambda_\rho \rho & (1 + \rho^2) \sin \theta & -(1 + \rho^2) \cos \theta & 0 \\ \lambda_\theta \cos \theta & \lambda_\theta \sin \theta & -\lambda_\theta \rho & -\rho \cos \theta & -\rho \sin \theta & -1 \end{bmatrix} \quad (14)$$

where $\lambda_\rho = -(A\rho \cos \theta + B\rho \sin \theta + C)$ and $\lambda_\theta = B \cos \theta - A \sin \theta$ with $A = -A_2/D_2$, $B = -B_2/D_2$ and $C = -C_2/D_2$. As for image points, 3D parameters are involved in the translational components of $\mathbf{L}_\delta$, here through A , B and C that represents a 3D plane to which the straight line belongs.

The analytical form of the interaction matrix is also available for image moments related to planar and almost-planar objects of any shape [6], as well as for features selected from the epipolar geometry [45] and, of course, also for coordinates of 3D points, parameters of 3D geometrical primitives, and pose parameters, assuming these features are perfectly estimated. For instance, the interaction matrix $\mathbf{L}_\mathbf{X}$ of the coordinates $\mathbf{X} = (X, Y, Z)$ of any 3D point expressed in the camera frame is given by (see Exercise 3.3.4):

$$\mathbf{L}_\mathbf{X} = [-\mathbf{I}_3 \ [\mathbf{X}]_\times] \quad (15)$$

while the interaction matrix of the coordinates of the camera center expressed in the object frame, denoted ${}^o\mathbf{t}_c$ as for the translation between the camera frame and the object frame, is given by (see Exercise 3.3.5):

$$\mathbf{L}_{{}^o\mathbf{t}_c} = [{}^o\mathbf{R}_c \ \mathbf{0}_3] \quad (16)$$

As for rotational parameters, the best representation to be used in control is the $\theta\mathbf{u}$ parameters where $\mathbf{u}$ is the unitary axis of rotation and θ the angle of rotation around this axis. Indeed, by considering the $\theta\mathbf{u}$ parameters related to the rotation matrix ${}^c\mathbf{R}_c$ between the desired and the current camera frames, the corresponding interaction matrix is given by [31]:

$$\mathbf{L}_{\theta\mathbf{u}} = [\mathbf{0}_3 \ \mathbf{L}_\omega] \quad (17)$$

where

$$\mathbf{L}_\omega = \mathbf{I}_3 + \frac{\theta}{2}[\mathbf{u}]_\times + \left(1 - \frac{\text{sinc}\theta}{\text{sinc}^2(\theta/2)}\right)[\mathbf{u}]_\times^2 \quad (18)$$

which is singular only when $\theta = \pm 2\pi$.

In the recent years, following the seminal works presented in [13, 41], a new trend has concerned the use of direct image content as input of the control scheme [9, 20]. The main objective of these works is to avoid the extraction, tracking and matching of geometrical measurements, such as points of interest or edges, so that the system is extremely accurate and robust with respect to image processing errors. The basic idea is to consider the intensity of a set of pixels as visual features ($\mathbf{s} = \mathbf{I}$). From the classical assumption in computer vision stating that the intensity level of a moving point does not change (i.e., $I(\mathbf{x}, t) = I(\mathbf{x} + \mathbf{dx}, t + dt)$), it is possible to determine the interaction matrix corresponding to the intensity level of a pixel:

$$\mathbf{L}_I = -\left[\frac{\partial I}{\partial x} \ \frac{\partial I}{\partial y}\right] \mathbf{L}_\mathbf{x} \quad (19)$$

where $[\frac{\partial I}{\partial x} \ \frac{\partial I}{\partial y}]$ is the spatial gradient of the intensity along the x and y directions (see Exercise 3.3.6). Proceeding so leads to control a highly nonlinear system, with the drawback of a relatively small convergence domain, and, in general, not expected robot trajectory, plus the potential issue of robustness with respect to lighting variations. That is why the idea of direct photometric visual servoing has been expended by either considering other objective functions than $\|\mathbf{I} - \mathbf{I}^*\|$, such as the mutual information between the current and desired images [12], or other global image representations [11, 15], or by designing photo-geometric visual features [3].

All the works mentioned above have considered a classical vision sensor modeled by a perspective projection. It is possible to generalize the approach to any sort of sensors, such as omnidirectional cameras [5, 18], RGB-D sensors [47], the coupling between a camera and structured light [38, 43], and even 2D echographic

probes [35]. A large variety of visual features is thus available for many vision sensors.

Finally, methods also exist to estimate off-line or online a numerical value of the interaction matrix, by using neural networks for instance [46, 51] or the Broyden update [22, 25]. These methods are useful when the analytical form of the interaction matrix cannot be determined, which is usually the case when manipulating soft objects [40]. However, with such pure numerical estimations, no a priori analysis of the properties of the system is unfortunately possible.

3.3 Exercises

3.3.1 Compute $\mathbf{L}_x$

Determine the analytical form of $\mathbf{L}_x$ given in (9)

Solution By differentiating (8), we have

$$\dot{x} = \frac{1}{Z}\dot{X} - \frac{X}{Z^2}\dot{Z}, \quad \dot{y} = \frac{1}{Z}\dot{Y} - \frac{Y}{Z^2}\dot{Z} \quad (20)$$

The velocity of the point coordinates can be related to the velocity of the frame in which it is expressed thanks to the basic kinematic equation,

$$\dot{\mathbf{X}} = -\mathbf{v} - \boldsymbol{\omega} \times \mathbf{X} \quad (21)$$

that is

$$\dot{\mathbf{X}} = -\mathbf{v} - \boldsymbol{\omega} \times \mathbf{X} \Leftrightarrow \begin{cases} \dot{X} = -v_x + Y\omega_z - Z\omega_y \\ \dot{Y} = -v_y + Z\omega_x - X\omega_z \\ \dot{Z} = -v_z + X\omega_y - Y\omega_x \end{cases} \quad (22)$$

By injecting (22) in (20), we obtain

$$\begin{cases} \dot{x} = -\frac{1}{Z}v_x + \frac{Y}{Z}\omega_z - \omega_y + \frac{X}{Z^2}v_z - \frac{X^2}{Z^2}\omega_y + \frac{XY}{Z^2}\omega_x \\ \dot{y} = -\frac{1}{Z}v_y + \omega_x - \frac{X}{Z}\omega_z + \frac{Y}{Z^2}v_z - \frac{XY}{Z^2}\omega_y + \frac{Y^2}{Z^2}\omega_x \end{cases}$$

and (9) is finally found by replacing $\frac{X}{Z}$ by x and $\frac{Y}{Z}$ by y thanks to the perspective projection (8).

3.3.2 Compute $\mathbf{L}_\gamma$

Determine the analytical form of $\mathbf{L}_\gamma$ given in (11)

Solution From the definition of $\rho = \sqrt{x^2 + y^2}$ and $\theta = \text{Arctan}(y/x)$, we deduce, remembering that $\frac{d}{dt} \text{Arctan} z = \frac{\dot{z}}{1+z^2}$,

$$\dot{\rho} = \frac{1}{\rho}(x\dot{x} + y\dot{y}), \quad \dot{\theta} = \frac{x\dot{y} - y\dot{x}}{x^2 + y^2}$$

By injecting in the above equations $\dot{x} = \mathbf{L}_x \mathbf{v}$ and $\dot{y} = \mathbf{L}_y \mathbf{v}$ where $\mathbf{L}_x$ and $\mathbf{L}_y$ are given in (9), we obtain the analytical form given in (11) once x and y are replaced by $x = \rho \cos \theta$ and $y = \rho \sin \theta$.

3.3.3 Compute $\mathbf{L}_\delta$

Determine the analytical form of $\mathbf{L}_\delta$ given in (14)

Solution Whatever the camera motion, the image of a straight line is a straight line (by reducing the workspace to exclude the degenerate case where the camera optical center belongs to the line for which the image of the straight line is a single point). Remembering (13), we thus have

$$g(x, y, \rho, \theta) = 0, \quad \forall t$$

from which we deduce $\dot{g} = 0$, leading to

$$\frac{\partial g}{\partial x} \dot{x} + \frac{\partial g}{\partial y} \dot{y} + \frac{\partial g}{\partial \rho} \dot{\rho} + \frac{\partial g}{\partial \theta} \dot{\theta} = 0, \quad \forall (x, y) \in \mathcal{D}$$

and thus, from the form of g given by (13)

$$\dot{\rho} + (x \sin \theta - y \cos \theta) \dot{\theta} = \cos \theta \dot{x} + \sin \theta \dot{y}, \quad \forall (x, y) \in \mathcal{D}$$

This equation exhibits the two unknowns $\dot{\rho}$ and $\dot{\theta}$ we are looking for. It is valid for any point belonging to the line, so by selecting for instance the two following points: $(x_1, y_1) = (\rho \cos \theta, \rho \sin \theta)$ and $(x_2, y_2) = (\rho \cos \theta + \sin \theta, \rho \sin \theta - \cos \theta)$, we will obtain a linear system of two equations with two unknowns. Using the first point, we obtain as first equation

$$\dot{\rho} = \cos \theta \dot{x}_1 + \sin \theta \dot{y}_1$$

which is equivalent to

$$\mathbf{L}_\rho = \cos \theta \mathbf{L}_{x_1} + \sin \theta \mathbf{L}_{y_1}$$

We have from (9)

$$\begin{aligned} \mathbf{L}_{x_1} &= \begin{bmatrix} -1/Z_1 & 0 & x_1/Z_1 & x_1 y_1 & -(1+x_1^2) & y_1 \end{bmatrix} \\ \mathbf{L}_{y_1} &= \begin{bmatrix} 0 & -1/Z_1 & y_1/Z_1 & 1+y_1^2 & -x_1 y_1 & -x_1 \end{bmatrix} \end{aligned}$$

For any point in the image belonging to the line, the depth of the corresponding 3D point can be obtained from the second equation of (12). More precisely, its inverse is given by

$$1/Z = Ax + By + C$$

where $A = -A_2/D_2$, $B = -B_2/D_2$ and $C = -C_2/D_2$. For the point we consider, it is nothing but

$$1/Z_1 = Ax_1 + By_1 + C = A\rho \cos \theta + B\rho \sin \theta + C$$

By denoting $\lambda_\rho = -(A\rho \cos \theta + B\rho \sin \theta + C)$ and by further replacing x_1 by $\rho \cos \theta$ and y_1 by $\rho \sin \theta$, the expression of $\mathbf{L}_{x_1}$ and $\mathbf{L}_{y_1}$ transforms to

$$\begin{aligned} \mathbf{L}_{x_1} &= \begin{bmatrix} \lambda_\rho & 0 & -\lambda_\rho \rho \cos \theta & \rho^2 \cos \theta \sin \theta & -(1+\rho^2 \cos^2 \theta) & \rho \sin \theta \end{bmatrix} \\ \mathbf{L}_{y_1} &= \begin{bmatrix} 0 & \lambda_\rho & -\lambda_\rho \rho \sin \theta & 1+\rho^2 \sin^2 \theta & -\rho^2 \cos \theta \sin \theta & -\rho \cos \theta \end{bmatrix} \end{aligned}$$

from which we deduce

$$\mathbf{L}_\rho = \begin{bmatrix} \lambda_\rho \cos \theta & \lambda_\rho \sin \theta & -\lambda_\rho \rho & (1+\rho^2) \sin \theta & -(1+\rho^2) \cos \theta & 0 \end{bmatrix}$$

Similarly, by using the second point $(x_2, y_2) = (\rho \cos \theta + \sin \theta, \rho \sin \theta - \cos \theta)$, we obtain

$$\mathbf{L}_\rho + \mathbf{L}_\theta = \cos \theta \mathbf{L}_{x_2} + \sin \theta \mathbf{L}_{y_2}$$

where

$$\begin{aligned} \mathbf{L}_{x_2} &= \begin{bmatrix} -1/Z_2 & 0 & x_2/Z_2 & x_2 y_2 & -(1+x_2^2) & y_2 \end{bmatrix} \\ \mathbf{L}_{y_2} &= \begin{bmatrix} 0 & -1/Z_2 & y_2/Z_2 & 1+y_2^2 & -x_2 y_2 & -x_2 \end{bmatrix} \end{aligned}$$

and

$$1/Z_2 = -\lambda_\rho + A \sin \theta - B \cos \theta$$

By denoting $\lambda_\theta = B \cos \theta - A \sin \theta$, we obtain after basic but tedious developments

$$\mathbf{L}_\theta = \begin{bmatrix} \lambda_\theta \cos \theta & \lambda_\theta \sin \theta & -\lambda_\theta \rho & -\rho \cos \theta & -\rho \sin \theta & 0 \end{bmatrix}$$

By concatenating $\mathbf{L}_\rho$ and $\mathbf{L}_\theta$, we find hopefully the form of $\mathbf{L}_\delta$ given in (14).

3.3.4 Compute $\mathbf{L}_X$

Determine the analytical form of $\mathbf{L}_X$ given in (15)

Solution The solution is directly given from Eq. (22), noticing that $-\boldsymbol{\omega} \times \mathbf{X} = \mathbf{X} \times \boldsymbol{\omega} = [\mathbf{X}]_{\times} \boldsymbol{\omega}$.

3.3.5 Compute $\mathbf{L}^{o\mathbf{t}_c}$

Determine the analytical form of $\mathbf{L}^{o\mathbf{t}_c}$ given in (16)

Solution From the well known form of the inverse homegenous matrix representing any change of frames:

$${}^c\mathbf{M}_o^{-1} = {}^o\mathbf{M}_c = \begin{bmatrix} {}^o\mathbf{R}_c & {}^o\mathbf{t}_c \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} {}^c\mathbf{R}_o^T & -{}^c\mathbf{R}_o^T {}^c\mathbf{t}_o \\ \mathbf{0} & 1 \end{bmatrix}$$

we obtain ${}^o\mathbf{t}_c = -{}^c\mathbf{R}_o^T {}^c\mathbf{t}_o$, from which we deduce ${}^o\dot{\mathbf{t}}_c = -{}^c\dot{\mathbf{R}}_o^T {}^c\mathbf{t}_o - {}^c\mathbf{R}_o^T {}^c\dot{\mathbf{t}}_o$. Remembering (10) and (21) for which we select ${}^c\mathbf{t}_o$ as for $\mathbf{X}$, we obtain

$$\begin{aligned} {}^o\dot{\mathbf{t}}_c &= -{}^o\mathbf{R}_c [\boldsymbol{\omega}]_{\times} {}^c\mathbf{t}_o - {}^c\mathbf{R}_o^T (-\mathbf{v} - \boldsymbol{\omega} \times {}^c\mathbf{t}_o) \\ &= -{}^o\mathbf{R}_c [\boldsymbol{\omega}]_{\times} {}^c\mathbf{t}_o + {}^o\mathbf{R}_c \mathbf{v} + {}^o\mathbf{R}_c [\boldsymbol{\omega}]_{\times} {}^c\mathbf{t}_o \\ &= {}^o\mathbf{R}_c \mathbf{v} \end{aligned}$$

3.3.6 Compute $\mathbf{L}_I$

Determine the analytical form of $\mathbf{L}_I$ given in (19)

Solution The relation $I(\mathbf{x}, t) = I(\mathbf{x} + d\mathbf{x}, t + dt)$ is equivalent to $\frac{dI}{dt} = 0$. Since

$$\frac{dI}{dt} = \frac{\partial I}{\partial \mathbf{x}} \dot{\mathbf{x}} + \frac{\partial I}{\partial t}$$

we obtain

$$\frac{\partial I}{\partial t} = -\frac{\partial I}{\partial \mathbf{x}} \dot{\mathbf{x}} = -\frac{\partial I}{\partial \mathbf{x}} \mathbf{L}_X \mathbf{v}$$

4 Control

Once the modeling step has been performed, the design of the control scheme can be quite simple for holonomic robots.

4.1 Basics

At first, let us consider an eye-in-hand system such that the output of the control law is nothing but the camera velocity $\mathbf{v}_c$. Let us design the control law so that the system reacts as a decoupled first-order system, that is:

$$\dot{\mathbf{e}} = -\lambda \mathbf{e} \quad (23)$$

with $\mathbf{e} = \mathbf{s} - \mathbf{s}^*$ as defined in Eq. (1), and λ a positive scalar value. Note that this specification implies strong constraints on the desired behavior of the system. In fact, it imposes “straight line” trajectories in the “feature space” (see Exercise 4.4.1).

In case $\mathbf{s}^*$ is constant and the observed object is motionless, we have from (5):

$$\dot{\mathbf{e}} = \dot{\mathbf{s}} = \mathbf{L}_s \mathbf{v}_c \quad (24)$$

We deduce from (23) and (24) the ideal control law:

$$\mathbf{v}_c = -\lambda \mathbf{L}_s^+ \mathbf{e} = -\lambda \mathbf{L}_s^+ (\mathbf{s} - \mathbf{s}^*) \quad (25)$$

where $\mathbf{L}_s^+$ is the Moore-Penrose pseudoinverse of $\mathbf{L}_s$. It is ideal in the sense that, in case the robot is able to exactly apply the computed $\mathbf{v}_c$, and in case we have $\mathbf{L}_s \mathbf{L}_s^+ = \mathbf{I}$, then the system acts exactly as expected:

$$\dot{\mathbf{e}} = \mathbf{L}_s \mathbf{v}_c = -\lambda \mathbf{L}_s \mathbf{L}_s^+ \mathbf{e} = -\lambda \mathbf{e} \quad (26)$$

Unfortunately, we are usually not in this ideal case in practice. First, the property $\mathbf{L}_s \mathbf{L}_s^+ = \mathbf{I}$ is ensured if and only if the number k of features selected in $\mathbf{s}$ is at max 6 (note that $\mathbf{L}_s$ is a $k \times 6$ matrix) and $\mathbf{L}_s$ is of full rank k . Then, the interaction matrix contains some a priori unknown terms, such as the depth of the points in case image points are used as visual features in $\mathbf{s}$ (see (9) and (11)), or terms coming from measurements or estimations (see again (9) and (11), but also (15), (16), etc). An estimation or an approximation $\widehat{\mathbf{L}_s^+}$ has thus to be involved in the control scheme, leading to:

$$\mathbf{v}_c = -\lambda \widehat{\mathbf{L}_s^+} \mathbf{e} = -\lambda \widehat{\mathbf{L}_s^+} (\mathbf{s} - \mathbf{s}^*) \quad (27)$$

The natural question that arises is then the following: if it is not possible to perfectly achieve the expected behavior specified in (23), what are the possible choices for $\widehat{\mathbf{L}}_s^+$ so that the system converges? This is a classical problem in control and the famous Lyapunov theory allows answering this question. Let us first consider as Lyapunov function candidate the error $\mathcal{L} = \frac{1}{2}\|\mathbf{e}\|^2$. We have:

$$\dot{\mathcal{L}} = \mathbf{e}^T \dot{\mathbf{e}} = \mathbf{e}^T \mathbf{L}_s \mathbf{v}_c \quad (28)$$

Assuming now that the robot is able to perform perfectly the computed camera velocity (27), we obtain:

$$\dot{\mathcal{L}} = -\lambda \mathbf{e}^T \mathbf{L}_s \widehat{\mathbf{L}}_s^+ \mathbf{e} \quad (29)$$

The Lyapunov function $\mathcal{L}$ will thus decrease and the system will converge so that $\mathbf{e} = 0$, i.e., $\mathbf{s} = \mathbf{s}^*$ if the following sufficient condition is ensured:

$$\mathbf{L}_s \widehat{\mathbf{L}}_s^+ > 0, \quad \forall t \quad (30)$$

Once again, it is at least necessary that $k \leq 6$, and $\mathbf{L}_s$ and $\widehat{\mathbf{L}}_s^+$ are of full rank k . If this is the case, this condition gives an indication about the acceptable approximations in $\widehat{\mathbf{L}}_s^+$.

In case the number k of visual features is more than 6, which is usual in IBVS for controlling the 6 camera dof, selecting as Lyapunov function candidate $\mathcal{L} = \frac{1}{2}\|\widehat{\mathbf{L}}_s^+ \mathbf{e}\|^2$ allows obtaining another sufficient condition for local asymptotic stability (LAS):

$$\widehat{\mathbf{L}}_s^+ \mathbf{L}_s > 0, \quad \forall t \quad (31)$$

In that case, only LAS can be ensured, due to necessary approximations in the derivation of $\dot{\mathcal{L}}$ (see Exercise 4.4.2). It means that, if (31) is ensured, we are sure that the system will converge to the desired configuration if the initial configuration is “near” the desired one. Determining what “near” means is still an open problem. In case (30) is ensured, we have the global asymptotic stability (GAS) property, which means the system will converge whatever the initial configuration. This is really a stronger result, even if in practice, we have also to ensure that the object does not leave the camera field of view (in which case it would not be possible to compute the current value of the visual features), that the robot does not reach a joint limit (in which case the robot would not be able to apply the computed camera velocity), etc.

From (30) and (31) come the usual choices for $\widehat{\mathbf{L}}_s^+$. They are:

- $\widehat{\mathbf{L}}_s^+ = \mathbf{L}_s^{-1}{}_{(s)}$ when $\mathbf{L}_s$ is full rank and invertible (i.e., $k = 6$) and only depends of $\mathbf{s}$. This is the usual case in PBVS, ensuring the global asymptotic stability of

the system since $\mathbf{L}_s \widehat{\mathbf{L}}_s^+ = \mathbf{I}_6 > 0$. However, notice that $\mathbf{s}$ is here composed of 3D parameters that have to be estimated at each iteration of the servo.

- $\widehat{\mathbf{L}}_s^+ = \mathbf{L}_s^+(\mathbf{s}, \widehat{\mathbf{Z}})$. In this case, $\widehat{\mathbf{L}}_s^+$ is computed at each iteration of the control scheme using the current value of $\mathbf{s}$ and an estimation of the current depth parameters $\widehat{\mathbf{Z}}$. Assuming these parameters are perfectly estimated, we have $\widehat{\mathbf{L}}_s^+ \mathbf{L}_s = \mathbf{I}_6$, which of course satisfies (31).
- $\widehat{\mathbf{L}}_s^+ = \mathbf{L}_s^+(\mathbf{s}^*, \widehat{\mathbf{Z}}^*)$. In this case, $\widehat{\mathbf{L}}_s^+$ is constant and has to be computed only once at the beginning of the servo. Furthermore, only the desired depth parameters $\widehat{\mathbf{Z}}^*$ have to be set, which means no varying 3D parameters have to be estimated on line. Assuming $\widehat{\mathbf{Z}}^* = \mathbf{Z}$, we have $\widehat{\mathbf{L}}_s^+ \mathbf{L}_s = \mathbf{I}_6$ when $\mathbf{s} = \mathbf{s}^*$, ensuring the local asymptotic stability of the system.
- $\widehat{\mathbf{L}}_s^+ = 2 \left(\mathbf{L}_s(\mathbf{s}, \widehat{\mathbf{Z}}) + \mathbf{L}_s(\mathbf{s}^*, \widehat{\mathbf{Z}}^*) \right)^+$. In that case, since $\mathbf{L}_s(\mathbf{s}, \widehat{\mathbf{Z}})$ is involved in $\widehat{\mathbf{L}}_s^+$, the current depth parameters have to be estimated and $\widehat{\mathbf{L}}_s^+$ computed at each iteration of the servo. Note that here again we have $\widehat{\mathbf{L}}_s^+ \mathbf{L}_s = \mathbf{I}_6$ when $\mathbf{s} = \mathbf{s}^*$.

4.2 A Typical Example

We illustrate the choices for $\mathbf{s}$ and $\widehat{\mathbf{L}}_s^+$ on a typical example (see also Exercise 4.4.3 where a simple 2 dof configuration is studied). The goal is to position the camera so that it observes a square as a centered square in the image (see Fig. 6). Note that the initial camera pose has been selected far away from the desired pose, particularly with regard to the rotational motions, which are known to be the most problematic for IBVS. In the simulations presented in the following, no noise or modeling errors have been introduced and the 3D parameters involved are assumed to be perfectly known in order to allow comparison of different behaviors in perfect conditions.

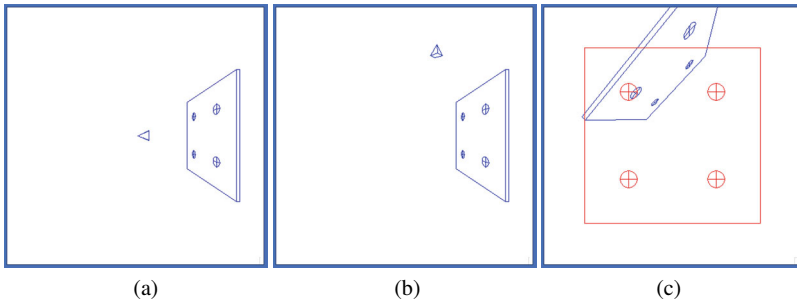


Fig. 6 Example of positioning task: (a) the desired camera pose w.r.t. a simple target, (b) the initial camera pose, (c) the initial and desired images of the target

The results of these simulations can be obtained easily using the ViSP¹ library [32]. Videos of experimental results using a camera mounted on a 6-dof robot arm are available at <http://handbookofrobotics.org/view-chapter/videodetails/34>.

4.2.1 IBVS

We first define $\mathbf{s}$ to include the x and y coordinates of the four image points forming the square: $\mathbf{s} = (x_1, y_1, \dots, x_4, y_4)$, $\mathbf{s}^* = (-a, a, a, a, -a, -a, -a, -a)$ where a is the half size of the square in the image plane. From (9), the interaction matrix has the following form:

$$\mathbf{L}_{\mathbf{s}(\mathbf{s}, \mathbf{Z})} = \begin{bmatrix} -1/Z_1 & 0 & x_1/Z_1 & x_1 y_1 & -(1+x_1^2) & y_1 \\ 0 & -1/Z_1 & y_1/Z_1 & 1+y_1^2 & -x_1 y_1 & -x_1 \\ -1/Z_2 & 0 & x_2/Z_2 & x_2 y_2 & -(1+x_2^2) & y_2 \\ 0 & -1/Z_2 & y_2/Z_2 & 1+y_2^2 & -x_2 y_2 & -x_2 \\ -1/Z_3 & 0 & x_3/Z_3 & x_3 y_3 & -(1+x_3^2) & y_3 \\ 0 & -1/Z_3 & y_3/Z_3 & 1+y_3^2 & -x_3 y_3 & -x_3 \\ -1/Z_4 & 0 & x_4/Z_4 & x_4 y_4 & -(1+x_4^2) & y_4 \\ 0 & -1/Z_4 & y_4/Z_4 & 1+y_4^2 & -x_4 y_4 & -x_4 \end{bmatrix}$$

while it has a simpler form at the desired configuration

$$\mathbf{L}_{\mathbf{s}(\mathbf{s}^*, \mathbf{Z}^*)} = \begin{bmatrix} -1/Z^* & 0 & -1/Z^* & -a^2 & -(1+a^2) & a \\ 0 & -1/Z^* & a/Z^* & 1+a^2 & a^2 & a \\ -1/Z^* & 0 & a/Z^* & a^2 & -(1+a^2) & a \\ 0 & -1/Z^* & a/Z^* & 1+a^2 & -a^2 & -a \\ -1/Z^* & 0 & a/Z^* & -a^2 & -(1+a^2) & -a \\ 0 & -1/Z^* & -a/Z^* & 1+a^2 & a^2 & -a \\ -1/Z^* & 0 & -a/Z^* & a^2 & -(1+a^2) & -a \\ 0 & -1/Z^* & -a/Z^* & 1+a^2 & -a^2 & a \end{bmatrix}$$

where Z^* is the desired depth of the four points.

The results obtained using $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = \mathbf{L}_{\mathbf{s}(\mathbf{s}, \mathbf{Z})}^+$ are given in Fig. 7. In this case, the trajectories of the points in the image are almost straight lines (not pure straight line since $\widehat{\mathbf{L}}_{\mathbf{s}}^+ \neq \mathbf{I}_8$). However, the behavior induced in 3D is not so satisfactory. The large camera velocities at the beginning of the servo indicate that the condition number of $\widehat{\mathbf{L}}_{\mathbf{s}}^+$ is high at the start of the trajectory, and the camera trajectory is far from a straight line.

¹ <https://visp.inria.fr>.

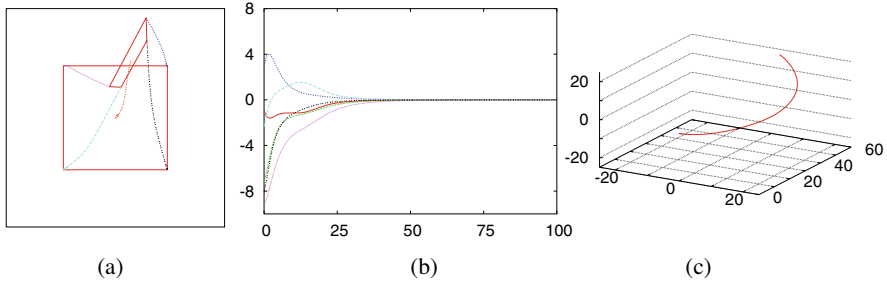


Fig. 7 IBVS system behavior using $\mathbf{s} = (x_1, y_1, \dots, x_4, y_4)$ and $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = \mathbf{L}_{\mathbf{s}}^+$ (a) image point trajectories including the trajectory of the center of the square, which is not used in the control scheme, (b) $\mathbf{v}_c$ components (cm/s and deg/s) computed at each iteration of the control scheme, and (c) the 3-D trajectory of the camera optical center expressed in the desired camera frame $\mathcal{R}_{c^*}$ (cm)

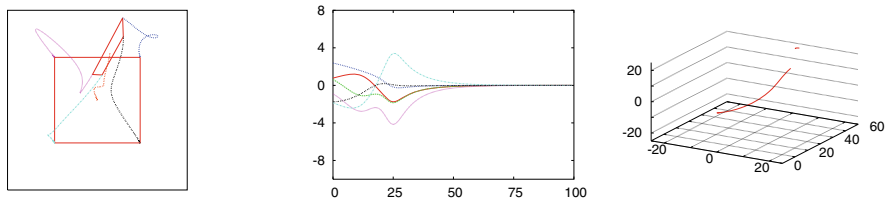


Fig. 8 IBVS system behavior using $\mathbf{s} = (x_1, y_1, \dots, x_4, y_4)$ and $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = \mathbf{L}_{\mathbf{s}^*}^+$

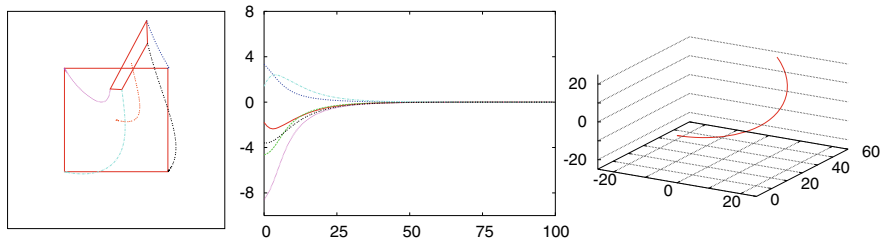


Fig. 9 IBVS system behavior using $\mathbf{s} = (x_1, y_1, \dots, x_4, y_4)$ and $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = 2(\mathbf{L}_{\mathbf{s}} + \mathbf{L}_{\mathbf{s}^*})^+$

The results obtained by using $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = \mathbf{L}_{\mathbf{s}^*}^+$ are shown in Fig. 8. Note that despite the large displacement that is required the system converges while the condition (31) is not ensured for the first thirty or so iterations. However, neither the behavior in the image, nor the computed camera velocity components, nor the 3D trajectory of the camera present desirable properties far from the convergence.

The choice $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = 2(\mathbf{L}_{\mathbf{s}(\mathbf{s}, \widehat{\mathbf{z}})} + \mathbf{L}_{\mathbf{s}(\mathbf{s}^*, \widehat{\mathbf{z}}^*)})^+$ provides good performance in practice. Indeed, as can be seen in Fig. 9, the camera velocity components do not include large oscillations, and provide a smooth trajectory in both the image and in 3D.

We now consider the case where the visual features are no more image points but the parameters related to the four straight lines forming the square (remember (13)). We thus select $\mathbf{s} = (\rho_1, \rho_2, \rho_3, \rho_4, \theta_1, \theta_2, \theta_3, \theta_4)$ from which we deduce $\mathbf{s}^* = (a, a, a, a, 0, \pi/2, \pi, 3\pi/2)$. From (14), we obtain

$$\mathbf{L}_{\mathbf{s}(\mathbf{s}, \mathbf{Z})} = \begin{bmatrix} \lambda_{\rho 1} \cos \theta_1 & \lambda_{\rho 1} \sin \theta_1 & -\lambda_{\rho 1} \rho_1 & (1 + \rho_1^2) \sin \theta_1 & -(1 + \rho_1^2) \cos \theta_1 & 0 \\ \lambda_{\rho 2} \cos \theta_2 & \lambda_{\rho 2} \sin \theta_2 & -\lambda_{\rho 2} \rho_2 & (1 + \rho_2^2) \sin \theta_2 & -(1 + \rho_2^2) \cos \theta_2 & 0 \\ \lambda_{\rho 3} \cos \theta_3 & \lambda_{\rho 3} \sin \theta_3 & -\lambda_{\rho 3} \rho_3 & (1 + \rho_3^2) \sin \theta_3 & -(1 + \rho_3^2) \cos \theta_3 & 0 \\ \lambda_{\rho 4} \cos \theta_4 & \lambda_{\rho 4} \sin \theta_4 & -\lambda_{\rho 4} \rho_4 & (1 + \rho_4^2) \sin \theta_4 & -(1 + \rho_4^2) \cos \theta_4 & 0 \\ \lambda_{\theta 1} \cos \theta_1 & \lambda_{\theta 1} \sin \theta_1 & -\lambda_{\theta 1} \rho_1 & -\rho_1 \cos \theta_1 & -\rho_1 \sin \theta_1 & -1 \\ \lambda_{\theta 2} \cos \theta_2 & \lambda_{\theta 2} \sin \theta_2 & -\lambda_{\theta 2} \rho_2 & -\rho_2 \cos \theta_2 & -\rho_2 \sin \theta_2 & -1 \\ \lambda_{\theta 3} \cos \theta_3 & \lambda_{\theta 3} \sin \theta_3 & -\lambda_{\theta 3} \rho_3 & -\rho_3 \cos \theta_3 & -\rho_3 \sin \theta_3 & -1 \\ \lambda_{\theta 4} \cos \theta_4 & \lambda_{\theta 4} \sin \theta_4 & -\lambda_{\theta 4} \rho_4 & -\rho_4 \cos \theta_4 & -\rho_4 \sin \theta_4 & -1 \end{bmatrix} \quad (32)$$

At the desired configuration, since $A = B = 0$, we have $\lambda_{\rho i} = -C = -1/Z^*$ and $\lambda_{\theta i} = 0$ for $i = 1$ to 4. Then, from the particular value of $\mathbf{s}^*$, we obtain a particularly simple and decoupled form for $\mathbf{L}_{\mathbf{s}(\mathbf{s}^*, \mathbf{Z}^*)}$:

$$\mathbf{L}_{\mathbf{s}(\mathbf{s}^*, \mathbf{Z}^*)} = \begin{bmatrix} -1/Z^* & 0 & a/Z^* & 0 & -(1 + a^2) & 0 \\ 0 & -1/Z^* & a/Z^* & (1 + a^2) & 0 & 0 \\ 1/Z^* & 0 & a/Z^* & 0 & (1 + a^2) & 0 \\ 0 & 1/Z^* & a/Z^* & -(1 + a^2) & 0 & 0 \\ 0 & 0 & 0 & -a & 0 & -1 \\ 0 & 0 & 0 & 0 & -a & -1 \\ 0 & 0 & 0 & a & 0 & -1 \\ 0 & 0 & 0 & 0 & a & -1 \end{bmatrix} \quad (33)$$

This explains that the behavior obtained using this new set of visual features and $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = \mathbf{L}_{\mathbf{s}^*}^+$ is quite satisfactory (see Fig. 10). Indeed, while a constant matrix is

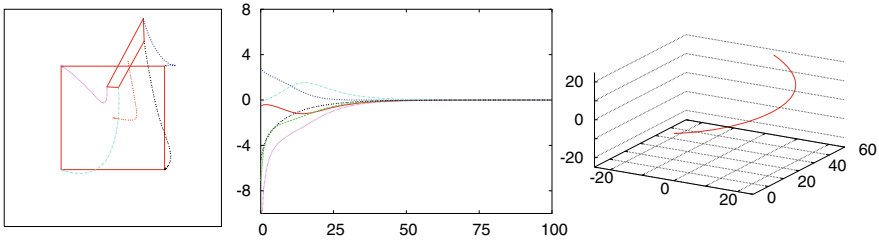


Fig. 10 IBVS system behavior using $\mathbf{s} = (\rho_1, \dots, \rho_4, \theta_1, \dots, \theta_4)$ and $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = \mathbf{L}_{\mathbf{s}^*}^+$

used for $\widehat{\mathbf{L}}_s^+$, which contains a single 3D parameter Z^* , the system behavior shown on Fig. 10 has the same nice properties than using $\widehat{\mathbf{L}}_s^+ = 2(\mathbf{L}_s + \mathbf{L}_{s^*})^+$ with $\mathbf{s} = (x_1, y_1, \dots, x_4, y_4)$.

4.2.2 PBVS

We now illustrate the behavior obtained with PBVS schemes, assuming that all pose parameters are perfectly estimated. We start by considering $\mathbf{s} = ({}^c\mathbf{t}_o, \theta\mathbf{u})$ where $\theta\mathbf{u}$ is the angle-axis parameterization of the rotation matrix ${}^c\mathbf{R}_c$. Note that this rotation matrix can be computed from ${}^c\mathbf{R}_o$ and ${}^c\mathbf{R}_o$ (${}^c\mathbf{R}_c = {}^c\mathbf{R}_o {}^c\mathbf{R}_o^T$) where ${}^c\mathbf{R}_o$ is obtained from a pose estimation on the current image, and ${}^c\mathbf{R}_o$ from a pose estimation on the desired image or directly if the desired pose has been set by specifying the task. Note that, from the choice of $\mathbf{s}$, we have $\mathbf{s}^* = ({}^c\mathbf{t}_o, \mathbf{0}) = (0, 0, Z^*, 0, 0, 0)$ in our example.

Since ${}^c\mathbf{t}_o$ is nothing but the coordinates of point O (the center of the square) expressed in the camera frame, we obtain from (15) and (17):

$$\mathbf{L}_{\mathbf{s}(\mathbf{s})} = \begin{bmatrix} -\mathbf{I}_3 & [{}^c\mathbf{t}_o]_{\times} \\ \mathbf{0}_3 & \mathbf{L}_{\omega} \end{bmatrix}$$

Matrix $\mathbf{L}_{\omega}$ is particular (see (18)): it is such that $\mathbf{L}_{\omega}\theta\mathbf{u} = \mathbf{L}_{\omega}^{-1}\theta\mathbf{u} = \theta\mathbf{u}$. By using $\widehat{\mathbf{L}}_s^+ = \mathbf{L}_s^{-1}(\mathbf{s})$, the control scheme (27) has thus the following form:

$$\begin{bmatrix} v_c \\ \omega_c \end{bmatrix} = -\lambda \begin{bmatrix} -\mathbf{I}_3 & [{}^c\mathbf{t}_o]_{\times} \mathbf{L}_{\omega}^{-1} \\ \mathbf{0}_3 & \mathbf{L}_{\omega}^{-1} \end{bmatrix} \begin{bmatrix} {}^c\mathbf{t}_o - {}^c\mathbf{t}_o^* \\ \theta\mathbf{u} \end{bmatrix} = \begin{bmatrix} -\lambda({}^c\mathbf{t}_o - {}^c\mathbf{t}_o^* + [{}^c\mathbf{t}_o]_{\times}\theta\mathbf{u}) \\ -\lambda\theta\mathbf{u} \end{bmatrix}$$

Ideally, that is if the pose parameters are perfectly estimated, the behavior of $\mathbf{e}$ will be the expected one ($\dot{\mathbf{e}} = -\lambda\mathbf{e}$) and the system will be GAS. The choice of $\mathbf{s}$ causes the rotational motion to follow a geodesic with an exponential decreasing speed and causes the translational parameters involved in $\mathbf{s}$ to decrease with the same speed. This explains the nice exponential decrease of the camera velocity components in Fig. 11. Furthermore, the trajectory in the image of point O follows a pure straight line since it corresponds to the perspective projection of a straight line (${}^c\mathbf{t}_o$ follows a pure exponential decrease, see Exercise 4.4.1). On the other hand, the camera trajectory does not follow a straight line since the camera is rotating during its motion.

Another PBVS scheme can be designed by using $\mathbf{s} = ({}^c\mathbf{t}_c, \theta\mathbf{u})$. In that case, we have $\mathbf{s}^* = \mathbf{0}$. By considering ${}^c\mathbf{t}_c$ instead of ${}^o\mathbf{t}_c$ in (16), and by using (17), we deduce

$$\mathbf{L}_{\mathbf{s}(\mathbf{s})} = \begin{bmatrix} {}^c\mathbf{R}_c & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{L}_{\omega} \end{bmatrix} \quad (34)$$

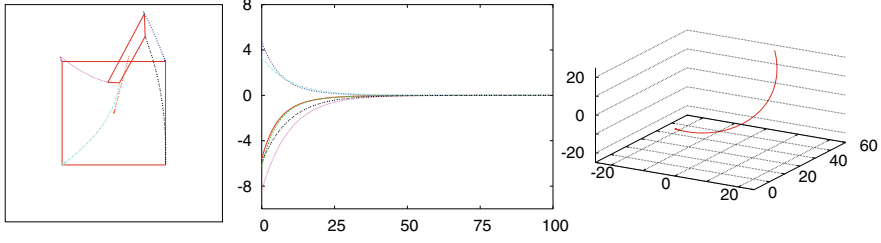


Fig. 11 PBVS system behavior using $\mathbf{s} = ({}^c \mathbf{t}_o, \theta \mathbf{u})$

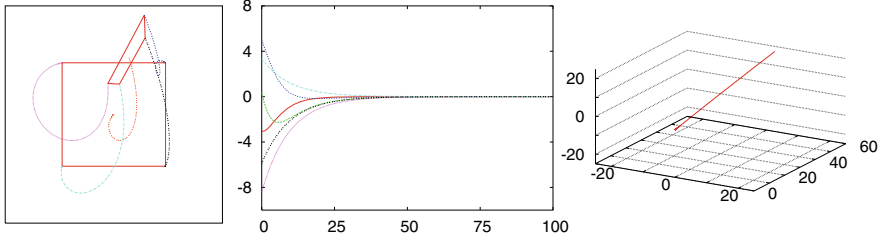


Fig. 12 PBVS system behavior using $\mathbf{s} = ({}^c {}^* \mathbf{t}_c, \theta \mathbf{u})$

Note the perfect decoupling between translational and rotational motions, which allows us to obtain a simple control scheme:

$$\begin{cases} \mathbf{v}_c = -\lambda {}^c \mathbf{R}_{c^*} {}^c {}^* \mathbf{t}_c \\ \boldsymbol{\omega}_c = -\lambda \theta \mathbf{u} \end{cases} \quad (35)$$

In this case, as can be seen in Fig. 12, if the pose parameters involved in (35) are perfectly estimated, the camera trajectory is a pure straight line, as expected since ${}^c {}^* \mathbf{t}_c$ is involved in $\mathbf{s}$, while the image trajectories are less satisfactory than before. Some particular configurations can even be found so that some points leave the camera field of view.

4.2.3 2 1/2 D VS

As already mentioned in Sect. 2, it may be interesting to combine 2D and 3D entities in the set of visual features $\mathbf{s}$. A first example was proposed in [31]. It consisted in selecting in $\mathbf{s}$ a first part $\mathbf{s}_t$ to control the translational components of the camera velocity and $\theta \mathbf{u}$ to control its rotational components: $\mathbf{s} = (\mathbf{s}_t, \theta \mathbf{u})$. More precisely, $\mathbf{s}_t$ is chosen as the coordinates $\mathbf{x}_g$ of the center of gravity of the target in the image, and

the logarithm of its depth: $\mathbf{s}_t = (\mathbf{x}_g, \log Z_g)$. With such a choice, the corresponding interaction matrix has the following form:

$$\mathbf{L}_{\mathbf{s}(\mathbf{s})} = \begin{bmatrix} \mathbf{L}_{s_t v} & \mathbf{L}_{s_t \omega} \\ \mathbf{0}_3 & \mathbf{L}_\omega \end{bmatrix}$$

where the first two rows of $\mathbf{L}_{s_t v}$ and $\mathbf{L}_{s_t \omega}$ are directly obtained from (9) while their third row comes from (15). More precisely, we have:

$$\mathbf{L}_{s_t v} = \frac{1}{Z_g} \begin{bmatrix} -1 & 0 & x_g \\ 0 & -1 & y_g \\ 0 & 0 & -1 \end{bmatrix} \text{ and } \mathbf{L}_{s_t \omega} = \begin{bmatrix} x_g y_g & -(1+x_g^2) & y_g \\ 1+y_g^2 & -x_g y_g & -x_g \\ -y_g & x_g & 0 \end{bmatrix}$$

Note the triangular form of $\mathbf{L}_{s_t v}$ and the same value on its diagonal. By using again $\widehat{\mathbf{L}}_{\mathbf{s}}^+ = \mathbf{L}_{\mathbf{s}}^{-1}(\mathbf{s})$, the control scheme (27) has now the following form:

$$\begin{bmatrix} v_c \\ \omega_c \end{bmatrix} = -\lambda \begin{bmatrix} \mathbf{L}_{s_t v}^{-1} & -\mathbf{L}_{s_t v}^{-1} \mathbf{L}_{s_t \omega} \\ \mathbf{0}_3 & \mathbf{L}_\omega^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{s}_t - \mathbf{s}_t^* \\ \theta \mathbf{u} \end{bmatrix} = \begin{bmatrix} -\lambda (\mathbf{L}_{s_t v}^{-1}(\mathbf{s}_t - \mathbf{s}_t^*) - \mathbf{L}_{s_t v}^{-1} \mathbf{L}_{s_t \omega} \theta \mathbf{u}) \\ -\lambda \theta \mathbf{u} \end{bmatrix}$$

The behavior obtained using this choice for $\mathbf{s}$ is given on Fig. 13. We can note the image trajectory of the center of gravity, which is a straight line as expected, and the nice decreasing of the camera velocity components, which makes this scheme very near from the first PBVS one.

Finally, selecting $\mathbf{s}$ as $\mathbf{s} = ({}^c \mathbf{t}_c, \mathbf{x}_g, \theta u_z)$ provides with a block-triangular interaction matrix of the form:

$$\mathbf{L}_{\mathbf{s}(\mathbf{s})} = \begin{bmatrix} {}^c \mathbf{R}_c & \mathbf{0}_3 \\ \mathbf{L}'_v & \mathbf{L}'_\omega \end{bmatrix}$$

where $\mathbf{L}'_v$ and $\mathbf{L}'_\omega$ can easily be computed from (9) and (17). This scheme is so that, in perfect conditions, the camera trajectory is a straight line (since ${}^c \mathbf{t}_c$ is a part of $\mathbf{s}$), and the image trajectory of the center of gravity of the object is also a straight line

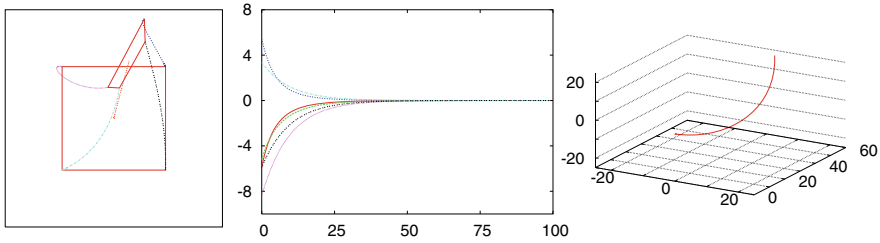


Fig. 13 2 1/2D VS system behavior using $\mathbf{s} = (\mathbf{x}_g, \log(Z_g), \theta \mathbf{u})$

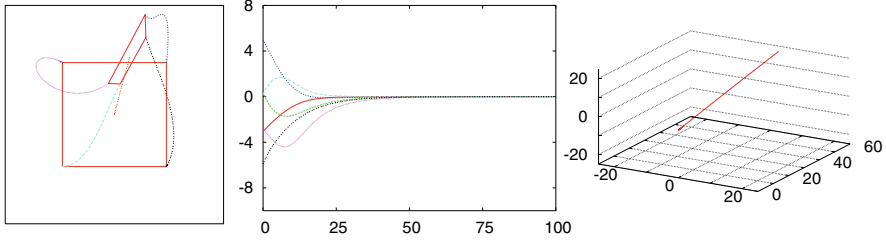


Fig. 14 2 1/2D VS system behavior using $\mathbf{s} = ({}^{c^*}\mathbf{t}_c, \mathbf{x}_g, \theta u_z)$

(since $\mathbf{x}_g$ is also a part of $\mathbf{s}$). The translational camera degrees of freedom are devoted to realize the 3D straight line, while the rotational camera degrees of freedom are devoted to realize the 2D straight line and compensate also the 2D motion of $\mathbf{x}_g$ due to the translational motion. As can be seen on Fig. 14, this scheme is particularly satisfactory in practice.

4.3 General Case

We now consider the robot joint velocities as outputs of the control scheme. By following the same reasoning as in Sect. 4.1 but using now the more general form (3) instead of (5), we obtain:

$$\dot{\mathbf{q}} = -\lambda \hat{\mathbf{J}}_s^+ \mathbf{e} + \hat{\mathbf{J}}_s^+ \frac{\partial \mathbf{s}^*}{\partial t} - \hat{\mathbf{J}}_s^+ \frac{\partial \hat{\mathbf{s}}}{\partial t} \quad (36)$$

where, in the first feedback term, λ is a positive (possibly varying) gain tuning the time-to-convergence of the system, and $\hat{\mathbf{J}}_s^+$ is the Moore-Penrose pseudoinverse of an approximation or an estimation of the features Jacobian. As already said above for the interaction matrix, the exact value of all its elements is indeed generally unknown since it depends of the intrinsic and extrinsic camera parameters, as well as of some 3D parameters such as the depth of the point in Eqs. (9) and (11). Methods for estimating these 3D parameters exist, either using the knowledge of the robot motion [14], or the knowledge of the 3D object model when it is available [33], or, up to a scalar factor, from partial pose estimation using the properties of the epipolar geometry between the current and the desired images [31].

The second term of the control scheme anticipates for the variation of $\mathbf{s}^*$ in the case of a varying desired value. The third term compensates as much as possible a possible target motion in the eye-in-hand case and a possible camera motion in the eye-to-hand case. They are both null in the case of a fixed desired value and a motionless target or camera. They serve as feedforward terms for removing the tracking error in the other cases [10].

Following the Lyapunov theory, the stability of the system can be studied [7]. Generally, as already said, visual servoing schemes can be demonstrated to be LAS if the errors introduced in $\hat{\mathbf{J}}_s$ are not too strong. Some particular visual servoing schemes can be demonstrated to be GAS under similar conditions. This is for instance the case for the pan-tilt camera control depicted on Fig. 2, for PBVS assuming the 3D parameters involved are perfectly estimated, and for well-designed IBVS schemes.

Finally, when the visual features do not constrain all the dof, it is possible to combine the visual task with supplementary tasks such as, for instance, joint limits avoidance or the visibility constraint (to be sure that the target considered will always remain in the camera field of view). In that case, the redundancy framework [39] can be applied and the new error to be regulated to zero has the following form:

$$\mathbf{e}_n = \hat{\mathbf{J}}_s^+ \mathbf{e} + (\mathbf{I} - \hat{\mathbf{J}}_s^+ \hat{\mathbf{J}}_s) \mathbf{e}_2 \quad (37)$$

where $(\mathbf{I} - \hat{\mathbf{J}}_s^+ \hat{\mathbf{J}}_s)$ is a projection operator on the null space of the visual task $\mathbf{e}$ so that the supplementary task $\mathbf{e}_2$ will be achieved at best under the constraint that it does not perturb the visual task. A similar control scheme to (36) is now given by

$$\dot{\mathbf{q}} = -\lambda \mathbf{e}_n - \frac{\partial \mathbf{e}_n}{\partial t} \quad (38)$$

This scheme has for instance been applied for the first example depicted on Fig. 16 where the rotational motion of the mobile robot is controlled by vision while its translational motion is controlled by the odometry to move at a constant velocity.

Any other more advanced control strategy can be applied such as optimal control [21, 42], coupling path planning and visual servoing [8, 26, 37], model predictive control [2, 17], or quadratic programming [1] when visual tasks and visual constraints have to be simultaneously handled with other tasks and constraints. Particular care has to be considered for underactuated and nonholonomic systems for which adequate control laws have to be designed [19, 30, 34, 36].

4.4 Exercises

4.4.1 An Exponential Decrease Ensures Straight Line Trajectories

Solution The solution of the differential equation (23) is

$$\mathbf{e}(t) = \mathbf{e}(0) \exp^{-\lambda t}$$

Let us consider that the visual features are composed of the Cartesian coordinates of an image point, $\mathbf{s} = (x, y)$, with desired value $\mathbf{s}^* = (x^*, y^*)$. After expanding the

above equation for each component of $\mathbf{e}$, we obtain

$$\exp^{-\lambda t} = \frac{x(t) - x^*}{x(0) - x^*} = \frac{y(t) - y^*}{y(0) - y^*}$$

from which we deduce

$$ax(t) + by(t) + c = 0, \quad \forall t$$

with $a = 1/(x(0) - x^*)$, $b = -1/(y(0) - y^*)$, and $c = y^*/(y(0) - y^*) - x^*/(x(0) - x^*)$. The point will thus move along the straight line to which $(x(0), y(0))$ and (x^*, y^*) belong.

This result can of course be generalised to any feature, such as for instance the coordinates of a 3D point that will follow a 3D straight line in the frame in which they are expressed. In case this frame moves, the trajectory achieved will not be a straight line in a static Euclidean frame. Similarly, a straight line behavior for the cylindrical coordinates of an image point will not produce a straight line in the Cartesian image frame.

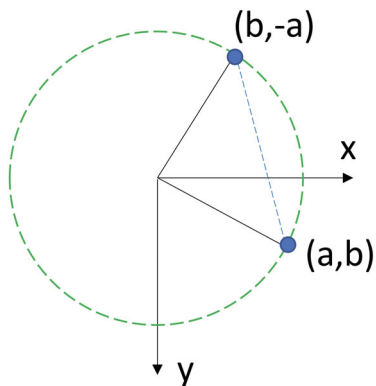
4.4.2 Local Asymptotic Stability with $\mathcal{L} = \frac{1}{2} \|\hat{\mathbf{L}}_s^+ \mathbf{e}\|^2$

Solution By denoting $\mathbf{e}_2 = \hat{\mathbf{L}}_s^+ \mathbf{e}$, the time derivative of $\mathcal{L}$ is

$$\begin{aligned} \dot{\mathcal{L}} &= \mathbf{e}_2^T \dot{\mathbf{e}}_2 \\ &= \mathbf{e}_2^T \hat{\mathbf{L}}_s^+ \dot{\mathbf{e}} + \mathbf{e}_2^T \hat{\dot{\mathbf{L}}}_s^+ \mathbf{e} \\ &= \mathbf{e}_2^T \hat{\mathbf{L}}_s^+ \mathbf{L}_s \mathbf{v}_c + \mathbf{e}_2^T \hat{\dot{\mathbf{L}}}_s^+ \mathbf{e} \\ &= -\lambda \mathbf{e}_2^T \hat{\mathbf{L}}_s^+ \mathbf{L}_s \hat{\mathbf{L}}_s^+ \mathbf{e} + \mathbf{e}_2^T \hat{\dot{\mathbf{L}}}_s^+ \mathbf{e} \\ &= -\lambda \mathbf{e}_2^T \hat{\mathbf{L}}_s^+ \mathbf{L}_s \mathbf{e}_2 + \mathbf{e}_2^T \hat{\dot{\mathbf{L}}}_s^+ \mathbf{e} \end{aligned}$$

In case $\hat{\mathbf{L}}_s^+$ is constant, the second term vanishes, which allows exhibiting condition (31). Note that only the local asymptotic stability can be ensured since we have $\mathcal{L} = 0$ with $\mathbf{e} \neq 0$ when $\mathbf{e}$ belongs to the nullspace of $\hat{\mathbf{L}}_s^+$, which may occur for some particular configurations. In case $\hat{\mathbf{L}}_s^+$ is not constant, the term $\hat{\dot{\mathbf{L}}}_s^+$ is too complex to be considered and, by restricting the analysis to the linearized system around $\mathbf{e} = 0$, condition (31) still remains for ensuring the local asymptotic stability. More details are given in [7].

Fig. 15 Initial and desired configurations for an image point



4.4.3 Behavior Analysis in a Simple 2 dof Configuration

Solution Let us consider that the camera has two degrees of freedom: the translation along the optical axis and the rotation around the optical axis. It observes a point with Cartesian coordinates (a, b) in the image while its desired position is $(b, -a)$, which corresponds to a rotation of $\pi/2$ rad around the optical axis (see Fig. 15).

By selecting the Cartesian coordinates of the image point as visual features: $\mathbf{s} = (x, y)$, and by reducing (9) to the two available degrees of freedom, i.e., its third and sixth columns, we obtain

$$\mathbf{L}_s = \begin{bmatrix} x/Z & y \\ y/Z & -x \end{bmatrix}$$

where Z is the depth of the observed point. $\mathbf{L}_s$ is a 2×2 full-rank matrix (apart when $x = y = 0$) whose inverse is given by

$$\mathbf{L}_s^{-1} = \frac{1}{x^2 + y^2} \begin{bmatrix} Zx & Zy \\ y & -x \end{bmatrix}$$

By using $\mathbf{v}_c = -\lambda \mathbf{L}_s^{-1}(\mathbf{s}, \hat{\mathbf{z}})(\mathbf{s} - \mathbf{s}^*)$, we obtain for the first iteration of the control scheme

$$\begin{bmatrix} v_z \\ \omega_z \end{bmatrix} = -\lambda \frac{1}{a^2 + b^2} \begin{bmatrix} \hat{Z}a & \hat{Z}b \\ b & -a \end{bmatrix} \begin{bmatrix} a - b \\ a + b \end{bmatrix} = \begin{bmatrix} -\lambda \hat{Z} \\ \lambda \end{bmatrix}$$

At the beginning of the servo, the motion of the camera will thus be composed of a rotational motion around the optical axis, but also of a backward motion along the optical axis. This motion is such that the image point follows a straight line in the image, as specified by the design of the control scheme. After some iterations, the

backward motion will be replaced by a forward motion so that the camera reaches its desired pose, corresponding to a pure rotational displacement.

By using now $\mathbf{v}_c = -\lambda \mathbf{L}_{\mathbf{s}}^{-1}(\mathbf{s}^*, \widehat{\mathbf{Z}}^*) (\mathbf{s} - \mathbf{s}^*)$ as control scheme, we obtain

$$\begin{bmatrix} v_z \\ \omega_z \end{bmatrix} = -\lambda \frac{1}{a^2 + b^2} \begin{bmatrix} \widehat{\mathbf{Z}}b & -\widehat{\mathbf{Z}}a \\ -a & -b \end{bmatrix} \begin{bmatrix} a - b \\ a + b \end{bmatrix} = \begin{bmatrix} \lambda \widehat{\mathbf{Z}} \\ \lambda \end{bmatrix}$$

The backward motion is now replaced by a forward motion, corresponding to the configuration where the error is small [7]. After some iterations, this forward motion will be compensated by a backward motion so that the camera reaches its desired pose.

Finally, by using $\mathbf{v}_c = -2\lambda \left(\mathbf{L}_{\mathbf{s}(\mathbf{s}, \widehat{\mathbf{Z}})} + \mathbf{L}_{\mathbf{s}(\mathbf{s}^*, \widehat{\mathbf{Z}}^*)} \right)^{-1} (\mathbf{s} - \mathbf{s}^*)$, we obtain

$$\begin{aligned} \begin{bmatrix} v_z \\ \omega_z \end{bmatrix} &= -2\lambda \begin{bmatrix} \frac{a+b}{\widehat{\mathbf{Z}}} & b-a \\ \frac{b-a}{\widehat{\mathbf{Z}}} & -(a+b) \end{bmatrix}^{-1} \begin{bmatrix} a-b \\ a+b \end{bmatrix} \\ &= -2\lambda \frac{1}{2(a^2 + b^2)} \begin{bmatrix} \widehat{\mathbf{Z}}(a+b) & \widehat{\mathbf{Z}}(b-a) \\ b-a & -(a+b) \end{bmatrix} \begin{bmatrix} a-b \\ a+b \end{bmatrix} = \begin{bmatrix} 0 \\ 2\lambda \end{bmatrix} \end{aligned}$$

which corresponds to a pure rotational motion. In all cases, the camera converges to its desired pose, but not at all with the same trajectory, the most natural one being obtained with the last case.

Let us now use the cylindrical coordinates of the point as visual features. We now have $\mathbf{s} = (\rho, \theta)$ and $\mathbf{s}^* = (\rho, \theta - \pi/2)$ with $\rho = \sqrt{a^2 + b^2}$ and $\theta = \arctan b/a$. From (11), the interaction matrix reduces to a nice diagonal form

$$\mathbf{L}_{\mathbf{s}} = \begin{bmatrix} \rho/Z & 0 \\ 0 & -1 \end{bmatrix}$$

Whatever the choice done for $\widehat{\mathbf{L}}_{\mathbf{s}}$, the output of the control scheme is now

$$\begin{bmatrix} v_z \\ \omega_z \end{bmatrix} = -\lambda \begin{bmatrix} \widehat{\mathbf{Z}}/\rho & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ \pi/2 \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda\pi/2 \end{bmatrix}$$

which corresponds to a pure rotation around the optical axis. To conclude, the choice of the visual features has clearly a strong impact on the behavior of the system.

The reader is encouraged to do the same exercise by considering $(-a, -b)$ as desired position of the point in the image, which corresponds to a rotation of π rad around the optical axis. The results in terms of camera trajectory are surprising even if they are as expected.

5 Applications

Potential applications of visual servoing are numerous. It can be used as soon as a vision sensor is available and a task is assigned to a dynamic system. A non-exhaustive list of examples is (see also Fig. 16)

- the control of a pan-tilt-zoom camera, as illustrated in Fig. 2 for the pan-tilt case;
- grasping using a robot arm;
- locomotion and dexterous manipulation with a humanoid robot;
- micro- or nano-manipulation of MEMS or biological cells;
- pipe inspection by an underwater autonomous vehicle;

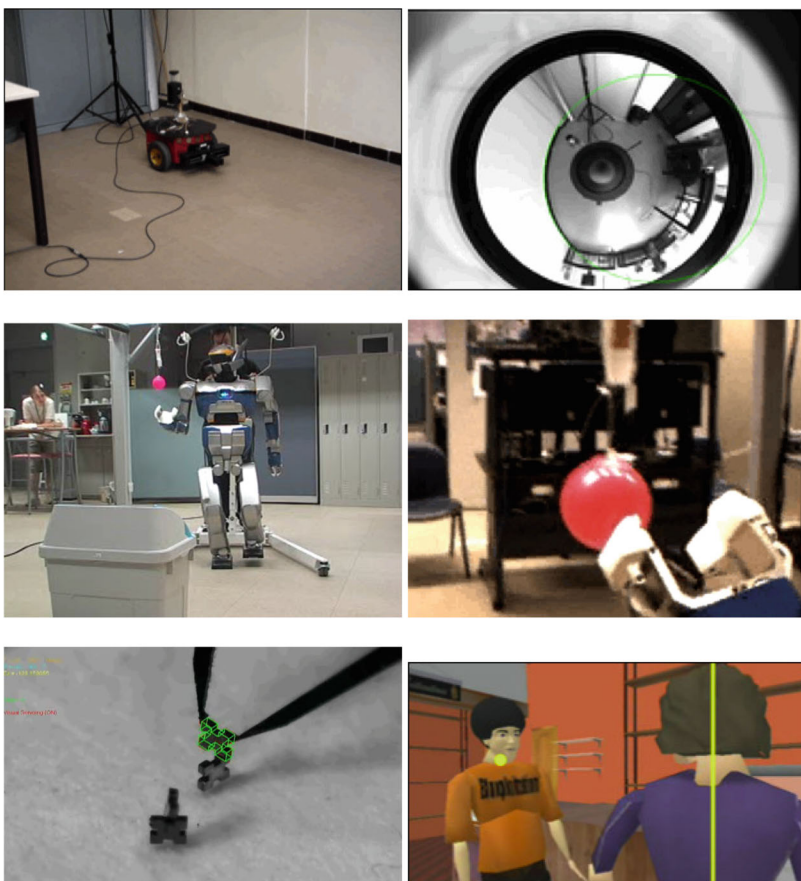


Fig. 16 Few applications of visual servoing: navigation of a mobile robot to follow a wall using an omnidirectional vision sensor (top line), grasping a ball with a humanoid robot (middle line), assembly of MEMS and film of a dialogue within the constraints of a script in animation (bottom line)

- autonomous navigation of a mobile robot in indoor or outdoor environment;
- aircraft landing;
- autonomous satellite rendezvous;
- biopsy using ultrasound probes or heart motion compensation in medical robotics.
- virtual cinematography in animation.

6 Summary and Future Directions

Visual servoing is a mature area. It is basically a nonlinear control problem for which numerous modeling works have been achieved to design visual features so that the control problem is transformed as much as possible to a linear control problem. On one hand, improvements on this topic are still expected for instantiating this general approach to particular applications. On the other hand, designing new control strategies is another direction for improvements, especially when supplementary data coming from other sensors (force, tactile, proximity sensors) are available. Finally, the current expansion of deep learning may rejuvenate the field [4, 28, 44], especially for the dense direct methods that use the same input and end-to-end approach.

Cross-References

This chapter contains cross-references to content published in other chapters belonging to the four “Robotic Goes MOOC” books: KNO2, KNO3. The full list of chapter abbreviations is available in the Preface.

References

1. D.J. Agravante, G. Claudio, F. Spindler, F. Chaumette, Visual servoing in an optimization framework for the whole-body control of humanoid robots. *IEEE Robot. Autom. Lett.* **2**(2), 608–615 (2016)
2. G. Allibert, E. Courtial, F. Chaumette, Predictive control for constrained image-based visual servoing. *IEEE Trans. Robot.* **26**(5), 933–939 (2010)
3. M. Bakthavatchalam, O. Tahri, F. Chaumette, A direct dense visual servoing approach using photometric moments. *IEEE Trans. Robot.* **34**(5), 1226–1239 (2018)
4. Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, P. Corke, Training deep neural networks for visual servoing, in *IEEE International Conference on Robotics and Automation (ICRA’18)* (2018), pp. 3307–3314
5. G. Caron, E. Marchand, E.M. Mouaddib, Photometric visual servoing for omnidirectional cameras. *Autonom. Robot.* **35**(2), 177–193 (2013)

6. F. Chaumette, Image moments: a general and useful set of features for visual servoing. *IEEE Trans. Robot.* **20**(4), 713–723 (2004)
7. F. Chaumette, S. Hutchinson, Visual servoing and visual tracking, in *Handbook of Robotics*, ed. by B. Siciliano, O. Khatib, chap. 24. Springer, Berlin (2008), pp. 563–583
8. G. Chesi, Visual servoing path planning via homogeneous forms and LMI optimizations. *IEEE Trans. Robot.* **25**(2), 281–291 (2009)
9. C. Collewet, E. Marchand, Photometric visual servoing. *IEEE Trans. Robot.* **27**(4), 828–834 (2011)
10. P. Corke, M. Good, Dynamic effects in visual closed-loop systems. *IEEE Trans. Robot. Autom.* **12**(5), 671–683 (1996)
11. N. Crombez, E.M. Mouaddib, G. Caron, F. Chaumette, Visual servoing with photometric Gaussian mixtures as dense features. *IEEE Trans. Robot.* **35**(1), 49–63 (2019)
12. A. Dame, E. Marchand, Mutual information-based visual servoing. *IEEE Trans. Robot.* **27**(5), 958–969 (2011)
13. K. Deguchi, A direct interpretation of dynamic images with camera and object motions for vision guided robot control. *Int. J. Comput. Vis.* **37**(1), 7–20 (2000)
14. A. De Luca, G. Oriolo, P. Robuffo Giordano, Feature depth observation for image-based visual servoing: theory and experiments. *Int. J. Robot. Res.* **38**(4), 422–450 (2008); **27**(10), 1093–1116
15. L.A. Dufлот, R. Reisenhofer, B. Tamadazte, N. Andreff, A. Krupa, Wavelet and shearlet-based image representations for visual servoing. *Int. J. Robot. Res.* **38**(4), 422–450 (2019)
16. B. Espiau, F. Chaumette, P. Rives, A new approach to visual servoing in robotics. *IEEE Trans. Robot. Autom.* **8**(3), 313–326 (1992)
17. R. Ginhoux, J. Gangloff, M. de Mathelin, L. Soler, M.A. Sanchez, J. Marescaux, Active filtering of physiological motion in robotized surgery using predictive control. *IEEE Trans. Robot.* **21**(1), 67–79 (2005)
18. H. Hadj-Abdelkader, Y. Mezouar, P. Martinet, F. Chaumette, Catadioptric visual servoing from 3D straight lines. *IEEE Trans. Robot.* **24**(3), 652–665 (2008)
19. T. Hamel, R. Mahony, Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach. *IEEE Trans. Robot. Autom.* **18**(2), 187–198 (2002)
20. S. Han, A. Censi, A. Straw, R. Murray, A bio-plausible design for visual pose stabilization, in *International Conference on Intelligent Robots and Systems (IROS'10)* (2010), pp. 5679–5686
21. K. Hashimoto, T. Ebine, H. Kimura, Visual servoing with hand-eye manipulator - optimal control approach. *IEEE Trans. Robot. Autom.* **12**(5), 766–774 (1996)
22. K. Hosoda, M. Asada, Versatile visual servoing without knowledge of true Jacobian, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)* (1994), pp. 186–193
23. S. Hutchinson, G. Hager, P. Corke, A tutorial on visual servo control. *IEEE Trans. Robot. Autom.* **12**(5), 651–670 (1996)
24. M. Iwatsuki, N. Okiyama, A new formulation of visual servoing based on cylindrical coordinate system. *IEEE Trans. Robot. Autom.* **21**(2), 266–273 (2005)
25. M. Jägersand, O. Fuentes, R. Nelson, Experimental evaluation of uncalibrated visual servoing for precision manipulation, in *IEEE International Conference on Robotics and Automation (ICRA'97)* (1997), pp. 2874–2880
26. M. Kazemi, K. Gupta, M. Mehrandezh, Randomized kinodynamic planning for robust visual servoing. *IEEE Trans. Robot.* **29**(5), 1197–1211 (2013)
27. W. Khalil, E. Dombre, *Modeling, Identification, and Control of Robots* (CRC Press, Boca Raton, 2002)
28. S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.* **17**(1), 1334–1373 (2016)
29. H.C. Longuet-Higgins, K. Prazdny, The interpretation of a moving retinal image. *Proc. R. Soc. Lond. B* **208**, 385–397 (1980)
30. G. Lopez-Nicolas, J.J. Guerrero, C. Sagues, Visual control through the trifocal tensor for nonholonomic robots. *Robot. Auton. Syst.* **58**(2), 216–226 (2010)

31. E. Malis, F. Chaumette, 2-1/2D visual servoing with respect to unknown objects through a new estimation scheme of camera displacement. *Int. J. Comput. Vis.* **37**(1), 79–97 (2000)
32. E. Marchand, F. Spindler, F. Chaumette, ViSP for visual servoing: a generic software platform with a wide class of robot control skills, in *IEEE Robotics and Automation Magazine, Special issue on “Software Packages for Vision-Based Control of Motion”*, ed. by P. Oh, D. Burschka, vol. 12, no. (4) (IEEE, Piscataway, 2005), pp. 40–52
33. E. Marchand, H. Uchiyama, F. Spindler, Pose estimation for augmented reality: a hands-on survey. *IEEE Trans. Visual. Comput. Graph.* **22**(12), 2633–2651 (2016)
34. G.L. Mariottini, G. Oriolo, D. Prattichizo, Image-based visual servoing for nonholonomic mobile robots using epipolar geometry. *IEEE Trans. Robot.* **23**(1), 87–100 (2007)
35. R. Mebarki, A. Krupa, F. Chaumette, 2D ultrasound probe complete guidance by visual servoing using image moments. *IEEE Trans. Robot.* **26**(2), 296–306 (2010)
36. R. Mebarki, V. Lippiello, B. Siciliano, Nonlinear visual control of unmanned aerial vehicles in GPS-denied environments. *IEEE Trans. Robot.* **31**(4), 1004–1017 (2015)
37. Y. Mezouar, F. Chaumette, Path planning for robust image-based control. *IEEE Trans. Robot.* **22**(10), 781–804 (2002)
38. G. Motyl, F. Chaumette, J. Gallice, Coupling a camera and laser stripe in sensor based control, in *Second International Symposium on Measurement and Control in Robotics* (1992), pp. 685–692
39. Y. Nakamura, H. Hanafusa, T. Yoshikawa, Task-priority based redundancy control of robot manipulators. *Int. J. Robot. Res.* **6**(2), 3–15 (1987)
40. D. Navarro-Alarcon, Y.H. Liu, J. Romero, P. Li, Model-free visually servoed deformation control of elastic objects by robot manipulators. *IEEE Trans. Robot.* **29**(6), 1457–1468 (2013)
41. S. Nayar, S. Nene, H. Murase, Subspace methods for robot vision. *IEEE Trans. Robot. Autom.* **12**(5), 750–758 (1996)
42. B. Nelson, P. Khosla, Strategies for increasing the tracking region of an eye-in-hand system by singularity and joint limit avoidance. *Int. J. Robot. Res.* **14**(3), 225–269 (1995)
43. J. Pagès, C. Collewet, F. Chaumette, J. Salvi, Optimizing plane-to-plane positioning tasks by image-based visual servoing and structured light. *IEEE Trans. Robot.* **22**(5), 1000–1010 (2006)
44. H. Pandya, A. Gaud, G. Kumar, K. Madhava Krishna, Instance invariant visual servoing framework for part-aware autonomous vehicle inspection using MAVs. *J. Field Robot.* **36**(5), 892–918 (2019)
45. G. Silveira, E. Malis, Direct visual servoing: vision-based estimation and control using only nonmetric information. *IEEE Trans. Robot.* **28**(4), 974–980 (2012)
46. I. Suh, Visual servoing of robot manipulators by fuzzy membership function based neural networks, in *Visual Servoing*. World Scientific Series on Robotics and Automated Systems, vol. 7 (World Scientific, Singapore, 1993), pp. 285–315
47. C. Teulière, E. Marchand, A dense and direct approach to visual servoing using depth maps. *IEEE Trans. Robot.* **30**(5), 1242–1249 (2014)
48. B. Thuilot, P. Martinet, L. Cordesses, J. Gallice, Position based visual servoing: keeping the object in the field of vision, in *IEEE International Conference on Robotics and Automation (ICRA’02)* (2002), pp. 1624–1629
49. R. Tsai, R. Lenz, A new technique for fully autonomous and efficient 3D robotics hand-eye calibration. *IEEE Trans. Robot. Autom.* **5**(3), 345–358 (1989)
50. L. Weiss, A. Sanderson, C. Neuman, Dynamic sensor-based control of robots with visual feedback. *IEEE J. Robot. Autom.* **3**(5), 404–417 (1987)
51. G. Wells, C. Venaille, C. Torras, Vision-based robot positioning using neural networks. *Image Vision Comput.* **14**(10), 715–732 (1996)
52. W. Wilson, C. Hulls, G. Bell, Relative end-effector control using cartesian position-based visual servoing. *IEEE Trans. Robot. Autom.* **12**(5), 684–696 (1996)



Joe Watson, Julen Urain, Joao Carvalho, Niklas Funk, and Jan Peters

Abstract Robot learning uses advanced machine learning and statistical methods to improve the effectiveness of algorithms and models used in robotics, including low-level control, dynamics models and high-level planning. Having such autonomous learning for physical systems has been a long-standing vision of robotics, artificial intelligence, and cognitive sciences. In the past two decades, the Robot Learning community has developed a large repertoire of learning techniques that enable robots to acquire new skills from data. This book chapter aims to provide a straightforward and intuitive introduction to the approaches that enabled these methods, starting from optimal control theory, and incorporating ideas from machine learning that facilitate model-based and model-free reinforcement learning and imitation learning. To enable practical learning on real robotic platforms, we describe inductive biases for robotic learning, such as movement primitives; and how they are used for sample-efficient, data-driven control.

1 Introduction

Over the past several decades, robotics research has yielded the necessary theory and understanding to design, program and deploy a vast variety of robotic platforms across a broad range of settings, from subterranean mapping to surgical procedures. More recently, advances and widespread adoption of information technology has led to increased interest in *machine learning* (ML) [5, 11], a research field devoted to the development and study of algorithms and models for prediction, statistical modelling and optimization. The intersection of robotics and machine learning is known as Robot Learning [82].

For robotics, machine learning provides a means of *data-driven* design—replacing human engineering with data acquisition and processing. As our robots

J. Watson (✉) · J. Urain · J. Carvalho · N. Funk · J. Peters

TU Darmstadt, Darmstadt, Germany

e-mail: joe.watson@robot-learning.de; julen.urain@robot-learning.de;

joao.carvalho@robot-learning.de; niklas.funk@robot-learning.de; jan.peters@robot-learning.de

are expected to perform increasingly more complex tasks where manual may become infeasible, this automation has the promise of accelerating the design of these systems by leveraging large datasets of experience. Breakthroughs in computer vision [48], natural language processing [107] and decision making [68, 105] have demonstrated that this paradigm enables performance that matches or even *surpasses* human capabilities. However, these breakthroughs occurred on either static or abstract tasks while robot learning requires considering dynamically changing tasks in real-world embedding. Thus, achieving similar results on real-world robots requires models and algorithms that respect and incorporate their physical embodiment.

1.1 What Is Robot Learning?

From the robotics perspective, Robot Learning constitutes the use of data-driven approaches to improve the effectiveness of the system. This paradigm builds on longstanding methods in robotics such as system identification [60, 112] and adaptive control [4]. From the machine learning perspective, Robot Learning considers models and algorithms that are relevant to the robotic setting, ranging from (semi-) supervised learning to make sense of visual and tactile perception [52, 57] to optimal interaction with the environment by reinforcement learning (RL) algorithms [47, 110].

Definition 1 (Robot Learning) Robot learning considers the intersection of robotics and machine learning. The goal is to provide methods that allow robots to acquire novel skills, improve its performance, or adapt to dynamic environments through algorithms applied to acquired data. These methods apply to every aspect of the robot, including the morphology, perception, motor control and high-level reasoning. The physical embodiment of the robot provides both design constraints and inspiration for algorithm design, such as safety, latency and inductive biases such as physics and geometry.

Moreover, the specific robotic setting motivates task- and system-specific inductive biases [6] is often necessary when data has to be generated in real-time on real system (which may suffer from damages as well as wear and tear) to ensure feasibility or at least to improve performance. Such inductive biases can include insights from physics-inspired models, geometry-constrained planning and approaches borrowed from control theory. Finally, the real-world embedded deployment requires additional considerations, such as computational efficiency, robustness, safety and continual learning. For a formal definition, see Definition 1. Examples are provided in Fig. 1.

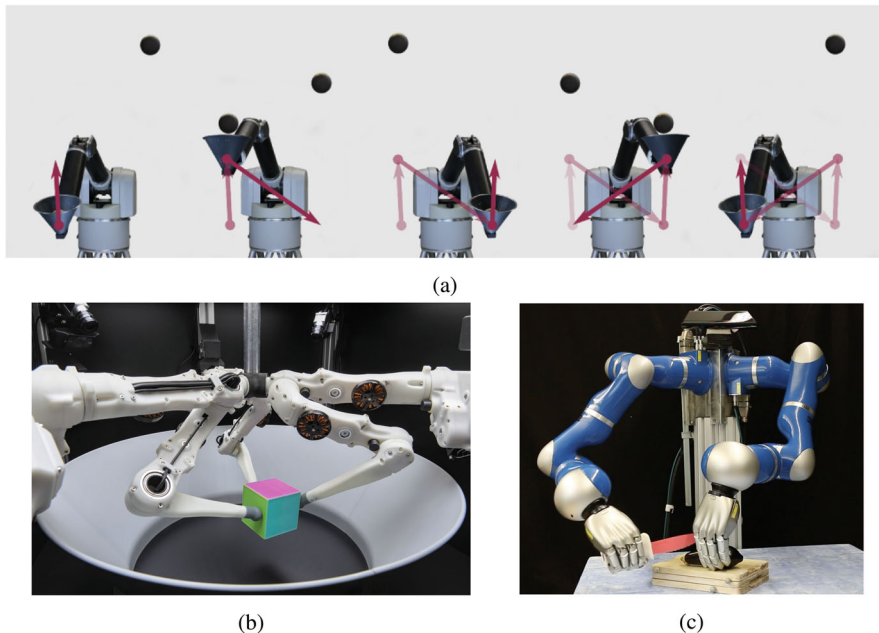


Fig. 1 Diverse applications of robot learning. (a) High-acceleration robot juggling [83], (b) precise dexterous object manipulation [26] (copyright MPI for Intelligent Systems), and (c) fine bi-manual manipulation [59]. Panel (b) adapted from [92] with permission from Felix Widmaier; Copyright MPI for Intelligent Systems

Given its broad definition, this introduction focuses on a specific aspect of Robot Learning, looking at methods for learning motor actions. We chart the evolution of classical ideas from robotics, such as optimal control and system identification, to their modern interpretations as reinforcement learning and statistical machine learning. We motivate the use of purely data-driven control design methods, such as model-free policy search,¹ and show how these methods can be extended to settings such as learning from demonstration (LfD; often also called imitation learning). Moreover, we discuss how robotics-inspired inductive biases can be incorporated into the statistical models, such as Bayesian linear regression with dynamic movement primitives for learning smooth robot motion from noisy demonstrations. Robot Learning covers many more aspects of robotics, such as perception, state estimation and scene understanding. While these topics are out of scope for this introduction, they are also relevant for learning motor actions, such as visuomotor control. Moreover, Robot Learning is intrinsically linked to the computation and

¹ The term “control policy” is shortened to just “policy” in the machine learning literature but typically includes both a control law and the trajectory generator from classical robotics.

instrumentation hardware available, as it informs the data that is available and the algorithms that are feasible. However, we do not discuss this aspect in this chapter.

1.2 Structure of the Chapter

This chapter looks specifically at Robot Learning in the context of learning motor control for robotics. Section 2 introduces optimal control and Markov decision processes (MDP) as the underlying framework. Section 3 covers the basics in statistical machine learning and deep learning for regression, which is required for learning models from data. How these models are used for system identification and data-driven control is discussed in Sect. 4, and for useful policy representations in Sect. 5. Section 6 considers data-driven control without the notion of an explicit dynamics model, using the formalism of model-free reinforcement learning. Finally, Sect. 7 looks at imitation learning, which involves learning from expert demonstrations.

Notation We adopt lowercase boldface letters $\mathbf{x}$ for vectors, uppercase boldface for matrices $\mathbf{X}$, otherwise x , X denote scalar variables. The expression $\mathbf{x} \sim p(\mathbf{x})$ means that a random variable $\mathbf{x}$ is distributed according to the distribution $p(\mathbf{x})$. The log-likelihood under this distribution is expressed as $\mathcal{L}(\mathbf{x}) = \log p(\mathbf{x})$. A multivariate Normal distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance $\boldsymbol{\Sigma} \in S_+^n$ is $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $+$ denotes positive (semi-) definiteness. For a random variable $\mathbf{x}$, $\mathbb{E}[\mathbf{x}]$, $\mathbb{V}[\mathbf{x}]$ and $\mathbb{H}[\mathbf{x}]$ denote the mean, covariance and entropy respectively. The Kullback-Leibler divergence between distributions $p(\mathbf{x})$ and $q(\mathbf{x})$ is denoted $D_{\text{KL}}[p(\mathbf{x}) \parallel q(\mathbf{x})]$. For scalar function $f(\mathbf{x}, \mathbf{y})$, we use the shorthand $\mathbf{f}_{\mathbf{x}} = \partial f / \partial \mathbf{x}$ and $\mathbf{F}_{\mathbf{x}\mathbf{y}} = \partial^2 f / \partial \mathbf{x} \partial \mathbf{y}$ for Jacobians and Hessians respectively. Vector-valued functions have a matrix Jacobian $\mathbf{F}_{\mathbf{x}}$.

2 Optimal Control from Robot Forward Models

Classical control covers a multitude of criteria for control design, such as stability, frequency response and disturbance rejection. As machine learning methods tend to be presented as an optimization problem, optimal control is the preferred control foundation in Robot Learning. A highly effective robot learning approach can be obtained by simply swapping the manually acquired forward model in the optimal control literature with a model identified by supervised learning. Moreover, optimal control covers not only continuous state and action spaces, but also discrete spaces, so it can be applied to both continuous control and also more abstract forms of robot decision making.

This section introduces the Markov decision process for optimal control in the discrete setting. For continuous control, we first look at the linear quadratic regulator (LQR), which has a closed-form solution for the optimal control law. Finally, we

discuss approximate optimal control methods for the nonlinear dynamical systems that we often encounter in robotics.

2.1 Markov Decision Processes

Before introducing methods for obtaining optimal controllers, we first have to define the problem. A wide range of problems in robotics ranging from path planning to low level control can be formalized as Markov decision processes (MDPs, Definition 2, Figs. 2 and 3) [8, 86].

Definition 2 (Markov decision process) A stationary Markov decision process (MDP) describes the stochastic dynamics of a sequential decision making process that satisfies the Markov property. In particular, it is defined by state space $s \in \mathcal{S}$, action space $a \in \mathcal{A}$, transition dynamics $p(s_{t+1}|s_t, a_t)$, reward function $r(s_t, a_t)$, and initial state probabilities $\mu_0(s)$. Importantly, the transition dynamics fulfill the Markov property, i.e. the state transitions only depend on the current state and action and are thus conditionally independent of previous states and actions, $p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots) = p(s_{t+1}|s_t, a_t)$. Note that non-stationary MDPs follow the same definition but include time-varying transition dynamics, $p_t(s_{t+1}|s_t, a_t)$ and reward functions $r_t(s_t, a_t)$.

MDPs do not only define the transition dynamics but also include a function that evaluates the quality of the current state and action. Hence, the goal is to obtain policies π that are reward optimal. A policy is a function mapping from states to actions which are either be deterministic, $a = \pi(s)$, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, or stochastic, i.e., defining a probability distribution over all actions given the current state, $a \sim \pi(\cdot|s)$, $\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Given the properties of MDPs, Richard Bellman formulated the

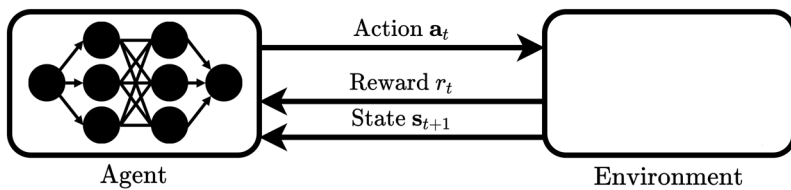
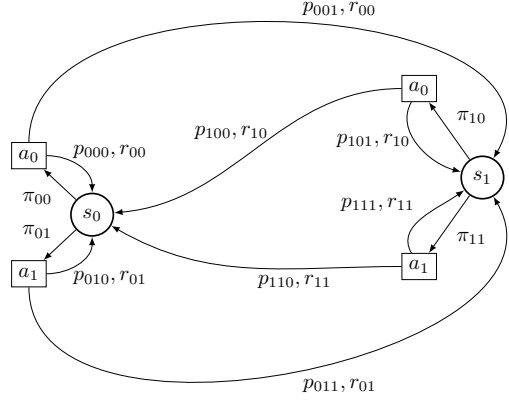


Fig. 2 Illustration of a sequential decision making process in which an agent is interacting with an environment, such as optimal control of a robot. In this example, the agent's actions are computed based on a parametric neural network policy $\mathbf{a}_t = \pi_\theta(s_t)$ and applied to a robotic manipulation task which then yields the next state s_{t+1} and reward r_t . The setting is described as Markov decision process

Fig. 3 A diagram of a stationary Markov decision process with two states $\{s_0, s_1\}$ and two actions $\{a_0, a_1\}$. The policy and dynamics are represented by probabilities π_{sa} and $p_{sas'}$, with the next state s' and the reward values r_{sa}



“principle of optimality” in 1957 stating that “an optimal sequence of controls in a multistage optimization problem has the property that whatever the initial stage, state and controls are, the remaining controls must constitute an optimal sequence of decisions for the remaining problem with stage and state resulting from previous controls considered as initial conditions.” In short, Bellman’s principle of optimality implies that every part of the optimal solutions must itself be optimal, which directly results into a recursion often called the Bellman equation²,

$$V_t^*(s) = \mathbb{E}_{p, \pi^*} \left[\sum_{\tau=t}^T r(s_\tau, \mathbf{a}_\tau) \right] = \max_{\mathbf{a}_t} \underbrace{r(s_t, \mathbf{a}_t) + V_{t+1}^*(s_{t+1})}_{Q_t^*(s_t, \mathbf{a}_t)} = \max_{\mathbf{a}_t} Q_t^*(s_t, \mathbf{a}_t), \quad (1)$$

with value function $V_t(s_t)$, state-action value function $Q_t(s_t, \mathbf{a}_t)$ that is sometimes also referred to as Q -function,³ optimal policy π^* , optimal value and Q -function V^*, Q^* . All methods that exploit this idea of decomposing the original problem into smaller subproblems belong to the family of dynamic programming (DP) algorithms. It is relatively straightforward to apply these ideas in tabular environments with discrete sets of actions, states and known transition dynamics, and results in the two algorithms of value and policy iteration [8, 10, 110]. In value iteration (VI, cf. Algorithm 1), one computes the optimal value function recursively starting from the final state, which directly also yields the optimal policy, i.e. the action that actually maximizes the value in each state. In policy iteration (PI, cf. Algorithm 2), we are starting with a fixed policy and first compute all the Q -values $Q_t^\pi(s_t, \mathbf{a}_t)$ under the current policy, and then update the policy to greedily choose the actions that yield

² The Bellman equation was named in Bellman’s honor but was known to Carathéodory, von Neumann, Morgenstern, Wald and others before Bellman generalized it into the principle of optimality and the generic method of dynamic programming.

³ Following on from value, Q stands for *quality*.

Algorithm 1 Value iteration

Init: $V_T^*(s_T) \leftarrow r(s_T)$, $V_{t \neq T}^*(s_{t \neq T}) = 0$
repeat
 /* Compute Q -function for each state action pair and point in time
 $Q_t^*(s_t, \mathbf{a}_t) \leftarrow r(s_t, \mathbf{a}_t) + \sum_{s_{t+1}} p(s_{t+1}|s_t, \mathbf{a}_t) V_{t+1}^*(s_{t+1})$
 /* Compute V -function for each state and point in time
 $V_t^*(s_t) \leftarrow \max_{\mathbf{a}_t} Q_t^*(s_t, \mathbf{a}_t)$
until $V_t^*(s_t)$ converged for all states and points in time
return Optimal policy: $\pi_t^*(s_t) = \arg \max_{\mathbf{a}_t} Q_t^*(s_t, \mathbf{a}_t)$

Algorithm 2 Policy iteration

Init: $V_T^\pi(s_T) \leftarrow r(s_T)$, $V_{t \neq T}^\pi(s_{t \neq T}) = 0$, $\pi \leftarrow \text{random}$
repeat
repeat
 /* Compute Q -function for each state action pair and point in time under current policy
 $Q_t^\pi(s_t, \mathbf{a}_t) \leftarrow r(s_t, \mathbf{a}_t) + \sum_{s_{t+1}} p(s_{t+1}|s_t, \mathbf{a}_t) V_{t+1}^\pi(s_{t+1})$
 /* Compute V -function for each state and point in time
 $V_t^\pi(s_t) \leftarrow Q_t^\pi(s_t, \mathbf{a}_t = \pi_t(s_t))$
until $V_t^\pi(s_t)$ converged
 /* Update Policy
 $\pi_t(s_t) \leftarrow \arg \max_{\mathbf{a}_t} Q_t^\pi(s_t, \mathbf{a}_t)$
until Convergence of policy
return Optimal policy: $\pi_t^*(s_t) \leftarrow \pi_t(s_t)$

highest value in each state. This process is repeated until convergence and also yields the optimal policy. Policy iteration is known to be computationally more effective as it allows effects of policy changes to propagate through the value function before requiring redundant use of the max-operator on an unchanged value function.

2.2 The Linear Quadratic Regulator

For special cases, Markov decision processes with continuous states $\mathbf{s} \in \mathbb{R}^{d_s}$ and actions $\mathbf{a} \in \mathbb{R}^{d_a}$ are solved analogously. For example, we can perform optimal control using the value-iteration approach in a tractable fashion when the dynamics function is a time-varying linear (affine) state-space model and the reward is time-varying quadratic, where weights $\mathbf{Q}_t$ and $\mathbf{R}_t$ are positive definite matrices and $\mathbf{s}_g$ is the goal state. This setting is known as the linear quadratic regulation (LQR), see [13], with

$$\mathbf{s}_{t+1} = \mathbf{f}_t(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{A}_t \mathbf{s}_t + \mathbf{B}_t \mathbf{a}_t, \quad r_t(\mathbf{s}_t, \mathbf{a}_t) = -(\mathbf{s}_g - \mathbf{s}_t)^\top \mathbf{Q}_t (\mathbf{s}_g - \mathbf{s}_t) - \mathbf{a}_t^\top \mathbf{R}_t \mathbf{a}_t.$$

For finite-horizon optimal control, we also can specify a terminal reward, only in the state $r_T(s_T) = -(s_g - s_T)^\top \mathbf{Q}_T (s_g - s_T)$. The Bellman equation,

$$V_t(s_t) = \max_{a_t} Q_t(s_t, a_t) = \max_{a_t} r_t(s_t, a_t) + V_{t+1}(f_t(s_t, a_t)),$$

can be solved analytically as affine systems combined with quadratic rewards imply that the value function $V_t(s_t) = s_t^\top \mathbf{V}_t s_t + \mathbf{v}_t^\top s_t + v_t$ is always a quadratic form.

This Bellman equation is solved by a parametric version of the value iteration algorithm. Here, the value function is initialized by $V_T = \mathbf{Q}_T$ and, subsequently, the recursive parametric update equation,

$$\mathbf{V}_t = \mathbf{Q}_t + \mathbf{A}_t^\top \mathbf{V}_{t+1} \mathbf{A}_t - \mathbf{A}_t^\top \mathbf{V}_{t+1} \mathbf{B}_t (\mathbf{R}_t + \mathbf{B}_t^\top \mathbf{V}_{t+1} \mathbf{B}_t)^{-1} \mathbf{B}_t^\top \mathbf{V}_{t+1} \mathbf{A}_t, \quad (2)$$

is computed backwards in time from $t = T-1$ down to $t = 1$. Due the specific structure of Eq. (2), it is known as a discrete algebraic Riccati equation (DARE). Solving the value function for the optimal action, we arrive at a time-varying affine controller in s_t ,

$$\begin{aligned} a_t^* &= \arg \max_{a_t} Q_t(s_t, a_t) = -(\mathbf{R}_t + \mathbf{B}_t^\top \mathbf{V}_{t+1} \mathbf{B}_t)^{-1} (\mathbf{B}_t^\top \mathbf{V}_{t+1} \mathbf{A}_t s_t - \mathbf{B}_t^\top v_{t+1}), \\ &= \mathbf{K}_t s_t + \mathbf{k}_t. \end{aligned}$$

If the system is controllable, the control policy's parameters $\mathbf{K}_t, \mathbf{k}_t$ will converge for $t = \hat{t} \ll T$ for sufficiently large T long before $t = 1$. Thus, the stationary linear controller $a^* = \mathbf{K}_{\hat{t}} s + \mathbf{k}_{\hat{t}}$ is an infinite-horizon optimal control law.

2.3 Nonlinear Approximate Optimal Control

The LQR setting was restricted to time-varying linear dynamics and quadratic rewards. For many control problems in robotics, we have nonlinear dynamics and non-quadratic objectives. In this section, we briefly describe two possible approaches to this problem that perform approximate optimal control.

Differential Dynamic Programming Inspired by LQR, we compute a local second-order Taylor approximation of the dynamics and reward function computed along a nominal trajectory $(\bar{s}_t, \bar{a}_t)$ for every time-step $t = 1 : T$ by

$$\begin{aligned} r_t(\bar{s}_t + \delta_s, \bar{a}_t + \delta_a) &= r_t(\bar{s}_t, \bar{a}_t) + [\mathbf{r}_{s_t} \ \mathbf{r}_{a_t}] \begin{bmatrix} \delta_s \\ \delta_a \end{bmatrix} + \begin{bmatrix} \delta_s \\ \delta_a \end{bmatrix}^\top \begin{bmatrix} \mathbf{R}_{ss_t} & \mathbf{R}_{sa_t} \\ \mathbf{R}_{sa_t}^\top & \mathbf{R}_{aa_t} \end{bmatrix} \begin{bmatrix} \delta_s \\ \delta_a \end{bmatrix}, \\ f_t(\bar{s}_t + \delta_s, \bar{a}_t + \delta_a) &= f_t(\bar{s}_t, \bar{a}_t) + [\mathbf{F}_{s_t} \ \mathbf{F}_{a_t}] \begin{bmatrix} \delta_s \\ \delta_a \end{bmatrix} + \mathbf{f}_t'', \end{aligned}$$

with a Jacobian matrix F and second-order term f_t'' that is computed by

$$f_t^{(i)''} = \begin{bmatrix} \delta_s \\ \delta_a \end{bmatrix}^\top \begin{bmatrix} F_{ss_t}^{(i)} & F_{sa_t}^{(i)} \\ F_{sa_t}^{(i)\top} & F_{aa_t}^{(i)} \end{bmatrix} \begin{bmatrix} \delta_s \\ \delta_a \end{bmatrix},$$

for every dimension i from 1 to d_s .

With these second-order approximations, we construct an approximate value functions using the backward recursions, and obtain local time-varying linear controllers. Due to its use of differentiation, this approach is known as differential dynamic programming (DDP) [41]. One key difference between DDP and LQR is the requirement of a trajectory about which to linearize the dynamics and cost. Therefore this algorithm is iterative, where a forward pass computes the linearization trajectory, and the backward pass performs local optimal control. The linearization trajectory in the forward pass is obtained by simulating the current controller on the model of the dynamical system. For iteration i ,

$$s_{t+1}^i = f_t(s_t^i, a_t^i), \quad a_t^i = K_t^{i-1} s_t^i + k_t^{i-1}.$$

Due to the use of the Hessian, this optimization is equivalent to a Newton method [12]. Given its local nature, line search routines are often used to ensure optimization stability and convergence. One issue with DDP in practice is that the dynamics model Hessians are high-dimensional tensors and therefore expensive to compute and slow down optimization. Ignoring these terms, and using the Jacobians to approximate the Hessians instead, is equivalent to Gauss-Newton optimization [12]. This approach also requires careful regularization and line search for convergence, due to the additional approximations. This approximate DDP method is known as sequential LQR or iterative LQR (iLQR) [56, 104].

Approximate Dynamic Programming An alternative strategy is to use *function approximation* for the value functions and policy. Given a set of transitions $(s_t^i, a_t^i, r_t^i, s_t^i)$, $t=1:T$, $i=1:N$ over N rollouts, the stationary optimal value function and policy are iteratively estimated by using Bellman's equation to compute regression targets. We will cover the details of function approximation, and how the *approximate* dynamic programming (ADP) [10] objectives are optimized, in more detail in the next section. As ADP provides a means of performing optimal control from sampled MDP transitions, it is the foundation on which many reinforcement learning algorithms are constructed, which are discussed in more detail in Sect. 6.

3 Supervised Machine Learning: Acquiring Models from Data

This section will introduce some fundamental ideas from supervised machine learning for how to learn models from data. In the previous section, we assumed a fully engineered dynamics model, which we used to perform optimal control. We also already discussed the idea of function approximation to learn unknown value functions and policies of approximate optimal control. This section aims to fill the gap left for didactic reasons.

Supervised machine learning is an important tool with many applications in Robot Learning including:

- Learning models of unknown dynamical systems for control (i.e. using machine learning for structure and system identification in Sect. 4).
- Approximate policies and value functions from samples policies in approximate dynamic programming (Sect. 4) and reinforcement learning (Sect. 6).
- Approximate policies from experience or demonstrations (e.g., in imitation learning in Sect. 7).
- Learning other unknown robotics models, such as kinematic, sensor and camera models.

We first consider the simplest model, a linear model [11], $\mathbf{y} = \mathbf{x}\mathbf{W}$, where $\mathbf{y} \in \mathbb{R}^{d_y}$, $\mathbf{x} \in \mathbb{R}^{d_x}$, $\mathbf{W} \in \mathbb{R}^{d_x \times d_y}$. Fitting n datapoints, where $\mathbf{X} \in \mathbb{R}^{n \times d_x}$, $\mathbf{Y} \in \mathbb{R}^{n \times d_y}$ and minimizing the squared error objective using the L2 norm $|\cdot|^2$ yields,

$$\mathbf{W}^* = \arg \min_{\mathbf{W}} |\mathbf{Y} - \mathbf{X}\mathbf{W}|^2 = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y},$$

which is known as the ‘normal equation’. When learning from only few datapoints, we often require regularization of $\mathbf{W}$ to prevent overfitting to the observed data

$$\mathbf{W}^* = \arg \min_{\mathbf{W}} |\mathbf{Y} - \mathbf{X}\mathbf{W}|^2 + \beta^2 |\mathbf{W}|^2 = (\mathbf{X}^\top \mathbf{X} + \beta^2 \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{Y}.$$

Adding a $|\mathbf{W}|^2$ term into the objective to keep the weights small is known as *ridge* or *Tikhonov* regularization, where β is a hyperparameter [11].

The simplicity of the linear model in $\mathbf{x}$ alone is often quite limiting. Thus, it is popular to perform linear regression in a feature space $\phi(\mathbf{x}) \in \mathbb{R}^{d_\phi}$ instead, which is a transformation of $\mathbf{x}$. Such feature transformations include adding a bias for affine models $\phi(\mathbf{x}) = [\mathbf{x} \ 1]^\top$ or polynomial features for quadratic model, e.g. $\phi(\mathbf{x}) = [x_1^2 \ x_2^2 \ x_1 x_2 \ x_1 \ x_2 \ 1]^\top$. Moreover, we would also like to incorporate a notion of *uncertainty* into our learned parameters and model predictions.

3.1 Bayesian Linear Regression and Nonparametric Gaussian Processes

In the introduction of linear regression so far, we have taken a so-called “frequentist perspective” and assumed the existence of a *single*, unknown, optimal weight parameter value $\mathbf{W}^*$. The fallacy of this assumption was evident in the need for parameter regularization, keeping the values small, in order to avoid overfitting when presented with small sets of data. Bayesian methods take an alternative approach where many parameter values are considered and their plausibility is assessed by maintaining probability distributions over the model variables, initialized with a prior distribution. Such probabilistic treatment has two key effects: The initial probability distribution (prior) regularizes the parameter updates, and propagating the parameter uncertainty through the model provides uncertainty quantification of the model’s prediction. Such predictive uncertainty can be used for downstream tasks concerning safety, exploration and decision making—crucial in robot learning. The catch with Bayesian methods is the increased complexity in the derivations and implementations. While some Bayesian models can be learned in closed form with exact inference, many are intractable and require approximate inference methods, such as variational inference and Monte Carlo methods [5].

Definition 3 (Bayes’ Theorem) Bayes’ theorem is derived from conditional probability theory. When θ is an unknown model parameter, it is used to construct Bayesian statistical models. Given a data likelihood $p(\mathcal{D}|\theta)$ and a prior over parameters $p(\theta)$,

$$\underbrace{p(\theta|\mathcal{D})}_{\text{Posterior}} = \frac{\overbrace{p(\mathcal{D}|\theta)}^{\text{Likelihood}} \overbrace{p(\theta)}^{\text{Prior}}}{\underbrace{p(\mathcal{D})}_{\text{Evidence}}}$$

where the evidence, or marginal likelihood, is the probability of the data given the model $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)d\theta$. The probability of the data represents the normalizing constant for the probability density function in Bayes’ rule; it is useful for model selection, i.e., for deciding which Bayesian model is best for the data, and hyperparameter optimization.

When implementing Bayesian models, the fundamental tool is Bayes rules, which describes the updated distribution (posterior) as a combination of the prior and observed data (likelihood), as described in Definition 3. To provide a worked example of Bayesian methods, consider estimating a position $\mathbf{x}$ using a noisy sensor $\mathbf{y} = \mathbf{x} + \epsilon$, corrupted by Gaussian noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$. To make an sensible

estimate from limited samples, we place a Gaussian prior on x which describes our initial guess $\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. The sensor noise ϵ defines our Gaussian likelihood. Given a single estimate y_1 , the posterior distribution $\mathcal{N}(x_1|y_1)$ is computed using Bayes rule in closed-form

$$\boldsymbol{\Sigma}_1 = (\boldsymbol{\Sigma}_0^{-1} + \frac{1}{\sigma^2} \mathbf{I})^{-1}, \quad \boldsymbol{\mu}_1 = \boldsymbol{\Sigma}_1 (\boldsymbol{\Sigma}_0^{-1} \boldsymbol{\mu}_0 + \frac{1}{\sigma^2} y_1).$$

Note this update can be used recursively for new data, by using the posterior as the subsequent prior. This aspect of Bayesian methods make them popular for localization and state estimation [22, 97] when the Markov assumption is used.

Returning to regression, we focus on one-dimensional targets and place a multivariate Gaussian prior on $\mathbf{w}$, and use Bayesian inference in a similar manner to the small example above. The equations are summarized in Definition 4, and these models are also known as Gaussian processes [90]. Note that when $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\boldsymbol{\Sigma}_0 = \mathbf{I}$, the mean updated matches the ridge regression update above where $\beta \equiv \sigma^2$.

Definition 4 (Gaussian Process as Bayesian Linear Regression) For a linear model with Gaussian likelihood, where $\phi_x = \phi(x)$

$$y = \phi_x^\top \mathbf{w} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2),$$

with Gaussian prior $p(\mathbf{w}) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. For a dataset $\mathcal{D}$ of n points $\{\Phi, \mathbf{y}\}$, $\Phi \in \mathbb{R}^{n \times d_\phi}$ and $\mathbf{y} \in \mathbb{R}^{n \times 1}$, the weight posterior $p(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)$ where,

$$\boldsymbol{\Sigma}_n = \left(\boldsymbol{\Sigma}_0^{-1} + \frac{1}{\sigma^2} \Phi^\top \Phi \right)^{-1}, \quad \boldsymbol{\mu}_n = \boldsymbol{\Sigma}_n \left(\boldsymbol{\Sigma}_0^{-1} \boldsymbol{\mu}_0 + \frac{1}{\sigma^2} \Phi^\top \mathbf{y} \right).$$

The closed-form Gaussian posterior predictive for query point $\mathbf{x}_*$ is

$$p(y_* | \mathbf{x}_*, \mathcal{D}) = \mathcal{N}(\mu_{y_*}, \sigma_{y_*}^2), \quad \mu_{y_*} = \phi_*^\top \boldsymbol{\mu}_n, \quad \sigma_{y_*}^2 = \sigma^2 + \phi_*^\top \boldsymbol{\Sigma}_n \phi_*.$$

Note that for Bayesian linear regression, the predictive uncertainty is now a function of the input. This uncertainty can be broken down into two components

$$\sigma_{y_*}^2 = \underbrace{\sigma^2}_{\text{Aleatoric}} + \underbrace{\phi_*^\top \boldsymbol{\Sigma}_n \phi_*}_{\text{Epistemic}}.$$

Aleatoric uncertain describes that statistical uncertainty, for example sensor noise, and is inherent to the problem independent from the dataset. Epistemic uncertainty describes model uncertainty and arises from uncertain model parameters given on the observed data. Once enough data is observed, Bayesian linear regression becomes more confident in $\mathbf{w}$ and the epistemic uncertainty disappears as $\Sigma_n \rightarrow \mathbf{0}$. Note that using just the (regularized) posterior mean, rather than the full posterior, is called a *maximum a posteriori* (MAP) update [5].

What are the limits of feature regression? Designing features is a crucial design choice in many machine learning applications, and the more features the richer the modeling capacity. For this reason, features are often flexible and parametrized, and then optimized using the evidence. For example, radial basis function (RBF; see [11]) features $\phi_i(\mathbf{x}) = \exp(-(\mathbf{x} - \mathbf{c}_i)/2l_i^2)$ are defined by a centre $\mathbf{c}$ and length-scale l . An alternative approach for working with richer feature spaces is to use kernel methods [36]. The kernel is the inner product of a feature space at two points $k(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2)$, $k: \mathbb{R}^{d_x} \times \mathbb{R}^{d_x} \rightarrow \mathbb{R}$. Importantly, the kernel is always a scalar function, regardless of the feature dimension d_ϕ , and, thus, we can define kernels that represent infinite feature spaces! By working purely with the kernel, rather than the features explicitly, we can enjoy the benefit of the rich representation while remaining computationally tractable. Two examples of kernels $k(\mathbf{x}_1, \mathbf{x}_2)$ are

$$k_{\text{RBF}}(\mathbf{x}_1, \mathbf{x}_2) = \exp(-|\mathbf{x}_1 - \mathbf{x}_2|^2/2l^2),$$

$$k_{\text{arcsine}}(\mathbf{x}_1, \mathbf{x}_2) = \frac{2}{\pi} \arcsin \left(\frac{2\bar{\mathbf{x}}_1^\top \Sigma \bar{\mathbf{x}}_2}{\sqrt{(1 + 2\bar{\mathbf{x}}_1^\top \Sigma \bar{\mathbf{x}}_1)(1 + 2\bar{\mathbf{x}}_2^\top \Sigma \bar{\mathbf{x}}_2)}} \right), \quad \bar{\mathbf{x}} = [\mathbf{x}, \ 1]^\top.$$

Such RBF kernel is derived from an infinite limit of RBF features, and is *stationary* as it depends only on the distance between points. The arcsine kernel [126] is derived from the limit of a single hidden layer neural network (see next section) with erf function activations and a Gaussian distribution over the weights and bias. It is non-stationary. To use kernels for Bayesian linear regression, we rewrite the posterior update to use only the inner feature product, rather than the outer product [90]. This model is described in Definition 5. The result is that the predictive distribution is computed using the data directly, as we must use a data matrix $\mathbf{K} = k(\mathbf{X}, \mathbf{X})$ that computes the inner-products over the whole dataset. As the model is now explicitly represented in a flexible manner using the data, rather than finite parameters that need optimizing, we call it *non-parametric*. Non-parametric does not imply that there are no parameters but that the number of parameters of the model grows implicitly with the number of data points. Thus, the model complexity is now dependent on the dataset size. This scaling is a quality, as model complexity increases with the dataset size, but becomes an issue when working with very large datasets—both due to computation and storage reasons.

Definition 5 (Non-parametric Gaussian Processes) Given a Gaussian likelihood $p(y|\mathbf{x}, \sigma^2)$ and kernel k , construct the data matrix $\mathbf{K} = k(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{n \times n}$ from n data points. Again, there is a closed-form Gaussian predictive $p(\mathbf{y}_* | \mathbf{x}_*, \mathcal{D}) = \mathcal{N}(\mu_{\mathbf{y}_*}, \sigma_{\mathbf{y}_*}^2)$,

$$\mu_{\mathbf{y}_*} = \mathbf{k}_*(\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}, \quad \sigma_{\mathbf{y}_*}^2 = \sigma^2 + k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*(\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}_*^\top,$$

where $\mathbf{k}_* = k(\mathbf{x}_*, \mathbf{X}) \in \mathbb{R}^{1 \times n}$.

Due to the inversion of the data matrix used in the predictive distribution (Definition 5), GP regression scales $\mathcal{O}(n^3)$ with the dataset size. This computational complexity has motivated *sparse* approaches to GPs that improve scaling through approximation. Such approximations includes inducing points, which approximate the dataset with fewer (optimized) samples [114], and random features, which approximate the kernel in a Monte Carlo fashion with a finite feature space [87].

While the description of GPs above is focused on the weight-space perspective, GPs also permit a function-space view. The kernel, prior and likelihood can be combined into a *covariance function* $C(\mathbf{X}) = \sigma^2 \mathbf{I} + \sigma_0^2 k(\mathbf{X}, \mathbf{X})$, which in turn describes the GP prior $\mathcal{GP}(m(\mathbf{x}), C(\mathbf{x}))$. Evaluating n inputs returns an n -dimensional multivariate Gaussian. Sampling this Gaussian acts samples (evaluated) functions from the function-space specified as the prior, as shown in Fig. 4. Due to this perspective, GPs are often defined through their covariance function, rather than their feature space or kernel.

3.2 Automatic Differentiation for Nonlinear Function Approximation

While non-parametric models are very expressive, their computational and memory complexity limit them for many large-scale tasks in practice. Moreover, both features and/or kernels are predefined. Such manual choices are especially problematic as the representation of our data has a significant impact on which functions the model can represent. This circumstance naturally leads to the desire to also learn and adapt the features depending on the problem that needs to be solved. For this purpose, we will now consider parametrized feature functions $\phi_\theta(\mathbf{x})$ with learnable parameters θ . For our feature regression model, it took the form $\hat{\mathbf{y}} = \mathbf{f}_\theta(\mathbf{x}) = \mathbf{W}\phi_\theta(\mathbf{x}) = \phi_\theta(\mathbf{x})$. Note that the linear weights of the model can just be viewed as additional fixed parameters of our representation that is to be learned. Describing a model in this fashion recursively, we arrive at what are known as *neural networks* (NN), a biologically-inspired architecture in which nonlinear features are built by

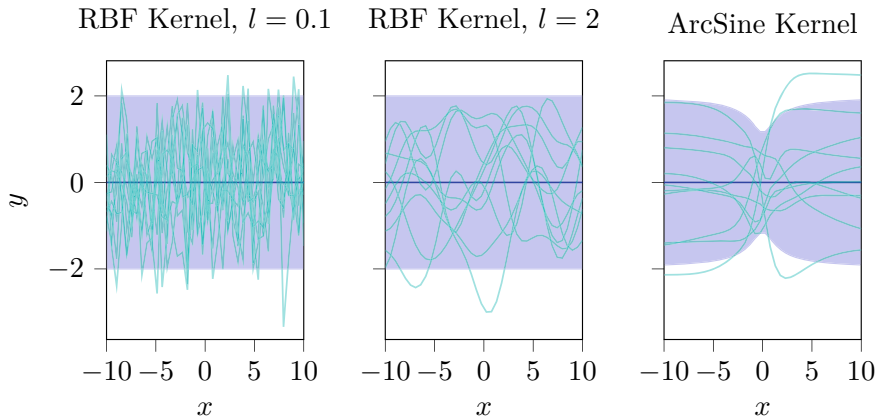


Fig. 4 Illustrating three one-dimensional, zero-mean Gaussian processes priors using different covariance functions. The plots show the GPs in blue (2 standard deviations) and function samples in cyan. Function samples are realized by sampling the $n \times n$ covariance matrix defined linearly between -10 and 10 . Observe the RBF covariance function looks similar to IID Gaussian noise for small lengthscales, as the cross-correlation between inputs is close to zero. For larger lengthscales, the larger cross-correlations manifest through smoother function samples. As the RBF covariance function is stationary, the GP has constant variance along x . Contrast this to the arcsine covariance function, which is non-stationary so the variance varies with x . The function samples are not periodic, and rather emulate the erf activation function of the neural network architecture used to derive the kernel

propagating the intermediate representations through linear transformations and nonlinear ‘activation functions’,

$$\begin{aligned}\hat{y} = \phi_{\theta}(\mathbf{x}) &= \mathbf{W}_n(\dots f_2(\mathbf{W}_2(f_1(\mathbf{W}_1\mathbf{x})))) = \mathbf{W}_n(\dots f_2(\mathbf{W}_2\phi_{\theta_1}(\mathbf{x}))), \\ &= \mathbf{W}_n(\dots f_2(\mathbf{W}_2\mathbf{l}_1)) = \mathbf{W}_n(\dots \phi_{\theta_1, \theta_2}(\mathbf{x})) = \mathbf{W}_n(\dots \mathbf{l}_2),\end{aligned}\tag{3}$$

where each layer essentially consists of first applying a linear transformation $\mathbf{W}$ to the layer input $\mathbf{l}_i$, followed by a nonlinear activation function $f(\cdot)$ to yield the output, which will be essentially the input to the next layer, i.e. $\mathbf{l}_{i+1}(\mathbf{x}) = f_{i+1}(\mathbf{W}_{i+1}\mathbf{l}_i(\mathbf{x}))$. The weights of the linear transformation contain the parameters that are to be learned, the activation functions remain a choice to be made beforehand, and the layer input depends on the output of the previous layer. Examples for activation functions are the sigmoid, hyperbolic tangent or the rectified linear activation function (ReLU). When combined, these layers produce feature that are optimized for fitting data. The activation functions influence the nature of the features. For example, ReLU activations produce piecewise-linear features, while tanh activations produce features that transitional smoothly between two saturation points. With this model structure, a NN is defined by the width of each layer and the number of layers. An additional benefit of this structure is that the model can perform *representation learning* by learning a lower-dimensional intermediate

representation of a high-dimensional input, such as RGB images. These learned intermediate features indicate increased generalization performance through the ability to compress the representation.

However, the question remains how to actually obtain the parameters, given the complexity of the multi-layered representation and the nonlinear activation functions. We will make use of gradient-based optimization and motivate automatic differentiation. Consider the loss that is obtained by making a prediction with a two layered neural network, i.e., $L(\mathbf{y}, \hat{\mathbf{y}})$ with prediction $\hat{\mathbf{y}}(\mathbf{x}) = \mathbf{f}_2(\mathbf{W}_2 \mathbf{l}_1) = \mathbf{f}_2(\mathbf{W}_2 \mathbf{f}_1(\mathbf{W}_1 \mathbf{x}))$. As we want to minimize the loss, we have to adapt every parameter in the opposite direction of the gradient of the loss function with respect to the parameter. Assume that we calculated the prediction $\hat{\mathbf{y}}$ incrementally, in the forward path

$$\mathbf{l}_1 = \mathbf{f}_1(\mathbf{W}_1 \mathbf{x}), \quad \mathbf{l}_2 = \mathbf{f}_2(\mathbf{W}_2 \mathbf{l}_1), \quad L(\mathbf{y}, \hat{\mathbf{y}}) = L(\mathbf{y}, \mathbf{l}_2).$$

We now determine each parameter's influence on the loss by taking the appropriate gradient in the backwards pass,

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} = \frac{\partial L}{\partial \mathbf{l}_2}, \quad \frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{l}_2} \frac{\partial \mathbf{l}_2}{\partial \mathbf{W}_2}, \quad \frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \mathbf{l}_2} \frac{\partial \mathbf{l}_2}{\partial \mathbf{l}_1} \frac{\partial \mathbf{l}_1}{\partial \mathbf{W}_1}.$$

As the individual gradients that emerge from the use of the chain rule can be calculated straightforwardly, and are actually reused for calculating the gradients for the layers closer to the input, this combined procedure of forward and backward pass is tractable and does scale to very deep networks. This combination of dynamic programming and automatic differentiation is known as the backpropagation algorithm. Finally, all the parameters are updated according to $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \nabla_{\boldsymbol{\theta}} L$ with learning rate α .

It has been shown that feedforward neural networks are extremely powerful representations and there exist theoretical proofs that one single layered neural network is sufficient to represent any function [18, 38]. Yet, deeper networks can be seen as performing a hierarchical decision process which in turn often simplifies the representation of complex functions and results in requiring less parameters. While the gradient based optimization scheme is very efficient it also introduces several difficulties. Just to name two examples: the repetitive multiplication of gradients causes very small gradients for the initial layers of deep networks, which slows optimization. This phenomenon is known as ‘vanishing gradients’. Secondly, vast amount of parameters and their ambiguous representation results in many local optima. To counteract these issues, architectures with skip connections and the addition of momentum terms in the parameter update have been proposed. For more background information and details, we refer the interested reader to [2, 29, 51, 99].

The combination of neural network features and Bayesian linear regression is known as the neural linear model (NLM) [50, 77]. While such methods provide NNs with uncertainty quantification, they are weaker at expressing uncertainty compared to nonparametric GPs for fixed features. This weakness results from optimization

Algorithm 3 Expectation maximization for model learning

Initialization: Initial model parameters θ_0 , measurements Y , initial state distribution $p(x_1)$, dynamics likelihood $p(x_{t+1}|x_t, \theta)$ and measurement model $p(y_t|x_t, \theta)$
repeat
 /* E-step: Bayesian filtering and smoothing of latent state trajectory given model and data
 Trajectory posterior $p(X|Y, \theta_t)$ by calculating $p(x_t|y_{1:T}, \theta_t)$ for $t=1:T$
 /* M-step: Optimize θ for expected log-likelihood objective given latent state trajectory
 $\theta_{t+1} = \arg \max_{\theta} \mathbb{E}_{X \sim p(\cdot|Y, \theta_t)} [\mathcal{L}(X, Y, \theta)]$
until converged
return Learned model θ

for the data when acquiring the learned features. Standard training does not enforce feature diversity, therefore the model can be relatively overconfident outside of the data distribution and especially between data points. As a result, such models need to be trained in a way that specifically encourages feature diversity [50, 123, 124]. More generally, Bayesian neural networks (BNN) have a weight distribution over all parameters [72]. The nonlinear nature of neural networks means that approximate inference is required for inference over all parameters, and poses a significant challenge for achieving accurate inference for large models and datasets [40].

4 Model Learning for Control

After covering optimal control in Sect. 2, a logical approach for data-driven control is to first learn an approximate dynamics model from data, and then apply optimal control algorithms using that model. This approach is ubiquitous in practical robotics, and is commonly known under the moniker of ‘model-based reinforcement learning’ (MBRL) or ‘optimal control with learned forward models’ in the context of Robot Learning [21]. The idea of identifying model parameters for dynamical systems from data is well-established in robotics as *system identification* [3, 112] while the model structure is often computed using physical insights. In Robot Learning, ‘Model Learning’ refers to the combination of system parameter and structure identification using machine learning. This section will discuss model learning from the perspective of Robot Learning, and follow-up with an overview of how optimal control can be effectively applied to approximate dynamics models.

4.1 Model Learning: System Identification Meets Machine Learning

System identification generally requires specifying a parametrized dynamics model derived from physics, and then proposes a means of learning the model parameters from data. The key consideration is learning model parameters that are *physically*

plausible, for example positive masses and lengths, which may require constrained optimization or reparameterization to preserve the constraint [27].

In Sect. 3, we introduced feature regression and neural network function approximation, which fits data using more general, expressive models that take no inspiration from physics. The use of such flexible models reduces the necessity to carefully design and implement the model, but means more data coverage over the domain of interest is required to fit these models sufficiently as generalization is no longer guaranteed. In other words, physical plausibility is now enforced through data coverage and not the inherent structure, which is problematic if the model is used outside of the data distribution (OOD) or in increasingly high-dimensional spaces. The benefit of these unstructured models is that they make no assumptions about the data, which make them more widely applicable and simpler to implement. Crucially, they are also able model phenomena that may not be captured by the physics model, such as friction effects and other disturbances.

Viewing physics-derived models as ‘white-box’ models, and unstructured function approximators as ‘black-box’ models, we motivate an intermediate approach of ‘grey-box’ modelling. Such ‘grey-box’ modelling could involve augmenting the function approximators with inductive biases such as energy conservation [61], or combining physics-based models with black-box components, capturing the residual error [75] or complex friction phenomena in the actuators [62]. The challenge of grey-box modelling is capturing the necessary trade-off in model structure, and training the model such that each component learns its intended phenomena adequately. While conceptually the white- and black-box components are distinct, during learning they are inherently coupled.

Another important objective across the whole spectrum of models is the training objective and algorithm. Dynamics models can be trained using forward or inverse dynamics [76]. Forward dynamics can be trained for single-step prediction, or could be trained to minimize multi-step prediction error to improve long horizon performance [76]. Gradient-based optimization can also be used, either using analytical gradients or automatic differentiation. However, care must often be taken to ensure physically plausible parameters using constraints or virtual parameters [62, 106, 117]. Taking the inference perspective, the expectation-maximization (EM) algorithm can be used [23], which naturally incorporates model-based data smoothing into the system identification procedure (Algorithm 3). While EM is performed exactly for linear Gaussian dynamical systems [103], it required approximate inference for nonlinear dynamical systems [28, 101].

4.2 Optimal Control for Approximate Dynamics Models

Combining the ideas from Sects. 2 and 4.1, a viable approach to data-driven control is combining optimal control algorithms with learned dynamics models. Compared to model-free methods (see Sect. 6), that perform more of a trial-and-error approach for learning, combining finite-horizon optimization with approximate dynamics

incorporates an inductive bias that improved the sample efficiency of learning, reaching better performance with less experience. For example, assuming local extrapolation of the approximate dynamics, the planning of optimal control should direct data-collection towards regions where either the model is uncertain or constitute approximately optimal trajectories. Model-free exploration lacks such an informed search strategy. A key pitfall of model-based, data-driven control is designing optimal control solvers to operate effectively under model error. For example, optimal control solvers are encouraged to exploit model errors that improve control performance, that could lead to catastrophic controls that damage the robot, as they have no indication of the model accuracy. For example, consider a navigation task of a maze where the robot assumes that walls do not exist. Greedily optimizing for navigation would lead to repeated crashing rather than a safer exploration strategy. One means to alleviate this greedy phenomena is to use Bayesian models for the dynamics, and use the epistemic uncertainty as a measure for possible prediction error [100]. Maintaining a distribution over the future state trajectory, the expected future reward evaluation is regularized by the uncertainty. To appreciate why, consider a quadratic state reward with a Gaussian belief in the state. Due to the quadratic law, the expected reward incorporates the state variance as well as the mean,

$$r(x) = -x^2, \quad x \sim \mathcal{N}(\mu, \sigma^2), \quad r(\mathbb{E}[x]) = -\mu^2, \quad \mathbb{E}[r(x)] = -\mu^2 - \sigma^2.$$

As $\mathbb{E}[r(x)] \leq r(\mathbb{E}[x])$, incorporating the uncertainty in our state predictions into reward evaluation will regularize the expected reward as long as uncertainty exists. However, computing an accurate trajectory distribution can pose a challenge in the nonlinear case due to the necessity of approximate inference methods.

Another issue with model-based reinforcement learning is long-horizon prediction. Compounding prediction errors render long-horizon planning inaccurate, especially if the model is trained using one-step prediction error. To avoid these prediction errors, model predictive control (MPC) [66] is often used for MBRL. MPC uses replanning each time step over a shorter time horizon, in order to reach to prediction errors or disturbances. The consequence of this adaptivity is greater computational cost due to the repeated replanning computation. Another benefit of MPC is the amortization of the optimal control problem, which can avoid convergence to local optima for complex tasks such as locomotion [113].

Finally, optimal control methods are designed for true dynamics functions, which are generally continuous and smooth by definition or for numerical reasons, e.g. smooth contact models. Learned dynamics models, especially neural networks with discontinuous activations, do not behave in this smooth fashion and may have a ‘rough’ quality. Due to this aspect, gradient-based optimal control solvers are less suitable for optimization, as they will likely suffer numerical issues or get stuck in local optima. Some approximate model classes, such as kernel-based and time-varying linear models, are well-suited to gradient-based optimization [20, 55]. As a result of this phenomena, sample-based black-box optimization methods, such as

the cross-entropy methods (CEM) [95], have become a popular MPC solver for deep MBRL [16, 127].

5 Policy Representations for Robotics

A policy $\pi(\mathbf{a}|\mathbf{s})$ is a function that maps a state $\mathbf{s} \in \mathcal{S}$ to an action $\mathbf{a} \in \mathcal{A}$. In Sect. 2, the optimal control derivation returned the form of the policy, which was derived to be discrete or time-varying linear. In the context of learning a robot policy by reinforcement learning (Sect. 6) or imitation learning (Sect. 7), the choice of the parameterized model to represent a policy is no longer defined and will directly influence in the behaviour of our robot. We desire the policy to have properties that facilitate safe and effective learning in continuous state and action spaces. Desirable properties for such policies could include *stability*, *smoothness* or *adaptability* to name a few.

5.1 From Torque Control to Motion Generation

To properly impose the desired properties in the policies, it is common to abstract from the robot control signals and model our policies as motion generators. In robotics, the common action space is the torque signal $\boldsymbol{\tau}$ applied on each joint of the robot. The state $\mathbf{s} = [\mathbf{q}, \dot{\mathbf{q}}, \mathbf{c}]^\top$ is composed of both the joints position and velocity information, $\mathbf{q}, \dot{\mathbf{q}}$ and the additional sensors input $\mathbf{c}$. Thus, a robot policy is a function that computes a desired torque signal given both the joint's and the sensors input, $\boldsymbol{\tau} = \pi(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{c})$. To properly integrate the desired properties of stability, smoothness or adaptability, we can abstract from the torque signal and control the robot directly in the motion space, $\ddot{\mathbf{q}} = \pi(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{c})$. Given the robot's dynamics model is usually known,

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \boldsymbol{\tau} - \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q}),$$

with $\mathbf{M}$ is the inertia matrix, $\mathbf{c}$ the Coriolis and centrifugal forces and $\mathbf{g}$ the gravity vector; we can smartly model the torque policy to set the desired acceleration in the robot. With $\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_{\text{des}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$, the robot's joint acceleration is $\ddot{\mathbf{q}} = \ddot{\mathbf{q}}_{\text{des}}$ as long as the inertia matrix is invertible. Given this change of variables, the policy becomes a dynamical system. As we will show along this section, imposing a set of desired properties in our policy models becomes trivial when modelling the policies as dynamical systems. These motion generator type policies are widely known as *movement primitives*.

5.2 Types of Movement Primitives

The literature in movement primitives is vast. Depending on the desired properties, a wide set of policy models are proposed that can cover different desired properties from stroke-motion primitives, periodic primitives, orientation primitives or space-constrained primitives. In this section we introduce three motion primitives that are representative of different classes of primitives. We will introduce, phase and state dependant primitives $\pi(\mathbf{q}, \dot{\mathbf{q}}, \zeta)$, phase dependant primitives $\pi(\zeta)$ and state dependant primitives $\pi(\mathbf{q}, \dot{\mathbf{q}})$.

Dynamic Movement Primitives (DMP) These primitives represent a globally-stable, robust and adaptable parameterized dynamic system that can learn highly nonlinear dynamics [39, 98]. The model is composed of two elements: a state-dependant second order linear attractor and a time dependant nonlinear forcing term that reshapes that linear dynamics to follow the desired nominal behavior.

Definition 6 (Dynamic Movement Primitives) A dynamic movement primitive for a single degree-of-freedom trajectory q , of a point-to-point movement is defined by

$$\tau \ddot{q} = \alpha(\beta(g - q) - \dot{q}) + f(\zeta), \quad \tau \dot{\zeta} = -\gamma \zeta,$$

with α and β the gains of the linear attractor, g the target position, γ the gain for the phase dynamics and f the nonlinear forcing term. The forcing term is modelled to become zero when ζ approaches to zero.

A common choice to represent the forcing term is by a linear weighting of a set of basis functions, ϕ ,

$$f(\zeta) = \frac{\sum_{k=0}^K w_k \phi_k(\zeta)}{\sum_{k=0}^K \phi_k(\zeta)},$$

with each basis function modelled by an exponential basis function,

$$\phi_k(\zeta) = \exp(-(\zeta - \mu_k)^2 / \sigma_k).$$

The exponential basis function tends to zero the further ζ is from the center μ_k , so the forcing term will tend to zero the bigger the phase is, $\lim_{\zeta \rightarrow \infty} f(\zeta) = 0$. Additionally, dynamic movement primitives allows to adapt to different target positions by changing the target state g while maintaining the nonlinear shape due to the forcing term f .

Probabilistic Movement Primitives Paraschos et al. [78] represent phase dependant, probabilistic motion primitives for both stroke and periodic movements. The proposed model is composed of two parts. The time dependant desired position is computed by a linearly-weighted sum of basis functions,

$$p(\mathbf{q} \mid \zeta, \mathbf{w}) = \psi(\zeta)^\top \mathbf{w} + \epsilon_q, \quad \epsilon_q \sim \mathcal{N}(0, \sigma^2),$$

with additional white noise. Then, rather than having fix values for the weights $\mathbf{w}$ as in DMP, we assume they are sampled from a Gaussian distribution,

$$\mathbf{w} \sim \mathcal{N}(\mathbf{w} \mid \mu_w, \Sigma_w).$$

This is closely related to the Bayesian linear regression model from Sect. 3, with phase ζ as the input and the trajectory as the target, but it is used as *generative* model [5] rather than a regression model. Training these models is similar to direct regression when a unique latent trajectory is assumed. For modelling a more diverse distribution over trajectories, an EM procedure is used instead.

Definition 7 (Probabilistic Movement Primitives) A probabilistic movement primitive (ProMP) represents a distribution in the position, $\mathbf{q}$ in terms of the weights $\mathbf{w}$, that belongs to a distribution parameterized by θ ,

$$\begin{aligned} p(\mathbf{q} \mid \zeta, \theta) &= \int_{\mathbf{w}} p(\mathbf{q} \mid \zeta, \mathbf{w}) p(\mathbf{w} \mid \theta) d\mathbf{w}, \\ &= \int_{\mathbf{w}} \mathcal{N}(\mathbf{q} \mid \psi(\zeta)^\top \mathbf{w}, \sigma^2 \mathbf{I}) \mathcal{N}(\mathbf{w} \mid \mu_w, \Sigma_w) d\mathbf{w}, \\ &= \mathcal{N}(\mathbf{q} \mid \psi(\zeta)^\top \mu_w, \psi(\zeta)^\top \Sigma_w \psi(\zeta) + \sigma^2 \mathbf{I}). \end{aligned}$$

In contrast with dynamic movement primitives that directly provides the desired acceleration; ProMP proposes to learn a distribution in the position. Therefore, to control the robot we assume a tracking controller that exactly reproduces the given trajectory distribution.

Stable Vector Fields (SVF) Proposed a set of state dependant dynamic systems that are stable by construction Khansari-Zadeh and Billard [44, 45]. Diffeomorphism-based SVFs can represent nonlinear dynamics while remaining stable towards a target or a limit cycle (Fig. 5). In contrast with DMPs or ProMPs, these motion primitives depend only in the robot state and do not require an additional phase variable. This property makes them an ideal choice for reactive motion generation or human-robot interaction, given they will naturally adapt their movements without the requirement of a phase estimation. One possible approach to build SVFs is by exploiting a set of diffeomorphic functions. Diffeomorphism-based SVFs [73, 89, 119] are a family of dynamic systems that are composed of

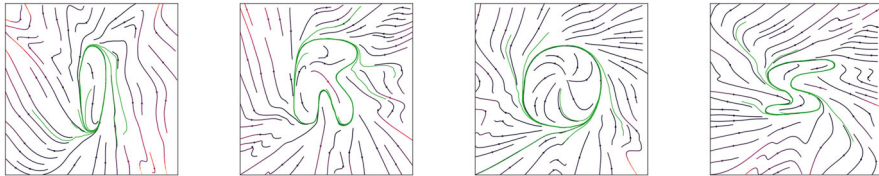


Fig. 5 A visual example of learned stable vector fields given a set of letter shape trajectories demonstrations [119]. The field produces stable limit cycles that reproduce the expert demonstrations in a time-invariant fashion

two parts. First, a globally stable linear dynamic system is defined in a latent space $\mathcal{Z} = -\mathbf{z}$. Then, a parameterize diffeomorphic function, Φ is given between the observation space $\mathcal{Q}$ and the latent space $\mathcal{Z}$.

Definition 8 (Diffeomorphism-Based Stable Vector Fields) Given a globally stable dynamic system $\dot{\mathbf{z}} = g(\mathbf{z})$ in a latent space $\mathcal{Z}$ and a diffeomorphic map from the observation space $\mathcal{Q}$ to the latent space $\mathcal{Z}$, $\Phi : \mathcal{Q} \rightarrow \mathcal{Z}$, then the dynamics in the observation space,

$$\dot{\mathbf{q}} = J_{\Phi}^{-1} g(\Phi(\mathbf{q})),$$

are also globally stable.

SVF usually represent first-order stable dynamic systems. To control the robot, we can apply an error correction in the acceleration,

$$\ddot{\mathbf{q}}_{\text{des}} = \alpha(\dot{\mathbf{q}}_{\text{des}} - \dot{\mathbf{q}}).$$

To model the diffeomorphic function Φ , several representations has been proposed from Gaussian processes to invertible neural networks [119], such as those used in normalizing flows.

6 Reinforcement Learning

A compelling notion from cognitive science and artificial intelligence is the ability to learn optimal decision making from experience, improving performance through trial and error in a systematic fashion [110]. Section 2 showed how to solve optimal control using value function methods, namely value iteration (Algorithm 1), policy iteration (Algorithm 2). These methods recover the optimal policy by maximizing the state-action value function $\pi^*(s) = \arg \max_a Q^*(s, a)$. Although theoretically

sound, these methods make strong assumptions that do not always hold in practice. Section 2.3 discussed the challenges and approximations required for nonlinear optimal control, when the value function and optimal policy can not be obtained exactly. Section 4 highlighted the strong assumption of knowing the dynamics model, and the challenges faced when combining optimal control solvers with approximate dynamics models.

This section looks at model-free reinforcement learning, which covers a broad range of methods. These approaches include value-based methods (Sect. 6.1), which develop on the idea of approximate dynamic programming. An alternative strategy is to optimize the policy directly, using a broad of numerical methods (Sect. 6.2). Policy gradient approaches use gradient estimation methods to obtain low variance updates from policy evaluation data, and policy search methods in general use a range of numerical methods for optimizing the policy. We will discuss episodic relative entropy policy search, which uses inference-based methods for proximal optimization. When moving from optimal control to reinforcement learning, another key aspect is *exploration*, which is arguable orthogonal to optimal control but essential for effective exploration.

6.1 Value Function Methods

In Sect. 2.3, we discussed approximate dynamic programming, where value functions and policies are obtained using function approximation. As this approach is already a combination of optimal control and machine learning methods, it translates naturally into a reinforcement learning approach.

Definition 9 (Infinite Horizon Objective) The infinite horizon discounted return is,

$$J_\pi = \mathbb{E}_{s_0 \sim \mu_0, s_{t+1} \sim \mathcal{P}(s_t, \mathbf{a}_t), \mathbf{a}_t \sim \pi(s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \mathbf{a}_t) \right] = \mathbb{E}_{s_0 \sim \mu_0} \left[V^\pi(s_0) \right],$$

with initial state distribution μ_0 , transition probability distribution $\mathcal{P}$ and discount factor $0 \leq \gamma < 1$ that trades off the short term and long-term rewards.

In contrast to Sect. 2, classical reinforcement learning is typically directly formulated for the infinite horizon objective as in Definition 9 [110], which is easier in the discounted formulation introduced here (it can be introduced straightforwardly in Sect. 2 but was omitted for simplicity). Definition 10 introduces, the state and state-action value functions for continuous spaces, and the corresponding optimal ones from Bellman's principle of optimality.

Algorithm 4 Temporal difference learning for acquiring V^π

Init: $V_0(\mathbf{s}) \leftarrow 0, t = 0, \alpha \in [0, 1]$
repeat $t = t + 1$
 Observe transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t)$ using policy π
 /* Compute TD error
 $\delta_t = r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)$
 /* Update value function
 $V_{t+1}(\mathbf{s}_t) = V_t(\mathbf{s}_t) + \alpha \delta_t$
until Convergence of V
return Value function V^π

Definition 10 (Value Functions of Arbitrary Policies) The value functions induced by policy π , used in policy iteration,

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{s})} [r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim \mathcal{P}(\mathbf{s}, \mathbf{a})} [V^\pi(\mathbf{s}')]],$$

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim \mathcal{P}(\mathbf{s}, \mathbf{a}), \mathbf{a}' \sim \pi(\mathbf{s})} [Q^\pi(\mathbf{s}', \mathbf{a}')].$$

The optimal value functions, used in value iteration,

$$V^*(\mathbf{s}) = \max_{\mathbf{a}} r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim \mathcal{P}(\mathbf{s}, \mathbf{a})} [V^*(\mathbf{s}')],$$

$$Q^*(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim \mathcal{P}(\mathbf{s}, \mathbf{a})} \left[\max_{\mathbf{a}'} Q^*(\mathbf{s}', \mathbf{a}') \right].$$

With ADP, given a set of N transition samples from a policy acting on an environment $\mathcal{D} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1, \dots, N}$, and without access to the transition model, we use the samples to learn the V and Q functions using *temporal difference* (TD) learning [109]. To understand TD learning let us start first with its tabular version. Given a single transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t)$, we want to update the V function. If we have an estimate for the value function, and using Definition 10 we write $V(\mathbf{s}_t) \approx \hat{V}(\mathbf{s}_t) = r_t + \gamma V(\mathbf{s}_{t+1})$. We define $\delta_t = \hat{V}(\mathbf{s}_t) - V(\mathbf{s}_t) = r_t + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t)$ as the TD error, which is then used to update the current value as,

$$V_{\text{new}}(\mathbf{s}_t) = V(\mathbf{s}_t) + \alpha \delta_t = (1 - \alpha)V(\mathbf{s}_t) + \alpha(r_t + \gamma V(\mathbf{s}_{t+1})),$$

where $\alpha \in [0, 1]$ is an hyperparameter to control the update. With $\alpha=1$ the new value is solely based on the prediction. Note that if the TD error is positive, then the value increases, and otherwise it decreases. The TD learning sampled-based version for computing the value function of a policy π in discrete spaces is presented in algorithm 4.

To obtain an optimal control policy from a value function we instead learn the state-action value function Q , whose TD update equations for Q -functions are,

$$\begin{aligned}\delta_t &= r_t + \gamma Q_t(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q_t(\mathbf{s}_t, \mathbf{a}_t), \\ Q_{t+1}(\mathbf{s}_t, \mathbf{a}_t) &= Q_t(\mathbf{s}_t, \mathbf{a}_t) + \alpha \delta_t.\end{aligned}$$

There are two ways to choose the action $\mathbf{a}_{t+1}$ used for the next-state estimate. In the SARSA⁴ algorithm, $\mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t)$ [96]. These samples are *on-policy* and produce estimates of the current Q -function. In the Q -learning algorithm, $\mathbf{a}_t = \arg \max_{\mathbf{a}} Q_t(\mathbf{s}_t, \mathbf{a})$ [122]. These samples are *off-policy*, because $\mathbf{a}_{t+1}$ is obtained using Q rather than π , and produce Q -function estimates of the *optimal* policy. Note that this action selection only tells us which action we should select to estimate the Q -function at the next-state.

As the agent is traversing the environment, we still need to decide which action to take in the current state. Should we take the best action according to our current estimate, or should we take another action at random? Behaving optimally and using the best action is known as ‘greedy’ behaviour, and may result in converging to local optima. Random actions perform exploration, but are suboptimal policies. This phenomenon is a well-established problem in reinforcement learning literature, known as the *exploration-exploitation* trade-off [110]. Two common strategies to ensure exploration are the ε -greedy and softmax policies (Definition 11).

Definition 11 (Exploration Policies) Two common strategies for action exploration are,

$$\varepsilon\text{-greedy policy: } \pi(\mathbf{a}|\mathbf{s}) = \begin{cases} \arg \max_{\mathbf{a}'} Q(\mathbf{s}, \mathbf{a}') & \text{with probability } 1 - \varepsilon, \\ \text{other action} & \text{with probability } \varepsilon, \end{cases}$$

randomly switches between optimal and random actions.

$$\text{Softmax policy: } \pi(\mathbf{a}|\mathbf{s}) = \frac{\exp(\beta Q(\mathbf{s}, \mathbf{a}))}{\sum_{\mathbf{a}'} \exp(\beta Q(\mathbf{s}, \mathbf{a}'))}.$$

samples from a Boltzmann distribution, which originated from statistical thermodynamics, $p(\mathbf{a}|\mathbf{s}) \propto \exp(-E(\mathbf{s})/T)$ with energy $E(\mathbf{s})$ and temperature T , where the Q -function is the negative energy function and β is the inverse temperature.

Using these exploration strategies, the Q -learning algorithm [122] (Algorithm 5) is a classic method to learn an optimal Q -function, and the basis for many of the

⁴ SARSA stands for state–action–reward–state–action.

Algorithm 5 Q -Learning

Init: $Q(s, \mathbf{a}) \leftarrow$ arbitrarily, $\forall s \in \mathcal{S}, \forall \mathbf{a} \in \mathcal{A}$, $Q(s \text{ is terminal}, \cdot) = 0$, $\alpha \in [0, 1]$
repeat for each episode
 /* Sample an initial state
 $\mathbf{s} \leftarrow \mathbf{s}_0$
repeat for each step
 Sample $\mathbf{a}$ from exploration policy (ε -greedy, softmax)
 Observe transition $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$
 /* Update the Q -function
 $Q(\mathbf{s}, \mathbf{a}) \leftarrow Q(\mathbf{s}, \mathbf{a}) + \alpha [r + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')]]$
 $\mathbf{s} \leftarrow \mathbf{s}'$
until $\mathbf{s}$ is a terminal state
until Convergence of Q
return Q^*

recent deep reinforcement learning algorithms such as deep Q-learning (DQN). In the tabular version, this algorithm is proven to converge as long as all states and actions are infinitely visited and $\alpha \rightarrow 0$ as training progresses.

Due to the curse-of-dimensionality, we cannot use the tabular version of Q -learning for continuous state spaces. For policy evaluation, we can approximate the value function with a function approximator, e.g. a neural network with parameters ϕ , $Q^\pi(\mathbf{s}, \mathbf{a}) \approx Q_\phi(\mathbf{s}, \mathbf{a})$. Assuming Q_ϕ is differentiable w.r.t. ϕ , to find the optimal parameters we can use a combination of TD learning and stochastic gradient descent on a mean squared error (MSE) of sampled transitions $\mathcal{D} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1, \dots, N}$ following a policy π . Using bootstrapping, we iteratively optimize the value function parameters $\phi^* = \arg \min_{\phi} \mathcal{L}_{\text{BS}}(\phi)$,

$$\mathcal{L}_{\text{BS}}(\phi) \approx \frac{1}{N} \sum_{i=1}^N \left(\hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}_i) - Q_\phi(\mathbf{s}_i, \mathbf{a}_i) \right)^2,$$

$$\hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi(\cdot | \mathbf{s})} [Q_\phi(\mathbf{s}'_i, \mathbf{a}')],$$

$$\phi_{k+1} = \phi_k - \alpha_k \frac{d}{d\phi} \mathcal{L}_{\text{BS}}(\phi),$$

where $\hat{Q}^\pi$ is a bootstrapped target. In the context of value function learning, bootstrapping (BS) means that we use the old approximation to get the target values for a new approximation. Given an offline dataset of transitions, and a parametrized approximation Q_ϕ , the fitted Q -iteration algorithm uses a similar approach to Q -learning to find the (local) optimal parameters ϕ^* that minimize the bootstrapped MSE [24, 93].

Learning value function approximations with TD learning and stochastic gradient descent presents several challenges. One is that it is not a ‘proper’ gradient descent routine, because the target values change after each parameter update. Nevertheless, for the target we ignore this aspect and keep the parameters ϕ

Algorithm 6 Generic model-free algorithm to learn π^*

```

Init: current policy  $\pi_k$ 
repeat
  /* Exploration
  Generate (multiple) trajectories  $\tau$  following the current policy  $\pi_k$ 
  /* Policy Evaluation
  Assess the quality of the whole trajectory or single actions
  /* Policy Update
  Compute new policy  $\pi_{k+1}$  using the trajectories and evaluations
until Convergence of  $\pi$ 
return  $\pi^*$ 

```

fixed [67]. This approach introduces a bias in the optimization, which means we are optimizing a different objective than the mean squared error. In certain cases the optimization procedure can even diverge, for example when training with off-policy samples [120]. One workaround is to maintain two value functions, where one is used for the target values and another is updated, and the target function is updated at a slower pace [67].

6.2 Policy Search Methods

Value function methods are well motivated and developed, but if we are only interested in the optimal policy, the added complexity of modelling the approximate value functions and ill-posed bootstrap training can introduce many issues. Many of these problems can be avoided by performing policy search [21, 80]. Instead of learning value functions to obtain a policy, we can learn the policy directly. Moreover, using task-specific policies is possible and beneficial, as discussed in Sect. 5. For instance, for a robotic task it seems more reasonable to learn the parameters of a dynamic movement primitive instead of a generic Q -function [70, 98].

For value-based methods, we used the Q -function to sample from a Boltzmann distribution. For policy search methods, we can parametrize the policy distribution explicitly. Similarly, for a Gaussian policy, which are popular for continuous control, we have $\pi(\mathbf{a}|\mathbf{s}; \theta) = \mathcal{N}(\mu_\theta(\mathbf{s}), \Sigma_\theta(\mathbf{s}))$ [31], a Gaussian process as seen in Sect. 3. To make our explanation of model-free policy search clearer, we describe a generic procedure in Algorithm 6.

An important distinction in policy search methods is whether we learn the policy parameters θ in an episodic or step-based manner. The distinction of both methods is in the policy evaluation and update parts in Algorithm 6.

6.3 Episodic Policy Search

In episodic policy search, we learn a high-level policy that encodes a search distribution $\pi(\theta; \omega)$ over the parameters of a lower-level policy $\pi(\mathbf{a}|\mathbf{s}; \theta)$. For instance, $\pi(\theta; \omega)$ can be a Gaussian distribution $\pi(\theta; \omega) = \mathcal{N}(\theta; \omega = \{\mu, \Sigma\})$. Often in this context the lower-level policy is a deterministic policy $\mathbf{a} = \pi(\mathbf{s}; \theta)$, which reduces the variance in the returns and allows for smooth exploration in the real robot [21]. Algorithm 7 shows the generic approach to episodic policy search.

Definition 12 (Episodic Policy Search) Formally, episodic policy search solves the problem,

$$\omega^* = \arg \max_{\omega} J_{\omega} = \arg \max_{\omega} \mathbb{E}_{\theta \sim \pi(\theta; \omega)} [R(\theta)],$$

where $R(\theta)$ is the expected episodic return of the lower-level policy.

Using the Gaussian process perspective from Sect. 3, we can see the high-level as the weight distribution that's iteratively updated using the data (episodic experience) and the lower-level policy is the regression model itself.

Policy Gradient Solving the policy update in Algorithm 7 without any constraints involves maximizing an expectation. For that we can resort to gradient-based methods, which include the computation of a gradient estimate, followed by a policy update by applying that gradient with a learning rate α , $\omega_{k+1} = \omega_k + \alpha \nabla_{\omega} J_{\omega}$, where the gradient can be estimated using methods such as finite differences or likelihood ratio gradients (Definition 13) [69, 79].

Algorithm 7 Generic episodic policy search algorithm

Init: initial upper-level policy $\pi(\theta; \omega_0)$

repeat

/* Exploration

Sample lower-level policies from the upper-level distribution $\theta^i \sim \pi(\theta; \omega_k), i = 1, \dots, N$

/* Policy Evaluation

Collect returns $\mathcal{D}_{\text{episode}} = \{\theta^i, R^i\}_{i=1, \dots, N}$, with $R^i = \sum_{t=0}^{T-1} r_t^i$

/* Policy Update

Obtain a new policy $\pi(\theta; \omega_{k+1})$ based on $\mathcal{D}_{\text{episode}}$

until Convergence of $\pi(\theta; \omega)$

return $\pi(\theta; \omega^*)$

Definition 13 (Likelihood Ratio Episodic Policy Gradient) The likelihood ratio gradient of the expected return $J_\omega = \mathbb{E}_{\theta \sim \pi(\theta; \omega)} [R(\theta)]$ is given by,

$$\nabla_\omega J_\omega = \nabla_\omega \int \pi(\theta; \omega) R(\theta) d\theta \approx \frac{1}{N} \sum_{i=1}^N \nabla_\omega \log \pi(\theta^i; \omega) R^i.$$

Usually an ω -independent baseline b is subtracted to R^i to obtain a lower variance gradient estimate, while keeping the estimator unbiased.

Probabilistic Search To overcome the premature convergence seen with some policy gradient methods, an alternative is to perform a gradient step by considering the underlying probability metric [125]. However, in this approach a learning rate is still an hyperparameter to be tuned. A different direction is to write the optimization to ensure that policy updates are only ε -apart in distribution space, according to the KL-divergence. This idea is behind the episodic version of the relative entropy policy search (REPS) algorithm (Definition 14) [81], which does not solve the policy update by gradient updates, but rather by solving the constrained optimization problem in closed-form using Lagrangian multipliers. The update resembles the recursive Bayesian update, see Sect. 3, where the exponential return acts as an unnormalized likelihood. As a result, approximate inference methods such as importance sampling as used to update the parameter distribution (Fig. 6).

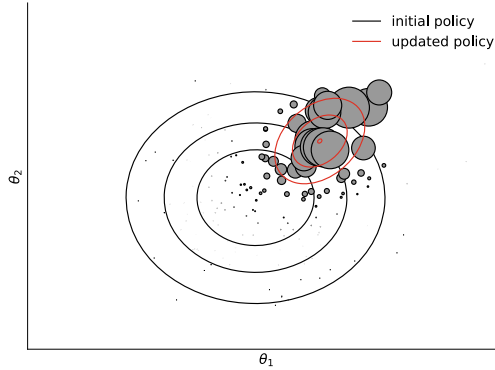


Fig. 6 Illustration of a policy update in parameter space under a Gaussian parameterization. The grey circles represent samples from the initial policy, and their size the corresponding reward. The updated policy Gaussian (red) was approximated from the weighted samples. The REPS algorithm applies a constraint to this update in terms of the KL divergence between the posterior and prior parameter distributions

Definition 14 (Relative Entropy Policy Search) Episodic policy search can be formulated as a relative entropy search problem,

$$\max_{\pi(\theta; \omega)} \int \pi(\theta; \omega) R(\theta) d\theta \quad \text{s.t. } D_{\text{KL}}[\pi(\theta; \omega) \| q(\theta)] \leq \varepsilon, \quad \int \pi(\theta; \omega) d\theta = 1,$$

where in a sequential policy update q is usually the previous (or initial) policy. The closed-form solution to this maximization problem is,

$$\pi(\theta; \omega) \propto q(\theta) \exp(R(\theta)/\eta^*),$$

where η^* is the Lagrangian multiplier that results from minimizing the dual function $\eta^* = \arg \min_{\eta} \eta \varepsilon + \eta \log \int q(\theta) \exp(R(\theta)/\eta) d\theta$.

6.4 Step-Based Policy Search

Contrary to episodic policy search, which explores in parameter space, in step-based methods the exploration is done in the action space by the lower-level policy $\pi(\mathbf{a}|\mathbf{s})$ [21]. Algorithm 8 shows the generic approach to step-based policy search.

Definition 15 (Step-Based Policy Search) Formally, step-based policy search solves the problem,

$$\theta^* = \arg \max_{\theta} J_{\theta}, \quad J_{\theta} = \mathbb{E}_{\tau \sim \mu_0, \mathcal{P}, \pi_{\theta}} [R(\tau)] = \int p(\tau; \theta) R(\tau) d\tau,$$

where $\tau = \mathbf{s}_0, \mathbf{a}_0, \dots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}$ is a state-action trajectory, μ_0 is the initial state distribution, $\mathcal{P}$ the transition dynamics, π_{θ} the parametrized policy, and $R(\tau)$ the trajectory return.

Algorithm 8 Generic step-based policy search algorithm

Init: initial lower-level policy $\pi(\mathbf{a}|\mathbf{s}; \theta)$

repeat

 /* Exploration

 Sample trajectories τ^i using the (stochastic) lower-level policy

 /* Policy Evaluation

 Collect step-based returns $\mathcal{D}_{\text{step}} = \{\mathbf{s}_t^i, \mathbf{a}_t^i, Q_t^i\}_{t=0, \dots, T-1; i=1, \dots, N}$, with $Q_t^i = \sum_{h=t}^{T-1} r_h^i$

 /* Policy Update

 Obtain a new policy $\pi(\mathbf{a}|\mathbf{s}; \theta_{k+1})$ based on $\mathcal{D}_{\text{step}}$

until Convergence of $\pi(\mathbf{a}|\mathbf{s}; \theta)$

return $\pi(\mathbf{a}|\mathbf{s}; \theta^*)$

The step-based exploration and policy evaluation are straight forward to compute, but how do we approach the policy update? We can again use a gradient-based approach and perform gradient ascent on the objective J_θ . Again, we make use of the likelihood ratio trick to compute the gradient of an expectation. we write the trajectory distribution as $p(\tau; \theta) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, \mathbf{a}_t) \pi(\mathbf{a}_t|s_t; \theta)$ due to the Markovian structure of the MDP. The likelihood ratio gradient results in,

$$\begin{aligned} \nabla_\theta J_\theta &= \int p(\tau; \theta) \nabla_\theta \log p(\tau; \theta) R(\tau) d\tau, \\ &= \int p(\tau; \theta) \nabla_\theta \left(\sum_{t=0}^{T-1} \log \pi(\mathbf{a}_t|s_t; \theta) \right) R(\tau) d\tau, \end{aligned} \quad (4)$$

where we used the fact that the transition probability does not depend on θ .

Hence, to compute the policy gradient, it suffices to be able to sample trajectories from the environment, without requiring the analytical form of the model. The return of the trajectory $R(\tau)$ can be decomposed further, by noticing that rewards that appear before the state and action at time t can be dropped, since they were not caused by future states and actions [7]. After some algebraic manipulation and noticing that rewards to come after a state-action pair are simply the Q -function, we arrive at the policy gradient theorem in Definition 16 [111].

Definition 16 (Policy Gradient Theorem) The policy gradient theorem computes the gradient of the infinite horizon objective from Definition 9 w.r.t. the policy parameters as,

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\mathbf{s} \sim \mu_\gamma^\pi} \left[\int \nabla_\theta \pi(\mathbf{a}|\mathbf{s}; \theta) Q^\pi(\mathbf{s}, \mathbf{a}) d\mathbf{a} \right], \\ &= \mathbb{E}_{\mathbf{s} \sim \mu_\gamma^\pi} \left[\int \pi(\mathbf{a}|\mathbf{s}; \theta) \nabla_\theta \log \pi(\mathbf{a}|\mathbf{s}; \theta) Q^\pi(\mathbf{s}, \mathbf{a}) d\mathbf{a} \right], \end{aligned}$$

where $\mu_\gamma^\pi \mu_\gamma^\pi(\mathbf{s}) = (1 - \gamma) d_\gamma^\pi(\mathbf{s}) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(\mathbf{s}_t = \mathbf{s} | \mathbf{s}_0, \pi)$, is the discounted state distribution, with $\mathbf{s}_0$ the initial state.

An important conclusion of this theorem is that we need only to be able to sample from the discounted state distribution, but do not need to compute its gradient. In practice, to reduce the estimator's variance, while keeping its unbiasedness, the Q function estimate is replaced by the advantage function $A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s})$.

6.5 Episodic- or Step-Based Policy Search?

There are clear benefits and drawbacks of both approaches, and both methods should be considered taking into account the robotic application.

Some characteristics of the episode-based methods are the exploration in parameter space, which allows for more sophisticated exploration strategies of deterministic policies that can be deployed in a real robot [46, 84]. It is efficient for a small amount of parameters (up to ~ 100); generalization and multi-task learning of open-loop policies such as DMPs. It is also a structure-less optimization problem and it can be solved with any black-box optimizer, such as genetic algorithms or Bayesian optimization [14].

Some characteristics of the step-based methods are exploration in action space allows for finer grade exploration at every time step. A stochastic exploration in action space is difficult to deploy in the real system and correlated exploration is needed [121]. Future rewards can be traced to a particular action, which leads to better credit assignment [108]. There is less variance in policy evaluation, and there are more data points to update the policy. It is less likely to create unstable policy updates, as it uses the structure of the MDP. The policy gradient theorem shows that we can update the policy immediately after collecting a single sample [25, 31, 58, 111].

7 Imitation Learning

In contrast to reinforcement learning, in imitation learning we optimize a policy to *imitate* a set of provided demonstrations. It is common to classify imitation learning algorithms between those that aim to learn an explicit action generator (policy) $\pi : \mathcal{S} \rightarrow \mathcal{A}$, that generates actions similar to the ones from the demonstrations and the algorithms that aim to learn an implicit state-action function (reward) $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that will learn to set a high reward to those state-action pairs that are similar to the ones in the demonstrations. We will refer to the first as imitation learning and to the second as inverse reinforcement learning algorithms.

7.1 Learning Policies by Imitation

In this subsection, we introduce a set of algorithms to learn a robot policy π given a set of state-action demonstrations $\mathcal{D}$. *Behavioral cloning* (BC) refers to the approach of directly learning a conditioned distribution from a set of demonstrations,

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{s, a \sim p(\mathcal{D})} [\log \pi(a|s)], \quad (5)$$

which is closely related to regression or density estimation. An alternative approach considers the problem of learning a policy that matches the state-action occupancy distribution of the expert demonstrations,

$$\pi^* = \min_{\pi} \psi(\rho_{\pi}, \rho_{\mathcal{D}}) - \lambda \mathbb{H}(\pi), \quad (6)$$

with ψ being a divergence distance that measures how close the stationary distribution of the expert demonstrations and the policy are. $\mathbb{H}(\pi)$ is the ‘causal entropy’ of the policy that helps during the learning process by regularizing π . We call this approach *apprenticeship learning* (AL) [35].⁵

Behavioral Cloning Revising Eq. 5, the BC approach directly models the condition action-state distribution using regression or density estimation. As a result, it can be used offline and does not require online data collection. While this method is very effective for many problems, the limitations of naive behavioral cloning are clear from the formulation. The approach needs many demonstrations to generalize effectively and it may easily suffer from *covariate shift* [85]. In regression, covariate shift refers to the performance degradation when a model is used outside its training data distribution. For BC, it manifests as a policy that cannot act or recover in unseen states. Given the problem of behavioral cloning only optimizes over the instant state-action pairs; if the robot finds itself in a state that does not belong to the demonstrations, the robot has to rely on extrapolation and interpolation to know what action to take.

A possible solution to this problem is given by DAGGER [94]. This algorithm proposes to measure how certain the robot is in a particular state. By computing the state distribution, if the robot lies in a state that was not in the demonstrations, the robot will be uncertain on which action to take. If the robot is uncertain, it will ask the teacher for additional demonstrations. These demonstrations will be added to the set of demonstrations and the robot will re-learn the policy. DAGGER proposes to solve the imitation learning problem by actively increasing the demonstrations dataset.

Another solution is proposed by DART [49]. This algorithm, rather than querying additional demonstrations from the teacher, it will inject additional noise in the action while the teacher provides demonstrations. This noise injection allows to generate more diverse demonstrations and encourages robust policies. The recorded demonstrations will provide information on how to recover from states close to the optimal trajectories.

Apprenticeship Learning In contrast with behavioral cloning, apprenticeship learning proposes to learn a policy that matches the stationary state-action pairs rather than the conditioned distribution. The obtained policy is going to be more

⁵ Note, apprenticeship learning is sometimes used interchangeably with imitation learning in the older literature.



Fig. 7 Performance of a learned policy for helicopter acrobacy. The policy was trained by apprenticeship learning to mimic a set of expert demonstrations doing acrobatics [17]. Figure reproduced with permission from Pieter Abbeel

robust than the policies learned by behavioral cloning. On the other hand, apprenticeship learning methods require a dynamic model from which new samples can be generated, or online samples.

Different apprenticeship learning algorithms have proposed different divergence distances ψ , to measure the distance between the occupancy distributions $\rho(s, \mathbf{a})$. The classical apprenticeship learning [1] proposes to model the distance metric ψ by contrastive divergence,

$$\psi(\rho_\pi, \rho_D) = \max_c \mathbb{E}_{s, \mathbf{a} \sim \rho_\pi} [c(s, \mathbf{a})] - \mathbb{E}_{s, \mathbf{a} \sim \rho_D} [c(s, \mathbf{a})],$$

while a more modern approach, generative adversarial imitation learning (GAIL) [35], gets inspiration from generative adversarial networks (GAN) and substitute the contrastive divergence by the Jensen-Shannon divergence,

$$\psi(\rho_\pi, \rho_D) = \max_c \mathbb{E}_{s, \mathbf{a} \sim \rho_\pi} [\log(c(s, \mathbf{a}))] + \mathbb{E}_{s, \mathbf{a} \sim \rho_D} [\log(1 - c(s, \mathbf{a}))].$$

In practice, apprenticeship learning algorithms combine a reinforcement learning phase with a cost learning phase. Given an initial policy π a set of trajectory demonstrations are generated by evaluating the policy on the system. The cost function $c(s, \mathbf{a})$ is trained to set high cost to the samples generated by the policy π and low cost to the samples generated by the demonstrations. Then, the policy is trained by reinforcement learning to maximize over the cost function $c(s, \mathbf{a})$. By iteratively training both the cost and the policy, the policy will learn to generate trajectories that match the stationary distribution of the expert demonstrations. One of the most successful use cases of apprenticeship learning is for helicopter acrobacy learning (Fig. 7). A policy trained on human expert demonstrations was able to perfectly mimic the expert on doing acrobacies in the sky.

7.2 Learning Rewards from Data: Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) deals with the problem of learning the reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that the expert demonstrations are trying to maximize [74].

While most of the apprenticeship learning algorithms learn a proxy reward function, in this section we will focus in the learning of the reward itself.

The problem of inverse reinforcement learning can be framed as finding the true reward $r^*(s_t, \mathbf{a}_t)$ where,

$$\mathbb{E}_{\mathbf{a}_t \sim \pi_{\mathcal{D}}(s_t), s_{t+1} \sim p(\cdot | s_t, \mathbf{a}_t), s_0 \sim p(s_0)} \left[\sum_t \gamma^t r^*(s_t, \mathbf{a}_t) \right] \geq \mathbb{E}_{\mathbf{a}_t \sim \pi(s_t), s_{t+1} \sim p(\cdot | s_t, \mathbf{a}_t), s_0 \sim p(s_0)} \left[\sum_t \gamma^t r^*(s_t, \mathbf{a}_t) \right] \quad \forall \pi \in \Pi,$$

such that the reward r^* is maximized for the behavioral cloned policy $\pi_{\mathcal{D}}$ compared to the other policies in a set Π . The problem definition is challenging given it is an ill-posed problem. Setting $r(s_t, \mathbf{a}_t) = 0$ satisfies the inequality, which is not a meaningful reward function. Moreover, we have only a finite set of expert demonstrations and not the whole expert state-action distribution. We assume the demonstrations come from an optimal expert policy rather than a suboptimal one, and finally we assume we can consider all possible policies through Π .

In the following, we introduce two methods to tackle the problem: Maximum margin [91] and maximum entropy IRL [130, 131]. In both cases, we will consider a reward function modelled as a linearly-weighted sum of features $r(s) = \mathbf{w}^\top \phi(s)$. Given the features are linear,

$$\begin{aligned} \hat{R}^* &= \mathbb{E}_{\mathbf{a}_t \sim \pi_{\mathcal{D}}(s_t), s_{t+1} \sim p(\cdot | s_t, \mathbf{a}_t), s_0 \sim p(s_0)} \left[\sum_t \gamma^t r^*(s_t) \right], \\ &= \mathbb{E} \left[\sum_t \gamma^t \mathbf{w}^\top \phi(s_t) \right] = \mathbf{w}^\top \mathbb{E} \left[\sum_t \gamma^t \phi(s_t) \right] = \mathbf{w}^\top \Phi(\pi_{\mathcal{D}}), \end{aligned}$$

where $\Phi(\pi_{\mathcal{D}})$ represents the average sum of discounted features induced by $\pi_{\mathcal{D}}$.

Maximum Margin Inverse Reinforcement Learning Ratliff et al. [91] proposes to solve the IRL problem by optimizing

$$\min_{\mathbf{w}} \mathbf{w}^\top \mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^\top \Phi(\pi_{\mathcal{D}}) \geq \mathbf{w}^\top \Phi(\pi) + m(\pi_{\mathcal{D}}, \pi), \quad \forall \pi \in \Pi.$$

The optimization problem will search for the smallest weights that satisfy that the reward in the demonstrations policy $\pi_{\mathcal{D}}$ is bigger than the reward for the rest of the policies up to a margin term $m(\pi_{\mathcal{D}}, \pi)$. The chosen margin should increase for policies that are very different from $\pi_{\mathcal{D}}$. This approach is closely related to support vector machines, which are maximum margin models for classification. The connection here is that our reward function is acting as a classifier to differentiate $\pi_{\mathcal{D}}$ from the other policies in Π .

Maximum Entropy Inverse Reinforcement Learning Ziebart et al. [130] frames the problem as a maximum entropy problem. Given τ is a state-action trajectory and given an expert policy $\pi_{\mathcal{D}}$, we aim to find,

$$\arg \max_p \mathbb{H}(p) = - \int_{\tau} p(\tau) \log p(\tau) d\tau \quad \text{s.t.} \quad \int_{\tau} p(\tau) \Phi(\tau) d\tau = \Phi(\pi_{\mathcal{D}}).$$

with p being a distribution over the trajectories. The optimal solution of the maximum entropy problem is $p(\tau) \propto \exp\{\mathbf{w}^{*\top} \Phi(\tau)\}$, that is the maximally uncertain distribution subject to it agrees with all known constraints (expert policy). The optimal return for a trajectory is $R(\tau) = \mathbf{w}^{*\top} \Phi(\tau)$.

8 Outlook

This chapter has introduced Robot Learning in the context of optimizing controllers from sampled data. By building on ideas from optimal control, we have motivated model-based and model-free methods for policy optimization by incorporating tools from machine learning, including function approximation and gradient estimation. Moreover, we have extended these ideas to imitation learning, where policies and rewards are estimated from expert human demonstration.

Due to the breadth and depth of Robot Learning, many topics and settings were not discussed in this chapter. This includes discussing the learning challenges on specific robot platforms, such as autonomous vehicles and legged robots. We also omitted specific tasks that are important robotic problems, such as navigation, robotic grasping, state estimation and human-robot interaction. As an outlook, we wish to briefly discuss some key topics and open problems in the field of Robot Learning.

Deep Reinforcement Learning Considers the combination of reinforcement learning algorithms with deep neural networks [32, 68, 102]. This approach has demonstrated impressive performance on complex tasks and has been widely adopted by the research community. However, these methods have seen less adoption in practical applications and products, due to their performance variance and lack of performance guarantees. Due to their sample efficiency and lack of safety guarantees, their use is mainly limited to learning in simulators rather than physical systems.

Inductive Biases for Deep Robot Learning One means of improving sample efficiency and safety is to incorporate useful structures into machine learning models that enforce the desired characteristics [6]. The flexibility of automatic differentiation means that a broad range of structure can be incorporated into deep learning models, such as computer vision models [33], state estimation algorithms [30, 43], non-holonomic constraints [63], stability guarantees [119] and energy conservation [61]. Striking the right balance with inductive biases, providing enough

useful structure without underfitting to the data, can provide Robot Learning methods with a greater degree of interpretability, safety and sample efficiency for deployment on real-world systems.

Parallelized and Differential Simulators Extending the notion of inductive biases eventually results in a feature-complete differentiable simulator, that model rigid-body physics, contact and rendering [19, 42, 116, 129]. This enables simulators that themselves learn from data. Moreover, gradients from the simulator can be used to accelerate learning in a model-based fashion. Finally, implementing these simulators on hardware accelerators such as GPUs provides dramatic speed-ups for sample-based methods such as deep reinforcement learning, which in turn accelerates Robot Learning research and deployment.

Transferring from Simulation to Reality The widespread adoption of simulators in Robot Learning methods introduces the issue of the ‘reality gap’, the difference between the real-world and the simulated replication. Methods for ‘sim2real’ transfer consider means of designing or perturbing the simulator during training to minimize the adverse effects of the reality gap [71, 88, 115]. Effective sim2real methods, combined with advances in simulation, would facilitate more real-world deployment of many powerful RL algorithms.

Task and Motion Planning (TAMP) considers higher-level robot control that requires many subsystems, such as perception, motor control and navigation [15, 128]. While this chapter has focused on ML for motor control, ML methods can also be combined with classical approaches for TAMP to improve performance in a data-driven fashion [37, 54]. For example, it enables joint optimization of each system, allowing representations from perception to be optimized to aid planning, and navigation to be fine-tuned for motion control. Advances in TAMP are crucial if robots are to be able to carry out complex tasks in everyday, unstructured environments.

Safe Learning for Robotics The restriction of many Robot Learning methods to simulators is in part due to safety concerns when learning on physical systems. While ‘safe’ is an ambiguous term, it generally refers to algorithms that can learn with certain guarantees that prevent catastrophic failure of the robot [9, 34]. Safe methods can be developed by considering inductive biases, such as motion constraints of the robot, as seen in Sect. 5 with motion primitives. Safety can also be incorporated by considering the statistical uncertainty of the learned models.

Multimodal Robot Learning A quality of deep learning models are their ability to easily combine different ‘data modalities’, such as vision, sound, speech and other telemetry [29]. Advances in vision and natural language processing can enable robots that can learn to integrate diverse sensor inputs [53], and also interpret and communicate with humans via speech or gestures for richer human-robot interaction [64, 65, 118].

In conclusion, Robot Learning represents an exciting paradigm for robotics, with many opportunities for robots to learn, adapt and improve the skills they need to carry out complex tasks and integrate with society.

9 Cross References

This chapter contains cross-references to content published in other chapters belonging to the four “Robotic Goes MOOC” books: KNO2, KNO3, KNO7. The full list of chapter abbreviations is available in the Preface.

References

1. P. Abbeel, A.Y. Ng, Apprenticeship learning via inverse reinforcement learning, in *Proceedings of the Twenty-First International Conference on Machine Learning* (2004), p. 1
2. C.C. Aggarwal, et al., Neural networks and deep learning. Springer **10**, 978–973 (2018)
3. K.J. Åström, P. Eykhoff, System identification—a survey. *Automatica* **7**, 123–162 (1971)
4. K.J. Åström, B. Wittenmark, *Adaptive Control*, 2nd edn. (Addison-Wesley, Boston, 1994)
5. D. Barber, *Bayesian Reasoning and Machine Learning*. (Cambridge University Press, Cambridge, 2011)
6. J. Baxter, A model of inductive bias learning. *J. Artif. Intell. Res.* **12**, 149–198 (2000)
7. J. Baxter, P.L. Bartlett, Infinite-horizon policy-gradient estimation. *J. Artif. Int. Res.* **15**(1), 319–350 (2001)
8. R. Bellman, *Dynamic Programming* (Princeton University Press, Princeton, 1957)
9. F. Berkenkamp, M. Turchetta, A.P. Schoellig, A. Krause, Safe model-based reinforcement learning with stability guarantees, in *Advances in Neural Information Processing Systems* (2017), pp. 908–919
10. D. Bertsekas, *Dynamic Programming and Optimal Control* (Athena Scientific, Nashua, 2012)
11. C.M. Bishop, *Pattern Recognition and Machine Learning* (Springer, New York, 2006)
12. S. Boyd, L. Vandenberghe, *Convex Optimization* (Cambridge University Press, Cambridge, 2004)
13. A.E. Bryson, *Applied Optimal Control: Optimization, Estimation and Control* (Routledge, London, 2018)
14. R. Calandra, A. Seyfarth, J. Peters, M. Deisenroth, Bayesian optimization for learning gaits under uncertainty. *Ann. Math. Artif. Intell.* **76**, 5–23 (2015)
15. S. Cambon, R. Alami, F. Gravot, A hybrid approach to intricate motion, manipulation and task planning. *Int. J. Robot. Res.* **28**(1), 104–126 (2009)
16. K. Chua, R. Calandra, R. McAllister, S. Levine, Deep reinforcement learning in a handful of trials using probabilistic dynamics models, in *Advances in Neural Information Processing Systems* (2018)
17. A. Coates, P. Abbeel, A.Y. Ng, *Autonomous Helicopter Flight Using Reinforcement Learning* (Springer, Boston, 2017), pp. 75–85
18. G. Cybenko, Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **2**(4), 303–314 (1989)
19. J. Degraeve, M. Hermans, J. Dambre, et al., A differentiable physics engine for deep learning in robotics. *Front. Neurobot.* **13**, 6 (2019)

20. M. Deisenroth, C.E. Rasmussen, Pilco: a model-based and data-efficient approach to policy search, in *International Conference on Machine Learning* (2011)
21. M.P. Deisenroth, G. Neumann, J. Peters, et al., *A Survey on Policy Search for Robotics*. Foundations and Trends® in Robotics (Now Publishers, Norwell, 2013)
22. F. Dellaert, Factor graphs: exploiting structure in robotics. *Annual Rev. Control Robot. Autonom. Syst.* **4**(1), 141–166 (2021)
23. A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc. B (Methodological)* **39**, 1–22 (1977)
24. D. Ernst, P. Geurts, L. Wehenkel, Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.* **6**, 503–556 (2005)
25. S. Fujimoto, H. van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 (2018), pp. 1587–1596
26. N. Funk, C. Schaff, R. Madan, T. Yoneda, J.U. De Jesus, J. Watson, E.K. Gordon, F. Widmaier, S. Bauer, S.S. Srinivasa, T. Bhattacharjee, Walter, M.R., J. Peters, Benchmarking structured policies and policy optimization for real-world dexterous object manipulation. *IEEE Robot. Autom. Lett.* **7**(1), 478–485 (2022)
27. A.R. Geist, S. Trimpe, Structured learning of rigid-body dynamics: a survey and unified view from a robotics perspective. *GAMM-Mitteilungen* **44**(2), e202100009 (2021)
28. Z. Ghahramani, S.T. Roweis, Learning nonlinear dynamical systems using an em algorithm, in *Advances in Neural Information Processing Systems* (1999), pp. 431–437
29. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016). <http://www.deeplearningbook.org>
30. T. Haarnoja, A. Ajay, S. Levine, P. Abbeel, Backprop KF: learning discriminative deterministic state estimators, in *Advances in Neural Information Processing Systems* (2016)
31. T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor, in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 80 (2018), pp. 1856–1865
32. T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor, in *International Conference on Machine Learning*. *Proceedings of Machine Learning Research* (2018), pp. 1861–1870
33. A. Handa, M. Bloesch, V. Pătrăucean, S. Stent, J. McCormac, A. Davison, GVNN: neural network library for geometric computer vision, in *European Conference on Computer Vision (ECCV)* (2016)
34. L. Hewing, K.P. Wabersich, M. Menner, M.N. Zeilinger, Learning-based model predictive control: toward safe learning in control. *Ann. Rev. Control Robot. Autonom. Syst.* **3**(1), 269–296 (2020)
35. J. Ho, S. Ermon, Generative adversarial imitation learning. *Adv. Neural Inf. Process. Syst.* **29**, 4565–4573 (2016)
36. T. Hofmann, B. Schölkopf, A.J. Smola, Kernel methods in machine learning. *Ann. Stat.* **36** (2008)
37. C. Hogg, U. Kuter, H. Munoz-Avila, Learning methods to generate good plans: integrating HTN learning and reinforcement learning, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24 (2010), pp. 1530–1535
38. K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators. *Neural Netw.* **2**(5), 359–366 (1989)
39. A.J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, S. Schaal, Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Comput.* **25**(2), 328–373 (2013)
40. P. Izmailov, S. Vikram, M.D. Hoffman, A.G.G. Wilson, What are bayesian neural network posteriors really like? in *Proceedings of the 38th International Conference on Machine Learning* (18–24 Jul 2021), ed. by M. Meila, T. Zhang, *Proceedings of Machine Learning Research*, vol. 139, pp. 4629–4640 (2021)
41. D.H. Jacobson, D.Q. Mayne, *Differential Dynamic Programming* (North-Holland, Amsterdam, 1970)

42. K.M. Jatavallabhula, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Levesque, K. Xie, K. Erleben, L. Paull, F. Shkurti, D. Nowrouzezahrai, S. Fidler, gradSim: differentiable simulation for system identification and visuomotor control, in *International Conference on Learning Representations (ICLR)* (2021)
43. R. Jonschkowski, D. Rastogi, O. Brock, Differentiable particle filters: end-to-end learning with algorithmic priors, in *Proceedings of Robotics: Science and Systems, Pittsburgh* (2018)
44. S.M. Khansari-Zadeh, A. Billard, Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Trans. Robot.* **27**(5), 943–957 (2011)
45. S.M. Khansari-Zadeh, A. Billard, Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robot. Autonom. Syst.* **62**(6), 752–765 (2014)
46. J. Kober, J. Peters, Learning motor primitives for robotics, in *IEEE International Conference on Robotics and Automation (ICRA)* (2009), pp. 2112–2118
47. J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: a survey. *Int. J. Robot. Res.* (2013)
48. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **25**, 1097–1105 (2012)
49. M. Laskey, J. Lee, R. Fox, A. Dragan, K. Goldberg, Dart: noise injection for robust imitation learning, in *Conference on Robot Learning*. *Proceedings of Machine Learning Research* (2017), pp. 143–156
50. M. Lázaro-Gredilla, A.R. Figueiras-Vidal, Marginalized neural network mixtures for large-scale regression. *Trans. Neural Netw.* **21**(8), 1345–1351 (2010)
51. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. *Nature* **521**, 7553 (2015). 436–444
52. M.A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, J. Bohg, Making sense of vision and touch: self-supervised learning of multimodal representations for contact-rich tasks, in *2019 International Conference on Robotics and Automation (ICRA)* (IEEE, Piscataway, 2019), pp. 8943–8950
53. M.A. Lee, Y. Zhu, P. Zachares, M. Tan, K. Srinivasan, S. Savarese, L. Fei-Fei, A. Garg, J. Bohg, Making sense of vision and touch: learning multimodal representations for contact-rich tasks. *IEEE Trans. Robot.* **36**(3), 582–596 (2020)
54. M. Leonetti, L. Iocchi, P. Stone, A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artif. Intell.* **241**, 103–130 (2016)
55. S. Levine, P. Abbeel, Learning neural network policies with guided policy search under unknown dynamics, in *Advances in Neural Information Processing Systems* (2014)
56. W. Li, E. Todorov, Iterative linear quadratic regulator design for nonlinear biological movement systems, in *1st International Conference on Informatics in Control, Automation and Robotics* (2004)
57. Q. Li, O. Kroemer, Z. Su, F.F. Veiga, M. Kaboli, H.J. Ritter, A review of tactile information: perception and action through touch. *IEEE Trans. Robot.* **36**(6), 1619–1634 (2020)
58. T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in *International Conference on Learning Representations, (ICLR)* (2016)
59. R. Lioutikov, O. Kroemer, J. Peters, G. Maeda, Learning manipulation by sequencing motor primitives with a two-armed robot, in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems*. *Advances in Intelligent Systems and Computing*, vol. 302 (Springer, Berlin, 2014), pp. 1601–1611
60. L. Ljung, *System Identification - Theory for the User* (Prentice Hall, Upper Saddle River, 1999)
61. M. Lutter, C. Ritter, J. Peters, Deep lagrangian networks: using physics as model prior for deep learning, in *International Conference on Learning Representations (ICLR)* (2019)
62. M. Lutter, J. Silberbauer, J. Watson, J. Peters, A differentiable newton euler algorithm for multi-body model learning, in *Robotics: Science and Systems Conference (RSS), Workshop on Structured Approaches to Robot Learning for Improved Generalization* (2020)

63. M. Lutter, J. Silberbauer, J. Watson, J. Peters, Differentiable physics models for real-world offline model-based reinforcement learning, in *IEEE International Conference on Robotics and Automation (ICRA)* (2021)
64. C. Lynch, P. Sermanet, Language conditioned imitation learning over unstructured data, in *Robotics: Science and Systems* (2021)
65. C. Matuszek, L. Bo, L. Zettlemoyer, D. Fox, Learning from unscripted deictic gesture and language for human-robot interactions, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1 (2014)
66. A. Mesbah, Stochastic model predictive control: an overview and perspectives for future research. *IEEE Control Syst. Mag.* **36**(6), 30–44 (2016)
67. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M.A. Riedmiller, Playing atari with deep reinforcement learning (2013). CoRR abs/1312.5602
68. V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015)
69. S. Mohamed, M. Rosca, M. Figurnov, A. Mnih, Monte Carlo gradient estimation in machine learning. *J. Mach. Learn. Res.* **21**, 1–62 (2020)
70. K. Muelling, J. Kober, O. Kroemer, J. Peters, Learning to select and generalize striking movements in robot table tennis, in *AAAI Fall Symposium on Robots that Learn Interactively from Human Teachers* (2012), pp. 263–279
71. F. Muratore, M. Gienger, J. Peters, Assessing transferability from simulation to reality for reinforcement learning. *IEEE Trans. Pattern Anal. Mach. Intell.* (2019)
72. R.M. Neal, Bayesian learning for neural networks. PhD Thesis, University of Toronto (1995)
73. K. Neumann, J.J. Steil, Learning robot motions with stable dynamical systems under diffeomorphic transformations. *Robot. Autonom. Syst.* **70**, 1–15 (2015)
74. A.Y. Ng, S.J. Russell, et al., Algorithms for inverse reinforcement learning, in *International Conference on Machine Learning*, vol. 1 (2000), p. 2
75. D. Nguyen-Tuong, J. Peters, Using model knowledge for learning inverse dynamics, in *International Conference on Robotics and Automation* (2010)
76. D. Nguyen-Tuong, J. Peters, Model learning for robot control: a survey. *Cognit. Process.* **12**(4), 319–340 (2011)
77. S.W. Ober, C.E. Rasmussen, Benchmarking the neural linear model for regression, in *Symposium on Advances in Approximate Bayesian Inference* (2019)
78. A. Paraschos, C. Daniel, J. Peters, G. Neumann, et al., Probabilistic movement primitives. *Advances in Neural Information Processing Systems* (2013)
79. J. Peters, S. Schaal, Policy gradient methods for robotics, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2006), pp. 2219–2225
80. J. Peters, S. Schaal, Reinforcement learning of motor skills with policy gradients. *Neural Netw.* **21**(4), 682–697 (2008)
81. J. Peters, K. Mülling, Y. Altün, Relative entropy policy search, in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (2010), pp. 1607–1612
82. J. Peters, R. Tedrake, N. Roy, J. Morimoto, *Robot Learning* (Springer, Boston, 2010), pp. 865–869
83. K. Ploeger, M. Lutter, J. Peters, High acceleration reinforcement learning for real-world juggling with binary rewards, in *Conference on Robot Learning (CoRL)* (2020)
84. K. Ploeger, M. Lutter, J. Peters, High acceleration reinforcement learning for real-world juggling with binary rewards, in *Conference on Robot Learning (CoRL)* (2020)
85. D.A. Pomerleau, Alvin: An autonomous land vehicle in a neural network. Tech. Rep., Carnegie-Mellon University Pittsburgh, Artificial Intelligence and Psychology Department (1989)
86. M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (Wiley, Hoboken, 2014)
87. A. Rahimi, B. Recht, Random features for large-scale kernel machines, in *Advances in Neural Information Processing Systems*, ed. by J. Platt, D. Koller, Y. Singer, S. Roweis, vol. 20 (Curran Associates, New York, 2008)

88. F. Ramos, R. Possas, D. Fox, Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators, in *Robotics: Science and Systems* (MIT Press, Cambridge, 2019)
89. M.A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, N. Ratliff, Euclideanizing flows: diffeomorphic reduction for learning stable dynamical systems, in *Learning for Dynamics and Control*. Proceedings of Machine Learning Research (2020), pp. 630–639
90. C.E. Rasmussen, C.K.I. Williams, *Gaussian Processes for Machine Learning* (MIT Press, Cambridge, 2005)
91. N.D. Ratliff, J.A. Bagnell, M.A. Zinkevich, Maximum margin planning, in *Proceedings of the 23rd International Conference on Machine Learning* (2006), pp. 729–736
92. Real robot challenge, <https://real-robot-challenge.com/>
93. M. Riedmiller, Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method, in *Proceedings of the 16th European Conference on Machine Learning, ECML'05* (Springer, Berlin, 2005), pp. 317–328
94. S. Ross, G. Gordon, D. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings (2011), pp. 627–635
95. R.Y. Rubinstein, D.P. Kroese, *The Cross Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation* (Information Science and Statistics) (Springer, New York, 2004)
96. G.A. Rummery, M. Niranjan, On-line Q-learning using connectionist systems. Tech. Rep. TR 166, Cambridge University Engineering Department, Cambridge (1994)
97. S. Särkkä, *Bayesian Filtering and Smoothing* (Cambridge University Press, Cambridge, 2013)
98. S. Schaal, Dynamic movement primitives-a framework for motor control in humans and humanoid robotics, in *Adaptive Motion of Animals and Machines* (Springer, Berlin, 2006), pp. 261–280
99. J. Schmidhuber, Deep learning in neural networks: an overview. *Neural Netw.* **61**, 85–117 (2015)
100. J.G. Schneider, Exploiting model uncertainty estimates for safe dynamic control learning, in *Advances in Neural Information Processing Systems* (1997)
101. T.B. Schön, F. Lindsten, J. Dahlin, J. Wågberg, C.A. Naesseth, A. Svensson, L. Dai, Sequential monte carlo methods for system identification. *IFAC-PapersOnLine* **48**(28), 775–786 (2015)
102. J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in *International Conference on Machine Learning*. Proceedings of Machine Learning Research (2015), pp. 1889–1897
103. R.H. Shumway, D.S. Stoffer, An approach to time series smoothing and forecasting using the EM algorithm. *J. Time Ser. Anal.* **3**(4), 253–264 (1982)
104. A. Sideris, J.E. Bobrow, An efficient sequential linear quadratic algorithm for solving non-linear optimal control problems, in *Proceedings of the 2005, American Control Conference* (IEEE, Piscataway, 2005), pp. 2275–2280
105. D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016)
106. G. Sutanto, A. Wang, Y. Lin, M. Mukadam, G. Sukhatme, A. Rai, F. Meier, Encoding physical constraints in differentiable Newton-Euler Algorithm, in *Learning for Dynamics Control (LADC)* (2020)
107. I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in *Advances in Neural Information Processing Systems* (2014), pp. 3104–3112
108. R.S. Sutton, Temporal credit assignment in reinforcement learning. Ph.D. thesis, University of Massachusetts Amherst. AAI8410337 (1984)
109. R.S. Sutton, Learning to predict by the methods of temporal differences. *Mach. Learn.* **3**(1), 9–44 (1988)

110. R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, 2018)
111. R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in *Advances in Neural Information Processing Systems (NIPS)* (1999), pp. 1057–1063
112. A. Tangirala, *Principles of System Identification: Theory and Practice* (CRC Press, Boca Raton, 2018)
113. Y. Tassa, Theory and implementation of biomimetic motor controllers. Ph.D. Thesis, Hebrew University of Jerusalem (2011)
114. M. Titsias, Variational learning of inducing variables in sparse Gaussian processes, in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics* (16–18 Apr 2009), vol. 5. Proceedings of Machine Learning Research (2005), pp. 567–574
115. J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in *International Conference on Intelligent Robots and Systems (IROS)* (2017)
116. M.A. Toussaint, K.R. Allen, K.A. Smith, J.B. Tenenbaum, Differentiable physics and stable modes for tool-use and manipulation planning, in *Robotics: Science and Systems* (2018)
117. S. Traversaro, S. Brossette, A. Escande, F. Nori, Identification of fully physical consistent inertial parameters using optimization on manifolds, in *International Conference on Intelligent Robots and Systems (IROS)* (2016)
118. M. Tucker, D. Aksaray, R. Paul, G.J. Stein, N. Roy, Learning unknown groundings for natural language interaction with mobile robots, in *Robotics Research* (Springer, Berlin, 2020), pp. 317–333
119. J. Urain, M. Ginesi, D. Tateo, J. Peters, Imitationflow: learning deep stable stochastic dynamic systems by normalizing flows, in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, Piscataway, 2020), pp. 5231–5237
120. H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, J. Modayil, Deep reinforcement learning and the deadly triad. CoRR abs/1812.02648 (2018)
121. H. van Hoof, D. Tanneberg, J. Peters, Generalized exploration in policy search, in *Machine Learning* (2017), pp. 1705–1724
122. C.J.C.H. Watkins, P. Dayan, Q-learning. *Mach. Learn.* **8**(3), 279–292 (1992)
123. J. Watson, J.A. Lin, P. Klink, J. Pajarinen, J. Peters, Latent derivative Bayesian last layer networks, in *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics* (2021)
124. J. Watson, J.A. Lin, P. Klink, J. Peters, Neural linear models with functional Gaussian process priors, in *Third Symposium on Advances in Approximate Bayesian Inference* (2021)
125. D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, J. Schmidhuber, Natural evolution strategies. *J. Mach. Learn. Res.* **15**, 949–980 (2014)
126. C.K. Williams, Computing with infinite networks, in *Advances in Neural Information Processing Systems* (1997), pp. 295–301
127. G. Williams, N. Wagener, B. Goldfain, P. Drews, J.M. Rehg, B. Boots, E.A. Theodorou, Information theoretic MPC for model-based reinforcement learning, in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, Piscataway, 2017), pp. 1714–1721
128. J. Wolfe, B. Marthi, S. Russell, Combined task and motion planning for mobile manipulation, in *International Conference on Automated Planning and Scheduling (ICAPS)* (2010)
129. S. Zhao, W. Jakob, T.M. Li, Physics-based differentiable rendering: from theory to implementation, in *ACM SIGGRAPH 2020 Courses (SIGGRAPH'20)*, New York. Association for Computing Machinery, New York
130. B.D. Ziebart, A.L. Maas, J.A. Bagnell, A.K. Dey, et al., Maximum entropy inverse reinforcement learning, in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence AAAI, Chicago*, vol. 8 (2008), pp. 1433–1438
131. B.D. Ziebart, J.A. Bagnell, A.K. Dey, Modeling interaction via the principle of maximum causal entropy, in *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML* (2010)

Knowledge Representation and Reasoning



Michael Beetz and Daniel Nyga

Abstract In this chapter we investigate how to use knowledge representation and reasoning methods developed in artificial intelligence in order to implement robot control systems that can autonomously accomplish complex and underdetermined tasks, automatically adapt to task contexts, learn from experience, and forestall execution failures through action planning. We will use fetch and place tasks to be performed by a household assistant robot as our running example and show how these methods enable robots to make informed decisions about where to look for and find objects, how to pick them up, hold them, and where to place them.¹ The chapter will introduce fact and rule-based knowledge representation as it is provided by logic and apply it to reason about robots, their components and capabilities, objects and environments, and actions and their effects. We will conclude by discussing how knowledge representation can be employed to equip robots with introspective reasoning capabilities, which allow the robot to answer queries regarding to what it is doing, why, how, what is predicted to happen, etc.

1 Introduction

In this chapter we will introduce and discuss methods to program autonomous robots to accomplish vague human-scale manipulation tasks such as setting the table or making popcorn. One of the key challenges in programming robots for such tasks is that robots have to be programmed in ways so as to successfully work in open worlds, i.e. worlds that the programmer does not fully know at the time of writing the programs. This requires the robot to have decisional autonomy: when

¹ We call the actions “fetch and place” rather than the more common “pick and place” to emphasize that our actions include the subtask of finding the intended or adequate objects.

M. Beetz (✉) · D. Nyga
Institute of Artificial Intelligence, University of Bremen, Bremen, Germany
e-mail: beetz@cs.uni-bremen.de; nyga@cs.uni-bremen.de

setting the table the robot has to decide where to look for plates and glasses, which objects to put on the table, how to grasp them, and where to place them. Many of these decisions depend on the specific objects, their shapes, weight, and other properties, the structure of the environments, and the capabilities of the executing robots. We will now study which knowledge robots have to be equipped with to make such decisions, how this knowledge can be encoded such that it is machine understandable, and how robots can reason from such knowledge in order to decide on the appropriate course of action.

To make our discussions more tangible we will take a home assistance robot as our running example, which is supposed to prepare popcorn in a kitchen environment. In order to translate an instruction, such as “make popcorn”, into action a robot has to infer missing information including missing action steps and their appropriate order. In order to get a better intuition about the complexity and details of such manipulation tasks, consider the snapshots of the cooking activity, which are depicted in Fig. 1. A video featuring a robot performing the complete popcorn preparation task can be found at <https://www.youtube.com/watch?v=cTCJSNjTdo0>.

Competently making popcorn requires robots to solve a number of challenging reasoning tasks. The robot has to infer implicit goals and unmentioned subactions, including that in order to cook popcorn the unpopped corn has to be put into a

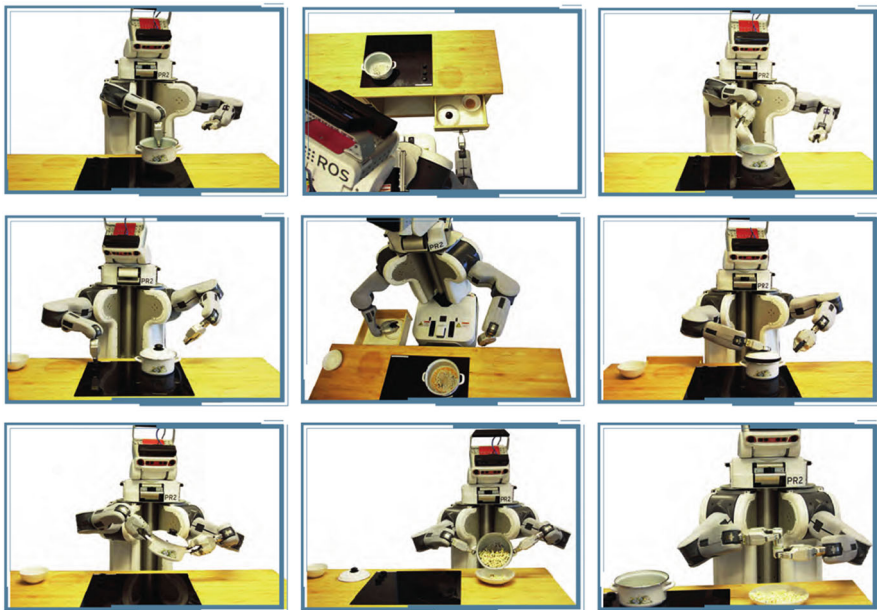


Fig. 1 Action steps for popcorn making: (1) putting the cooking pot on the stove, (2) opening the drawer, (3) pouring the corn into the pot, (4) switching on the drawer, (5) grasping the lid, (6) putting the lid on the pot, (7) distributing the corn evenly in pot, (8) pouring the popcorn onto the plate, (9) salting the popcorn

cooking pot, the pot be placed on a stove top, the knob for the respective stove be turned in order to switch the stove on. The robot has to wait for the popcorn to be done, the lid has to be removed from the pot in order to pour it onto a plate.

Other kinds of needed knowledge might be that a pot with a lid on top is heavier than without it and that a filled pot should be held upright. When pouring the pot should not be held too high above the plate to avoid spilling. Also, a hot pot should be only grasped by its handles. For grasping objects the robot should choose the grasp type and position that is most reliable, efficient, and controllable. Additional task knowledge includes knowledge about where the popcorn is stored, that the salt grinder has to be turned upside down, and the drawers open outwards.

In this chapter we proceed as follows. We will first introduce the rational agent model of robots that will provide us with the conceptual vocabulary needed to discuss how robots perceive and act in their environments in order to accomplish their goals. We will then introduce state transition systems as our first model of representing actions and how their actions change the state of their environments. Then we will explain how declarative, logic-based languages can be used to represent these state transition models as symbolic knowledge bases and how we can use logical reasoning to derive the course of action to achieve a given goal. We will further discuss the limitations of the transition system model with atomic actions and give an outlook how knowledge representation and reasoning can be extended to mitigate these limitations.

2 The Rational Robot Agent Model

We begin our discourse about robot agency with introducing a framework of concepts that enables us to formalize the interaction of robots and the environment they act in, how goals and objectives of the robots can be stated and the goal achievement through robot actions be measured, and how robots should select their course of action in order to maximize the impact of its actions. To this end, we introduce the model of a *rational robot agent*, which is inspired by the original definition of a rational agent by Russell and Norvig [1]. For our purpose we specialize the conceptual framework of rational agents to physical robots and their control programs as illustrated in Fig. 2.

In the rational robot framework a *robot agent* is conceptualized as an entity that acts in an *environment* in order to achieve its goals. The agent perceives the environment through its *sensors* and changes the state of the environment through its physical actions. The agent is controlled through a function that maps percepts from its sensors and prior knowledge into an action that the robot executes. We further conceptualize the processes with which robots decide on their course of action and how the actions change the environment as an iterative interaction between the robot and the environment it is operating in. In each iteration the robot agent

1. perceives the state of the environment,

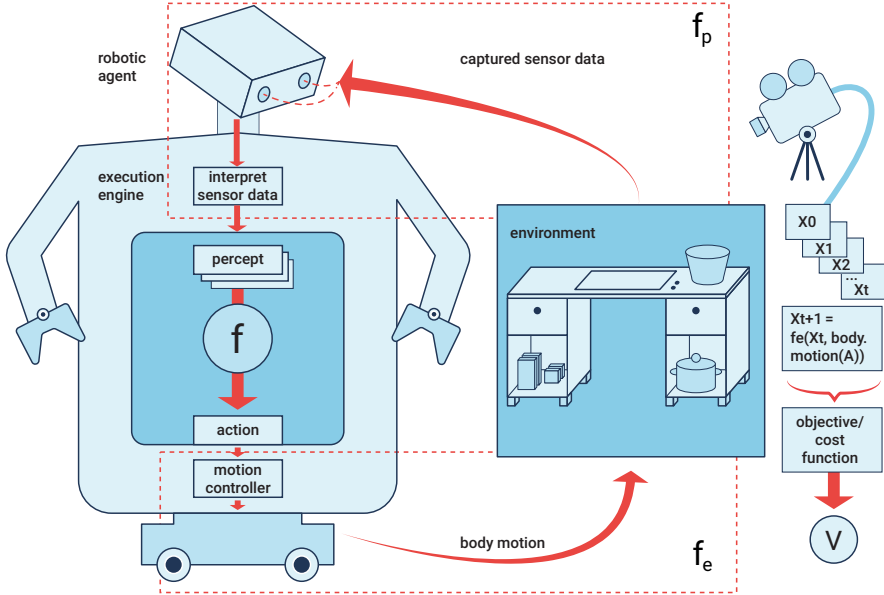


Fig. 2 Conceptualization of a robot agent and the interaction with its environment: the state of the environment is perceived by the agent’s sensors. The interpreted sensor readings are passed as observations to the agent function f , which derives from goals. The agent function selects an action, which in turn is carried out by the agent’s motion controllers. The motion of the robot agent’s body causes changes to the environment. The evolution of the environment states X^T serves as a basis for assessing the agent’s performance V

2. decides on the next action, and
3. executes the action in order to change its environment.

The repeated execution of the steps (1) to (3) forms a so-called *perception-action loop*. To formalize this loop, we define the set of symbols representing the robot’s percepts as $\mathcal{O}$ (observations), the set of actions the robot agent can execute as $\mathcal{A}$, and the set of environment states as $\mathcal{X}$. The dynamics of the interaction between the robot agent and the environment is then formalized as the interplay of three functions: the *perceptual filter*, the *state transition function*, and the *agent function*.

The *perceptual filter* $f_p : \mathcal{X} \mapsto \mathcal{O}$ represents the perception capabilities of the robot agent. It takes the world state as its argument and maps it into the space of possible percepts. In many realistic scenarios, the robot agent can only perceive a part of the environment state, which means $\mathcal{O} \neq \mathcal{X}$. The reasons are that (1) the robot sensors have restricted range and are limited through occlusions and (2) the sensors only measure certain physical quantities of the world (e.g. distance to the next obstacles using laser sensors vs. visual appearance using RGB cameras).

The *state transition function*, $f_e : \mathcal{X} \times \mathcal{A} \mapsto \mathcal{X}$, models how the actions of the robot agent change the environment. The function has two arguments, namely

the environment state s and an action a executed in this state, and it returns the environment state s' that results from executing a in s .

Let us now consider how the world evolves when the robot executes sequences of actions. The clock generates a sequence of (discrete) timesteps $T = \{1, 2, \dots\}$, which is the time index set. Then, running the perception-action loop of the robot generates a sequence of timesteps which each timestep identifying an environment state. Thus, the sequence of environment states represents the evolution of the environment. We denote this sequence as the function $X^T : T \mapsto \mathcal{X}$ that assigns every time step precisely one environment state. Analogously, the robot agent's perceptual filter produces a trajectory of percepts, which we denote by $O^T : T \mapsto \mathcal{O}$. Finally, the *agent function* maps the information that the robot agent has collected up to a certain time step t through its percepts into the action selected to be executed in the current cycle, $f : O^t \mapsto \mathcal{A}$. The agent function thus produces a sequence of actions that the robot agent will execute over the course of time, $A^T : T \mapsto \mathcal{A}$.

Putting everything together, we can represent a robot acting in an environment as a pair $\langle A, E \rangle$ with the environment E being represented as a tuple $\langle \mathcal{X}, X_0, f_e, f_p \rangle$, and the robot A as a tuple $\langle \mathcal{O}, \mathcal{A}, f \rangle$, such that the following holds:

1. $X^T(0) = X_0$
2. $X^T(t+1) = f_e(X^T(t), A^T(t)), \forall t \in T$,
3. $O^T(t) = f_p(X^T(t)), \forall t \in T$,
4. $A^T(t) = f(O^t), \forall t \in T$,

where $\mathcal{X}$ denotes the set of possible environment states with a dedicated initial state X_0 , the set of percepts $\mathcal{O}$, the set of possible actions $\mathcal{A}$, a state transition function f_e , a perceptual filter f_p , and an agent function f . Given an agent A and an environment E , we denote the state sequence that is produced by the agents function f in E as the *effects*(f, E).

Our definitions so far only describe the interactions between an agent that is situated in an environment, but they do not measure for the quality of the action sequence A^T the agent function f produces. As the purpose of the agent function is to change the environment according to the goals, we measure the performance of the agent function in terms of the physical changes it causes in the environment. In order to assess the quality of an agent's choice of actions, we introduce a *utility function* $U : \mathcal{X}^T \mapsto \mathbb{R}^+$, where $\mathcal{X}^T = \{X^T \mid X^T : T \mapsto \mathcal{X}\}$ denotes the set of all state sequences. The utility function thus assigns a real-valued score to every possible state sequence, which measures the quality of the state transitions. The agent f 's performance in E can in turn be defined as $V_U(f, E) = U(\text{effects}(f, E))$.

It is important to note that typically the real world is subject to uncertainty. This means that the perceptual filter f_p and the state transitions f_e are non-deterministic in most cases. In real-world scenarios it is thus impossible to access the actual world state and to predict with certainty an action's effects. Instead, in such cases, a distribution over *all* possible environments $\mathcal{E}$ has to be considered. Then, the agent f 's performance can be defined as the *expected* utility of the environment that f

produces:

$$V_U(f, \mathcal{E}) = \sum_{E \in \mathcal{E}} P(E) \cdot U(\text{effects}(f, E)),$$

where $P(E)$ denotes the probability of the world being in state E .

The utility function U gives us an objective measure to assess the quality of an agent's decisions on actions with respect to desired task outcomes and thereby allows us to compare the quality of different agent functions. It also allows us to define our notion of *rationality*. Let $V_U(f, E)$ be the expectation that is obtained by an agent function f in a class of environments $\mathcal{E}$ according to U . Then an *ideal rational agent* is defined by the agent function f^* , such that

$$f^* = \arg \max_f V_U(f, \mathcal{E}).$$

According to this definition, for every possible sequence of percepts, the ideal rational robot agent selects an action that is expected to maximize the agent's performance measure.

3 Knowledge-Enabled Agent Architecture

Let us now turn to the issue of how to realize the agent function. In its simplest form, we can realize the agent function as a function that maps a percept and directly map to the action to be executed next. This kind of realizing the agent function is called a *reflex agent*, which is often implemented in the form of simple if-then rules of the form “if you bump into an obstacle, turn around by 90°.”

For robot agents that are to accomplish more complex tasks, a purely reactive action selection strategy is not sufficient. This holds for several reasons including that often environments are only partially observable, such that the best decision cannot be made on the current percept only. For example, consider the objects in a closed drawer, which the robot should remember after it has detected them. An effective approach to dealing with partial observability is to maintain an internal representation of the state of the environment, which is updated according to the percepts in every iteration of the perception-action loop. For example, the state of the environment can include spatial relations of objects (e.g. locations and containers, etc.) as well as their internal state (e.g. switched on, fill level, etc.). We call this internal representation of the environment a *model* of the environment or a *world model*.

In many cases, the performance of the agent function can be further improved if the agent takes, in addition, its experiences and expectations into account. For complex tasks such as meal preparation, the action selection can be informed not only by the state of the environment but also based on the experience collected from

former actions and the expectations about the effects of future actions. For example, in order to pick up an object from a drawer, the drawer must be opened first to make the objects inside accessible in the case that it is closed. In order to place an object at a specific location, it must be grasped with the agent's gripper first and in order to grasp an object, the gripper must be opened and must not already hold a different object. In order to take the right decisions, the agent thus must also have a model of its actions and their effects in the environment and which conditions the environment must be fulfilled for an action to be applicable. We refer to the specification of an action's effects and their applicability as an action's *pre* and *post conditions*. The preconditions represent conditions under which an action is executable and the postconditions specify the effects that the execution of an action has in the environment.

We consider a *knowledge-enabled robot agent* to be a robot agent that is equipped with symbolic representations of the rational robot agent model. An advantage of knowledge-enabled robot agents over reactive agents is that they can infer action sequences that are predicted to achieve a given goal. In this setting, the goal is being specified as the set of environment states that satisfy the given goal. For example, a possible goal specification of a popcorn preparation task is "the corn has popped and is located in a bowl. All devices are switched off."

A possible realization of the process of inferring a goal-achieving action sequence a knowledge-enabled agent function is visualized in Fig. 3. The action sequence generator enumerates all possible sequences of actions that the robot agent can perform, starting from the initial state of the environment model. By successively applying the pre- and postconditions of all actions in a sequence, the environment models' final state after execution of the whole sequence is obtained. If the resulting state satisfies a goal state, the agent function has found a solution and the respective action sequence is returned for execution. We call such an action sequence a *robot action plan*. If the goal state is not satisfied in the resulting environment state, the next action sequence from the action sequence generator is examined. The process of finding a sequence of actions that achieves goal states is called the *planning problem*.

In the following sections, we will describe the environment model, the action models and the algorithm for finding action sequences for achieving goals.

3.1 Modelling States

Let us now turn to the question of how to equip robot agents with the knowledge they need to solve planning problems. The knowledge should be sufficient to infer an action sequence for every goal the robot is capable to achieve given its perception and action capabilities but not infer action sequences that are not executable or do not achieve the goal. Modeling the world and how it is changed through actions is called *ontological knowledge engineering*.

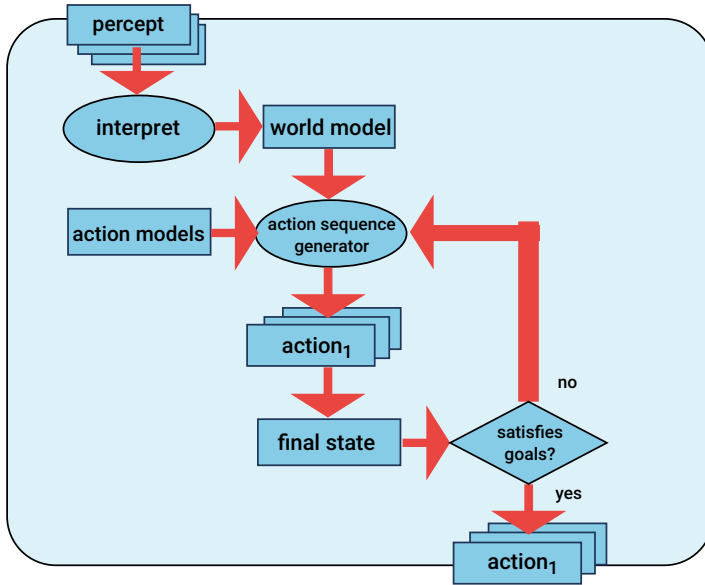


Fig. 3 Schema of a knowledge-enabled implementation of the rational agent function: the sequence of percepts is first interpreted by a perception component of the agent to get an updated formal representation of the world model. Based on the agent's actions and the world state, all possible action sequences are enumerated. By projecting forward the effects of each action in the sequence, a final state is obtained. If the final state satisfies the agent's goals, a plan for achieving a goal has been found, which is returned as a solution

An important design choice for representing the knowledge is to choose an appropriate level of abstraction for the world and action models. If models are too detailed they tend to make unnecessary distinctions and thereby yield unnecessarily large search spaces for action sequences. If the models are too coarse-grained they tend to combine states for which it is not possible to propose a common action model. For the purpose of this chapter, we choose the coarsely discretized and abstract representation depicted in Fig. 4.

We conceptualize the task of popcorn making as a combination actions that transport objects from one location to another one and actions that change the state of objects. The structure of the environment is modelled through a small set of discrete locations including parts of a kitchen counter that has an integrated stove top. The stove top is located in the center area of the counter top and has a turning knob for switching it on and off. To the left and the right of the stove, there is one counter top area on each side where objects can be (intermediately) put down. In the bases below the two counter tops there are one drawer and one shelf each on the left and the right side. In addition to the kitchen counter and its locations we consider the following movable objects: a bowl, in which the unpopped corn is located initially, a plate on which the popcorn is to be served on, a cooking pot, a lid and a salt grinder. We further assume that at each location, there can only be one object at a time.

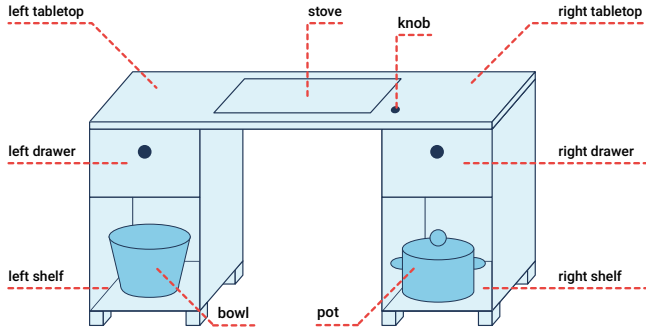


Fig. 4 Abstract state representation for the popcorn world: a kitchen counter with two tabletops (left/right), a stove with a knob in the center, a shelf and a drawer in each of the bases

To model actions that change the states of objects we have to include those states in the environment representation. For example, turning the knob of the stove causes the stove top to heat up, and drawers can be opened and closed. Similarly, cooking the corn on the hot stove for a while will cause it to pop and grinding salt over an object will change the object's state to be salted.

State Space Thus, our conceptualization structures the environment into seven locations: the left and right tabletop, the stove, the left and right drawer, and the left and right shelf. We also choose an atomic representation for the mobile objects by giving them a symbolic name. These objects are the grinder, the bowl, the plate, the corn, and the lid. Some objects have states: the drawers can be open or closed, the knob can be switched on or off, the stove can be hot or cold, and the popcorn can be popped or not.

To represent this state space, we implement the location $\mathbb{L}$, the object states $\mathbb{A}$, and the container states $\mathbb{S}$ as state variables:

- $\mathbb{L} = \{\text{stove} - \text{top}, \text{countertop} - \text{left}, \text{countertop} - \text{right}, \text{drawer} - \text{right}, \text{drawer} - \text{left}, \text{shelf} - \text{left}, \text{shelf} - \text{right}\},$
- $\mathbb{O} = \{\text{grinder}, \text{bowl}, \text{plate}, \text{corn}, \text{lid}, \perp\},$ and
- $\mathbb{A} = \{\text{opened}, \text{closed}, \text{salted}, \text{hot}, \text{popped}, \text{switched} - \text{on}\}.$

The values of the state variables are taken from their respective value domain. Each location variable l in $\mathbb{L}$ can take as its value a mobile object o from $\mathbb{O}$ representing that object o is at location l with $\perp$ representing that l is empty. The object variables $\mathbb{O}$ take on the states $\mathbb{A}$ as their values. For simplicity reasons we abstract away that the states of objects are specific to their categories.²

² We ensure that each object can be in a state of its own state space through the state transitions that are defined in the action models in the subsequent Sect. 3.2.

Three of the objects are containers, namely the two bowls and one cooking pot, with their content represented again as variables, *bowl-content*, *plate-content*, and *cooking-pot-content*, whose domains are the non-enumerable ‘substance’ objects $\mathbb{S} = \{\text{corn}, \perp\}$, i.e. they may contain the corn or be empty.

State of the environment are then represented by assigning a value to each state variable. Consequently, the space of possible world states is given by

$$\mathcal{X} = \mathbb{O}^7 \times \mathcal{P}(\mathbb{A})^5 \times \mathbb{S}^3.$$

Initial State Typically, the representation of the initial (or current) state of the environment is generated by interpreting and abstracting the percepts of the robot agent. If the agent perceives the cooking pot on the left counter, for example, the agent function updates the *countertop-left* component of the world model vector accordingly. At this point it matters whether the world is fully observable or only subsets of the state variables can be observed. For the purpose of our running example we assume that the environment is fully observable. The environment state depicted in Fig. 4 can be represented as a tuple of state variables with the following values:

$$\begin{aligned} \langle \text{stove} - \text{top} = \perp, & \quad \text{countertop} - \text{left} = \text{grinder}, \\ \text{countertop} - \text{right} = \perp, & \quad \text{drawer} - \text{left} = \text{lid}, \\ \text{drawer} - \text{right} = \text{bowl}, & \quad \text{shelf} - \text{left} = \text{plate}, \\ \text{shelf} - \text{right} = \text{cooking} - \text{pot}, & \quad \text{grinder} - \text{attrs} = \emptyset, \\ \text{bowl} - \text{attrs} = \emptyset, & \quad \text{plate} - \text{attrs} = \emptyset, \\ \text{corn} - \text{attrs} = \emptyset, & \quad \text{lid} - \text{attrs} = \emptyset, \\ \text{bowl} - \text{content} = \text{corn}, & \quad \text{plate} - \text{content} = \perp, \\ \text{cooking} - \text{pot} - \text{content} = \perp \rangle \end{aligned}$$

Goal State A goal of the robot agent is a partially specified world state that entails the desired outcome of the task. For example, for a popcorn preparation task, the goal G can be specified by

$$\begin{aligned} \{\text{popped}, \text{salted}\} &\subseteq \text{corn} - \text{attrs} \\ \text{switched} - \text{on} &\notin \text{stove} - \text{attrs} \\ \text{plate} - \text{content} &= \text{corn} \\ \text{countertop} - \text{left} &= \text{plate}, \end{aligned}$$

i.e. the corn must have popped and be salted, the stove must be switched off and the popcorn must be on the plate, which has been put on the left counter top. If a world

state X in which all variables in G are assigned their values in G , X is said to *satisfy* the goal G , which we denote in symbols by $X \models G$.

3.2 Modelling Actions

Robot agents execute actions to change the state of objects and the environment. An action is modelled as an *action schema* that consists of

1. an action symbol and a list of parameters, which identify the action itself and possible variations thereof. As an example, consider the action symbol *grasp*/2, which denotes the action to grasp an object with the robot's gripper. The suffix '/2' indicates that *grasp* has two parameters, for example the object to be grasped and the gripper to use.
2. preconditions specify under which conditions an action is executable and has its specified postconditions (effects). Preconditions are partially defined environment states. For example, possible preconditions for grasping an object are that the gripper to be used does not already hold an object and that the object to be grasped is reachable and liftable.
3. postconditions specify the effects that the execution of a particular action has on the environment. If its preconditions hold, grasping an object, for instance, causes the object to be in the agent's gripper and to disappear from its original location.

Note that this representation is symbolic and abstracts away from how actions are executed. When using this representation we have to assume that the robot is equipped with an action execution system that can execute the symbolic action representation *grasp*(plate,right) in all environment states that satisfy the action precondition and causes effects as specified by the postconditions of the action. We will discuss the issues in realizing action execution systems that satisfy this assumption in Sect. 4.6.

Modeling actions entails several intensively studied representation problems. The first one is the *frame problem*, that is effectively and compactly representing what remains unchanged when an action gets executed. When we execute actions the physical effects are typically local and few, the principle of inertia implies that the rest does not change. The second one is the *qualification problem*, the problem of representing and checking the conditions under which an action is executable and has its specified effects. The conditions often include very rare ones and other ones that are risky to check. For example, the action of starting a car include checking whether the cables are connected, gas is in the tank, and no bananas in the tailpipe. The third one is the *ramification problem*, which includes reasoning about the effects indirectly caused by the action. For example, if the robot transports a box it also transports everything that is contained in the box.

Let us consider the examples of moving an object from a certain location to a different location:

action:	<i>move</i>
parameters:	<i>obj</i> : the object to move <i>from</i> : the object's original location <i>to</i> : the object's target location
preconditions:	$from = obj$ $obj \in \mathbb{O}$ $to = \perp$
postconditions:	$from = \perp$ $to = obj$

In this example, the *move* action can be parameterized with the object that is to be moved, its original location, and the target location, abstracting away from metric to topological relations. The preconditions specify that the object must be located at the given original location, the target location must be empty, and the object must be among the movable objects in $\mathbb{O}$ in order to render the *move* applicable. The postconditions specify that, after *move* has been executed, the environment state is modified, such that the objects original location is empty (i.e. holds the $\perp$ object) and the target location holds the object. An action $A \in \mathcal{A}$ is called *applicable* at time step t , if the world state $X^T(t)$ satisfies the action's preconditions, in symbols $X^T(t) \models \text{pre}(A)$.

3.3 Modelling Environment Interaction and Evolution

The state transition system defined by the state space and the state transition function defines a graph with the nodes of the graph representing the environment states and the links from node s to node s' with the label a represents that the execution of action a in state s results in state s' . Parts of the graph are depicted in Fig. 5. Thus, the actions together with their pre- and postconditions define the state transition function f_e by specifying the way how world states change when actions are performed.

We represent time T as a finite index set of time steps. For any finite sequence of actions A^T , the evolution of the world under execution of A^T can be projected from the initial state X_0 forward by checking the preconditions of $A^T(t)$ against the world state $X^T(t)$ and projecting the postconditions of the respective action to the next world state $X^T(t+1)$ for all time steps t . If $f_e(X^T(t^* - 1), A^T(t^* - 1)) \models G$, where $t^* = \max(T)$, the action sequence A^T produces a final state that satisfies the agent's goal G . We call A^T a *plan* in this case.

Consider a dedicated initial world state X_0 as shown in the middle red circled environment state in Fig. 5. Starting from X_0 and by enumerating all possible actions

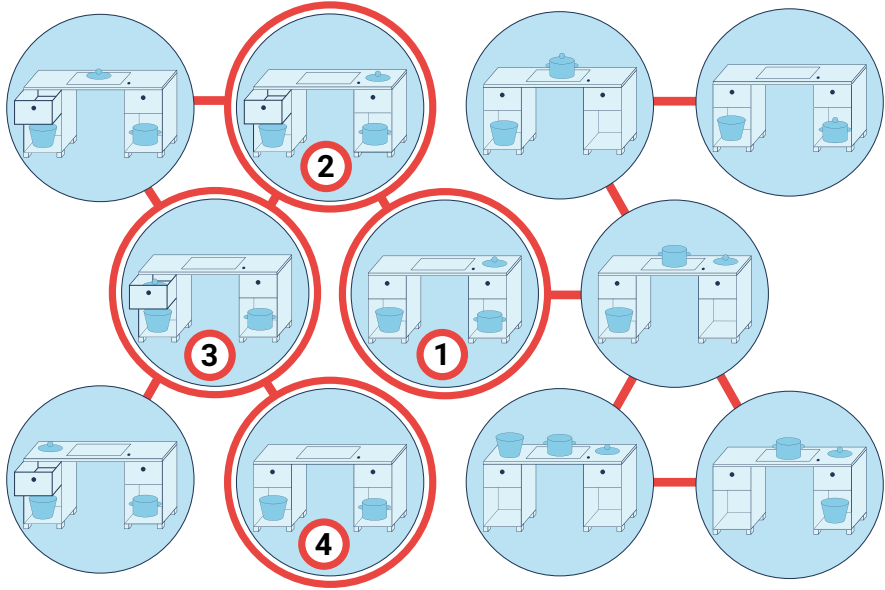


Fig. 5 State transition model with atomic state transitions: all neighboring states differ in the effects of precisely one atomic action. In this example, the actions of moving an object from one location to another and opening a drawer are considered. For spatial reasons, the state space graph is incomplete. The exemplary state sequence (1)–(4) is a possible solution to the task of “tidying up the kitchen”, which may be specified as “the bowl and the pot are in shelves, the lid is in a drawer,” The goal can be achieved by opening one of the drawers, putting the lid into it, and closing the drawer

$A \in \mathcal{A}$ that are applicable by a robotic agent in X_0 , we can generate a directed graph of all possible successor states in the following way:

1. add X_0 as a node to the empty graph.
2. for every action $A \in \{a \mid a \in \mathcal{A}, X_0 \models \text{pre}(a)\}$:
 - (a) add to the graph a new node $X'_0 = f_e(X_0, A)$
 - (b) introduce an edge from X_0 to X'_0 and label it with A .

If we recursively repeat step (2) of this procedure for every successor state X'_0 , we obtain a graph of all possible environment states that can be established by the robot agent starting in X_0 as shown in Fig. 5. Such a graph is called a *state space problem graph*.

The planning problem can now be rephrased as the problem of finding a reachable state in the state space problem graph that satisfies the agent’s goals. As the state space problem graph grows exponentially with the length of action sequences we need to equip robot agents with search methods that can plan without exhaustive exploration of the solution space. For a broad and in-depth explanation of search algorithms, we refer to [2].

4 Logic-Based Knowledge Representation and Reasoning

We will now consider how the conceptualization of the popcorn making scenario that is described in Sect. 3 can be represented in order to infer sequences of robot actions that achieve the goals of the robot. To this end, we use the state space model of popcorn making as our running example. The problem of how to represent the necessary knowledge and reason with it is studied in the field of knowledge representation and reasoning (KR&R).

Shapiro [3] considers *knowledge representation and reasoning* as the study that is concerned with understanding, designing, and implementing ways of representing information in computers so that programs (agents) can use this information

- to derive information that is implied by it,
- to converse with people in natural language,
- to decide what to do next
- to plan future activities,
- to solve problems in areas that normally require human expertise.

McCarthy [4] identifies three fundamental features that an agent must support in order to make use of knowledge about the world:

1. A set of explicit facts³ about the world that the agent already knows (or rather believes) to be true, including atomic propositions, rules, and equivalences, for instance.
2. A TELL-interface that accepts new facts to be added to or removed from (1)
3. An ASK-interface that deduces new beliefs from that ones already existing in (1).

Item (3) refers to the capability of *reasoning* about aspects of the world, which denotes the use of representations of propositions in (1.) in order to derive new ones. In other words, the ability to reason allows to make propositions explicit that are implicitly entailed by the propositions already known.

The basic idea of the use of knowledge representation and reasoning for problem solving is visualized in Fig. 6. A *representation* is a system of symbolic encodings of propositions that are believed to be true in the real world. Consequently, if A represents B , then A is a placeholder for B and is usually more accessible than B . The mapping from a representational symbol to its counterpart in the real world is called its *semantics*. Our everyday life is full of symbols. We use ☺ for restaurant, + for first aid, and we refer to people by their names when talking about them. Symbols do not only stand for things in the world, but can also be used to make general propositional statements. For example, $\text{!} \rightarrow \text{!} \blacktriangleright$ stands for “the restrooms are on the right.”

³ Note, according to the terminology of logics we use the term *fact*, although in robotics the information robots use are often beliefs rather than facts that is that they cannot be guaranteed to be true.

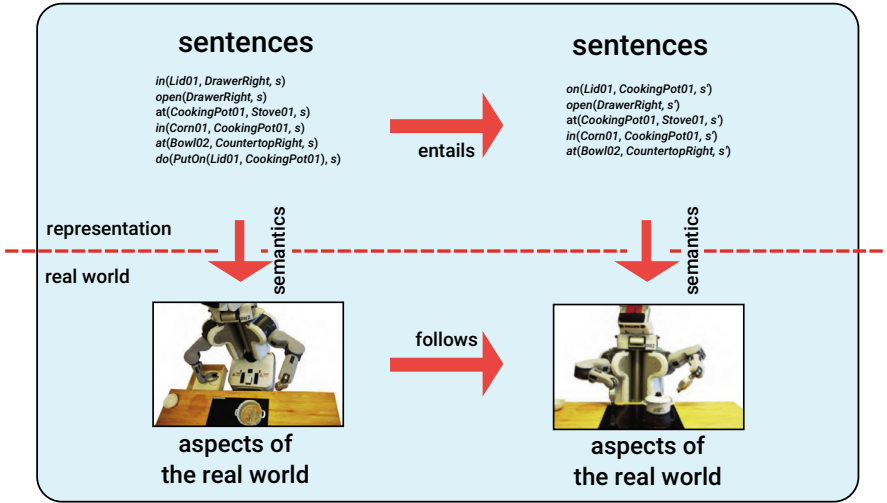


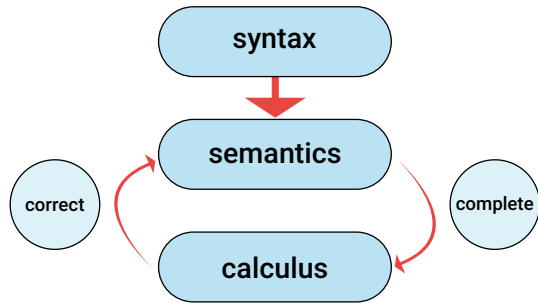
Fig. 6 Knowledge representation hypothesis: “any process capable of reasoning intelligently about the world must consist in part of a field of structures, of a roughly linguistic sort, which in some fashion represent whatever knowledge and beliefs the process may be said to possess.” [5]

Representations are typically abstractions of entities and processes in the real-world. They are reduced to those aspects of the real world that are relevant for solving a particular category of problems. For our popcorn preparation setting, for instance, the groceries and kitchen items are task-relevant, whereas the song currently played on the radio is not relevant for most tasks.

Propositional rules specify how symbols relate to each other and can be used for reasoning. Reasoning denotes the process of manipulating the symbols in the representation to deduce a new representation. Such rules are also called *sentences*. For example, a rule like “IF *perishable* is true THEN *in_refrigerator* is true” can be used for reasoning to draw the conclusion that *in_refrigerator* holds in the context underlying the model whenever *perishable* is believed to be true.⁴ The new deduced representation is then *entailed* by the original one. In an ideal situation, the propositions that are deduced from a representation match the aspects of the real world subject to their semantics.

⁴ Note, the world may not comply with these axioms and there will always be inherent uncertainty in every ‘true’ statement.

Fig. 7 Components of a logic knowledge representation: syntax, semantics, and calculus



4.1 Logic-Based Knowledge Representation

We choose symbolic logic to formalize the entailment of conclusions from premises, in order to represent our model of popcorn making and robot agency in general. As shown in Fig. 7, a logic formalism consists of three components: its syntax, its semantics, and its calculus:

1. the *syntax* determines what can be expressed by defining the set of symbol structures that constitute the logic language. The language is typically defined recursively through a set of atomic symbols and grammatical rules that define how symbolic expressions in the language can be combined to form more complex structures
2. the *semantics* determines what symbolic expressions mean by assigning the truth value to symbolic expression, where the semantics of atomic expressions is specified by an interpretation function and the semantics of a compound expression is computed from its constituents and the logical operator that connects the constituents such as the “and,” “or,” and “negation” operator.
3. the *calculus* that consists of symbolic expressions asserted to be true and rules for generating new symbol expressions from existing ones.

If we use logic to encode problem-solving knowledge then the calculus of a logic can operate as a problem-solver. To this end, knowledge engineers have to carefully define the semantics of the atomic expressions. In addition, the engineer has to assert a set of compound symbolic expressions as being true. The goal is to determine a compact set of expressions (an axiomatization) that together with the rules of the calculus creates only symbol structures that are entailed by the semantics, it is that the axiomatization is *correct* and that all symbolic expressions that are entailed can be generated by applying the calculus to the axiomatization, that is that the axiomatization is *complete*. Thus, if the knowledge engineer comes up with a correct and complete axiomatization of a problem domain, then a symbolic expression that represents some proposition in this domain is true if and only if it can be derived from the axiomatization using the calculus. This is important because this gives us an automatable computation process to test the truth of a symbolic expression.

Propositional Logic Let us consider *propositional logic* (PL) as our first example of a logic. In PL, every symbol is a Boolean proposition that can take the values *true* or *false*. A symbol thus is an atomic sentence, which cannot be further decomposed into finer-grained sentences. Two sentences may be connected by means of logical connectives $\wedge$ ('and'), $\vee$ ('or'), $\neg$ ('not'), $\Rightarrow$ ('implies') and $\Leftrightarrow$ ('equivalent') to form more complex sentences (in descending operator precedence). Brackets () may be used to accommodate precedence of connectives. In PL, we can represent the knowledge that the corn will pop when it is in the pot, the pot is located on the stove and the stove is switched on as follows:

$$\text{corn} - \text{in} - \text{pot} \wedge \text{pot} - \text{on} - \text{stove} \wedge \text{stove} - \text{switched} - \text{on} \Rightarrow \text{corn} - \text{popped}$$

The semantics of this representation of an implication is that, whenever we encounter the antecedent of both *corn-in-pot*, *pot-on-stove*, and *stove-switched-on* to be true, we can be sure that *corn-popped* will also be true. Starting from those three antecedent expressions, a calculus of PL should thus be able to derive *corn-popped* as a new expression.

PL is simple and intuitive, but at the same time limited in expressiveness and generality. What this means becomes evident if we try to apply the above representation to a kitchen with more than one stove tops, let us assume four. In order to represent the same knowledge, in the new environment we would need four sentences, one for every stove top, i.e.

$$\begin{aligned} &\text{corn} - \text{in} - \text{pot} \wedge \text{pot} - \text{on} - \text{stove} - 1 \\ &\quad \wedge \text{stove} - 1 - \text{switched} - \text{on} \Rightarrow \text{corn} - \text{popped} \\ &\text{corn} - \text{in} - \text{pot} \wedge \text{pot} - \text{on} - \text{stove} - 2 \\ &\quad \wedge \text{stove} - 2 - \text{switched} - \text{on} \Rightarrow \text{corn} - \text{popped} \\ &\text{corn} - \text{in} - \text{pot} \wedge \text{pot} - \text{on} - \text{stove} - 3 \\ &\quad \wedge \text{stove} - 3 - \text{switched} - \text{on} \Rightarrow \text{corn} - \text{popped} \\ &\text{corn} - \text{in} - \text{pot} \wedge \text{pot} - \text{on} - \text{stove} - 4 \\ &\quad \wedge \text{stove} - 4 - \text{switched} - \text{on} \Rightarrow \text{corn} - \text{popped}. \end{aligned}$$

Especially in large and complex environments, the representation of knowledge in PL can thus become unwieldy and cumbersome.

First-Order Logic First-order logic (FOL, sometimes also first-order predicate logic, FOPL) is a generalization of PL that allows more complex and more expressive sentences than PL does. Unlike in PL, the representational symbols in FOL stand for single individual entities in the real world. Thus in FOL, a symbol alone is not a well-formed expression. Symbols may identify instances of real physical objects in the world, such as *CookingPot-03* or *Corn-0815*, but may also refer to abstract things like 42, *Kilogram*, or *X*. Propositions are made by means of

predicates that are used to establish certain relations between symbols. For example, a binary *in* predicate can be used to establish a relation between the symbols for the corn and the cooking pot:

$$in(Corn - 0815, CookingPot - 03).$$

This Statement is a well-formed FOL expression and can be interpreted as Boolean proposition that is either *true* or *false*: either the corn is located in the pot or it is not. An expression that consists of a single predicate symbol and a list of representational symbols is called a *ground atom*. Representational symbols are also called *constant symbols*.

Like in PL, expressions in FOL may be combined using the logical connectives to form more complex expressions, such as

$$\begin{aligned} & in(Corn - 0815, CookingPot - 03) \\ & \wedge on(CookingPot - 03, Stove - 01) \\ & \wedge switched_on(Stove - 01) \Rightarrow popped(Corn - 0815). \end{aligned}$$

FOL has *variables*, which act as placeholders for representational symbols in a sentence. Variables help to formulate sentences about *some* or even *all* things in the universe. FOL is therefore suitable for representing universal knowledge and rules very compactly. Quantifying operators may be used to express restrictions on the number of possible variable substitutions for which the sentence must be true. There are two quantifiers, namely a *universal* and an *existential* quantifier. A sentence Φ containing a universally quantified variable x is written as $\forall x. \Phi$, (literally “for all x ”) and a sentence Ψ containing an existentially quantified variable is written $\exists x. \Psi$ (literally, “there exists x ”). Consequently, using the universal quantifier $\forall$, we can represent the knowledge that *in general*, corn pops when it is put in a cooking pot, which is located on a stove and the stove is turned on:

$$\begin{aligned} & \forall x, y, z. type(x, Corn) \wedge type(y, CookingPot) \wedge type(z, Stove) \\ & \wedge in(x, y) \wedge on(y, z) \wedge switched_on(z) \Rightarrow popped(x), \end{aligned}$$

where we have introduced an additional relation *type*, which assigns an object category to every physical entity in the universe. This is necessary, as in standard FOL, the domain of every variable symbol ranges over all constant symbols and the state of being ‘popped’ is only sensible for objects of the type ‘corn’. Variables in a FOL sentence can be bound to a specific constant symbol by replacing every occurrence of the variable in the sentence by the respective symbol. We call the act of assigning a variable a symbol a *substitution*. To indicate that a variable x is being substituted by the symbol X in a sentence Φ , we write $\Phi[x/X]$.

Predicates can be used to attribute certain properties to entities in the world. The *type* predicate, for instance, is *binary* and refers to the property of being of a certain

type, like being a *Stove* or a *CookingPot*. Note that in this case, *CookingPot*, *Corn* and *Stove* do not represent instances of objects in the world but rather abstract type determiners. Object instances are represented by the constant symbols replacing the first argument of *type*. The predicates *switched_on* and *popped* are *unary* and attribute internal states to objects, namely the ‘switched on’ state and the ‘having popped’ state. *on* and *in* are *binary*, i.e. they establish a relation between pairs of entities. The predicates and constant symbols can thus be used to formulate facts about the state in which a robot agent believes the environment to be.

Besides constant and predicate symbols, a third kind of symbols are *functions*. The term ‘function’, in context of FOL, however, does not refer to a kind of sub-routine that takes arguments and returns a value. It is rather to be understood as an elegant way of referring to related things without the need of introducing a symbol for them. For example, we naturally talk about “the location of the corn” without explicitly mentioning what the actual location is. The label “location of the corn” always refers to the correct thing, even if we do not know what it is. Functions in FOL give us the same flexibility in computational knowledge representation. The function *location(Corn01)*, for instance, is a way of denoting the location of the corn in every possible state of the environment without knowing about its actual location. A function is a valid representational symbol itself, so functions can also be nested. *location(location(Corn01))*, for example, denotes the location of the corn’s location. As functions are representational terms, however, they are not valid sentences. Valid sentences about functions can be made through predicates, just like ordinary constant symbols, or by making their identity explicit using the equality operator ‘=’. At this point it is important to note that the semantics of a FOL is typically non-injective and non-surjective. This means that the mapping from symbols to objects in the real world is not necessarily complete and unambiguous, such that there may be objects that are not represented by any symbol, and there may be objects referred to by more than one symbol. The fact that two symbols refer to the identical object can be stated with the equality operator. In our running example, for instance,

$$location(location(Corn01)) = TableTopLeft$$

states that the location of the corn’s location points to the same object as *TableTopLeft*. The nested *location* function can be regarded as a representational ‘shortcut’ without the need to introduce a symbol for the location of the corn, if it is irrelevant in the expression, if the corn is located in the cooking pot or in the bowl, for instance. Functions in FOL are extremely powerful, but they also severely affect the computational complexity of reasoning, as we will discuss in greater detail in Sect. 4.4.

FOL is a powerful formalism to describe real-world situations such as our popcorn world. However, the representations presented above describe rather static situations, ‘snapshots’ of the kitchen counter taken at a particular time step. In the following, we will introduce a framework that brings dynamics into FOL representations and thus allows to manipulate and evolve the world over time.

4.2 Logic-Based Action Planning

In order to model the evolution of the world over time, we describe a framework in FOL called *situation calculus*. In situation calculus, world states and actions' pre- and postconditions are bound to an abstract symbol representing a certain *situation*. In context of a robot agent, a situation represents one cycle in the perception-action loop:

1. Technically, a *situation* is denoted by a term in FOL, i.e. the constant symbol S_0 denoting the initial situation, or a function $do(a, s)$ representing the situation that is obtained when an action a is executed in situation s .
2. *Actions* are represented by terms in FOL. Constant symbols stand for actions without arguments (e.g. *Stop*), and functions represent actions that can be parameterized (e.g. *Open(DrawerLeft)*). There is a dedicated predicate $possible(a, s)$, which states that an action a is applicable in situation s .
3. *Fluents* represent certain aspects of a world state that may change from one situation to another. Fluents can be represented by predicates, like $\neg contains(Pot01, Corn01, S_0)$ or functions like $temperature(StoveCenter, S_0)$. Fluents are always bound to specific situations that they refer to because they may change their values over time. States that are constant over all situations are called *atemporary* and do not need to be assigned a situation. Examples of such predicates are the *type* assertions as introduced above, such as $type(Corn01, Corn)$.

The function *do* is also called *projection* as it predicts the states of future situations that are obtained through action execution. By nesting multiple *do* functions, one can represent the final situation that is obtained after execution of a complete sequence of actions. For example, the situation after execution of the three actions

- $Put(Pot01, StoveCenter)$
- $Pour(Bowl01, Pot01)$
- $TurnKnob(switch(StoveCenter))$

can be denoted by

$$\begin{aligned} &do(=TurnKnob(switch(StoveCenter)), \\ &do(=Pour(Bowl01, Pot01), \\ &do(Put(Pot01, StoveCenter), S_0))) \end{aligned}$$

By convention, the last argument in the list of arguments (if any) always denotes the situation symbol.

State Axioms We have to describe the state of the environment in the initial situation S_0 through fluent and atemporal axioms. In order to describe the popcorn world, for instance, we have to assert the types of objects in our environment:

$$\begin{aligned}
 &type(TabletopLeft, Location) \wedge type(StoveCenter, Location) \\
 &\quad \wedge type(TabletopRight, Location) \\
 &\quad \wedge type(DrawerLeft, Drawer) \wedge type(DrawerRight, Drawer) \\
 &\quad \wedge type(ShelfLeft, Shelf) \wedge type(ShelfRight, Shelf) \\
 &\quad \wedge type(Corn01, Corn) \wedge type(Pot01, CookingPot) \\
 &\quad \wedge type(Bowl01, Bowl) \wedge type(Plate01, Plate)
 \end{aligned}$$

The atemporal axioms may also represent a subsumption relation in order to construct a hierarchy of concepts, e.g.

$$\begin{aligned}
 &\forall x. (type(x, Bowl) \vee type(x, CookingPot) \vee type(x, Plate)) \Rightarrow type(x, Container) \\
 &\forall x. (type(x, Bowl) \vee type(x, CookingPot) \vee type(x, Corn) \vee type(x, Bowl) \\
 &\quad \vee type(x, Lid) \vee type(x, SaltGrinder)) \Rightarrow type(x, MovableObject),
 \end{aligned}$$

which enables us to refer to collections of object types in later axioms. The fluents represent the mutable parts of a situation, such as the object locations:

$$\begin{aligned}
 &location(Pot01, ShelfRight, S_0) \\
 &\quad \vdots
 \end{aligned}$$

Note that, in standard FOL, it is insufficient to only specify the positive axioms but we also need to declare what does *not* hold. It is, however, common to make a *closed-world assumption*, which means that the framework assumes all ground atoms to be false, whose truth values have not been explicitly declared.

Action Axioms Actions are represented by two kinds of axioms, the *possibility axiom*, which specifies under which circumstances an action is possible, and a *successor state axiom*, which describes the way in which the successor situation is affected by an action. The possibility axioms are sentences in FOL of the form

$$\forall s. \psi(a) \Rightarrow possible(a, s),$$

where $\psi(a)$ is a sentence with a free variable a specifying the preconditions of the action A . The successor states are specified by axioms of the form

$$\forall a, s. possible(a, s) \Rightarrow \left(\phi(do(a, s)) \Leftrightarrow \left(\varepsilon_{\phi}^+(a, s) \vee \left(\phi(s) \wedge \neg \varepsilon_{\phi}^-(a, s) \right) \right) \right), \quad (1)$$

where $\phi(s)$ is a fluent declaration in situation s and $\varepsilon_{\phi}^{+}(a, s)$ is a specification of the positive effects of an action a on ϕ in situation s . Analogously, $\varepsilon_{\phi}^{-}(a, s)$ specifies the negative effects of action a on ϕ in s . An axiom of the form of Eq. (1) can be read as follows: “In a situation s where action a is applicable, the fluent ϕ will hold after execution of a if and only if a has made ϕ true (positive effect), or ϕ was already true in s and a did not make ϕ false (negative effect).”

The situations in situation calculus can be compared to single frames on a filmstrip, which is why these axioms are also called *frame axioms*. The action effects specifications determine the changes from one frame to the next one. However, only considering the aspects that change is not enough. In most cases, only a small subset of fluents will be affected by an action and nearly all others stay the same. In FOL, this needs to be explicitly stated. In order to account for this problem, which is also known as the *frame problem*, it is necessary to also consider $\varepsilon_{\phi}^{-}(a, s)$ in the successor state axiom.

As an exemplary action, we consider the action of moving an object from one location to another. In our popcorn world, we consider seven locations, at which movable objects may be put down, i.e. two counter tops, two shelves, two drawers and the stove top. For simplicity, we assume that each location is able to hold only one single object at a time. Let us first examine the possibility axiom for $Put(o, l)$, which denotes the action of putting an object o to the location l . Obviously, based on the above presumptions, $Put(o, l)$ is only possible if o is movable and l is not already occupied by another object. In addition, we have to consider that objects in drawers are not accessible unless the drawer is open. We can capture this knowledge in a fluent *accessible* and a possibility axiom,

$$\begin{aligned} \forall o, s. \text{accessible}(o, s) &\Leftrightarrow \exists l. \text{location}(o, s) = l \wedge (\text{type}(l, \text{Drawer}) \Rightarrow \text{open}(l, s)) \\ \forall o, l, s. &(\text{type}(o, \text{MovableObject}) \wedge \text{type}(l, \text{Location}) \\ &\wedge \neg(\exists o'. \text{location}(o', s) = l) \wedge \text{accessible}(o, s)) \Rightarrow \text{possible}(Put(o, l), s). \end{aligned}$$

For the *accessible* fluent of an object o to hold in situation s , o 's location must either not be a drawer, or, if it is a drawer, it must be open. Otherwise, the object is not accessible. The positive effect of $Put(o, l)$ is that the object will be at location l in the successor state, whereas the negative effect is that it will disappear from its original location:

$$\begin{aligned} \forall a, o, l, s. \text{possible}(a, s) &\Rightarrow \\ (\text{location}(o, \text{do}(a, s)) = l &\Leftrightarrow a = Put(o, l) \\ \vee (\text{location}(o, s) = l \wedge \neg(\exists l'. l' &\neq l \wedge a = Put(o, l'))), \end{aligned}$$

i.e. the location of an object o will be l in the successor state, if and only if o has been put to l or it has already been at l and was not put away to some other location l' .

4.3 Inferring a Popcorn Making Plan

In this section, we introduce an axiomatization of the popcorn world, which allows us to derive a sequence of actions that achieves a subgoal of the popcorn making plan. We leave the specification of remaining axioms as an exercise to the reader. Let us consider the action sequence

1. *Put*(*Pot01*, *StoveCenter*),
2. *Pour*(*Bowl01*, *Pot01*),
3. *TurnKnob*(*switch*(*StoveCenter*)),
4. *Wait*(3),
5. *Put*(*Plate01*, *TabletopLeft*),
6. *Pour*(*Pot01*, *Plate01*),

We want to model actions and states in a way, such that a robot agent is able to automatically derive this action sequence as a plan that achieves its goal of having made popcorn in the kitchen environment and having served it on a plate. The initial state of the environment shall be

$$\begin{aligned}
 \text{location}(\text{Bowl01}, S_0) &= \text{CounterTopRight} \\
 \text{location}(\text{Pot01}, S_0) &= \text{ShelfRight} \\
 \text{location}(\text{Plate01}, S_0) &= \text{ShelfLeft} \\
 \text{in}(\text{Corn01}, \text{Bowl01}, S_0) \\
 \neg \text{switchedOn}(\text{StoveCenter}, S_0) \\
 \neg \text{popped}(\text{Corn01}, S_0)
 \end{aligned} \tag{2}$$

i.e. the unpopped corn is in the bowl on the right counter top, the cooking pot is in the right shelf, the plate in the left shelf, and the stove is switched off.

Let the goal be that the corn has popped and is on a plate on one of the counter tops, i.e.

$$\begin{aligned}
 \exists c, p, s. \text{type}(c, \text{Corn}) \wedge \text{popped}(c, s) \wedge \text{type}(p, \text{Plate}) \\
 \wedge \text{in}(c, p, s) \wedge \text{location}(p, s) \in \{\text{TabletopLeft}, \text{TabletopRight}\}.
 \end{aligned} \tag{3}$$

By using existentially quantified entities, we have made the goal specification independent of specific instance names in the model.

In the previous section, we already provided a model of the *Put* action, which we can reuse so we move on and create a model of *Pouring*. The *Pour*(*f*, *t*) action denotes the transfer of a substance contained from a source container *f* to a destination container *t*. For simplicity, we will assume here that any container can hold exactly one unit of volume, and that *Pour* will cause a complete transfer, i.e. there will not be any leftovers in the source container after pouring. As

preconditions, we require the destination container to be empty and unobstructed. Unobstructed means that the container is not covered by anything (like a lid), and that there is sufficient free space above its opening such that the pouring can be performed. Obviously, in our popcorn world, a container is only unobstructed if it is located at one of the counter tops or the stove, and there is nothing on top of it, i.e.

$$\begin{aligned} \forall o, s. \text{unobstructed}(o, s) &\Leftrightarrow \text{location}(o, s) \\ &\in \{\text{CounterTopLeft}, \text{CounterTopRight}, \text{StoveCenter}\} \\ &\wedge \neg(\exists o'. \text{on}(o', o, s)). \end{aligned}$$

We can now define the preconditions, under which pouring is possible, as

$$\begin{aligned} \forall f, t, s. (\text{type}(f, \text{Container}) \wedge \text{type}(t, \text{Container}) \\ \wedge \exists c_1. \text{in}(c_1, f, s) \wedge \neg(\exists c_2. \text{in}(c_2, t, s)) \\ \wedge \text{accessible}(f, s) \wedge \text{unobstructed}(t, s)) \Rightarrow \text{possible}(\text{Pour}(f, t), s), \end{aligned}$$

i.e. both f and t must be containers, f must be non-empty and accessible, and t must be empty and unobstructed. As postconditions, *Pour* will cause the contents of the source container to be in the destination container, whereas it ceases being in the source container:

$$\begin{aligned} \forall a, t, f, c, s. \text{possible}(a, s) \Rightarrow \\ (\text{in}(c, t, \text{do}(a, s)) \Leftrightarrow a = \text{Pour}(f, t) \wedge \text{in}(c, f, s) \\ \vee (\text{in}(c, t, s) \wedge \neg(\exists t'. t' \neq t \wedge a = \text{Pour}(f, t'))), \end{aligned}$$

The most interesting part of our popcorn example is the event when the corn pops. It is so interesting because it involves not only actions that transform the world state, but also physical and chemical processes that are captured in logical axioms. We will model the process in three parts: (1) the functionality of a switch, (2) the functionality of a stove, and (3) the popping process.

The core functionality of a switch is to toggle the state of an electrical device d in a situation s between *switchedOn* and $\neg$ *switchedOn*. We introduce a *TurnKnob*(k) action, which operates on a knob and flips an electrical device's state. It is possible to execute *TurnKnob*(k) if k is a knob and belongs to an electrical device:

$$\begin{aligned} \forall k, s. (\text{type}(k, \text{Knob}) \wedge \exists d. \text{switch}(d) = k \wedge \text{type}(d, \text{ElectricalDevice})) \\ \Rightarrow \text{possible}(\text{TurnKnob}(k), s), \end{aligned}$$

where $switch(d)$ is a function denoting the switch of an electrical device. $TurnKnob(k)$ will change the state of the device it belongs to from $\neg switchedOn(d, s)$ to $switchedOn(d, do(TurnKnob(k), s))$, and back:

$$\begin{aligned} \forall a, d, s. possible(a, s) \Rightarrow \\ (switchedOn(d, do(a, s)) \Leftrightarrow a = TurnKnob(switch(d)) \\ \wedge \neg switchedOn(d, s) \\ \vee switchedOn(d, s) \wedge a \neq TurnKnob(switch(d))), \end{aligned}$$

When a stove is switched on, everything on its top will be exposed to heat. We can capture this introducing a fluent $heated(o, s)$,

$$\begin{aligned} \forall o, s. heated(o, s) \Leftrightarrow \exists c, l. type(l, Stove) \wedge switchedOn(l, s) \\ \wedge location(c, l, s) \wedge (c = o \vee in(o, c, s) \wedge type(c, CookingPot)). \end{aligned}$$

Note that in this formula, we have also modeled the heat conductivity of cooking pots, such that everything inside a pot will also be exposed to heat, when the pot is on a stove and the stove is switched on.

The final missing step in our world model is the popping process of the corn. In order for the corn to pop, it must be exposed to heat for 3 minutes. In a real-world setting, it would be desirable if a robot agent were able to spend the time until the corn has popped more useful, but for now we assume that it bridges this idle time with waiting. To this end, we introduce an action $Wait(t)$, during which the agent sleeps for the given amount of time. First, let us specify that it is always possible to wait and do nothing:

$$\forall t, s. possible(Wait(t), s)$$

Time consuming processes like cooking or boiling can be elegantly modeled as postconditions of a waiting action, given that the world model satisfies certain conditions. In our example, we can state that the corn will have popped as a result of waiting for three minutes in a situation in which the corn is exposed to heat, i.e.

$$\begin{aligned} \forall a, c, s. possible(a, s) \Rightarrow \\ (popped(c, do(a, s)) \Leftrightarrow a = Wait(3) \wedge heated(c, s) \wedge \neg popped(c, s) \\ \wedge type(c, Corn) \vee popped(c, s). \end{aligned}$$

Note that, in this case, there is no negative effect in the state successor axiom as the property of having popped is irreversible.

Starting from the initial state in Eq. (2), it can be proven that our goal state in Eq. (3) can be reached by the action sequence

```
do(Pour(Pot01, Plate01),
  do(Put(Plate01, TabletopLeft),
    do(Wait(3),
      do(TurnKnob(switch(StoveCenter)),
        do(Pour(Bowl01, Pot01),
          do(Put(Pot01, StoveCenter), S0))))))
```

Note that this solution is not unique. There are many alternative action sequences that achieve the goal state and many other ways of modelling the actions and processes one can think of. In KR&R, there is an entire subfield called qualitative physics and qualitative process theory that addresses this problem. A more thorough introduction to this field is beyond the scope of this chapter. We refer the interested reader to [5].

4.4 Complexity of Logical Reasoning

Representing models of the world and actions in situation calculus using FOL is extremely powerful and expressive, but comes with practical limitations. The perhaps most severe limitation affects the computational complexity of reasoning. Reasoning in logic denotes the automated verification of sentences given the facts and rules about the world. As an example, consider a rule, such as

$$\forall x. \text{type}(x, \text{Bowl}) \Rightarrow \text{type}(x, \text{Container}), \quad (4)$$

as part of a taxonomic relation stating that every object of the type bowl also is of the type container, and the fact $\text{type}(\text{Bowl01}, \text{Bowl})$ that assigns the entity *Bowl01* the type *Bowl*. In order to prove that $\text{type}(\text{Bowl01}, \text{Container})$ also holds, we have to find a substitution of x in Eq. (4), such that the formula gets satisfied by its atomic constituents. This is the case for the substitution $[x/\text{Bowl01}]$, as

$$\begin{aligned} & (\text{type}(x, \text{Bowl}) \Rightarrow \text{type}(x, \text{Container}))[x/\text{Bowl01}] \\ & \equiv \text{type}(\text{Bowl01}, \text{Bowl}) \Rightarrow \text{type}(\text{Bowl01}, \text{Container}) \\ & \equiv \neg \text{type}(\text{Bowl01}, \text{Bowl}) \vee \text{type}(\text{Bowl01}, \text{Container}), \end{aligned}$$

which is satisfied if and only if $\text{type}(\text{Bowl01}, \text{Container})$ holds. The reasoning problem in FOL can thus be phrased as a search problem in the space of possible variable substitutions, which is called the *Herbrandt universe*. The elimination of variables by substitution through constant or function symbols is also called *grounding* or *propositionalization*, as a sentence whose variables have been substituted corresponds to an expression in propositional logic. However, the problem of finding

a truth assignment satisfying a sentence in PL has been proven NP complete [6], so the computational expense of reasoning does scale exponentially in the number ground atoms. In addition, the Herbrandt universe typically is infinite as it contains both constant symbols and functions. As a consequence, it is not guaranteed that a search algorithm will terminate in all cases. It can be shown that every proposition that is entailed by a knowledge base in FOL can be deduced from a finite subset of the Herbrandt universe, whereas no method exists that is capable of proving non-entailment for all propositions that are not entailed by the knowledge base. This makes the reasoning problem in FOL *semi-decidable* [7–9]. This means that for reasoning tasks that robots have to do it often pays off to limit the expressiveness of the representation language to what is needed and to think about whether the inference tasks can also be accomplished by more specific algorithms that can exploit the problem structure. Fast reasoning is particularly important for robot agents that are to act efficiently as in many cases the robot can start acting only after the reasoning tasks are completed.

4.5 Implementations of Planning Systems

Due to the undecidability of FOL, practical implementations of KR&R systems use subsets FOL that make reasoning feasible. We will in the following briefly depict the most prominent action planning systems for robot agents.

In principle, there are two possible ways of solving a planning problem, namely *progression* planning and *regression* planning. Progression planning starts with the initial situation S_0 and builds up a state space graph as described in Sect. 3.3 by applying action sequences to S_0 and monitoring the progress of the model, until a sequence is found that produces a final situation satisfying the goal specification. In contrast, regression planning starts with the goal specification, and searches backwards an action sequence whose first action is applicable in S_0 . In many cases, regression planning is computationally more efficient than progression planning. The most important reason for this is that in regression planning, only the relevant actions need to be considered, i.e. actions whose effects make the goal specification effectively true, whereas in progression planning, all applicable actions have to be considered.

STRIPS One of the first and best-known frameworks for formulating and solving planning problems is the STanford Research Institute Problem Solver (STRIPS) by Fikes and Nilsson [10]. In essence, STRIPS is based on situation calculus but uses a restricted terminology to formulate rules. In STRIPS, action pre- and postconditions and goal specifications are required to be conjunctions of function-free literals. A literal is an atomic expression, which may also be negated. Action postconditions are also required to be function-free literals. By disallowing function symbols, the Herbrandt universe in STRIPS is finite and thus decidability can be guaranteed. In addition, STRIPS does not produce copies of all axioms in every situation, but

maintains a single world representation that is modified by the reasoning procedures. The state representation only consists of positive literals. All literals not explicitly mentioned are assumed to be *false* (closed-world assumption). Negative effects of actions cause the respective literals to be deleted from the world state, positive effects are added to it. Action effects are thus typically divided into an *addition* and a *deletion* list. A plan in STRIPS is an action sequence that, when being executed in the initial state, leads to a state that satisfies the goal.

PDDL The Planning domain definition language (PDDL) developed by Ghallab et al. [11]. Like in STRIPS, actions are defined by their preconditions and effects. However, these conditions are allowed to take not only the form of conjunctions of literals, but also universally and existentially quantified variables and implications are allowed. PDDL is based on a *typed* logic, i.e. variables may have domain constraints, which can simplify knowledge engineering enormously since most entities in the real world can be assigned some sort of category. A very powerful feature of PDDL is the concept of hierarchical action descriptions, which allows to model activities that are composed of multiple simpler actions. These subactions can be specified to be executed in a series or in parallel. For example, in a table-setting scenario, the order in which items are brought to the table is irrelevant, whereas in a recipe, the steps must be executed one after another. The result in such cases is called a *partially-ordered* plan. PDDL has been designed as a language unifying many features of planning systems at the time of its inception with the goal of fostering comparability, reuse and exchange of models. It has since then evolved to a quasi-standard that is used in many international planning competitions.

4.6 Discussing Logic-Based Planning of Popcorn Making

The logic-based approach to reasoning about the actions of a robot agent has several attractive advantages over other forms of robot decision making. Given a model formalized as a set of logical axioms, which correctly and completely represents a robot, its actions, and the environment in some model, a robot agent can compute action plans for every goal that it is able to accomplish [12]. Further, every action plan that the robot agent computes this way is guaranteed to be correct with respect to the formalized model. The set of axioms represents the knowledge about actions, their preconditions, and effects in a very modular, transparent, and explicit way. For every deduced action plan the robot agent can use the proof for the plan existence as an explanation of how the derived plan can achieve the goal. Finally, the competence of the robot agent can be incrementally increased by stating additional axioms formalizing new objects, actions, and environment conditions.

These advantages motivated many researchers to pursue a logic-based approach to reasoning about action. The logical approach to reasoning about action is presented in several textbooks, most notably in the ones by Reiter [13], Genesereth and Nilsson [14], Davis [15], and Mueller [16]. Seminal research papers promoting

the logical approach to reasoning about action include [4, 17], [18–20]. There have also been attempts to standardize knowledge representation and reasoning, as for example, in the Knowledge Interchange Format [21].

One caveat about the logical approach is that its advantages can only be leveraged if the respective robots, environment, and actions can be faithfully formalized as a consistent set of axioms. How to come up with sets of axioms that are consistent, logically entail all intended conclusions, and forbid unwanted ones is often a difficult knowledge engineering and research activity.

The research work on knowledge-based reasoning about actions is much too extensive to be covered in a book chapter. Therefore, in the remainder of this section we sketch relevant extensions to the logic-based reasoning approach and refer to textbooks and seminal papers for more thorough and deeper accounts of the respective research topics.

When trying to come up with the appropriate logic-based axiomatizations for reasoning about robot actions one often experiences a lack of expressiveness and limitations of the state transition system model with atomic state transitions. In response to these limitations, various extensions have been proposed to deal with parallel actions [22], temporal aspects of action execution [23–25], resource consumption through actions [26, 27], and other shortcomings of the original approach.

Another limitation that is particularly critical for the control of robot agents is the inability of dealing with nondeterministic and probabilistic aspects of actions [28]. To respond to these limitations researchers have investigated non-monotonic [29, 30], probabilistic [31, 32] and decision-theoretic approaches [33–36] to decide on the course of action that has the highest expected utility. In order to further increase the expressive power of the representation mechanisms researchers have also proposed approaches that can reason about the information preconditions of actions, such as knowing the combination before opening a safe [37, 38] and for representing not only the state of the environment but also an agent's beliefs with respect to the environment state [39, 40].

Some of the reasoning problems addressed are particularly relevant for autonomous robotics such as the indistinguishability of objects with respect to their perceptual features. Perhaps the best known is the “bomb into the toilet” problem [41], in which a robot looking at two identical packages is asked to disarm the bomb by putting it into the toilet. In this case, the proper plan is to put both packages into the toilet because it is the only way to ensure that the bomb is disarmed. This problem is closely related to the so-called *symbol grounding problem* [42]. The symbol grounding problem entails the problem that the semantics of logic languages assigns symbolic names objects in the real world as their meanings. This is critical because in many cases the perceptual apparatus of robots cannot effectively distinguish between different objects in question. One way of dealing with these problems is to extend the scope of reasoning also to the grounding of symbols. When setting the table the identity of a cup typically does not matter as long as the cup is clean and empty. When disarming a bomb, all candidate individuals should be disarmed in order to avoid an explosion. Only reasoning

systems that can reason about the consequences of grounding symbols one way or the other can deal in informed ways with cases where the robot control system cannot properly ground a symbolic representation.

A heated debate about deduction being sufficient for realizing the common-sense capabilities of humans was triggered by the so-called Yale shooting problem [43]. The problems in solving the Yale shooting problem were taken by a number of researchers as an indication that a single logical reasoning mechanism might not be sufficient to solve both prospective and diagnostic reasoning tasks in reasoning about action [41]. One way of dealing with this issue is to employ multiple expert reasoners [44], which, however, also yields the problem of how to deal with situations where different reasoners propose different solutions.

There have also been attempts to formalize more detailed and realistic models of manipulation actions in logic formalisms. This research agenda was originally been put forward in the two naive physics manifestos by Hayes [20, 45]. Perhaps the most insight providing activity was a challenge put forward by Davis and colleagues in which they asked for axiomatizing the task of cracking an egg and separating the egg yolk from the egg white [46]. The reasoning problem is that given an activity description of an egg cracking activity the reasoner has to answer an open collection of questions including: *“What happens if: The cook brings the egg to impact very quickly? Very slowly? The cook lays the egg in the bowl and exerts steady pressure with his hand? The cook, having cracked the egg, attempts to peel it off its contents like a hard-boiled egg? The bowl is made of looseleaf paper? of soft clay? The bowl is smaller than the egg? The bowl is upside down? The cook tries this procedure with a hard-boiled egg? With a coconut? With an M&M?”* This exercise showed the dependence of the axiomatizations on the proper choice of abstractions and how quickly axiomatizations become overly complex as action models become more realistic [47].

If we apply the knowledge representation and reasoning methods proposed in the previous section to the control of robot agents we see an important weakness of this approach. This weakness is that choosing the state transition system model with atomic state transitions as a model we make the assumption that the effects of actions are independent of the movements that a robot executes. This contradicts the view of robotics as the field that studies how to move the body of the robot to accomplish tasks.

5 KR&R Within Perception-Action Loops

In order to extend the knowledge representation and reasoning techniques we have discussed so far to reason about *how* to execute actions we will now consider more realistic representations of perception-action loops. This approach takes the view that a computational model for controlling a robot agent accomplishing complex manipulation tasks has to explain how vague instructions such as “pour the popcorn onto the plate” can be translated into the body movements that the robot has to

generate. To infer such a body motion the robot the robot needs to infer that it has to grasp the pot by the handles, hold the pot horizontally, tilt the pot around the axis between the handles, remove the lid before pouring, and so on.

5.1 Linking Representation to Perception and Motion

In order to embody knowledge representation and reasoning into robot control systems we need to address the following aspects:

- proposing suitable representations of motions that enable us to combine knowledge representation and reasoning with control engineering, cognitive psychology, and robot learning;
- maintaining and acquiring environment and object representations that are detailed enough for object manipulation; and
- maintaining and updating the symbolic representations by the robot perception system.

5.1.1 Representation and Reasoning About Motions

Let us first look at suitable representations for robot body motions and their physical effects. To develop such representations we can take inspiration of how humans accomplish their manipulation actions through the learning, planning, and execution of body motions, which is intensively investigated in a research field that [48] call action science. Action science lies in the intersection of several research disciplines including cognitive psychology, cognitive neuro science, and sport science. These research threads contribute to the representation and reasoning about actions as they investigate more detailed and realistic models of manipulation actions.

For example, [49] propose a widely used conceptual model for representing and reasoning about the motions that constitute human manipulation actions. This model is depicted in Fig. 8. In this model the body motions of manipulation actions are composed of motion phases. For example, fetching and placing an object is composed of a reaching, a lifting, a transport, and a release phase. Each of these motion phases has a motion goal. The goal of the reaching phase is that the object is grasped, of the lifting phase that the object is lifted off the supporting surface, of the transport phase that the object is at the destination, and of the releasing phase that the object has left the hand. The transitions between the motion phases are so-called force-dynamic events [50]. The force-dynamic events, which also cause distinctive perceptual events in the perceptual system of the robot agent are the contact of the hand with the objects, the hand feeling the gravity of the object, the collision of the object with the destination surface, and the hand losing the grip of the object. The individual motion phases also have knowledge preconditions. When starting the

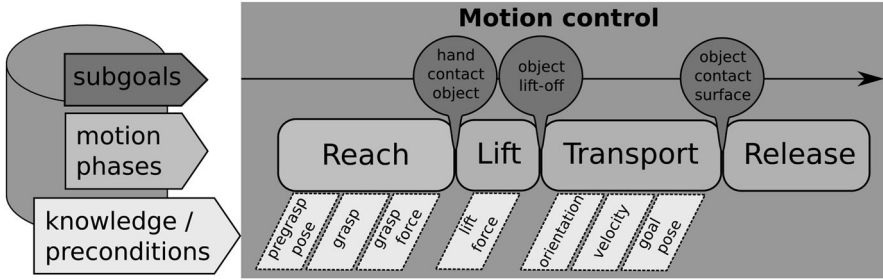


Fig. 8 Model of motion phases of a fetch and place action according to Flannigan et al.

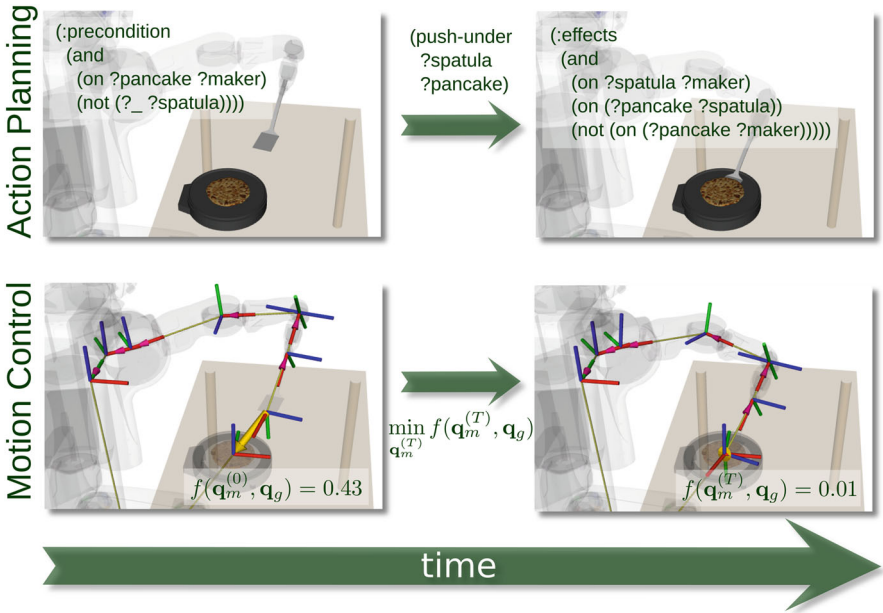


Fig. 9 Comparing the state transition system model with the constraint- and optimization-based control model of manipulation actions

reaching motion the robot must have decided on the target pose, when starting the lifting motion the robot must have committed to a holding pose, and so on.

Another approach to represent the manipulation actions at the motion level is to represent the conceptual model underlying the constraint- and optimization-based control frameworks [51, 52]. In constraint- and optimization-based control the bodies of robot agents are represented as chains of control frames (6D poses of key control points of the robot). Figure 9 compares the state transition model with the constraint- and optimization-based control model. While the logic approach represents the objects and effects, the control approach specifies constraints and objectives on motions. Therefore, the semantics of the manipulation actions, in

particular the causal relation between motion parameterization and their physical effects, has to be encoded into the constraints and objectives for the motions.

5.1.2 Knowledge-Enabled Robot Programming

When applying the Flannagan model the task of reasoning is to infer the motion parameters for the individual motion phases and in constraint- and optimization-based control the reasoning system has to infer the motion constraints and objective functions to be optimized. This can be accomplished through knowledge-enabled robot programming.

In the knowledge-enabled programming style programmers equip robots with a combination of generalized action plans and a knowledge base of general, modular knowledge pieces. The basic idea of knowledge-enabled programming is to separate knowledge from the control program and to modularise it into small broadly applicable chunks, such as “grasp objects by their handles”, “hold filled containers upright”, “do not squeeze breakable items too hard”, etc. The program would ask: “How should I pick up the object in front of me?”. A reasoning mechanism would then collect the relevant knowledge units and combine them in order to propose the appropriate grasp. The primary advantage is that these knowledge pieces can be collected and that they can be reused by other robots accomplishing different tasks with different objects in different environments [53]. This reuse of generalized knowledge can, therefore, drastically accelerate the realisation of new robot applications. Figure 11 shows examples of such generalized knowledge chunks, which can be conveniently stated as logical axioms.

This idea of leveraging generalized plans for manipulation actions is also supported by a number of cognitive scientists including Schmidt [54] suggest that movements within a class, such as fetch and place and pouring, share the same order of sub-movements and their relative timing. If these programs are to be executed in different contexts they are called generalized motion programs (or motion schemata) and include variable parameters (e.g., coordinates, body poses, timing, forces) are to be specified to the program. The parameterizations define the details of the movement and how it is executed for particular objects in particular contexts for different purposes. These generalized motion programs are called generalized motion plans if they cannot only be executed but also reasoned about and modified at execution time.

Figure 10 show a generalized motion plan for pouring actions. The plan consists of the signature of the plan and the body of the plan. The signature includes the name of the plan, which is *pour* and the formal parameters names *theme*, *source*, and *dest* specifying the destination of the theme to be poured out of the source container. The formal parameters are typed: the theme is some substance, the source is a container object that contains the theme, and the destination is a location.

```
def-plan pour ( <theme> ) : (some stuff)
                <source> ) : (an object
                           (type container)
                           (contains <theme>))
                           (affordance (an action
                                         (type pick-up)
                                         (body-part (a body-part (type hand))))))

                <dest> ) : (a location))

begin
1. reach( <source>)) /* (includes grasp, pregrasp) */
2. lift( <source>)) (a location (above <destination>)))
3. tilt( <source>))
   until (amount (some stuff (at <destination>)))
           ≥ (amount (<theme>))
end
```

Fig. 10 A generalized motion plan for a pouring action

• if a container is filled and open, then hold it upright • if an object can break, then don't squeeze too hard • if the task context is pouring, then grasp close to the center of mass • if the task context is pouring, then don't grasp mouth	• choose motion parameters that are predicted to succeed • grasp an object such that you have good visual feedback • don't get too close to breakable objects
---	--

Fig. 11 Generalized knowledge for placing substances

The power and elegance of knowledge-enabled programming comes from facilitating the application and combination of relevant knowledge pieces for specific manipulation tasks. This enables robot agents to accomplish manipulation tasks in new contexts much more competently, flexibly, and robustly.⁵

Consider again the task of picking up an object. Instead of implementing the knowledge of how to pick up an object as a piece of code tailored for the respective context the knowledge of how to pick up objects is decomposed into a collection of generalized knowledge pieces as listed in Fig. 11. This way the knowledge pieces can be applied to different robots, objects to be picked up, the tasks for which the object are to be picked up, and the scenes in which the objects are to be picked up.

⁵ As [53] puts it: “Robots are already making large strides in their abilities, but as the generalisable knowledge representation problem is addressed, the growth of robot capabilities will begin in earnest, and it will likely be explosive. The effects on economic output and human workers are certain to be profound.”

Generalized plans are typically invoked with vague action descriptions such as “pour the popcorn onto a plate”, which might be stated as a symbolic description as follows:

```
(perform
  (an action
    (type putting)
    (theme (some substance
      (type popcorn))))
    (destination (a location
      (on (an object (type plate))))))
```

The action description is vague or underdetermined because many pouring behaviors are possible executions of the action description. In Fig. 1 we have already seen two of them: the one-handed pouring of the popcorn from the bowl into the pot (upper row right) and the pouring of the popcorn from the pot onto the plate. Thus, in order to transform the underdetermined action description into a motion specification that can be executed by the generalized motion plan, the robot agent has to make a number of reasoned decisions based on current context possibly at planning or execution time. Examples of such decisions are which popcorn is meant and how much? The robot also has to choose a plate that is unused, clean, and has enough capacity.

In order to accomplish underdetermined action descriptions the robot agents have to be equipped with inference mechanisms that enable the robot agents to answer the body motion query:

```
how do I have to move my body
  in order to
    accomplish the given action description
    for the current task
    with the objects and in the context
    that I see or believe
```

When getting the instruction to pour popcorn onto the plate the robot has to first of all infer that it has to manipulate the container that contains the popcorn, rather than the popcorn itself as stated in the instruction. This requires the application of commonsense knowledge about the use of containers [55]. The robot also has to disambiguate “putting” into a pouring action, which again requires commonsense or the experience that pouring is more adequate than spooning with a scoop.

```
(perform
  (an action
    (type pouring)
    (object-acted-on
      (a container
        (contains (some substance
          (type popcorn))))))
```

```
(destination
  (destination (a location
    (on (an object (type plate))))))
```

After the original instruction is transformed into a description of the action to be performed, the robot has to decide on the body movements that it intends to realize the action with. To do so, the robot has to infer the key movement constraints that ensure that the pot will not fall down, the popcorn on the plate and not spilled on the table including to hold the pot with two hands by its handles and horizontally, to tilt the pot holding it in the right height and centered above the plate, and so on.

```
(perform (an action
  (type pouring)
  (object-acted-on (an object ...))    (destination (a location ...))
  (grasp-type ...)    (grasp-points ...)    (reaching-trajectory ...)
  ... )
```

An alternative is to specify an action in terms of its effects rather than in terms of its movement constraints and objectives by stating that the robot is to pour pancake mix onto a pancake maker such that the pancake mix forms a circle of a certain size as shown below. The advantage of such an action specification is that it gives more flexibility to the robot agent on how to perform the pouring. This flexibility has to be paid for with the requirement of broader and deeper understanding of intuitive physics and the ability to regress effects into the motions that can achieve them.

```
(perform (an action
  (type pouring)
  (theme (a container
    (contains (some substance (type pancake-mix))))))
  (destination (a location
    (on (the object (type pancake-maker))))))
  (results-in (an object
    (type pancake)
    (shape circular)
    (size 12cm))) )
```

5.2 Reasoning for Object Manipulation

We have just seen the kinds of refinements of action descriptions that make the context-specific execution of underdetermined action descriptions possible. So, let us now look at the knowledge representation and reasoning that can inform these refinements. The situation calculus representation we have discussed in Sect. 5.1.1 cannot provide the necessary reasoning because of its use of action models that assert that actions cause the effects specified in the models if the preconditions of

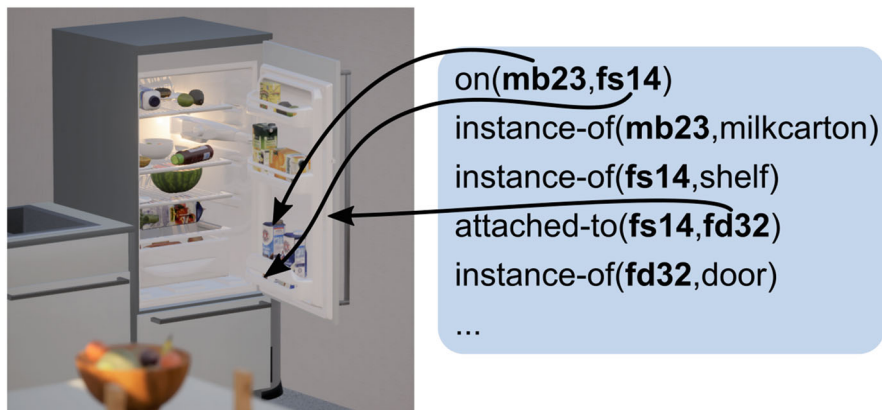


Fig. 12 Typical representation of a scene inside a refrigerator using predicate logic

the actions are satisfied, which is only valid if the effects are independent of *how* actions are executed.

When robots execute actions the physical effects of the actions depend on how the actions are executed and what the specific action context is. In recent years more detailed representations of objects, environments, robots, and tasks have gained a lot of momentum. Reasons are advances in representation learning in deep networks, which allow the automated learning of task-specific representations, and the ones in simulating and rendering virtual reality environments for computer games and other applications.

In this section we will look at the combination of logic representations with the ones used by simulation and rendering engines to enable robot agents to reason about how to move their bodies to accomplish underdetermined action descriptions. To do so, we will look at the problem of picking up a milk box from the refrigerator.

In predicate logic we can represent state of affairs such as entity *mb23* is a milkbox located on a shelf that is attached to the door of the fridge. This state of affairs is depicted in Fig. 12. If we want to represent the scene in more detail including the structure, the geometry, the appearance, the content, the physical properties, and the articulation of the objects in the environment the knowledge base becomes huge and difficult to reason with.

Representations with the necessary levels of detail have been successfully developed in modern virtual reality technology, in which huge realistic environments are represented very comprehensively and very detailed and physically simulated and rendered in real time and with currently available computing power. For example, already in 2015 the Unreal game engine⁶ showed the technology demonstration “a boy and his kite”, which featured realistic physics-based rendering of a 100

⁶ <https://www.unrealengine.com>.

square miles environment with 15 million pieces of vegetation, realistically moving animals, and physics-enabled animation of a kite flying in changing wind conditions in real time at 30fps.

Virtual reality systems represent environments as scene graphs, that are specifically designed data structures that contain all the information needed to render images and animations of 3D scenes. The scene graph is a tree-like data structure where the nodes represent subscenes and the child relations from a node p to the nodes $c_1, \dots, c_n$ state that the scene p contains all subscenes $c_1, \dots, c_n$. The leaf nodes, called shape nodes, represent the geometry, appearance, lighting, texture, surface attributes, and material of the respective physical entity. The bodies of the entities are typically CAD models or meshes. The inner nodes hold a group or composition of children nodes that might be constrained by physical relations such as attachment, support, and articulation. Also children nodes might represent the subscene at different levels of detail based on distance to viewer. Or, they might represent different states of a subscene like a container being open or closed. For rendering purposes the inner nodes might also contain light sources and local cameras. The inner nodes also include transformations that move, rotate, and scale the subscenes within its own reference frame.

Scene graphs are highly optimized and facilitated by very fast and efficient algorithms for storing, managing very complex and detailed scenes, for retrieving information from them, for visual rendering them and simulating the physical effects of actions performed in the scenes.

Using scene graphs, we can realize detailed and multi resolution representations by linking symbolic entity names to the nodes in the scene graphs they represent, as it is shown in Fig. 13. This way the KR&R system can make the detailed information available that is needed for the contextualization such as the geometry of the milk box, its parts, and its visual appearance. In the scene graph the milkbox is implemented as a subtree that contains all the data and computation mechanisms needed to realistically render and simulate the milkbox. This level of detail is sufficient for inferring body motions.

Using the scene graphs, we can realize virtual knowledge bases using symbolic predicates that compute their truth values through procedural attachments. As shown for example in Fig. 14, we can define a predicate *expected-camera-image*(*Img*), which is true for the image *Img* that the robot expects to see given its belief state. When the KR&R reasoner tests the truth value of the statement *expected-camera-image*(*Img*) it represents the beliefs of the robot as a scene graph, creates a virtual camera that matches the technical specification of its own camera, places the camera according its pose estimates, renders an image, and binds the image to the symbolic variable *Img*. A predicate *image-segment*(*Obj*, *Segs*, *Img*) relates each object *Obj* in an image *Img* with the image segments *Segs* depicting it by asserting that all other objects are invisible. Thus, using the detailed representations of scene graphs and the rendering and simulation capabilities of virtual reality engines the reasoning capabilities of KR&R frameworks can be substantially improved.

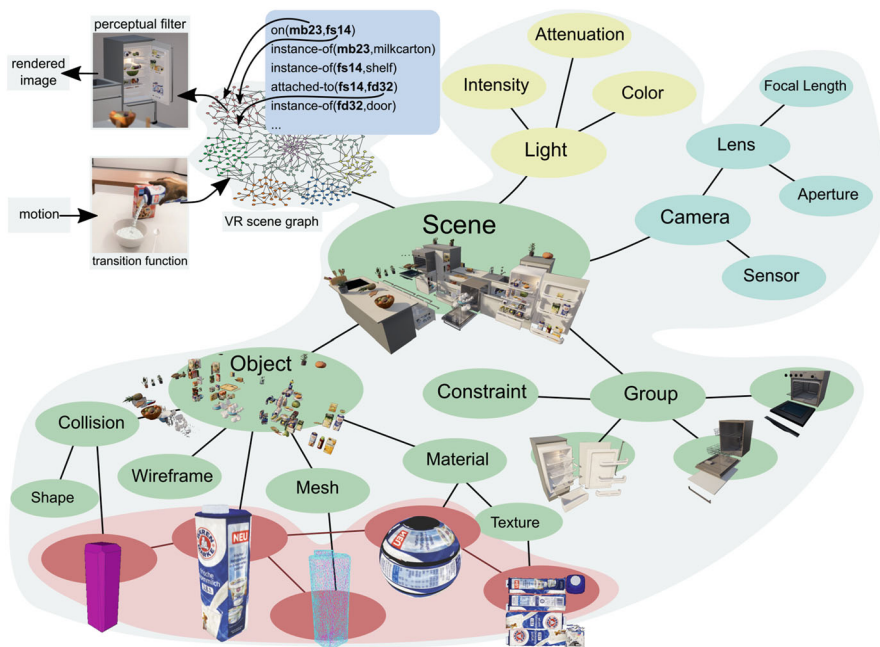


Fig. 13 Linking symbolic names of entities to nodes in scene graphs that represent these entities. The subtrees of the nodes contain the detailed representations of entities that are needed to infer context-specific parameterizations of actions

6 Conclusion

Knowledge representation and reasoning provides both, huge opportunities for the investigation of intelligent robots as well as huge challenges for KR&R research. The opportunities are well stated by Pratt [53]: “Robots are already making large strides in their abilities, but as the generalizable knowledge representation problem is addressed, the growth of robot capabilities will begin in earnest, and it will likely be explosive.” The challenges become obvious if we look at research in the field of action science. The brain has orchestrated very diverse, sophisticated, and powerful reasoning techniques to achieve the cognitive capability to accomplish complex goal-directed object manipulation capabilities. The representation and reasoning capabilities include simulation-based [56], probabilistic [57], imagistic [58] reasoning, and prospection [59]. In addition, other information processing mechanisms including situation models that enable flexible, context-specific, goal-directed behavior as well as memory mechanisms [60] are not understood well enough to build informative models of their operation. Other key research questions that deserve intensive research activities are how to ground reasoning techniques into the perception and action mechanisms of robot agents and how to combine

1. Write the predicate `robotPart(Robot, Part, PartType)` where *Part* is reachable by *Robot* through the transitive closure of the parthood relationship `dul:'hasPart'`, and *PartType* is a type of *Part*, and a subclass of `dul:'PhysicalObject'`. Tip: The type of an object can be obtained by `rdf:type`, and its subclasses by `rdfs:subClassOf`.
2. Write a query that answers the following questions: (1) What are the reference frames associated to the base link of arms of the PR2 robot, and (2) how are the arms positioned relative to the base of the robot? Tips: The PR2 robot is identified by `pr2:'PR2_0'`, the type of arm components is `pr2:'PR2Arm'`, and their base link reference frame can be obtained through their property `urdf:hasBaseLinkName`. The base of the robot is of type `pr2:'PR2Base'`. The position of an object *O* can be obtained via the 2-ary predicate `tf:tf_get_pose`, written as `tf:tf_get_pose(O, [F, Position, _])`, where the position of *O* is expressed relative to the reference frame *F*.

Exercise 2: Episode Structure

In this exercise, you will write queries for a recorded robotic NEEM. Under the following link you can find the website where you can test your queries on the NEEM: <https://data.open-ease.org/2021-robot-mooc-ex2>

The link sends you to a real query interface for a recorded episode. With the *Examples* button you can try different queries on the episode.

Write the following queries:

1. Write a query that yields the shortest action in which a transporting task was executed. Tip: You can use `event_interval(Action, Start, End)`, where *Action* is the given action, and *Start* and *End* the start- and endpoint of the action. Use `number(End)` to ensure that *Start* and *End* is bound to a rational number. To obtain the executed task you can use the property `dul:executesTask`. The type of the task is `soma:'Transporting'`.
2. Write a query that yields the type of tasks that are executed in actions in which an agent manipulates an item. Tip: Being an item in an action is the role an object plays when a task is executed, the roles of a task are denoted by the `dul:isTaskOf` property. The type of the role of manipulated items is `soma:'Item'`, and, for agents in a task, the role type is `soma:'AgentRole'`.

Solutions

Solution 1: Robot Structure

```

1. robotPart(Robot, Part, PartType) :-
    triple(Robot, dul:'hasPart', Component),
    (
        robotPart1(Component, Part, PartType)
    ;
        robotPart(Component, Part, PartType)
    ).

robotPart1(Component, Component, PartType) :-
    triple(Component, rdf:type, PartType),
    triple(PartType, rdfs:subClassOf, dul:'PhysicalObject').

2. % Get the reference frame of the robot base
robotPart('http://knowrob.org/kb/PR2.owl#PR2_0', Base,
'http://knowrob.org/kb/PR2.owl#PR2Base'),
triple(Base, urdf:hasBaseLinkName, BaseFrame),
% Get the arm and the corresponding reference frame
robotPart('http://knowrob.org/kb/PR2.owl#PR2_0', Arm,
'http://knowrob.org/kb/PR2.owl#PR2Arm'),
triple(Arm, urdf:hasBaseLinkName, ArmFrame),
% Get the position of the arm relative to the baseframe
tf:tf_get_pose(Arm, [BaseFrame, Position, _]).

```

Solution 2: Episode Structure

```

1. % Get all events and their durations
findall([Duration, Evt],
    (
        event_interval(Evt, Begin, End),
        % Ensure that the Action executes a Transporting task
        triple(Evt, dul:'executesTask', Tsk),
        triple(Tsk, rdf:type, soma:'Transporting'),
        % Ensure that the end is bound to a number
        number(End),
        % Calculate the duration
        Duration is End - Begin
    ),
    Durations),
% Get the minimal member of the solutions
min_member([MinDuration, ShortestEvent], Durations).

2. % Get the executed Task
triple(Act, dul:'executesTask', Tsk),
% Ensure that there is a agent

```

```
% and item during the Action
triple(Tsk, dul:isTaskOf, ObjectRole),
triple(ObjectRole, rdf:type, soma:'Item'),
triple(Tsk, dul:isTaskOf, AgentRole),
triple(AgentRole, rdf:type, soma:'AgentRole'),
% Return the type of the task
triple(Tsk, rdf:type, TskType).
```

Cross-References

This chapter contains cross-references to content published in other chapters belonging to the four “Robotic Goes MOOC” books: KNO6, INT5, INT7, INT8, IMP7. The full list of chapter abbreviations is available in the Preface.

References

1. S.J. Russell, P. Norvig, *Artificial Intelligence — A Modern Approach*, 3 int. ed. (Pearson Education, London, 2010)
2. S. Edelkamp, S. Schrödl, *Heuristic Search - Theory and Applications* (Academic Press, Cambridge, 2012). ISBN 978-0-12-372512-7
3. S. Shapiro, Knowledge representation and reasoning logics for artificial intelligence (2009)
4. J. McCarthy, Programs with common sense, in *Semantic Information Processing* (MIT Press, Cambridge, 1968), pp. 403–418
5. R.J. Brachman, H.J. Levesque (eds.), *Readings in Knowledge Representation* (Morgan Kaufman, Los Altos, 1985)
6. S.A. Cook, The complexity of theorem-proving procedures, in *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (ACM, New York, 1971), pp. 151–158
7. A.M. Turing, On computable numbers, with an application to the entscheidungsproblem. Proc. Lond. Math. Soc. **42**, 230–265 (1936)
8. A. Church, A note on the entscheidungsproblem. J. Symb. Log. **1**(1), 40–41 1936
9. A. Church, Correction to a note on the entscheidungsproblem. J. Symb. Log. **1**(3), 101–102 (1936)
10. R.O. Fikes, N.J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving. Technical Report 43R, AI Center, SRI International (1971). <http://www.ai.sri.com/shakey/>
11. M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL—the planning domain definition language. AIPS-98 Planning Committee (1998)
12. R.A. Kowalski, Algorithm = logic + control. Commun. ACM **22**(7), 424–436 (1979). <http://dblp.uni-trier.de/db/journals/cacm/cacm22.html#Kowalski79>
13. R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems* (MIT Press, Cambridge, 2001)
14. M. Genesereth, N. Nilsson, *Logical Foundations of Artificial Intelligence* (Morgan Kaufmann, San Mateo, 1987)
15. E. Davis, *Representations of Commonsense Knowledge* (Morgan Kaufmann, San Mateo, 1990)
16. E.T. Mueller, *Commonsense Reasoning* (Morgan Kaufmann, San Mateo, 2006). ISBN 0-12-369388-8

17. J. McCarthy, P.J. Hayes, Some philosophical problems from the standpoint of artificial intelligence. *Mach. Intell.* **4**, (1969). <http://www-formal.stanford.edu/jmc/>
18. P.J. Hayes, In defense of logic, in *International Joint Conference on Artificial Intelligence* (1977), pp. 559–565. <http://dli.iit.ac.in/ijcai/IJCAI-77-VOL1/PDF/099.pdf>
19. P. Hayes, The naive physics manifesto, in *Expert Systems in the Micro Electronic Age*, ed. by D. Michie (Edinburgh University Press, Edinburgh, 1979), pp. 242–270
20. P. Hayes, The second naive physics manifesto, in *Formal Theories of the Commonsense World*, ed. by J.R. Hobbs, R.C. Moore (Ablex, Norwood, 1985), pp. 1–36
21. M. Genesereth, Knowledge interchange format. Proposed draft (1998). <http://logic.stanford.edu/kif/dpans.html>
22. C. Baral, M. Gelfond, Reasoning about effects of concurrent actions. *J. Log. Program.* **31**, 85–117 (1997)
23. D. McDermott, A temporal logic for reasoning about processes and plans. *Cognit. Sci.* **6**(2), 101–155 (1982). ISSN 0364-0213
24. J.F. Allen, Maintaining knowledge about temporal intervals. *Commun. ACM* **26**(11), 832–843 (1983). ISSN 0001-0782
25. D. McDermott, Reasoning about plans, in *Formal Theories of the Commonsense World*, ed. by J.R. Hobbs, R.C. Moore (Ablex, Norwood, 1985), pp. 269–317
26. T. Dean, M. Wellmann, *Planning and Control* (Morgan Kaufmann, San Mateo, 1991)
27. R. Dechter, *Constraint Processing*. The Morgan Kaufmann Series in Artificial Intelligence (Morgan Kaufmann, San Francisco, 2003). ISBN 978-1-55860-890-0. <http://www.sciencedirect.com/science/book/9781558608900>
28. R. Fagin, J.Y. Halpern, N. Megiddo, A logic for reasoning about probabilities. *Inf. Comput.* **87**(1–2), 78–128 (1990). ISSN 0890-5401. [http://dx.doi.org/10.1016/0890-5401\(90\)90060-U](http://dx.doi.org/10.1016/0890-5401(90)90060-U). <http://www.sciencedirect.com/science/article/pii/089054019090060U>. Special Issue: Selections from 1988 {IEEE} Symposium on Logic in Computer Science
29. J. McCarthy, Circumscription—A form of non-monotonic reasoning. *Artif. Intell.* **13**(1–2), 27–39 (1980)
30. J. McCarthy, Applications of circumscription to formalizing common-sense knowledge. *Artif. Intell.* **28**, 89–116 (1986)
31. S. Hanks, Practical temporal projection, in *Proceedings of AAAI-90* (1990), pp. 158–163
32. M. Beetz, H. Grosskreutz, Probabilistic hybrid action models for predicting concurrent percept-driven robot behavior. *J. Artif. Intell. Res.* **24**, 799–849 (2005)
33. C. Boutilier, T. Dean, S. Hanks, Decision theoretic planning: structural assumptions and computational leverage. *J. AI Res.* **11**, 1–94 (1998)
34. P. Haddawy, L. Rendell, Planning and decision theory. *Knowl. Eng. Rev.* **5**, 15–33 (1990)
35. P. Haddawy, S. Hanks, Utility models for goal-directed decision-theoretic planners. *Comput. Intell.* **14**, 392–429 (1993)
36. L.P. Kaelbling, M.L. Littman, A.R. Cassandra, Planning and acting in partially observable stochastic domains. *Artif. Intell.* **101**(1–2), 99–134 (1998). ISSN 0004-3702
37. R.C. Moore, A formal theory of knowledge and action, in *Formal Theories of the Commonsense World*, ed. by J.R. Hobbs, R.C. Moore (Ablex, Norwood, 1985), pp. 269–317
38. L. Morgenstern, Knowledge preconditions for actions and plans, in *International Joint Conference on Artificial Intelligence* (1987), pp. 867–874
39. M. Bratman, D. Israel, M. Pollack, Plans and resource-bounded practical reasoning. *Comput. Intell.* **4**(4), 349–355 (1988)
40. A.S. Rao, M.P. Georgeff, *Modeling Rational Agents Within a BDI-Architecture* (Morgan Kaufmann, San Francisco, 1991). <http://portal.acm.org/citation.cfm?id=284860.284916>
41. D. McDermott, A critique of pure reason. *Comput. Intell.* **3**, 151–60 (1987)
42. S. Harnad, The symbol grounding problem. *CoRR. cs.AI/9906002* (1999). <http://arxiv.org/abs/cs.AI/9906002>.
43. S. Hanks, D. McDermott, Nonmonotonic logic and temporal projection. *Artif. Intell.* **33**(3), 379–412 (1987)

44. D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A.A. Kalyanpur, A. Lally, J. William Murdock, E. Nyberg, J. Prager, N. Schlaefter, C. Welty, Building watson: an overview of the deepqa project. *AI Mag.* **31**(3), 59–79 (2010). <https://doi.org/10.1609/aimag.v31i3.2303>. <https://www.aaai.org/ojs/index.php/aimagazine/article/view/2303>
45. P. Hayes, Naïve physics i: Ontology for liquids, in *Formal Theories of the Commonsense World*, ed. by J.R. Hobbs, R.C. Moorepages (Ablex, Norwood, 1985), pp. 71–107
46. R. Miller, L. Morgenstern, Common sense problem page. <http://www-formal.stanford.edu/leora/commonsense/>
47. L. Morgenstern, Mid-sized axiomatizations of commonsense problems: a case study in egg cracking. *Stud. Log.* **67**(3), 333–384 (2001)
48. W. Prinz, M. Beisert, A. Herwig (eds.), *Action Science: Foundations of an Emerging Discipline* (The MIT Press, Cambridge, 2013)
49. J.R. Flanagan, M.C. Bowman, R.S. Johansson, Control strategies in object manipulation tasks. *Curr. Opin. Neurobiol.* (2006)
50. J.M. Siskind, Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic. *J. Artif. Intell. Res.* **15**, 31–90 (2001)
51. O. Khatib, A unified approach for motion and force control of robot manipulators: the operational space formulation. *IEEE J. Robot. Autom.* **3**, 43–53 (1987)
52. R. Smits, T.D. Laet, K. Claes, H. Bruyninckx, J.D. Schutter, iTASC: a tool for multi-sensor integration in robot manipulation, in *Multisensor Fusion and Integration for Intelligent Systems*, ed. by H.K.H. Hahn, S. Lee. *Lecture Notes in Electrical Engineering*, vol. 35 (Springer, Berlin, 2009), pp. 235–254
53. G.A. Pratt, Is a cambrian explosion coming for robotics? *J. Econ. Perspect.* **29**(3), 51–60 (2015)
54. G.L. Drescher, *Made-Up Minds: A Constructivist Approach to Artificial Intelligence* (MIT Press, Cambridge, 1991)
55. E. Davis, G. Marcus, N. Frazier-Logue, Commonsense reasoning about containers using radically incomplete information. *Artif. Intell.* **248**, 46–84 (2017). <https://doi.org/10.1016/j.artint.2017.03.004>
56. G. Hesslow, Conscious thought as simulation of behaviour and perception. *Trends Cognit. Sci.* **6**(6), 242–247 (2002). ISSN 13646613. [http://dx.doi.org/10.1016/s1364-6613\(02\)01913-7](http://dx.doi.org/10.1016/s1364-6613(02)01913-7)
57. T.L. Griffiths, C. Kemp, J.B. Tenenbaum, *The Cambridge Handbook of Computational Cognitive Modeling*. *Bayesian Models of Cognition* (Cambridge University Press, Cambridge, 2008)
58. S.M. Kosslyn, Mental images and the brain. *Cognit. Neuropsychol.* **22**, 333–347 (2005)
59. D. Vernon, M. Beetz, G. Sandini, Prospection in cognition: The case for joint episodic-procedural memory in cognitive robotics. *Front. Robot. AI* **2**(19), 2015. ISSN 2296-9144. <https://doi.org/10.3389/frobt.2015.00019>. http://www.frontiersin.org/humanoid_robotics/10.3389/frobt.2015.00019/abstract
60. E. Tulving, Episodic and semantic memory 1. *Org. Memory Lond. Acad.* **381**(e402), 4 (1972)
61. E.T. Mueller, *Commonsense Reasoning* (Morgan Kaufmann, San Mateo, 2006). ISBN 0-12-369388-8

An Updated Tutorial on Graph-Based SLAM



Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard

Abstract Simultaneous localization and mapping (SLAM) consists in the concurrent construction of a model of the environment (the map), and the estimation of the state of the robot moving within it. The SLAM community has made astonishing progress over the last 30 years, enabling large-scale real-world applications and witnessing a steady transition of this technology to industry. We survey the current state of SLAM and consider future directions. We start by presenting what is now the de-facto standard formulation for SLAM. We then review related work, covering a broad set of topics including robustness and scalability in long-term mapping, metric and semantic representations for mapping, theoretical performance guarantees, active SLAM and exploration, and other new frontiers. This paper simultaneously serves as a position paper and tutorial to those who are users of SLAM. By looking at the published research with a critical eye, we delineate open challenges and new research issues, that still deserve careful scientific investigation. The paper also contains the authors' take on two questions that often animate discussions during robotics conferences: Do robots need SLAM? and Is SLAM solved?

©2010 IEEE. Reprinted, with permission, from G. Grisetti, R. Kümmerle, C. Stachniss and W. Burgard. A tutorial on graph-based SLAM, IEEE Intelligent Transportation Systems Magazine 2(4):31–43, 2010, revised version.

G. Grisetti (✉)
Sapienza University of Rome, Rome, Italy

R. Kümmerle
Magazino GmbH, München, Germany

C. Stachniss
University of Bonn, Bonn, Germany

W. Burgard
University of Freiburg, Freiburg im Breisgau, Germany

1 Introduction

To efficiently solve tasks such as transportation, search and rescue, or automated vacuum cleaning, robots need a map of their environment. The availability of an accurate map allows for the design of systems that can operate in complex environments only based on their onboard sensors and without relying on external reference systems like, e.g., GNSS. The acquisition of maps of indoor environments, where typically no GNSS information is available, has been a major research focus in the robotics community over the last decades. Learning maps under pose uncertainty is often referred to as the simultaneous localization and mapping (SLAM) problem. In the literature, a large variety of solutions to this problem is available. These approaches can be classified either as filtering or smoothing. Filtering approaches model the problem as an online state estimation task where the state of the system consists in the *current* robot position and the map. The estimate is augmented and refined by incorporating new measurements as they become available. Popular techniques like Kalman and information filters [14, 70], particle filters [29, 35, 53], or information filters [25, 75] fall into this category. To highlight their incremental nature, filtering approaches are usually referred to as online SLAM methods. Conversely, smoothing approaches estimate the full trajectory of the robot from the full set of measurements [22, 51, 62]. These approaches address the so-called full SLAM problem, and they typically rely on least-squares error minimization techniques.

Figure 1 shows four types of robots using SLAM technology: autonomous cars, industrial mobile manipulation robots, agricultural robots, and exploration robots. Image (a) shows an instrumented car that uses maps to localize itself in traffic. Image (b) depicts a mobile manipulation robot that uses accurate position estimates and does not require induction lines on factory floors. The robot in image (c) uses SLAM technology to accurately estimate the positions of the plants on the field in a precision agriculture setting. Image (d) shows a robot that uses SLAM technology to

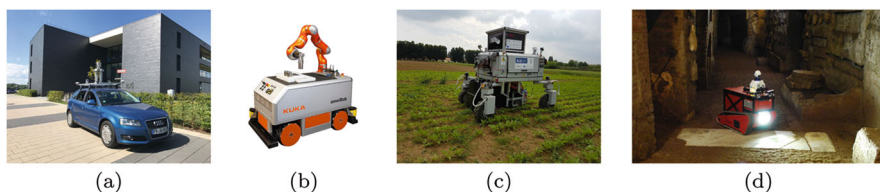


Fig. 1 Applications of SLAM technology: (a) Self-driving cars can benefit greatly from highly accurate maps created with the SLAM approaches described here as it allows them to always stay on the road. (b) Robots in logistic applications or performing mobile manipulation tasks like the KUKA robot “Omnirob” require SLAM methods for highly precise indoor navigation. (c) Also, agricultural robots require the capability of highly precise outdoor navigation, which can be achieved with the SLAM methods described here. (d) Autonomous exploration robots such as the Rovina platform needs SLAM techniques to efficiently explore abandoned underground structures and autonomously find their way out afterward

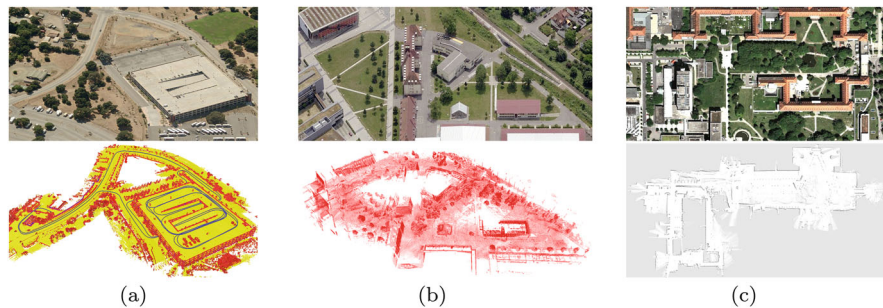


Fig. 2 (a) A 3D map of the Stanford parking garage acquired with an instrumented car (bottom), and the corresponding satellite view (top). This map has been subsequently used to realize an autonomous parking behavior. (b) Point cloud map acquired at the university of Freiburg (courtesy of Kai. M. Wurm) and relative satellite image. (c) Occupancy grid map acquired at the hospital of Freiburg. Top: a bird's eye view of the area, bottom: the occupancy grid representation. The gray areas represent unobserved regions, the white part represents traversable space while the black points indicate occupied regions

autonomously explore catacombs, which humans are not allowed to enter. Figure 2 illustrates 2D and 3D maps estimated by the SLAM algorithm discussed in this paper.

An intuitive way to address the SLAM problem is via its so-called graph-based formulation. Solving a graph-based SLAM problem involves constructing a graph whose nodes represent robot poses or landmarks. An edge between two nodes encodes a sensor measurement that constrains the connected poses. Such constraints can be contradictory since observations are typically affected by noise. Once the robot has constructed such a graph, the crucial problem is to find a configuration of the nodes that is maximally consistent with the measurements. This involves solving a potentially large error minimization problem.

The graph-based formulation of the SLAM problem has been proposed in the robotics community by Lu and Milios in 1997 [51], although mapping based on the least-squares approach has a 200-year-old history [3]. It took several years to make this formulation popular in robotics due to the comparably high complexity of solving the error minimization problem, potentially online, using standard techniques. Recent insights into the structure of the SLAM problem and advancements in the fields of sparse linear algebra resulted in efficient approaches to the optimization problem at hand. Consequently, graph-based SLAM methods have undergone a renaissance and currently belong to state-of-the-art techniques concerning speed and accuracy. This tutorial aims to introduce the SLAM problem in its probabilistic form and guide the reader to the synthesis of an effective and state-of-the-art graph-based SLAM method. To follow this tutorial, basic knowledge of linear algebra, multivariate minimization, and probability theory is required.

2 Probabilistic Formulation of SLAM

Solving the SLAM problem consists of estimating the robot trajectory and the map of the environment as the robot moves in it. Due to the inherent noise in the sensor measurements, a SLAM problem is usually described by means of probabilistic tools. The robot is assumed to move in an unknown environment, along a trajectory described by the sequence of random variables $\mathbf{x}_{1:T} = \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$. While moving, it acquires a sequence of odometry measurements $\mathbf{u}_{1:T} = \{\mathbf{u}_1, \dots, \mathbf{u}_T\}$ and perceptions of the environment $\mathbf{z}_{1:T} = \{\mathbf{z}_1, \dots, \mathbf{z}_T\}$. Solving the full SLAM problem consists of estimating the posterior probability of the robot's trajectory $\mathbf{x}_{1:T}$ and the map $\mathbf{m}$ of the environment given all the measurements plus an initial position $\mathbf{x}_0$:

$$p(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{z}_{1:T}, \mathbf{u}_{1:T}, \mathbf{x}_0). \quad (1)$$

The initial position $\mathbf{x}_0$ defines the position of the map and can be chosen arbitrarily. For the convenience of notation, in the remainder of this document, we will omit $\mathbf{x}_0$. The poses $\mathbf{x}_{1:T}$ and the odometry $\mathbf{u}_{1:T}$ are usually represented as 2D or 3D transformations in $SE(2)$ or in $SE(3)$, while the map can be represented in different ways. Maps can be parametrized as a set of spatially located landmarks, by dense representations like occupancy grids, by surface maps, or by raw sensor measurements. The choice of a particular map representation depends on the sensors used, the characteristics of the environment, and the estimation algorithm. Landmark maps [53, 70] are often preferred for environments where locally distinguishable features can be identified and when (monocular) cameras are used. In contrast, dense representations [29, 35, 76] are usually used in conjunction with range sensors. Figure 2 illustrates three typical dense map representations for 3D and 2D: multilevel surface maps, point clouds, and occupancy grids. Figure 3 shows a typical 2D landmark-based map.

Estimating the posterior given in (1) involves operating in high-dimensional state spaces. This high dimensionality would result in intractable solutions if the SLAM problem did not have a well-defined structure. This structure arises from certain and commonly done assumptions, namely the static world assumption and the Markov assumption. A convenient way to describe this structure is via the dynamic Bayesian network (DBN) depicted in Fig. 4. A Bayesian network is a graphical model that describes a stochastic process as a directed graph. The graph has one node for each random variable in the process, and a directed edge (or arrow) between two nodes models a conditional dependence between them.

In Fig. 4, one can distinguish blue/gray nodes indicating the observed variables (here $\mathbf{z}_{1:T}$ and $\mathbf{u}_{1:T}$) and white nodes which are the hidden variables. The hidden variables $\mathbf{x}_{1:T}$ and $\mathbf{m}$ model the robot's trajectory and the map of the environment. The connectivity of the DBN follows a recurrent pattern characterized by the state transition model and by the observation model. The transition model $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$ is represented by the two edges leading to $\mathbf{x}_t$ and represents the probability

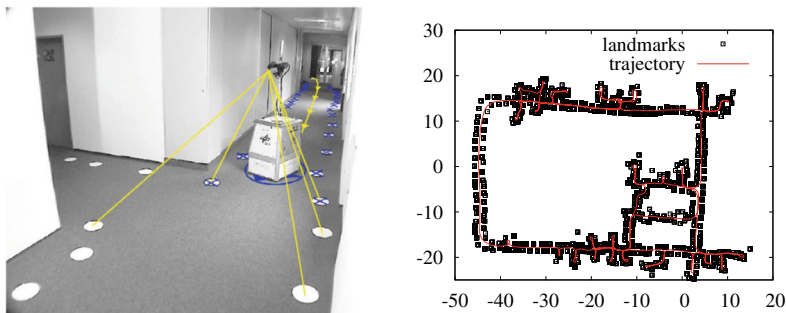


Fig. 3 Landmark based maps acquired at the German Aerospace Center. In this setup the landmarks consist in white circles painted on the ground that are detected by the robot through vision, as shown in the left image. The right image illustrates the trajectory of the robot and the estimated positions of the landmarks. These images are courtesy of Udo Frese and Christoph Hertzberg

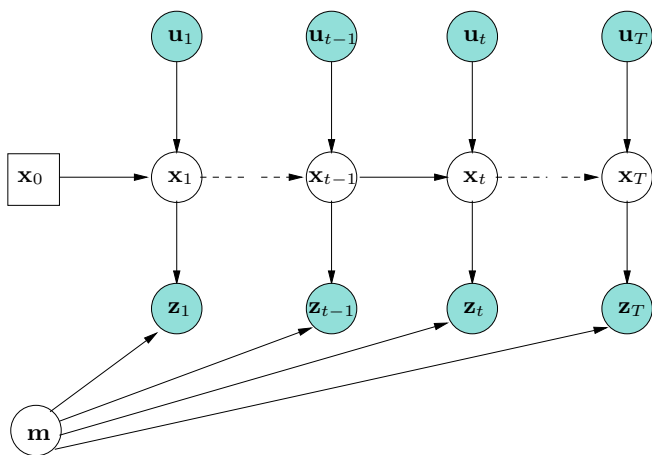


Fig. 4 Dynamic Bayesian Network of the SLAM process

that the robot at time t is in $\mathbf{x}_t$ given that at time $t - 1$ it was in $\mathbf{x}_t$ and it acquired an odometry measurement $\mathbf{u}_t$.

The observation model $p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}_t)$ represents the likelihood of obtaining the observation $\mathbf{z}_t$ given that the robot is at location $\mathbf{x}_t$ in the map $\mathbf{m}$. The arrows in Fig. 4 entering $\mathbf{z}_t$, represent the observation model. The exteroceptive observation $\mathbf{z}_t$ depends only on the current location $\mathbf{x}_t$ of the robot and on the (static) map $\mathbf{m}$. Expressing SLAM as a DBN highlights its temporal structure, and therefore this formalism is well suited to describe filtering processes that can be used to tackle the SLAM problem.

An alternative representation to the DBN is via the so-called “graph-based” or “network-based” formulation of the SLAM problem, which highlights the

underlying spatial structure. In graph-based SLAM, the nodes in the graph represent the poses of the robot [45, 51]. Spatial constraints between poses that result from observations z_t or from odometry measurements $\mathbf{u}_t$ are encoded in the edges between the nodes. A pose graph can be seen as a special case of a factor graph, where the variables are the nodes of the graph, and the factors encode the likelihood of a relative measurement between the nodes in the graph's edges.

Graph-based SLAM algorithms construct such a graph out of the raw sensor measurements. Each node in the graph represents a robot position and a measurement acquired at that position. An edge between two nodes represents a spatial constraint relating the two robot poses. A constraint consists of a probability distribution over the relative transformations between the two poses. These transformations are either odometry measurements between sequential robot positions or are determined by aligning the observations acquired at the two robot locations. Once the graph is constructed, one seeks to find the configuration of the nodes, i.e., robot poses, that best satisfies the constraints. Thus, in graph-based SLAM the problem is decoupled into two tasks: constructing the graph from the raw measurements (graph construction) and determining the most likely configuration of the poses given the edges of the graph (graph optimization). The graph construction is usually called front-end, and it depends heavily on the used sensors. The second part is called back-end and relies on an abstract representation of the data in the form of the SLAM graph and is sensor-agnostic. A short example of a front-end for 2D laser SLAM is described in Sect. 4. In this tutorial, we will describe an easy-to-implement but efficient back-end for graph-based SLAM. Figure 5 depicts an uncorrected pose-graph and the corresponding corrected one.

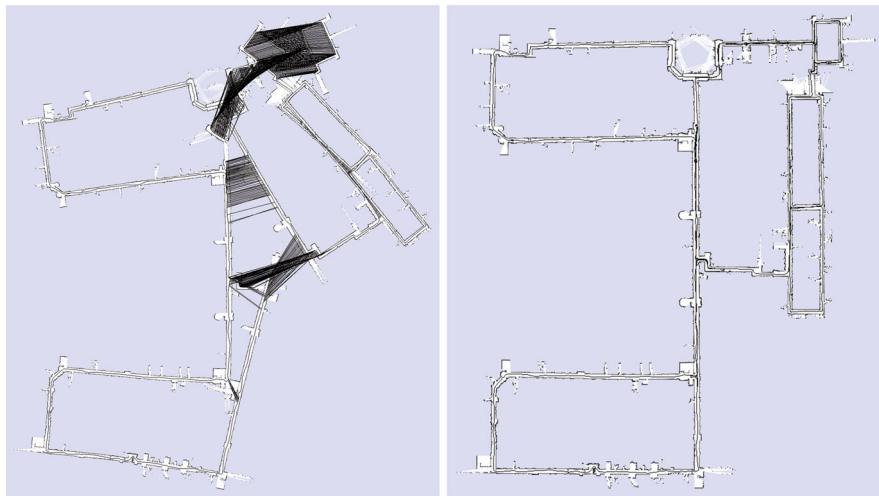


Fig. 5 Pose-graph corresponding to a data-set recorded at MIT Killian Court (courtesy of Mike Bosse and John Leonard) (left) and after (right) optimization. The maps are obtained by rendering the laser scans according to the robot positions in the graph

3 Graph-Based SLAM

A graph-based SLAM approach constructs a more abstract estimation problem by abstracting the raw sensor measurements. These raw measurements are replaced by the edges in the graph, which can then be seen as “virtual measurements”. More in detail, each edge between two nodes represents with a probability distribution over the relative locations of the two poses, conditioned to their mutual measurements. In general, the observation model $p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}_t)$ is multi-modal and therefore the Gaussian assumption does not hold. This means that a single observation $\mathbf{z}_t$ might result in multiple potential edges connecting different poses in the graph. Thus, the graph connectivity itself might need to be described as a probability distribution. Directly dealing with this multi-modality in the estimation process leads to a combinatorial explosion of the complexity. As a result, most practical approaches restrict the estimate to the most likely topology. Thus, one needs to determine the most likely constraint resulting from an observation. This decision depends on the probability distribution over the robot poses. This problem is known as data association and is usually addressed by the SLAM front-end. To compute the correct data-association, a front-end usually requires a consistent estimate of the conditional prior over the robot trajectory $p(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:T}, \mathbf{u}_{1:T})$. This requires interleaving the execution of the front-end and of the back-end while the robot explores the environment. Therefore, the accuracy and the efficiency of the back-end is crucial when designing a solid SLAM system. In this tutorial, we will not describe sophisticated approaches to the data association problem. Such methods tackle association by means of spectral clustering [62], joint compatibility branch and bound [56], or backtracking [36]. We rather assume that the given front-end provides consistent estimates.

If the observations are affected by Gaussian noise and the data association is known, a graph-based mapping algorithm aims to compute a Gaussian approximation of the posterior over the robot trajectory. This involves computing the mean of this Gaussian as this node configuration maximizes the likelihood of the observations. Once the mean is known, the information matrix of the Gaussian can be obtained as explained in Sect. 3.2. In the following, we will characterize the task of finding this maximum as a constraint optimization problem. We will also introduce parts of the notation illustrated in Fig. 6.

Let $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_T)^T$ be a vector of parameters, where $\mathbf{x}_i$ describes the pose of node i . Let $\mathbf{z}_{ij}$, and $\mathbf{\Omega}_{ij}$ be respectively the mean and the information matrix of a virtual measurement between the node i and the node j . This virtual measurement is a transformation that makes the observations acquired from i maximally overlap with the observation acquired from j . Let $\hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ be the prediction of a virtual measurement given a configuration of the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. Usually, this prediction is the relative transformation between the two nodes. The log-likelihood l_{ij} of a measurement $\mathbf{z}_{ij}$ is therefore

$$l_{ij} \propto [\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]^T \mathbf{\Omega}_{ij} [\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]. \quad (2)$$

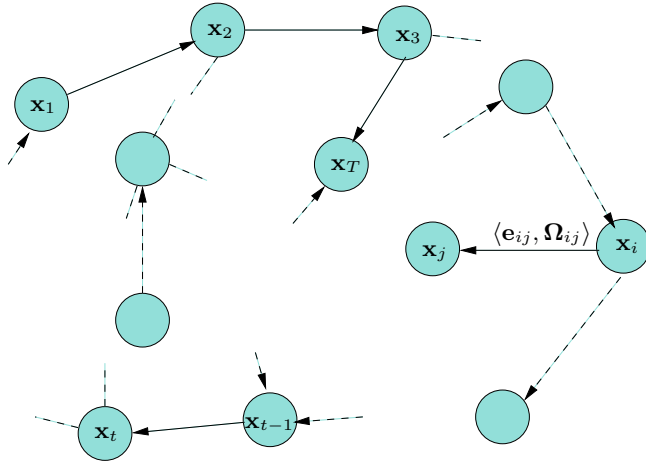


Fig. 6 A pose-graph representation of a SLAM process. Every node in the graph corresponds to a robot pose. Nearby poses are connected by edges that model spatial constraints between robot poses arising from measurements. Edges $\mathbf{e}_{t-1:t}$ between consecutive poses model odometry measurements, while the other edges represent spatial constraints arising from multiple observations of the same part of the environment

Let $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ be a function that computes a difference between the expected observation $\hat{\mathbf{z}}_{ij}$ and the real observation $\mathbf{z}_{ij}$ gathered by the robot. For simplicity of notation, we will encode the indices of the measurement in the indices of the error function

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j). \quad (3)$$

Figure 7 illustrates the functions and the quantities that play a role in defining an edge of the graph. Let $\mathcal{C}$ be the set of pairs of indices for which a constraint (observation) $\mathbf{z}$ exists. The goal of a maximum likelihood approach is to find the configuration of the nodes $\mathbf{x}^*$ that minimize the negative log likelihood $\mathbf{F}(\mathbf{x})$ over all the observations

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i, j \rangle \in \mathcal{C}} \underbrace{\mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}}_{\mathbf{F}_{ij}}, \quad (4)$$

thus, it seeks to solve the following equation:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \mathbf{F}(\mathbf{x}). \quad (5)$$

In the remainder of this section, we will describe an approach to solve Eq. (5) and to compute a Gaussian approximation of the posterior over the robot trajectory.

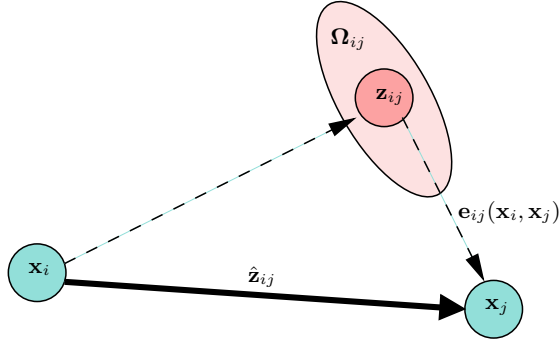


Fig. 7 Aspects of an edge connecting the vertex $\mathbf{x}_i$ and the vertex $\mathbf{x}_j$. This edge originates from the measurement $\mathbf{z}_{ij}$. From the relative position of the two nodes, it is possible to compute the expected measurement $\hat{\mathbf{z}}_{ij}$ that represents $\mathbf{x}_j$ seen in the frame of $\mathbf{x}_i$. The error $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ depends on the displacement between the expected and the real measurement. An edge is fully characterized by its error function $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ and by the information matrix $\mathbf{\Omega}_{ij}$ of the measurement that accounts for its uncertainty

Whereas the proposed approach utilizes standard optimization methods, like the Gauss-Newton or the Levenberg-Marquardt algorithms, it is efficient because it effectively exploits the structure of the problem. We first describe a direct implementation of traditional nonlinear least-squares optimization. Subsequently, we introduce a workaround that allows dealing with the singularities in the representation of the robot poses elegantly.

3.1 Error Minimization via Iterative Local Linearizations

If a good initial guess $\check{\mathbf{x}}$ of the robot's poses is known, the numerical solution of Eq. (5) can be obtained by using the popular Gauss-Newton or Levenberg-Marquardt algorithms. The idea is to approximate the error function by its first order Taylor expansion around the current initial guess $\check{\mathbf{x}}$

$$\mathbf{e}_{ij}(\check{\mathbf{x}}_i + \Delta \mathbf{x}_i, \check{\mathbf{x}}_j + \Delta \mathbf{x}_j) = \mathbf{e}_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x}) \quad (6)$$

$$\simeq \mathbf{e}_{ij} + \mathbf{J}_{ij} \Delta \mathbf{x}. \quad (7)$$

Here, $\mathbf{J}_{ij}$ is the Jacobian of $\mathbf{e}_{ij}(\mathbf{x})$ computed in $\check{\mathbf{x}}$ and $\mathbf{e}_{ij} = \mathbf{e}_{ij}(\check{\mathbf{x}})$. Substituting

Eq. (7) in the error terms $\mathbf{F}_{ij}$ of Eq. (4), we obtain:

$$\begin{aligned} \mathbf{F}_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x}) &= \mathbf{e}_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x})^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x}) \end{aligned} \quad (8)$$

$$\simeq (\mathbf{e}_{ij} + \mathbf{J}_{ij} \Delta \mathbf{x})^T \boldsymbol{\Omega}_{ij} (\mathbf{e}_{ij} + \mathbf{J}_{ij} \Delta \mathbf{x}) \quad (9)$$

$$= \underbrace{\mathbf{e}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}}_{\mathbf{c}_{ij}} + 2 \underbrace{\mathbf{e}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{J}_{ij}}_{\mathbf{b}_{ij}} \Delta \mathbf{x} + \Delta \mathbf{x}^T \underbrace{\mathbf{J}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{J}_{ij}}_{\mathbf{H}_{ij}} \Delta \mathbf{x} \quad (10)$$

$$= \mathbf{c}_{ij} + 2\mathbf{b}_{ij} \Delta \mathbf{x} + \Delta \mathbf{x}^T \mathbf{H}_{ij} \Delta \mathbf{x} \quad (11)$$

With this local approximation, we can rewrite the function $\mathbf{F}(\mathbf{x})$ in Eq. (4) as

$$\mathbf{F}(\check{\mathbf{x}} + \Delta \mathbf{x}) = \sum_{(i,j) \in \mathcal{C}} \mathbf{F}_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x}) \quad (12)$$

$$\simeq \sum_{(i,j) \in \mathcal{C}} \mathbf{c}_{ij} + 2\mathbf{b}_{ij} \Delta \mathbf{x} + \Delta \mathbf{x}^T \mathbf{H}_{ij} \Delta \mathbf{x} \quad (13)$$

$$= \mathbf{c} + 2\mathbf{b}^T \Delta \mathbf{x} + \Delta \mathbf{x}^T \mathbf{H} \Delta \mathbf{x}. \quad (14)$$

The quadratic form in Eq. (14) is obtained from Eq. (13) by setting $\mathbf{c} = \sum \mathbf{c}_{ij}$, $\mathbf{b} = \sum \mathbf{b}_{ij}$, and $\mathbf{H} = \sum \mathbf{H}_{ij}$. It can be minimized in $\Delta \mathbf{x}$ by solving the linear system

$$\mathbf{H} \Delta \mathbf{x}^* = -\mathbf{b}. \quad (15)$$

The matrix $\mathbf{H}$ is the information matrix of the system since it is obtained by projecting the measurement error in the space of the trajectories via the Jacobians. Alternatively, it can be seen as the approximated hessian of the cost function $\mathbf{F}(\mathbf{x})$. By construction, $\mathbf{H}$ is symmetric and semi-positive definite. Furthermore, it is also sparse in our case, having a number of non-zero off-diagonal blocks equal to twice the number of constraints.

The linearized solution is obtained by adding to the initial guess the computed increments

$$\mathbf{x}^* = \check{\mathbf{x}} + \Delta \mathbf{x}^*. \quad (16)$$

The popular Gauss-Newton algorithm iterates the linearization in Eq. (14), the solution in Eq. (15), and the update step in Eq. (16). In every iteration, the previous solution is used as the linearization point and the initial guess.

The procedure described above is a general approach to multivariate function minimization, here derived for the special case of the SLAM problem. However, the general approach assumes that the space of parameters $\mathbf{x}$ is Euclidean, which is not valid for SLAM and may lead to suboptimal solutions.

3.2 Considerations About the Structure of the Linearized System

According to Eq. (14), the matrix $\mathbf{H}$ and the vector $\mathbf{b}$ are obtained by summing up a set of matrices and vectors, one for every constraint. Every constraint will contribute to the system with an addend whose *structure* depends on the position in the state vector of the two connected nodes i and j , and whose *value* depends on the values of the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. Hence, the Jacobian $\mathbf{J}_{ij}$ has the following form:

$$\mathbf{J}_{ij} = \begin{pmatrix} \mathbf{0} \cdots \mathbf{0} & \underbrace{\mathbf{A}_{ij}}_{\text{node } i} & \mathbf{0} \cdots \mathbf{0} & \underbrace{\mathbf{B}_{ij}}_{\text{node } j} & \mathbf{0} \cdots \mathbf{0} \end{pmatrix}. \quad (17)$$

Here, $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ are the derivatives of the error function with respect to $\mathbf{x}_i$ and $\mathbf{x}_j$. From Eq. (10), we obtain the following structure for the block matrix $\mathbf{H}_{ij}$:

$$\mathbf{H}_{ij} = \begin{pmatrix} \ddots & & & & \\ & \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij} & \cdots & \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij} & \\ & \vdots & \ddots & \vdots & \\ & \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij} & \cdots & \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij} & \\ & & & & \ddots \end{pmatrix} \quad (18)$$

$$\mathbf{b}_{ij} = \begin{pmatrix} \vdots \\ \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij} \\ \vdots \\ \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij} \\ \vdots \end{pmatrix} \quad (19)$$

For simplicity of notation we omitted the zero blocks.

Algorithm 1 summarizes an iterative Gauss-Newton procedure to determine both the mean and the information matrix of the posterior over the robot poses. Since most of the structures in the system are sparse, we recommend using memory-efficient representations to store the Hessian $\mathbf{H}$ of the system. Since the structure of the Hessian is known in advance from the connectivity of the graph, we recommend to pre-allocate the Hessian once and then update it in place by looping over all edges whenever a new linearization is required. Each edge contributes to the blocks $\mathbf{H}_{[ii]}$, $\mathbf{H}_{[ij]}$, $\mathbf{H}_{[ji]}$, and $\mathbf{H}_{[jj]}$ and to the blocks $\mathbf{b}_{[i]}$ and $\mathbf{b}_{[j]}$ of the coefficient vector. An additional optimization is to compute only the upper triangular part of $\mathbf{H}$ since it is symmetric. Note that the error of a constraint $\mathbf{e}_{ij}$ depends only on the relative position of the connected poses $\mathbf{x}_i$ and $\mathbf{x}_j$. Accordingly, the error $\mathbf{F}(\mathbf{x})$ of a particular

configuration of the poses $\mathbf{x}$ is invariant under a rigid transformation of all the poses. This results in Eq. (15) being underdetermined. To solve this system numerically, it is common practice to constrain one of the increments $\Delta \mathbf{x}_k$ to be zero. This can be done by adding the identity matrix to the k th diagonal block $\mathbf{H}_{[kk]}$. Without loss of generality in Algorithm 1, we fix the first node $\mathbf{x}_1$. An alternative way to fix a particular node of the pose-graph consists in suppressing the k^{th} block row and the k^{th} block column of the linear system in Eq. (15).

In the batch case, the dominant operation of this algorithm is the solution of the linear system $\mathbf{H}\Delta \mathbf{x} = -\mathbf{b}$. In SLAM, the number of variables grows at most with the length of the trajectory. Since the sensor range is limited, each new observation will introduce measurements proportional to the density of the nodes in the map. In the case of repeated observations of the same area, this might result in a quadratic increase of the measurements. However, modern systems employ some graph pruning heuristics that marginalize out nodes too close to each other. Using this strategy bounds the number of variables to the size of the explored area instead of the length of trajectory, which is convenient for long acquisitions. Accordingly, the number of measurements, and thus of non-zero off-diagonal blocks of the $\mathbf{H}$ matrix, can be kept roughly proportional to the number of variables ensuring a sparse pattern. Exploiting these intrinsic properties, we can efficiently solve the linear system in Eq. (15).

The literature of sparse linear system solutions is quite rich and the methods proposed can be classified into two main groups: iterative and direct methods. The former computes an iterative solution to the linear system by following the gradient of the quadratic form. These techniques often use a *pre-conditioner*, i.e., preconditioned conjugate gradient (PCG), which improves convergence by manipulating the matrix to enhance the effect of each gradient step. Further information about these approaches can be found in work of Saad [66]. Iterative methods might require a quadratic time in computing the exact solution of a linear system. However, they can become the only option when the dimension of the problem is huge due to their limited memory requirements. Direct methods, instead, compute the solution (up to numerical errors) of the linear system without iterations, usually leveraging on some matrix decomposition followed by backsubstitution. Typical methods include: the *Cholesky* factorization $\mathbf{H} = \mathbf{L}\mathbf{L}^T$ or the *QR-decomposition*. A crucial parameter controlling the performances of a sparse direct linear solver is the fill-in, that is, the number of new non-zero elements introduced by the specific factorization. A key aspect in reducing the fill-in is the ordering of the variables. Since computing the optimal reordering is NP-hard, approximated techniques [6, 20, 43] are generally employed. Figure 8 shows the effect of different variable ordering on the *same* system matrix. A larger fill-in results in more demanding computations. We refer to works of Agarwal and Olson [1] and Davis [21] for a more detailed analysis about this topic.

We remark that the ordering of the variables depends only on the structure of the linear system, not on the values of the $\mathbf{H}$ matrix. The former does not change during the iterations. Therefore the ordering can be computed once at the beginning.

Algorithm 1 Gauss Newton algorithm for solving pose graphs. It computes the mean $\mathbf{x}^*$ and the information matrix $\mathbf{H}^*$ of the multivariate Gaussian approximation of the robot pose posterior from a graph of constraints

Require: $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}_{1:T}$: initial guess. $\mathcal{C} = \{\langle \mathbf{e}_{ij}(\cdot), \mathbf{\Omega}_{ij} \rangle\}$: constraints

Ensure: $\mathbf{x}^*$: new solution, $\mathbf{H}^*$: new information matrix

```

// find the maximum likelihood solution
while  $\neg$ converged do
   $\mathbf{b} \leftarrow \mathbf{0}$     $\mathbf{H} \leftarrow \mathbf{0}$ 
  for all  $\langle \mathbf{e}_{ij}, \mathbf{\Omega}_{ij} \rangle \in \mathcal{C}$  do
    // Compute the Jacobians  $\mathbf{A}_{ij}$  and  $\mathbf{B}_{ij}$  of the error function
     $\mathbf{A}_{ij} \leftarrow \left. \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i} \right|_{\mathbf{x}=\tilde{\mathbf{x}}}$     $\mathbf{B}_{ij} \leftarrow \left. \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j} \right|_{\mathbf{x}=\tilde{\mathbf{x}}}$ 
    // compute the contribution of this constraint to the linear system
     $\mathbf{H}_{[ii]} += \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij}$     $\mathbf{H}_{[ij]} += \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij}$ 
     $\mathbf{H}_{[ji]} += \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij}$     $\mathbf{H}_{[jj]} += \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij}$ 
    // compute the coefficient vector
     $\mathbf{b}_{[i]} += \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$     $\mathbf{b}_{[j]} += \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$ 
  end for
  // keep the first node fixed
   $\mathbf{H}_{[11]} += \mathbf{I}$ 
  // solve the linear system using sparse Cholesky factorization
   $\Delta \mathbf{x} \leftarrow \text{solve}(\mathbf{H} \Delta \mathbf{x} = -\mathbf{b})$ 
  // update the parameters
   $\tilde{\mathbf{x}} += \Delta \mathbf{x}$ 
end while
 $\mathbf{x}^* \leftarrow \tilde{\mathbf{x}}$ 
 $\mathbf{H}^* \leftarrow \mathbf{H}$ 
// release the first node
 $\mathbf{H}_{[11]}^* -= \mathbf{I}$ 
return  $\langle \mathbf{x}^*, \mathbf{H}^* \rangle$ 

```

3.3 Least Squares on a Manifold

A common approach in numeric to deal with non-Euclidean spaces is to perform the optimization on a manifold. A manifold is a mathematical space that is not necessarily Euclidean on a global scale but can be seen as Euclidean on a local scale [49]. Note that the manifold-based approach described here is similar to the way of minimizing functions in $SO(3)$ as described by Taylor and Kriegman [74].

In the context of the SLAM problem, each parameter block $\mathbf{x}_i$ consists of a translation vector $\mathbf{t}_i$ and a rotational component α_i . The translation $\mathbf{t}_i$ clearly forms a Euclidean space, while the rotational components α_i span over the non-Euclidean 2D or 3D rotation group $SO(2)$ or $SO(3)$. To avoid singularities, these spaces are usually described in an over-parametrized way, e.g., by rotation matrices or quaternions. Directly applying Eq. (16) to these over-parametrized representations breaks the constraints induced by the over-parametrization, which results in additional degrees of freedom and thus introduces errors in the solution. To overcome this problem, one can use a minimal representation for the rotation

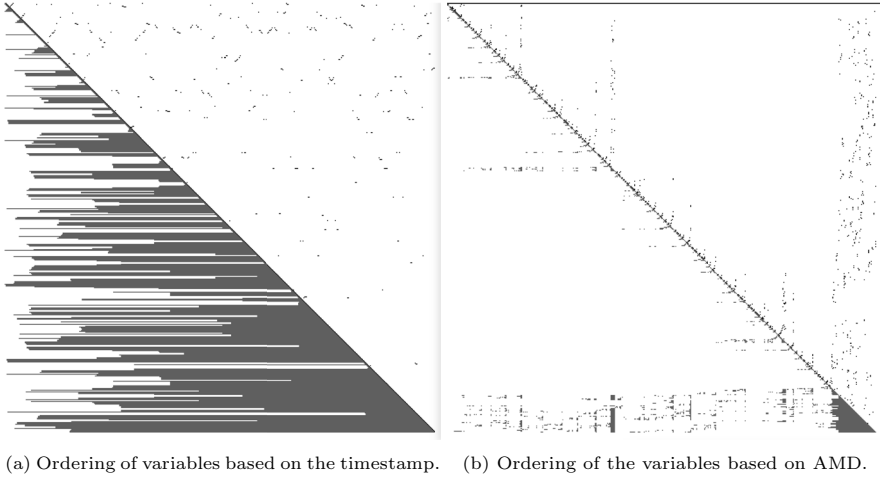


Fig. 8 Effects of the AMD variable reordering on the fill-in of the Cholesky decomposition of matrix $\mathbf{H}$, for the same problem. Black pixels in the upper triangular part indicate the non-zero blocks of $\mathbf{H}$. The lower triangular part shows in gray the fill in introduced by the Cholesky factorization. (a) shows that ordering the variables according to their trajectory timestamp leads to a significant fill in. In contrast (b) illustrates how AMD ordering results in a substantially sparser Cholesky factor.

(e.g., Euler angles in 3D). This, however, is subject to singularities. The singularities in the 2D case can be easily recovered by normalizing the angle. In 3D, however, this procedure is not straightforward.

An alternative is to consider the underlying space as a manifold and to define an operator $\boxplus$ that maps a local variation $\Delta \mathbf{x}$ in the Euclidean space to a variation on the manifold: $\Delta \mathbf{x} \mapsto \mathbf{x} \boxplus \Delta \mathbf{x}$. We refer the reader to the work of Hertzberg [38] for the mathematical details. With this operator, a new error function can be defined as

$$\check{\mathbf{e}}_{ij}(\Delta \tilde{\mathbf{x}}_i, \Delta \tilde{\mathbf{x}}_j) = \mathbf{e}_{ij}(\check{\mathbf{x}}_i \boxplus \Delta \tilde{\mathbf{x}}_i, \check{\mathbf{x}}_j \boxplus \Delta \tilde{\mathbf{x}}_j) \quad (20)$$

$$= \mathbf{e}_{ij}(\check{\mathbf{x}} \boxplus \Delta \tilde{\mathbf{x}}) \simeq \check{\mathbf{e}}_{ij} + \tilde{\mathbf{J}}_{ij} \Delta \tilde{\mathbf{x}}, \quad (21)$$

where $\check{\mathbf{x}}$ spans over the original over-parametrized space, for instance, quaternions. The term $\Delta \tilde{\mathbf{x}}$ is a small increment around the original position $\check{\mathbf{x}}$ and is expressed in a minimal representation.

As an example, in 3D SLAM a good choice of the parametrization of the rotations is the *vector part* of the unit quaternion. In more detail, one can represent the increments $\Delta \tilde{\mathbf{x}}$ as 6D vectors $\Delta \tilde{\mathbf{x}}^T = (\Delta \tilde{\mathbf{t}}^T \tilde{\mathbf{q}}^T)$, where $\Delta \tilde{\mathbf{t}}$ denotes the translation and $\tilde{\mathbf{q}}^T = (\Delta q_x \Delta q_y \Delta q_z)^T$ is the vector part of the unit quaternion representing the 3D rotation. Conversely, $\check{\mathbf{x}}^T = (\tilde{\mathbf{t}}^T \tilde{\mathbf{q}}^T)$ uses a quaternion $\tilde{\mathbf{q}}$ to encode the rotational part. Thus, the operator $\boxplus$ can be expressed by first converting $\Delta \tilde{\mathbf{q}}$ to a full quaternion $\Delta \mathbf{q}$ and then applying the transformation $\Delta \mathbf{x}^T = (\Delta \mathbf{t}^T \Delta \mathbf{q}^T)$ to $\check{\mathbf{x}}$.

In the equations describing the error minimization, these operations can nicely be encapsulated by the $\boxplus$ operator. The Jacobian $\tilde{\mathbf{J}}_{ij}$ can be expressed by

$$\tilde{\mathbf{J}}_{ij} = \left. \frac{\partial \mathbf{e}_{ij}(\tilde{\mathbf{x}} \boxplus \Delta \tilde{\mathbf{x}})}{\partial \Delta \tilde{\mathbf{x}}} \right|_{\Delta \tilde{\mathbf{x}}=\mathbf{0}}. \quad (22)$$

Since in the previous equation $\mathbf{e}$ depends only on $\Delta \tilde{\mathbf{x}}_i$ and $\Delta \tilde{\mathbf{x}}_j$ we can further expand it as follows:

$$\begin{aligned} \tilde{\mathbf{J}}_{ij} &= \left(\underbrace{\left. \frac{\partial \mathbf{e}_{ij}(\tilde{\mathbf{x}} \boxplus \Delta \tilde{\mathbf{x}})}{\partial \Delta \tilde{\mathbf{x}}_i} \right|_{\Delta \tilde{\mathbf{x}}=\mathbf{0}}}_{\tilde{\mathbf{A}}_{ij}} \quad \underbrace{\left. \frac{\partial \mathbf{e}_{ij}(\tilde{\mathbf{x}} \boxplus \Delta \tilde{\mathbf{x}})}{\partial \Delta \tilde{\mathbf{x}}_j} \right|_{\Delta \tilde{\mathbf{x}}=\mathbf{0}}}_{\tilde{\mathbf{B}}_{ij}} \quad \dots \right) \end{aligned} \quad (23)$$

Using the rule for the partial derivatives and exploiting the fact that the Jacobian is evaluated in $\Delta \tilde{\mathbf{x}} = \mathbf{0}$, the non-zero blocks become:

$$\frac{\partial \mathbf{e}_{ij}(\tilde{\mathbf{x}} \boxplus \Delta \tilde{\mathbf{x}}_i)}{\partial \Delta \tilde{\mathbf{x}}_i} = \underbrace{\frac{\partial \mathbf{e}_{ij}(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i}}_{\mathbf{A}_{ij}} \cdot \underbrace{\left. \frac{\tilde{\mathbf{x}}_i \boxplus \Delta \tilde{\mathbf{x}}_i}{\partial \Delta \tilde{\mathbf{x}}_i} \right|_{\Delta \tilde{\mathbf{x}}=\mathbf{0}}}_{\mathbf{M}_i} \quad (24)$$

$$\frac{\partial \mathbf{e}_{ij}(\tilde{\mathbf{x}} \boxplus \Delta \tilde{\mathbf{x}}_j)}{\partial \Delta \tilde{\mathbf{x}}_j} = \underbrace{\frac{\partial \mathbf{e}_{ij}(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_j}}_{\mathbf{B}_{ij}} \cdot \underbrace{\left. \frac{\tilde{\mathbf{x}}_j \boxplus \Delta \tilde{\mathbf{x}}_j}{\partial \Delta \tilde{\mathbf{x}}_j} \right|_{\Delta \tilde{\mathbf{x}}=\mathbf{0}}}_{\mathbf{M}_j} \quad (25)$$

Accordingly, one can easily derive from the Jacobian not defined on a manifold of Eq. (17) a Jacobian on a manifold just by multiplying its non-zero blocks with the derivative of the $\boxplus$ operator computed in $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_j$.

With a straightforward extension of the notation, we can insert Eq. (21) in Eq. (9). This leads to the following linear system:

$$\tilde{\mathbf{H}} \Delta \tilde{\mathbf{x}}^* = -\tilde{\mathbf{b}}. \quad (26)$$

Since the increments $\Delta \tilde{\mathbf{x}}^*$ are computed in the local Euclidean surroundings of the initial guess $\tilde{\mathbf{x}}$, they need to be remapped into the original overparametrized space by the $\boxplus$ operator. Accordingly, the update rule of Eq. (16) becomes

$$\mathbf{x}^* = \tilde{\mathbf{x}} \boxplus \Delta \tilde{\mathbf{x}}^*. \quad (27)$$

Thus, formalizing the minimization problem on a manifold consists of first computing a set of increments in a local Euclidean approximation around the initial guess by Eq. (26), and second accumulating the increments in the global non-Euclidean space by Eq. (27). Note that the linear system computed on a manifold representation has

the same structure as the linear system computed in Euclidean space. One can easily derive a manifold version of a graph minimization from a non-manifold version only by defining an $\boxplus$ operator and its Jacobian $\mathbf{M}_i$ w.r.t. the corresponding parameter block.

3.4 Dealing with Outliers

In Sect. 3.3, we described a methodology to solve a pose-graph by minimizing the negative log-likelihood of the measurements. More concisely, we compute the optimal state $\mathbf{x}^*$ such that:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argminn}} \sum_{\langle i, j \rangle \in \mathcal{C}} \mathbf{e}_{ij}(\mathbf{x})^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}(\mathbf{x}). \quad (28)$$

The procedure reported in the previous section assumes all edges *correct*, albeit affected by noise. Still, in many real case scenarios, this is not the case. This is mainly due to aspects that are hard to model and ultimately lead to incorrect data associations. These *wrong* edges are commonly referred to as *outliers*. On the contrary, *inliers* represent the *correct* measurements.

A common assumption made by several techniques to reject outliers is that the inliers tend to agree towards a common solution while outliers do not. This fact is at the root of consensus schemes such as RANSAC [26]. In the context of pose-graph optimization, the quadratic nature of the error terms leads to over-accounting for measurement whose error is significant, albeit those measurements are typically outliers. However, there are circumstances where all errors are quite large, even if no outliers are present. A typical case occurs when we start the optimization from an initial guess far from the optimum.

A possible solution to this issue lies in optimizing under a cost function that grows sub-quadratically with the error. Indicating with $u_{ij}(\mathbf{x})$ the L1 Omega-norm of the error term in Eq. (28), its derivatives with respect to the state variable $\mathbf{x}$ can be computed as follows:

$$u_{ij}(\mathbf{x}) = \sqrt{\mathbf{e}_{ij}(\mathbf{x})^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}(\mathbf{x})} \quad (29)$$

$$\frac{\partial u_{ij}(\mathbf{x})}{\partial \mathbf{x}} = \frac{1}{u_{ij}(\mathbf{x})} \mathbf{e}_{ij}(\mathbf{x})^T \boldsymbol{\Omega}_{ij} \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}}. \quad (30)$$

We can generalize Eq. (28) by introducing a scalar function $\rho(u)$ that computes a new error term as a function of the L1-norm. Equation (28) is a special case for

$\rho(u) = \frac{1}{2}u^2$. Hence, our new problem will consist in minimizing the following function:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argminn}} \sum_{\langle i,j \rangle \in \mathcal{C}} \rho(u_{ij}(\mathbf{x})) \quad (31)$$

Going more in detail and analyzing the gradients of Eq. (31), we have the following relation, where we omitted the indices ij being clear that we refer to a single measurement:

$$\begin{aligned} \frac{\partial \rho(u(\mathbf{x}))}{\partial \mathbf{x}} &= \left. \frac{\partial \rho(u)}{\partial u} \right|_{u=u(\mathbf{x})} \frac{\partial u(\mathbf{x})}{\partial \mathbf{x}} \\ &= \left. \frac{\partial \rho(u)}{\partial u} \right|_{u=u(\mathbf{x})} \frac{1}{u(\mathbf{x})} \mathbf{e}(\mathbf{x})^T \boldsymbol{\Omega} \frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}} \\ &= \gamma(\mathbf{x}) \mathbf{e}(\mathbf{x})^T \boldsymbol{\Omega} \frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}} \end{aligned} \quad (32)$$

where

$$\gamma(\mathbf{x}) = \left. \frac{\partial \rho(u)}{\partial u} \right|_{u=u(\mathbf{x})} \frac{1}{u(\mathbf{x})}. \quad (33)$$

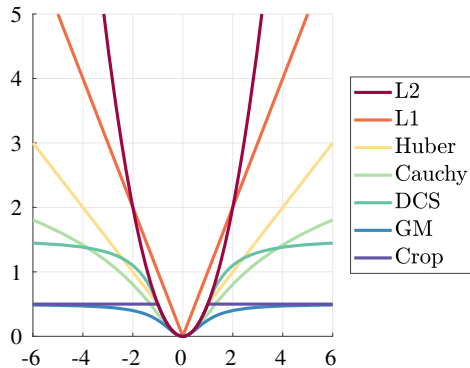
The *robustifier* function $\rho(\cdot)$ acts on the gradient, modulating the magnitude of the error term through a scalar function $\gamma(\mathbf{x})$. Still, we can also compute the gradient of the Eq. (28) as follows:

$$\frac{\partial \mathbf{e}_{ij}(\mathbf{x})^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}} = 2 \mathbf{e}_{ij}(\mathbf{x})^T \boldsymbol{\Omega}_{ij} \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}}. \quad (34)$$

We notice that Eqs. (34) and (32) differ by a scalar term $\gamma(\mathbf{x})$ that depends on $\mathbf{x}$. By absorbing this scalar term at each iteration in a new information matrix $\tilde{\boldsymbol{\Omega}}_{ij}(\mathbf{x}) = \gamma_{ij}(\mathbf{x}) \boldsymbol{\Omega}_{ij}$, we can rely on the iterative algorithm illustrated in the previous sections to implement a robust estimator. At each iteration, we compute $\gamma_{ij}(\mathbf{x})$ based on the result of the previous iteration. As the iteration proceeds, the scaling factors $\gamma_{ij}(\mathbf{x})$ will stabilize to a value, and the rest of the machinery proceeds by minimizing a quadratic form constructed on reweighted addends, as usual. This schema is called iterative reweighted least squares (IRLS).

The use of robust cost functions biases the information matrix of the system $\mathbf{H}$. Accordingly, if we want to recover an estimate of the solution uncertainty when using robust cost functions, we need to “undo” the effect of function $\rho(\cdot)$. This can be achieved easily by recomputing $\mathbf{H}$ after convergence, considering only inliers and disabling the robustifier—i.e. setting $\rho(u) = \frac{1}{2}u^2$. Figure 9 illustrates some of the most common cost function used in robotics and computer vision. Further information on modern, robust cost function can be found in the work of

Fig. 9 Commonly used robust kernel functions. The kernel threshold is set to 1 in all cases



MacTavish et al. [52] and Chebrolu et al. [15]. Algorithm 2 provides a robustified version of the Gauss-Newton method for SLAM.

4 Practical Applications

In this section, we describe applications of the proposed methods. We first present a complete 2D mapping scenario and then briefly discuss the application to 3D and the advantage of the manifold representation.

It illustrates the application to 2D, and we processed the data recorded with the mobile robot equipped with a laser range finder at the Intel Research Laboratory in Seattle. This data consists of odometry measurements describing 2D transformations corresponding to the platform's movements between consecutive time frames and 2D laser range data.

The graph is constructed in the following way:

- Whenever the robot moves more than 0.5 m or rotates more than 0.5° in radian, the algorithm adds a new vertex to the graph and labels it with the current laser observation.
- We match this laser scan with the previously acquired one to improve the odometry estimate and add the corresponding edge to the graph. We employ a variant of the scan-matcher described by Olson [61].
- When the robot reenters a known area after traveling for a long time in a previously unknown region, the algorithm seeks for matches of the current scan with the past measurements (loop closing). If a matching between the current observation and the observation of another node succeeds, the algorithm adds a new edge to the graph. The edge is labeled with the relative transformation that makes the two scans to overlap best. Matching the current measurement with all previous scans would be extremely inefficient and error prone, since it does not consider the known prior about the robot location. Instead, the algorithm selects the candidate nodes in the past as the ones whose 3σ marginal

Algorithm 2 Robustified Manifold version of Algorithm 1. While this algorithm has the same computational complexity, it is substantially more robust than the non-manifold version, especially in the 3D case

Require: $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}_{1:T}$: initial guess. $\mathcal{C} = \{\langle \mathbf{e}_{ij}(\cdot), \mathbf{\Omega}_{ij} \rangle\}$: constraints

Ensure: $\mathbf{x}^*$: new solution

```

while  $\neg$ converged do
  // Compute the auxiliary Jacobians  $M_{1:T}$  over the manifold
  for all  $\tilde{\mathbf{x}}_i \in \tilde{\mathbf{x}}$  do
     $\mathbf{M}_i \leftarrow \left. \frac{\tilde{\mathbf{x}}_i \boxplus \Delta \tilde{\mathbf{x}}_i}{\partial \Delta \tilde{\mathbf{x}}_i} \right|_{\Delta \tilde{\mathbf{x}}=0}$ 
  end for
   $\tilde{\mathbf{b}} \leftarrow \mathbf{0} \quad \tilde{\mathbf{H}} \leftarrow \mathbf{0}$ 
  for all  $\langle \mathbf{e}_{ij}, \mathbf{\Omega}_{ij} \rangle \in \mathcal{C}$  do
    // Compute the L1 norm of the error
     $u_{ij} \leftarrow \sqrt{\mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}}$ 
    // Compute the scaling factor
     $\gamma_{ij} \leftarrow \left. \frac{\partial \rho(u)}{\partial u} \right|_{u=u_{ij}} \frac{1}{u_{ij}}$ 
    // Compute a temporary Omega
     $\mathbf{\Omega} \leftarrow \gamma_{ij} \mathbf{\Omega}_{ij}$ 
    // Compute the Jacobians  $\mathbf{A}_{ij}$  and  $\mathbf{B}_{ij}$  of the error function
     $\mathbf{A}_{ij} \leftarrow \left. \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i} \right|_{\mathbf{x}=\tilde{\mathbf{x}}} \quad \mathbf{B}_{ij} \leftarrow \left. \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j} \right|_{\mathbf{x}=\tilde{\mathbf{x}}}$ 
    // Project the Jacobians through the manifold
     $\tilde{\mathbf{A}}_{ij} \leftarrow \mathbf{A}_{ij} \mathbf{M}_i \quad \tilde{\mathbf{B}}_{ij} \leftarrow \mathbf{B}_{ij} \mathbf{M}_j$ 
    // compute the nonzero Hessian blocks
     $\tilde{\mathbf{H}}_{[ii]} += \tilde{\mathbf{A}}_{ij}^T \mathbf{\Omega} \tilde{\mathbf{A}}_{ij} \quad \tilde{\mathbf{H}}_{[ij]} += \tilde{\mathbf{A}}_{ij}^T \mathbf{\Omega} \tilde{\mathbf{B}}_{ij}$ 
     $\tilde{\mathbf{H}}_{[ji]} += \tilde{\mathbf{B}}_{ij}^T \mathbf{\Omega} \tilde{\mathbf{A}}_{ij} \quad \tilde{\mathbf{H}}_{[jj]} += \tilde{\mathbf{B}}_{ij}^T \mathbf{\Omega} \tilde{\mathbf{B}}_{ij}$ 
    // compute the coefficient vector
     $\tilde{\mathbf{b}}_{[i]} += \tilde{\mathbf{A}}_{ij}^T \mathbf{\Omega} \mathbf{e}_{ij} \quad \tilde{\mathbf{b}}_{[j]} += \tilde{\mathbf{B}}_{ij}^T \mathbf{\Omega} \mathbf{e}_{ij}$ 
  end for
  // keep the first node fixed
   $\mathbf{H}_{[11]} += \mathbf{I}$ 
  // solve the linear system using sparse Cholesky factorization
   $\Delta \tilde{\mathbf{x}} \leftarrow \text{solve}(\tilde{\mathbf{H}} \Delta \tilde{\mathbf{x}} = -\tilde{\mathbf{b}})$ 
  // update the parameters
  for all  $\tilde{\mathbf{x}}_i \in \tilde{\mathbf{x}}$  do
     $\tilde{\mathbf{x}}_i \leftarrow \tilde{\mathbf{x}}_i \boxplus \Delta \tilde{\mathbf{x}}_i$ 
  end for
end while
return  $\mathbf{x}^*$ 

```

covariances contains the current robot pose. These covariances can be obtained as the diagonal blocks of the inverse of a reduced Hessian $\mathbf{H}_{\text{red}}$, which is obtained from $\mathbf{H}$ by removing rows and the columns of the newly inserted robot pose. $\mathbf{H}_{\text{red}}$ is the information matrix of all the trajectory when assuming fixed the current position.

- The algorithm performs the optimization whenever a loop closure is detected.



Fig. 10 Intel research lab. Left: unoptimized pose graph overlaid on top of the resulting map. Right: the optimized pose graph and the resulting consistent map

At the end of the run, the graph consists of 1802 nodes and 3546 edges. Even for this relatively large problem, the optimization can be carried on in 100 ms on a standard laptop (Intel Core2@2.4 GHz). Since the robot travels at a velocity of around 1 m/s, the graph optimization can be executed after adding every node instead of after detecting a loop closure. Figure 10 shows the effect of the optimization process on the trajectory, while Fig. 11 illustrates the uncertainty ellipses. The robot is located in the region where the ellipse becomes small. Note that the poses in $SE(2)$ do not need to be over parametrized, so in this case there is no advantage in utilizing manifolds.

Extending the SLAM algorithm presented in the previous section to 3D is relatively straightforward. We only have to replace the 2D scan matching and loop closure detection with their 3D counterparts that operate on 3D point clouds instead of on single laser scans. In our implementation, we utilize the popular ICP algorithm [10], and for determining the loop closures, we use the algorithm by Steder et al. [72]. Additionally, each node of the graph and each constraint lives in $SE(3)$. Typical outputs of this algorithm are illustrated in Fig. 2a, b.

The minimum number of parameters required to represent an element of $SE(3)$ is 6, and one possible choice consists of a 3D translation vector plus the three Euler angles. Utilizing this parametrization leads to Algorithm 1. However, this minimal representation is subject to singularities that can be avoided by utilizing an over-parametrized state space. Alternatively, one can describe the relative perturbations of the optimization problem $\Delta \tilde{\mathbf{x}}$ in a minimal representation while leaving the poses in the original over-parametrized space. This leads to Algorithm 2, which shows substantially better convergence properties than Algorithm 1 [31].

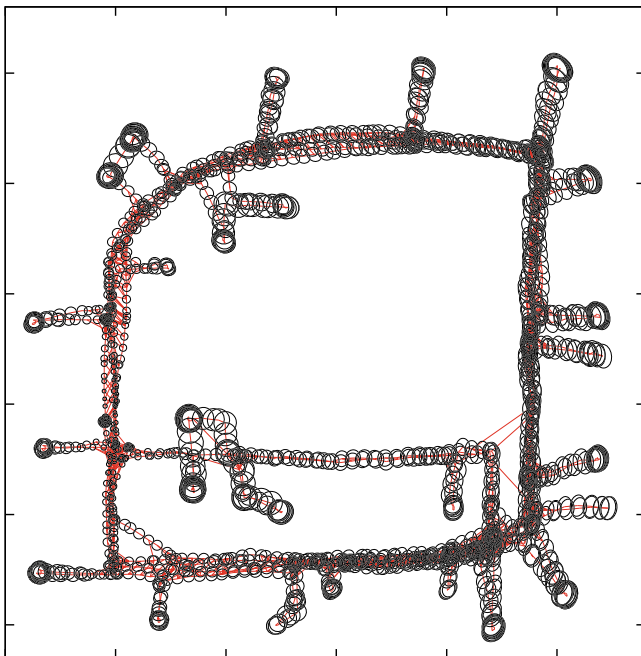


Fig. 11 Pose uncertainty estimate for a real-world data set

5 Further Reading

SLAM has been a highly relevant topic in the robotics community for more than two decades [7, 8, 12, 71]. In those years, a large number of methods have been proposed. Throughout this tutorial, we will focus on graph-based approaches and consider only such approaches when discussing related work. We suggest to the reader interested in more details about the current state of the art in SLAM to read the survey by Cadena et al. [12], and for an overview the book chapter by Stachniss et al. [71].

In the context of graph-based approaches in robotics, Lu and Milios [51] were the first to refine a map by globally optimizing the system of equations to reduce the error introduced by constraints. They approached the problem by performing a nonlinear least squares optimization, as reported in this chapter. However, the original formulation of [51] relied on a Euclidean parameterization of the poses in the graph similar to Algorithm 1. In this work, the authors did not specify how to solve the sparse linear system, and common implementations were using dense solvers. This choice induced a computational bottleneck in the system. This work relied on a front end based on a ICP [50] to construct the graph but was unable to detect large loops. Subsequently, Gutmann and Konolige [34] proposed an effective way for constructing such a network and for detecting loop closures through a

correlative search. The cost of the back-end was amortized by running it only when a large closure was detected.

Since the work of Lu and Milios, many back-ends have been proposed. All these approaches focus on finding a minimum error configuration in the pose-graph. These methods can be roughly classified based on the principle used to minimize the error function. Nonlinear relaxation approaches are iterative methods that, at each step, move each node in the graph sequentially to a position that better satisfies the constraints with the node's neighbors. Albeit easy to implement and robust, this class of methods suffers from low convergence speed. Howard et al. [40] were among the first to use relaxation to localize the robot and build a map. Frese et al. [27] propose a variant of Gauss-Seidel relaxation called multi-level relaxation (MLR). It applies relaxation at different resolutions, thus improving the convergence speed of the algorithm.

Gradient descent approaches, in contrast, attempt to minimize the error in the pose graph by computing at each iteration an update for all poses that reduces the error. This update vector is chosen along the gradient of the error function. Similar to relaxation techniques, gradient-based approaches suffer from slow convergence speed. Stochastic gradient descent, a subclass of gradient descent approaches, aim at minimizing a subset (or even only one) constraint at a time. Olson et al. [62] presented an efficient sequential parametrization for the poses in the graph that allows implementing pose updates in logarithmic time when using stochastic gradient descent. Grisetti et al. [30] proposed an extension of Olson's approach that uses a tree parametrization of the nodes in 2D and 3D. Using a spanning tree restricts the effect of each constraint's update to the set of poses along with small loops, resulting in "larger" steps that improve the convergence speed.

In this chapter, we focused on nonlinear least squares (NLS) back-ends that are most common nowadays since they exhibit the highest convergence speed among iterative techniques. NLS approaches stem from the original work of Lu and Milios [51], and they mostly differ in the way the graph is parameterized or in the way the linear system is solved. Dellaert and Kaess [22] were the first to exploit sparse matrix factorizations to solve the linearized problem in off-line SLAM. Subsequently, Kaess et al. [41] presented iSAM, an online version of Dellaert's and Kaess's previous work [22] that exploits partial reorderings to compute the sparse factorization, and more recently iSAM 2.0 [42] that embeds an ad-hoc strategy to solve the linear system. Kuemmerle et al. [46] proposed a C++ system that supports the optimization of arbitrary sparse least squares (and thus also pose graph) problems called g^2o . This package decouples the representation of the graph from the techniques used to solve the linear system and to compute the initial guess for the optimization. Along the same line of generalization, Google proposed the Ceres solver [4], which supports automatic differentiation to compute the Jacobian matrices.

Hierarchical extensions to these approaches have been proposed by Ni et al. [58] and Grisetti et al. [32, 33]. These approaches employ a divide-and-conquer strategy to partition the original large problem into smaller ones and then utilize these partial solutions to construct a sparse global problem. The global problem solution is the

arrangement of the partial solutions (local maps) in the space. In particular Ni et al. [58] utilizes nested dissection and an efficient parametrization to solve large off-core problems involving arbitrary types of variables efficiently. Conversely, Grisetti et al. [32] encapsulates the relations between sub-problems in nonlinear constraints connecting them, but it is restricted to operate on pose-graphs, while Grisetti et al. [33] is not restricted to pose-graphs.

Most optimization techniques focus on computing the best configuration of poses, given the constraints, and are called SLAM back-ends. In contrast, SLAM front-ends seek to interpret the sensor data to obtain the constraints that are the basis for the optimization approaches. Olson [60], for example, presented a front-end with outlier rejection based on spectral clustering. For making data associations in the SLAM front-ends statistical tests such as the χ^2 test or joint compatibility test [56] are often applied. The work of Nüchter et al. [59] aims at building an integrated SLAM system for 3D mapping. The main focus lies on the SLAM front-end for finding constraints. For optimization, a variant of the approach of Lu and Milios [51] for 3D settings is used. More recent approaches use surfels for efficiently representing and registering large scenes [9] still using the ideas of pose graphs. Also the advancements in learning-based approaches affected the SLAM community for building semantic maps [17] or removing dynamic objects from the scene [19].

All approaches presented so far are iterative and subject to the initial guess. Remarkably, Carlone et al. [13] proposed a linear formulation for the 2D pose graph problems that converges in one iteration and whose result does not depend on the initial guess. Unfortunately, this approach is restricted to special shapes of the covariance matrices.

The methods proposed in this chapter constitute the base for an effective back-end and are at the root of most modern systems for graph optimization. However, a back-end is only a part of the SLAM system. For completeness, in the remainder of this section, we will briefly review some well-known graph-slam systems that rely on these back-ends. One of the first online large-scale stereo visual SLAM systems that appeared in literature is FrameSLAM [2]. With this work, Konolige and Agrawal introduced a complete feature-based SLAM approach with bundle adjustment running in real-time. For their approach, they use CenSure features and integrate IMU information into the visual odometry computation.

More recently, full graph-based SLAM systems also operate on raw data. Most of them rely on either monocular or stereo cameras or lasers. Klein et al. [44] presented a compact monocular system that can run on limited hardware. The authors achieve a high frame rate by decoupling the optimization and tracking tasks performed by two different threads. Subsequently, Pire et al. [64] presented a compact, appearance-based stereo visual method S-PTAM. S-PTAM runs on two threads for achieving real-time computation. The first thread is in charge of tracking the position of the camera and relies on BRIEF features, while the second thread performs anytime full bundle adjustment and is based on g2o [46]. In contrast to the other presented algorithms S-PTAM is not performing explicit relocalization and relies on full bundle adjustment for preserving the map consistency. This results

in a computational complexity that grows with the size of the environment being mapped.

Mur-Artal and Tardos [54] introduced the open-source mono and stereo visual SLAM system ORB-SLAM2. The system originated from the prominent, monocular ORB-SLAM [55] published by the same authors. ORB-SLAM2 achieves extraordinary performance thanks to a highly reliable tracking front end and frequent relocalization using ORB features. ORB-SLAM uses compact data structures to store the system state that partially inspired the architecture of several other visual SLAM systems. ORB-SLAM closes loops in real-time by utilizing a bag of words approach first proposed by Galvez-Lopez and Tardos [28]. Like S-PTAM, ORB-SLAM2 employs g^2o for local bundle adjustment. The ORB-SLAM2 pipeline runs on three parallel threads that handle tracking, relocalization, and optimization.

S-LSD-SLAM proposed by Engel et al. [23] is a direct, hence featureless SLAM approach operating in large-scale at high processing speeds faster than real-time on a single thread. Engel exploits static and temporal stereo image changes at pixel level while also considering lightning changes. The system can perform relocalization by supporting components for loop closure detection, such as BoW. When a closure is detected, a back-end optimization is triggered on the entire map. Schlegel et al. [68] recently presented ProSLAM, a straightforward yet efficient stereo SLAM pipeline targeted at education. It uses a custom build front-end based on binary features retrieved using a binary search structure [67], and relies on g^2o as optimization back-end. ProSLAM can run at 150 Hz on a single thread of an i7 processor on standard datasets.

In 2010, the appearance of depth cameras triggered the development of 3D dense mapping systems [37]. Notably, Kinect Fusion (KinFu) [57] was one of the first methods that exploited the number-crunching abilities of modern GPUs to build incrementally a dense volumetric map from depth images. It used a truncated signed distance function (TSDF) to represent the map and a variant of point-to-plane ICP to register subsequent measurements. This approach was initially targeted at small environments since it did not enforce global map consistency. Whelan et al. [78] were the first to use a Kinect Fusion-based tracker to build a SLAM system able to operate in large scale called Continuous. This approach incrementally builds a pose-graph by assembling local maps obtained by processing subsequences of measurements with KinFu. It relies on visual methods that exploit the RGB channel of the depth camera to detect loop closures. Continuous uses iSAM as back-end. Other extensions aims at removing dynamics from the RGB-D data [63].

The ATLAS framework [11] constructs a two-level hierarchy of graphs and employs a Kalman filter to construct the bottom level. Then, a global optimization approach aligns the local maps at the second level. Similar to ATLAS, Estrada et al. proposed Hierarchical SLAM [24] as a technique for using independent local maps. Vincent et al. proposed Karto SLAM [77] a graph-based SLAM system for 2D laser data. Similar to Google Cartographer [39] these methods combine a scan-registration front-end with a graph optimization back-end to build an occupancy representation of the environment that can be used for navigation. Subsequently Lazáro et al. [48] applied the same concepts to implement a map merging scheme

that reduces the number of nodes in the graph by merging nearby local maps. This limits the size of the graph and thus improves the run-time while enhancing the robustness to environment dynamics.

We are observing a community trend in unifying and combining front-ends, back-ends, and their components in a coherent system. Schneider et al. [69] organized the different tools in an environment called MATLAB, which allows the user to control the execution of each algorithm through a command-line interface. MATLAB standardizes data structures and interfaces and thus renders MATLAB a convenient tool for analyzing and testing different algorithms and combinations of them. Labbé and Michaud [47] recently proposed Real-time Appearance Based Mapping (RTAB-Map), a generalized pipeline for depth and laser systems. Once again, this standardizes a SLAM pipeline by offering multiple front-ends that can deal with heterogeneous data.

Very recent approaches aim at utilizing deep learning methods to improve the capabilities of robots to solve the SLAM problem (as well as other challenges in robotics [73]). The greatest potential for deep learning definitely lies in the front-end to improve the robustness of the loop closure detection [16, 18, 65]. However, deep learning methods have also been applied successfully for change detection [5], a critical task for robots required to solve the life-long SLAM problem in which the map has to be updated frequently due to changes in the environment.

6 Conclusions

This chapter presented a tutorial on graph-based SLAM and provided the essential ingredients needed to build a SLAM system. Many state-of-the-art approaches to SLAM utilize these ideas to realize highly efficient implementations that can solve fairly large SLAM problems.

Appendix

In the following we will provide the definitions and the derivations for the Jacobians to implement the suggested algorithm. Due to space limitations we do not expand the Jacobians in the 3D case. However, these Jacobians can either be computed numerically or by using a computer algebra system.

Error Functions and Jacobians for the 2D Case

The basic entities in the 2D case are defined as

$$\mathbf{x}_i^\top = (\mathbf{t}_i^\top, \theta_i) \quad (35)$$

$$\mathbf{z}_{ij}^\top = (\mathbf{t}_{ij}^\top, \theta_{ij}) \quad (36)$$

where $\mathbf{t}_i$ and $\mathbf{t}_{ij}$ are 2D vectors and θ_i and θ_{ij} are rotation angles which are normalized to $[-\pi, \pi)$. The error function is

$$\mathbf{e}_{ij}(\mathbf{x}) = \begin{pmatrix} \mathbf{R}_{ij}^\top (\mathbf{R}_i^\top (\mathbf{t}_j - \mathbf{t}_i) - \mathbf{t}_{ij}) \\ \theta_j - \theta_i - \theta_{ij} \end{pmatrix}, \quad (37)$$

where $\mathbf{R}_i$ and $\mathbf{R}_{ij}$ are the 2×2 rotation matrices of θ_i and θ_{ij} with the following structure

$$\mathbf{R}_i = \begin{pmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{pmatrix}. \quad (38)$$

The Jacobians of the error function are

$$\mathbf{A}_{ij} = \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i} = \begin{pmatrix} -\mathbf{R}_{ij}^\top \mathbf{R}_i^\top \mathbf{R}_{ij}^\top \frac{\partial \mathbf{R}_i^\top}{\partial \theta_i} (\mathbf{t}_j - \mathbf{t}_i) \\ \mathbf{0}^\top & -1 \end{pmatrix} \quad (39)$$

$$\mathbf{B}_{ij} = \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j} = \begin{pmatrix} \mathbf{R}_{ij}^\top \mathbf{R}_i^\top \mathbf{0} \\ \mathbf{0}^\top & 1 \end{pmatrix}. \quad (40)$$

The $\boxplus$ operator is defined as

$$\mathbf{x} \boxplus \Delta \tilde{\mathbf{x}} = \mathbf{x} + \Delta \tilde{\mathbf{x}} \quad (41)$$

The angles are normalized to $[-\pi, \pi)$ after applying the increments. The Jacobians of the manifold in the 2D case evaluate to the identity matrix:

$$M_i = \left. \frac{\mathbf{x}_i \boxplus \Delta \tilde{\mathbf{x}}_i}{\partial \Delta \tilde{\mathbf{x}}_i} \right|_{\Delta \tilde{\mathbf{x}}=\mathbf{0}} = \mathbf{I}_3 \quad (42)$$

$$M_j = \left. \frac{\mathbf{x}_j \boxplus \Delta \tilde{\mathbf{x}}_j}{\partial \Delta \tilde{\mathbf{x}}_j} \right|_{\Delta \tilde{\mathbf{x}}=\mathbf{0}} = \mathbf{I}_3 \quad (43)$$

Error Functions for the 3D Case

The basic entities in the 3D case are defined as

$$\mathbf{x}_i^\top = (\mathbf{t}_i^\top, \mathbf{q}_i^\top) \quad (44)$$

$$\mathbf{z}_{ij}^\top = (\mathbf{t}_{ij}^\top, \mathbf{q}_{ij}^\top), \quad (45)$$

where $\mathbf{q}$ denotes the unit quaternion $\mathbf{q}^\top = (q_x, q_y, q_z, q_w)^\top$, i.e., $\|\mathbf{q}\| = 1$. The error function is

$$\mathbf{e}_{ij}(\mathbf{x}) = \left(\mathbf{z}_{ij}^{-1} \oplus (\mathbf{x}_i^{-1} \oplus \mathbf{x}_j) \right)_{[1:6]}, \quad (46)$$

where $\oplus$ is the motion composition operator

$$\mathbf{x}_i \oplus \mathbf{x}_j = \begin{pmatrix} \mathbf{q}_i(\mathbf{t}_j) \\ \mathbf{q}_i \cdot \mathbf{q}_j \end{pmatrix} \quad (47)$$

and the operator $(\cdot)_{[1:6]}$ selects the first 6 elements of its vector argument.

The Jacobians of the error function are:

$$\mathbf{A}_{ij} = \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i} \quad (48)$$

$$\mathbf{B}_{ij} = \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}. \quad (49)$$

The $\boxplus$ operator maps $\Delta \tilde{\mathbf{x}}_i^\top = (\Delta \tilde{\mathbf{t}}_i^\top, \Delta \tilde{\mathbf{q}}_i^\top)$ to the original space

$$\mathbf{x}_i \boxplus \Delta \tilde{\mathbf{x}}_i = \mathbf{x}_i \oplus \begin{pmatrix} \Delta \tilde{\mathbf{t}}_i \\ \Delta \tilde{\mathbf{q}}_i \\ \sqrt{1 - \|\Delta \tilde{\mathbf{q}}_i\|^2} \end{pmatrix}, \quad (50)$$

where $\Delta \tilde{\mathbf{t}}_i$ denotes the translation and $\Delta \tilde{\mathbf{q}}_i^\top = (\Delta q_x, \Delta q_y, \Delta q_z)^\top$ is the vector part of the unit quaternion representing the 3D rotation and thus $\|\Delta \tilde{\mathbf{q}}_i\| \leq 1$. The Jacobians of the manifold in the 3D case are given by

$$M_i = \left. \frac{\mathbf{x}_i \boxplus \Delta \tilde{\mathbf{x}}_i}{\partial \Delta \tilde{\mathbf{x}}_i} \right|_{\Delta \tilde{\mathbf{x}}=0} \quad (51)$$

$$M_j = \left. \frac{\mathbf{x}_j \boxplus \Delta \tilde{\mathbf{x}}_j}{\partial \Delta \tilde{\mathbf{x}}_j} \right|_{\Delta \tilde{\mathbf{x}}=0}. \quad (52)$$

References

1. P. Agarwal, E. Olson, Variable reordering strategies for SLAM, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, Piscataway, 2012), pp. 3844–3850
2. M. Agrawal, K. Konolige, FrameSLAM: From bundle adjustment to real-time visual mapping. *IEEE Trans. Robot.* **24**, 1066–1077 (2008)

3. P. Agarwal, W. Burgard, C. Stachniss, A survey of geodetic approaches to mapping and the relationship to graph-based SLAM. *IEEE Robot. Autom. Mag.* **21**(3), 63–80 (2014). <https://doi.org/10.1109/MRA.2014.2322282>
4. S. Agarwal, K. Mierle, The Ceres Solver Team, Ceres solver. Apache-2.0, version 2.2 (2023). <https://github.com/ceres-solver/ceres-solver>
5. Alcantarilla, P.F., Stent, S., Ros, G., Arroyo, R., Gherardi, R., Street-view change detection with deconvolutional networks, in *Proceedings of Robotics: Science and Systems (RSS)* (2016)
6. P.R. Amestoy, T.A. Davis, I.S. Duff, An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.* **17**(4), 886–905 (1996)
7. T. Bailey, H.F. Durrant-Whyte, Simultaneous localisation and mapping (SLAM): part I. *IEEE Robot. Autom. Mag.* **13**(2), 99–110 (2006)
8. T. Bailey, H.F. Durrant-Whyte, Simultaneous localisation and mapping (SLAM): part II. *IEEE Robot. Autom. Mag.* **13**(3), 108–117 (2006)
9. J. Behley, C. Stachniss, Efficient surfel-based SLAM using 3D laser range data in urban environments, in *Proceedings of Robotics: Science and Systems (RSS)* (2018)
10. P.J. Besl, N.D. McKay, A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* **14**(2), 239–256 (1992)
11. M. Bosse, P.M. Newman, J.J. Leonard, S. Teller, An ATLAS framework for scalable mapping, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2003), pp. 1899–1906
12. C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. Leonard, Past, present, and future of simultaneous localization and mapping: toward the robust-perception age. *IEEE Trans. Robot.* **32**(6), 1309–1332 (2016)
13. L. Carlone, R. Aragues, J. Castellanos, B. Bona, A linear approximation for graph-based simultaneous localization and mapping, in *Robotics: Science and Systems* (MIT Press, Cambridge, 2012)
14. J.A. Castellanos, J.M.M. Montiel, J. Neira, J.D. Tardós, The SPmap: a probabilistic framework for simultaneous localization and map building. *IEEE Trans. Robot. Autom.* **15**(5), 948–953 (1999)
15. N. Chebrolu, T. Labe, O. Vysotska, J. Behley, C. Stachniss, Adaptive robust kernels for non-linear least squares problems. *IEEE Robot. Autom. Lett.* **6**, 2240–2247 (2021)
16. Z. Chen, A. Jacobson, N. Sunderhauf, B. Upcroft, L. Liu, C. Shen, I. Reid, M. Milford, Deep learning features at scale for visual place recognition, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2017), pp. 3223–3230
17. X. Chen, A. Milioto, E. Palazzolo, P. Giguere, J. Behley, C. Stachniss, SuMa++: efficient LiDAR-based semantic SLAM, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019)
18. X. Chen, T. Labe, A. Milioto, T. Rohling, J. Behley, C. Stachniss, OverlapNet: a siamese network for computing LiDAR scan similarity with applications to loop closing and localization. *Autonom. Robot.* **46**, 61–81 (2021)
19. X. Chen, S. Li, B. Mersch, L. Wiesmann, J. Gall, J. Behley, C. Stachniss, Moving object segmentation in 3D LiDAR data: a learning-based approach exploiting sequential data. *IEEE Robot. Autom. Lett.* **6**(4), 6529–6536 (2021)
20. T.A. Davis, A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.* **30**(2), 165–195 (2004)
21. T.A. Davis, *Direct Methods for Sparse Linear Systems*, vol. 2 (SIAM, Philadelphia, 2006)
22. F. Dellaert, M. Kaess, Square root SAM: simultaneous location and mapping via square root information smoothing. *Int. J. Robot. Res.* **25**, 1181–1204 (2006)
23. J. Engel, J. Stuckler, D. Cremers, Large-scale direct SLAM with stereo cameras, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015)
24. C. Estrada, J. Neira, J.D. Tardos, Hierarchical SLAM: real-time accurate mapping of large environments. *IEEE Trans. Robot.* **21**(4), 588–596 (2005)

25. R. Eustice, H. Singh, J.J. Leonard, Exactly sparse delayed-state filters, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Barcelona (2005), pp. 2428–2435
26. M.A. Fischler, R.C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* **24**(6), 381–395 (1981)
27. U. Frese, P. Larsson, T. Duckett, A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Trans. Robot.* **21**(2), 1–12 (2005)
28. D. Gálvez-López, J.D. Tardós, Bags of binary words for fast place recognition in image sequences. *IEEE Trans. Robot.* **28**(5), 1188–1197 (2012)
29. G. Grisetti, C. Stachniss, W. Burgard, Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Trans. Robot.* **23**(1), 34–46 (2007)
30. G. Grisetti, C. Stachniss, S. Grzonka, W. Burgard, A tree parameterization for efficiently computing maximum likelihood maps using gradient descent, in *Proceedings of Robotics: Science and Systems (RSS)*, Atlanta (2007)
31. G. Grisetti, R. Kümmerle, C. Stachniss, W. Burgard, A tutorial on graph-based SLAM. *IEEE Intell. Transp. Mag.* **2**(4), 31–43 (2010)
32. G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, C. Hertzberg, Hierarchical optimization on manifolds for online 2D and 3D mapping, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2010)
33. G. Grisetti, R. Kümmerle, K. Ni, Robust optimization of factor graphs by using condensed measurements, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2012)
34. J.S. Gutmann, K. Konolige, Incremental mapping of large cyclic environments, in *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)* (1999)
35. D. Hähnel, W. Burgard, D. Fox, S. Thrun, An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas (2003), pp. 206–211
36. D. Hähnel, W. Burgard, B. Wegbreit, S. Thrun, Towards lazy data association in SLAM, in *Proceedings of the International Symposium of Robotics Research (ISRR)*, Siena (2003), pp. 421–431
37. P. Henry, M. Krainin, E. Herbst, X. Ren, D. Fox, RGB-D mapping: using Kinect-style depth cameras for dense 3D modeling of indoor environments. *Int. J. Robot. Res.* **31**(5), 647–663 (2012)
38. C. Hertzberg, A framework for sparse, non-linear least squares problems on manifolds. Master's thesis, University Bremen (2008)
39. W. Hess, D. Kohler, H. Rapp, D. Andor, Real-time loop closure in 2D LiDAR SLAM, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2016)
40. A. Howard, M.J. Matarić, G. Sukhatme, Relaxation on a mesh: a formalism for generalized localization, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2001)
41. M. Kaess, A. Ranganathan, F. Dellaert, iSAM: fast incremental smoothing and mapping with efficient data association, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Rome (2007)
42. M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, F. Dellaert, iSAM2: incremental smoothing and mapping with fluid relinearization and incremental variable reordering, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2011)
43. G. Karypis, V. Kumar, Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.* **48**(1), 96–129 (1998)
44. G. Klein, D. Murray, Parallel tracking and mapping for small AR workspaces, in *Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality* (2007)
45. K. Konolige, J. Bowman, J.D. Chen, P. Mihelich, M. Calonder, V. Lepetit, P. Fua, View-based maps. *Int. J. Robot. Res.* **29**(10), (2010)

46. R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, g2o: a general framework for graph optimization, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2011)
47. M. Labbé, F. Michaud, RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *J. Field Robot.* **36**, 416–446 (2019)
48. M.T. Lázaro, R. Capobianco, G. Grisetti, Efficient long-term mapping in dynamic environments, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018)
49. J.M. Lee, *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics, vol. 218 (Springer, Berlin, 2003)
50. F. Lu, E. Milios, Robot pose estimation in unknown environments by matching 2d range scans, in *IEEE Computer Vision and Pattern Recognition Conference (CVPR)* (1994), pp. 935–938
51. F. Lu, E. Milios, Globally consistent range scan alignment for environment mapping. *Autonom. Robot.* **4**, 333–349 (1997)
52. K. MacTavish, T.D. Barfoot, At all costs: a comparison of robust cost functions for camera correspondence outliers, in *2015 12th Conference on Computer and Robot Vision* (IEEE, Piscataway, 2015), pp. 62–69
53. M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, FastSLAM: a factored solution to simultaneous localization and mapping, in *Proceedings of the National Conference on Artificial Intelligence (AAAI), Edmonton* (2002), pp. 593–598
54. R. Mur-Artal, J.D. Tardós, ORB-SLAM2: an open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Trans. Robot.* (2017)
55. R. Mur-Artal, J. Montiel, J. Tardos, ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Trans. Robot.* **31**, 1147–1163 (2015)
56. J. Neira, J.D. Tardós, Data association in stochastic mapping using the joint compatibility test. *IEEE Trans. Robot. Autom.* **17**(6), 890–897 (2001)
57. R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, A. Fitzgibbon, KinectFusion: real-time dense surface mapping and tracking, in *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)* (2011)
58. K. Ni, F. Dellaert, Multi-level submap based SLAM using nested dissection, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2010)
59. A. Nüchter, K. Lingemann, J. Hertzberg, H. Surmann, 6D SLAM with approximate data association, in *Proceedings of the International Conference on Advanced Robotics (ICAR)* (2005), pp. 242–249
60. E. Olson, Robust and efficient robotic mapping. Ph.D. Thesis, MIT, Cambridge (2008)
61. E. Olson, Real-time correlative scan matching, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2009)
62. E. Olson, J. Leonard, S. Teller, Fast iterative optimization of pose graphs with poor initial estimates, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2006), pp. 2262–2269
63. E. Palazzolo, J. Behley, P. Lottes, P. Giguere, C. Stachniss, ReFusion: 3D reconstruction in dynamic environments for RGB-D cameras exploiting residuals, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019)
64. T. Pire, T. Fischer, J. Civera, P. De Cristóforis, J. Berlles, S-PTAM: stereo parallel tracking and mapping. *J. Robot. Autonom. Syst.* **93**, 27–42 (2017)
65. N. Radwan, A. Valada, W. Burgard, Vlocnet++: deep multitask learning for semantic visual localization and odometry. *IEEE Robot. Autom. Lett.* **3**(4), 4407–4414 (2018)
66. Y. Saad, *Iterative Methods for Sparse Linear Systems*, vol. 82 (SIAM, Philadelphia, 2003)
67. D. Schlegel, G. Grisetti, HBST: a hamming distance embedding binary search tree for feature-based visual place recognition. *IEEE Robot. Autom. Lett.* **3**, 3741–3748 (2018)
68. D. Schlegel, M. Colosi, G. Grisetti, ProSLAM: graph SLAM from a programmer's perspective, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2018)

69. T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, R. Siegwart, maplab: an open framework for research in visual-inertial mapping and localization. *IEEE Robot. Autom. Lett.* **3**(3), 1418–1425 (2018)
70. R. Smith, M. Self, P. Cheeseman, Estimating uncertain spatial relationships in robotics, in *Autonomous Robot Vehicles*, ed. by I. Cox, G. Wilfong (Springer, Berlin, 1990), pp. 167–193
71. C. Stachniss, J. Leonard, S. Thrun, *Springer Handbook of Robotics*. Simultaneous Localization and Mapping, 2nd edn., chap. 46 (Springer, Berlin, 2016)
72. B. Steder, G. Grisetti, W. Burgard, Robust place recognition for 3D range data based on point features, in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)* (2010)
73. N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, P. Corke, The limits and potentials of deep learning for robotics. *Int. J. Robot. Res.* **37**(4–5), 405–420 (2018)
74. C.J. Taylor, D.J. Kriegman, Minimization on the Lie group $SO(3)$ and related manifolds. Technical Report 9405, Yale University (1994)
75. S. Thrun, Y. Liu, D. Koller, A.Y. Ng, Z. Ghahramani, H. Durrant-Whyte, Simultaneous localization and mapping with sparse extended information filters. *Int. J. Robot. Res.* **23**(7–8), 693–716 (2004)
76. R. Triebel, P. Pfaff, W. Burgard, Multi-level surface maps for outdoor terrain mapping and loop closing, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2006)
77. R. Vincent, B. Limketkai, M Eriksen, Karto SLAM. <http://wiki.ros.org/karto>
78. T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, Kintinuous: spatially extended KinectFusion, in *RSS RGB-D Workshop* (2012)