Tales for Makers

Real-World Projects to Modify, Hack, and Reinvent

Enrico Miglino

Apress®

Maker Innovations Series

Jump start your path to discovery with the Apress Maker Innovations series! From the basics of electricity and components through to the most advanced options in robotics and Machine Learning, you'll forge a path to building ingenious hardware and controlling it with cutting-edge software. All while gaining new skills and experience with common toolsets you can take to new projects or even into a whole new career.

The Apress Maker Innovations series offers projects-based learning, while keeping theory and best processes front and center. So you get hands-on experience while also learning the terms of the trade and how entrepreneurs, inventors, and engineers think through creating and executing hardware projects. You can learn to design circuits, program AI, create IoT systems for your home or even city, and so much more!

Whether you're a beginning hobbyist or a seasoned entrepreneur working out of your basement or garage, you'll scale up your skillset to become a hardware design and engineering pro. And often using low-cost and open-source software such as the Raspberry Pi, Arduino, PIC microcontroller, and Robot Operating System (ROS). Programmers and software engineers have great opportunities to learn, too, as many projects and control environments are based in popular languages and operating systems, such as Python and Linux.

If you want to build a robot, set up a smart home, tackle assembling a weather-ready meteorology system, or create a brand-new circuit using breadboards and circuit design software, this series has all that and more! Written by creative and seasoned Makers, every book in the series tackles both tested and leading-edge approaches and technologies for bringing your visions and projects to life.

More information about this series at https://link.springer.com/bookseries/17311.

Tales for Makers

Real-World Projects to Modify, Hack, and Reinvent

Enrico Miglino

Apress®

Tales for Makers: Real-World Projects to Modify, Hack, and Reinvent

Enrico Miglino
Drongen, Belgium

ISBN-13 (pbk): 979-8-8688-0079-5 ISBN-13 (electronic): 979-8-8688-0080-1
https://doi.org/10.1007/979-8-8688-0080-1

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Miriam Haidara
Development Editor: James Markham
Project Manager: Jessica Vakili
Copy Editor: Kezia Endsley

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 NY Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on the GitHub repository: https://github.com/Apress/Software-Testing-For-Managers. For more detailed information, please visit https://www.apress.com/gp/services/source-code.

If disposing of this product, please recycle the paper

Table of Contents

About the Author

Enrico Miglino, a technical writer since the mid-1980s, has written hundreds of articles for magazines worldwide. He is a chemist, an engineer, a developer, and one of the most active creators in the global Maker community. He has won many international challenges with original and popular projects. His projects, based on Linux embedded platforms and microcontrollers, have been supported by the sponsorship of Arduino, Cypress, Nordic, Xilinx, AWS, Altium, Elegoo, Photon, and other companies. A top member of the Element14.com community, Enrico Miglino has taught webinars and workshops for three years on a wide range of Maker topics.

PART I

Where Is Tommy?

Ray Badmington and Tommy, his son, planned to spend two full weeks of their summer holidays enjoying the sun of the California beach.

Ray had been caring for his ten-year-old son on his own for almost a decade. He could still remember all the new things he had to learn—repairing toys, cooking, playing video games, becoming an expert in comic characters and heroes, and much more.

Regardless of their plans for a future as a family, after a quick illness, the two were forced to share the loss of their wife and mother. From a certain perspective, for Ray, Tommy represented the cure, as Ray had to concentrate on being helpful and supportive of his son. He did not have much time to think about this loss or leave space for sadness and depression.

Obliged by fate, Ray also changed his habits. To support the needs of his growing intelligent and curious son, he discovered how all the theories he had studied for years could easily apply to everyday life. It was only necessary to add some creativity and improvisation.

The theoretical engineer Ray Badminton, focused on astrophysics and other scientific investigations that result in abstruse concepts for ordinary people, changed his mind. To adapt quickly, he discovered a whole new world.

As time passed in his new role, he discovered new manual skills. He learned to create physical things, assemble, apply electronics, programming, and practice. This became a new way for him to face reality: Ray was learning the art of *making*.

Ray's transformation was complete in a few years. Always with Tommy beside him, he became one of the most creative makers ever.

3D printing, knowing almost any programming language, electronic design, recycling and upcycling appliances, woodworking, and building mechanics were only some of his newly discovered talents. Ray became an expert at making tools to solve challenging problems—hacking, rebirthing, and recycling.

"One of these days, this knowledge may save our lives," Ray often commented to Tommy when he saw his son fascinated by his experiments.

The two made a holiday plan and firmly decided to follow it: relaxing sunbaths and strolling along the beach. Ray and Tommy did this almost daily when arriving in Marina del Rey, California.

Ray spent most of his younger years in Green Bay, Wisconsin, his hometown along the coast of Lake Michigan. He remembered the Green Bay island, the beach, and the boy he was. Nothing to do with the sea and California beach.

Father and son walked along Los Angeles' coast that hot, sunny late morning. Both remained silent for a long time, walking on the seaside, enjoying the sun. It was still early, and in a couple of hours, that quiet place would be invaded by a noisy crowd, but now the only sound was coming from the sea.

A light breeze was caressing their skin while strolling along the sandy beach. Around 3pm—the idea was to stop for something to eat and a fresh drink—they left the beach to a concrete stair to reach Marina del Rey's north jetty, where they sat for a while on a concrete bench.

"Twenty-five thousand steps!," said Tommy as his smartwatch buzzed, marking a new daily record. "Look, Dad! We walked so far," he added. Enjoying the silence, the sea waves, and the landscape, they were considerably far from the hotel seafront where they were staying.

Tommy, in many ways, was a typical teenager like his school friends. He had a lot of friends, was a baseball player (despite being from Wisconsin, he was a Red Socks fan), was good at school, and was involved in a lot of activities.

Indeed, he was still young, but he always enjoyed spending time with his father, especially during these two holiday weeks, alone somewhere else but in their hometown. The situation became something like an adventure. Suddenly, Tommy was attracted by a strange mechanism on top of a pole rising from the concrete of the jetty next to the parking lot.

He stood up, moved by curiosity, and stepped closer for a deeper look. Ray was fully relaxed, observing the ocean waves lazily breaking up on the sand. He was thinking about decades ago when he was a few years younger than Tommy. In one of the cases when you remember a forgotten memory, Ray recalled a text on a leaflet that fascinated him for many seasons, "Visit BDTH 6159 and enjoy the fun!"

BDTH 6159 was one of the biggest amusement parks in America in the late 50s. Located in Los Angeles, it spanned between Marina del Rey and Venice Beach. He realized they were in the place he had dreamt of for a long time, a mythological place of his youth so far away to be almost impossible to reach.

Ray remembered the summers spent at his family's Lake Michigan cabin, loitering around and dreaming about that California amusement park. Nowadays, all that remained of that dream was the sand beach and a few restroom barracks.

"Dad, come on! Look, I found a cryptex." Tommy's scream brought Ray back to reality.

"It is not a cryptex; it is an old parking meter," Ray answered at first sight, walking in his direction for a closer look at Tommy's discovery. Its appearance, size, and shape resembled a late 50s parking meter, but looking closely, many of the details did not match.

"Uhm….," Ray mumbled, exploring the curious device on all sides.

"The coin slot is missing." Ray continued touching the device's metal surface, searching for hidden features. Inspecting the front side, Ray pointed out four rotary wheels with numbers from zero to nine.

"Maybe you are right. I have never seen one in person before, but this is very similar to a cryptex illustration I found in an article in *Scientific American* some years ago." Tommy was excited about the discovery and to be able to recognize it immediately. While Ray found other machine details that corresponded to his expectations, Tommy started playing with the rotating wheels.

"Strange—this old tool is still in excellent condition," Ray commented. "Most of them were destroyed or lost long ago. I am curious to know who put this device on the jetty. The installation seems recent."

While Tommy rotated the four numbered gauges, the arrow on the glass screen slowly moved clockwise. This occurred only when a certain number was selected, but neither of the two noted this detail. The last rotation of the fourth gauge moved the arrow to the rightmost side notch of the dial. After a loud crack, the arrow slowly moved to the left, accompanied by the sound of gears rotating inside the mechanism.

"Hey! Where do you go?" Ray said to Tommy. As the sound of the gears ended, Tommy walked back to the seaside without saying a word. Ray moved a step in his direction, but Tommy disappeared from his sight as if he had crossed an invisible border. Ray was speechless!

As when waking from a deep sleep and obliged to react quickly to an unexpected event, in a few seconds, Ray was forced to dismiss the lazy mood that had animated him until that moment.

He forced himself to find all his self-control to face what he recognized immediately was an emergency.

When faced with a sudden emergency, common sense suggests calling 911 for immediate help. For Ray, this was not a solution—that would mean police questioning, difficulty explaining the inexplicable, and losing precious time keeping the mind focused on the concern.

Ray, used to taking a pragmatic approach, immediately excluded this as a useless solution.

Instead of moving a single muscle, Ray started thinking about the best solution, trying to stay calm and positive. If he tried to explain what happened to someone else, he couldn't be trusted to tell the truth.

Furthermore, Ray was conscious that if anyone could pragmatically face a supernatural event like this, it was him: Ray Badmington.

"Let me see what really happened," Ray mumbled, thinking. "It must be something related to the cryptex."

"A cryptex is a device to keep a secret, which means that any obvious solution can't be the right one. I should find the correct code sequence for the cipher," Ray said, following the path of his thoughts while the sun was lazily starting to fall down the horizon.

During the next five hours, Ray tried all the most complex sequences he could imagine: the first four pi decimals, the numbers of Bernoulli, the four-color map solution, and the first four stochastic numbers. But every time he reached the fourth number, the arrow returned to the leftmost position on the gauge, and nothing happened.

This is when Ray thought facing the problem from a different perspective was necessary.

"Darn! Cryptex devices are used to hide words, not numbers, and this model can't be an exception". This seemed to Ray a great idea, but what kind of word could unlock the gears again? Alone in front of the diabolic engine, while the sunset was already tracking long shadows on the sand, Ray realized he was trying to solve a challenging problem, maybe the most complex of his life.

"Ray, relax!" He tried thinking about what happened between the passage that Tommy opened and the four-number code.

"I need to communicate with this machine to open the passage Tommy walked through." He felt excited, as if he were one step closer to solving the mystery.

"During the late 50s, people communicated through the telephone. Rotary telephones! The rotary dial showed numbers and characters. This might be the key."

Ray tried to remember the correct association: 2 = A, B, C, 3 = D, E, F. 4 = G, H, I, and so on. He thought it would work like the old SMS text input method on early cell phones. Ray remembered his first Nokia 3310, where he had to press the corresponding number one, two, or three times to get a character and write words.

Moved by excitation and curiosity, Ray rotated the four wheels to compose the number 6736, meaning the word OPEN.

The gauge indicator reached the extreme right again, and the rotating gears' well-known sound started again. The cryptex had been unlocked.

When the gears inside the engine stopped ticketing at the same point where Tommy disappeared, a big, dark-red curtain (that reminded Ray of a theater stage before a show) became visible. Without hesitation, Ray rushed immediately in that direction.

He crossed the curtain when the sun disappeared under the horizon line. The last sound Ray heard was the gauge indicator buzzing, returning to zero, and the curtain closing behind him.

CHAPTER 1

Make a Graphical Cryptex with Arduino Nicla

Including contributions of Furio Piccinini in developing the NodeJS and React servers.

Tip The word *cryptex* is a neologism coined by Dan Brown in his 2003 novel *The Da Vinci Code*, denoting a portable vault used to hide secret messages. Justin Kirk Nevins created the first cryptex physical model in 2004. (Source: Wikipedia)

This project replicates the cryptex described in the tale about Ray Badmington in this section. To make it, I followed a series of initial assumptions:

- The board must be used as a motion sensor.
- The cryptex interface must resemble the parking meter described in the first tale.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_1

- The Nicla motion sensor should interface with a PC via the USB-to-Serial connector.
- The cryptex interface must be accessible from the network.

To achieve these goals, I used different software technologies, as well as various programming languages depending on the context:

- C for the Nicla board firmware
- Python for the USB connection
- JavaScript (NodeJS and ReactJS) for the server side

I designed the projects described in this section to apply functional model-integrating technologies to the real world. These projects demonstrate how electronics with software and microcontrollers can solve complex problems. The model's architecture is also easily reusable in different contexts.

1.1. Arduino Nicla Sense ME

Developed by Arduino and Bosch, the Arduino Nicla Sense ME is one of the best sensor boards in the Arduino family. This thumb-sized device includes:

- 6-axis IMU (Inertial Measurement Unit) based on the Bosch BHI260AP:
  - 16-bit 3-axis accelerometer
  - 16-bit 3-axis gyroscope
  - A dual 32-bit CPU cure with a floating-point RISC processor and a 4-channel micro DMA controller

    - An external 2MB flash memory (QSPI connector)
    - The processor also features self-learning AI software and other firmware features for swim analytics, pedestrian status recognition, and so on
- High-performance pressure sensor based on the Bosch BMP390
- 3-axis magnetometer based on the Bosch BMM150
- Environmental sensing with AI capabilities based on the Bosch BME688:
    - Air pressure detection
    - Humidity level
    - Temperature
    - E-nose gas sensor (CO2 concentration and air quality index)

A dedicated ATSAMD11D14A-MUT microcontroller operates the Serial-to-USB bridge and the debugger interface. See Figure 1-1.

Figure 1-1. *The thumb-sized Arduino Nicla Sense ME board on its small container box*

The Nicla board also includes an ANNA-B112 BLE (Bluetooth Low Energy) module for wireless communication. The wide range of sensors on a single board makes the Nicla Sense ME an advanced environmental monitoring system. The complete board documentation is available on the official Arduino site (`https://docs.arduino.cc/hardware/nicla-sense-me`). Regardless of the board form factor—Arduino Nicla is the smaller one of the Arduino family—the PCB GPIO pins maintain the classical size of 2.45mm. The board easily fits into a breadboard, perfect for circuit experiments and prototyping. See Figure 1-2.

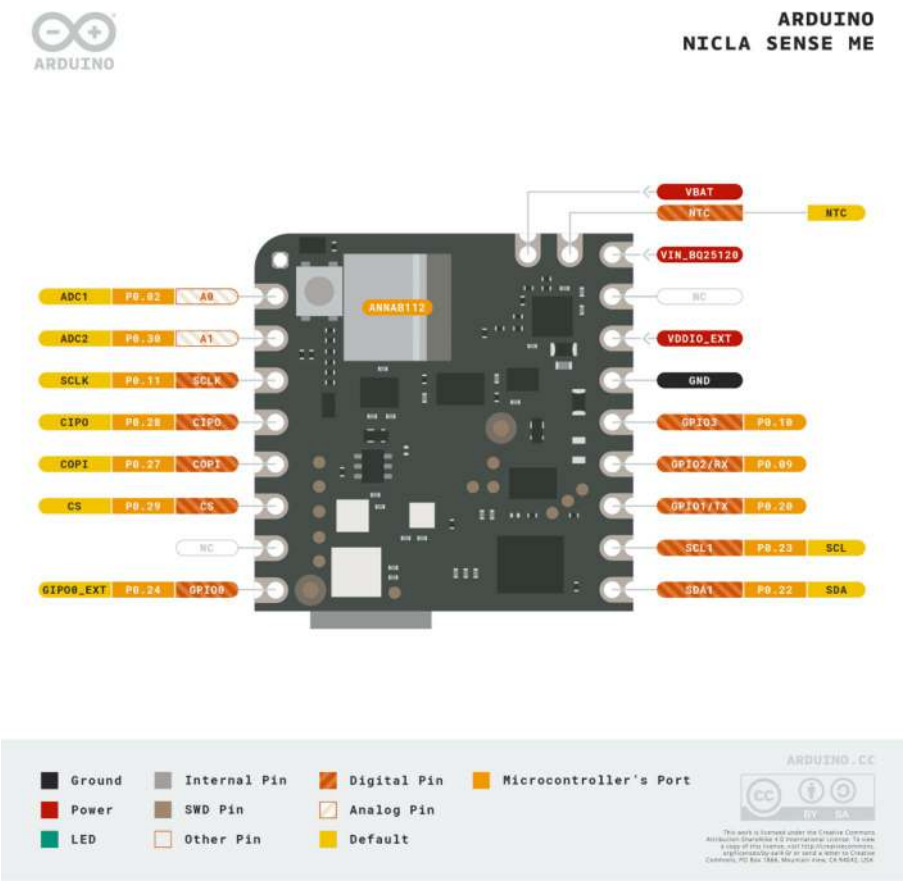

Figure 1-2. *The Arduino Nicla Sense ME GPIO pinout. (Credit: Arduino.cc)*

1.1.1. Nicla GPIO Pinout

In addition to its high-profile characteristics, the Nicla board includes an RGB LED and nine GPIO pins.

Not all the GPIO pins are available externally, depending on the board's usage. Most of the exposed pins are shared internally—like the UART pins shared with the sensors. Regardless of this limit, at least two digital I/Os are available with two analog A/D converters.

Indeed, the availability of the I2C serial bus allows the board features to expand by adopting many solutions, like using the board itself as a shield connected to other Arduino boards or adding an I2C GPIO expander.

1.1.2. It's Arduino!

I think that the development of the Nicla board is the result of a considerable effort to make it usable by makers and experimenters and easy to integrate with the Arduino ecosystem. There are another couple of details that make it unique and incredibly versatile.

Soldering the pins on the PCB to a pin rail Nicla Sense ME also works as a shield mounted on the MKR Arduino boards. Using, for example, the MKR WiFi 1010, you can expose a Nicla sensor directly to the Internet through a wireless connection. The connection between the two boards uses the I2C protocol.

A second option is the ESLOV connector. Designed by the Arduino team as an efficient IoT solution to connect sensors and small devices, this connector gives a further dimension to the possibilities of the board. Based on the I2C physical layer, this connector can communicate the Nicla board to—for example—an Arduino Portenta board.

More details on the ESLOV protocol specifications and the wiring schematics are available on the Arduino site (https://blog.arduino.cc/2016/09/28/eslov-is-the-amazing-new-iot-invention-kit-from-arduino/). Indeed, the Arduino Nicla Sense ME can be programmed with the Arduino IDE. Behind the easy C/C++ programming offered to the developers throughout the IDE, the Nicla runs a relatively complex Mbed-OS operating system.

1.1.3. Other Components

A few extra components must be added to the Nicla board to complete the hardware part of this project, resulting in a minimal circuit. In this case, the board is used as a motion sensor. In addition, I added a temporary push button connected to a GPIO input pin to instruct the board that the following detected movements should be interpreted as gestures. See Figure 1-3.

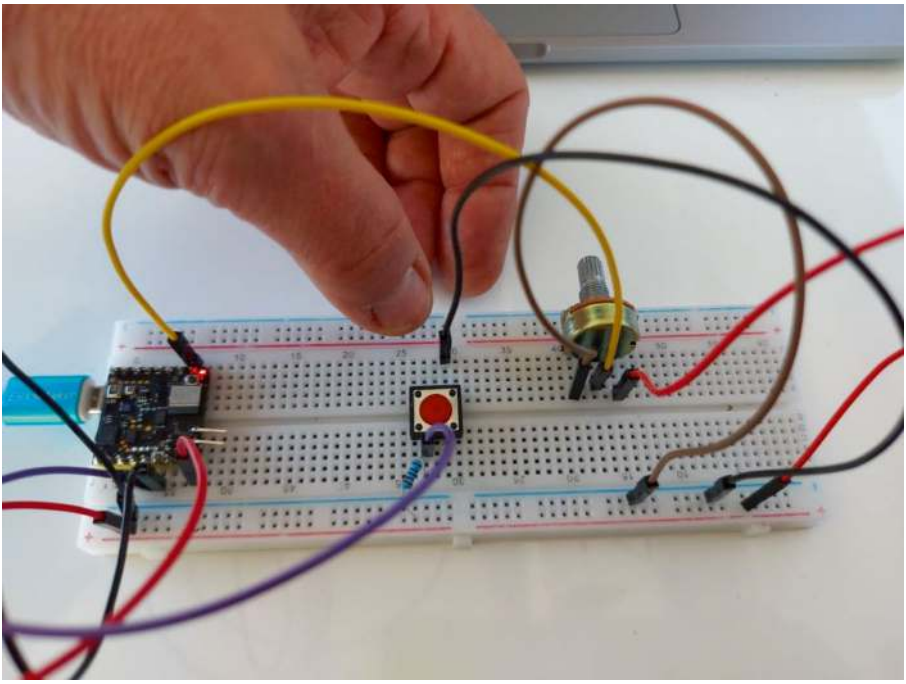

Figure 1-3. *The breadboard wiring the prototype circuit with the temporary push button and a potentiometer connected to the Nicla GPIO digital input 03 and the analog input 01*

To make the gesture control more comfortable, a potentiometer consents to calibrate the reactivity of the board. I used a pull-up resistor to wire the temporary push button in my first circuit draft. Still, it is unnecessary, as the GPIO pins already include the pulling resistors controlled by the software.

Tip All the sources and related files are available as open source, Apache license, on the GitHub repository.

1.2. Programming the Nicla Sense ME

The first version of the software I developed for the Nicla board was very challenging; the most complex part was the conversion of the motion sensors. The x-y-z data feed should be converted to a gesture indicating a direction.

I introduced the temporary push button in the circuit to limit the number of samples: gestures are detected from the sensor data only when the button is pressed.

Following this direction, I met a lot of issues when finding a better number of samples from the sensors and integrating the three-axis values. To speed up the development, I also added a potentiometer connected to one of the two A/D input ports of the Nicla. The analog value from the port, in the range of 1024 values, is used to fine-tune the sensor's response.

The Arduino software is built in two parts: the C source (the file with the `.ino` extension using the Arduino IDE) and a header file with the preprocessor parameters and the structures definition used by the program.

Tip I documented the sources as much as possible for better readability and understanding. The comments follow the *Doxygen* style to automatically create a fully documented local site with the Doxygen tool. It is an open source multiplatform project available online at https://doxygen.nl/.

1.2.1. The Header File

Listing 1-1 shows the header file contents. The second part of the file defines the structures used by the functions to organize the data.

Listing 1-1. The Header File Contents

```
/**
Polling frequency to read the sensors. This is the base
(minimum) reading frequency to which is added the trimmer fine-
tuning reading value, never bigger than MAX_POLL_FREQ
*/
#define POLL_FREQ 100
/** Maximum polling frequency (in milliseconds) */
#define MAX_POLL_FREQ 250
/** Minimum analog value read from potentiometer (needs
calibration) */
#define POT_MIN 0
/** Maximum analog value read from potentiometer (needs
calibration) */
#define POT_MAX 600
/**
```

```
Minimum delta value between two sequential readings to consider
the motion as a gesture. Delta reading is the absolute
difference between two sequential readings.
*/
#define DELTA_ROLL 10.00
/**
Minimum delta value between two sequential readings to consider
the motion as a gesture. See the DELTA_ROLL note
*/
#define DELTA_PITCH 10.00
/**
Minimum delta value between two sequential readings to consider
the motion as a gesture. See the DELTA_ROLL note
*/
#define DELTA_HEADING 10.00 // Not used
/** Number of sequential readings of the control button to
check its stable status */
#define MAX_BUTTON_READS 10000
/** Json keyword */
#define J_RIGHT "\"right\": "
/** Json keyword */
#define J_LEFT  "\"left\": "
/** Json keyword */
#define J_UP  "\"up\": "
/** Json keyword */
#define J_DOWN  "\"down\": "
/** Json keyword */
#define J_OPEN "{"
/** Json keyword */
#define J_CLOSE "}"
```

```
enum ledColors {
  RED,
  GREEN,
  BLUE,
  OCRE,
  LIGHTBLUE
};
/**
 Defines all the parameters to manage the sensors state and
 convert them dynamically to the corresponding motion sensor
 directions.
*/
```

Note Every gesture value is a two-position array to avoid creating the JSON object when there are multiple readings with the same pattern. The direction state can be a single value, as every gesture should correspond to a single action.

```
struct gestures {
  //! Direction current state
  bool right[2];
  //! Direction current state
  bool left[2];
  //! Direction current state
  bool up[2];
  //! Direction current state
  bool down[2];
};
/**
```

```
Last couple of reading values for the sensor parameters
involved in the gesture control (roll, pitch, heading).
*/
struct niclaSensors {
  float roll[2];
  float pitch[2];
  float heading[2];
};
/**
RGB LED color parameters
*/
struct rgbLEDColors {
  int red;
  int green;
  int blue;
};
```

1.2.2. The Program Source

According to the standard architecture of the Arduino IDE *Sketch*, the source architecture is divided into four main parts:

1. The `includes` (headers and library headers)
2. The `setup()` initialization function
3. The `loop()` function, which defines the business logic of the program
4. The program functions

For better readability, the `loop()` function exposes the logic while the tasks are defined in separate functions.

Note In the following code block, there are three preprocessor definitions (_DEBUG_, _LOGGING_, and _CALIBRATION_) that can be optionally defined. _DEBUG_ enables or disables the output to the serial terminal's extra information. _LOGGING_ adds logging information to the serial terminal, but the output data is no longer usable. _CALIBRATION_ enables or disables the readings from the A/D input pin to fine-tune the sensitivity of the readings.

Listing 1-2 shows the first part of the program. Note that some of the global variables are based on the structure types defined in the header file.

Listing 1-2. The First Part of the Program

```
#include "Arduino.h"
#include "Nicla_System.h"
#include "Arduino_BHY2.h"
#include "ParkingMeter.h"
#include "Streaming.h"
#undef _DEBUG_
#undef _LOGGING_
#undef _CALIBRATION_
/**
Sensors class instance - orientation
*/
Sensor device_orientation(SENSOR_ID_DEVICE_ORI);
/**
Orientation object to retrieve the pitch, roll and heading
punctual values.
*/
SensorOrientation orientation(SENSOR_ID_ORI);
/**
```

```
Calibration pin (analog), connected to the 50K potentiometer
*/
const int pinCalibration = A0;
/**
Button digital pin. The motion detection is working only when
the red button is pressed.
*/
const int pinButton = GPIO3;
/**
Status of the five gestures, according to the last sensor
reading.
The sensor is used to read the dominant gesture direction, if
any, after the last reading. IF a gesture is recognized(pitch,
roll, heading) the corresponding gesture direction is set
to true.
*/
#include "Arduino.h"
#include "Nicla_System.h"
#include "Arduino_BHY2.h"
#include "ParkingMeter.h"
#include "Streaming.h"
#undef _DEBUG_
#undef _LOGGING_
#undef _CALIBRATION_
/**
Sensors class instance - orientation
*/
Sensor device_orientation(SENSOR_ID_DEVICE_ORI);
/**
Orientation object to retrieve the pitch, roll and heading
punctual values.
```

```
*/
SensorOrientation orientation(SENSOR_ID_ORI);
/**
Calibration pin (analog), connected to the 50K potentiometer
*/
const int pinCalibration = A0;
/**
Button digital pin. The motion detection is working only when
the red button is pressed.
*/
const int pinButton = GPIO3;
/**
Status of the five gestures, according to the last sensor
reading.
The sensor is used to read the dominant gesture direction,
if any, after the last reading. *IF* a gesture is
recognized(pitch, roll, heading) the corresponding gesture
direction is set to true.
*/
```

Note The sensor reading can be bigger or smaller with respect to the previous reading. When the difference is greater than the minimum delta, the corresponding value is set to `true`. Only one value can be `true` after a motion is detected.

```
gestures motionDirection;
/**
Double value readings of the sensors updated every loop cycle
*/
niclaSensors sensorsReading;
```

Listing 1-3 shows the `setup()` and `loop()` functions. `setup()` is run once when the board is powered or reset, then `loop()` runs indefinitely to manage the business logic of the program.

Listing 1-3. The setup() and loop() Functions

```
/**
Initialization
*/
void setup()
{
  pinMode(pinButton, INPUT);
  /** Serial initialization */
  Serial.begin(19200);
  /** Start the board */
  BHY2.begin();
/**
Initialize the orientation component of the sensor
*/
  orientation.begin();
  // Start RGB Led
  nicla::begin();
  nicla::leds.begin();
  initReadings();
  initMotionDirection();
  flashLed();
}
```

Note `setup()` includes the calls to the Nicla initialization functions. These are calls to C++ libraries that are part of the Nicla development system, which is installed when the hardware board is installed in the Arduino IDE.

```
/**
Application main loop
*/
void loop()
{
#ifdef _CALIBRATION_
  // For potentiometer calibration only. Need a serial
     output monitor
  calibratePot();
  // waits 2 milliseconds before the next loop for the analog-
     to-digital
  // converter to settle after the last reading:
  delay(500);
#else
  static auto pollFreq = millis();
  /** Adjusted POLL_FREQ value */
  unsigned long pFrequency;
  // Adjusts the polling frequency with the fine tuning value
Note: The line of code below reads the frequency every sampling
cycle, adding the calibration value (from the analog input pin)
  pFrequency = POLL_FREQ; + getCalibration();
  // Update function should be continuously polled
  BHY2.update();
  // Checks for the frequency before updating the sensor data
  if(millis() - pollFreq >= pFrequency) {
```

```
//    if(checkButton() == true){
    if(true){
      // Checks if a motion has been detected and creates
         the json
      // string
      if(checkDeltaToMotion()){
#ifdef _LOGGING_
        Serial.println("Motion detected");
        Serial.print(" pitch :");
        Serial.print(sensorsReading.pitch[1]);
        Serial.print(" roll :");
        Serial.print(sensorsReading.roll[1]);
        Serial.println("Motion flags: roll");
        Serial.print("left ");
        // The motion direction considered is only the
           current last
        // reading and the previous one is ignored, as it
           has already
        // been logged in the previous loop view.
        Serial.print(motionDirection.left[1]);
        Serial.print(" right ");
        Serial.print(motionDirection.right[1]);
        Serial.print(" up ");
        Serial.print(motionDirection.up[1]);
        Serial.print(" down ");
        Serial.println(motionDirection.down[1]);
#endif // Debug
        sendJson();
        // Resets the time counter
        pollFreq = millis();
```

```
      } // Delta motion
    } // Check Button
  } // Timer
#endif // Calibration
}
```

Listing 1-4 shows that all the subtasks of the program are organized in separate functions. This approach is easier to manage and more readable. In a more complex situation, it`s also possible to create separate classes—included in the main program—that cover specific tasks.

Listing 1-4. Subtasks of the Program Organized in Separate Functions

```
/**
Initializes the sensor readings.
*/
void initReadings() {
  int j;
  for(j = 0; j < 2; j++) {
    sensorsReading.roll[j] = 0;
    sensorsReading.pitch[j] = 0;
    sensorsReading.heading[j] = 0;
  }
}
/**
Initializes the gestures pattern to false when starting
*/
void initMotionDirection() {
  int j;
  for(j = 0; j < 2; j++) {
    motionDirection.right[j] = false;
    motionDirection.left[j] = false;
```

```
    motionDirection.up[j] = false;
    motionDirection.down[j] = false;
  }
}
/**
Saves the current gestures pattern to the previous gesture and
initializes the new gestures to false before setting the last
pattern read from the sensors.
*/
void swapMotionDirection() {
    motionDirection.right[0] = motionDirection.right[1];
    motionDirection.right[1] = false;
    motionDirection.left[0] = motionDirection.left[1];
    motionDirection.left[1] = false;
    motionDirection.up[0] = motionDirection.up[1];
    motionDirection.up[1] = false;
    motionDirection.down[0] = motionDirection.down[1];
    motionDirection.down[1] = false;
}

/**
Compares the four directions of the motionDirection pattern.
If at least one is different, a new motion is detected, else
it is the same pattern of the previous. If none of the new
motionDirection fields is not zero, the motionDirection should
be excluded as well.
*/
bool isNewMotionDirection() {
  bool isMotion;
  int j;
  isMotion = false;
```

```
  if(motionDirection.right[0] != motionDirection.right[1])
    if(motionDirection.right[1] == true)
      isMotion = true;
  if(motionDirection.left[0] != motionDirection.left[1])
    if(motionDirection.left[1] == true)
      isMotion = true;
  if(motionDirection.up[0] != motionDirection.up[1])
    if(motionDirection.up[1] == true)
      isMotion = true;
  if(motionDirection.down[0] != motionDirection.down[1])
    if(motionDirection.down[1] == true)
      isMotion = true;
  return isMotion;
}
/**
Updates the readings and swaps the last value to the
previous reading
*/
void updateReadings() {
  // Moves last reading to previous
  sensorsReading.roll[0] = sensorsReading.roll[1];
  sensorsReading.pitch[0] = sensorsReading.pitch[1];
  sensorsReading.heading[0] = sensorsReading.heading[1];
  // Acquires the current value
  sensorsReading.roll[1] = orientation.roll();
  sensorsReading.pitch[1] = orientation.pitch();
  sensorsReading.heading[1] = orientation.heading();
  // If we are on the first reading, duplicates the
     sensor value
  if(sensorsReading.roll[0] == 0)
    sensorsReading.roll[0] = sensorsReading.roll[1];
```

```
  if(sensorsReading.pitch[0] == 0)
    sensorsReading.pitch[0] = sensorsReading.pitch[1];
  if(sensorsReading.heading[0] == 0)
    sensorsReading.heading[0] = sensorsReading.heading[1];
}
/**
Calculates the three delta values between the last two
readings, then sets the motionDirection structure accordingly.
\note The function does not implement multiple motions
together. The first detected valid direction is set.
@return true if a motion has been detected, otherwise
return false.
*/
bool checkDeltaToMotion() {
  /** Delta for last roll value */
  float dRoll;
  /** Delta for last pith value */
  float dPitch;
  /** Delta for last heading value (not used) */
  float dHeading;
  /**
  Motion detection return flag
  */
 bool motionDetected = false;
  // Swaps the last reading and initializes the new one
  swapMotionDirection();
  // Updates the reading status
  updateReadings();
  // Calculates the delta values
  dRoll = sensorsReading.roll[1] - sensorsReading.roll[0];
  dPitch = sensorsReading.pitch[1] - sensorsReading.pitch[0];
```

```
  dHeading = sensorsReading.heading[1] - sensorsReading.
  heading[0];
#ifdef _LOGGING_
  Serial.print("Delta_pitch :");
  Serial.print(dPitch);
  Serial.print("Delta_roll :");
  Serial.println(dRoll);
#endif
  // Checks for delta ranges. Gets the biggest delta between
     roll and pitch
  if( (abs(dRoll) >= DELTA_ROLL) ) { // &&  (abs(dRoll) >
  abs(dPitch)) ){
    // Decides the roll direction for left/right
    if(dRoll > 0)
      motionDirection.right[1] = true;
    else
      motionDirection.left[1] = true;
    motionDetected = true;
  }
  else {
    if(abs(dPitch) >= DELTA_PITCH) {
        // Decides the pitch direction for up/down
      if(dPitch > 0)
        motionDirection.down[1] = true;
      else
        motionDirection.up[1] = true;
      motionDetected = true;
    }
  }
  return motionDetected;
}
```

```
/**
Creates the Json string and sends it to the serial.
\note The Json string object is created only if the current
gestures pattern is not the same as the last read, otherwise
it's part of a multiple reading and should be ignored.
The pattern is also ignored if it doesn't have any direction
set to true
*/
void sendJson(){
  // Checks if it is a new pattern (different from all false)
  if(isNewMotionDirection()) {
    Serial << J_OPEN << J_RIGHT << motionDirection.right[1]
<< ", " <<
            J_LEFT << motionDirection.left[1] << ", " <<
            J_UP << motionDirection.up[1] << ", " <<
            J_DOWN << motionDirection.down[1] <<
            J_CLOSE << '\n' << endl;
  }
}
/**
Reads the analog value from the potentiometer and maps it to
the min/max range of the fine tuning for the sensor readings.
\returns The calibration value in milliseconds to adds to the
current milliseconds frequency
value.
*/
unsigned long getCalibration() {
  /** Potentiometer analog reading */
  int potCalibration;
  /** frequency calibration value */
  int freqCalibration;
```

```
  // reads the calibration pin
  potCalibration = analogRead(pinCalibration);
  // Maps the reading value
  freqCalibration = map(potCalibration, POT_MIN, POT_MAX, 1,
  (MAX_POLL_FREQ - POLL_FREQ));
  return freqCalibration;
}
/**
Checks the button status and changes the RGB LED color,
accordingly.
\note To avoid erratically readings the status of the digital
pin is read MAX_BUTTON_READS times and if the value is not
stable the button is considered in off state.
\returns true if the button is pressed, else returns false.
*/
bool checkButton() {
  /** Button status */
  bool isButton;
  /** digital pin readings accumulator */
  int buttonState;
  int j;
  buttonState = 0;
  for(j = 0; j < MAX_BUTTON_READS; j++) {
   if(digitalRead(pinButton) == HIGH) {
     buttonState++;
   }
  }
#ifdef _DEBUG_
  Serial << "Button " << digitalRead(pinButton) << endl;
#endif
  if(buttonState == MAX_BUTTON_READS) {
```

```
    setColor(true, BLUE);
    isButton = true;
  }
  else {
    setColor(true, RED);
    isButton = false;
  }
#ifdef _DEBUG_
  Serial << "isButton " << isButton << " - buttonState " <<
  buttonState << endl;
#endif
  if(isButton == false){
    sensorsReading.roll[1] = orientation.roll();
    sensorsReading.pitch[1] = orientation.pitch();
    sensorsReading.heading[1] = orientation.heading();
  }
  return isButton;
}
/**
Sets the predefined colors to the RGB LED. If the isOn
parameter is set to true, the RGB LED is powered on else it is
set to off.
\param isOn Flag to set the LED on or Off.
\param color The led color. Only if isOn is true, the setting
has effect
*/
void setColor(bool isOn, ledColors color){
  /** Color structure with colors patterns */
  rgbLEDColors rgbColors;
    // Checks if the flag is set, else disables the RGB LED
    if(isOn){
```

```
switch(color) {
  case   RED:
  rgbColors.red = 0xff;
  rgbColors.green = 0x01;
  rgbColors.blue = 0x01;
  break;
  case GREEN:
  rgbColors.red = 0x00;
  rgbColors.green = 0xff;
  rgbColors.blue = 0x05;
  break;
  case BLUE:
  rgbColors.red = 0x01;
  rgbColors.green = 0x01;
  rgbColors.blue = 0xff;
  break;
  case OCRE:
  rgbColors.red = 0xa8;
  rgbColors.green = 0x7b;
  rgbColors.blue = 0x14;
  break;
  case LIGHTBLUE:
  rgbColors.red = 0x00;
  rgbColors.green = 0x08;
  rgbColors.blue = 0x14;
  break;
  default:
  rgbColors.red = 0xff;
  rgbColors.green = 0xff;
  rgbColors.blue = 0xff;
  break;
}
```

```
      // Sets the LED to color
      nicla::leds.setColor(rgbColors.red, rgbColors.green,
      rgbColors.blue);
    } else {
      // Disables the LED
      nicla::leds.setColor(off);
    }
}
/**
Flashes the LED to start colors at initialization
*/
void flashLed() {
  setColor(true, RED);
  delay(250);
  setColor(true, GREEN);
  delay(250);
  setColor(true, OCRE);
  delay(250);
}
/**
Calibrates the potentiometer. When this function runs the
application features are disabled.
\note This calibration function should be used to detect max
and min values of the potentiometer analog read to set the two
*/
#ifdef _CALIBRATION_
void calibratePot(){
  int sensorValue = 0;
  // reads the analog value:
  sensorValue = analogRead(pinCalibration);
  // prints the results to the Serial Monitor:
```

```
  Serial.print("potentiometer = ");
  Serial.println(sensorValue);
}
#endif
```

1.2.3. A Slice of Python

The gestures coming from the Nicla Sense ME board, packed in a JSON string, should be sent to a computer through the USB-serial cable for further processing.

The multiplatform high-portability of Python is the best solution: easy to program, the Python module's role is to manage the hardware connection on one side and communicate to the NodeJS backend server on the other.

As I explain in the following paragraphs, the choice of backend and frontend servers increases the portability of the application. Of course, all the components run on the same laptop in my development platform. Still, I can expand the applications simply by moving the servers to other computers, whether running on a local network or the cloud.

From this perspective, the only software module that must reside on the computer connected to the Nicla board is the Python module. It is not a difficult task and is designed to run as a task from the terminal, starting as a small service or a background activity consuming a minimal amount of the resources.

I used a Ubuntu 20.4 LTS virtual machine in this project. Indeed, it can run on any other device, including small, embedded Linux boards supporting the USB connection and access to the Internet.

1.3. The Cloud-ready Servers

To implement the cryptex mechanism, I used two separate servers: the frontend and backend. As a matter of fact, it would be possible to use only one server to do both tasks, thereby introducing a more challenging development architecture. This second option requires that the frontend server also to cover all the backend tasks with a global reduction of performance.

With this double-server architecture, it is possible to access the frontend interface by multiple users and interact with the cryptex decoding process from many remote locations, equipped with a Nicla Sense ME and running the Python module.

The server APIs are REST calls executed with the `POST` method. Regardless of the complexity of the data—in this case, I manage a little information—it is best practice to use the `POST` method to avoid exposing the call's contents in the URL with the `GET` method.

The NodeJS and React servers have been developed with the invaluable help of my friend and rewarded maker, Furio Piccinini (`@furio` on `element14.com`).

1.3.1. The NodeJS Backend

Every time the Python module detects a new motion, it calls the NodeJS server that exposes an API. According to the user interface, there are four possible actions:

- Right
- Left
- Up
- Down

The server validates the action when the Python program calls the NodeJS server API with the direction information. After validation, it triggers a new event, calling the React server through another API with the direction.

The NodeJS server is data-agnostic. This makes it possible for two different clients—connected from two locations—to interact with the cryptex interface.

1.3.2. The React Frontend

As mentioned, the frontend server developed with ReactJS can run on a separate computer than the NodeJS. Therefore, a REST API triggers the NodeJS server to receive inputs, updating the user interface accordingly.

The interface is designed with multiple transparent PNG images, which mimic a vintage parking meter with some modifications. The gauge, showing the remaining parking time in the tool, shows the correctness of the introduced numbers instead. A left-right gesture with the Nicla changes the selection of the digit, while an up-down movement increases or decreases the currently selected digit. As a result, every number can change from 0 to 9.

The advantage of using ReactJS is the incredible number of open source modules available in the repositories. I used one of the many components in the `npm` libraries to design the gauge indicator and the other interface components.

1.4. Using Postman to Test the Server APIs

Creating RESTful APIs to communicate with servers is a polite way to develop server-based software, but the architecture may be challenging. For example, working with two servers in the workflow described in the previous paragraphs can be a painful task to debug.

It has been years since I have dramatically improved the final revision and setup of the servers using Postman (`postman.com`): it is a commercial tool including an excellent free plan that requires registration. The few limitations of the Postman free plan make it worthwhile to any maker project and small teams.

Postman can be used from a browser or by downloading the free desktop application. It provides much more than basic API testing—it is possible to store data for every API, use any kind of call, and create a collection of APIs for every project to test the whole server usage.

In addition, other features simplify the development of the clients to connect to the server:

- When an API test works fine, it is possible to download the *scriptlet* of its usage in the most common languages, including Python, Java, JavaScript, and Go.
- Adding an accurate description to the API calls and using self-explanatory test data, it is easy to create complete API documentation for web publishing or for creating a PDF document.
- The collection can be exported locally in a reusable JSON file.

PART II

The Great Amusement Park

As Ray stepped through the curtain, a wave of unfamiliarity washed over him as if he had been transported to a realm beyond his comprehension.

He couldn't describe the sensation, but he was sure he was in another place or maybe even another world.

Surprisingly, this quasi-darkness, a dimly lit space that seemed to be neither fully dark nor fully light, was not scary, but strangely familiar. Ray tried to remember if there was some connection to his past life, some forgotten dream, or one of his kid's nightmares, but the past didn't help him.

After a few seconds, his eyes adapted to the light. He was in a large corridor. A long, straight perspective where it was impossible to distinguish the contours. The side walls were clean with no signs or indications. There was only one option: going ahead.

Counting his steps—about 70cm each—Ray calculated the location of a door approximately every 100 meters.

All the doors were identical, and he encountered no resistance when opening them. His curiosity grew with every door he passed.

The darkness was interrupted along the corridor by yellowish, low-light lamps on the center top of the ceiling. Ray stopped counting the doors after the first several hundred. After crossing some doors, he also tried to reopen one behind him. Looking in the opposite direction, the scenario

was always the same—a dark corridor with a door 100 meters away. When he tried to leave one of the doors open behind him before reaching the next, the door closed softly, emitting a silent "click."

After countless openings, he finally saw EXIT sprayed on the left wall. This was the first change after a long time that sparked his curiosity.

Ray understood that he had walked through the last door when, after opening it, a bright light suddenly hurt his eyes. Walking the long corridor path, the force pushing him door after door, surrounded by a surreal silence, he had become more and more alarmed was that his beloved son Tommy might be in danger, something Ray couldn't quite imagine.

Step after step, this worry grew in his mind, replacing the feeling of confidence transmitted by the place.

The sudden change of situation, the change of light, and the echo of a familiar noise—a sound that Ray couldn't quite place but felt he had heard before—helped Ray recover a more optimistic feeling.

As he adapted to the new and more comfortable lighting conditions, Ray was curious about his destination. After this long walk in the dark, he had no doubt that he had arrived.

The cryptex, a mysterious puzzle box, the curtain, and the countless doors—at least that is what he supposed—were all part of an uncommon path, a passage to reach another place. Curious and worried at the same time, Ray couldn't wait to see what this new world held.

Ray looked around, and what he saw reminded him of his younger dreams, mixed with a sense of déjà vu.

At first sight, he saw an abandoned amusement park—the ideal place for a Rob Zombie movie. There were many carousels, rides, odd buildings, barracks, and shooting galleries.

Here and there, semi-abandoned mobile homes were parked between carousels, judging by their state, maybe for decades. Regardless of its rundown appearance, Ray can't avoid feeling attracted and fascinated by the soul of the place.

"It is like a sleeping giant," Ray murmured. At that moment, searching for Tommy, the biggest priority in his mind, assumed a different meaning. He was almost sure his son was somewhere around, and nothing dangerous could happen to him.

Ray felt the sensation of being alone-not-alone. His thoughts were only accompanied by the whisper of the wind.

"Hello, sir. Can I help you?"

The voice came from his left, from a woman. Her presence added a surreal touch to the scene, the inexplicable inside the inexplicable. Meanwhile, she walked closer to him.

"Hello, my name is Ray. I arrived..." Ray could not tell from where and how he arrived, so he avoided explaining much more about it.

"I am searching for my son Tommy, but..." Ray was not sure how to explain why he thought Tommy was there; it was just a supposition.

"Sir, if you are searching for Tommy, that guy who entered the park a few hours ago, you can search for him yourself." Answered the woman, "Just walk around; he should still be visiting the park. I suppose he is enjoying the attractions."

Enchanted by her sweet voice, only the search of Tommy made him desist to continue a conversation with her.

At that precise moment, Ray thought he had met his destiny.

"Sorry to be so rude," the woman said.

"I am Sonya. It's been such a long time since I've talked with someone that I almost forget how to behave with guests."

Despite being interested in Sonya, Ray was impatient to leave the conversation and start his search. Without speaking a word, she seemed to understand Ray's emotions perfectly.

"I suggest you start searching for your son down that lane," Sonya said, pointing to an entrance about 40 meters away.

"Thank you!," said Ray, moving immediately in that direction.

"See ya!," Sonya said as Ray moved away quickly. Ray waved his hand without turning back.

WELCOME TO BDTH 6159, THE BEST AMUSEMENT PARK IN THE WORLD!

Now that Ray was close to the lane, he could read the faint remains of the amusement park entrance, a symbol of the park's former glory and the start of his journey. He eventually also saw the ticket booth for visitors.

The characters and those words, an unforgettable memory for him, had the power to change his mind. It seemed Ray had forgotten his concern.

"This is a dreamy place!," He thought. His adventure was really starting now, and he felt optimistic, captured by the fascinating, decadent scenario the park was offering to his eyes. The once vibrant rides now stood in eerie silence, their paint peeling and their lights dimmed. This created a hauntingly beautiful sight.

As Ray continued down the lane, he noticed how time had weathered the structures around him. The peeling paint and rusted metal gave the amusement park an almost nostalgic charm, a relic from a forgotten era. He wandered farther into the park, finding a large, semi-abandoned carousel.

He paused, captivated by the melancholic beauty of the ride, imagining the laughter and joy it had once brought to children and families.

Ray's thoughts drifted to his own childhood. He could almost hear the distant echoes of music and the excited chatter of visitors. The memories were only a reminder of simpler times. His reverie was broken by the sound of footsteps echoing through the empty park. Ray turned, expecting to see Sonya or perhaps another visitor, but the path behind him was empty. He felt a chill run down his spine, the feeling of being watched creeping up on him. Shaking off his unease, he pressed on, determined to find Tommy.

As he walked, Ray noticed more signs of recent activity. Footprints in the dust, a half-eaten candy apple on a bench, and a freshly discarded ticket stub. These clues reassured him that Tommy was nearby and that he was not alone in this strange place. Ray's journey through the park took

him past a series of game booths, their long-faded prizes covered in grime. Amidst the decay, there was a sense of resilience, a spirit that refused to be completely extinguished.

Eventually, Ray found himself at the entrance of another large structure, its faded sign reading, "The Fun House."

The whimsical design of the building contrasted sharply with its state. Ray hesitated momentarily, then decided to venture inside, hoping to find more clues about Tommy's whereabouts. The interior of the Fun House was a maze of mirrors and twisted corridors. Ray's reflection seemed to follow him at every turn, creating an unsettling illusion of being surrounded by countless versions of himself. The air was filled with the faint smell of old popcorn and sawdust, adding to the surreal atmosphere.

Each step echoed loudly, breaking the silence that enveloped the park. He knew he was getting closer to finding Tommy, but the journey proved to be more challenging than he had anticipated.

Ray started following the lane's path. A series of barracks, their once vibrant colors now faded, lined the right side, stretching until a curve obscured the rest.

Some of the barrack windows on the lane side were covered with dust and dirt. Ray was attracted by the third barrack window, from which an irregular flashing light was visible. At first sight, it seemed to him the flickering of a neon near the end of its life, but approaching closely, Ray saw that the light was also changing color.

"There is a screen inside there," he thought.

Ray was speechless, seeing better inside after cleaning the dust from the screen with his hand.

"Tommy!," he screamed. Then he started compulsively knocking on the glass. What he saw was beyond his imagination; his son was inside a futuristic room, playing in front of a game console.

Tommy was totally focused on the animation on the screen, ignoring what was happening around him.

Ray continued knocking and moving his hands in front of the window, trying to catch Tommy's attention, until he saw his son standing up and leaving the room through a door on the opposite side.

Ray was not one to be easily discouraged, but for the second time in a few minutes, he was speechless.

With more questions than answers, Ray found a door near the corner of the barracks he had not noted before, as he was too excited for what he saw through the window.

The door was unlocked, but as he stepped inside, the room offered the same spectacle he had seen through the other windows.

A silent single-room space with no lights, except for a console screen.

"He was playing here just a few seconds ago!," thought Ray. He saw other similar game consoles, all protected by transparent plastic covers. Dust was everywhere.

Almost confused by what he experienced, Ray went out of the barracks.

Sonya was beside the door. The woman's smile—Ray imagined she was a woman, but it was impossible to say her age—and her sweet and calm movements had the power to change his mood immediately.

"What are you searching for, Ray?," asked Sonya with a low and warm voice.

"I am sure I saw Tommy there, but I found only an empty, silent room when I went inside."

"If you saw your son through the window, it means he was there. These windows use delayed glass," she continued like it was the most natural and obvious thing. Ray assumed an interrogative expression, and so she continued.

"It is a special material with the property to capture images on one side and release them slowly, delayed in time, on the other. What you watched through the window happened some hours before."

Ray's understanding did not contribute to lessening his shock.

"Ray, don't be afraid and go ahead. Follow the path and enter the park; you will soon meet your son," concluded Sonya, putting a hand on his shoulder.

He felt that her gesture transmitted the warm sound of her voice to his body.

Ray turned to Sonya, but he was alone again.

Trying to observe and memorize every detail around him, he cautiously started down the lane again until he disappeared behind the corner of the barracks row.

CHAPTER 2

How to Maintain a Secure Rollercoaster

Contribution of Jan Cumps, Element14.com community member.

Figure 2-1. *Iconic representation of the big rollercoaster of the BDTH6159 amusement park created with DALL-E*

The most iconic representation of the amusement park is the rollercoaster: a ride that mixes pure enjoyment and terror and is frequently the most popular attraction in the park (see Figure 2-1). The rollercoaster has also been the subject of horror and romance novels and movies.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_2

Regrettably, some of these thrilling rides have a darker side in the real world. Tragedies often occur due to a lack of maintenance.

Indeed, this project shows how a specific family of sensors can contribute to granting security in this kind of structural engineering.

Note When I started writing this book, in collaboration with the Element14.com engineering community, we launched a challenge between the community members for a project related to the story of Ray and Tommy in the BDTH6159 amusement park. This project is the selected one, designed by my friend and maker, Jan Cumps, a top member of the Element14 community. At that time, he was thinking about how to test a vibration sensor to detect low-frequency vibrations, like those that can prelude risks to structures like dams, bridges, and rollercoasters. This chapter is the complete project, which is easy to replicate for personal applications.

I should say that I only participated marginally in this project, giving some help and advice to Jan, who managed the building, but more importantly, had the "maker idea."

2.1. A Vibration Simulator Made Easy

The first step in testing and developing the vibration simulator is designing a cheap but efficient simulation environment. The experimental conditions should be reproducible multiple times and always give the same results.

In addition, it should include the range of vibrations in the frequency range of a real-world application. For this reason, the choice of a second-hand audio subwoofer to generate test vibrations resulted in the ideal device. See Figure 2-2.

Figure 2-2. *Details of the audio subwoofer modified for the vibration platform. The top platform is screwed to the four corners and is connected to the subwoofer speaker through a plastic cylinder, which is glued on both sides*

Low vibrations mainly refer to the frequency range of audible sounds (roughly between 20 and 20.000 Hz). In the case of structures, the terms “low” and “high” assume a different meaning. Two parameters were identified:

- **Natural frequency**: The frequency at which a particular structure or material is expected (and designed) to vibrate due to its intrinsic characteristics.
- **Amplitude threshold**: The vibration motion in mm that is expected in normal conditions. Higher thresholds may indicate structural damage, causing serious concerns.

In the three examples, average acceptable values are as follows:

- **Bridges**: The natural frequency is around 1-5Hz. Slight higher frequencies can generate harmonics inside the structure, and the usual 1mm vibration amplitude can transform in a dangerous situation.
- **Dams**: The natural frequency is under 1Hz. Infrequent vibrations of 1-2mm amplitude, also infrequent, represent an alarming situation.
- **Rollercoasters**: The natural frequency ranges from 5-10Hz because their structure is mainly based on metal. Vibrations over a few mm of amplitude are dangerous.

To complete the simulator, we added a rectangular elastic platform made of cardboard and compact Styrofoam fixated to the four corners of the subwoofer box. The center of the vibrating surface was connected to the speaker’s center through an acrylic cylinder glued on both sides. See Figure 2-3.

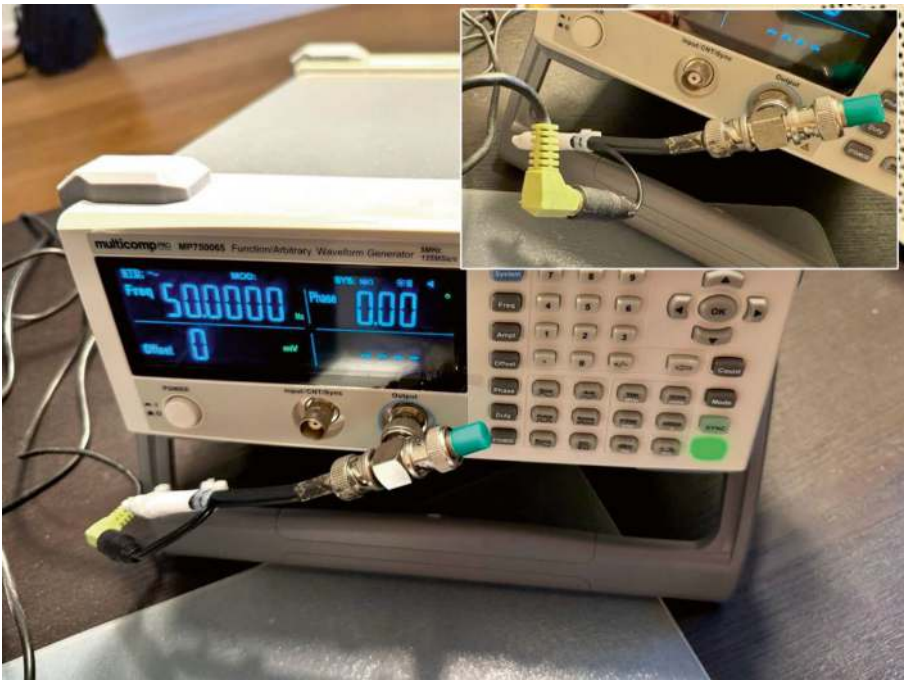

Figure 2-3. *The frequency generator used to generate a stable frequency. To send the signal to the subwoofer audio input, a small adapter from the coaxial cable output to the audio jack plugged into the amplified subwoofer input was sufficient*

We used a frequency generator to generate a precise and stable frequency. The subwoofer input signal, connected to the frequency generator's output signal, can be amplified properly, thanks to the small amplifier included in the original device.

After setting the desired frequency, the simulation surface vibrates accordingly, while the subwoofer amplifier allows the vibration intensity to be changed as needed.

2.1.1. The PCB 603C01 Sensor for Industrial Applications

After testing the simulation model, the project's second phase involves verifying its efficiency by collecting data with a sensor. See Figure 2-4.

Figure 2-4. *The piezo electric vibration sensor for industrial applications used for testing the setup. The sensor (center of image) is connected to a coaxial cable for signal detection, while the opposite side is firmly bolted to the vibration platform*

Thanks to the Element14 community and the producer's sponsorship, Jan could produce the detection tests with the industrial vibration sensor Piezo Electric Accelerometer PCB 603C01 by the IMI Sensors division of the PCB Piezotronics company.

This high-precision and sensitive sensor can be connected differently depending on the application.

Note The range of vibrations that raise alarms in a real-world application varies by structure. A detailed engineering analysis via monitoring systems determines this range. Alarms are set based on natural frequencies, expected load conditions, and the potential for resonance inside the structure.

Thanks to the Element14 community and the producer's sponsorship, Jan could produce the detection tests with the industrial vibration sensor Piezo Electric Accelerometer PCB 603C01 by the IMI Sensors division of the PCB Piezotronics company.

This high-precision and sensitive sensor can be connected differently depending on the application. See Figure 2-5.

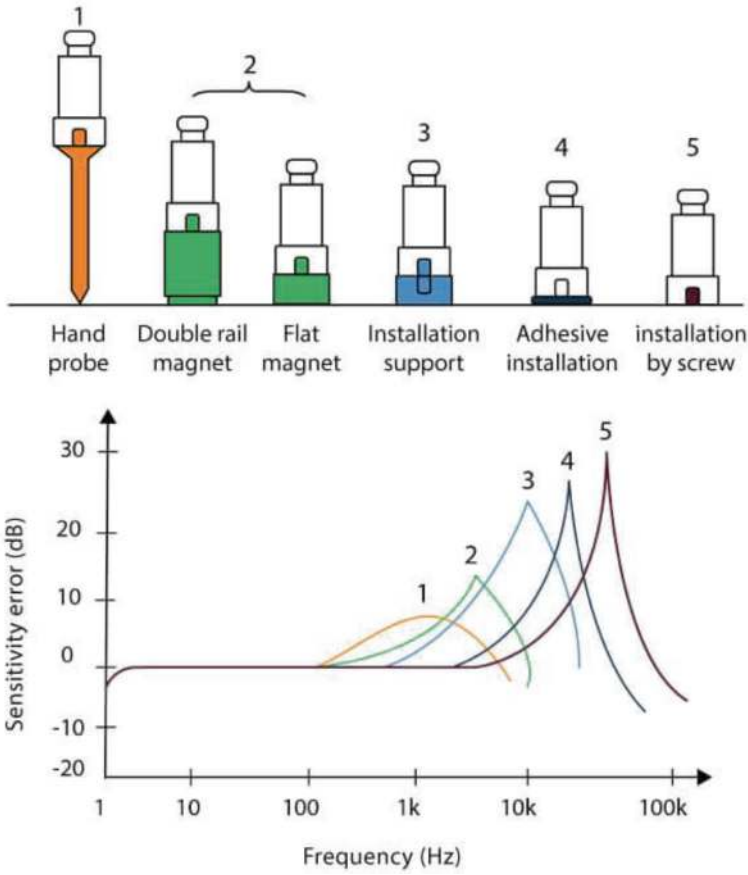

Figure 2-5. *The sensor has several fixation ways, according to the different environments and types of applications*

In this case, it was sufficient to firmly bolt the PCB 603C01 to the upper surface in order to connect it to the test vibration platform.

Due to the low currents, the sensor must be connected to a proper coaxial connector to measure the values.

2.2. Raspberry Pi to Process Realtime Data

With the described setup, the simulation platform generates a measurable output signal. At this point, we must choose a reliable and quality DAQ (Data Acquisition) system to collect the simulator information for testing purposes and real-world data.

Thanks to the sponsors and Element14 community, we set up the hardware DAQ using a Raspberry Pi 4 with an MCC 172—IEPE Measurement DAQ *HAT* (Hardware Attached on Top) mounted on top (https://digilent.com/shop/mcc-172-iepe-measurement-daq-hat-for-raspberry-pi/). See Figure 2-6.

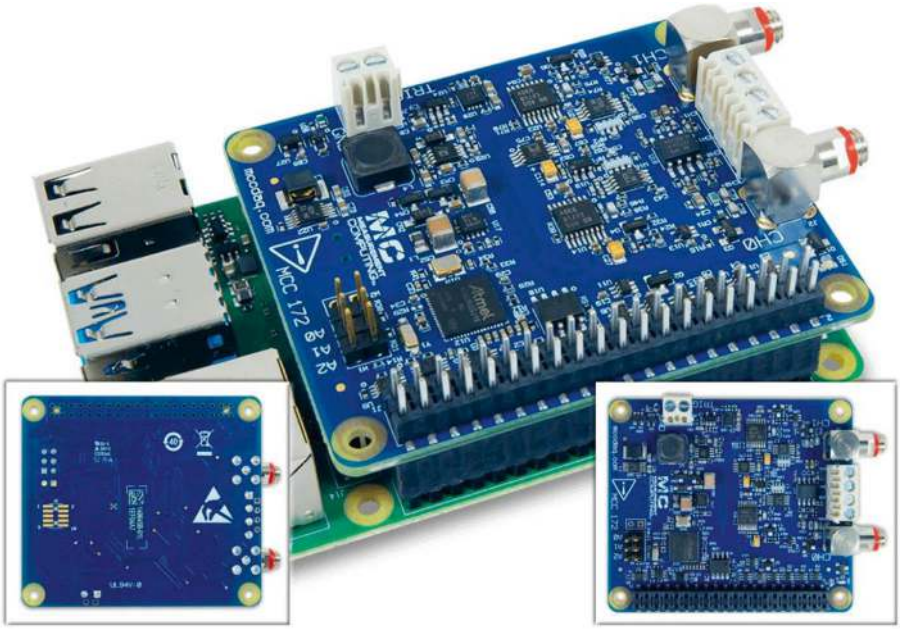

Figure 2-6. *The MCC 172—IEPE Measurement DAQ HAT for data acquisition from the piezo electric sensor*

The HAT is a two-channel ADC (Analog to Digital Converter) with local control and auxiliary power supplies. It is specialized in sampling piezo sensors and microphones to gather audio and vibration metrics.

It can acquire a fast data burst to send it to a Raspberry Pi host for further processing and analysis.

2.2.1. Analog Specifications

- 50.000 samples per second
- IEPE power supply for each channel
- The power supply is switchable, has 23 V compliance, and has a typical 4mA constant current
- It can deliver 4mA constantly as long as the required voltage does not exceed 23V

2.2.2. Digital Specifications

- Digital trigger input
- Local microcontroller and FIFO data buffer that can hold 49,000 samples
- Raspberry Pi as host via SPI
- Stackable: A single Raspberry Pi can handle up to three HATs at full speed on both channels

See Figure 2-7.

Note Due to the popularity and wide range of possibilities offered by the Raspberry Pi Linux Embedded platform and its powerful performance, this device has gained a privileged position in industrial applications beyond the makers' world.

Many companies and individual makers also started developing *Raspberry Pi HATs*. Anyone can develop a custom HAT following the simple specifications provided as open source by the Raspberry Pi Foundation, which are available on GitHub (https://github.com/raspberrypi/hats).

The following link is just an example of the plethora of Pi HATs available on the market for the different models of the embedded Linux board: https://thepihut.com/collections/raspberry-pi-hats

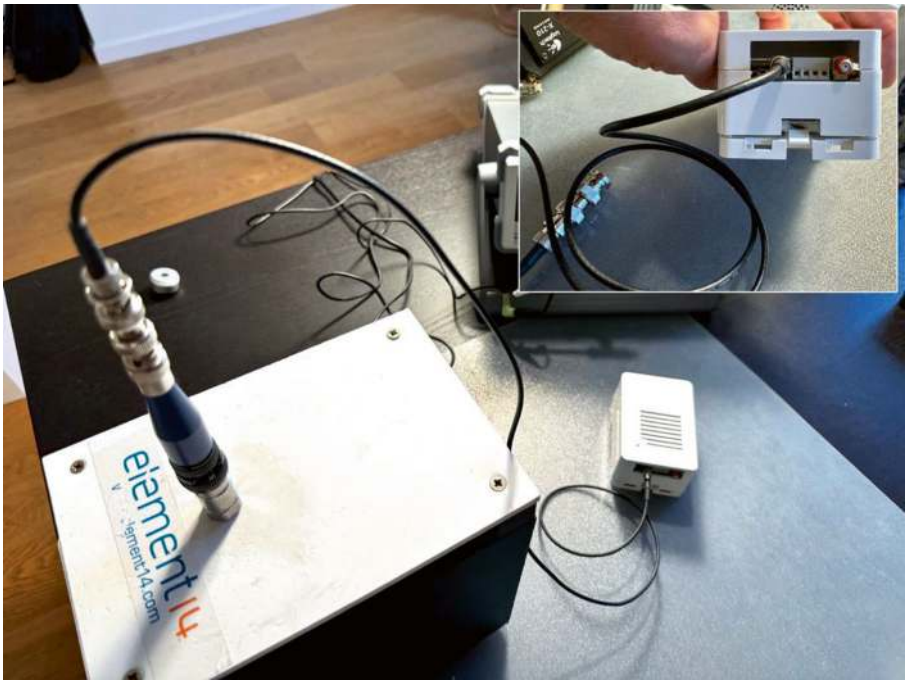

Figure 2-7. *The complete testing setup with the Raspberry Pi and the HAT connected to the vibration sensor through a coaxial cable*

2.3. The Data Processing Software

Data collection from the Raspberry Pi using the MCC 172—IEPE Measurement DAQ HAT can be achieved using many different approaches: Python, MATLAB, LabVIEW, C/C++ are only some of the possible cases.

The core of the software development is the HAT library (available on GitHub at `https://github.com/mccdaq/daqhats`). The first functionality test was done using the FFT (Fast Fourier Transform) example in the HAT library.

Then, keeping the examples as a reference, Jan developed a first test Linux program to run it on the Raspberry Pi Raspbian OS. The development was done with Eclipse, but this IDE is not mandatory.

Note This chapter's software folder includes all the examples, including the HAT libraries.

2.3.1. Using LabVIEW for Data Processing

> *LabVIEW is a graphical programming environment that provides unique productivity accelerators for test system development, such as an intuitive approach to programming, connectivity to any instrument, and fully integrated user interfaces (see https://www.ni.com/en/shop/LabVIEW.html).*

LabVIEW is a commercial application that also provides a community edition for developers and for personal use. For this reason, we decided to use this platform to illustrate the software implementation for data processing.

Note Initially developed about 40 years ago on Mac platforms, LabVIEW has been integrated into Windows and Linux. Unfortunately, the application is no longer supported for Mac platforms (OSX 14 Sonoma), and the preexisting licenses for Mac have been converted from annual subscription to permanent, without support after version 2003 Q3.

This may be a minor limitation, but the most recent 2024 version is only supported on Windows and Linux 64-bit with Intel processors. In practice, LabVIEW does not work on all the hardware based on ARM processors, whether they are Windows or Linux platforms.

The best open source alternative to LabVIEW is the Java-based MyOpenLab, which I suggest you use if you don't own an Intel-based computer.

The principle of the LabVIEW platform is to visually design virtual components, connecting them to the data source and processing them to obtain the desired measures.

For this test, Jan used three components:

- A service wrapper so that the virtual instrument is available upon boot and runs in the background (daemon run).
- Example LabVIEW flows used as training material.
- A showcase with the vibration test piezo electric sensor.

The HAT library connects the wrapper to the Raspberry Pi hardware DAQ.

The SCPI parser was used in the Jan Breuer library (https://github.com/j123b567/scpi-parser). See Figure 2-8.

Note SCPI (Standard Commands for Programmable Instruments) is defined as an additional layer on top of the IEEE 488.2-1987 specification, "Standard Codes, Formats, Protocols, and Common Commands."[4] The standard specifies a common syntax, command structure, and data formats for all instruments. It introduces generic commands (such as CONFigure and MEASure) that could be used with any instrument (Source: Wikipedia https://en.wikipedia.org/wiki/Standard_Commands_for_Programmable_Instruments).

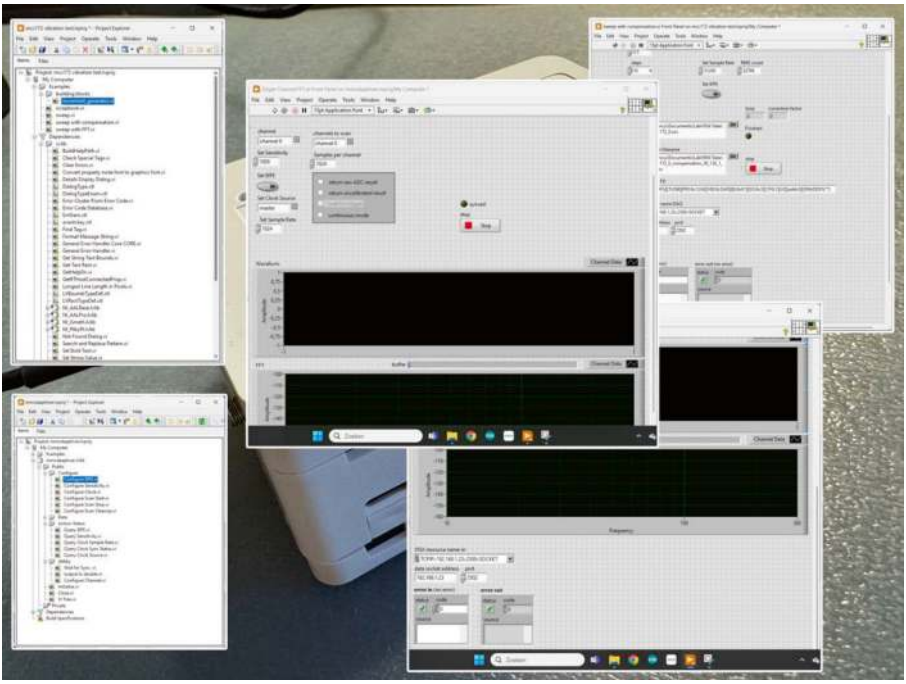

Figure 2-8. *Some examples of the user interface designed with the graphic editor of LabVIEW. The UI exposes interactive components to the user for setting parameters and virtual instruments showing real-time data coming from the probes*

The software project is divided into two components: the graphics user interface, where the data is shown, and the driver. The driver is the design part telling LabVIEW how to retrieve the data from the Raspberry Pi and organize it, including range controls, error management, and other features. See Figure 2-9.

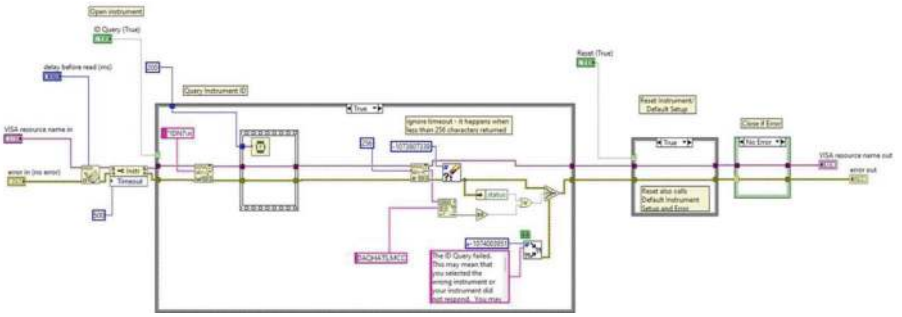

Figure 2-9. *The preview snippet of the custom LabVIEW driver for the custom instrument streaming data from the Raspberry Pi*

The visual design is then converted and eventually customized into the source code. Those parts of the design that need to be set or configured are connected to their corresponding UI design components. The user accesses the final instrument using the interface widgets, providing radio buttons, switches, parameter settings, and so on. See Figure 2-10.

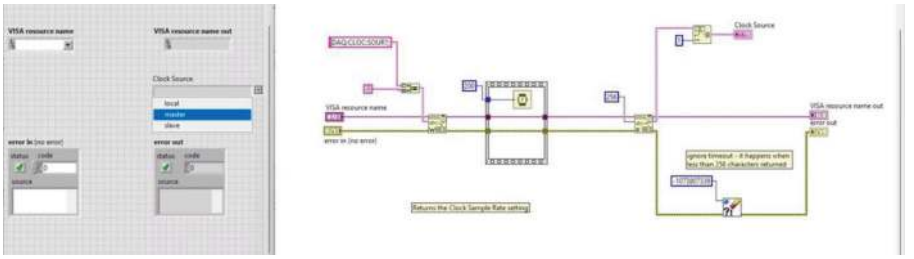

Figure 2-10. *Clock source configuration and sample rate. This part of the LabVIEW instrument includes user interaction*

Thanks to LabVIEW, the Raspberry Pi HAT (the DAQ) became programmable through a network connection to the LabVIEW virtual instrument driver.

Proceeding in this way, Jan designed a LabVIEW control panel, developed on the laptop, so that the configurable areas become available, including proper feedback.

The compiled binaries of the LabVIEW design can run directly on the Raspberry Pi. See Figure 2-11.

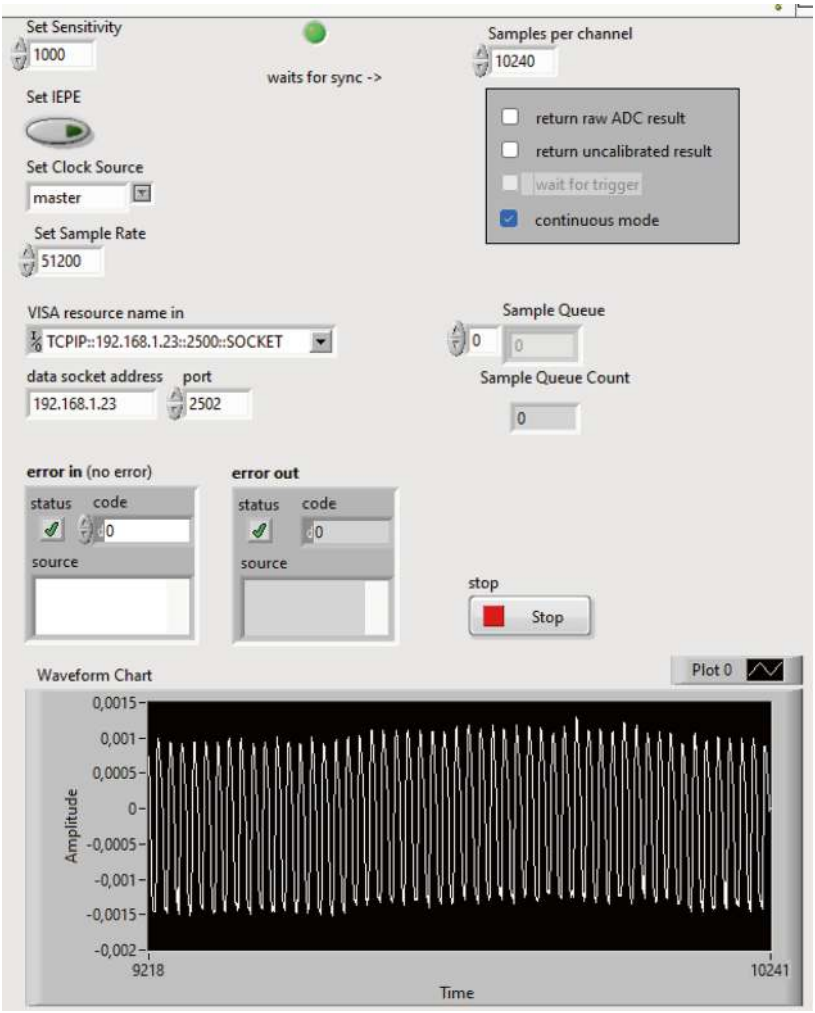

Figure 2-11. *The LabVIEW interface connected to the Raspberry Pi HAT DAQ retrieving real data. The workflow asks for a set of samples, collects the results, and shows them on a LabVIEW waveform chart*

A further customization is modifying the project (available in the repository) to save the streamed data on disk to retrieve them when not connected for analysis.

2.4. Making a Cheap Piezo Sensor

After assessing an efficient workflow widely tested with the subwoofer simulation platform and the PCB 603C01 for industrial applications, Jan successfully implemented the same solution with a cheap piezo sensor. See Figure 2-12.

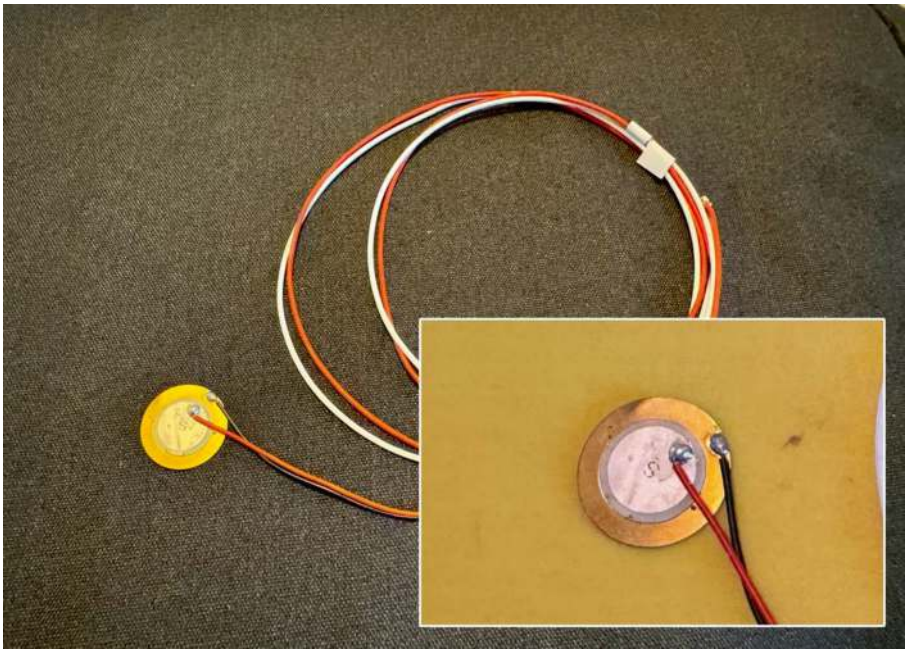

Figure 2-12. *The jellybean piezo sensor is an alternative to making a cheap vibration sensor. These components are available in the market at a very low price (ten pieces cost an average of $10)*

This small, cheap device has all the credentials needed to play the role, but depending on the vibration amplitude, it can generate very high voltages, which can damage the HAT's ADC (Analog to Digital Converter) input. For this reason, a filter-and-buffer circuit is needed.

This device is similar to the one that generates the spark to light the gas of a lighter or cooking plate. The filter circuit was designed by another Element14.com top community member, Michael Kellet (https://community.element14.com/members/michaelkellett). See Figure 2-13.

Figure 2-13. *With a simple commercial filter, the response of the jellybean piezo, when ticked with a pen, is still too high to avoid the risk of damaging the ADC input*

Michael Kellet designed a small circuit that protects the analog input with a couple of diodes and uses a charge amplifier to buffer the signals. See Figure 2-14.

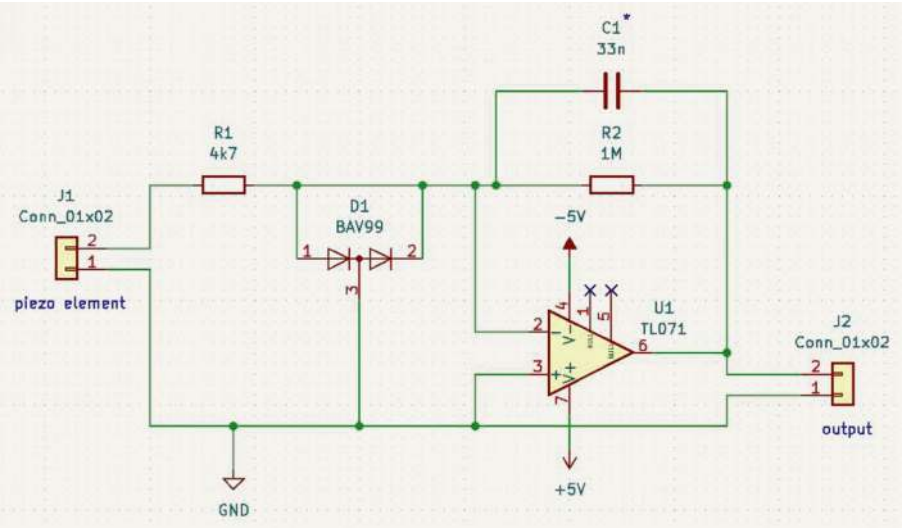

Figure 2-14. *Michael Kellet's circuit to interface a jellybean piezo sensor to the HAT analog input*

After designing the circuit with KiCAD, Jan made the PCB and built the circuit. See Figure 2-15.

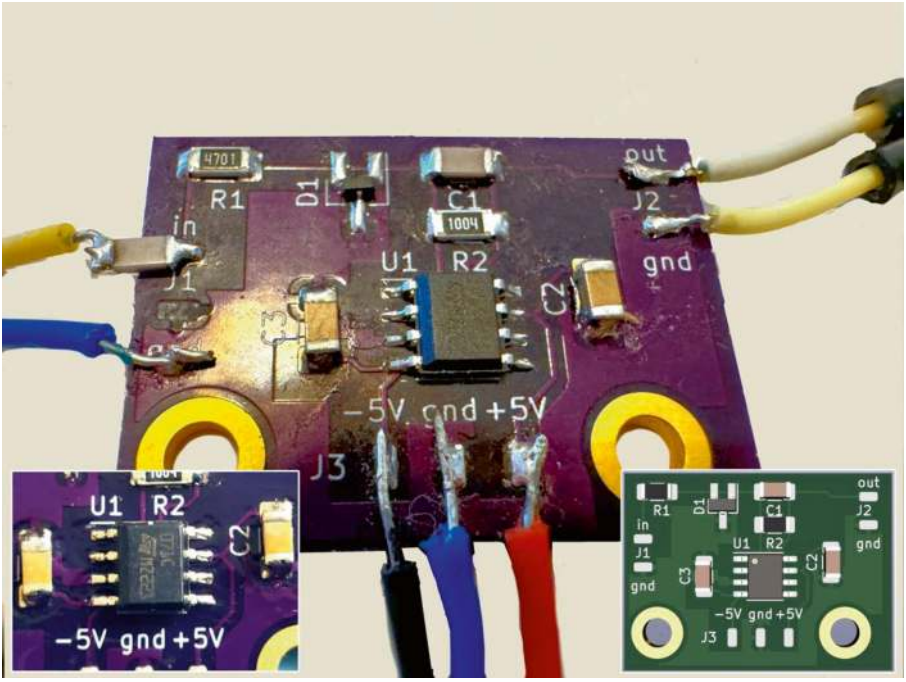

Figure 2-15. *The KiCAD PCB design and the final circuit are assembled and ready to connect the jellybean piezo sensor to the ADC input of the Raspberry Pi HAT DAQ*

With the filter and buffer circuit described here, the sensor operates inside the desired range—the signal is within the voltage rails. Note that the HAT DAQ operates in the 5V range. With a different ADC, for example, one operating in the 3.3V range, we need a further circuit to step down the current. See Figure 2-16.

Figure 2-16. *The oscilloscope response of the piezo electric sensor ticked with a pen, applying the buffered filter*

CHAPTER 3

The Scary Mirror

Figure 3-1. *The scenographic effect of the "scary mirror" during the last phase of software development*

The mirror was a particular fascination in ancient mythologies and modern cultures. Think of the Greek myth of Narcissus, who falls in love with his image reflected in a pool of water, and Perseus, who uses Athena's mirror shield to avoid seeing Medusa and being transformed into stone.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_3

In Roman mythology, Venus is represented with a mirror to symbolize beauty and self-admiration. Fenghuang of Chinese mythology often carries a mirror to show his pure heart, while Japanese mythology says that the sun goddess Amaterasu is attracted out of a cave seeing her image in a mirror.

Contemporary artists, from M.C. Escher to J.L. Borges, frequently used mirrors as symbols (M.C. Escher—The Magic Mirror, lithography, 1946).

Magic mirrors are also present in popular tradition and fairy tales; we all know the magic mirrors in Snow White (Grimm brothers), Beauty and the Beast (Jeanne-Marie Leprince de Beaumont), The Snow Queen (Hans Christian Andersen), Alice Through the Looking Glass (Lewis Carroll), and others.

This project explores the creation of a real magic mirror with something special (see Figure 3-1).

3.1. The Magic Mirror Platform

Released under the MIT License, `MagicMirror2` is a modular platform at the core of many projects. This platform, developed mostly on JavaScript, is portable to several platforms; it can also run on Windows, but it is preferably used with the Linux operating system. See Figure 3-2.

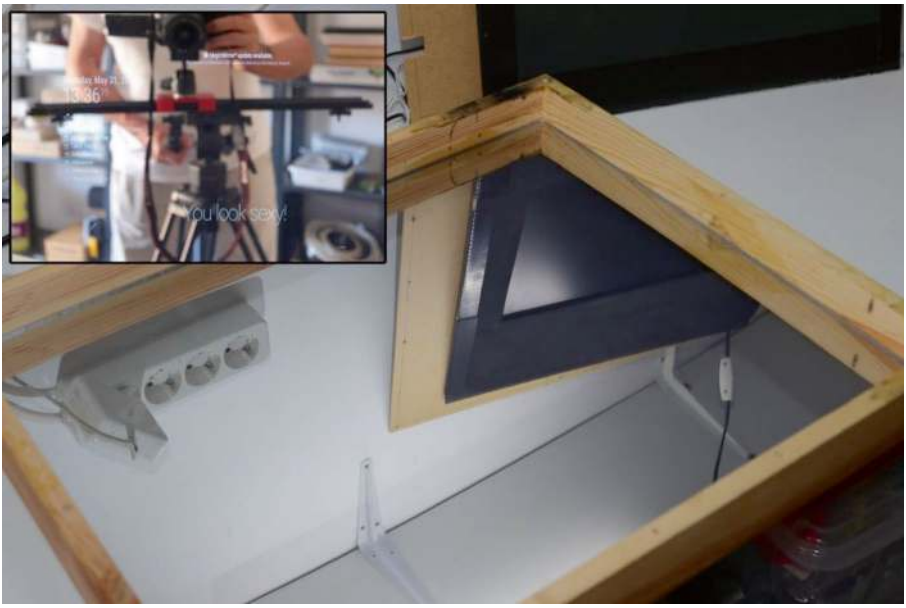

Figure 3-2. *The first magic mirror structure I developed*

The modularity of `MagicMirror2` and the possibility of developing custom plugins attracted a vast community of contributors; nowadays, hundreds of plugins support many features and are easy to install and customize.

The introduction of the makers' market of the powerful Linux-based Raspberry Pi, supporting the `MagicMirror2` features, boosted the system's popularity. It can work on any HDMI screen as well as on the Raspberry Pi touch screen. From a technical point of view, the application will run on a full screen, showing some unique content according to the settings and the installed plugins. See Figure 3-3.

Figure 3-3. *The assembled backside of the magic mirror, including the Raspberry Pi and the HDMI screen controller. For this first creation, I followed a very basic setup with the Raspberry Pi running only the MagicMirror2 platform*

Making the screen content behind a semi-reflection mirror visible depends on the characteristics of the displayed data: the size of the fonts, the source of information, the color of the font, and so on.

When the screen showing the `MagicMirror2` application on a black background lies behind a semi-transparent mirror, only the displayed data is visible; the screen background and the free surface of the mirror are not visible. See Figure 3-4.

Figure 3-4. *The mask components of the back of the mirror create the scenery effect*

I have experimented with making my own magic mirror on a Raspberry Pi in the past, appreciating the incredible effect generated by this simple building.

When I made the first magic mirror, the software platform was still in its early stages; the semi-reflective mirror was the most expensive and difficult-to-find component. Due to the popularity acquired by this kind of making, the prices of semi-reflective mirrors went down considerably in less than a year.

3.1.1. The MagicMirror2 Platform Architecture

The platform architecture is server-client based; the client provides the display information—and eventually the interaction—while the server manages the `MagicMirror2` process. For this reason, using a Linux machine is preferable; from this perspective, a Raspberry Pi undoubtedly represents the ideal hardware.

The installation process in the Raspberry Pi, or any other Linux system like Ubuntu, is easy:

1. Download and install the latest NodeJS version.
2. Check if git is installed on your machine by executing `git` or otherwise install it.
3. Clone the repository: `https://github.com/MagicMirrorOrg/MagicMirr` *or*
4. Enter the repository: `cd MagicMirror/`
5. Install the application: `npm run install-mm`
6. Copy the config sample file: `cp config/config.js.sample config/config.js`
7. Start the application: `npm run start`

(Source: MagicMirror2 installation documentation.)

Entirely developed and maintained as a NodeJS application, the platform—both the client window and the server—uses JavaScript to configure and customize with a set of JSON files.

All the information, including the list of all available plugins and the complete documentation, can be accessed from the GitHub repository at `https://github.com/MagicMirrorOrg/MagicMirror`.

3.2. Push the Mirror Beyond the Limits

The idea behind the Scary Mirror project is to improve the classical magic mirror features with advanced interactivity. Of course, this was possible using the powerful Raspberry Pi model B3.

I aimed to create an automated window that displays updated information like weather, calendar appointments, email notifications, and so on, and a system that reacts to the user's presence in front of it.

To achieve this goal, the design became more complex. In addition, to make it possible to place the Scary Mirror virtually everywhere, including in an uncommon place, the whole project needs to be powered by batteries.

3.2.1. PIR Sensor

The easy-to-use PIR (Passive InfraRed) sensor detects motion by sensing infrared radiation emitted by objects. PIR sensors are very common devices used in many applications.

They can be calibrated at a specific distance (between 30cm and about 5m), so when a person's movement is detected inside the range, they trigger a signal that a GPIO logic input can read.

The `MagicMirror2` server can run a predefined window by default until the PIR sensor detects movement. This triggers a different sequence shown on the screen behind the semi-reflecting mirror.

3.2.2. Pi Camera

The pinhole camera available for the Raspberry Pi includes many visual effects that can be added to the making. The Pi Camera driver, including the live effects, can be fully controlled from the terminal through a Bash script, a Python script, or any other way a process can be launched from the Linux operating system, including an independent process configured in the crontab schedule.

3.2.3. Audio Effects

Associating images and short videos, especially unexpected sequences that surprise the observer, with proper audio effects improves the user experience.

Starting from the Raspberry Pi 3B, the audio output has been changed from PWM to stereo DAC with a 3.5mm output jack. This signal (it can be directly connected to an earbud) is sent to a portable battery-operated amplifier.

3.2.4. Lighting Effects

Since these are designed to operate in a dark corner, adding surround lighting effects is essential. To achieve this goal, the initial idea was to use an Arduino dedicated to controlling a 128-Neopixel light strip controlled by the Raspberry Pi via the I2C protocol.

However, the I2C protocol became unstable while the Linux system was running the NodeJS `MagicMirror2` server, so I opted to keep the lighting effect independent of the rest of the system. See Figure 3-5.

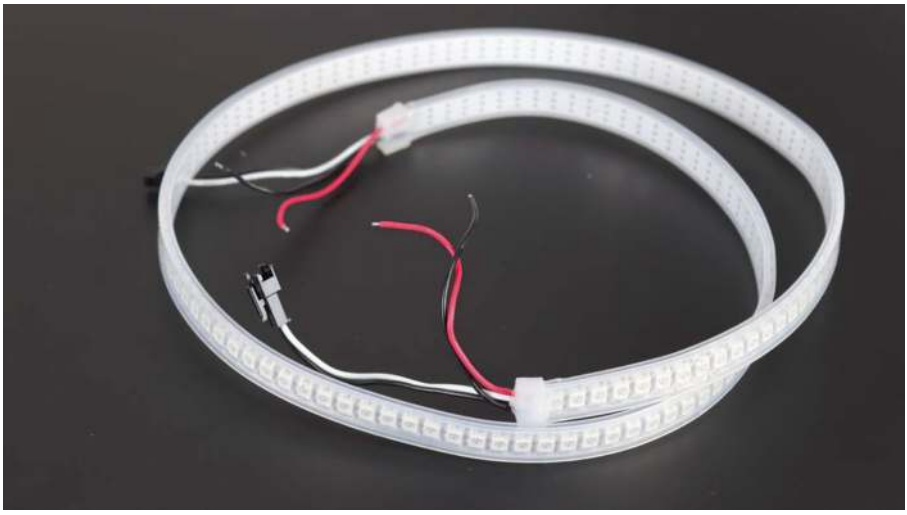

Figure 3-5. *The 128-Neopixel LED strip used to create the lighting effects*

3.3. Making the Mirror

Instead of making a magic mirror from scratch, I searched for a proper structure to upcycle.

The best choice was a round IKEA bathroom mirror with an internal light. The original mirror was quite ugly (and it was on sale for less than ten dollars), but when I saw it, I imagined the final result and it was love at first sight.

It was round, with a large plastic frame perfect for fitting some of the components, and the diameter was perfect. The internal structure supporting the electronic boards, batteries, and wiring was obtained by recycling the mirror's cardboard box.

3.3.1. Preparing the Frame

Figure 3-6. *The disassembled mirror frame*

The bottom hole—used for the light power button—was carved to fit the PIR sensor. The two-round-holed grids correspond with the two internal amplifier speakers. See Figure 3-6.

On the top center of the frame, a small hole hosts the Pi Camera lens. After sanding, the frame was painted black. See Figure 3-7.

Figure 3-7. *The back side of the Scary Mirror frame*

The best magic mirror effect is obtained when the display screen fits close to the back of the semi-reflecting mirror.

In the final making, the frame is the first layer the observer sees, while the mirror, screen, and components are placed together on a support, like making a box. This is why the 5cm thick frame is ideal to include all the components.

Observing the back side of the frame, the PIR sensor is already glued to the bottom hole. The two speakers are glued in correspondence to the grid holes of the frame, and the small 2W amplifier with its battery holder is also glued to the frame.

The Neopixel LED strip is in place around the frame border and inserted into a cardboard rail. These components are connected to the internal audio and lighting effects block. These parts should be considered the peripherals of the Scary Mirror device.

3.3.2. The Internal Cardboard Block

As mentioned, the entire assembly of the Scary Mirror is contained in a 5cm thick cardboard block obtained by recycling the IKEA packaging.

The semi-reflecting mirror should stay as close to the screen on its back as possible. For this project, I used the six-inch Raspberry Pi touch display. The final assembled block is pressed against the plastic frame, keeping all the parts together. See Figure 3-8.

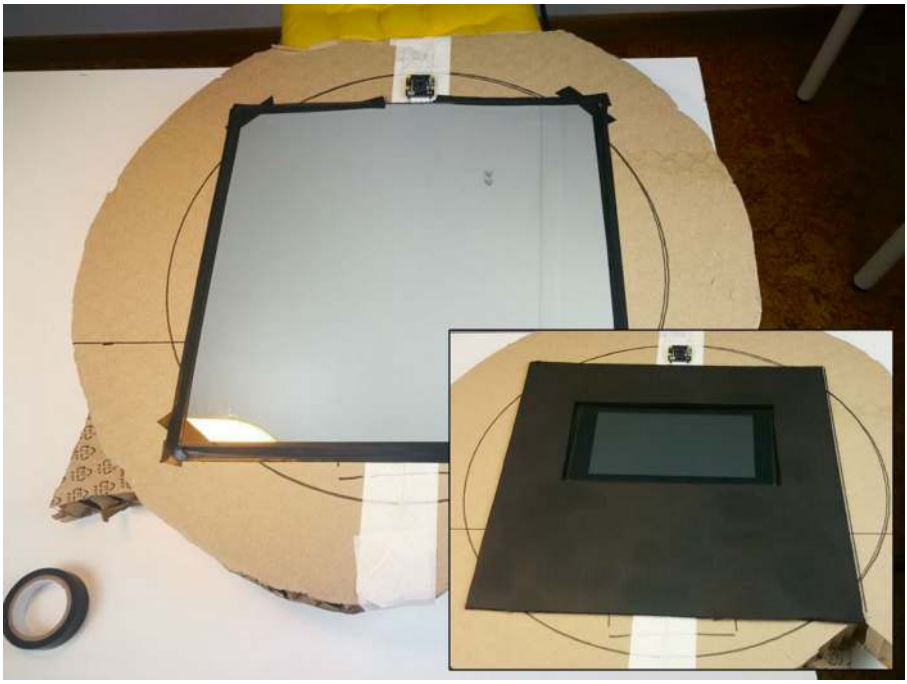

Figure 3-8. *The front side of the cardboard block is exposed to the observer's view. The detail shows the Raspberry Pi display with the black cardboard mask to create the perfect reflecting background*

On the back side of the cardboard block, I engraved the space to fit the Raspberry Pi, the Arduino, and the other components, including the battery pack.

To fix the cardboard and all the components close to the frame, I used three big screws passing through the plastic frame, which kept the internal build in place. See Figure 3-9.

Figure 3-9. *Detail of one of the three screws that block the internal build of the Scary Mirror and keep it in place when the mirror is hung vertically*

3.4. Electronics, Wiring, and Powering

To make the Scary Mirror battery-operated, I used various power sources.

The Raspberry Pi has a LiPo battery and a HAT Pi Juice, which I have used in other projects. This component is not essential, as the Raspberry Pi can be powered with multiple 5Vcc power sources, including a 4000mA power bank.

A 2000mA smartphone power bank powers the Arduino and the Neopixel LED strip, while I recycled the audio amplifier's original two AA battery holders. See Figure 3-10.

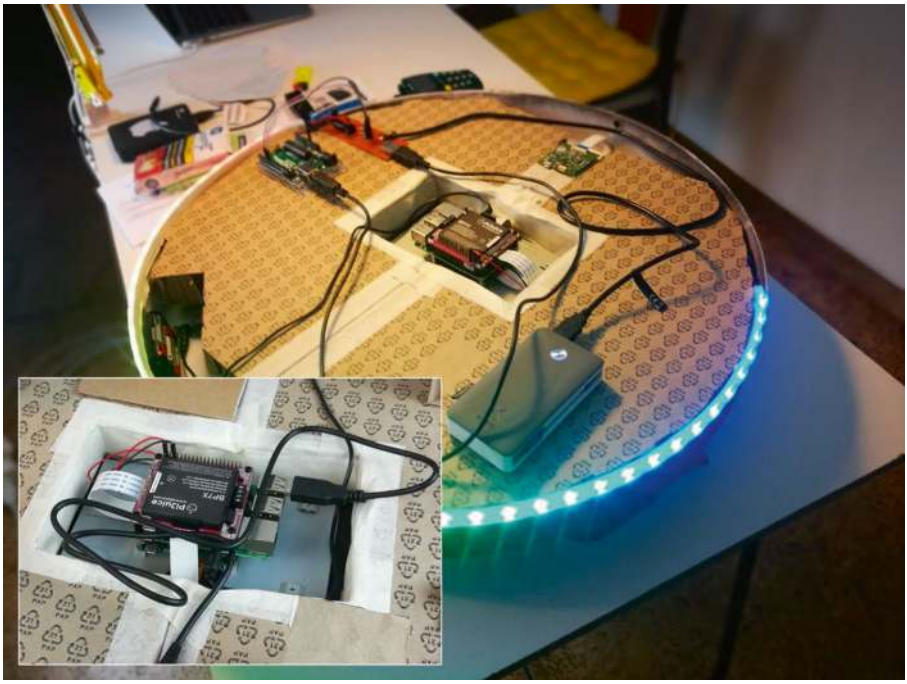

Figure 3-10. *Powering test of the components to verify the duration of the batteries*

3.4.1. Power Supply Issue

The power of the smartphone power bank is sufficient for the Arduino and the Neopixel LED strip.

As mentioned, the Arduino manages the Neopixel lighting sequence; for this reason, I set the software while powering the system with a USB power adapter, temporarily excluding the battery supply. When everything worked fine, I connected the power bank, and the board worked fine for about ten seconds. Then the power automatically shut down.

I spent hours searching the Internet for this unexpected behavior until I discovered that modern smartphone USB power banks include a circuit that turns off the external power battery when the smartphone is fully charged.

To avoid overcharging the smartphone battery, the external source only works when the charging device periodically generates a pulse requiring more charging power. After about ten seconds, the power bank automatically powers off without this pulse.

The solution has been implemented with a timer interrupt in the Arduino UNO board software. This low-priority interrupt timer triggers automatically every five seconds, sending a pulse to the power bank from one of the available pins of the Arduino UNO GPIO.

The logic positive signal generated by the Arduino UNO is sent to a simple circuit built by a transistor. It triggers the power bank USB power line, simulating the power request expected from a smartphone that is not fully charged.

The result worked fine, as the pulse acts as a "keep alive" signal in the network connections. See Figure 3-11.

Figure 3-11. *The periodic pulse generated by the Arduino UNO GPIO pin is sent to the power bank every five seconds*

3.4.2. Driving Large Arrays of Neopixel LEDs

A high-frequency signal drives the Neopixel LEDs through the Neopixel Arduino UNO library.

Working with large arrays of Neopixel LEDs, as in this case, raises concerns about the stability of the signal along the array. This can make it difficult to drive the LED's color and reach the desired LED in the array.

According to the Neopixel documentation, this problem can be easily solved with a simple add-on circuit to the LED array connector, as close as possible to the GPIO signal. See Figure 3-12.

Figure 3-12. *The completed assembly of the Scary Mirror. To cover the exposed parts around the mirror and improve the scenographic effect, I hot-glued some black tulle pieces around the device*

The power output signal provided by the Arduino UNO board is limited to 20mA, which is not sufficient to power more than 10-20 LEDs directly from the board. The array signal and ground should be connected to the Arduino UNO GPIO, but an external source should provide the Neopixel array power.

To improve the stability of the array signal, a polarized 1000 uF capacitor must be placed between the array's GND and Vcc power lines. See Figure 3-13.

Figure 3-13. *The back side of the Scary Mirror. A large portion of the frame circumference holds the 128-Neopixel LED strip. The parts that should be hidden on the front side are covered by a tulle textile*

3.5. The Software

The project software is split into the C Arduino sketch to control the lighting effects and the `MagicMirror2` package, which is properly configured to run on the Raspberry Pi and start at boot.

3.5.1. Arduino UNO Sketch

The first part of the source defines the constants and hardcoded values for the pulse generator and the Neopixel library.

The Neopixel library instance requires three parameters:

1. Number of pixels (Neopixel LEDs) in the strip
2. The Arduino pin number used to control the LEDs
3. The pixel-type flag

The Neopixel library supports several different Neopixel types, depending on the provider. These differ in the control signal frequency, according to the library header constants (see Listing 3-1):

- `NEO_KHZ800`: 800 KHz bitstream (most Neopixel products with WS2812 LEDs)
- `NEO_KHZ400`: 400 KHz (classic v1 FLORA pixels, WS2811 IC drivers)
- `NEO_GRB`: Pixels wired for GRB bitstream (as for most Neopixel products)
- `NEO_RGB`: Pixels wired for RGB bitstream (v1 FLORA pixels)
- `NEO_RGBW`: Pixels are wired for RGBW bitstream (Neopixel RGBW products)

Listing 3-1. The Initial Definitions of Constants and Hardcoded Settings

```
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include <TimerOne.h>

#ifdef __AVR__
  #include <avr/power.h>
#endif

#define NEOPIXEL_PIN 6

//! Pulse pin to avoid the power pack automatically shut down.
#define PULSE_PIN 5
//! Uses the builtin LED to monitor the pulse generation to
keep the power bank
//! battery always on
#define PULSE_MONITOR LED_BUILTIN
//! Pulse duration. Should be empirically calibrated on the
power bank used
#define PULSE_MS 5
//! Interval between two pulses (long interval, about 1 second)
#define PULSE_INTERVAL 5000000
//! Number of Neopixel LEDs in the strip
#define NEOPIXEL_LEDS 144
//! I2C Slave address
#define I2C_ADDR 0x08
```

To simplify the program's business logic and limit the lighting effects to predetermined colors, I empirically selected and hardcoded a set of RGB colors with easy-to-understand symbolic names.

In the `setup()` function, the Neopixel library is initialized, and then the last instruction arm, the timer, interrupts and starts it.

The interrupt must be launched in the `setup()` as the last instruction to avoid interferences during the library initialization. See Listing 3-2.

Listing 3-2. The setup() Function

```
void setup() {

#ifdef _DEBUG
  Serial.begin(9600);           // start serial for output
#endif

  strip.begin();
  strip.show(); // Initialize all pixels to 'off'

  Wire.begin(I2C_ADDR);
  Wire.onReceive(receiveEvent);

  pinMode(PULSE_PIN, OUTPUT);
  pinMode(PULSE_MONITOR, OUTPUT);

  digitalWrite(PULSE_PIN, HIGH);
  digitalWrite(PULSE_MONITOR, LOW);

  Timer1.initialize(PULSE_INTERVAL);
  Timer1.attachInterrupt(pulsePower);
  Timer1.start();
}

// Predefined colors
#define PINK strip.Color(255, 64, 64)
#define WHITE strip.Color(255, 255, 255)

#define FIRE1 strip.Color(255, 64, 0)
#define FIRE2 strip.Color(255, 96, 0)
#define FIRE3 strip.Color(255, 128, 0)
#define FIRE4 strip.Color(255, 96, 32)
```

```
#define FIRE5 strip.Color(255, 32, 64)
#define FIRE6 strip.Color(255, 32, 96)

#define BLUE1 strip.Color(0, 0, 255)
#define BLUE2 strip.Color(0, 32, 255)
#define BLUE3 strip.Color(0, 64, 255)
#define BLUE4 strip.Color(0, 96, 255)

#define PURPLE1 strip.Color(96, 16, 255)
#define PURPLE2 strip.Color(96, 16, 128)
#define PURPLE3 strip.Color(96, 16, 96)
#define PURPLE4 strip.Color(96, 16, 64)
```

The `loop()` function executes a sequence of lighting effects on the Neopixel strip, while the timer interrupt generates the power supply pulse independently. See Listing 3-3.

Listing 3-3. The loop() Function

```
void loop() {
    colorWipe(FIRE1, 10); colorWipe(FIRE2, 10);
    colorWipe(FIRE3, 10);
    colorWipe(FIRE4, 10); colorWipe(FIRE5, 10);
    colorWipe(FIRE6, 10);

    delay(500);

    colorWipe(BLUE1, 10); colorWipe(BLUE2, 10);
    colorWipe(BLUE3, 10); colorWipe(BLUE4, 10);

    delay(500);

    colorFlash(BLUE1, 10);
    delay(500);
    colorFlash(BLUE2, 10);
    delay(500);
```

```
    colorFlash(BLUE3, 10);
    delay(500);
    colorFlash(BLUE4, 10);

    delay(1000);
    /* */
}
```

The third part of the source is a collection of mathematical functions that create a colored light sequence on the LED strip using the hardcoded RGB colors. See Listing 3-4.

Listing 3-4. Functions That Create a Colored Light Sequence

```
//! Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait) {
  for(uint16_t i=0; i<strip.numPixels(); i++) {
    strip.setPixelColor(i, c);
    strip.show();
    delay(wait);
  }
}

//! Fill the dots one after the other with a color
//! Then off in the same sequence
void colorWipeOnOff(uint32_t c, uint8_t wait) {
  // Pixels on
  for(uint16_t i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, c);
    strip.show();
    delay(wait);
  }
```

```
  // Pixels off
  for(uint16_t i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 0x000);
    strip.show();
    delay(wait);
  }
}

void colorFlash(uint32_t c, uint8_t wait) {
    setColor(c);
    delay(wait);
    setColor(0);
}

void setColor(uint32_t c) {
    uint16_t i;

    for(i = 0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, c);
    }
    strip.show();
}

void rainbow(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256; j++) {
    for(i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel((i+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}
```

```
// Slightly different, this makes the rainbow equally
distributed throughout
void rainbowCycle(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256*5; j++) { // 5 cycles of all colors on wheel
    for(i=0; i< strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel(((i * 256 / strip.
      numPixels()) + j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Input a value 0 to 255 to get a color value.
// The colors are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85) {
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if(WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}
```

The timer callback function triggers the pulse generation to keep the power bank power supply for the Arduino UNO board and the Neopixel LED strip alive. See Listing 3-5.

Listing 3-5. The Timer Callback Function

```
//! Timer callback function.
//! Stop the timer, output the pin for 20 ms then restart
the timer
void pulsePower(void)
{
  Timer1.stop();
  digitalWrite(PULSE_PIN, LOW);
  digitalWrite(PULSE_MONITOR, LOW);
  delay(PULSE_MS);
  digitalWrite(PULSE_PIN, HIGH);
  digitalWrite(PULSE_MONITOR, HIGH);
  Timer1.start();
}.
```

3.5.2. Raspberry Pi Software

Implementing the `MagicMirror2` platform on the Raspberry Pi Linux Raspbian is really easy. As the platform is frequently updated due to the large contributor community, I suggest following the last version installation instructions on the `MagicMirror2` GitHub repository.

After installing the package and testing it with the default (basic) configuration, you have to prepare to collect material from the Internet before proceeding to the specific setup for the Scary Mirror project.

Of course, the described project is designed to be modular, also according to the flexibility of the `MagicMirror2` platform, not a how-to recipe. The basic structure can be modified to create another original version of a magic mirror build, reusing the same software with minimal changes.

Collecting Contextual Media

Remaining in line with the scenic context of the magic mirror version of this project, I sourced from the Internet a set of short movie clips showing flames on a black background and a series of creepy sounds to make the viewer experience better.

The software package for this chapter includes a selection of the media I used in the build.

Running Multiple Tasks on Startup

The Pi Camera, PIR sensor, and `MagicMirror2` platform start on boot. I preferred to keep an easier-to-manage configuration, avoiding converting the Bash and Python scripts to services. This approach produces the same visual effect but is simple to modify and start/stop during development.

The simple startup script of the `MagicMirror2` platform launches the NodeJS server and the client on the same machine, which accesses the configuration files to start the modules according to the timing. Note that after the NodeJS is started, the current folder is set again to the home folder. The other scripts will start to manage the PIR sensor and camera from there. See Listing 3-6.

Listing 3-6. The Basic Startup Script

```
#!/bin/bash
# Launch Magic Mirror, as we don't want
# to start it as a service but scheduled
# in cron at boot
# ----------------------------------

cd /home/pi/MagicMirror
npm start
cd /home/pi

# That's all!
```

The PIR sensor Python script detects the observer's presence when a user enters the detection range. When the detection occurs, a partially transparent, bluish camera preview is enabled for a hardcoded number of seconds. A surprising effect, while the magic mirror loops the flame videos and sounds, is granted! See Listing 3-7.

Listing 3-7. The PIR Sensor Python Script

```
#!/usr/bin/python3
import paho.mqtt.publish as publish
import RPi.GPIO as GPIO
import time
import subprocess
from picamera import PiCamera

GPIO.setmode(GPIO.BOARD)
PIN_TRIGGER = 7 # Distance sensor trigger pin
PIN_ECHO = 11 # Distance sensor echo pin
# Mosquitto server IP Address - Set this value accordingly
mqttServer = "192.168.1.30"
# Mosquitto server channel name
mqttChannel = "magic_mirror"
# Distance detection pause duration after a valid movement
# has been detected and a message has been sent to the mqtt
# server (7of9) in minutes
mqttPause = 1
camera = PiCamera()

# Shows the camera preview for a number of seconds
def camPreview(sec):

    # print("camPreview()")

    camera.rotation = 180 # Image is bottom-top
```

```
    # Show the camera preview
    camera.start_preview(alpha=0)
    # And brightness 0
    camera.brightness = 0

    # Progressively increase the brightness
    # to the maximum
    for i in range(0, 50):
        camera.brightness = i
        time.sleep(0.1)
    # Leave the preview visible for the desired
    # number of seconds
    time.sleep(sec)
    # camera.stop_preview()

    print("camPreview() END")

# Initialize the GPIO pins for the distance sensor
def sensorInit():
    # Disable warnings if the program is launched twice
    # and the channel is already in use. It is anyway
      reset by the
    # new process.
    GPIO.setwarnings(False)

    GPIO.setup(PIN_TRIGGER, GPIO.OUT)
    GPIO.setup(PIN_ECHO, GPIO.IN)
    GPIO.output(PIN_TRIGGER, GPIO.LOW)

    time.sleep(2)

# Executes a sensor reading cycle, including the pause
# for the sensor settle.
# If the distance is between 2 and 400 cm (4 m) the reading
```

```
# is valid, else zero is returned.
#
# if isPause is true, after a reading pauses for a fixed period
# else return immediately to the caller
def checkDistance(isPause):
    # Set trigger to high then after 0.1 ms
    # set it to low as the sensor needs a
    # pulse of this length to start
    GPIO.output(PIN_TRIGGER, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(PIN_TRIGGER, GPIO.LOW)

    # define start/end time variables before executing
    # a distance reading. This is unnecessary, but if
    # the variables are not initialized randomly an error
    # of variable use before its definition may occur.
    # This is probably due to cases when the start time
    # reading is not yet been finished processing and the
    # echo (end time) reading while is already started.
    pulse_start_time = 0
    pulse_end_time = 0

    # Save start time until a transition does not occur
    # then save the time after the transition from the
    # echo input
    while GPIO.input(PIN_ECHO) == 0:
        pulse_start_time = time.time()
    while GPIO.input(PIN_ECHO) == 1 :
        pulse_end_time = time.time()

    # Calculate the pulse duration then calculate the distance
    # in cm according to the ultrasound speed
    pulse_duration = pulse_end_time - pulse_start_time
    distance = round(pulse_duration * 17150, 2)
```

```
    # Ignore out of range calculations, distance should be
    # detected between 2 and 400 cm
    if(distance <= 2 or distance > 400):
        distance = 0

    # Wait for the sensor to settle before starting another
    # reading cycle
    GPIO.output(PIN_TRIGGER, GPIO.LOW)
    if(isPause):
        time.sleep(0.5)

    # Return che reading
    return distance

# Send the distance value to the mqtt server
#
# Param: distance The value to send to the mqtt server
def mqttDistance(distance):
    publish.single(mqttChannel, distance, hostname=mqttServer)

# ------------------------------------------------
# Main process
# ------------------------------------------------
if __name__ == '__main__':

    camPreview(60)

    # Initialize the distance sensor pins
    sensorInit()
    checkDistance(True)

    # Infinite loop for continuous distance detection
    while True:
        dist = checkDistance(False)
        time.sleep(0.05)
```

```
# If the flag is true at the end of the distance
# check cycle, a message is sent to 7of9
comingThru = True
# Minimum/Maximum distance ranges in cm to consider a
# movement (no matter the direction)
minDelta = 1
maxDelta = 100

# If the distance is in the range, execute other
# five readings. If the distance remains in range,
# this means that it is true that something - maybe
# human? - is moving with respect to the mirror.
for moving in range(0, 5):
    newDist = checkDistance(False)
    time.sleep(0.05)
    delta = abs(dist - newDist)
    if( (delta >= minDelta) and
        (delta <= maxDelta) and
        (newDist <= maxDelta) ):
        comingThru = True
    else:
        comingThru = False

    # Update last distance read. This will be the sent
    # value if a movement around has been detected
    dist = newDist

# If the coming is true, the mqtt message is sent
if(comingThru):
    # Send the message to the server
    mqttDistance(dist)
    # Show a camera preview
    camPreview(60)
```

```
            # After sending the message, the cycle pauses
              for about
            # one minute to avoid a continuous sequence of
              voice messages
            time.sleep(mqttPause)
            # Disable the monitor
```

Also, the Python PIR process is launched by the Linux cron at startup as a Bash script, and then the program runs indefinitely. In Listing 3-8, developed for experimental test only, the PIR script is set to run as a service.

Listing 3-8. The Python3 Script

```
# Definition of the Python3 script
# distance_sensor as a service to run
# automatically in background
# This service definition file should be copied
# in /lib/systemd/system
# If for any reason the service is aborted, it is
# restarted automatically

[Unit]
Description=Ultrasonic distance sensor
After=multi-user.target

[Service]
Type=simple
ExecStart=/home/pi/distance_sensor.py
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

3.5.3. MagicMirror2 Configuration and Modules

MagicMirror2 modules, listed and documented in the GitHub repository, are easy to install. Sometimes installed modules are no longer needed. It is possible to remove them, but I suggest keeping them inactive for future use. It's best to clean up the modules and unneeded files only after the whole configuration works as expected.

The defaultmodules.js, shown in Listing 3-9, is an example of the standard module listed after installation.

Listing 3-9. The Default Modules List

```
/* Magic Mirror
 * Default Modules List
 *
 * By Michael Teeuw http://michaelteeuw.nl
 * MIT Licensed.
 */

// Modules listed below can be loaded without the 'default/'
   prefix. Omitting the default folder name.

var defaultModules = [
    "alert",
    "calendar",
    "clock",
    "compliments",
    "currentweather",
    "helloworld",
    "newsfeed",
    "weatherforecast",
    "updatenotification",
    "weather"
];
```

```
/*************** DO NOT EDIT THE LINE BELOW ***************/
if (typeof module !== "undefined") {module.exports =
defaultModules;}
```

The NodeJS server uses HTML, JavaScript, and CSS to customize the entire display behavior and user interface. Listing 3-10 shows the `MagicMirror2` main CSS, where it is possible to set up fonts, scripts, positions, and many other features. The best way to do this is to change the style and see the effects in real-time.

Listing 3-10. The MagicMirror2 main CSS File

```
html {
  cursor: none;
  overflow: hidden;
  background: #000;
}

::-webkit-scrollbar {
  display: none;
}

body {
  margin: 60px;
  position: absolute;
  height: calc(100% - 120px);
  width: calc(100% - 120px);
  background: #000;
  color: #aaa;
  font-family: "Roboto Condensed", sans-serif;
  font-weight: 400;
  font-size: 2em;
```

```
  line-height: 1.5em;
  -webkit-font-smoothing: antialiased;
}

/**
 * Default styles.
 */

.dimmed {
  color: #666;
}

.normal {
  color: #999;
}

.bright {
  color: #fff;
}

.xsmall {
  font-size: 15px;
  line-height: 20px;
}

.small {
  font-size: 20px;
  line-height: 25px;
}

.medium {
  font-size: 30px;
  line-height: 35px;
}
```

```
.large {
  font-size: 65px;
  line-height: 65px;
}

.xlarge {
  font-size: 75px;
  line-height: 75px;
  letter-spacing: -3px;
}

.thin {
  font-family: Roboto, sans-serif;
  font-weight: 100;
}

.light {
  font-family: "Roboto Condensed", sans-serif;
  font-weight: 300;
}

.regular {
  font-family: "Roboto Condensed", sans-serif;
  font-weight: 400;
}

.bold {
  font-family: "Roboto Condensed", sans-serif;
  font-weight: 700;
}

.align-right {
  text-align: right;
}
```

```
.align-left {
  text-align: left;
}

header {
  text-transform: uppercase;
  font-size: 15px;
  font-family: "Roboto Condensed", Arial, Helvetica,
sans-serif;
  font-weight: 400;
  border-bottom: 1px solid #666;
  line-height: 15px;
  padding-bottom: 5px;
  margin-bottom: 10px;
  color: #999;
}

sup {
  font-size: 50%;
  line-height: 50%;
}

/**
 * Module styles.
 */

.module {
  margin-bottom: 30px;
}

.region.bottom .module {
  margin-top: 30px;
  margin-bottom: 0;
}
```

```
.no-wrap {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}

.pre-line {
  white-space: pre-line;
}

/**
 * Region Definitions.
 */

.region {
  position: absolute;
}

.region.fullscreen {
  position: absolute;
  top: -60px;
  left: -60px;
  right: -60px;
  bottom: -60px;
  pointer-events: none;
}

.region.fullscreen * {
  pointer-events: auto;
}

.region.right {
  right: 0;
  text-align: right;
}
```

```
.region.top {
  top: 0;
}

.region.top .container {
  margin-bottom: 25px;
}

.region.bottom .container {
  margin-top: 25px;
}

.region.top .container:empty {
  margin-bottom: 0;
}

.region.top.center,
.region.bottom.center {
  left: 50%;
  -moz-transform: translateX(-50%);
  -o-transform: translateX(-50%);
  -webkit-transform: translateX(-50%);
  -ms-transform: translateX(-50%);
  transform: translateX(-50%);
}

.region.top.right,
.region.top.left,
.region.top.center {
  top: 100%;
}

.region.bottom {
  bottom: 0;
}
```

```
.region.bottom .container:empty {
  margin-top: 0;
}

.region.bottom.right,
.region.bottom.center,
.region.bottom.left {
  bottom: 100%;
}

.region.bar {
  width: 100%;
  text-align: center;
}

.region.third,
.region.middle.center {
  width: 100%;
  text-align: center;
  -moz-transform: translateY(-50%);
  -o-transform: translateY(-50%);
  -webkit-transform: translateY(-50%);
  -ms-transform: translateY(-50%);
  transform: translateY(-50%);
}

.region.upper.third {
  top: 33%;
}

.region.middle.center {
  top: 50%;
}
```

```
.region.lower.third {
  top: 66%;
}

.region.left {
  text-align: left;
}

.region table {
  width: 100%;
  border-spacing: 0;
  border-collapse: separate;
}
```

Complete descriptions of the customizable files are available from the GitHub `MagicMirror2` documentation pages.

PART III

Escape from the Mirrors

Ray moved along a large lane toward the amusement park buildings. Walking along what seemed the main entry, Ray couldn't avoid realizing that time left visible signs of his passage. What had previously been a curated and engaging gravel alley became an offroad path, difficult to follow. With every step, a small dirt cloud followed Ray, and this same dirt covered all the surfaces around him. It was a world that had surrendered to the ravages of time.

The large lane ended before a ride he had felt had been abandoned for years. It divided into two smaller tracks, one to the right and one to the left, following the ride's perimeter.

The ride was a 1985 "Breakdance" dancing hall and was covered by a white and red circus tent. As Ray approached the entrance, he heard a rhythmic bass inside the tent.

Following Sonya's suggestion, Ray pushed aside the curtain with one hand, put his head inside, and cautiously entered. The darkness was almost complete, interrupted only by some laser and colored flashing lights following the rhythm. Ray stepped inside but could not see anything; he needed a few minutes for his vision to adapt to the absence of light.

"Tommy always makes fun of me because I never leave home without what he calls my gadgets," Ray thought with a smile. "These are not gadgets; these are my precious tools," continued Ray, following the flow of his thoughts.

In the meantime, he removed his leather belt from his trousers. This was a hacked one, an old leather belt he received as a gift years ago on Christmas Eve. Ray transformed it into a project, one he was proud of. After pushing some buttons, not without difficulty in the full dark, the front side of the belt started scrolling the text "Searching for Jack! URGENT" in bright green characters.

Ray slowly started twisting around, keeping the belt on top of his head to make it visible from a distance.

Helped by the weak light of the belt, Ray moved to the center of the hall. He was not sure if he was alone, but he felt there were no other occupants. Of course, this was just a hypothesis; the dark made distinguishing most of the shapes impossible.

"Hey! Who's searching for Jack?," someone said from behind while a hand was touching his right shoulder. Ray turned around, distinguishing the dark silhouette of a man, at least 25 centimeters taller than him and maybe double his weight. The man took him under the arm, gently pushing Ray toward the entrance. As the two were out of the tent, the sound almost disappeared, like magic.

"I am Ray," he said, looking at the man under the sunshine. Jack was a big man with black hair. His apparently aggressive look, at first sight, was mitigated by his broad, sincere smile. The deep green eyes immediately communicated a comfortable feel to Ray. His deep and calm voice also contributed to Ray feeling comfortable with him.

"Hey, Ray, nice to meet you," Jack said. "How can I help you?"

"Well, I am searching for my son, Tommy. Sonya told me he might be around here." Jack observed Ray with a thoughtful expression.

"I see… Sorry, man, I haven't seen him. The breakdance show is one of the last on the path. If he comes here, or if I see him, I'll tell him his father is searching for him." Ray nodded and realized that the place also had a logic, an order that could be followed while visiting. He was unsure if he was moving in the right direction.

"Is there a path I should follow to visit all the marvels of this place?" Ray asked Jack, looking up at him. "I mean," Ray followed "Is there a map or something like a map that I can check to understand this place?"

Jack answered with an embarrassed expression. "There are flyers distributed at the park entrance, illustrating the attractions on one side and a map with the suggested path on the back."

"Maybe I can find one of them somewhere?," Ray asked anxiously.

"Uh, it's been a long time since I've seen one. I think that the leaflets have not been reprinted for years. The last visitors may have taken the last few."

"Understood," Ray said, discouraged.

"If I remember right, continued Jack, "It's a good idea to follow the arrows," Jack pointed to the other side of the lane, "to reach the mirror's labyrinth. You might be able to find more signs of your son."

Ray was very interested in his words. "Anyway, if you can't find him there, it has an excellent shortcut to reach the other pavilions of BDTH, where it is easier to meet someone."

Suddenly, after a quick look at his watch, Jack changed his expression. "Darn! It is late; I should prepare for the next show. If I see Tommy, I will tell him you are searching for him, don't worry!," Jack yelled, crossing the curtain and disappearing inside the breakdance tent.

Ray, alone again, crossed the lane. He saw a small sign, a white arrow sign, that reported, "Ready for the next marvel? Follow the arrows ahead." It was attached to a pole on the opposite sidewalk. He started walking in that direction, checking every detail, but finding any sign resembling an arrow was impossible. Ray walked about 500 meters straight. There were no arrows at all. He stopped, thinking about what to do.

"If there were any arrows or signs, I would have already found one," Ray murmured, coming back to where he left Jack. He decided to explore the terrain deeper and search for any small indication. To avoid missing any hidden detail, he strolled slowly, looking at the shaded zones with the help of his multifunction portable lamp.

Labyrinth

Ray powered on his multifunction portable lamp without checking the "mode" switch; it seemed the device was not working in the daylight. He shook it, thinking it had a bad contact, maybe due to the dust or some debris in his pocket. As he moved the light, a piece of blue-purple text appeared where the light was projected. Surprised, Ray checked the switch position: U.V.

"Ultraviolet!," Ray exclaimed.

Moving the UV light spot, he discovered that the ground was full of signs painted with a kind of fluorescent ink, fully transparent and hidden by regular light. Maybe an effect for the late evening and night visitors.

In fact, observing over his head, he saw a long wire of lamps that, powered off, resembled black glass bulbs.

"These are wood's lamps," Ray murmured, following the arrows reporting the text "to mirrors." In a few minutes, he was in front of "The Mirrors Labyrinth."

"What a strange entry," Ray thought.

It was at least a 4 meter wide rectangular entry with a large frame around it. The entry was some meters deep, and the walls progressively stretched until they reached the size of a small door entry.

"You're smart, dear!," said a familiar voice behind him.

"Sonya," Ray replied, not surprised. "It seems you are following me." He hadn't been thinking of her, but seeing her now, he had to admit he was pleased she was there.

"I prefer to say I am taking care of you," she commented.

"I admit, you make me comfortable in this weird situation and I'd like to have more time to spend with you, but unfortunately, I have other priorities now."

She smiled, nodding, "Maybe soon," Sonya said while he stepped inside the mirror's labyrinth. Later, Ray couldn't say if she had actually said this or if it was his imagination.

Ray had always been attracted to mazes and was a fan of enigmas. Regardless of the urgency, he was fascinated by the BDTH park and its marvels and was intrigued by Sonya's enigmatic character.

Animated by an optimistic feeling, Ray explored the first corridors of the labyrinth. As expected, of course, all the walls were mirrors, and the ground was continuous and devoid of references related to his position. There were no indications, not a single sign, to indicate the direction or the path he was walking.

In less than 15 minutes, Ray was lost.

He had no idea where he was in relation to the entrance. Of course, he had no idea of the extension of the labyrinth. It might have been a few meters from the exit. The corridors were relatively narrow, designed for walking a single person at a time.

For a half hour, Ray systematically explored a part of the maze, discovering some interesting things. Despite being sure he followed the same short path, it frequently changed. Ray found that some mirrors rotated 90 degrees or slid to reveal a new corridor or hide one. In some cases, as he walked around the corner, as a wall silently rotated, changing the labyrinth track along the way.

"This is really weird," Ray thought. "The maze is unstable, which may be why I became lost so quickly."

The mirror walls were about two and a half meters high, making it impossible to see over them or climb up. There was no ceiling, and the high sky, with its white clouds, gave Ray the sensation of complete isolation from his surroundings.

Ray also discovered another interesting fact: a small drone flew over the labyrinth at regular intervals. He supposed it was patrolling to see the visitors' progress. He had no idea how long guests could remain trapped between the mirror corridors before someone would help them. Ray had no intention of waiting for help.

Considering it almost useless to waste his time trying to escape from a continuously changing maze, Ray stopped walking and had an idea.

First, he connected a small WiFi module to his inseparable handheld computer, Palm T3, and started writing a simple Java script. The code editor was almost difficult to manage on this old appliance, but after a while, he could see if a WiFi device was present nearby.

"Now, I need to wait a while," Ray murmured. "If I am right, I need to wait another five minutes."

Ray thought about all the times Tommy had kidded him. He was still using vintage tools and considered them "invaluable." Ray smiled to himself, hearing the familiar buzzing of the drone in the air. After a few seconds, the Palm software captured the drone's WiFi address and credentials.

"I have about 25 minutes." Ray murmured, looking at his wristwatch. After another intense coding session, Ray captured a frame of the drone camera stream on the next drone passage. With his device's reduced resources, he could only get a single frame. In a few seconds, the 16-level gray image appeared on the screen. It was not much, but it was sufficient to determine the labyrinth's map.

Ray estimated it to be a rectangular surface of about 200 square meters. He was very satisfied with his result. It was now possible to find the exit regardless of the moving mirrors.

He started following the path identified in the image.

"I am lucky that the drone flies so high that a single frame is sufficient to catch the whole maze plant," Ray said to himself.

He captured a new image upon every drone passage to correct his direction according to the mirrors, which changed position. After another couple of hours and four more drone fly-bys, the mirror wall in front of him slid to one side, showing the labyrinth exit.

Outside, Sonya welcomed Ray with a smile, walking toward him.

CHAPTER 4

Machine Learning with a Drone

Contribution of Furio Piccinini, Element14.com community member.

Figure 4-1. *A drone with a mission…*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_4

During his journey, in this third tale, Ray escapes from the mirror labyrinth, hacking a drone inspecting him. In this project, I will hack a quasi-toy drone to manage and inspect an environment recognition. To achieve this goal, a small Tello drone (a sub-brand of the popular DJI) is programmed to inspect indoor areas, and an embedded machine-learning model is used to acquire environmental information and process the data on the ground with an application.

4.1. The Tello Drone

The Tello drone (www.ryzerobotics.com/tello) is a small commercial drone mainly used to teach kids in STEM workshops or simply as a high-tech toy. I was drawn to this object for several reasons. It is exceptionally lightweight (80g) and ideal for flying in small areas, especially indoors. See Figure 4-1.

Despite its small dimensions, it is a jewel of technology. It includes a camera for HD video and photos, a WiFi connection with a computer, and programmable features. It can be programmed with an extension of the visual coding editor Scratch 3.0 (ideal for teaching coding to kids). Still, it also includes complete SDK (Software Development Kit) documentation to control the drone features through APIs.

Simple commands sent to the WiFi can achieve actions such as flying up, down, right, left, shooting a photo, rotating, going ahead or forward, and more. This capability pushed me to use it for this project, so I bought one for less than $150, including the extra battery charger kit with two batteries. With three batteries, it can operate for about 40 minutes.

Figure 4-2. *The Tello drone's bare metal structure. Removing the plastic cover allows the drone to fly with an extra payload of up to 50g without losing movement stability. To test the payload, I added a small weight (white piece) to reach 50g of extra weight*

As the small device was in my hands, the first test I tried was to verify the payload practically: the drone can fly without difficulties with 40g of extra weight and reach 50g without the plastic cover shell, which does not impact the drone's functionality in flight. See Figure 4-2.

4.1.1. Programming the Drone

When the drone is connected through its private WiFi, real-time instructions can be sent through REST APIs. The user sees a RESTful HTTP server to which the flying features can be accessed.

Until I was sure it was possible to set a predefined flying path that could be repeated systematically over time, any other project parts were not feasible. The inspiring idea mimicked what Ray did in the fictional story: patrolling an indoor area multiple times to retrieve environmental data.

To achieve this step, I implemented an application on the laptop side to accomplish this first goal.

4.1.2. Autopilot Software

The Tello drone SDK documents the APIs available through a WiFi connection on private ports. From a laptop, sending fly commands, retrieving video streams, and driving image acquisition is all possible.

There are two series of REST calls to the drone WiFi server; the first, which I am not interested in, uses the UDP connection to stream the video and capture images from the drone camera.

The second is what I used for the Drone Control Python application.

The application source code acquires (from a JSON configuration file) the instructions to execute—once or looping—and send them to the drone. See Listing 4-1.

Listing 4-1. The Application Source Code

```
# Tello Python3 DroneControl application
# For more information, check the Tello drone SDK
#
# Important! To make the drone work the computer should be
connected
# to the Drone WiFi (drone access point: 192.168.10.1)s
#
# Author: Enrico Miglino <balearicdynamics@gmail.com>
# Version: 0.2
# Date: May 2022
```

```
import threading
import socket
import sys
import time
import logging
import json

# Fixed parameters to connect UDP channel to the drone
host = ''
port = 9000
locaddr = (host,port)
# Socket IP and port are hardware defined by the drone
tello_address = ('192.168.10.1', 8889)
```

The `dronecontrol.log` and `dronecontrol.json` hardcoded files define, respectively, the filenames of the log file to keep track of the events and the fly instructions.

The fly instructions are loaded in memory in a list processed after the program has set up all the parameters and established the connection to the drone. See Listing 4-2.

Listing 4-2. Global Variables and Hardcoded Names

```
# Other global variables and hardcoded names
log_file = 'dronecontrol.log'   # Log file name
log_level = logging.INFO        # Default log level (maybe
INFO, WARNING, ERROR, CRITICAL, DEBUG)
json_file = './dronecontrol.json' # Drone control file
```

If the `manual_control` flag is set to `true`, instead of loading the JSON file instructions, the program accepts direct commands from the terminal. This setting is for testing purposes and is useful while creating the JSON script file. See Listing 4-3.

Listing 4-3. Accepting Terminal Commands for Testing

```
manual_control = False          # bypass the Json file
                                processing and accept terminal
                                commands
# Current command in execution from the "fly" list
cmdIndex = int(0)

# Define the UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Lists with the drone path commands
# Commands to manage the fly along the desired path.
droneFly = ['']
# Number of times the fly sequence should be repeated
flyLoops = int(0)
```

The function `initSocket()` creates the UP socket for communication with the drone, as shown in Listing 4-4.

Listing 4-4. The initSocket() Function

```
def initSocket():
    '''
    Create the UDP socked to send and receive commands to the
    drone via WiFi access
    '''
    # Bind the socket to the host. Local address IP is
      left empty
    # as the client IP is assigned by the drone via DHCP
    sock.bind(locaddr)
    logging.info('Drone socket bound')
```

`loadDroneControl()` is called to load the fly instructions from the JSON script file, as shown in Listing 4-5.

Listing 4-5. The loadDroneControl() Function

```
def loadDroneControl():
    '''
    Load the Json drone control file in a dictionary and
    organize the list
    '''
    global flyLoops
    global droneFly

    with open(json_file) as file:
        dictionary = json.load(file)

    # Load the list of commands
    droneFly = dictionary['fly']

    # Load limits and configuration parameters
    flyLoops = int(dictionary['loops'])

    logging.info('Loaded dronecontrol Json file completed.')
    logging.info('--- Commands to execute %d', len(droneFly))
```

The `recvDroneResponse()` function waits for the drone's response. A timeout generates an error to avoid an infinite loop. All functions should wait for the return code from the REST call, because the drone's callback response sometimes takes a long time (one to two seconds). See Listing 4-6.

Listing 4-6. The recvDroneResponse() Function

```
def recvDroneResponse():
    '''
    Wait for a response from the drone after a command is sent.
    The method enable a timeout thread to avoid waiting
    indefinitely if
```

```
    some error occurs (manual stop, crash, signal lost).
    '''
    time.sleep(2.0)
```

The low-level `sendDroneCommand(cmd)` function sends the UDP REST call. A time sleep of one second before and half a second after is needed to compensate for the relatively long time required by the drone WiFi server to process the command. See Listing 4-7.

Listing 4-7. The sendDroneCommand(cmd) Function

```
def sendDroneCommand(cmd):
    '''
    Send a command to the drone via UDP and wait for the
    response
    '''
    time.sleep(1)
    msg = cmd.encode(encoding="utf-8")
    sent = sock.sendto(msg, tello_address)
    time.sleep(5.0)
```

The high-level `processDroneCommand(cmdIndex)` function processes the command and parameters from the command's script dictionary and executes the low-level call to the drone UPD server. See Listing 4-8.

Listing 4-8. The processDroneCommand(cmdIndex)Function

```
def processDroneCommand(cmdIndex):
    '''
    Send a command to the drone via UDP from the commands
    processing
    list.
    '''
    global droneFly
    sendDroneCommand(droneFly[cmdIndex])
```

The main application function in Listing 4-9 initializes the configuration files and the JSON script, starts the communication via UDP with the WiFi drone (hardcoded parameters), and starts processing the fly commands script.

Note that if the `manual_control` flag is set, the communication will proceed from the terminal for testing purposes. In any case, to enable the drone SDK after the communication has been established, the first command should be the `command` keyword.

To avoid unexpected crashes, the command processor is forced to check the battery status every minute. When the value reaches a risk point, the drone is forced to land.

Listing 4-9. The Main Application Function

```
def main():
    '''
    Main application.
    Enable the connection and if there are no errors, process
    the Json file and execute path with the drone.
    '''
    global manual_control

    # Initialize logging
    logging.basicConfig(filename=log_file, level=log_level,
                        format='%(asctime)s %(message)s',
                        datefmt='%m/%d/%Y %I:%M:%S %p')

    logging.info('*** DroneControl new session started ***')

    if(manual_control):
        logging.info('Manual control is enabled.')
        print ('\r\n\r\n* Manual Drone Control *\r\n')
```

```
print ('Available: command takeoff land flip forward
back left right \r\n
up down cw ccw speed speed?\r\n')
print ('end -- exit the application.\r\n')

receiveResponseThread = threading.Thread(target=recv
DroneResponse)
receiveResponseThread.start()

while True:
    try:
        msg = input("");

        if not msg:
            break
        if 'end' in msg:
            print ('...')
            sock.close()
            logging.info('End of application. Socket
            closed')
            break
        # Send data
        msg = msg.encode(encoding="utf-8")
        sent = sock.sendto(msg, tello_address)
        logging.info('Manual control. Sent command:
        %s', msg)
    except KeyboardInterrupt:
        sock.close()
        logging.info('End on forced keyb interrupt.
        Socket closed')
        break
```

```
else:
    # Automated fly based on the droncontrol Json file.
    logging.info('Json dronecontrol.json file loading')
    # Load the json control file
    loadDroneControl()

    # Activate the SDK APIs on the Drone
    sendDroneCommand("command")
    recvDroneResponse()
```

The command processor section of the main application function processes the dictionary commands in sequences, preparing the REST API according to the required parameters. See Listing 4-10.

In addition, the processor also supports some commands not directly related to the Tello drone SDK RESTful APIs, like the option to set the series of commands in a loop, indicating the number of times a sequence or a subgroup of instructions should be repeated. This section also takes care of the stability of the communication and closing the socket session when the series of commands ends.

Listing 4-10. The Main Application Function

```
cmdIndex = len(droneFly)
loopIndex = int(0)
logging.info("Start command processor with %d
commands", cmdIndex)
# Process the required number of times the whole
  command set
while(loopIndex < flyLoops):
    logging.info('--- Executes the command sequence %d
    of %d', loopIndex + 1, flyLoops)
```

```
            cmdPosition = int(0)            # Initial command
                                            in the list.
            # Process all the commands in the list
            while(cmdPosition < cmdIndex):
                # ----------------- Process the next command
                in the queue
                logging.info("API :" + droneFly[cmdPosition])
                processDroneCommand(cmdPosition)
                recvDroneResponse()
                # Update the command position
                cmdPosition = cmdPosition + 1

            # Update the loop counter
            loopIndex = loopIndex + 1

        logging.info('*** Session closed ***')

# Application entry point

if __name__ == '__main__':
    main()
```

The JSON Script File

```
{
    "loops": 1,
    "fly": [
        "speed 100",
        "takeoff",
        "up 50",
        "land"
    ]
}
```

This example shows a sequence script file that can be processed by the drone SDK Restful APIs.

The Log File

The creation of the log file is important to track the timecoding of the events that run during a sequence. This information is useful because it can be integrated into the ML data acquisition to position the acquisition in the correct timeframe. See Listing 4-11.

Listing 4-11. Example of a Complete Log File Describing Two Fly Sessions

```
05/30/2022 06:15:38 PM *** DroneControl new session started ***
05/30/2022 06:15:38 PM Json dronecontrol.json file loading
05/30/2022 06:15:38 PM Loaded dronecontrol Json file completed.
05/30/2022 06:15:38 PM --- Commands to execute 4
05/30/2022 06:15:46 PM Start command processor with 4 commands
05/30/2022 06:15:46 PM --- Executes the command sequence 1 of 1
05/30/2022 06:15:46 PM API :speed 100
05/30/2022 06:15:54 PM API :takeoff
05/30/2022 06:16:02 PM API :up 50
05/30/2022 06:16:10 PM API :land
05/30/2022 06:16:18 PM *** Session closed ***
06/04/2022 02:06:39 PM *** DroneControl new session started ***
06/04/2022 02:06:39 PM Json dronecontrol.json file loading
06/04/2022 02:06:39 PM Loaded dronecontrol Json file completed.
06/04/2022 02:06:39 PM --- Commands to execute 4
06/04/2022 02:06:47 PM Start command processor with 4 commands
06/04/2022 02:06:47 PM --- Executes the command sequence 1 of 1
06/04/2022 02:06:47 PM API :speed 100
06/04/2022 06:13:38 PM *** DroneControl new session started ***
06/04/2022 06:13:38 PM Manual control is enabled.
```

4.2. The Arduino Nicla ME

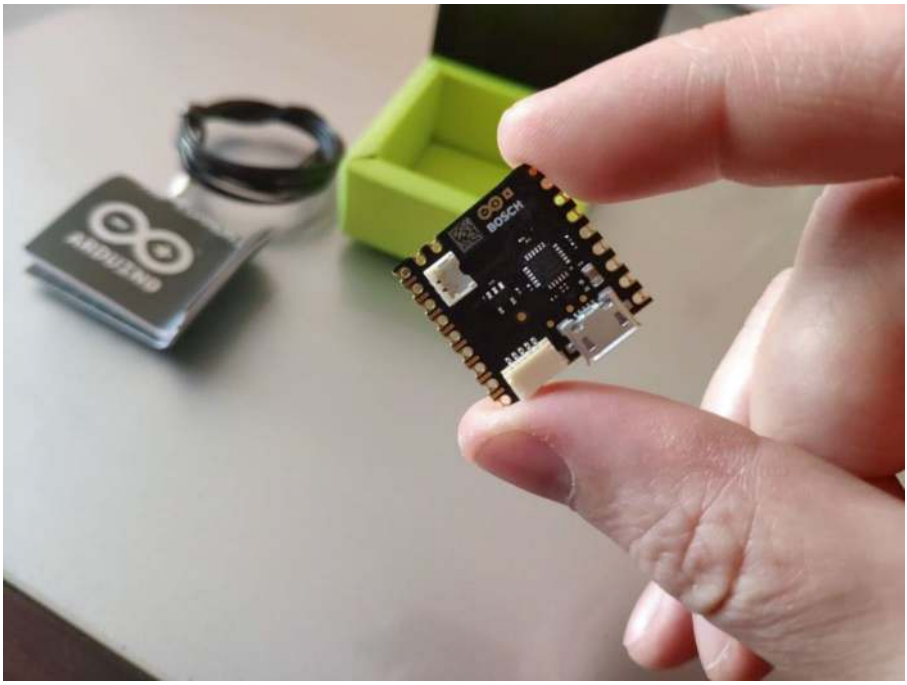

Figure 4-3. *The minimal size of the Arduino Nicla SE. This small board is perfect for collecting data from the Tello drone and sending it to the ground station (a laptop computer)*

Developed in partnership with Bosch (https://store.arduino.cc/products/nicla-sense-me), this tiny and ultra-lightweight board requires low power, supports embedded AI features, and can send data remotely through the BLE. See Figure 4-3.

The board incorporates many environment sensors, from air quality to temperature, CO2, and others, as well as direction and movement sensors and a six-axis inclinometer. See Figure 4-4.

Here are the key characteristics of the device:

- Tiny size, packed with features
- Low power consumption
- Adds sensing capabilities to existing projects
- When battery-powered, it becomes a complete standalone board
- Powerful processor, capable of hosting intelligence on the Edge
- Measures motion and environmental parameters
- Robust hardware, including industrial-grade sensors with embedded AI
- BLE connectivity maximizes compatibility with professional and consumer equipment
- 24/7 always-on sensor data processing at ultra-low power consumption

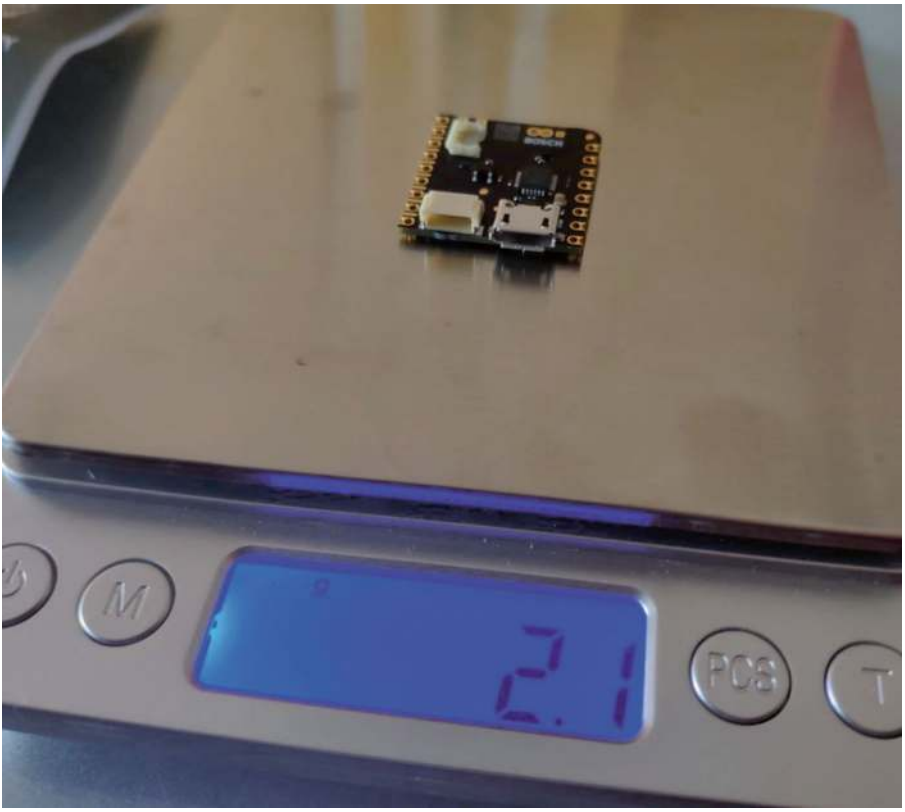

Figure 4-4. *The Nicla board only weighs about 2 grams. Including a 3.3V battery, the extra payload is considerably lower than the maximum extra weight the Tello drone can lift off*

4.2.1. Assembling the Sensor Acquisition Device

Thanks to the complete features of the Nicla board and its extremely low weight, we can create a small unit to acquire data and that is easy to fit on the Tello drone.

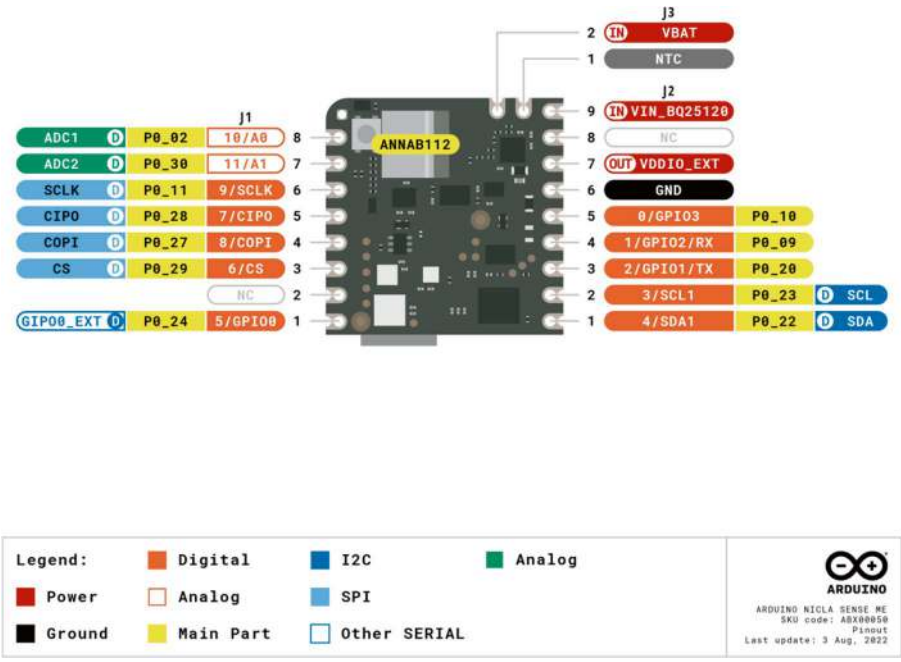

Figure 4-5. *The pinout of the Arduino Nicla. Note that there are two power pins, where the J3 (VBAT) pin accepts power from a battery*

A small 3.3V LiPo battery is directly connected to the board's GND and VBAT pins to power it. According to the Nicla pinout scheme (see Figure 4-5, keeping the battery connected to the board when a USB power source is also connected puts the battery in charge. In this case, to further reduce the weight, I directly soldered the battery to the board. See Figure 4-6.

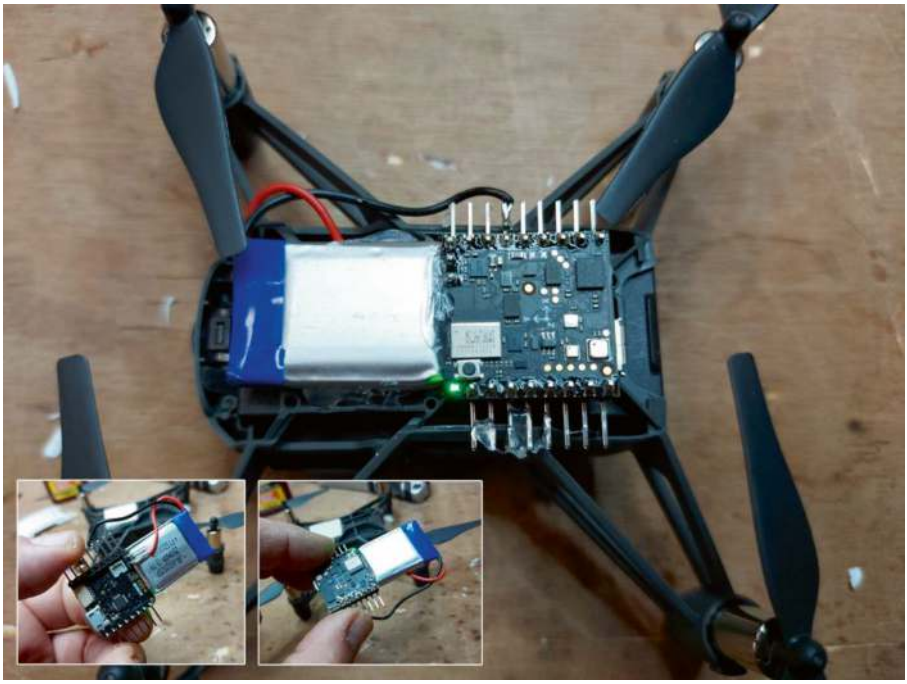

Figure 4-6. *The assembly of the board with the battery on top of the Tello drone. Soldering the battery to the board is the only wiring needed. It is also possible to add a small switch button to power the battery on/off*

Note In this project, I did not make any drastic changes to the drone itself. As a matter of fact, these are two independent devices work in synch. The drone acts as the carrier to correctly position the data acquisition system inside several points of the environment, while the Nicla board acts as the data collector.

To further minimize the weight of the modified drone, I have not included a power switch to the battery; the battery and board have been hot glued together on the top of the drone.

4.3. Nicla Bluetooth-Web Communication

The Arduino Nicla board does not include WiFi support, but as this project has been designed, BLE communication is more than sufficient for small-area coverage and indoor applications.

To efficiently retrieve Bluetooth sensor data, the Arduino source code will be set on the Nicla board, which is provided by Arduino with the Nicla MBed OS as the core operating system. This firmware configuration makes it possible to program the board with the Arduino IDE (minimum version 1.8), loading the Nicla proper libraries.

The Nicla OS includes WebBLE features, so it is possible to access the sensor's data via Bluetooth WEB. To experiment with this feature, a simple online dashboard is available on the Arduino site (https://arduino.github.io/ArduinoAI/NiclaSenseME-dashboard/). This web page is distributed as open source, and it is included in the chapter's `sources` package.

4.3.1. Customizing the Nicla Dashboard

Starting from the original repository of the web dashboard (https://github.com/arduino/ArduinoAI), I realized how it works.

Note The provided dashboard only works with Google Chrome, which supports a special feature to connect a web server (provided by the Nicla) using Bluetooth instead of the usual TCP/IP connection.

The other challenging problem was implementing data retrieval and saving on a file. For this task, with the help of the maker and friend Furio Piccinini, we developed a NodeJS engine to accomplish all the backend features.

The Go Webserver

The bridge between the WebBLE features of the Nicla board serving the sensors data, exposed to a normal HTTP webserver, was implemented in the Go language.

Note The Go sources should be compiled on the platform to receive the information. Detail about this language is available on GitHub (https://github.com/golang/go), including binary versions of the compiler for most of the platforms (https://go.dev/dl/).

The webserver code enables an HTTP port and passes all the HTTP messages to a listener that processes the requests, as shown in Listing 4-12.

Listing 4-12. The Webserver Code

```
package webserver

import (
    "arduino/bhy/static"
    "fmt"
    "log"
    "net/http"

    "github.com/pkg/browser"
)

func errCheck(e error) {
    if e != nil {
        log.Fatal(e)
    }
}
```

```
func startServer() {
    fmt.Printf("Starting server at port 8000\n")

    fileServer := http.FileServer(http.FS(static.Content))
    http.Handle("/", fileServer)
    http.ListenAndServe(":8000", nil)
}

func Execute() {
    go startServer()

    e := browser.OpenURL("http://localhost:8000/sensor.html")

    errCheck(e)

    select {}
}
```

The two sources, `parser.go` and `sensor.go`, retrieve the physical data from the board sensors and parse the raw information for use by the laptop applications.

The JSON Mapping Files

The `sensor-type-map.json` and `parse-scheme.json` files define the sensor data format, IDs, and other details (parameters, measure units, etc.) based on the technical specifications of the Nicla board provided on the Arduino site (https://docs.arduino.cc/tutorials/nicla-sense-me/cheat-sheet/).

The `parse-scheme.json` file specifically defines the data characteristics for every sensor available on the board, as shown in Listing 4-13.

Listing 4-13. The parse-scheme.json File

```
{
  "types":
  [
    {
      "id": 0,
      "type": "quaternion",
      "parse-scheme":
      [
        {"name": "x", "type": "int16", "scale-factor":
        0.000061035},
        {"name": "y", "type": "int16", "scale-factor":
        0.0000610351},
        {"name": "z", "type": "int16", "scale-factor":
        0.000061035},
        {"name": "w", "type": "int16", "scale-factor":
        0.000061035},
        {"name": "accuracy", "type": "uint16", "scale-factor":
        0.000061035}
      ]
    },
    {
      "id": 1,
      "type": "xyz",
      "parse-scheme":
      [
        {"name": "x", "type": "int16", "scale-factor": 1},
        {"name": "y", "type": "int16", "scale-factor": 1},
        {"name": "z", "type": "int16", "scale-factor": 1}
      ]
    },
```

```
{
  "id": 5,
  "type": "BSECOutput",
  "parse-scheme":
  [
    {"name": "Temperature (compensated)", "type": "float",
    "scale-factor": 1},
    {"name": "Humidity (compensated)", "type": "float",
    "scale-factor": 1}
  ]
},

{
  "id": 6,
  "type": "BSECOutputV2",
  "parse-scheme":
  [
    {"name": "IAQ(Mobile)", "type": "uint16", "scale-
    factor": 1},
    {"name": "IAQ(Stationary)", "type": "uint16", "scale-
    factor": 1},
    {"name": "bVOC-Equivalents(ppm)", "type": "uint16",
    "scale-factor": 0.01},
    {"name": "CO2-Equivalents(ppm)", "type": "uint24",
    "scale-factor": 1},
    {"name": "Accuracy", "type": "uint8", "scale-
    factor": 1}
  ]
},

{
  "id": 7,
```

```
      "type": "BSECOutputV2Full",
      "parse-scheme":
      [
        {"name": "IAQ(Mobile)", "type": "uint16", "scale-
        factor": 1},
        {"name": "IAQ(Stationary)", "type": "uint16", "scale-
        factor": 1},
        {"name": "b-VOC-Equivalents(ppm)", "type": "uint16",
        "scale-factor": 0.01},
        {"name": "CO2-Equivalents(ppm)", "type": "uint24",
        "scale-factor": 1},
        {"name": "Accuracy", "type": "uint8", "scale-
        factor": 1},
        {"name": "Compensated-Temperature(°C)", "type":
        "int16", "scale-factor": 0.003906},
        {"name": "Compensated-Humidity(%)", "type": "uint16",
        "scale-factor": 0.002},
        {"name": "Compensated-Gas Resistance(Ohms)", "type":
        "float", "scale-factor": 1}
      ]
    }
  ]
}
```

In the `sensor-type-map.json` file, I included only the sensors I am interested in retrieving data about and finalized the project. Note that every element of the sensor's JSON array corresponds to the firmware ID (see Listing 4-14). The `name` value concerns the sensor description, while the `scheme` is the class name whose data specifications are defined in the `parse-scheme.json` file.

Listing 4-14. The sensor-type-map.json File

```
{
  "4": {
    "name": "ACCELEROMETER",
    "scheme": "xyz",
    "value": 0,
    "dashboard": 1
  },

  "34": {
    "name": "ROTATION",
    "scheme": "quaternion",
    "dashboard": 1
  },

  "128": {
    "name": "TEMPERATURE",
    "scheme": "singleRead",
    "parse-scheme":
    [
      {"name": "val", "type": "int16", "scale-factor": 0.01}
    ],
    "value": 0,
    "sampleCount": 0,
    "dashboard": 1
  },

  "129": {
    "name": "BAROMETER",
    "scheme": "singleRead",
    "parse-scheme":
```

```
    [
      {"name": "val", "type": "uint24", "scale-factor":
      0.0078125}
    ],
    "value": 0,
    "sampleCount": 0,
    "dashboard": 1
  },
  "130": {
    "name": "HUMIDITY",
    "scheme": "singleRead",
    "parse-scheme":
    [
      {"name": "val", "type": "uint8", "scale-factor": 1}
    ],
    "value": 0,
    "sampleCount": 0,
    "dashboard": 1
  },
  "131": {
    "name": "GAS",
    "scheme": "singleRead",
    "parse-scheme":
    [
      {"name": "val", "type": "uint32", "scale-factor": 1}
    ],
    "value": 0,
    "sampleCount": 0,
    "dashboard": 1
  }
}
```

The Custom Dashboard

The custom dashboard uses the same procedure as the Arduino dashboard and is perfect for testing the data stream coming from the Nicla.

The Connect button also uses the Chrome browser feature to connect to the Nicla WebBLE.

This method is insufficient to manage data retrieval and make AI predictions about the environment based on the sensor's data. For this reason, I made the sensors through RESTful APIs from the Go server that should run in the background to manage the acquisition process.

4.4. Data Acquisition with NodeJS

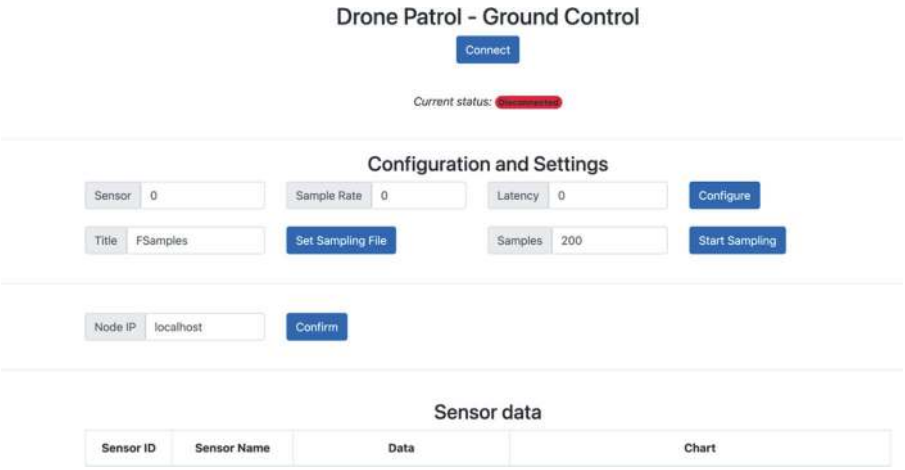

Figure 4-7. *The custom dashboard showing the data needed for the project. The NodeJS server uses the dashboard to manage the acquisition from the Nicla Sense ME*

After starting from a terminal, the Go server application with the Nicla powered on. In a discoverable proximity, we can start acquiring data from the sensors. See Figure 4-7.

At this point, a NodeJS server demands the business logic to manage the information and transform it into a dataset that can be used to train the Neuton.ai ML engine to identify the state and indoor area based on the sensor information.

Every acquisition cycle should be saved in a file with the relative timestamp in an understandable format for the Neuton.ai prediction.

4.4.1. NodeJS Architecture

The NodeJS architecture is structured through an app that connects to the Go server, acquiring the sensor's data. This information is obtained through a cyclic acquisition via a RESTful API.

The Node server connects to the Go server using different ports. When the user starts the NodeJS application, the custom dashboard described previously appears on the browser.

Interactive Dashboard

The difference between the custom dashboard used in the project and the Arduino Nicla dashboard is that the latter connects to the Go server and saves the parameters defined in the form HTTP page.

These features include, but are not limited to:

- The ID of the sensors that should be considered for the acquisition.
- Node IP to which the localhost should connect.
- Title that is also the sample filename root.
- The number of samples.

Ideally, after the NodeJS server is running and the connection between the two servers has been established, we will start the Tello drone autonomous fly and press the Start Sampling button on the page.

Note The entire source code of the NodeJS server is available in the `sources` package of this chapter. Of course, this approach to data collection is one of many possibilities, depending on the user's choice.

4.4.2. The Final Data Structure

The collected data should be cataloged with a tag to identify the sensors in a particular environment. This first step is done by creating a CSV file for every acquisition according to the `dataset_template` file.

The template defines the sensor values we intend to use to depict the environment and a tag identifying the kind of environment:

```
Temperature;Barometer;Humidity;Gas;TAG
```

Listings 4-15 through 4-17 show abstracts of different CSV values read during a series of sampling sequences acquired in two different locations: Pavia (Italy) and Dénia (Spain).

Listing 4-15. First Environment: Dénia Balcony on a Sunny Day

```
41.72;129448;29;26129;DeniaSunny
41.72;129445;29;26129;DeniaSunny
41.72;129445;29;26006;DeniaSunny
41.72;129447;29;26006;DeniaSunny
41.72;129447;29;26006;DeniaSunny
41.72;129448;29;26006;DeniaSunny
41.72;129448;29;25644;DeniaSunny
41.72;129448;29;25644;DeniaSunny
41.72;129447;29;25644;DeniaSunny
41.72;129449;29;25644;DeniaSunny
```

Listing 4-16. Second Environment: Dénia Coffee Room

```
39.06;129359;87;5377;DeniaCoffee
39.06;129416;87;5377;DeniaCoffee
39.06;129419;87;5377;DeniaCoffee
39.06;129416;87;5377;DeniaCoffee
39.06;129416;87;5330;DeniaCoffee
39.06;129417;87;5330;DeniaCoffee
39.06;129415;87;5330;DeniaCoffee
39.06;129416;87;5330;DeniaCoffee
39.06;129416;87;5351;DeniaCoffee
39.06;129415;87;5351;DeniaCoffee
```

Listing 4-17. Third Environment: Pavia on a Sunny Day

```
34.88;127950;37;7195;Pavia_sunny
34.88;127951;37;7195;Pavia_sunny
34.88;127952;37;7195;Pavia_sunny
34.88;127952;37;7156;Pavia_sunny
34.88;127950;37;7156;Pavia_sunny
34.88;127948;37;7156;Pavia_sunny
34.88;127948;37;7156;Pavia_sunny
34.88;127948;37;7180;Pavia_sunny
34.88;127949;37;7180;Pavia_sunny
34.88;127949;37;7180;Pavia_sunny
```

Joining the Sample Files

With a sample Bash shell script, all the collected samples are joined to train the Neuton.ai model. As shown in Listing 4-18, the shell script `dataset.sh` builds the samples from different locations and environments, thus creating the training model.

Listing 4-18. The dataset.sh Shell Script

```
# Create the dataset header file from template
cat dataset_template > Neuton_$2.csv
# Append data samples
cat $1*.csv >> Neuton_$2.csv
echo "Dataset created"
```

Note The full set of test samples is included in the `sources` package repository of this chapter. The complete how-to for using the Neuton.ai machine-learning engine is described in the next chapter.

CHAPTER 5

Introduction to Neuton.ai

> *Artificial intelligence (AI) is the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. Nowadays, the term is a generic definition that covers many different technologies applied in fields of research, biomedical, electronics, robotics, environment, etc. (Wikipedia)*

Unfortunately, the meaning of Artificial Intelligence (AI) is confusing to many; it generically represents a wide range of topics. For example, Machine Learning (ML), Deep Learning (DL), and neural networks are some of the many applications of AI. Indeed, the semantic algorithms that recognize text, voice synthesis, artificial vision, and many others can all be considered part of the AI domain.

An AI-powered device does not tell us so much about the nature of the application, and it is not uncommon for advertisers to use the term AI for marketing purposes only.

Among all the disciplines in the domain of AI, Machine Learning should thank its wide diffusion to the possibilities offered by the last generations of microcontrollers.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_5

Due to the complexity of the recursive algorithms and the massive amount of data involved in creating AI applications, the more computation power that is available, the more powerful and sophisticated the tasks that the computer can do. That is still true, yet small microcontrollers can now contribute to AI use.

By focusing machine performance on a single aspect, it is now possible to embed forecast features into small microcontrollers. It is like being in front of a "microbrain" that understands a single topic very well.

Tip Tiny Machine Learning and Embedded Machine Learning both fall into this particular AI application.

5.1. The AI Platform

For the design of microcontroller projects using ML—several are presented in the following chapters—I adopted the Neuton.ai Tiny Machine Learning platform. Among other AI platforms that can perform the same task—including TensorFlow, Microsoft Azure, Rainbird, Infosys Nia, Premonition, and Vital A.I—Neuton.ai specializes in creating machine learning applications for microcontrollers.

The microcontroller devices enabled for Machine Learning are part of the AI on the Edge technology. The reason is related to their application and their characteristics:

- Low power devices
- Easy to interface with any sensors
- Fit perfectly in IoT devices (home automation, lighting control, weather forecast, etc.)

- Easy to integrate into appliances (washing machines, automotive, door opening, automated parking control, media devices, etc.)
- Low cost
- Easy to integrate outside on the field

These microcontrollers can be used in two ways. They can operate as standalone machines providing feedback directly to the appliance they control. Alternatively, the microcontroller board includes Bluetooth or WiFi to connect to another device or a central unit and send data directly to the cloud for further processing.

5.2. Machine Learning Workflow

Before stepping into a real-world example illustrating how to use Neuton.ai to optimize the process, I clarify the main steps defining the scenario of Machine Learning for embedded devices.

5.2.1. Dataset Creation

The first step is to identify what kind of data you need. For example, if you want to detect a gesture, an accelerometer's data should change in time. Alternatively, you could recognize a gesture in at least two other ways:

- Point a camera at the subject and extract motion data from the frames.
- Detect a motion that interacts with the subject with infrared or other spatial sensors (e.g., ultrasonic sensor, PIR, etc.)

When the method you want to use for every different gesture is straightforward, you should collect big sensor data to create a spatial description of this gesture.

Using an accelerometer, for example, you can collect some thousand repetitions for every gesture and tag the block of data with the gesture name. Using an accelerometer, you need to collect thousands of readings for each gesture, and then tag the corresponding data blocks with the gesture's name.

The dataset creation is the most tedious part, but data collection accuracy makes the dataset helpful for gesture recognition.

The data collected in several sessions should be the same kind to be comparable. This means creating a dataset representing the model for every gesture you want to recognize.

5.2.2. Dataset Normalization

Neuton.ai—and in general, other AI platforms—can process a normalized dataset representing the model of the event, object, or condition to predict. The data should be uploaded to the platform as CSV (Comma Separated Value) text files. The first line of the file contains the name of every column, like in this example:

Data should be format-coherent and expressed in the right unit, according to the kind of sensor used. This gives the algorithm a good chance to analyze the data, developing a neural network for predicting and recognizing an unknown set.

Collected data—through sensor reading, data generation, simulation, etc.—is responsible for providing a coherent and comparable dataset for the training model and the test reading for prediction and recognition. See Table 5-1.

Table 5-1. *Series of Normalized Data from the Dataset Used in the Neuton.ai Use Case "Gearbox Fault Diagnosis" (https://lab.neuton.ai/#/cases)*

Sensor Data Fragment from a Test Series			
a1	**a2**	**a3**	**a4**
-0.16937761238	-1.2820543584	3.3027539435999995	-1.5569588601999997
3.9457410835999998	-0.22090258186	-0.0034851002966	-0.17464850696
0.88871022544	0.6942371149800001	-0.035490590174000004	-0.4702505948
-2.3327733436	-1.5788384226	1.3059838798	-1.2955140891999999
1.7554748897999999	2.0848483022	0.34871502555999995	0.5027409449800001
2.4869702596	-0.62464950676	0.9410951777199998	0.25570988570000003
-0.5938061236400001	-0.31905961867999993	1.8753324926	-1.1444671102
0.18368032631999998	0.9592888138399999	0.26345873072	-0.87423351498
0.9494980096600001	-0.27133757314	-0.21703965912	-1.676866462

5.2.3. Model Training

The AU framework, which usually resides on a high-performance local computer or a server on the cloud, has an engine to process the dataset model creating the neural network. This is the so-called "neural network training."

Analyzing the distribution of the data in the model (the dataset), the AI engine (the neural network) creates its internal logical network representing any single tag (a group of coherent data, that is, a dataset row) as a recognizable identity.

When the training ends, the neural network can identify an unknown dataset of the same kind of model with higher accuracy. Depending on the complexity of the model, the volume of data, and the available computational power, the training process can last a few seconds, minutes, or hours.

5.2.4. Prediction

Prediction is the phase where the AI-trained neural network expresses its own "intelligence." Typically, using a specific API call to the AI server, you can send a single group of data to the engine that, according to the training, can predict what kind of event or object it is. For example, what happens in the AI engine? First, it searches for the best fit of the unknown dataset inside the neural network. The prediction accuracy is directly related to the volume of data used in training; more extensive training data typically leads to higher accuracy.

After completing the training, the neural network can be represented as an abstraction using almost any programming language, and the data model is no longer needed.

By compacting the neural network—many algorithms can do this—it is possible to create, for example, a C language library and a small function interface that can do almost the same job as the more complex

neural network created on the server. This compact version of the prediction engine can be included in a microcontroller, which provides the data needed for the prediction directly connected to the sensors. The conversion of the neural network to a programming language makes it possible to embed the prediction features into the microcontroller firmware, the embedded Machine Learning device.

In a few words, the microcontroller supporting the embedded ML feature incorporates a program library containing the neural network encoded as source code—usually created in C.

Of course, the engine that trains and configures the parameters to process the neural network resides on the AI framework server. Creating a new software library requires re-creating a new one to reconfigure the engine, changing some rules or control parameters, or adding more data to the training. The microcontroller firmware should be rebuilt on the platform server.

5.3. The Neuton.ai Framework

After having worked with TensorFlow for small ML projects on microcontrollers, I started adopting the Neuton framework for two main reasons:

- The engine creates robust neural networks specialized in managing predictions with microcontrollers.
- The workflow is easier to manage than other more comprehensive alternatives.

This platform has limitations that do not impact the prediction performance; it is a tailored platform specialized in ML applications. The linear workflow makes it easy for users to go straight to the goal, and the process can also be customized for a wide range of possibilities.

As the platform dataset only accepts data in CSV (Comma Separated Value) format, it is virtually possible to manage sensor information provided that the source data is converted into CSV format.

To connect to the platform, you need to sign in with a Google account, which manages the data stored by the framework on the Google cloud in the account's personal space. See Figure 5-1.

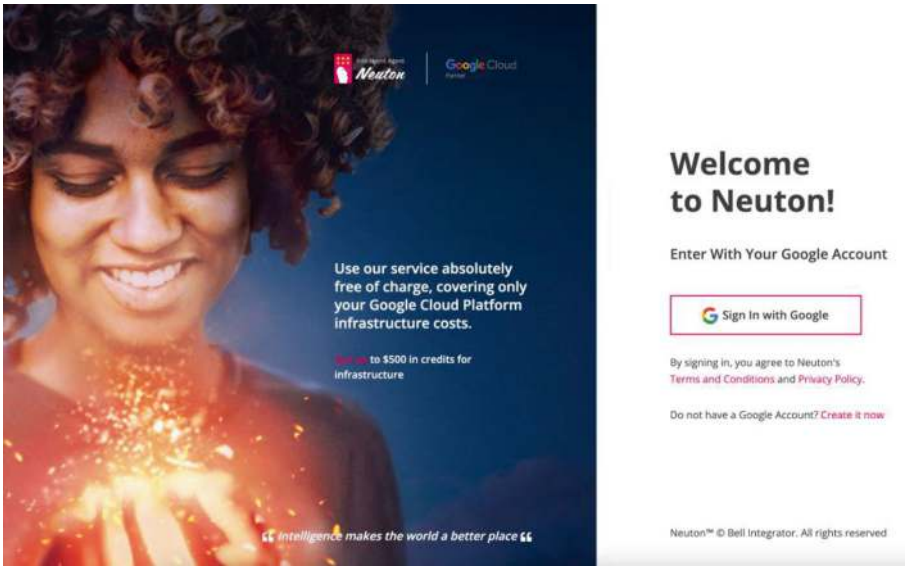

Figure 5-1. *The lab.neuton.ai home page*

5.4. Creating a Solution with Neuton.ai

For a better understanding, I used images to illustrate the process of creating a solution using the Neuton platform. From the Neuton Google account shown in Figure 5-2, you create projects and check the data storage and parameters.

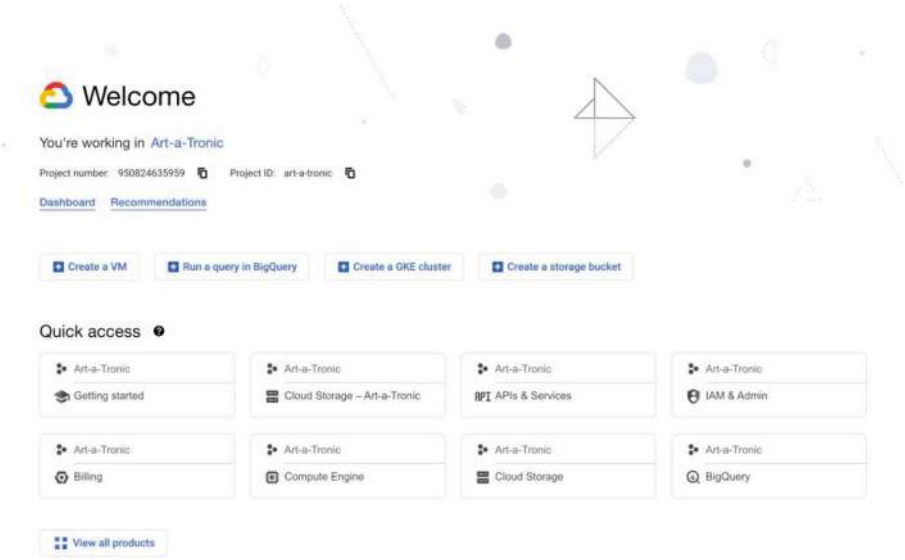

Figure 5-2. *The Google cloud site, which is also accessible from the Neuton main menu*

The Neuton platform temporarily uses the Google cloud to store the AI engine's data. A Google account is needed to access the Neuton site (lab.neuton.ai). The same Google account used to log in to the Neuton platform should be subscribed to the Google cloud.

A billing registration is required for the subscription. However, for many experiments, using just the Neuton platform, Google's free tier (which provides approximately 300 USD for every new subscription) is enough. All of Neuton's features are free for non-commercial purposes.

Warning The Google account is accessible from the side menu of `lab.neuton.ai`. However, a solution can't be created if the logged-in account is not subscribed to an active Google account. After subscribing, you can use the platform for free, paying only for Google's infrastructure costs.

You can create solutions with Neuton for free by registering for a Zero Gravity account. It is a good idea to explore the use cases available from the sidebar menu to understand in detail the workflow variations offered by the platform. By the way, audio and keyword spotting tasks that are relevant and applicable to many AI solutions can also be successfully solved using Neuton.

From activity recognition to mechanical diagnostics, sensors management, and more, these use cases represent examples of real-world applications. Every use case is explained in detail, while the original trained and test dataset can be downloaded locally. This makes it possible to replicate these examples, covering many of the possibilities offered by the platform.

If you select the My Solutions sidebar button, the page shows the list of solutions already created by the currently logged user.

When a solution has been saved and closed—the training engine is not working—it is available but doesn't consume Google cloud resources. Every solution implies using a dataset, which is a group of files created automatically by the platform when importing the source data CSV file. All the processed datasets are stored in numeric folders on the user's Google cloud space. The dataset can be reviewed, downloaded, or deleted anytime, by accessing the list from the Dataset Storage button on the sidebar. See Figure 5-3.

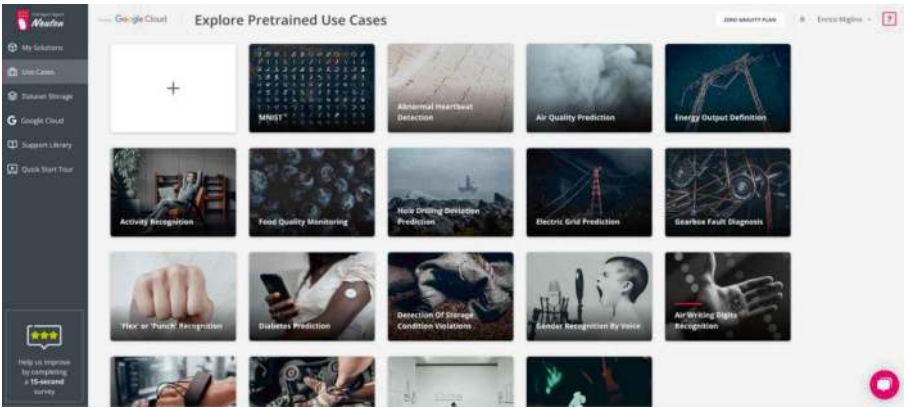

Figure 5-3. *The Neuton.ai use cases page, where you can open and explore complete solutions*

5.4.1. Step-by-step Solution

Look at the Gearbox Fault Diagnosis use case: Detects broken tooth conditions in the gearbox based on the vibration data.

To experiment with creating a solution, this section replicates this example. First, download the two solution files: the dataset used to design and train the model and a series of test data to verify how the prediction works.

5.4.2. A Few Words on this Use Case

Even though the algorithm operates on a series of vibration sensor values, the neural network process can deliver robust and consistent predictions with high reliability. See Table 5-2. Note that, in addition to the sensor data (the first four columns), the Target column classifies every row of the dataset.

Table 5-2. *The First Ten Lines of the Dataset Used to Train the Neural Network*

Gearbox 10-40-90 Training				
a1	**a2**	**a3**	**a4**	**target**
-0.169381	-1.2820799999999999	3.3028199999999996	-1.5569899999999999	0
3.94582	-0.22090700000000002	-0.00348517	-0.174652	0
0.8887280000000001	0.6942510000000001	-0.0354913	-0.47026	0
-2.33282	-1.57887	1.3060100000000001	-1.29554	0
1.75551	2.08489	0.348722	0.5027510000000001	0
2.4870200000000002	-0.6246619999999999	0.9411139999999999	0.255715	0
-0.5938180000000001	-0.31906599999999996	1.87537	-1.14449	0
0.183684	0.9593079999999999	0.263464	-0.874251	0
0.949517	-0.271343	-0.21704400000000001	-1.6769	0

The dataset consists of a series of vibration sensor samples depicting two scenarios: when the gear has a broken tooth and when the gear is intact.

Open the Solutions page and choose Add New Solution. Assign a name and optionally an introductory description. In this window, you should inform the platform about the kind of data, organized into three main categories:

- **Audio**: Refers to datasets like frequency samples in the range of 20-20.000 Hz, which is the audible audio frequency range.
- **Sensor data**: Refers to any homogeneous data collection from sensors. It is the category to select.
- **Tabular data**: This kind of data can't be embedded in a microcontroller due to the massive amount of information, so it's not relevant here.

5.4.3. Step 1: Upload the Dataset

As mentioned, to be accepted by the Neuton platform, the dataset should be prepared as a CSV file. In this example, every record will contain the measured fields and, lastly, a number. This number is the classification used by the ML engine as the TAG, where 0 means healthy gear and 1 or more means one or more broken teeth. See Figure 5-4.

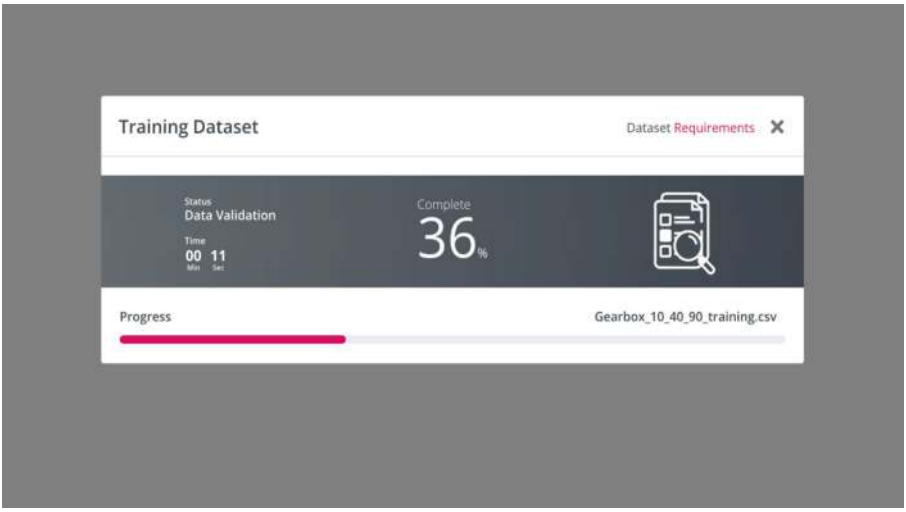

Figure 5-4. *After the training dataset has been uploaded (or chosen from an existing file on Google cloud), the imported CSV file is checked for validation. If the file does not respect the dataset requirements, the platform send a notification*

After uploading, the dataset will be verified automatically to avoid computing errors or issues in the following steps.

Note If the training must be replaced—for example, because new data has been added—it can be deleted and replaced. After the training data is uploaded, it is possible to retrieve it from the Google cloud storage at any time, for example, to apply the same dataset to another project or change the inference parameters.

5.4.4. Step 2: Train the Dataset

At this point is possible to customize the dataset—to which you attribute a meaning and address the engine for proper training according to your needs.

On the right side of the window, you select the target variable, while on the left side, you select all the fields that will be part of the data variables. If none of the fields on the left is chosen, all the data will be included when training the neural network.

In the second window of the training settings, more details are available. The kind of model settings depends on the meaning you attribute to the data and how the AI engine should consider it from a numerical point of view. See Figure 5-5.

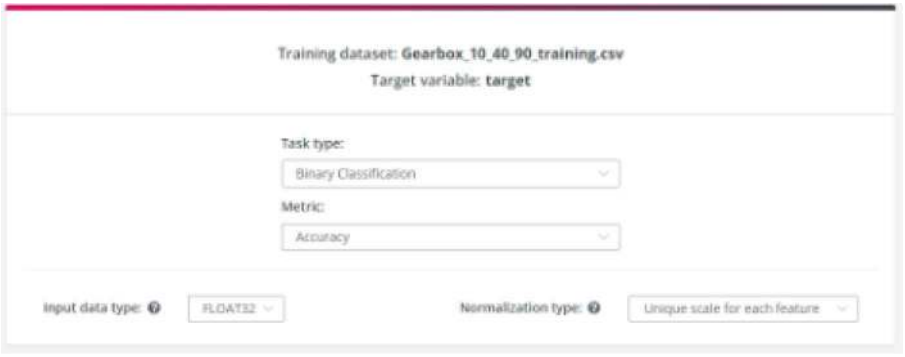

Figure 5-5. *The second screen of the training dataset configuration, where it is possible to select the kind of ML task. In this case, you need a binary classification*

Leave the proposed default settings untouched in this experiment and press the Start Training button.

While the training process is underway, a training status window shows the progress corresponding to the amount of data processed from the dataset. You should expect that as the progress indicator rises, the accuracy increases correspondingly. At the bottom of the page, a dynamic multi-dimensional graph shows how well the trained neural network is "able to understand" an unknown set of data according to the provided dataset. See Figure 5-6.

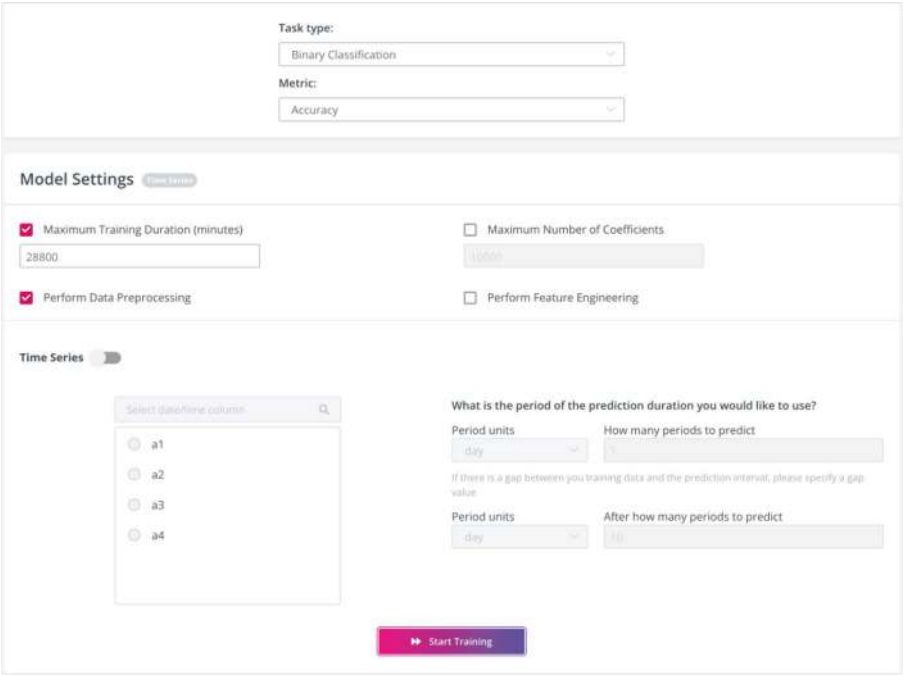

Figure 5-6. *The second screen of the training dataset configuration, where it is possible to select the type of ML task. In this case, you need a binary classification*

For digital signal processing, use the Windowing option. If you have collected the data and know the window size for each sample, input the size corresponding to your case. Otherwise, select Auto Determination, and Neuton will choose the optimal window size.

You can also experiment with feature extraction to make the model smaller and more accurate. You are free to edit the number and type of features to be extracted from each variable. Choose a method of subsequent feature elimination and remove lag features by clicking the corresponding checkboxes. See Figure 5-7.

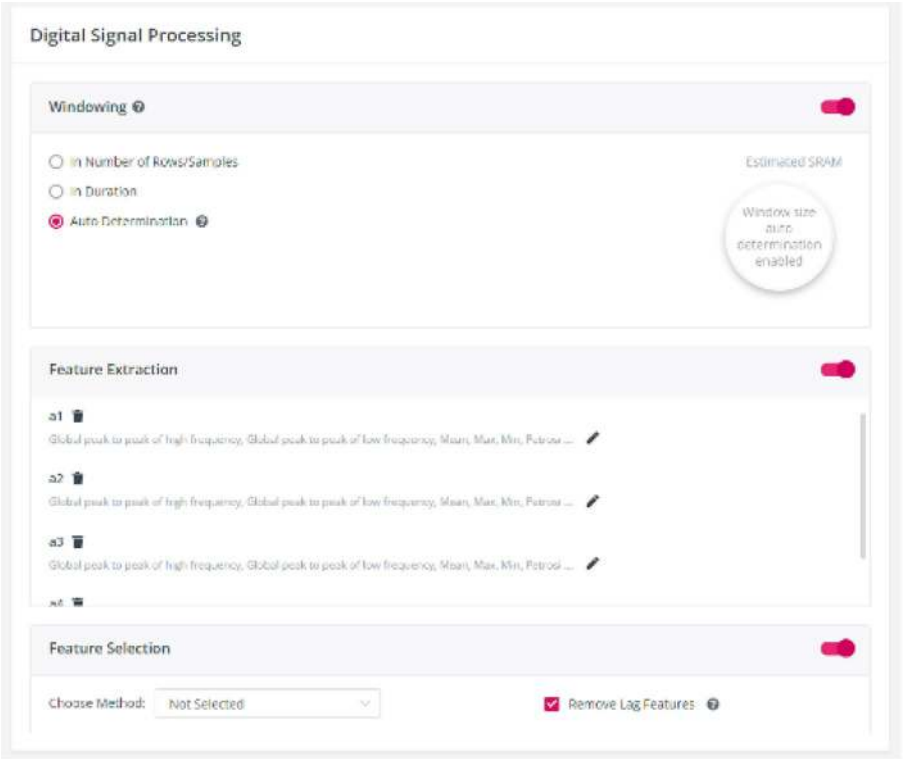

Figure 5-7. *Digital signal processing signal*

Select the bit-depth of the model. To achieve the best model size and inference time, choose the 8-bit precision option. If you have restrictions on the model size, set the maximum number of coefficients here. See Figure 5-8.

Figure 5-8. *Model settings*

During training, a training status window shows the progress corresponding to the amount of data processed from the dataset. You should expect the accuracy to increase as the progress indicator grows. At the bottom of the page, a dynamic multi-dimensional graph shows how well the trained neural network is "able to understand" an unknown set of data according to the provided dataset.

If the accuracy doesn't reach the desired level, you'll need to review the dataset and produce a new one with better cases describing the model.

The training phase may take a considerable amount of time to complete. Training the dataset used in this example takes a couple of hours. However, you can configure an SMS notification on the platform to alert you once the training is complete.

5.4.5. Step 3: Download the Ready-to-Use C Library

The first option is the one this example is most interested in: using a C library to embed the predictive neural network on a microcontroller. See Figure 5-9.

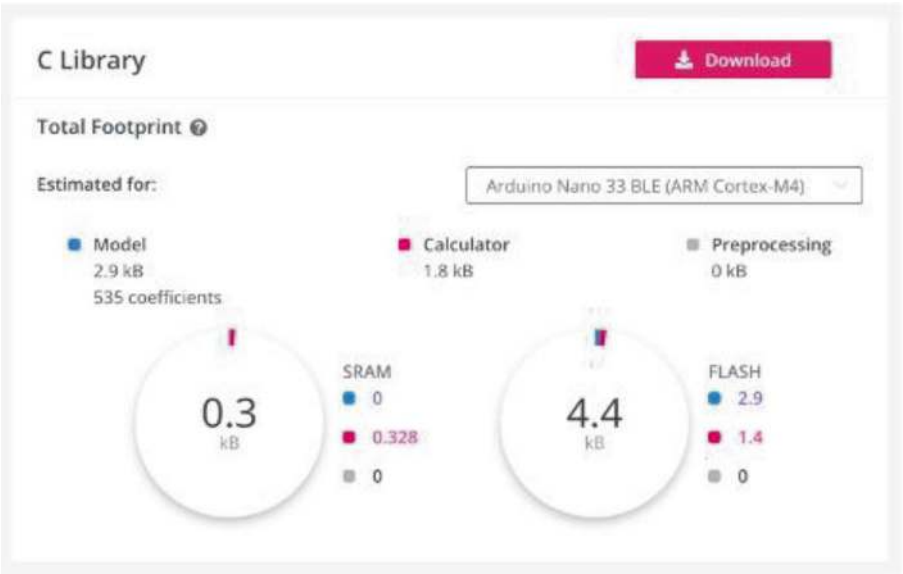

Figure 5-9. *The prediction download box of the C language library for microcontroller ML embedding*

The C language embeddable library—downloaded as a compressed file—includes all the sources and a readme file explaining how to implement it in the microcontroller code and how to use it.

The archive contains a library for inference with the following files:

- `neuton.h`: The header file of the library
- `neuton.c`: The library source code
- `model/model.h`: The model header file
- `StatFunctions.h` and `StatFunctions.c`: The statistical functions for preprocessing

All files are an integral part of the library and you don't need to make any changes to them.

The library is written following the C99 standard, so it is quite universal and does not have any strict requirements for your selected hardware. The ability to use the library depends mainly on the amount of memory available for its operation.

The deployment into the firmware project consists of the following steps:

- Copying all files from the archive to the project and including the header file of the library.
- Creating a float array with model inputs and passing it to the `neuton_model_set_inputs` function.
- Calling `neuton_model_run_inference` and processing the results.

It's time to download the model and run inference on a device. Neuton's models are so compact that they can run natively on memory-constrained MCUs, even with 8- and 16-bit precision.

Another option is *prediction*, which applies to an unknown small dataset the neural network created with the model. With a model sufficient to describe all the possible conditions, the neural network can provide high-accuracy predictions over the unknown data.

The prediction is made on a small, unknown dataset using the neural network model. If the model adequately covers all possible conditions, the neural network can deliver high-accuracy predictions for the unknown data.

Note The Neuton prediction module requires a virtual machine containing the solution data; this operation consumes Google cloud resources. For this reason, it is worth enabling the module only when it is needed.

The Neuton prediction window provides the Web Prediction and a REST set of API methods. The Web Prediction can be used to test the neural network. It accepts a test data file of the same format and fields of the dataset model, without the Tag column. After the data upload, the prediction generates a downloadable file and a preview. See Figure 5-10.

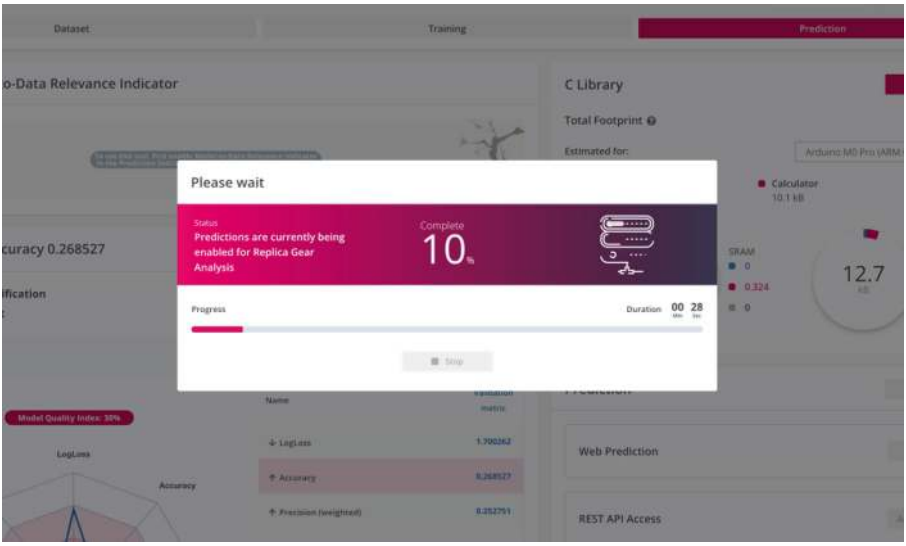

Figure 5-10. *The prediction starts the virtual machine on Google cloud containing the neural network and the prediction engine—every solution has its own VM*

For every record stored in the test file, the prediction table shows the predicted target according to the neural network applied to that specific value. See Figure 5-11.

Figure 5-11. *The web preview prediction table. Consider this preview for testing purposes while using the more complete RESTful APIs for integration in your development (when developing a web application)*

Indeed, the raw numbers in the downloaded CSV formatted file are not so good for a real-world application.

I mentioned that for every solution—when the prediction is enabled—a dedicated virtual machine runs on the cloud. The predictive neural network runs on a dedicated server whose features are available via a RESTful call (there are options for `POST` and `GET`). The server exposes four APIs:

- `/v2/predict`
- `/v2/predict/file`
- `/v2/predict/status/(UUID)`
- `/v2/predict/result/(UUID)`

The API's calling mechanism is shown in a scriptlet that's available in four different languages: Python, Java, C#, and Scala. It is easy to integrate into your code. See Listing 5-1.

Listing 5-1. The Python Sample by Neuton.ai to Connect to the Google Cloud Server Running the Prediction from a Web Application via RESTful API Calls

```
import sys
import requests
import json

def send_request(URL, file_path):
    upload_file = requests.post(url + '/v2/predict/file',
    files={'file': open(file_path, 'r')})
    uuid = json.loads(upload_file.text)['data']
    print('Request has been sent.')

    _is_processed = True
    while _is_processed:
        check_status = requests.get(url + '/v2/predict/status/
        {}'.format(uuid))
        _is_processed = is_processed_file(check_status.text)
    print('Request was processed')

    get_result = requests.get(url + '/v2/predict/result/{}'.
    format(uuid))
    file_name = "result_{}.csv".format(uuid)
    with open(file_name, 'w') as file:
        data = json.loads(get_result.text)['data']
        file.writelines(data.split('\n '))
    print('Result saved in file: {}'.format(file_name))

def is_processed_file(response) -> bool:
    _status = json.loads(response)['data']
    return "SUCCESS" != _status and "ERROR" != _status
```

```
if __name__ == "__main__":
    from argparse import ArgumentParser
    parser = ArgumentParser()
    parser.add_argument('--url',
                        dest='url',
                        required=True,
                        help='Ready solution URL')
    parser.add_argument('--file_path',
                        dest='file_path',
                        required=True,
                        help='Path to file for prediction')
    args = parser.parse_args()
    send_request(args.url, args.file_path)
```

Warning Remember to disable the prediction VM when it is no longer needed!

The third option—available without enabling the prediction virtual machine—is the one this chapter is most interested in: using a C library to embed the predictive neural network on a microcontroller. See Figure 5-12.

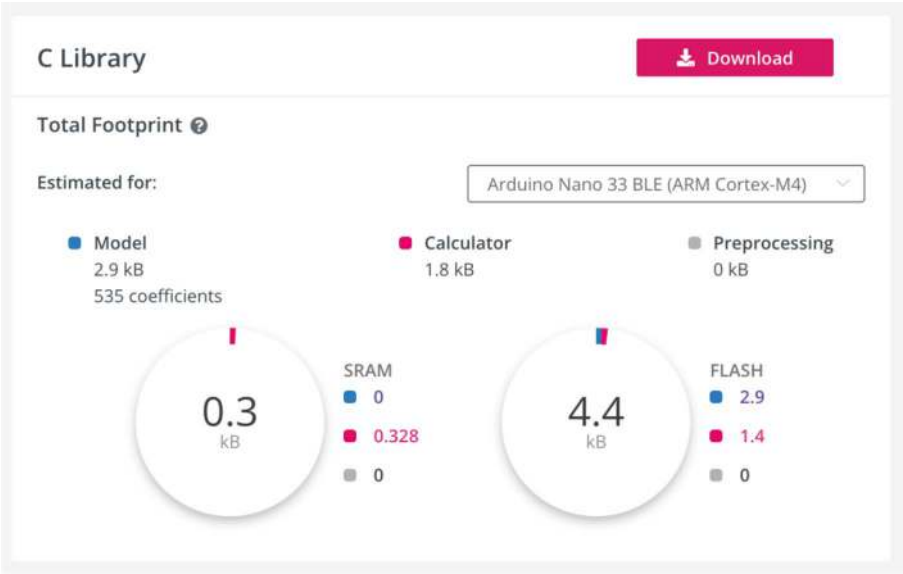

Figure 5-12. *The prediction download of the C language library for microcontroller ML embedding*

The C language embeddable library—downloaded as a compressed file—includes all the sources and a readme file explaining how to implement it in the microcontroller code and how to use it.

PART IV

A Path of Sounds

Ray was so happy to exit the mirrors' labyrinth that it was difficult to control his impulse to hug Sonya.

"Hey, Ray," she said. "Happy to see you," Ray answered.

A bit embarrassed by the situation, Ray put his hands in his trouser pockets and looked at his surroundings. A short gravel lane connected the mirrors' exit to the "Music Garden." He started walking in that direction, followed by Sonya. It was a quiet green place with short, curated grass. The sun filtered through the large branches of the oak trees. Little platforms of different shapes were in the distance, on top of small hills.

Ray walked to the nearest platform, about 500 meters away, and Sonya followed beside him.

"How did you reach the exit from the maze?," she asked, breaking the silence. Ray explained his strategy, hacking the drone. Sonya took him gently under the arm. He tried to behave like everything was normal, but it wasn't.

It was complex to deal with his contrasting emotions between the sweetness of that moment and the task he was focused on: finding Tommy.

"It is strange to me being here, walking together. You always disappear so quickly."

The moment he said this, he thought he ruined that moment. Surprisingly, Sonya instead moved a bit closer to him. Ray could perceive the warmth of her arm, experiencing a long forgotten sensation.

"I am allowed to accompany you through the Music Garden," she answered.

"You should know I will still be with you when I disappear."

"I should admit that finding the exit from the maze was challenging. I felt like I was under a weird exam when I passed those thousand doors," Ray answered.

"From a certain perspective, you are right," Sonya replied. "But we are in an amusement park, Ray, so why not enjoy it for a while?"

Her answer left Ray with more unanswered questions than before, but he decided to go with the flow and see what would happen.

When they climbed up the hill, Ray could see the group of musicians better. He was surprised to discover that they were all robots! The musicians had drums, a guitar, a trumpet, a contrabass, and a singer. They played a pleasant Dixieland repertoire. Observing closely, Ray saw they moved by a complex system of levers and gears, mainly hidden by the platform. Indeed, the brain of this fascinating engine was lying inside the platform.

"Ray, I should go now," Sonya said, interrupting Ray from observing the band. "You can go alone from here."

"Follow the bands on the hills, and you will easily cross the park." Without waiting for a reply, she left a soft caress on his shoulder, freed her arm, and went away. Ray was speechless at so many scene changes; Sonya was already far away when he answered.

"See you," Ray said softly. In the distance, Sonya waved her hand without turning around.

The next group was a trio playing classical music. Ray followed the track, enjoying almost every musical genre on a warm, sunny day. He was in front of a rock band playing famous 60s songs when they suddenly stopped playing. In a few seconds, the air was saturated with strange sounds coming from everywhere. It was like a rhythm played by mighty bells. Music without melody. The bells made an oppressive sound, a sequence of notes and resonances.

A single powerful sound filling the silence.

In about five minutes, the bell notes ended, leaving a long echo in the air. Then, as if following a silent order, everything returned to normal.

Ray was attracted by two tango dancers following the sound of a guitar and a bandoneon. He was very curious to see what kind of engine could move the two dancers so fluently. Behind a background curtain, to the opposite side of the scene, a man sitting on a chair was moving his hands like a master of puppets.

"It's you? Are you doing this show?," Ray asked the man on the chair.

"Yes, Sir. It's me," he answered. "You see, I make the movements in front of the camera," he said, indicating a lens in front of him. "And the automatons move and play the music." Ray was very curious about this technique.

"I see. So, all the musicians around," said Ray, indicating the surroundings with a wide gesture of his arm, "are robots. An incredible piece of art, a masterpiece!"

"Yes, it is. I am Greg, from maintenance service," He answered. "Nice to meet you."

"I am Ray," he replied.

"You are welcome, sir. I heard of you. You are that man searching for his son," Greg answered.

"I suppose you haven't see my son," Ray asked with a discouraged expression.

Greg started speaking again. "Actually, sir, I think I saw him. He crossed the Music Garden about one hour ago. Don't worry, he was strolling, enjoying the music. You will meet him soon."

In the meantime, the song finished. Greg moved on the chair, and Ray observed a series of cables connecting his back to the ground.

"What are these cables on your back?," Ray asked. "It seems you, too, are connected to an engine." Ray said.

"These are my connections, sir. We all are robots." He answered very earnestly. This was the last answer Ray expected.

"I have seen nothing similar in the other stages." Ray commented.

"True," Greg said. "Every stage plays by itself. Unfortunately, the engines are old, and sometimes one of them needs to be operated manually by me or another colleague. Then, during the night, the engine is repaired."

Ray heard Greg's admiration for the music installation in his words.

"Thank you for the very detailed explanation. And thank you for giving me some news about my son."

"You are welcome, sir. It's my pleasure. It is rare to be able to chat with a visitor."

Ray was about to leave the stage but turned again. "Greg, is there a strategy to follow the stages to reach the exit of the Music Garden?"

"Sure, that is easy," Greg answered. "When you reach a stage, walk to the nearest one you can see. In this way, you can easily reach the exit of the garden. Be careful; stages move to a different position inside the garden every night, creating a new path daily." Ray nodded. He was already used to the curious habits of BDTH 6159.

"Thank you again, Greg. And let me say you all are doing a great job."

After passing three other small musical stages, Ray found himself on top of the last hill. He saw the landscape sloping to a meadow surrounded by a fence. In front of him, the fence was interrupted by a wooden gate with a big sign on top:

"EXIT. Thank you for the visit."

Ray stepped out of the gate and met a doorman in a uniform. His reminded Ray of one of those revolutionary costumes used by the Beatles in the "Yellow Submarine" movie. As he approached the doorman, he spoke to Ray.

"Mr. Ray, I suppose," Ray widened his eyes, surprised and speechless.

"News travels fast, sir," he continued.

"In a connected world where almost nothing has happened in some years, a visitor is exceptional news, sir."

"Please, call me Ray," he answered. "I suppose you also know the reason I am visiting the park."

"Of course, Ray. This side of BDTH has a fast network; we all know why you are here."

The doorman was speaking with a deep voice. His sentences had no accent, and the sound seemed artificial to Ray.

"Sorry for my voice, Ray," he answered before he finished the sentence. "I use my voice so rarely that the assistance service ignores my maintenance. My speaking capabilities are, therefore, degrading. It has been over a year, and no one cares about me." The doorman explained sadly.

Observing him, Ray noted that only the upper part of his body was moving: his arms, hands, torso, and, of course, his facial expressions.

"Let me try to help you, Ray," the doorman said. "As you know, one hour or so, your son—he is your son, right?" "Yes, Tommy," Ray answered.

"I apologize for not introducing myself. I am model 736, 5.4 version 2.3, but you can call me Magnus," he continued.

"Okay, Magnus," Ray answered, a bit anxious.

"Please continue. What do you know about Tommy?"

"Ah, yes. As I told you about one hour or so, Tommy exited the Musical Garden. He followed the map on the back of the leaflets distributed to the visitors at the entrance." Ray was impatient, but Magnus spoke slowly with his monotone voice.

"If he follows the map, he will go to the Sandcastle attraction."

"Can you be more precise about the direction I should follow, Magnus?," Ray replied.

"Of course. Follow the alley for about 500m, then move to the right. At that point, you should see the indications for the dome."

"Many thanks, Magnus," said Ray while quickly taking his leave.

"It was a pleasure to chat with you, Ray," Magnus replied. "I would be happy to accompany you, but as you can see..." he said, pointing to the bottom of his body, anchored to the platform.

Ray smiled gratefully and disappeared from Magnus' sight in a couple of minutes.

CHAPTER 6

Introduction to MIDI

Figure 6-1. *The first prototype of the MIDI cardboard drum while recording a rhythm track on my Mac*

> *Following the arrows, the path drives Ray Badmington to a source of dancing music. It's incredible—but not magic (Ray is an engineer)—the original rhythm and noise generated by this unique source. He wonders about the captivating*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_6

> *sound inside the circular dome, filled with dancing holograms surrounded in the center by that skinny Rasta man playing alone over a couple of strange-shaped boxes.*

As you already know, the adventurous world of `BDTH6159` is an incredible place. Still, thinking about the real world, without special components and futuristic technologies, you can do something similar or—why not—something better. See Figure 6-1.

6.1. The Trick Is MIDI

Figure 6-2. *Some electronic instruments and effects "chained" together through the MIDI bus. Thanks to the flexibility of MIDI, a small MIDI bus allows you to send the same MIDI message from a source to multiple sources. Indeed, one of the devices is set as a "master clock" to keep all the others synchronized*

MIDI is the acronym for *Musical Instrument Digital Interface*. According to the Wikipedia definition, it is a technical standard that describes a communication protocol, digital interface, and electrical wiring that connects a wide variety of electronic musical instruments, computers, and related audio devices for playing, editing, and recording music.

Nowadays, MIDI is the universal and popular standard available on almost all electronic instruments and musical effects, taking advantage of the most recent communication protocols like USB, Bluetooth, and WiFi.

It was October 1982—over 40 years ago!—when Robert Moog, president of Moog Music, announced the MIDI standard. In 1983, Dave Smith, president of Sequential Circuits, demonstrated the MIDI interface by connecting a Prophet-600 and Roland JP-6 synthesizer. In the same year, Roland commercially released the Prophet-600, the Jupiter-6 synthesizer, the TR-909 drum machine, and the MSQ-700 sequencer equipped with the MIDI interface.

After the MIDI 1.0 specifications were published in 1985, the standard continued evolving until MIDI 2.0 was introduced in 2020. Over the years, MIDI protocol changed to support the new USB and Firewire connections, including support for Bluetooth and WiFi. See Figure 6-2.

Note MIDI 2.0—developed by a reduced group of companies and developers—has not been conceived to replace the MIDI 1.0 specifications. In a few words, all the MIDI 1.0 devices will not become obsolete in the future. On the contrary, MIDI 1.0 will coexist perfectly with the 2.0 specifications, which aim to add more features to the MIDI 1.0 standard.

According to Dave Smith's vision, the incredible potential of the MIDI protocol can be realized only if all the producers could adopt it. For this reason, it was released as one of the first open source technologies. See Figure 6-3.

Figure 6-3. *This echo effect is the remake of the legendary Boss RE-201 Space Echo, originally using a tape loop, launched by Roland in 1974. This model—able to simulate the same features of its ancestor—includes MIDI-In and MIDI-Out connectors. All the settings are manually accessible by the rotary controls and through MIDI commands*

6.2. The MIDI Protocol Essentials

Tip A full, comprehensive, and detailed specification of the MIDI protocol is available on the `midi.org` site. It is the official nonprofit trade organization of the MIDI standard, connecting companies who develop MIDI products and new MIDI specifications with everyone worldwide creating music and art with MIDI. The MIDI Association also publishes updates about the evolution of the protocol and its adaptions to the most recent communication technologies.

In this chapter, I provide some essential information for understanding how the protocol works, for proper use with electronic instruments and custom-developed devices. Note that regardless of almost half a century of evolution, the protocol's GM (General MIDI) specifications are still valid.

6.2.1. MIDI Communication

MIDI communication is based on serial communication: one wire to send, one to receive, and a ground wire. Some devices, like computers and synthesizers, support both directions—one MIDI-In plug and one MIDI-Out plug. These devices can be connected in a chain—for this reason, MIDI can also be considered a bus. This configuration allows multiple instruments to be synchronized with the same rhythm (MIDI clock signal).

For example, you can chain a MIDI keyboard connected to a drum machine and a sampler so all the devices play at the same rhythm. To achieve this result, you need a device acting as a master and all the others as slaves. The slave devices adapt their speed to the master clock to work in sync. See Figure 6-4.

Figure 6-4. *This Roland AIRA T-8 is a drum machine that includes the sounds of the iconic TR-606, TR-808, and TR-909 and the bass line sounds of the TB-303. It can be easily connected to a MIDI keyboard and used as the master clock in a chain with other MIDI devices*

In MIDI communication, you can functionally identify *commands* and *notes*. For example, a generic MIDI keyboard—which does not play any sound—is a simple interface. Its MIDI-Out cable can be plugged into the MIDI-In of a synthesizer or a drum machine to play the respective electronic instruments.

Two types of commands can be sent through the MIDI protocol: *program change* and *control change* commands. Of course, the devices should not necessarily support all the MIDI features. It depends on the specific characteristics of these devices.

6.2.2. The Protocol Format

MIDI information is built from one byte (eight bits). This means that MIDI information has a value between 0 and 255. Consider what that means in hexadecimal format:

MIDI information: A value between 0x00 - 0xFF

If you divide the byte into the MSB (Most Significant Byte) and LSB (Less Significant Byte), you have the following two hexadecimal value ranges:

LSB: 0x00 - 0x7F

MSB: 0x80 - 0xFF

The LSB part corresponds to the decimal values 0-127, while the MSB part corresponds to the decimal values 128-255.

Following this subdivision, every MIDI byte can be data or a command: this depends on the MSB part of the byte. If the value is less than 0x80, it is a data byte; if it is greater than 0x7F, it is a command byte.

In addition, the MIDI protocol commands can address 16 (0x0F) different MIDI channels.

6.2.3. General MIDI (GM)

To understand how the MIDI protocol manages all the aspects of the musical characteristics (notes, effects, beats, volume, etc.), you need to understand some practical information to see how it works. The best way is to look at the General MIDI 1 requisites (GM or GM1). See Figure 6-5.

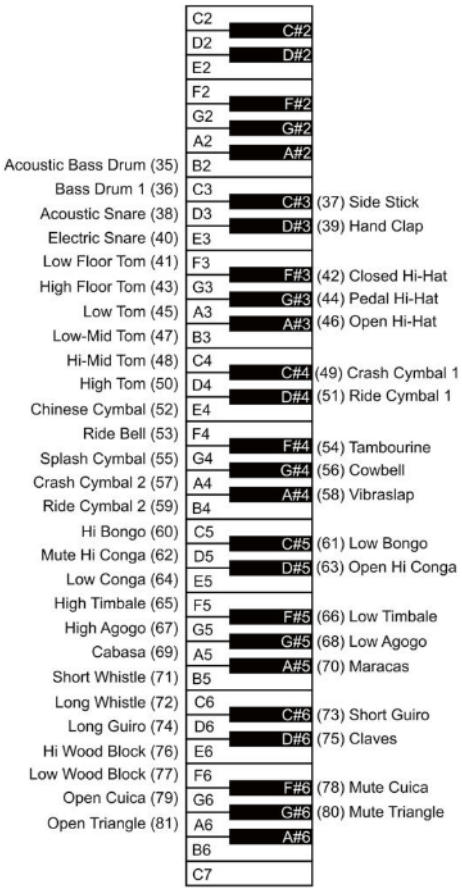

Figure 6-5. *The GM1 association of notes (numbers in parentheses) for the rhythmic instruments (Source: Wikipedia, lic. CC). This project is building a drum machine, so it's interested in the association of the notes to the drum instruments*

While MIDI 1.0 specifications define the physical protocol, first published in 1991, GM1 defines a standard specification for the electronic musical instruments supporting MIDI communication.

These specifications define the characteristics of any instrument adopting GM through a series of requirements that fit the different kinds of electronic musical instruments. See Figure 6-6.

Criterion	Requirement
Piano keys	84 piano keys.
Voices	Allow 24 voices to be available simultaneously for both melodic and percussive sounds (alternatively, allow 16 melodic and 8 percussive voices). All voices respond to note velocity.
Channels	Support all 16 channels simultaneously, each assignable to different instruments. Channel 10 is reserved for percussion. Support polyphony (multiple simultaneous notes) on each channel.
Instruments	Support a minimum of 128 MIDI Program Numbers (conforming to the GM 1 Instrument Patch Map) and 47 percussion sounds (conforming to the GM 1 Percussion Key Map).
Channel messages	Support for controller number 1, 7, 10, 11, 64, 100, 101, 121 and 123; support for channel pressure and pitch bend controllers.
Other messages	Respond to the data entry controller and the RPNs for fine and coarse tuning and pitch bend range, as well as all General MIDI Level 1 System Messages.

Figure 6-6. *The GM1 requirements for the electronic musical instruments (Source: Wikipedia)*

Of course, GM also defines the characteristics of the PC (Program Change) and CC (Control Change) MIDI commands. An exhaustive definition of the GM1 can be found in the Wikipedia article, "General MIDI" (December 15, 2023), at https://en.wikipedia.org/wiki/General_MIDI.

6.3. Arduino and the MIDI Library

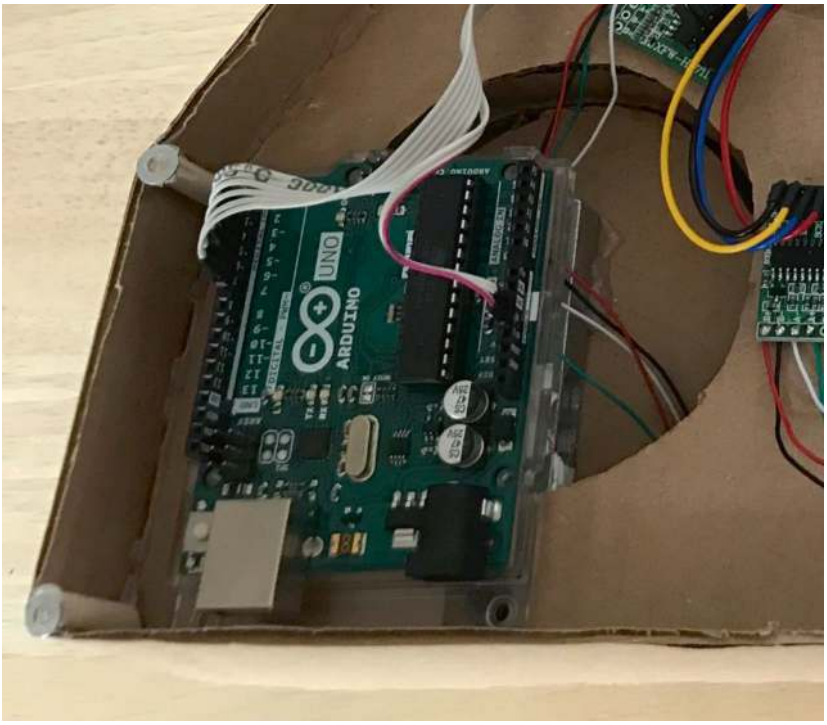

Figure 6-7. *Arduino UNO R3 is the core of this project; the microcontroller, with the MIDI library, in this project exposes itself as an electronic musical instrument, sending the notes corresponding to the configured drum instruments*

The easiest way to understand how the MIDI is used on the Arduino board is by analyzing the part of the Arduino sketch that manages the protocol. It is important to realize that this is a MIDI configuration where the Arduino board acts as an electronic instrument. See Figure 6-7 and Listing 6-1.

Listing 6-1. Implementation of the MIDI Library in the Arduino Sketch

```
#include <MIDI.h>
[...]
```

According to the General MIDI specifications, the MIDI channel of the drum machine will use the MIDI channel 10, the default percussion channel:

```
// Fixed MIDI Channel (the default percussions channel 10)
#define MIDI_CHANNEL 10
```

By default, the MIDI velocity value for the percussion notes is set to the full range (0-127), but custom limits can be hardcoded to a different range if needed. Be aware that the term *velocity* represents the note intensity. In this case, it is proportional to the pressure applied to the drum pads.

```
#define MIN_MIDI_VELOCITY 0
#define MAX_MIDI_VELOCITY 127
```

You need to create an instance of the library attached to a serial port; in the Arduino UNO, the serial port is interfaced to the USB-to-serial port, so the MIDI USB connection is automatically supported. The MIDI instance is initialized by the `begin()` method, as with the serial interface.

```
MIDI_CREATE_DEFAULT_INSTANCE();
MIDI.begin();
```

Note that `calcMIDIVelocity()` is one of the MIDI process functions; it sets the MIDI velocity (range 0-127), mapping the intensity pressure detected on a pad:

```
int calcMIDIVelocity(float sensor) {
  int velocity;

 velocity = map(padRead, -MIN_SENSOR_RANGE, MAX_SENSOR_RANGE,
 MIN_MIDI_VELOCITY, MAX_MIDI_VELOCITY);
```

```
 if( (velocity > MAX_MIDI_VELOCITY) || (velocity < MIN_MIDI_
 VELOCITY) ) {
   return 0;
 } else {
   return velocity;
 }
}
```

6.3.1. The MIDI Library Header

The MIDI Library header interfaces classes and sources for low-level hardware MIDI communication (serial and USB). The library's public methods reflect the General MIDI specifications through its header.

The MIDI Output methods, as defined in the `MIDI.h` header file, cover almost all the GM1 MIDI commands, both PC (Program Change) and CC (Control Change). See Listing 6-2.

Listing 6-2. The MIDI Library header

```
public:
    inline void sendNoteOn(DataByte inNoteNumber,
                           DataByte inVelocity,
                           Channel inChannel);

    inline void sendNoteOff(DataByte inNoteNumber,
                            DataByte inVelocity,
                            Channel inChannel);

    inline void sendProgramChange(DataByte inProgramNumber,
                                  Channel inChannel);
```

```
inline void sendControlChange(DataByte inControlNumber,
                              DataByte inControlValue,
                              Channel inChannel);

inline void sendPitchBend(int inPitchValue,    Channel
inChannel);
inline void sendPitchBend(double inPitchValue, Channel
inChannel);

inline void sendPolyPressure(DataByte inNoteNumber,
                             DataByte inPressure,
                             Channel inChannel) __
                             attribute__ ((deprecated));

inline void sendAfterTouch(DataByte inPressure,
                           Channel inChannel);
inline void sendAfterTouch(DataByte inNoteNumber,
                           DataByte inPressure,
                           Channel inChannel);

inline void sendSysEx(unsigned inLength,
                      const byte* inArray,
                      bool inArrayContainsBoundaries
                      = false);

inline void sendTimeCodeQuarterFrame(DataByte inTypeNibble,
                                     DataByte
                                     inValuesNibble);
inline void sendTimeCodeQuarterFrame(DataByte inData);

inline void sendSongPosition(unsigned inBeats);
inline void sendSongSelect(DataByte inSongNumber);
inline void sendTuneRequest();
```

```
    inline void sendCommon(MidiType inType, unsigned = 0);

    inline void sendClock()         { sendRealTime(Clock); };
    inline void sendStart()         { sendRealTime(Start); };
    inline void sendStop()          { sendRealTime(Stop);  };
    inline void sendTick()          { sendRealTime(Tick);  };
    inline void sendContinue()      { sendRealTime(Continue);  };
    inline void sendActiveSensing() { sendRealTime
                                    (ActiveSensing);  };
    inline void sendSystemReset()   { sendRealTime
                                    (SystemReset);  };

    inline void sendRealTime(MidiType inType);

    inline void beginRpn(unsigned inNumber,
                         Channel inChannel);
    inline void sendRpnValue(unsigned inValue,
                             Channel inChannel);
    inline void sendRpnValue(byte inMsb,
                             byte inLsb,
                             Channel inChannel);
    inline void sendRpnIncrement(byte inAmount,
                                 Channel inChannel);
    inline void sendRpnDecrement(byte inAmount,
                                 Channel inChannel);
    inline void endRpn(Channel inChannel);

    inline void beginNrpn(unsigned inNumber,
                          Channel inChannel);
    inline void sendNrpnValue(unsigned inValue,
                              Channel inChannel);
```

```
inline void sendNrpnValue(byte inMsb,
                          byte inLsb,
                          Channel inChannel);
inline void sendNrpnIncrement(byte inAmount,
                              Channel inChannel);
inline void sendNrpnDecrement(byte inAmount,
                              Channel inChannel);
inline void endNrpn(Channel inChannel);

inline void send(const MidiMessage&);
```

Of course, the library is also capable of reading and parsing MIDI messages to manage bidirectional communication.

CHAPTER 7

Crafting the Cardboard Drum

Figure 7-1. *The five-pad drum machine is fully assembled. It's not just a proof of concept but a programmable and flexible, Arduino-based, MIDI electronic musical instrument*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_7

After learning about some basic theoretical notes on the MIDI protocol and the Arduino MIDI library in the previous chapter, it's time to see how it was possible to make a fully working drum machine using an alternative method to the expensive commercial pad sensors.

7.1. Cheap and Recycled Stuff

> *A piezo electric sensor is a device that uses the piezo electric effect to measure changes in pressure, acceleration, temperature, strain, or force by converting them to an electrical charge. The prefix piezo is Greek for "press" or "squeeze." (Source: Wikipedia)*

Building a digital drum pad using a microcontroller and sensors available in commercial drum pads might not seem too complex. Still, you need a more powerful processor than an Arduino UNO R3.

The problem lies in the kind of the components. In this case, you have two choices: buy a prebuilt sensor pad or use a set of piezo electric sensors.

Prebuilt drum pads are expensive and normally include several piezo electric sensors that capture the pressure and vibration variations depending on the percussion applied to the pad. This factor increases the number of sensors that will trigger the signal, and multiple sensors of the same pad should be integrated to produce the correct MIDI response.

Of course, you could build the pads yourself—as I did in this project—but the problem of integrating multiple sensors remains. The challenging problem was finding an alternative technology to produce a similar result, with only a single sensor for every pad.

7.1.1. Adopting an Alternative Technology

Requirements

First of all, I wrote down a list of essential requirements to figure out what the final result should be and how it should behave:

- Digital drum pad sensors are circularly shaped and react to various pressures within a predefined range.
- Different kinds of solicitations should be converted into different MIDI velocity values for different notes.
- The goal is to make a MIDI drum pad, so the sound is generated by the connected synthesizer. The more the solicitation of the pads produces the correct response, the more the produced sound will be shaped correctly.
- The force applied to the surface of the pads should not exceed 5-6 kg; this is a limit related to building the hardware.

Using Load Cells

The solution to achieving a high sensitivity is by using a single sensor, which is oriented to adopt linear load cells.

Based on a principle different from that of the piezo electric sensors, the *load cells* are metal components—usually aluminum and available in many shapes—that change the resistance when a force is applied. The ideal shape for this project is a linear load cell fixed to the circular pad. See Figure 7-2.

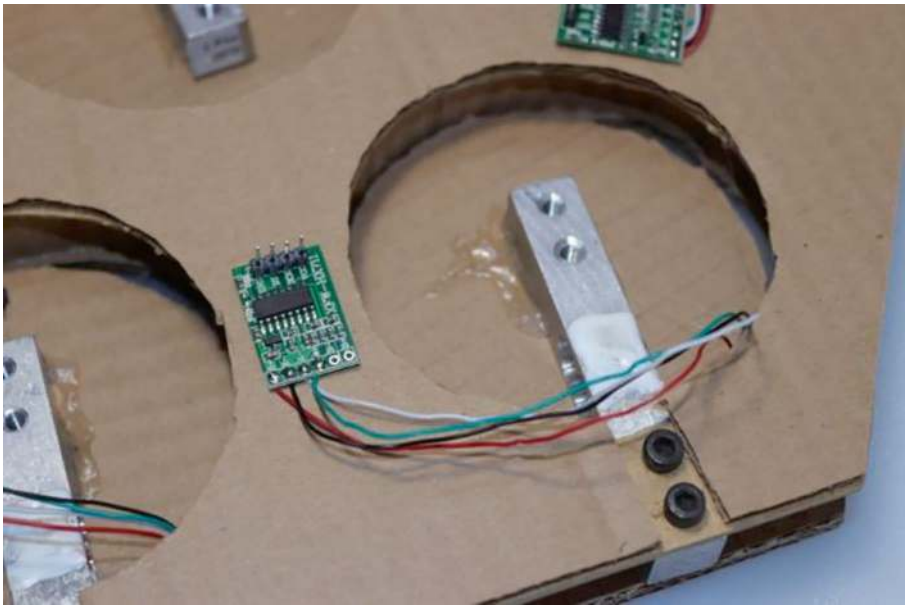

Figure 7-2. *Bottom view showing how the load cell is connected to the circular pad. The cell is screwed to the structure of the device and glued to the bottom side of the pad. Any vibration of the pad is transmitted to the load cell*

This technology also makes it is easier to manage with the Arduino. Every load cell is connected to an HX711 integrated circuit that, when powered, converts the variation of resistance of the load cell to a voltage variation. The Arduino GPIO analog inputs make this variation easy to read without needing external electronic comparators.

Note The drum pad prototype I made is designed to be played with hands. In this case, load cells supporting up to 5kg of solicitations were sufficient. The same technology will also work with pads designed to be played with drumsticks using more capable load cells.

7.2. Creating the Structure

Of course, I ran some preliminary tests to verify the performance and response of the load cells before starting the structure design. Still, a real test was possible only when the prototype was finished, as I could not find similar applications with these sensors. I designed the parts in the most accurate way possible.

7.2.1. Strong Parts

Two parts of the building are critical and subject to relatively strong mechanic solicitations: the base of the device and the pads. See Figure 7-3.

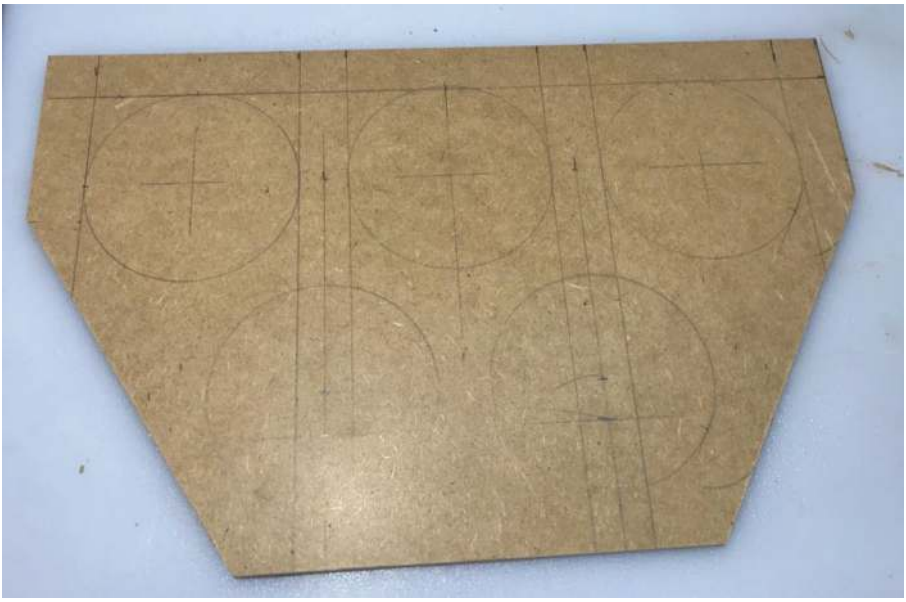

Figure 7-3. *A piece of recycled MDF is sufficiently robust to work as the base of the MIDI drum pad*

I obtained these two parts from a single piece of 3mm MDF to create the upper support where the load cells are screwed; the rigid part of the drum pads was obtained by making round holes in the MDF base.

Then, I increased the thickness of the structure by adding a couple of layers of corrugated cardboard; it is sufficiently robust and durable. I used cardboard recycled from strong shipment boxes. Making this first prototype, I focused on the quality of the result and the stability of the structure.

It is also possible to take the structure to the next level by painting it with a waterproof and protective paint.

The Pads

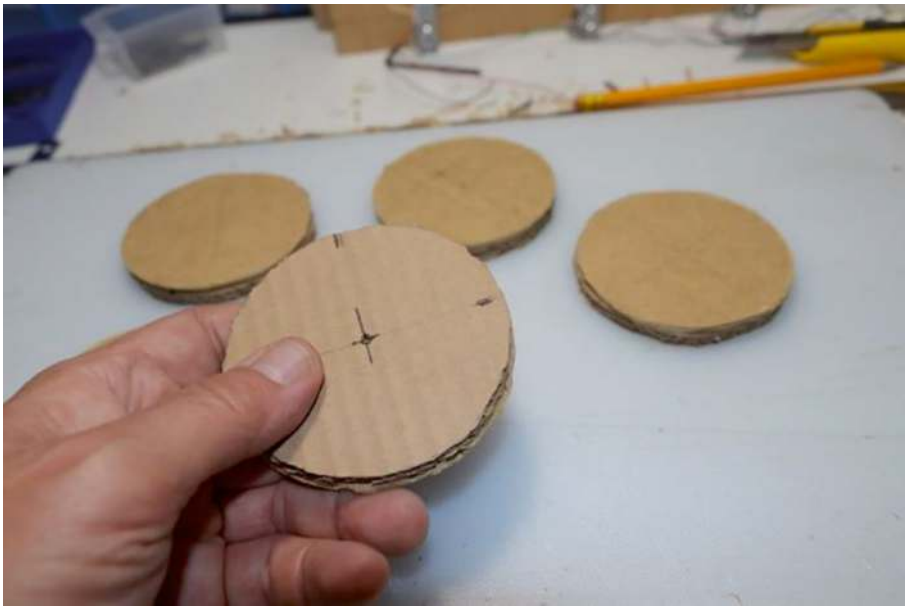

Figure 7-4. *The construction of the multi-layer circular pads*

The drum pads (see Figure 7-4) were built with a three-layer structure consisting of 3mm MDF for the base attached to the load cells, glued to which is a 3mm corrugated cardboard disc to soften the percussion and a surface finish on the top side (the striking area) made with a 1mm thick black polyurethane sheet.

The weak point of the pads is the cardboard layer; for this reason, they can be played by hand, but the surface can't support the pressure of the drumstick. Using alternative materials, such as compact foam or silicon layers, can give the pads a different response to the mechanic's solicitations. See Figures 7-5 and 7-6.

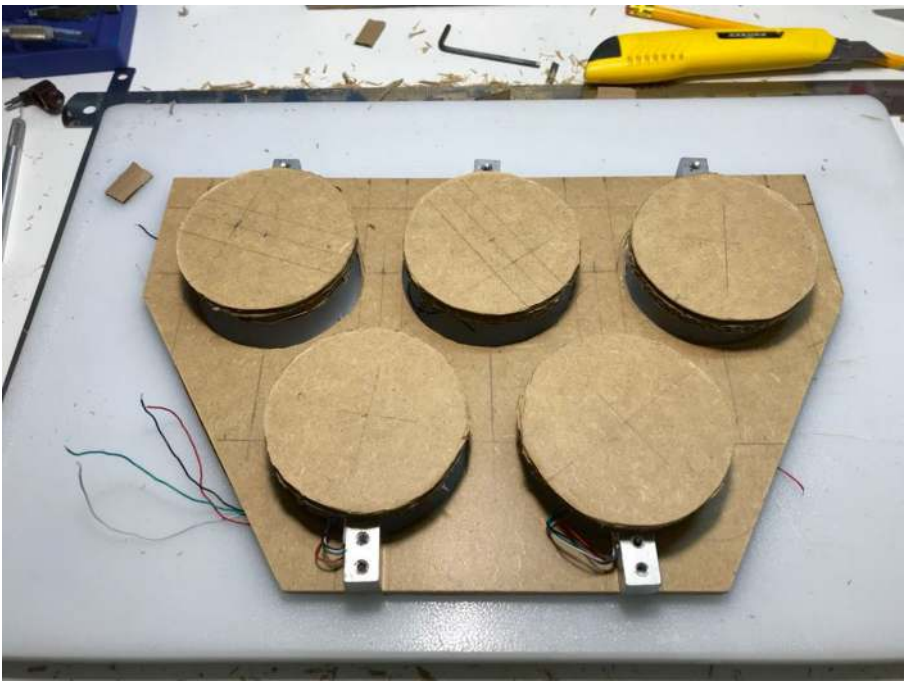

Figure 7-5. *The five pads glued over their respective linear load cells during the assembly phase*

Note The pad surface exposed to the touch is the hard MDF component, glued over the corrugated cardboard layer, then glued over the load cell. The corrugated cardboard—or any other soft material—contributes to smoothening the impact over the load cell. By experimenting with other kinds of materials, it is possible to obtain different response effects.

Figure 7-6. *Detail of the pads with the top layer composed of a 1mm thick black polyurethane sheet*

7.2.2. Fixing the Load Cells

One of the roles of the MDF base is to keep the five load cells firmly screwed in place. This is the only part of the drum machine that uses screws, as the pads are hot-glued on the opposite side of the cells. See Figure 7-7.

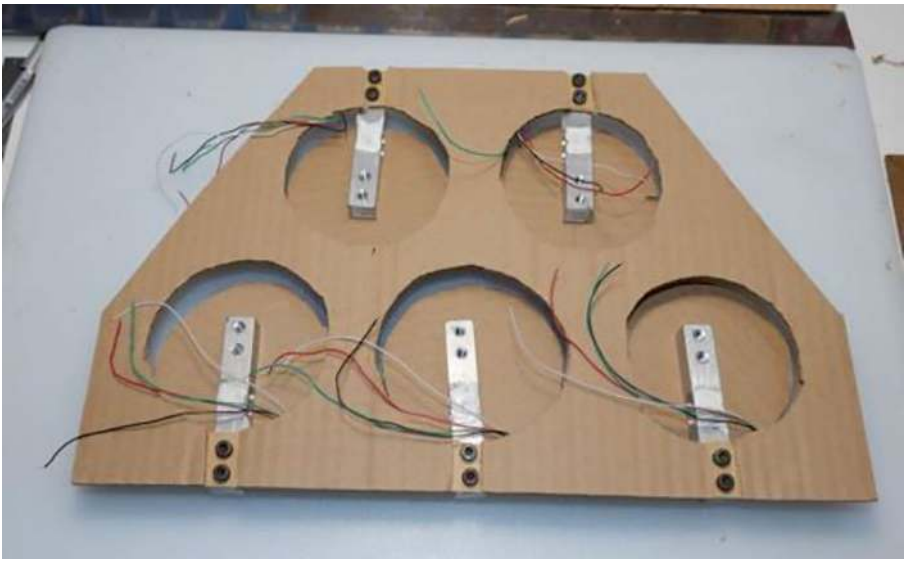

Figure 7-7. *Bottom view of the load cells screwed to the base and hot-glued to the pads*

The bottom base is also where I added the five HX711 amplifier circuits soldered to the cells. The HX711 circuit is a small PCB that converts the very low resistance variations of the load cells, when exposed to a mechanical solicitation, to a voltage variation read by the analog pins of the Arduino GPIO.

Wiring the Circuit

To understand how to wire all the signals to the Arduino board, it is necessary to see how the reading cycle of the HX711 amplifier circuit works:

- The five HX711 circuits are powered by the GND and 5 Vcc pins of the Arduino.
- The five analog signals (the load cells measure) are sent to five of the six analog inputs of the Arduino GPIO.
- The five clock signals are connected to a single PWM output signal of the Arduino GPIO.

The logic workflow consists of sending a clock signal to the HX711 and reading the corresponding analog value on the amplifier's corresponding pin. See Figure 7-8.

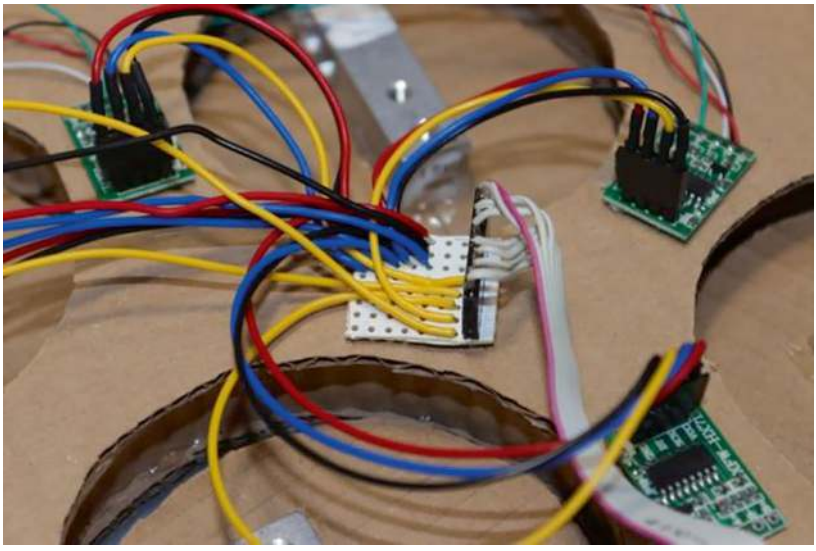

Figure 7-8. *Detail of the HX711 sensor wires collected on a breadboard PCB, which is connected to the Arduino GPIO through a flat cable*

Making the Case

All the case parts were made of sturdy corrugated cardboard, recycled from another shipment box.

A border around the structure completes the sides of the drum machine, encasing the device and leaving space at the bottom to host the HX711 board, circuitry and wiring, and the Arduino board. The case sides were hot-glued to the base; the case's shape and the cardboard's orientation give the necessary robustness. See Figure 7-9.

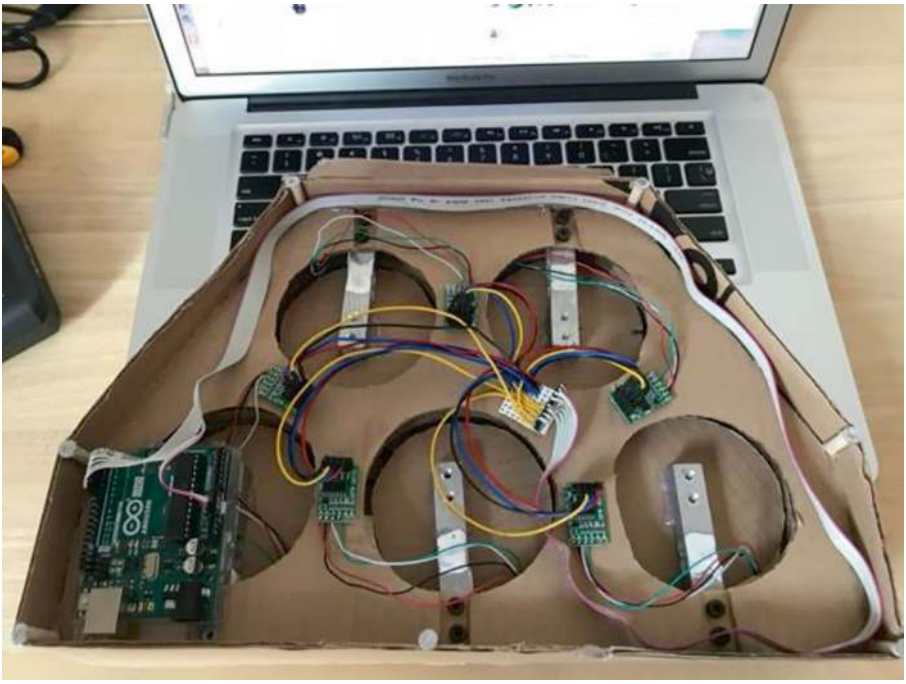

Figure 7-9. *The bottom side of the drum machine with the case glued to the base. The bottom box closure is another cardboard piece to cover the wiring and the Arduino board*

The top side of the box, made of cardboard as well, includes an inner circular spacer around every pad hole. This solution contributes to the robustness of the case on the percussion side. See Figure 7-10.

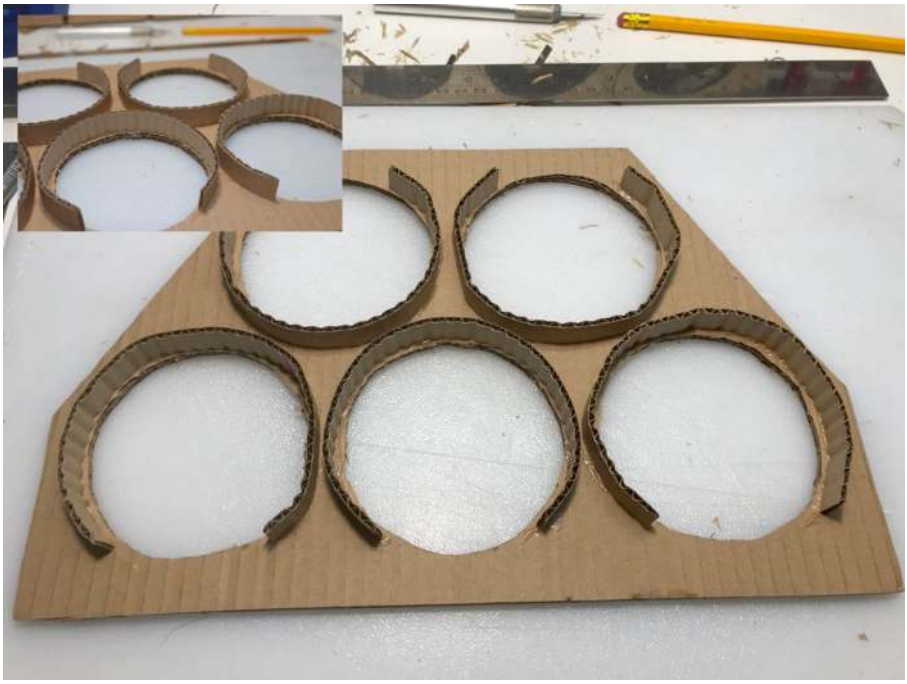

Figure 7-10. *Bottom view of the top cover of the drum machine. Around every hole of the pads, there is a cardboard spacer hot-glued to the cover*

7.3. The Sensors Software

The load sensors interfaced to the HX711 circuit are very popular in the makers' world; many Arduino projects use these sensors, thanks to the availability of the Arduino HX711 library.

This library is designed to use a single sensor on the board. However, the plan is to simultaneously use five load cells associated with every pad. To achieve this result, I used the HX711 library—not available through the Arduino IDE Library Manager.

The HX711 customized library used here, also available in the project software repository, is the open source version from GitHub (https://github.com/bogde/HX711) by Bogde.

The implementation of the library for reading in blocking and non-blocking modes made it possible to create polling over an array of five sensors using a single clock signal.

After including the HX711 library header (see Listing 7-1), the pad signals were associated with the first five analog inputs of the Arduino GPIO and a single output clock signal pin.

Listing 7-1. The HX711 Library Header

```
#include <HX711.h>

[...]

// Connection of pads pins
#define PAD_CLK 2     // Clock PIN
#define PAD1 3         // Pad 1 PIN
#define PAD2 4         // Pad 2 PIN
#define PAD3 5         // Pad 3 PIN
#define PAD4 6         // Pad 4 PIN
#define PAD5 7         // Pad 5 PIN
```

The program creates five instances of the library, initialized for every pad connected to the corresponding Arduino analog input. All the instances share the same clock signal pin.

```
// Define the weight sensor instance for every pad
HX711 pad0(PAD1, PAD_CLK);
HX711 pad1(PAD2, PAD_CLK);
```

```
HX711 pad2(PAD3, PAD_CLK);
HX711 pad3(PAD4, PAD_CLK);
HX711 pad4(PAD5, PAD_CLK);

// Create an array of pointers to the pad instances
HX711 *pads[NUM_PADS];
// Predefined notes mapped to the pads
int padsMap[NUM_PADS];

// Last value read from pad weight sensor
float padRead;
```

The `setup()` function initializes the library instances and starts the five HX711 sensors. Note that the `padsMap[]` array assigns the MIDI note number to every pad. This is the note—and its relative velocity—sent to the MIDI channel. It is possible to change these values to associate every pad with a different instrument, according to the General MIDI standard.

```
// Initialization
void setup() {

  // Create the pad vector to manage the weight sensors in loops
  pads[0] = &pad0;
  pads[1] = &pad1;
  pads[2] = &pad2;
  pads[3] = &pad3;
  pads[4] = &pad4;

  // Set up the pitch notes associated with every pad
  padsMap[0] = 38;
  padsMap[1] = 39;
  padsMap[2] = 40;
  padsMap[3] = 41;
  padsMap[4] = 42;
}
```

The main `loop()` functions applies relatively complex logic; in fact, to achieve the needed responsivity, the polling process must be as fast as possible. As a pad reading is detected, the applied mechanical pressure—read as an integer at the corresponding analog input of the Arduino GPIO—is converted to a MIDI velocity value. Then the MIDI note is sent to the connected MIDI instrument through the Arduino USB.

```
void loop() {
  int j;
  int mappedPad;

  // Loop on every pad to see if one has been pressed
  for(j = 0; j < NUM_PADS; j++) {
    padRead = pads[j]->get_units(NUM_READINGS) * -1;
    mappedPad = calcMIDIVelocity(padRead);

    if(mappedPad > 4) {
      MIDI.sendNoteOn(padsMap[j], 64, MIDI_CHANNEL); // Send a
      Note (pitch, velo, channel)
      MIDI.sendNoteOff(padsMap[j], 0, MIDI_CHANNEL);
    }
  }
}
```

The `calcMIDIVelocity()` function maps the sensors valued to the `min` and `max` note velocity range to convert them to MIDI velocity values.

```
int calcMIDIVelocity(float sensor) {
  int velocity;

 velocity = map(padRead, -MIN_SENSOR_RANGE, MAX_SENSOR_RANGE,
 MIN_MIDI_VELOCITY, MAX_MIDI_VELOCITY);
```

```
 if( (velocity > MAX_MIDI_VELOCITY) || (velocity < MIN_MIDI_
 VELOCITY) ) {
   return 0;
 } else {
   return velocity;
 }
}
```

CHAPTER 8

A Sound Sampler with Raspberry Pi

Figure 8-1. *The development bench of the Raspberry Pi sound sampler and player. Note that the Radio Magic to the left of the image is used as a noise source, as described in Chapter 17*

In music, a sequencer is a device that can control the order and timing of notes, beats, and other sound elements. It allows the musician to compose and arrange music by programming the sequence of events.

As with all musical devices, sequencers can also work with MIDI data, audio samples, or other forms of digital information. There are two primary types of sequencers:

- **Step sequencers:** Allow the introduction of musical patterns or loops step-by-step.
- **Linear sequencers:** Record and arrange music along a timeline, similar to a multitrack recording.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_8

In music, a *sampler* is a device or software for recording, processing, and playing audio samples.

The samples can be any sound, such as musical notes, spoken words, environmental noises, or recorded audio. With samplers, the musician can modify sound-altering parameters like pitch, duration, and timbre, and then trigger them in sequences and loops.

In this project, I tried to make a customized sampler and sequencer based on a Raspberry Pi, with some limitations and features that are usually not present in this kind of device. See Figure 8-1.

8.1. Project Requirements and General Approach

Considering the processing power of the Raspberry Pi, most of the project has been developed with software. Of course, some peripherals must be added to the system.

The Raspberry Pi 4B model is considerably fast. I have done some preliminary tests, but using a previous model of this embedded Linux platform did not perform as needed.

8.1.1. External Hardware

To make the system compact and usable in a real music production set, I opted to work with the LCD touchscreen. Starting with the 4B model, the Raspberry Pi has implemented a good digital audio output plug; the first versions of the boards used a PWM audio generator, whose quality was not sufficient for good sound quality.

Despite the good audio reproduction quality, the Raspberry Pi board does not include an ADC (Analog to Digital) feature. This means that audio sampling is not natively possible.

Figure 8-2. *One of the USB audio boards I tested on the Raspberry Pi when evaluating the project's feasibility*

To make this possible, I tried a series of different USB audio boards and tested their responses for audio sampling and streaming; the reference sampling frequency should be at least 48K. See Figure 8-2.

To test the reliability of the audio boards, I used a frequency generator to send several audible-range signals (20-20.000Hz) to the board and verified the sampled quality when running on the Raspberry Pi. See Figure 8-3.

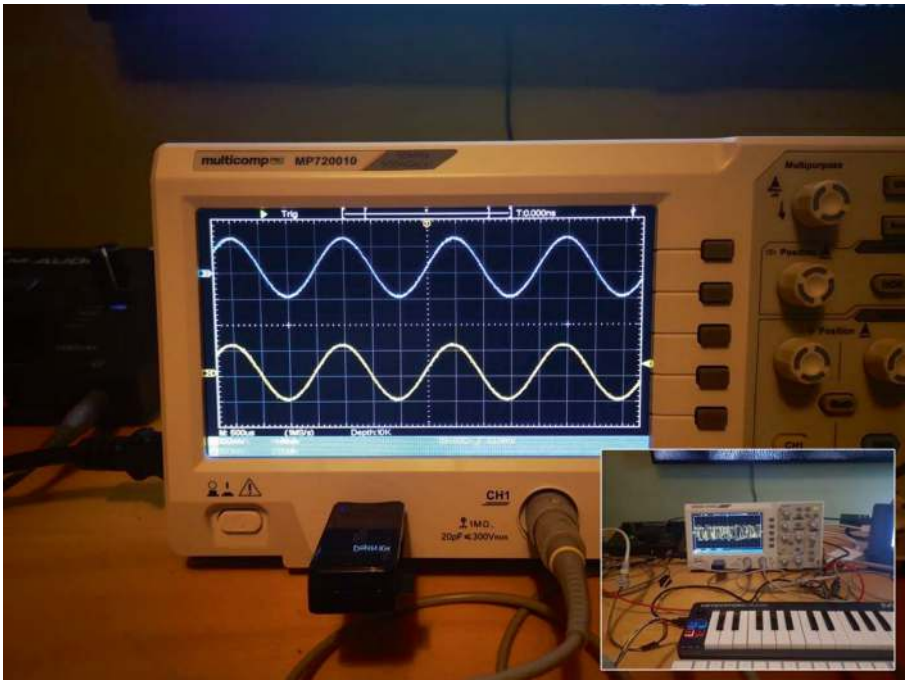

Figure 8-3. *Using a frequency generator, I tested the response of the audio board when connected to the Raspberry Pi, as well as with recorded samples, with a dual-channel oscilloscope*

The audio samples should be as clean as possible. I excluded the MP3 (compressed) format to work exclusively with waveform files (WAV) to achieve the required quality.

Of course, the WAV files are considerably bigger than the widely diffused MP3 ones, so I included a 180GB SSD disk for storage. The disk is connected to the Raspberry Pi through one of the two USB 3 ports available on the board. See Figure 8-4.

Figure 8-4. *The small M-AUDIO MIDI USB keyboard connected to the Raspberry Pi to play music*

8.1.2. Features List

- Sample sounds at 48K of variable length, without duration limits.
- Immediately associate the samples with a specific sound note.
- Extend the sampled note to the entire scale if the other positions are not yet occupied by a sample.
- Play samples were added manually without sampling.

- Record/stop sampling from the interface.
- Accept an external MIDI keyboard to play the samples.

Note As the MIDI protocol is based on the serial port, it is not difficult to implement a standard MIDI-Out port to the Raspberry Pi. However, I decided not to implement this feature because of the wide diffusion of the USB-MIDI interface.

8.2. The Sampling Session

Acquiring audio WAV samples is not only a question of hardware; the Raspberry Pi needs to be equipped with proper software.

8.2.1. Connecting the MIDI Keyboard and Audio Card

As mentioned, I decided to constrain the MIDI features to the USB MIDI connection. The audio card is connected through the USB in the same way.

From the Raspberry Pi terminal, the following command lists all the USB peripherals connected to the computer:

```
cat /proc/asound/cards
```

In this list, for example, device 1 is the MIDI USB keyboard and device 2 is the audio board:

```
0 [ALSA           ]: bcm2835_alsa - bcm2835 ALSA
                     bcm2835 ALSA
```

```
1 [K32            ]: USB-Audio - Keystation Mini 32
                     Keystation Mini 32 Keystation Mini 32 at
                     usb-3f980000.usb-1.5, full speed
2 [Device         ]: USB-Audio - USB Advanced Audio Device
                     C-Media Electronics Inc. USB Advanced
                     Audio Device at usb-3f980000.usb-1.3,
                     full speed
```

In the current software version, the USB devices are hardcoded in the program; it is not complex, and a significant improvement is needed to make these parametric for better flexibility.

8.2.2. The Sampler Box

Sampler Box (https://www.samplerbox.org/) is a project developed by Joseph Basquin. Released as open source and available on GitHub (this is a fork on my repository: https://github.com/alicemirror/SamplerBox), Sampler Box is essentially a MIDI player of preloaded samples based on the Raspberry Pi, but with some simple modifications to acquire audio samples, store them, and make them immediately playable.

The system is very fast and makes MIDI signals very responsive. This depends on the way the sampling and MIDI events are processed. C software is used to do this; luckily, it was possible to use this component in the project without changing it.

The integration between the Linux C source and the Python application was possible using *Cython*, which can compile a source and create hooks to use the compiled object as a library imported from the Python sources.

The modified and integrated Sampler Box is the audio core for sampling and MIDI playing. After solving these two project bottlenecks, I proceeded with the full design and GUI implementation.

8.3. Project GUI Design

Figures 8-5 and 8-6 show the design of the touchbutton grid and the buttons, respectively.

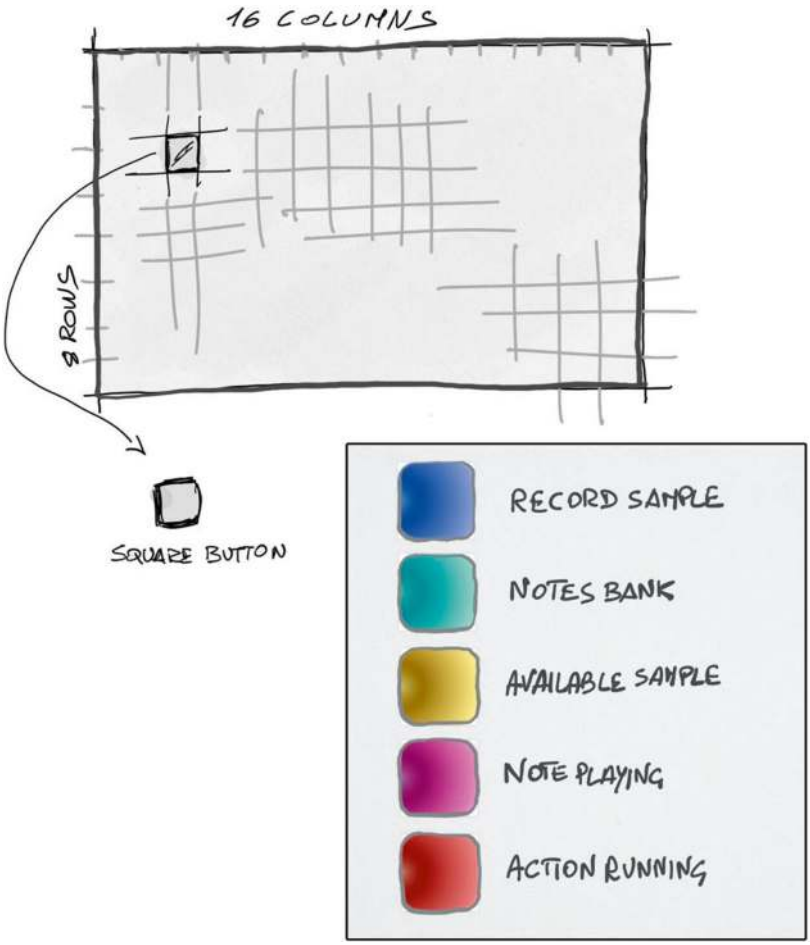

Figure 8-5. *The screen organization consists of a touchbutton grid. I was inspired by the Novation Launchpad Mini MK3 (https://novationmusic.com/products/launchpad-mini-mk3), designing the buttons to fill the entire screen and adopting a color-coding scheme according to the state and feature of the button*

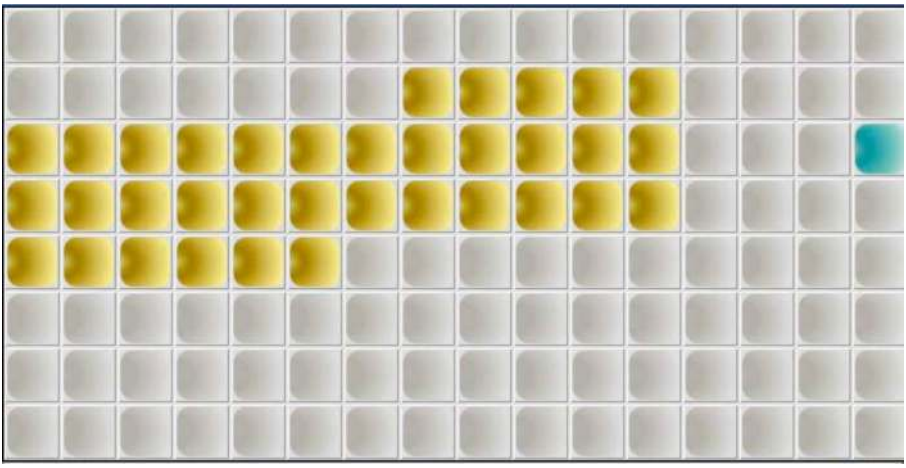

Figure 8-6. *Screen capture showing how squared buttons can fill the screen. Every state of the button will change color according to the associated function at a certain moment (sampling, playing, empty, etc.)*

As a pianist knows the position of the notes on the keyboard to play chords, it is not difficult to learn the color codes and note positions to orient on the touchscreen with some practice.

After running some practical design tests, I calibrated the base size of the buttons to fill the screen on a grid of 16 columns by 8 rows. This subdivision is perfect, as it covers the entire MIDI set of 128 values (0-127).

Every row's leftmost 12 buttons cover a full octave, while the four rightmost buttons are dedicated to four commands applied to the same row. The project's current version assumes that the first left button of every row corresponds to the C key (Do). This is a decision I made because I am used to playing C-tuned instruments (guitar, harmonica, and key), but it is possible to shift the first note assignment to a different tonality and change the entire setup. See Figure 8-7.

Practicing with this system—very similar to many other MIDI controllers—makes the user interface agile and flexible.

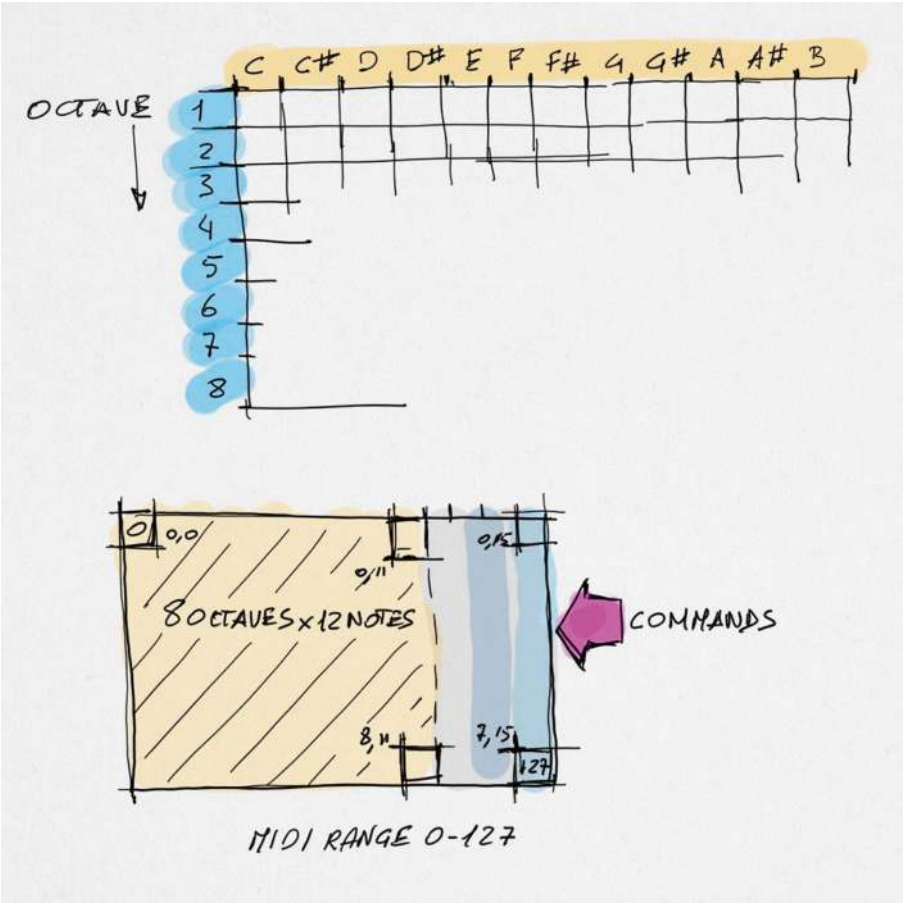

Figure 8-7. *The 128-key assignment scheme on the screen, covering the entire MIDI range from 0 to 127*

8.4. Cython and Other Prerequisites

Before exploring the application sources, it is necessary to install some software on the Raspberry Pi. If you are an expert user of the Linux Raspbian distribution (the Debian version for the Raspberry Pi), you can install the components and applications on your current Raspbian

setting. However, the following steps refer to a fresh, clean installation of the Raspberry Pi desktop version, without the extra software installed like Libre Office to save space on the microSD.

Note To work comfortably, I suggest using a 32- or 64-GB microSD card to install the operating system.

8.4.1. What Is Cython?

Cython is an optimizing static compiler for Python and for the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself.

Cython gives you the combined power of Python and C to let you:

- Write Python code that calls back and forth from and to C or C++ code natively at any point.
- Easily tune readable Python code into plain C performance by adding static type declarations, also in Python syntax.
- Use combined source code level debugging to find bugs in your Python, Cython, and C code.
- Interact efficiently with large data sets, for example, using multi-dimensional NumPy arrays.
- Quickly build your applications within the large, mature, and widely used CPython ecosystem.
- Integrate natively with existing code and data from legacy, low-level or high-performance libraries and applications.

The Cython language is a superset of the Python language that additionally supports calling C functions and declaring C types on variables and class attributes. This allows the compiler to generate very efficient C code from Cython code. The C code is generated once and then compiles with all major C/C++ compilers in CPython 2.6, 2.7 (2.4+ with Cython 0.20.x) as well as 3.5 and all later versions. We regularly run integration tests against all supported CPython versions and their latest in-development branches to make sure that the generated code stays widely compatible and well adapted to each version. PyPy support is work in progress (on both sides) and is considered mostly usable since Cython 0.17. The latest PyPy version is always recommended.

All of this makes Cython the ideal language for wrapping external C libraries, embedding CPython into existing applications, and for fast C modules that speed up the execution of Python code.

(Source: https://cython.org/.)

The Cython Programming Language

Part of the Cython Project, "Cython" is a programming language that makes writing C extensions for Python. Cython gives Python a high-level, object-oriented, functional, and dynamic programming.

Cython's main feature is support for optional static type declarations. The source code is translated into optimized C/C++ code and compiled as Python extension modules.

This allows for very fast program execution and tight integration with external C libraries while maintaining the high programmer productivity for which the Python language is well known.

The full documentation of the Cython language can be found at https://cython.readthedocs.io/en/latest/index.html.

8.4.2. Why You Should Use Cython

The sound sampler developed by Joesph Ernest for the Sampler Box should work fast, acquiring variable-length WAV samples from the external USB audio board.

For this reason, the repository contains the `samplerbox_audio.pyx` file; this is the Cython language source that, when compiled, generates the C/C++ code compiled as a Python extension module.

Note The repository does not include the compiled Python module. You should compile it after the installation of the components. The Cython source will use the specific gcc+ compiler and libraries present by default on the Linux installation.

The Cython source code (see Listing 8-1) generates the Python module. It is interesting to note that the numpy library is imported and will be included in the C/C++ Python module. This code is limited to the bare metal acquisition functions to keep it as fast and small as possible.

Listing 8-1. The Cython Source Code

```
import cython
import numpy
cimport numpy

def mixaudiobuffers(list playingsounds, list rmlist, int
frame_count, numpy.ndarray FADEOUT, int FADEOUTLENGTH, numpy.
ndarray SPEED):
    cdef int i, ii, k, l, N, length, looppos, fadeoutpos
    cdef float speed, newsz, pos, j
    cdef numpy.ndarray b = numpy.zeros(2 * frame_count, numpy.
    float32)      # output buffer
```

```
    cdef float* bb = <float *> (b.data)       # and its pointer
    cdef numpy.ndarray z
    cdef short* zz
    cdef float* fadeout = <float *> (FADEOUT.data)

    for snd in playingsounds:
        pos = snd.pos
        fadeoutpos = snd.fadeoutpos
        looppos = snd.sound.loop
        length = snd.sound.nframes
        speed = SPEED[snd.note - snd.sound.midinote]
        newsz = frame_count * speed
        z = snd.sound.data
        zz = <short *> (z.data)

        N = frame_count

        if ( (pos + frame_count * speed > length - 4) and
        (looppos == -1) ):
            rmlist.append(snd)
            N = <int> ((length - 4 - pos) / speed)

        if (snd.isfadeout):
            if (fadeoutpos > FADEOUTLENGTH):
                rmlist.append(snd)
            ii = 0
            for i in range(N):
                j = pos + ii * speed
                ii += 1
                k = <int> j
                if (k > length - 2):
                    pos = looppos + 1
                    snd.pos = pos
                    ii = 0
```

```
                    j = pos + ii * speed
                    k = <int> j
                bb[2 * i] += (zz[2 * k] + (j - k) * (zz[2 * k +
                2] - zz[2 * k])) * fadeout[fadeoutpos + i]
                # linear interpolation
                bb[2 * i + 1] += (zz[2 * k + 1] + (j - k) *
                (zz[2 * k + 3] - zz[2 * k + 1])) *
                fadeout[fadeoutpos + i]
            snd.fadeoutpos += i

        else:
            ii = 0
            for i in range(N):
                j = pos + ii * speed
                ii += 1
                k = <int> j
                if (k > length - 2):
                    pos = looppos + 1
                    snd.pos = pos
                    ii = 0
                    j = pos + ii * speed
                    k = <int> j
                bb[2 * i] += zz[2 * k] + (j - k) * (zz[2 * k +
                2] - zz[2 * k])          # linear interpolation
                bb[2 * i + 1] += zz[2 * k + 1] + (j - k) *
                (zz[2 * k + 3] - zz[2 * k + 1])

        snd.pos += ii * speed

    return b

def binary24_to_int16(char *data, int length):
    cdef int i
    res = numpy.zeros(length, numpy.int16)
```

```
    b = <char *>((<numpy.ndarray>res).data)
    for i in range(length):
        b[2*i] = data[3*i+1]
        b[2*i+1] = data[3*i+2]
    return res
```

A small `setup.py` program shown in Listing 8-2 is provided in the repository to compile the Cython sources. Based on my personal experience, I added some comment details to avoid issues during this step.

Listing 8-2. The setup.py Program

```
#
# Cytonize the samplerbox audio engine for Python 3
#
# E.M., September 2020
#
# Important note on compiling .pyx files with Python 3 on the
Raspbian
#
# 1. Install Cython with
#       $>pip3 install cython
# 2. Check that the installed version if 2.29 or higher, with
     the command $>cython -V
#       If Cython was already installed with a previous
        version, upgrade Cython for Python 3
#       to the last available version in the repository with
        the command
#       $>pip3 install --upgrade cython
#       Check the version again.
# 3. Add the comment
#       # cython: language_level=3
```

```
#       special setting in the comments on top of the
        pyx source
#       This sets the language level of the compiler globally
        for all the sources
#       If you are compiling multiple sources, and only some of
        these are Python 3, instead,
#       you should use the settings for language level only for
        those sources.
#       For more details read the Cython documentation:
#       https://cython.readthedocs.io/en/latest/src/userguide/
        migrating_to_cy30.html?highlight=python%203
# 4. If you need to import the C libraries of numpy there is a
     workaround that should
#       be applied to make the compilation work in the SPECIFIC
        CASE OF RASPBERRY PI if during the compilation
#       you get an error like
#       libf77blas.so.3: cannot open shared object file: No
        such file or directory
#       at the end of a series of compilation errors that
        involves numpy.
#       As described in the Numpy troubleshooting documentation
#       https://numpy.org/devdocs/user/troubleshooting-
        importerror.html
#       This is due to a missing library that should be
        installed from the Raspbian repositories with
        the command
#       $>sudo apt-get install libatlas-base-dev
#       This issue should be avoided also installing numpy with
        the Raspbian if the version if recent. Try before
#       with Buster distro or next ones, with the command
#       $>pip3 uninstall numpy
#       $>sudo apt install python3-numpy
```

```
#
# WARNING! Sometimes the numpy error does not appear if
compiling with setup.py. In this case, just try the manual
# compilation from the Python console. The mentioned error at
the end of a list of errors will happen when adding
# >>>import numpy
#
# Happy Cython with Python (3)!
#
# cython: language_level=3
from distutils.core import setup
from Cython.Build import cythonize
import numpy

setup(ext_modules = cythonize("samplerbox_audio.pyx"), include_
dirs=[numpy.get_include()])
```

After the Cython compilation, the output consists of two files, the C source, and the so module, which will be imported into the Python source.

Note The `.so` module does not need to be installed like the other Python modules, as the application will use it. It is sufficient that it resides on a path accessible by other sources.

8.4.3. The Graphic Library

The other indispensable component that should be present in the Raspberry Pi installation is the `TkInter` graphic library. It is not natively installed in the Python 3 setup and should be imported (https://docs.python.org/3/library/tk.html).

This GUI for Python is based on the Tcl language and the original Tk library for Linux. I decided to use this relatively unknown component because the design only requires the creation of buttons. The library is compact and small, and it is efficient for use on machines with limited resources.

8.5. The WAV Samples Organization

Every 12-button row (an octave) is considered a *bank*. Every bank can be defined with its own characteristics for playing in its corresponding JSON file. These settings are applied to all the bank notes.

8.5.1. Banks Definition

`volume` can be set as a decimal number between 0.1 and 10. Setting high volumes on some instruments can cause distortion.

The default position of the bank sound is assumed to be the central octave 0, but `transpose` can change to a higher or lower octave.

The `velocity` 64 is the default playing MIDI level. This parameter can be set between 1 and 127.

```
{
  "volume" : 1.0,
  "transpose" : 0,
  "velocity" : 64
}
```

An original feature of this build is that not all the notes in a bank need to be present.

8.5.2. The JSON Parameters File

This is a main JSON configuration file that defines all the parameters accessed by the application at boot.

The full contents of the `gui.json` configuration file are shown here. The two `samples` and `images` values define the path where the samples are stored and the images of the interface.

```
{
  "samples": "/media/pi/EXTERNAL/controlpanel/Samples/",
  "images": "/media/pi/EXTERNAL/controlpanel/images/",
```

The `rows, columns, buttonimages, buttonsize, frame_padX, frame_padY,` and `frame_border` values define the interface element size. These values are calibrated on the Raspberry Pi LCD touchscreen (800x600 resolution). The values can be changed if a different screen resolution is used.

```
  "rows" : 8,
  "columns" : 16,
  "buttonImages" : 8,
  "buttonsize" : 44,
  "frame_padX" : 0,
  "frame_padY" : 0,
  "frame_border" : 0,
  "offButtonImage" : "bNull",
  "imageType" : ".png",
```

The `audioDevice` and `midiDevice` IDs should be defined according to the USB device from the Linux OD.

```
  "audioDevice" : 2,
  "midiDevice" : "Keystation Mini 32 20:0",
  "maxPolyphony" : 80,
```

As mentioned, the `note_names` array defines the notes in the C major scale (Do maj):

```
  "note_names" : [  "c", "c#", "d", "d#", "e", "f", "f#", "g",
  "g#", "a", "a#", "b" ],
```

The `recordSampleRate, recorchChunkSize, recordChannels, recordDuration`, and `fadeoutLenght` functions define the sampling parameters. Note that `recordDuration` is limited by default to five seconds (minimum duration), but the sample continues until the sampling button is pressed.

```
  "recordSampleRate" : 44100,
  "recordChunkSize" : 4096,
  "recordChannels" : 1,
  "recordDuration" : 5,
  "fadeoutLength" : 30000
}
```

8.6. The Application

The application is defined in the `panel.py` source file. The main function initializes the parameters and the structure, creates the user interface on the screen, and then executes an infinite loop interrupt-driven to play, acquire samples, and manage files. See Listing 8-3.

Listing 8-3. The Main Application

```
if __name__ == "__main__":
    '''
    Main application
    '''
    # Initialize the GUI according to the json parameters
    load_GUI_parameters()
```

```
# Load the first samples bank (max 8) by default
load_bank_IDs(0)
# Create the GUI
make_panel()
# Show the first default bank settings
refresh_bank_buttons()

open_sound_device()
preset = 0
LoadSamples()

debugMsg('midi ports ' + str(midi_in[0].ports))
midi_in.append(rtmidi.MidiIn(midi_device.encode()))
midi_in[0].callback = MidiCallback
# midi_in[0].open_port(b'Keystation Mini 32 20:0')
midi_in[0].open_port(midi_device.encode())
previous = midi_in[0].ports
```

Note that the call to the `mainloop()` is a feature of the `TkInter` library:

```
# Start the main loop application
window.mainloop()
```

8.6.1. The Application Functions

This section describes only the most meaningful functions. The full software sources and documentation are available in the chapter repository.

The function in Listing 8-4 is called recursively to acquire the current status of any bank ID.

Listing 8-4. The Load_bank_IDs() Function

```
def load_bank_IDs(bank):
    '''
    Load the notes status array for one of the eight
    octaves of the
    selected bank.
    The note position in the array is set to 1 if the note
    has a file
    in the corresponding bank folder. Samples wav file names
    have the same
    name of the associated note.
    :param bank: The desired bank number
    '''

[. . .]
```

As shown in Listing 8-5, the JSON bank filename is built dynamically based on the ID.

Listing 8-5. The JSON Bank Filename

```
    # Build the Json selected bank file name
    j_name = "bank" + str(bank) + ".json"
    # Set the current bank ID
    current_bank = bank

    debugMsg("Loading " + j_name)

    with open(j_name) as file:
        dictionary = json.load(file)

    # Load the notes flags for every octave.
    # There are max eight octaves and the notes are
```

```
    # listed in the traditional order c, c#, d, d#, e, f, f#,
      a, a#, b
    # Every note that corresponds to a sample in the
      current bank.
    # If the file exists, the note position in the array is set
      to 1 else it is 0
```

The solution of creating an 8x12 matrix (eight arrays) makes it possible to load empty or incomplete octave notes:

```
    octave1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    octave2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    octave3 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    octave4 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    octave5 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    octave6 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    octave7 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    octave8 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

    # Loop all the notes of every bank to see if the
      corresponding file
    # exists. In this case the array position is set to 1
```

The simple file check of the dynamically created note name is used to determine the flag to set (1 or 0) to the octave notes arrays:

```
    for j in range(12):
        # Create the full file name of the note for every bank
        # then if the file exists, set the corresponding flag
        # # in the octave array.
        if(os.path.isfile(get_note_file_name(0, j))):
            debugMsg("j " + str(j) + " bFname " + get_note_
            file_name(0, j) + " file exists")
            octave1[j] = 1
```

```
else:
    octave1[j] = 0

if(os.path.isfile(get_note_file_name(1, j))):
    debugMsg("j " + str(j) + " bFname " + get_note_
    file_name(1, j) + " file exists")
    octave2[j] = 1
else:
    octave2[j] = 0

if(os.path.isfile(get_note_file_name(2, j))):
    debugMsg("j " + str(j) + " bFname " + get_note_
    file_name(2, j) + " file exists")
    octave3[j] = 1
else:
    octave3[j] = 0

if(os.path.isfile(get_note_file_name(3, j))):
    debugMsg("j " + str(j) + " bFname " + get_note_
    file_name(3, j) + " file exists")
    octave4[j] = 1
else:
    octave4[j] = 0

if(os.path.isfile(get_note_file_name(4, j))):
    debugMsg("j " + str(j) + " bFname " + get_note_
    file_name(4, j) + " file exists")
    octave5[j] = 1
else:
    octave5[j] = 0

if(os.path.isfile(get_note_file_name(5, j))):
    debugMsg("j " + str(j) + " bFname " + get_note_
    file_name(5, j) + " file exists")
    octave6[j] = 1
```

```
        else:
            octave6[j] = 0

        if(os.path.isfile(get_note_file_name(6, j))):
            debugMsg("j " + str(j) + " bFname " + get_note_
            file_name(6, j) + " file exists")
            octave7[j] = 1
        else:
            octave7[j] = 0

        if(os.path.isfile(get_note_file_name(7, j))):
            debugMsg("j " + str(j) + " bFname " + get_note_
            file_name(7, j) + " file exists")
            octave8[j] = 1
        else:
            octave8[j] = 0
```

Audio and MIDI Callback

As mentioned, the application is interrupt-driven. The user interface reacts to the buttons, but audio sampling and MIDI playing generate an interrupt intercepted by the respective functions.

The `AudioCallback()` function, shown in Listing 8-6, is associated with the audio hardware—for example, when a MIDI message sends the playnote feature. The parameters set in the `gui.json` configuration file concerning the chunk size and sampling rate are set to an acceptable default value. It is possible to increase these parameters if there is sufficient memory and the processor speed can support it. Also the audio board performance of the ADC/DAC should be verified to support it.

Listing 8-6. The AudioCallback() Function

```
def AudioCallback(outdata, frame_count, time_info, status):
    '''
    Callback associated to the audio hardware. It is
    executed when
    a MIDI message sends the playnote features.
    The audio callback is based on the samplerbox_audio
    library and
    can mix up to max_polyphony different audio buffers played
    together.

    :param outdata:
    :param frame_count:
    :param time_info:
    :param status:
    '''
    global globavolume

    rmlist = []
    ps.playingsounds = ps.playingsounds[-max_polyphony:]
    b = samplerbox_audio.mixaudiobuffers(ps.playingsounds,
    rmlist, frame_count, FADEOUT, FADEOUTLENGTH, SPEED)
    for e in rmlist:
        try:
            ps.playingsounds.remove(e)
        except:
            pass
    b *= globalvolume
    outdata[:] = b.reshape(outdata.shape)
```

The MidiCallback() function, shown in Listing 8-7, is called every time a MIDI message is received when an external MIDI device is connected through the USB MIDI interface.

Thanks to the interrupt on the MIDI interface, it is possible to play multiple notes on the touchscreen and the USB MIDI keyboard at the same time.

Listing 8-7. The MidiCallback() Function

```
def MidiCallback(message, time_stamp):
    '''
    Process the MIDI messages and plays the notes.

    :param message: The MIDI message packet
    :param time_stamp: The timestamp for correctly queue the
    messages
    '''
    global playingnotes, sustain, sustainplayingnotes
    global preset, globaltranspose, globalvolume
    global samples

    # Decode the MIDI message in its components
    messagetype = message[0] >> 4
    messagechannel = (message[0] & 15) + 1
    # Check if this MIDI message includes a note
    note = message[1] if len(message) > 1 else None
    midinote = note
    # Check if this MIDI message includes a specification of
      the velocity
    velocity = message[2] if len(message) > 2 else None

    # Assumes the message type (9) note on with velocity 0 is
    # a message type (8) note off
    if messagetype == 9 and velocity == 0:
        messagetype = 8
```

```
# If is a message type (9) note on apply eventual octave
  transposition and play
# the note
if messagetype == 9:
    debugMsg("messagetype is 9 (note on) globaltranspose "
    + str(globaltranspose))
    midinote += globaltranspose
    try:
        debugMsg("playing notes" + str(samples[midinote,
        velocity]))
        playingnotes.setdefault(midinote, []).append
        (samples[midinote, velocity].play(midinote))
    except:
        debugMsg("Exception, pass")
        pass

# Process the message type (8) note off applyin the sustain
  if it is active
elif messagetype == 8:  # Note off
    debugMsg("messagetype is 8 (note off) globaltranspose "
    + str(globaltranspose))
    midinote += globaltranspose
    if midinote in playingnotes:
        for n in playingnotes[midinote]:
            if sustain:
                sustainplayingnotes.append(n)
            else:
                n.fadeout(50)
        # Empty the played notes array
        playingnotes[midinote] = []
```

```
    # Process the message type (12) program change and load the
    # new group of samples.
    elif messagetype == 12:  # Program change
        debugMsg('Program change ' + str(note))
        preset = note
        LoadSamples()

    # Process the message type (11) for pedal off (associated
      to the sustain)
    # With note 64 and velocity < 64 (typical 0)
    elif (messagetype == 11) and (note == 64) and (velocity
    < 64):  # sustain pedal off
        for n in sustainplayingnotes:
            n.fadeout(50)
        sustainplayingnotes = []
        sustain = False

    # Process the message type (11) for pedal on (associated to
      the sustain)
    # With note 64 and velocity > 64 (typical 127)
    elif (messagetype == 11) and (note == 64) and (velocity
    >= 64):  # sustain pedal on
        sustain = True
```

PART V

The Dome with the Sandcastle

“How can a sandcastle be so magnificent?” Ray thought as he followed the sign out of the Music Garden. In the meantime, Sonya approached from his right.

His questioning expression was so clear that she answered without him asking.

“You will be fascinated and surprised, Ray,” Sonya said with her warm smile.

“I am really curious!” Ray replied. “Come! Sonya took his hand, pulling him in a rush. Ray responded to her grip, and within a few steps, they ran hand in hand toward the dome. The structure was different from anything Ray had imagined.

The last time Ray played a sport was years ago, and now his heartbeat was accelerated; his breath came in short, quick gasps from the effort. After the first look at the dome, his sight met Sonya’s eyes; he could only smile at her proud expression.

Ray realized that, until that moment, he had not considered how things between them had changed in so little time. It was evident that, for some reason, they instantly clicked. Their mutual passion for technology contributed to creating a meaningful bond. Ray appreciated Sonya’s calm

demeanor and her knowledge of all the details of this strange place. His connection with Sonya gave him a sense of reassurance as he embarked on this unexpected adventure with her constant presence beside him.

The dome was a circular structure, towering at least 30 meters high. Its interior was bathed in a surreal glow from the lights that danced and shimmered on the white sand below. The sandcastle in the center was a marvel; its spires and turrets intricately detailed, looking almost too delicate to exist. Each grain of sand seemed meticulously placed, and the way the light played off its surfaces gave it an ethereal effect. It was a sight that made Ray feel as if he had stepped into a dream or something magical.

The complexity of its architecture would take hours to understand all its secrets.

Sonya, a bit smaller than Ray, sweetly put her head against his right shoulder, enjoying Ray's surprise and admiration of what was in front of his eyes. She leaned her head gently against Ray's shoulder, a small smile on her lips. Yet, a shadow of worry crossed her mind. She had seen the sandcastle many times, but sharing it with Ray made her realize how much she cared about his opinion. Her voice was soft when she began to speak, masking a more profound, unspoken fear that this moment of wonder might be fleeting.

"Ray, I..." Sonya said. Ray smiled at her. His breath was regular now, but his heart was beating quickly.

"Sonya?" Suddenly, her tone of voice changed, as if she were trying to mask her feelings.

"Nothing important," she replied. "Look, Ray, it is magnificent, isn't it?"

Ray smiled again. "It's the most incredible architecture I ever saw." He noticed a shift in Sonya's expression but ignored this detail. Regardless of his intimate appreciation for her cautious approach, which gave him a sense of well-being, he was overwhelmed at the moment.

"It's incredible!" Ray remarked, admiring the colossal structure.

"The sandcastle is exceptionally delicate," Sonya continued. "The sand is maintained constantly, and the castle sides, the weakest top parts, are continuously adjusted," she explained.

"I see," Ray answered, interested in the explanation and focused on the sandcastle. Their hands mingled together.

He glanced at Sonya, and his engineering instincts tingled with unease. The castle looked delicate, and the thought of it collapsing under the slightest disturbance gnawed at him. Sonya's explanation about the castle's sensitivity to vibrations only heightened his anxiety. He couldn't shake the feeling that something could go wrong, even as he tried to focus on the beauty before him.

"Look, Ray. Do you see those small flying robots around the top of the castle?" Sonya asked, pointing her head to the upper part of the delicate building. They move continuously around, spraying a mixture of water and glue to keep the sandcastle stable. Ray nodded, holding her hand as if holding something precious.

"It seems it could fall at any moment," he said.

"The sandcastle's weak point is vibrations. This is the reason it is built over a thick bed of sand," agreed Sonya.

Ray was fascinated and very interested in the technical details provided by Sonya. Ray's engineering side was curious about the technology behind this incredible building. A small red light flashed on the top of the low wooden railing that bordered the perimeter. Ray noted it because Sonya unlocked a telephone handset from the bottom to answer what appeared to be a phone call.

Ray can't hide his curiosity as Sonya answered, "Yes?" Then her expression changed to a shine of happiness. After a few seconds, she ended the call and put the handset on the rail again.

"As I told you," she commented, "A ring bell here should be avoided; it is dangerous for the structure," she said, indicating the red light.

"I see," Ray answered, curious about the phone call.

"It resembles a telephone, but you see, it is also a robot. It is an efficient way to communicate news," she continued. "When a certain number of us are focused on a topic, I see if something related to it happens."

"I can imagine it was something related to the sandcastle…" Ray guessed to avoid asking her an explicit question.

"Oh, Ray, sorry! I don't know where my brain is these days," she said. "Someone has just seen a young boy on the opposite side of the dome. I imagine it might be your son."

"Tommy!" Ray whispered.

Tommy

"Dad!" A voice yelled on the other side of the wide circumference of the dome. "Tommy!" Replied Ray.

Finding Tommy in this strange and wondrous place felt like a miracle. Ray's heart swelled with relief and pride as he heard his son's voice calling out to him.

"Let him come to us," suggested Sonya before he moved a single step. "It is more secure if he walks around the rail," she concluded.

"Follow the perimeter. I am here!," Ray yelled to his son, indicating the direction with large signs of his arms. Animated by uncontainable happiness, Ray looked Sonya in her eyes, and then they held each other tightly in a deep hug, their joy finally overflowing in finding Tommy. Maybe it was the enthusiasm of the situation, but Ray enjoyed the sweetness of Sonya, her body near to his heart, his arms around her shoulders.

Tommy moved a few steps around the dome's perimeter, fenced off by the wooden rail. Unfortunately, his enthusiasm for meeting his father as soon as possible and his scientific and pragmatic education made him take the shorter path.

Ray and Sonya widened their eyes, seeing Tommy jumping over the rail to shorten his distance and running through the sand in their direction. Tommy ran effortlessly quickly, so telling him to step back was too late.

"Oh no!" Ray exclaimed. "Oh, my God!" said Sonya.

Tommy was about 20 meters from them when everything around started rumbling. A spectacular and disastrous domino effect started.

First, the highest pinnacles of the sandcastle became powder clouds in a matter of seconds. The maintenance robots flew away just before being hit by the castle, which started an unstoppable self-destruction process.

As more sand fell and the sandcastle shape became more undefined, a growing dust cloud progressively saturated the entire dome area.

Ray, Sonya, and Tommy remained immobile, unable to move a single muscle.

The catastrophic spectacle lasted almost five endless minutes. Then, the rumble suddenly stopped. It took another 15 minutes before the dust diminished, making it possible to see.

They looked like ghosts, covered in a fine white sand. In the distance, an alarm bell rang for another couple of minutes. Then, what remained was only dust and silence.

"Ray…," said Sonya. Ray was almost sure he saw tears on her cheek, maybe due to the dust, or he imagined it because of her voice, a tone he was not used to. The colored lights now lit the dome with white, bright light.

Tommy was shocked. He was still standing on the sand, his face inexpressive. He looked like he might start to cry at any moment.

A piece of the dome rail near a cabin, perhaps the control room, silently went down. A couple of men dressed in uniforms similar to the doorman of the "Music Garden" started crossing the dome in the direction of Tommy, walking with a martial step.

Sonya took the telephone headset from under the rail.

"There is another headset near you," she said, inviting Ray to take it. He was still too astonished to manifest any reaction. He obeyed.

As the two men reached Tommy, they took him under their arms.

"You are officially accused of the deliberate destruction of the sandcastle, an invaluable opera designed and built by the creator of BDTH 6159," they told Tommy in a solemn tone. Ray and Sonya heard this in their headsets.

"You will be processed and eventually condemned for this crime."

As they finished, their metallic voices stopped with a "click," and they pulled Tommy toward the cabin. Sonya became pale.

"Hey!" Ray screamed. "It's a mistake! He can't be guilty!" Ray was yelling at Tommy and the two, who ignored him completely.

In the meantime, someone put a hand on his shoulder.

"You are officially accused of participating in the deliberate destruction of the sandcastle, an invaluable opera designed and built by the creator of BDTH 6159. You will be processed and eventually condemned for this crime."

Speechless, Ray looked at Sonya dumbfounded. "Follow them, Ray. Meanwhile, I will try to discover what will happen. I will reach you as soon as possible, dear," said Sonya.

Before Ray could reply, the two men moved, accompanying him in the same direction where Tommy had gone.

CHAPTER 9

The Sand Machine Part 1

Figure 9-1. *A suggestive view of an image created with the Sand Machine*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_9

> *A mandala is a geometric configuration of symbols. In various spiritual traditions, mandalas may focus the attention of practitioners and serve as a spiritual guidance tool for establishing a sacred space. They aid in meditation and trance induction. In the Eastern religions of Hinduism, Buddhism, Jainism, and Shinto, mandalas are used as maps representing deities or paradises. (Source: Wikipedia)*

9.1. The Idea

Inspired by the previous episode of Ray Badmington and his son, I decided that creating a mandala machine was the best way to make it real. *Sand Machine* was the project's codename.

Digging deeper into the structure of mandalas (see Figure 9-1), I discovered that they can be easily re-created as a recursive design generated by complex mathematical functions. A suitable classification of the traditional mandalas and their geometry can be found in the article "Sacred Geometry Mandala Art" by Jonathan Quintin (`http://www.isibrno.cz/~gott/mandalas.htm`).

The first limit I set up was creating a monochromatic mandala design due to the practical realization I had in mind. According to what the traditional mandala represents, the idea is to draw the design variation over time—a continuously changing design following mathematical rules that can be parametrized to create new designs every loop.

The best representation of time passing is the sand in an hourglass; under a conceptual aspect, the Sand Machine can be considered a machine tracking time on the sand as a mesmerizing hourglass. See Figure 9-2.

Figure 9-2. *A perspective view of an Archimedean spiral drawn by the Sand Machine*

9.1.1. Mathematics

Implementing the mathematic algorithms for this project, I fixed some limits to constrain the wide range of available possibilities to draw a mandala figure with parametric equations. Therefore, I decided to base the drawing algorithms on the *cycloid* functions family. See Figure 9-3.

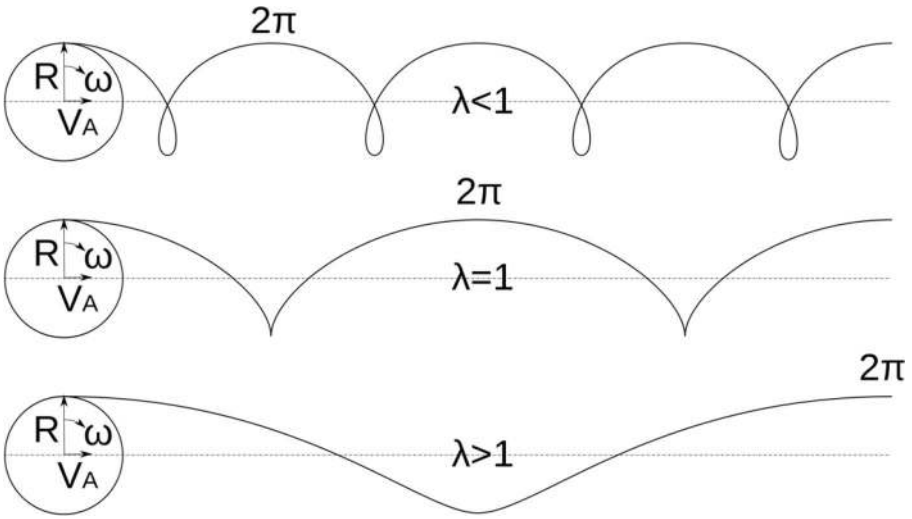

Figure 9-3. *Different kinds of cycloids changing the control parameters of the rotation. Credits: Licensed under the Creative Commons Attribution-Share Alike 3.0,* https://commons.wikimedia.org/wiki/File:Cycloids.svg

A *cycloid* is a curve generated by a point on a circle rotating along a line. It is possible to generate a quantity of different curves by changing parameters like the line's distance from the circle's center and radius.

For a complete mathematical description of cycloids, see Wikipedia: https://en.wikipedia.org/wiki/Cycloid.

The simplest cycloid is generated by a circle of radius *r* rotating over the x-axis on the positive side so that all the y-values are positive; the following two trigonometrical functions calculate every x-y coordinate:

$$x = r\left(t - sin\ sin\ t\right)$$

$$y = r\left(1 - cos\ cos\ t\right)$$

Cycloids can be classified into the following groups according to the position of the rotating point, the radius, and so on:

- **Trochoid:** The rotating point is inside (curtate) or outside (prolate) the circle.
- **Hypocycloid:** The circle rotates inside another circle instead of a straight line.
- **Epicycloid:** The circle rotates outside another circle instead of a straight line.
- **Hypotrochoid:** The drawing point is not on the rotating circle's edge but inside of it.
- **Epitrochoid:** The drawing point is not on the edge of the rotating circle, but is outside.

In this project, I am interested in drawing hypocycloids according to the design of the machine.

Why Cycloids?

The Sand Machine should "draw" a mandala shape on sand without interruption, like a plotter that can't pull up the pen after a recursive drawing has started.

According to this limitation, this kind of curve is perfect for the project's scope without limiting the infinite number of possible drawing variations. A good explanation (with animations) of the cycloids' equations and their cartesian conversion can be found on the Wolfram MathWorld site (https://mathworld.wolfram.com/Cycloid.html).

Coordinate Representation

There is a last important detail I had to take care of: the coordinate's representation of the curves. The cycloid formula is defined in a trigonometrical format.

At the same time, the final project should be able to draw (on sand) a curve using *cartesian coordinates* (a series of x-y pairs for every point of the curve). The coordinates conversion is the most complex mathematical concern and I will describe it better in the chapter dedicated to the software.

9.2. Mechanics

The mechanical solution for this project should answer the following question:

> *How can I draw on the sand with a tool that seems to move autonomously?*

I figured out several possibilities then. According to what was available in my lab, the concept focused on two different aspects:

- Using an iron ball moved by a magnet. This solution can do the job without any visible control, while the iron ball also has an excellent aesthetic impact while driving on the sand surface.
- Moving the ball following the path using an iron ball.

A powerful magnet is the choice! See Figure 9-4.

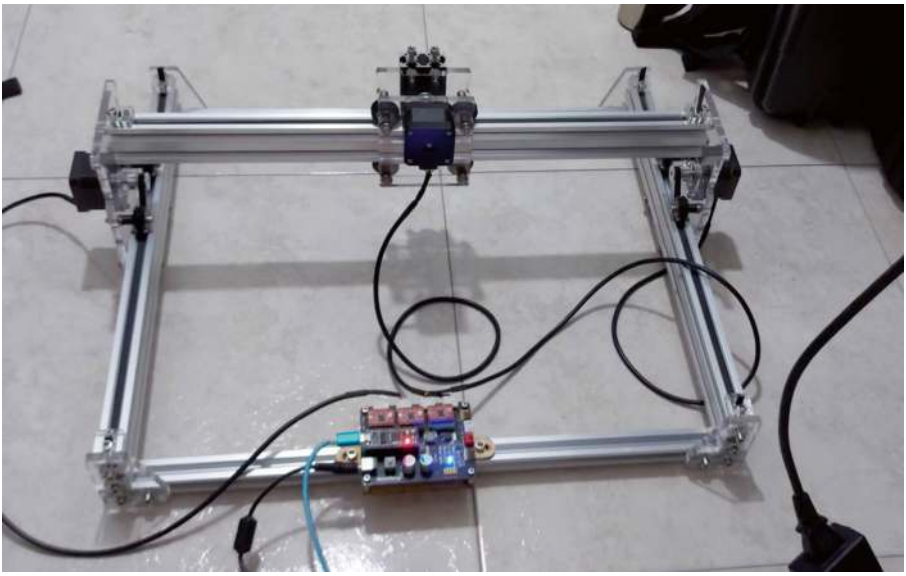

Figure 9-4. *The bare structure of an old low-power laser engraver is the ideal tool to recycle to move a magnet on a cartesian plane*

In my case, I adopted this solution because a simple structure was already available. Recycling, with some modifications, a homemade laser engraver resulted in the perfect mechanism to move the magnet along the cartesian axes. Indeed, any plotter-like structure can be used for the same task.

9.2.1. The Sand

Before starting this making adventure, I also checked the availability of some calibrated sand types for use as a nonpersistent drawing surface.

I have always been fascinated by the idea that visual art can change over time; from this perspective, drawing on sand is one of the best representations of this concept.

One of the most exciting aspects of mandala geometry is the use of colors; unfortunately, coloring is impossible, so I compensated for this aspect with the machine I had in mind. I also considered adding some color lighting effects to give the project a magical touch.

The type of sand was a challenging choice. The first try was to buy decorative white sand 1mm in diameter. However, this sand was not regular. This kind of sand—easy to find on Amazon.com—is perfect for decorative and model makers' purposes and is obtained by breaking silica into fine parts. The result is an irregular sand that, when moved by a ball rotating on the surface, produces a lot of fine dust deposing on the internal transparent cover of the Sand Machine.

The solution came from a very different product: the waterjet cutter machine, calibrated for spherical sand and a high-power jet of water to cut materials, including hard metal (see Figure 9-5). According to the experiments, this sand produced the best performance with minimal dust production.

Figure 9-5. *The sand used by the waterjet cutter is produced in a spherical shape of 1mm diameter. This artificial sand has the best performance to make the Sand Machine*

9.3. The Design

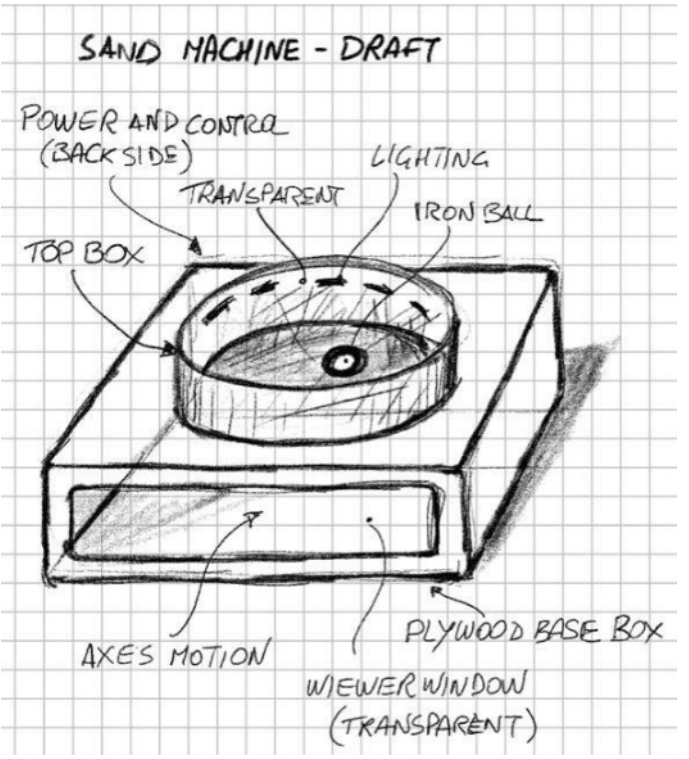

Figure 9-6. *The first draft design of the Sand Machine*

When making a project from scratch, including a structure like this machine, choosing the suitable building materials is an important aspect that can affect all the next steps.

This is why, before starting to make the project—the design first—I always try to focus on all the key points and issues to have a clear idea of the result. See Figure 9-6. Indeed, as the project is not a copy of another one already documented, I know the final result will be subject to modifications, changes, and adaptations as I proceed to make it.

9.3.1. From Draft to Components

According to the draft, two different boxes should be created: the base squared box that will host the moving X-Y coordinates—you can assume it is a significantly simplified CNC—and the top box exposing the mandala sand show.

The structure does not have to be particularly strong, as no mechanical stress is applied to the two boxes. I chose to make the structure using 3mm thick plywood, which is cheap and easy to laser-cut. The parts that should be instead will be laser-cut from a sheet of transparent acrylic, also 3mm thick.

Using an algorithm based on cycloids, the drawing field will be a round dome, so the top side was drafted with a cylindrical shape. Keeping this form factor but considering the difficulty of making a round-shaped box, I adopted a compromise—a regular polygon, which will definitely be an octagon. It was perfect for hosting the sand inside. See Figure 9-7.

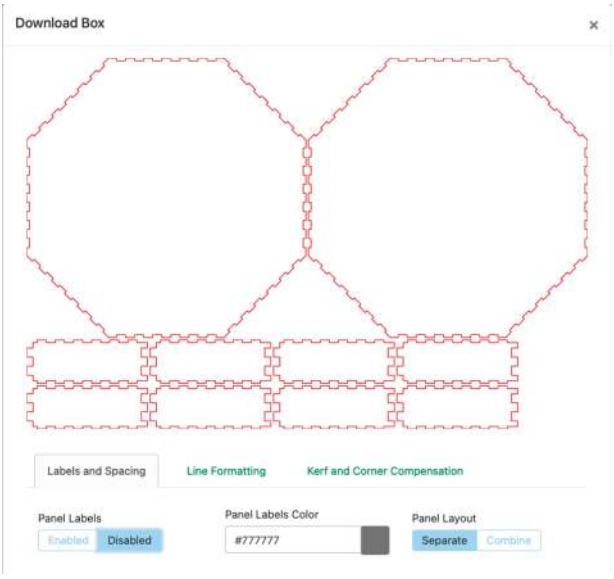

Figure 9-7. *These are the components of the octagonal box where the Sand Machine will draw the mandalas*

Thanks to this compromise, I saved a lot of time using the design tools offered for free by the Makercase portal (https://en.makercase.com/).

From this site, it is possible to design almost any kind of box, indicating the characteristics of the material and how the design components will be joined to create a wide range of containers. All the parts, generated online in seconds, can be exported in SVG format for easy editing with the Inkscape open source application or sent directly to the laser cutter.

While the top box was easy to draw without any modifications from the original file generated by the Makercase web application, the design of the bottom box required further changes. See Figure 9-8.

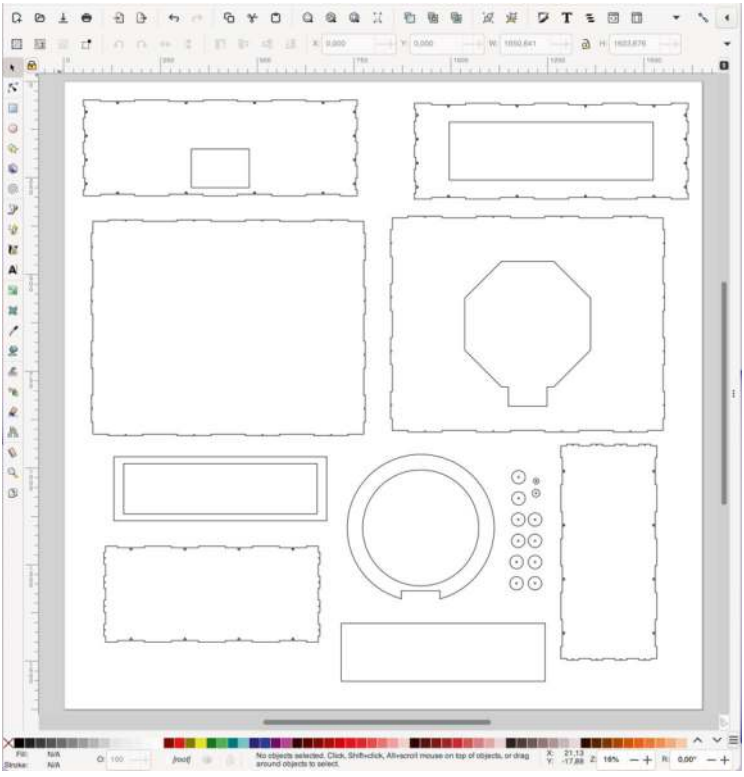

Figure 9-8. *The components of the bottom box of the Sand Machine, ready for laser cutting after being edited with Inkscape*

As shown in Figure 9-8, the small series of circles are not part of the box; they have been laser-cut from an acrylic sheet 4mm thick and screwed together to create the support for the magnet that will move the iron ball on the top side of the machine. See Figure 9-9.

Note The dimensions of the two boxes depend on two factors. The first is the physical dimension of the axis movement structure, which should be considered the area occupied when the position of the tool is at its maximum extension. The top box size, instead, depends on the effective area covered by the axis movement. For this reason, the dimensions vary depending on the kind of tools you decide to use.

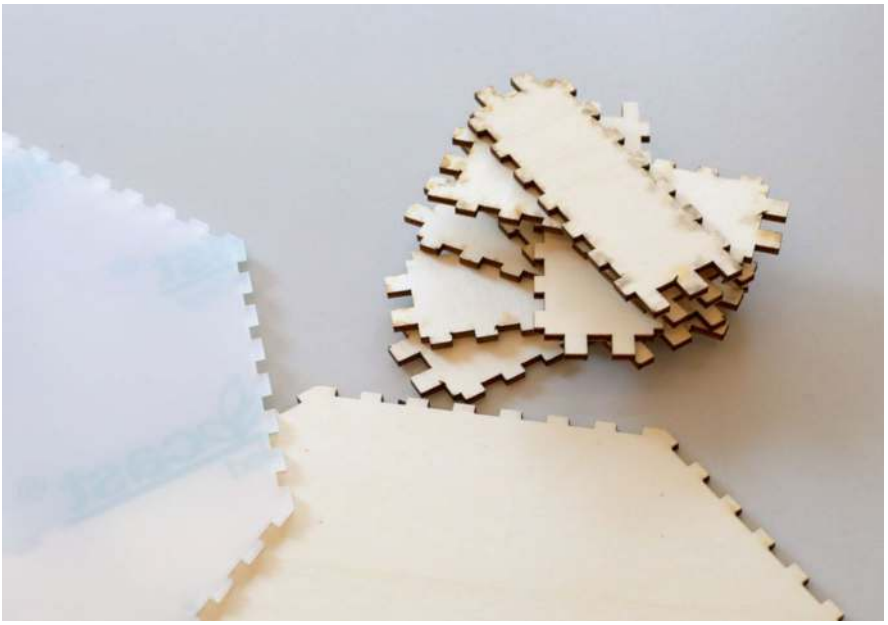

Figure 9-9. *A detail of some components after laser-cutting, ready for assembly*

CHAPTER 10

The Sand Machine Part 2

Figure 10-1. *The Sand Machine top box was already painted during the final assembly step. Note to the left the first decorative sand used for the first test, which was then replaced with the waterjet calibrated spherical sand*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_10

10.1. The Top Box

Considering some significant limitations, it was possible to simplify the material cutting using a modular building (see Figure 10-1). By adopting a magnet to move the iron sphere to draw on the sand, the bottom side of the box should be relatively thin. After several experiments, I found the most reliable compromise: an iron ball with a 10mm diameter can move easily over a few millimeters of sand, supported by the bottom surface made with plywood that's 3mm thick. See Figure 10-2.

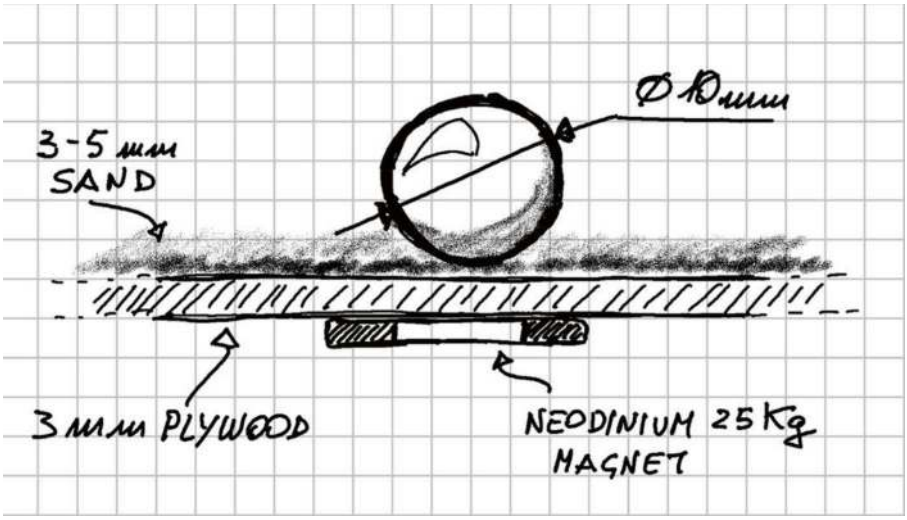

Figure 10-2. *The iron ball's asset inside the octagonal box (the top box) and the magnet. When the magnet works, it is separated from the iron ball only by the 3mm plywood base and the sand*

10.1.1. The Octagonal Dome

The graphic structure of mandalas consists of a recursive curve within a circular design. Therefore, the sand container must respect this characteristic. Indeed, building a circular dome required a different choice of materials and a more complex structure; for this reason, I approximated the sand dome (the top box) to a regular polygon.

Of course, it is possible to increase the number of sides, for example, to 12 or 16. In this case, the number of components increases, and the assembly process becomes more complex. The decision on the number of sides is the maker's. Regardless of the shape of the box, the action will be inside a circular area, constrained by the maximum extension of the CNC structure that moves the magnet.

Assembling the Sand Dome

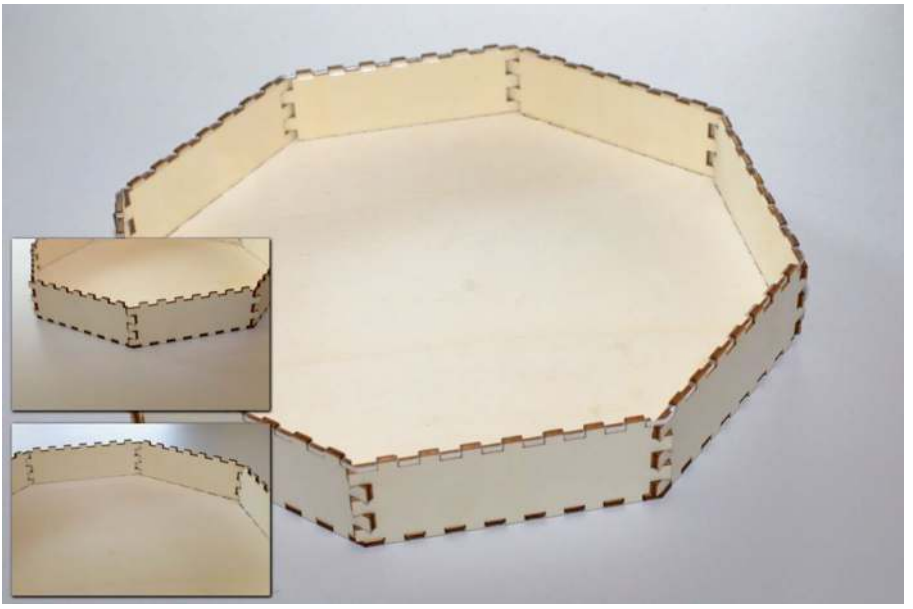

Figure 10-3. *Top view of the sand dome. The corners corresponding to the side junctions are glued together, while the top tooth is used to keep the transparent cover in place.*

After preassembling the box with paper adhesive tape to keep the pieces in place, I glued it along the internal and external borders with vinylic glue. See Figure 10-3.

Tip The glue along the borders—especially on the internal side of the box—has a double function: it keeps the components together and impermeabilizes the container to avoid dispersion of the sand.

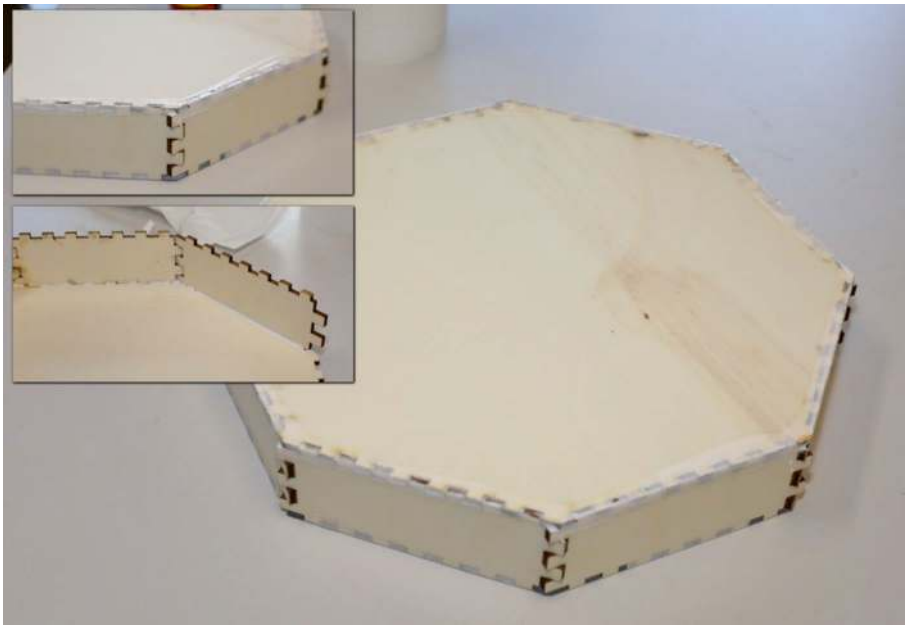

Figure 10-4. *When the assembly of the sand dome is completed, It is essential to check that all the components are filled with glue to avoid sand and dust getting outside while the Sand Machine is working*

Note Using the prototype for a considerable time, I noted that the waterjet sand also generates some dust due to the attrition of the iron ball on the sand. If the box is sealed, the Sand Machine remains clean (see Figure 10-4). It is strongly recommended to empty the sand and replace it after extended usage, about every 15-20 hours.

Assembling the Lighting

To add lighting to the sand drawing scenario, I opted for a set of Neopixel LED arrays. A comprehensive guide on Neopixel LED arrays can be found on the Adafruit site at https://learn.adafruit.com/adafruit-neopixel-uberguide.

The Neopixel library works fine on the most basic versions of Arduino UNO R3 (https://docs.arduino.cc/hardware/uno-rev3) and Arduino Nano (https://docs.arduino.cc/hardware/nano). Basically, the Neopixel LED arrays are special sets of RGB LEDs—there are many kinds available with different designs and a variable number of LEDs, up to 128—where every unit is addressed independently, setting the color. This enables a wide range of lighting effects, which is the reason these devices are found in hundreds of applications.

The limitation—especially with low processor power microcontrollers—is that the board cannot be used for other kinds of intensive processing and get good performance. This limitation makes it impossible to implement the lighting feature together with the board, which is also based on Arduino. For this reason, I implemented the lighting features of the Sand Machine using a dedicated Arduino UNO R3 board.

Note The Neopixel arrays are controlled by varying the frequency of a single pin. As they are individually addressable, single arrays can be chained together. To make them work as a single array, it is sufficient to configure the Neopixel library on the Arduino sketch, setting the total number of LEDs from the resulting chained arrays. See Figure 10-5.

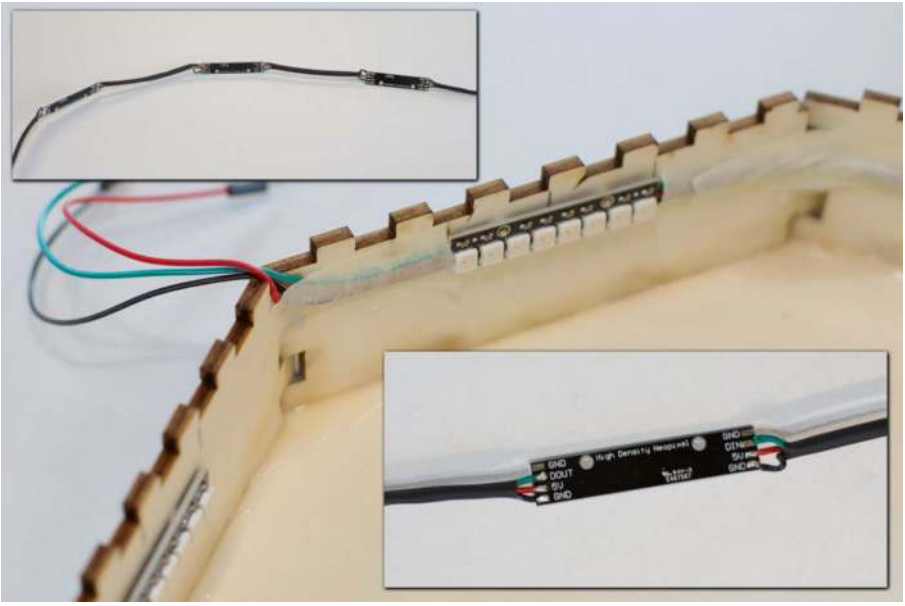

Figure 10-5. *The preassembled set of eight arrays of eight Neopixel LEDs I chained. Every group of eight LEDs is positioned on one of the eight sides of the box to light the whole sand dome*

The lighting configuration of the Sand Machine uses 64 Neopixel LEDs. Due to the high frequency needed to control the integrated circuits that drive every LED, with a large number of LEDs, it is necessary to add a 1000uF capacitor as near as possible to the power source of the Neopixel array.

Caution The power supply pin from the Arduino board has a limit of about 200mA. Suppose the number of LEDs in the Neopixel array increases over 64 units. In that case, providing a separate power source to the array is necessary instead of using the microcontroller's power pin.

In most the cases, it is an excellent practice to create electronics, circuits, and testing assemblies on the bench before installing them in the prototype. Unfortunately, I could not do that in this case. The components of the Sand Machine need to be assembled inside the structure to test the result continuously. Without the same prototype structure, the electronics and mechanics can't be tested. See Figure 10-6.

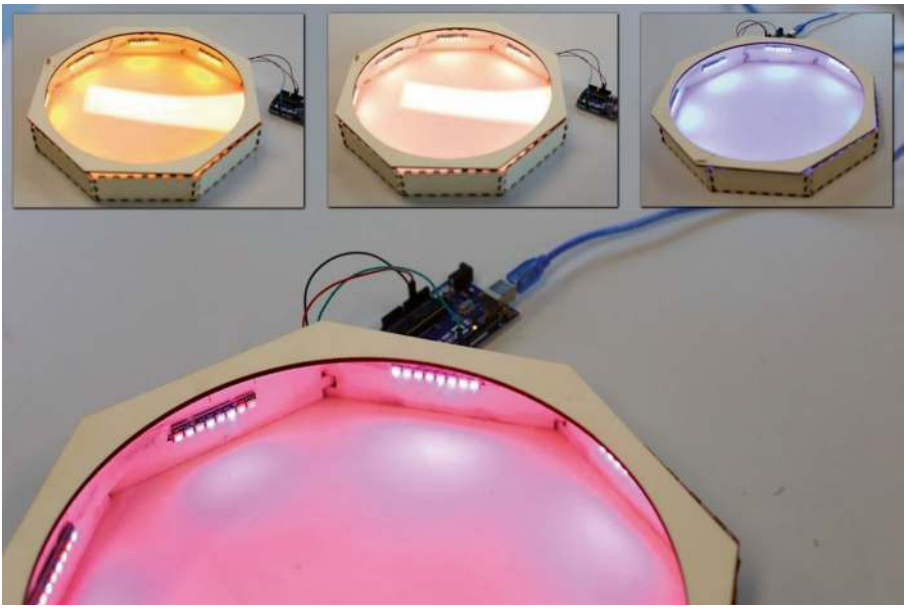

Figure 10-6. *A preassembled Neopixel array connected to the Arduino UNO R3 shows a flat color test*

After soldering the eight Neopixel arrays at a proper distance, I added a black heat shrinkable sheath to protect and hide the wiring connecting the eight chained arrays. The preassembled cable should be glued in place after painting the sand dome.

Painting the Sand Dome

Figure 10-7. *Detail of the interiors of the sand dome. The chained Neopixel arrays are distributed on the top of every side of the box, while the three wires (power and control signal) are connected to the Arduino UNO R3 through the bottom of the box*

For the external painting, I chose colors according to my taste. It does not matter which colors you choose, but for the internal side of the box—where the show should happen—I painted matte black to maximize the lighting effect and the reflections of the sand and the iron ball. See Figure 10-7.

When the sand dome is closed, the lighting effect is very interesting.

After being painted, the Neopixel strips were glued in position. For better wiring, I used a breadboard Arduino UNO shield, but no special circuit is needed.

10.2. The Neopixel LED Controller

The software of the dedicated Arduino UNO R3 is based on the Neopixel library for Arduino, developed by Adafruit. The library has reached version 1.11, and the Arduino official documentation is available on the Arduino site (https://www.arduino.cc/reference/en/libraries/adafruit-neopixel/). The library repository, instead, is available as open source on GitHub (https://github.com/adafruit/Adafruit_NeoPixel).

10.2.1. The Arduino Software

As mentioned, to build a neat project, I left the CNC firmware untouched and running on an Arduino Nano—demanding the lighting effects to a dedicated Arduino board. The sketch can work on both versions of microcontrollers—Arduino UNO and Nano. Development and testing were completed on boards based on the AVR 328p. See Listing 10-1.

Note In order to reduce the Neopixel burnout risk, add 1000uF capacitor across pixel power leads, add 300-5000hm resistor on the first pixel's data input, and minimize the distance between Arduino and the first pixel. Avoid connecting to a live circuit. If you must, connect to GND first.

Listing 10-1. The Sketch's First Part (Mandatory) Includes the Adafruit Library and the Definition of the Settings

```
#include <Adafruit_NeoPixel.h>

#ifdef __AVR__
  #include <avr/power.h>
#endif
```

```
//! GPIO Pin used to generate the Neopixel pulses
#define NEOPIXEL_PIN 6
//! Number of Neopixel LEDs in the strip
#define NEOPIXEL_LEDS 64

Adafruit_NeoPixel strip = Adafruit_NeoPixel(NEOPIXEL_LEDS,
NEOPIXEL_PIN, NEO_GRB + NEO_KHZ800);
```

The `Adafruit_Neopixel` library should be instantiated by the class constructor passing three parameters: the number of LEDs (total), the pin number connected to the Arduino board, and one of the settings available from the library. These settings depend on the kind of Neopixel model that is used in the implementation.

The current supported models are listed here:

- `NEO_KHZ800`: 800 KHz bitstream (most Neopixel products w/WS2812 LEDs)
- **`NEO`**`_KHZ400`: 400 KHz (classic v1, not v2, FLORA pixels, WS2811 drivers)
- `NEO_GRB`: Pixels are wired for GRB bitstream (most Neopixel products)
- `NEO_RGB`: Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
- `NEO_RGBW`: Pixels are wired for RGBW bitstream (Neopixel RGBW products)

More information about the Neopixel supported hardware can be found on the Adafruit website at https://learn.adafruit.com/adafruit-neopixel-uberguide?view=all.

Listing 10-2 shows the simple `setup()` function of the sketch. Here, the Arduino is dedicated to the control of the lighting effects, and the only required initialization is a call to the library instance, setting all the LEDs to off.

Listing 10-2. Simple setup() Function of the Sketch

```
//! Startup and initialization
void setup() {
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}
```

To simplify the management of the colors, after a series of experiments, I defined a series of constants that determine the selected colors to make it easier to select through their symbolic names. See Listing 10-3.

Listing 10-3. Set of Predefined Colors

```
// Predefined colors
#define PINK strip.Color(255, 64, 64)
#define WHITE strip.Color(255, 255, 255)

#define FIRE1 strip.Color(255, 64, 0)
#define FIRE2 strip.Color(255, 96, 0)
#define FIRE3 strip.Color(255, 128, 0)
#define FIRE4 strip.Color(255, 96, 32)
#define FIRE5 strip.Color(255, 32, 64)
#define FIRE6 strip.Color(255, 32, 96)

#define BLUE1 strip.Color(0, 0, 255)
#define BLUE2 strip.Color(0, 32, 255)
#define BLUE3 strip.Color(0, 64, 255)
#define BLUE4 strip.Color(0, 96, 255)

#define PURPLE1 strip.Color(96, 16, 255)
#define PURPLE2 strip.Color(96, 16, 128)
#define PURPLE3 strip.Color(96, 16, 96)
#define PURPLE4 strip.Color(96, 16, 64)
```

The main `loop()` function is simplified as well. The hard job is put in charge of the two running functions—`rainbow` and `rainbowCycle`. See Listing 10-4.

Listing 10-4. The rainbow and rainbowCycle Functions

```
//! Main loop
void loop() {

 rainbow(10);
 rainbowCycle(10);

}
```

This approach to the sketch code architecture makes the development, debugging, and parametrization of the light effect functions easy.

After the `loop()` function, I added a series of APIs to generate many different light cycling effects. By using different calls in a different order, it is possible to change the program behavior with no effort. See Listings 10-5 through 10-10.

Listing 10-5. Executes a Specific Command Flashing the White Light for a Predefined Time

```
void cmdFlash() {
    colorFlash(BLUE1, 1000);
}
```

Listing 10-6. Fills the Dots One After the Other with a Color

```
void colorWipe(uint32_t c, uint8_t wait) {
  for(uint16_t i=0; i<strip.numPixels(); i++) {
    strip.setPixelColor(i, c);
    strip.show();
```

```
    delay(wait);
  }
}
```

Listing 10-7. Fills the LEDs One After the Other with a Color, Then Sets Them Off in the Same Sequence

```
void colorWipeOnOff(uint32_t c, uint8_t wait) {
  // Pixels on
  for(uint16_t i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, c);
    strip.show();
    delay(wait);
  }
  // Pixels off
  for(uint16_t i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 0x000);
    strip.show();
    delay(wait);
  }
}
```

Listing 10-8. Flashes a Specific Color for a Number of Milliseconds

```
void colorFlash(uint32_t c, uint8_t wait) {
    setColor(c);
    delay(wait);
    setColor(0);
}
```

Listing 10-9. Sets All the Strip LEDs to the Desired Color

```
void setColor(uint32_t c) {
    uint16_t i;

    for(i = 0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, c);
    }
    strip.show();
}
```

Listing 10-10. Shows a Rainbow with All the Color Configuration and a Progressive Fill at the Desired Milliseconds Interval

```
void rainbow(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256; j++) {
    for(i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel((i+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Slightly different, this makes the rainbow equally
distributed throughout
void rainbowCycle(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256*5; j++) { // 5 cycles of all colors on wheel
    for(i=0; i< strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel(((i * 256 / strip.
      numPixels()) + j) & 255));
```

```
    }
    strip.show();
    delay(wait);
  }
}
```

The functions in Listing 10-11 is used by all the other functions. It retrieves an input value between 0 and 255 to calculate the color value. Colors are a transition `r - g - b`, back to `r`.

Listing 10-11. Sets a Color to the Specified LED

```
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85) {
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if(WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}
```

CHAPTER 11

The Sand Machine Part 3

Figure 11-1. *Internal view of the moving mechanism upcycled from a CNC structure inside the bottom box of the Sand Machine*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_11

11.1. The Bottom Box

The most challenging part of the project was the design of the bottom box and the extra components needed to make the CNC (computer numerical control) mechanics work as needed. Until the bottom part of the Sand Machine was completed, it was impossible to make any test with a very low tolerance. See Figure 11-1.

11.1.1. Top Side

Figure 11-2. *The Sand Dome inserted into the octagonal hole of the top side of the bottom box*

To make the top side as close as possible to the moving magnet, the top of the box was cut to the center: an octagonal shape to fit the Sand Dome (see Figure 11-2). The front and back sides required a last-minute update. To keep the CNC aluminum frame centered concerning the top octagon, I had to cut a rectangular opening in the back for air circulation and easy access to the CNC control board.

Another problem was that even after the box is assembled, it is necessary to put your hands inside, especially during the development phase. To address this need, I made a large front window—a plywood frame with thin acrylic coverage—that's removable for maintenance. It shows the mechanism at work while drawing on the sand.

11.1.2. Putting the Box Together

Figure 11-3. *Preassembled view of the box: the parts fit together on the first try!*

For the final assembly of the box, I followed the same gluing procedure I applied successfully to the top side. The front window is placed and removed using four small magnets, while the top cover was not glued to make any intervention from the top possible. Gluing and fixing the top of the box is the very last task at the end of the project. See Figure 11-3.

11.1.3. The Magnet Support

Figure 11-4. *Detail of the magnet support on the z-axis head*

The CNC frame was initially built for a small laser cutter and engraver. This means that the zero point is the surface of the cutting sheet, while the z-axis increments (positive values) pull down the z-axis. See Figure 11-4.

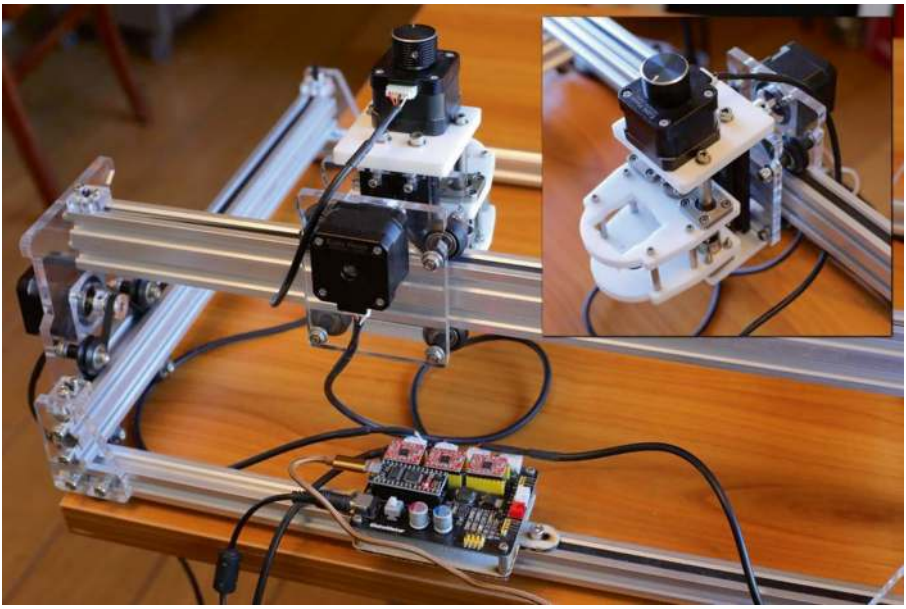

Figure 11-5. *The original assembly of the z-axis head. The motor is on top to move the laser to the bottom, while I have to invert the mechanics*

In this use case, instead, I wanted to reach the opposite—the zero point should be on the bottom, and the magnet should be pushed up (positive values) until the Neodymium magnet touches the bottom side of the octagonal base of the sand dome. See Figure 11-5.

Unfortunately, the z-axis motorized head is not designed with this scope, so I created a magnet support using a set of discs cut from a 4mm thick acrylic sheet. See Figure 11-6.

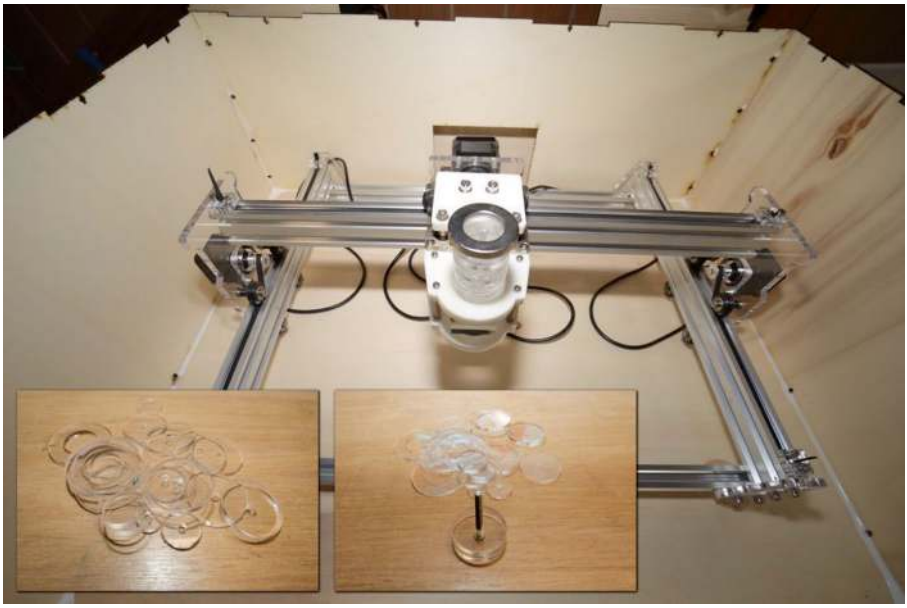

Figure 11-6. *Details of the construction of the magnet support, then glued to the top of the z-axis head. The acrylic disks are glued with cyanoacrylate glue and aligned with an M2 screw*

11.1.4. Completing the Build

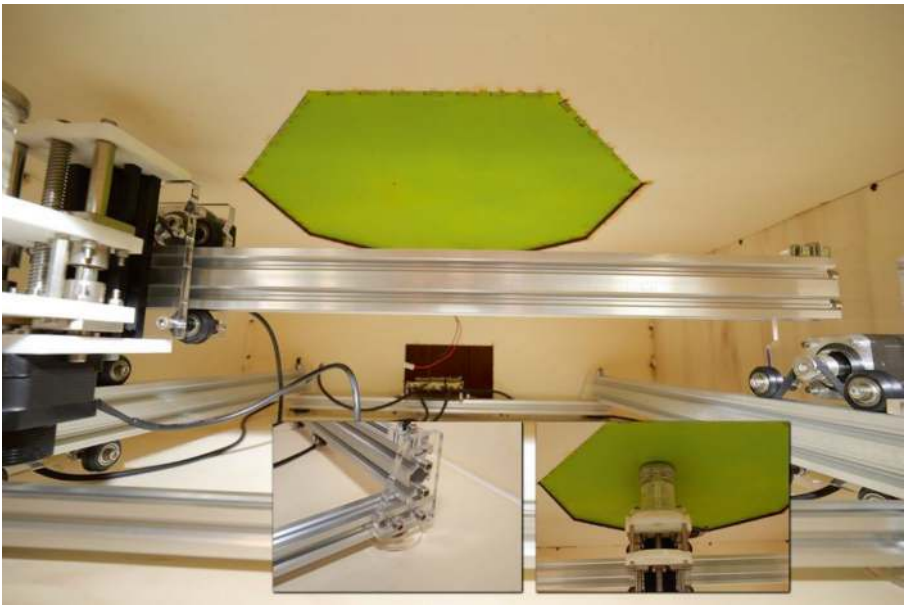

Figure 11-7. *Details of the final assembly. The CNC frame is kept in place with the four support rubbers, which fit inside four acrylic discs glued to the base*

Finally, I finished the building process!

The CNC frame has been fixed to the bottom of the box so that the magnet head (0, 0) axes origin is in the center of the octagonal hole (see Figure 11-7). Then, moving by hand, I verified that the x-y axes extension covers the sandbox area.

Lastly, I identified the exact height of the z-axis, moving it manually and measuring the distance with a caliper. The magnetic head should touch the bottom of the sandbox surface without pressing. Regardless of the thickness of the plywood (3mm) and the braking effect of the sand, small increments of the coordinates make the iron ball move smoothly. See Figure 11-8.

Figure 11-8. *Top view of the iron ball (2cm diameter) on the sand dome while testing the movements*

11.2. Controlling the Movement

The moving frame is a 3-axe CNC using four stepper motors: one for the x-axis, one for the x-axis, and two synchronized steppers for the y-axis. The motors are controlled by a popular Mana 3 controller board by EleksMaker (see https://www.amazon.com/EleksMaker-EleksMana-Stepper-Controller-Engraver/dp/B077992NK1/ref=cm_cr_arp_d_product_top?ie=UTF8). It is a 3-axes CNC controller based on Arduino Nano hosting a dedicated firmware supporting the standard G-Code. See Figure 11-9.

Note The board also includes a 5V CC-regulated power source and 12V CC power for powering the laser. Having removed the laser cutter device, I reused the 5V CC power source for the Arduino UNO R3 controlling the Neopixels lighting.

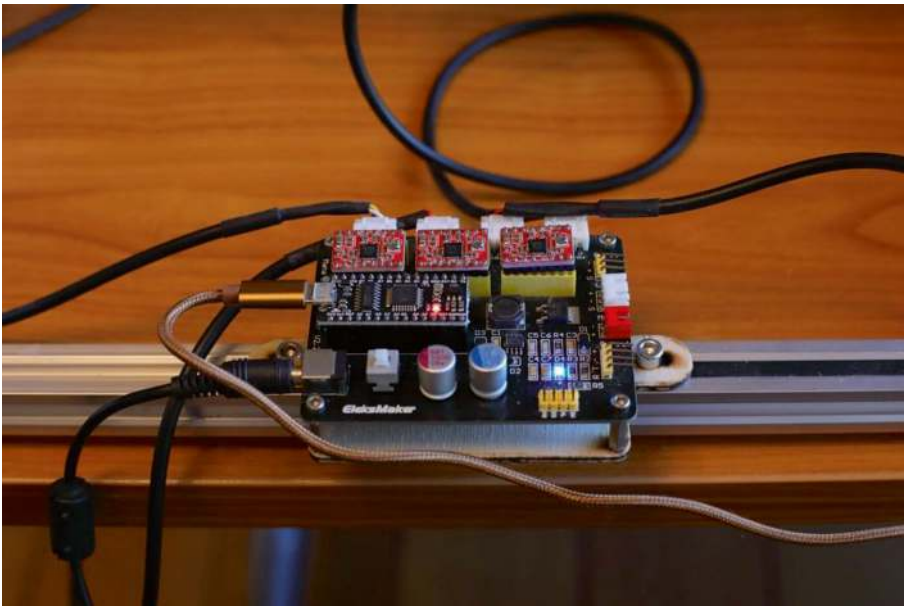

Figure 11-9. *The Mana 3-axes CNC controller assembled on the Sand Machine CNC frame. The three red boards are the x-y-z high-power motor controllers, while the board connected to the USB cable is the dedicated Arduino Nano*

The controller power should be used accordingly, depending on the torque of the stepper motors. In this case, I used Nema 17 stepper motors, a relatively low-power device, as the required mechanical effort is considerably low.

Note Several Mana 3 versions of this board report different firmware versions; the newer ones support more features and optimized firmware for better performance. In this case, I used an in-house device, upcycled for this project.

11.2.1. The Arduino CNC Firmware

While G-Code is one of the most adopted standards for CNC, 3D printers, and laser cutters, dedicated firmware should be adopted depending on the characteristics of the board.

> It became the standard de facto, created by Simen Svale Skogsrud (http://bengler.no/grbl) in 2009 and released under the open source license. The final, community-driven version 1.1 is not running on an incredible number of moving devices controlled by Arduino UNO, Arduino Nano, or Arduino Mega boards. If the maker movement were an industry, GRBL would be the industry standard.

(From the GRBL Wiki at https://github.com/grbl/grbl/wiki#grbl-v11-has-been-released-at-our-new-site-the-old-site-will-eventually-be-phased-out.)

The GRBL firmware interfaces the specific hardware controller and the standard G-Code command set.

The last update of this firmware can support many command sets; the same motor controller board can be used to drive a laser cutter/engraver, a filament or resin 3D printer, and a Mill machine, as well as find applications in some robotic projects.

Thanks to the worldwide diffusion of Arduino boards, any motor controller board designed around an Arduino is automatically GRBL-compliant.

11.2.2. What Is G-Code?

> *G-code (RS-274) is the most widely used CNC and 3D printing programming language. It is used mainly in computer-aided manufacturing to control automated machine tools and 3D printer slicer applications. The G stands for geometry.(Source: Wikipedia)*

G-Code is a textual language that enables you to control the multiple axes of a generic CNC machine. Regardless of whether the movements are focused on moving the heating extruder of a 3D printer, the rotating tool of a mill machine, or a laser cutter, the standard commands of the language and some possible customizable features make G-Code the most flexible solution.

The Language

This section contains a simplified description of the G-Code language, focused on the needs of this specific project. A full description of all the language features available from the GRBL firmware can be found on GitHub at `https://github.com/gnea/grbl`.

In this context, I am interested in focusing on how the G-Code commands (instructions) should be encoded to be parsed correctly by the firmware. A generic G-Code command has the following syntax:

```
Gnn Xnn Ynn Znn Fnn
```

Where `G` is the G-Code command ID, `X`, `Y`, and `Z` are the respective axes values, and `F` is the feed rate.

Note How the G-Code script values are parsed depends on the firmware configuration, like max motion speed, measure units, default feed rate, and so on. This solution makes the language easy to contextualize according to the hardware features and performance.

11.2.3. The Most Important G-Code Commands

G-Code commands are identified by a number: `G00`, `G01`, `G02`, and so on. This section explains the most common and frequent G-Code commands used in G-Code scripts.

G00: Rapid Positioning

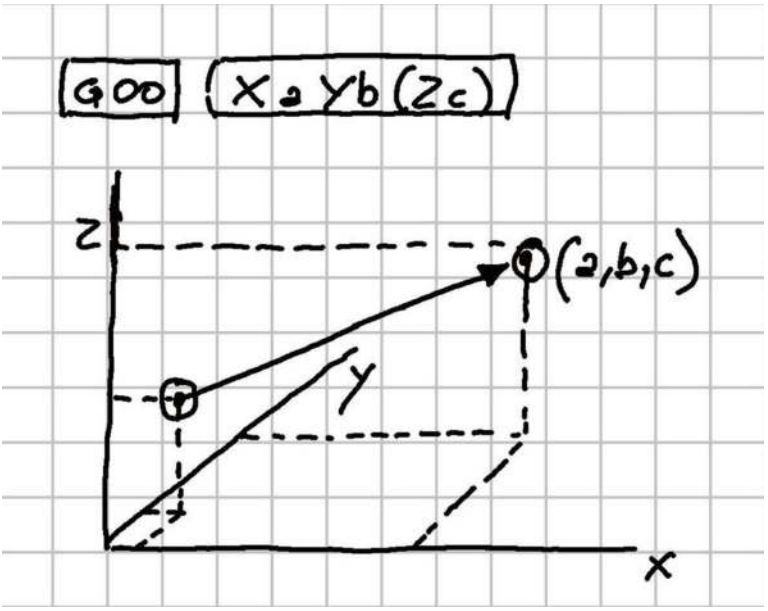

Figure 11-10. *The patch of the G00 command. The feed rate is not specified as the head (2D or 3D positioning) and is moved at the default max positioning speed.*

Typically, this command is used to move from the machine coordinates origin to the point where the job starts. The best practice is to set the z-axis to the desired height and move the tool to the desired x-y position. See Figure 11-10.

G01: Linear Interpolation

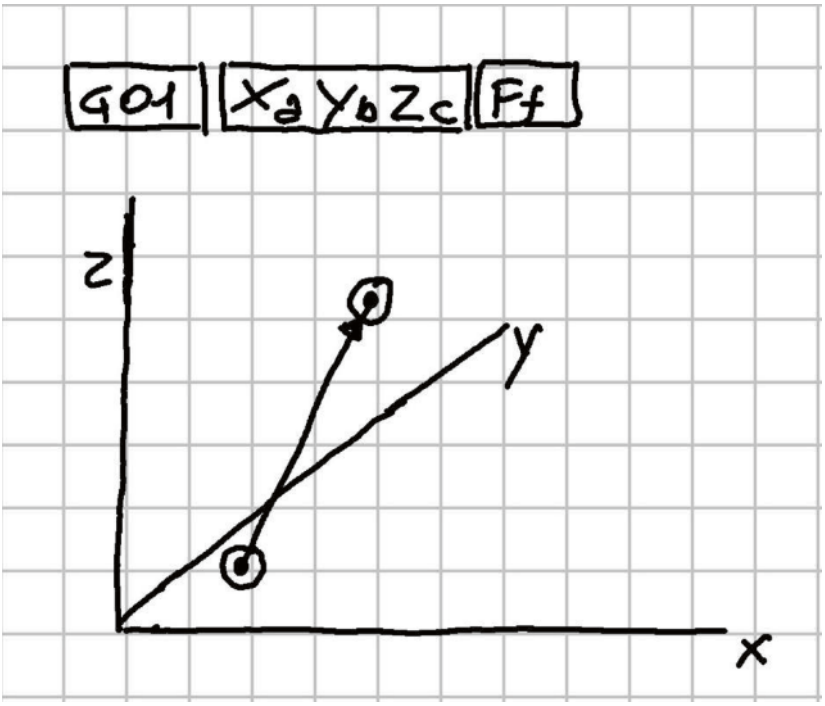

Figure 11-11. *The generic path to move the tool coordinates from one point to another. This command also requires the feed rate; slower feed rates give more precision than higher ones*

This simple three-axis move requires a considerable number of calculations to calculate the interpolation of all the points tracking the path of the movement. See Figure 11-11.

G02: Circular Interpolation Clockwise and G03: Circular Interpolation Counterclockwise

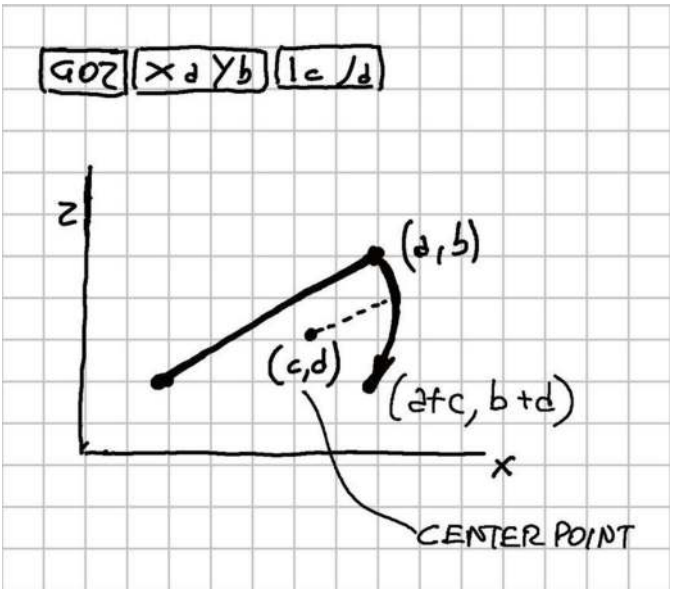

Figure 11-12. *Similar to the G01 command, this interpolation is done on a circular path from the current point and the new coordinates. Also needed are the (I, J) coordinates of the center point of the arc. This command supports the coordinates of the z-axis as well*

Similarly, the `G03` command executes the circular interpolation of a counterclockwise arc. The offsets of the arc endpoint and the center are relative to the starting point. See Figure 11-12.

Measure Units

The `G20` command selects the units in inches and `G21` is in mm according to the user's choices. These commands influence the parsing of the axes coordinates.

Working Plane Selection

Depending on the machine and tool orientation, the `G17`, `G18`, and `G19` commands select the working plane (the base surface—see Figure 11-13):

- `G17`: The X-Y plane
- `G18`: The X-Z plane
- `G19`: The Y-Z plane

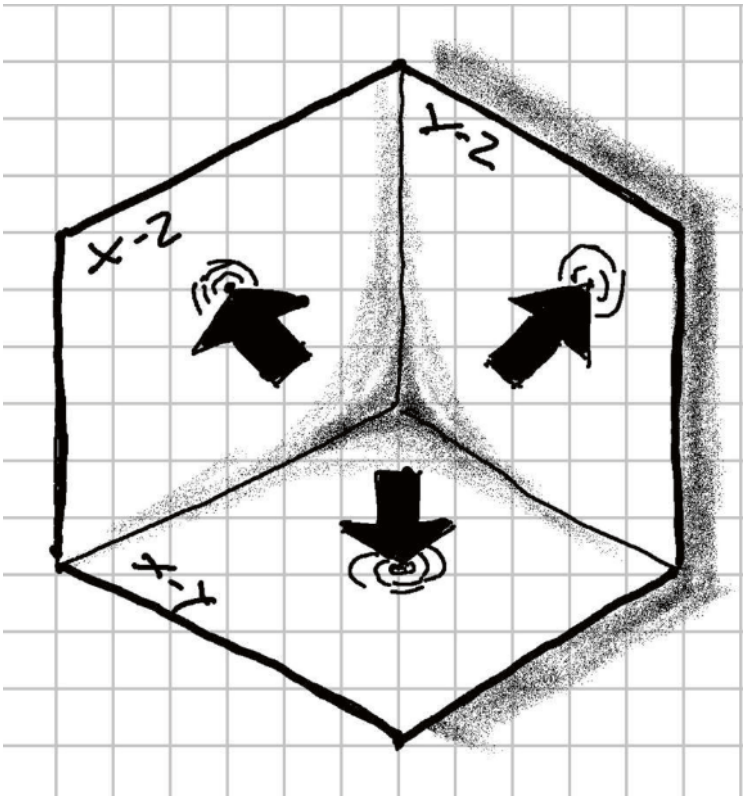

Figure 11-13. *The three possible orientations of the working tool in the 3D space are selectable by the G17, G18, and G19 commands*

G28: Home

Like the Rapid Positioning G00 command, G28 moves the tool along a straight line to the initial home position (the physical origin of the axes).

Normally, during a machining process, this command runs last at the end of the script; it is possible that returning home through a straight line from the last point will cause the tool to collide with the machined object.

To avoid this risk, the G28 command accepts an optional intermediate point (x, y, z) to follow a path that avoids collisions.

Positioning Mode

The G90 and G91 commands set the mode axes coordinates:

- G90: Absolute mode
- G91: Relative mode

In Absolute mode, the XYZ coordinates are expected to represent an exact point in the 3D space, while in Relative mode the coordinates values are considered incremental values with respect to the current position of the tool.

M Commands

The M commands instruct the machine to execute specific tasks. Some of these commands are specific to 3D printers, others to lasers and others to CNC. The following list shows the M commands used in this project. A complete list of M commands parsed by the GRBL firmware is found on the GRBL Wiki (https://github.com/gnea/grbl/wiki).

- M00: Program stop
- M02: End of program
- M30: End of program

CHAPTER 12

The Sand Machine Part 4

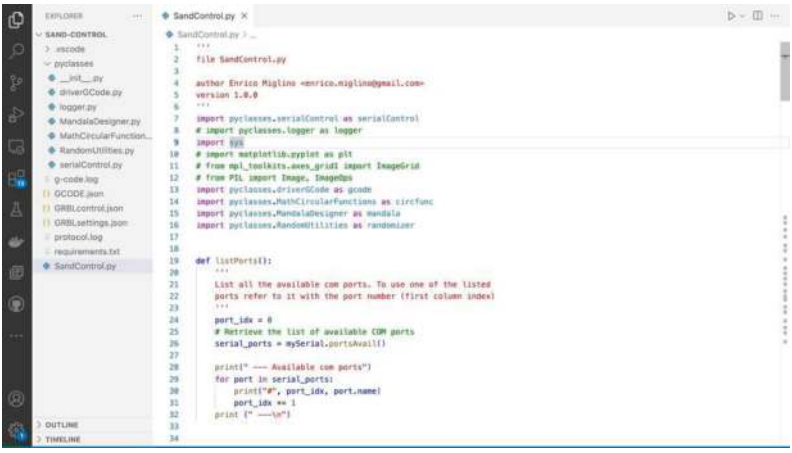

Figure 12-1. *The VS Code IDE shows the Python application and all the files of the Sand Machine controller*

The finished Sand Machine is a small CNC laser engraving, with some custom limitations due to the motion system, mechanics, and a circular working area.

What I tried to keep untouched is how the axes movements are controlled through the standard G-Code, including the commands supported by the GRBL firmware. See Figure 12-1.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_12

The connection between the machine and the controller is the USB-to-Serial interface; this method—adopted in most similar commercial CNCs—makes connecting it to many different platforms easy. I developed and used a MacBook Pro, but it will work fine with a Windows or Linux-embedded computer like the Raspberry Pi.

The software portability is granted using Python, so the control software is fully portable without requiring changes or modifications.

The rest is mathematics.

12.1. Software Architecture

Designing the software architecture, I tried to follow this statement as much as possible: the `SandControl` application should be expandable, portable, and parametrized.

In addition, I always try to write the main application to evidentiate the system's business logic, delegating the hard job to external Python classes that can be reused in the future if needed. This approach is more comfortable when new features have to be added and debugged.

A dedicated `DriverGCode` class manages the software protocol to interface the application with the machine, while a `SerialControl` class manages the low-level USB-to-Serial communication.

Instead of hardcoding the G-Code protocol, I implemented a group of JSON configuration files to initialize the machine and set the behavior of the commands sent for drawing. Following this approach, the result is an easy-to-configure and expand program with a limitless number of designs. See Figure 12-2.

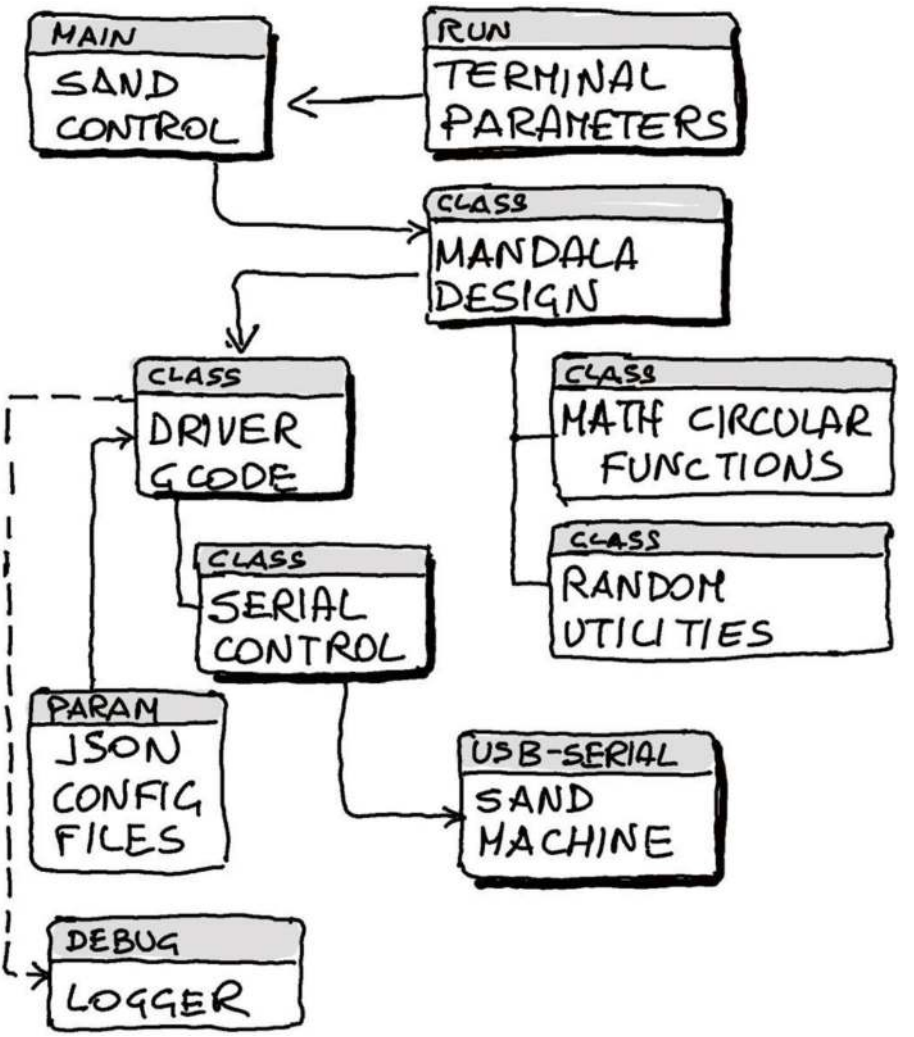

Figure 12-2. *Schematics of the SandControl Python software architecture*

12.2. G-Code Parametrization

The `driverGCode` class settings are defined through three JSON files:

- `GRBLsettings.json`
- `GRBLcontrol.json`
- `GOCDE.json`

The first two files reset the G-Code firmware (as mentioned, GRBL) to the desired operating conditions. In contrast, the third file defines some G-Code macro functions specific to the characteristics of the Sand Machine hardware size and shape.

This `GRBLsettings.json` file sets constraints to instruct the firmware on interpreting the generic G-Code protocol commands. Note that the values in Listing 12-1 are specific to this making. Limits should be redefined with a machine of different size or mechanics.

Listing 12-1. The GRBLsettings.json File

```
{
    "HomingFeed" : "$24=25.500",
    "HomingSeek" : "$25=500.500",
    "HomingDebounce" : "$26=250.500",
    "SoftLimits" : "$20=1",
    "xMaxRate" : "$110=1000.000",
    "yMaxRate" : "$111=1000.000",
    "zMaxRate" : "$112=50000.000",
    "xAccel" : "$120=100.000",
    "yAccel" : "$121=100.000",
    "zAccel" : "$122=100.000",
    "xMaxTravel" : "$130=400.000",
    "yMaxTravel" : "$131=250.000",
    "zMaxTravel" : "$132=15.000"
}
```

The GRBLcontrol.json file, shown in Listing 12-2, defines the firmware-specific control codes required to set parameters to the GRBL firmware. One of the command options of the SandControl program lists the current settings of all the GRBL firmware parameters.

Listing 12-2. The GRBLcontrol.json File

```
{
    "Settings" : "$$",
    "Parameters" : "$#",
    "GCodeMode" : "$C",
    "ParserState" : "$G"
}
```

The GCODE.json parameters file, shown in Listing 12-3, defines the constraints of the hardware structure. These commands automatically set the key points of the axes, like the lower and upper z-axis position, the maximum area extension, and so on.

Listing 12-3. The GCODE.json Parameters File

```
{
    "init" : "M3",
    "units" : "G21",
    "zdown" : "G00 Z -32.0000",
    "zup" : "G00 Z 32.0000",
    "home" : "G00 X -200.0000 Y -200.0000",
    "sand" : "G00 X 200.0000 Y 200.0000",
    "Response" : "ok",
    "program" : "%"
}
```

12.3. The SandControl.py Application

As I mentioned, the advantage of using dedicated classes for the application tasks—regardless of their reusability —concerns having the main application focused on the user interface and the business logic.

12.3.1. Imports

Making intensive use of the classes, most of the Python external libraries are used by the classes themselves. This means that every class includes the external libraries needed to work, acting as a "black box" utterly independent of the logic or the use of the class methods.

The following code shows the contents of the first lines of the main program. They define the classes that are part of the application. The `import` of the `sys` library manages the `line` command call parameters according to the desired program option:

```
import pyclasses.serialControl as serialControl
import sys
import pyclasses.driverGCode as gcode
import pyclasses.MathCircularFunctions as circfunc
import pyclasses.MandalaDesigner as mandala
import pyclasses.RandomUtilities as randomizer
```

12.3.2. Business Logic

The `__main__` application simply processes the command-line arguments and executes the corresponding call.

The current version of `SandControl.py` executes the command option and exits; it is also possible to convert it to a loop so the program can draw indefinitely until it is stopped. The program accepts one of the following commands:

- `<-l>`: Lists the available COM (serial) ports.
- `<-p port_ID>`: Shows the name (in Human Readable Format) of the port corresponding to the specified ID.
- `<-c command_name>`: Executes one of the G-Code commands defined in the `GCODE.json` configuration file (`init`, `units`, `zdown`, `zup`, `home`, or `sand`).
- `<-s port_ID mandala_design>`: Executes one of the predefined designs by name (`spiral`, `epicycloid`, `cardioid`, `nephroid`, or `ranunculoid`).
- `<-x port_ID>`: Sends the G-Code initialization sequence to the specified serial port ID.
- `<-i port_ID>`: Lists all the GRBL firmware settings.
- `<-m>`: Generates a random mandala design on the screen.
- `<-h>`: Shows a quick help.

Available to all the commands, the first task creates a local instance of the `serialControl` class (see Listing 12-4). The instance can be reused if the design command runs in a loop.

Listing 12-4. The GCODE.json Parameters File

```
if __name__ == "__main__":
    '''
    Main application.
    '''
    my_serial = serialControl.SerialControl()     # Serial
                                                  class
                                                  instance
```

```
    # Check for the command line arguments
    if len(sys.argv) <= 1:
        help()
    else:
        if sys.argv[1] == "-l":
            ''' List the available com ports '''
            list_ports()

        elif sys.argv[1] == "-h":
            ''' Help '''
            help()

        elif sys.argv[1] == "-p":
            '''
            Show the name of the port corresponding to the
            selected ID
            The second argument is the port ID.
            '''
            if len(sys.argv) <= 2:
                print("port ID parameter missing")
            else:
                if my_serial.create_name(sys.argv[2]):
                    print("Serial port name :" + my_serial.
                    get_port())
                else:
                    print("There is no port associated to
                    the id")

        elif sys.argv[1] == "-c":
            '''
            Run one of the predefined commands, as defined in
            the GCODE.json file.
            '''
```

```
    if len(sys.argv) <= 3:
        print("Command name parameter missing")
    else:
        gcode_driver = gcode.DriverGCode(sys.argv[2])
        gcode_driver.start_serial()
        gcode_driver.initGRBL()
        gcode_driver.run_command(sys.argv[3])
        gcode_driver.quit()

elif sys.argv[1] == "-s":
    '''
    Run one of the predefined designs, then return to
    home position
    '''
    if len(sys.argv) <= 3:
        print("A parameter is missing")
    else:
        sand_plot(sys.argv[3], sys.argv[2])

elif sys.argv[1] == "-x":
    '''
    Send the G-Code initialization to the specified
    serial port.
    '''
    if len(sys.argv) <= 2:
        print("Some required parameter is missing")
    else:
        gcode_driver = gcode.DriverGCode(sys.argv[2])
        gcode_driver.start_serial()
        gcode_driver.initGRBL()
        gcode_driver.quit()
```

```
    elif sys.argv[1] == "-i":
        '''
        List the G-Code firmware parameters.
        '''
        if len(sys.argv) <= 2:
            print("Some required parameter is missing")
        else:
            gcode_driver = gcode.DriverGCode(sys.argv[2])
            gcode_driver.start_serial()
            gcode_driver.paramsGRBL()
            gcode_driver.quit()

    elif sys.argv[1] == "-m":
        '''
        Run the mandala algorithm.
        '''
        # Note that here the radius is forced to a
        single value
        # to generate a single mandala.
        # TODO Set runtime parameters to the function.
        fig2 = randomizer.random_mandala(n_rows=None,
                                         n_columns=None,
                                         # radius=[8, 6, 3],
                                         radius=7,
                                         rotational_
                                         symmetry_order=9,
                                         symmetric_
                                         seed=True,
                                         number_of_
                                         elements=5,
                                         face_color="0.")
```

```
            fig2.tight_layout()
            mandala_image = [mandala.figure_to_image(fig2)]

            for im in mandala_image:
                im.show()

        else:
            ''' No command specified '''
            print("Command not specified\n")
            help()
```

12.3.3. Extra Functions

For better readability, the program also includes three helper functions, defined separately to keep the source that processes the command-line options smaller. These three functions are shown in Listings 12-5 through 12-7.

Listing 12-5. The list_ports() Function

```
def list_ports():
    '''
    List all the available com ports. To use one of the listed
    ports refer to it with the port number (first column index)
    '''
    port_idx = 0
    # Retrieve the list of available COM ports
    serial_ports = my_serial.available_ports()

    print(" --- Available com ports")
    for port in serial_ports:
        print("#", port_idx, port.name)
        port_idx += 1
    print (" ---\n")
```

Listing 12-6. The help() Function

```
def help():
    '''
    Show the help message to the terminal.
    '''
    print("----------------- Help ------------------")
    print("Available options:\n")
    print("-l List all the available com ports")
    print("-p [port ID] Show the name of the port")
    print("-i [port ID] Show the GRBL firmware parameters")
    print("-x [port ID] Initializes the GRBL firmware
    parameters (persistent)")
    print("-c [port ID] [Command] Run one of the predefined
    commands to the machine")
    print("-m [port ID] Run the mandala algorithm")
    print("-s [port ID] [Design] Executes one of the predefined
    designs:\nspiral, epicycloid, cardioid, nephroid,
    ranunculoid")
    print("-h This help")
    print("----------------- Help ------------------")
```

Listing 12-7. The sand_plot() Function

```
def sand_plot(name:str, portId:int):
    '''
    Plot on the sand a predefined circular function, connected
    to the portID serial port.
    '''
    sand_plot = circfunc.MathCircularFunctions(portId)

    if name == 'spiral':
        '''
```

```
        Plot an Archimedes' sprial
        '''
        sand_plot.spiral(12, 12)

    elif name == 'epicycloid':
        '''
        Plot an Epicycloid
        '''
        sand_plot.epicycloid()

    elif name == 'cardioid':
        '''
        Plot a single cuspid cardioid
        '''
        sand_plot.cardioid()

    elif name == 'nephroid':
        '''
        Plot a double cuspid nephroid
        '''
        sand_plot.nephroid()

    elif name == 'ranunculoid':
        '''
        Plot a five cuspids ranunculoid
        '''
        sand_plot.ranunculoid()
```

12.4. Class: SerialControl

The `serialControl` class covers two essential roles: low-level communication between the application and the Sand Machine hardware and the inspection and configuration of the communication port.

In addition, when the program instantiates the class, it creates an internal instance of the `logger` class. According to the *log level,* a log file is generated when an event occurs.

The serial class is content-independent: with its low-level and high-level methods, it can be used in any USB-to-Serial connection to send and receive data in several formats (lines, strings, bytes, etc.).

As shown in these `imports`, I used the Python `dataclasses` library to manage the serial port configuration more efficiently:

```
from dataclasses import dataclass
import pyclasses.logger as logger
import serial
from serial.tools import list_ports
```

The `__init__` function in Listing 12-8 is called automatically when the program instantiates the class. This initialization function collects all the information on the system concerning the USB-to-Serial ports. This makes it possible to use the instance of the class all around the program—especially by the other higher-level classes—using the command-line parameters passed to the main program.

Listing 12-8. The __init__ Function

```
class SerialControl:
    '''
    RS232 Serial communication methods.
    '''

    def __init__(self, logLevel = logger.LogLevel.CRITICAL):
        '''
        Class constructor. Load the list of all available
        com ports.
```

```
Note that the class initialization function does not
assign to the data sets the port name.
It will be added further according to the user
parameters.
'''

global ports
''' List of the available com ports '''
global protocol_log
''' Logger '''
global serial_settings
''' Serial configuration object '''

ports = list_ports.comports() # Retrieve the posts list
serial_settings = SerialParameters(115200, serial.
STOPBITS_ONE, serial.PARITY_NONE)
'''
Create the serial configuration object with default
settings.
Default values can be redefined by the config method.
'''

protocol_log = logger.Logger(
            'Serial event log',
            logger.LogLevel.DEBUG,
            'protocol.log',
            True)
''' Create the logger object'''

for level in logger.LogLevel:
    if level.value == logLevel.value:
        protocol_log.initLog(logger.LogLevel.INFO)
        ''' Initialize the logger level '''
```

12.4.1. Low-level Methods

- wait_bytes_UTF8(self, stripCRLF = False)

- wait_bytes(self)

- read_bytes(self, len = 1)

- read_line(self)

- print_multiline(self)

This group of methods controls the low-level communication to the connected hardware.

```
def wait_bytes_UTF8(self, stripCRLF = False):
    '''
    Receive the bytes available on the serial port.
    This method waits for the serial port available data.

    Parameter: stripCRLF if set to True, all the CR, LF and
    TAB characters are stripped from the byte array. This
    is a convenient flag to create a clean string by the
    callers.

    Data read are returned in a byte array.
    '''

    global ser_port
    ''' The serial connection '''

    bytes_block = bytearray('', encoding='utf-8')
    ''' Data bytes from the serial '''

    end_block = False
    ''' Reading block exit flag '''

    while end_block is False:
        while ser_port.in_waiting:
```

```
            ''' Read a single byte '''
            bytes_block.extend(self.read_bytes())

        if len(bytes_block) is not int(0):
            end_block = True

    if stripCRLF is True:
        bytes_block.strip(b'\t')
        bytes_block.strip(b'\r')
        bytes_block.strip(b'\n')

    return bytes_block

def wait_bytes(self):
    '''
    Receive the bytes available on the serial port.
    This method waits for the serial port available data.
    Data read are returned in a byte array.
    '''

    global ser_port
    ''' The serial connection '''

    bytes_block = bytearray()
    ''' Data bytes from the serial '''

    end_block = False
    ''' Reading block exit flag '''

    while end_block is False:
        while ser_port.in_waiting:
            ''' Read a single byte '''
            bytes_block.append(self.read_bytes())

        end_block = True

    return bytes_block
```

```
    def read_bytes(self, len = 1):
        '''
        Read a predefined number of bytes from the serial port.
        If none is specified, read a single byte
        '''

        global ser_port
        ''' The serial connection '''

        return ser_port.read(len)

    def read_line(self):
        '''
        Receive a line from the serial port. Expect the string
        ending *CR or CR+LF)
        '''

        global ser_port
        ''' The serial connection '''

        return ser_port.readline().decode("utf-8")

    def print_multiline(self):
        '''
        Experimental.
        Print multiple lines coming from the serial
        '''
        global ser_port
        ''' The serial connection '''

        while ser_port.in_waiting == 0:
            ''' Just wait for the buffer not empty '''

        while ser_port.in_waiting:
            print(ser_port.readline().decode("utf-8"))
```

12.4.2. The Dataclass Data Model

Using the Python decorator `@dataclass`—of course, it is necessary to import the `dataclass` library—you can create a regular Python class, including a data model. As in C++ any struct is a class; also, in Python, using this mechanism, it is possible to define a class behaving as a data structure, including custom methods. The data model class is a complex data structure that stores the serial port configuration. See Listing 12-9.

A complete definition of the `dataclass` and the implicit data model methods is available in the Python 3 documentation at https://docs.python.org/3/library/dataclasses.html.

Listing 12-9. The Data Model Class

```
@dataclass
class SerialParameters:
    '''
    Define a data class including the serial connection
    parameters.
    This class acts as a data structure where every
    communication configuration can be set in a new class
    instance.

    Parameters:\n
        port - Device name or None.
        baudrate - Baud rate such as 9600 or 115200 etc.
        bytesize - Number of data bits. Possible values:
        FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS\n
        parity - Enable parity checking. Possible values:
        PARITY_NONE, PARITY_EVEN, PARITY_ODD PARITY_MARK,
        PARITY_SPACE\n
        stopbits - Number of stop bits. Possible values:
        STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO\n
```

```
        tout - Data waiting from serial timeout,
        default 500 ms.
    '''

    def __init__(self, baud: int, stop: bytes, parity: bytes,
    port: str = None, dBits: int = 8, timeout = 2):
        '''
        Initializes the dataclass with the parameters and some
        default settings.
        The port name will be assigned separately after the
        initialization class.
        '''
        self.port = port
        self.baudrate = baud
        self.parity = parity
        self.stopbits = stop
        self.databits = dBits
        self.timeout = timeout

    port: str
    baudrate: int
    databits: int
    parity: bytes
    stopbits: bytes
    timeout:float
```

12.4.3. High-level Methods

Using the `SerialParameters` data model, high-level methods can manage the parameters of the serial port configuration as a single object. See Listing 12-10.

Listing 12-10. These Hardware Control Methods Work the USB-to-Serial Interface in Several Kinds of Data Transfers

```
    def create_name(self, port_id):
        '''
        Create the serial port name and assign it to
        the dataset
        Return true if the serial port exists, else
        return false.
        '''
        global ports
        ''' Ports list '''
        global protocol_log
        ''' Logger '''
        global serial_settings
        ''' Serial configuration object '''

        if (len(ports) - 1) >= int(port_id):
            '''
            If the port ID exists, assign the port name to the
            two serial settings data classes
            '''
            s_port = ports[int(port_id)].device
            serial_settings.port = s_port
            return True
        else:
            ''' Wrong port ID '''
            return False

    def close(self):
        '''
        Close the serial port.
        '''
```

```
        global ser_port
        ''' Serial port instance '''

        ser_port.flush()
        ser_port.close()

    def open(self):
        '''
        Open the serial port with the current configuration
        settings and wait for bytes response data, if any.
        The method returns that bytes available in the queue
        after the port has been opened.
        '''
        global ser_port
        ''' Serial port instance '''

        ser_port.open()

        return self.wait_bytes()

    def open_terminal(self, stripCRLF = False):
        '''
        Open the serial port for a communication terminal. It
        is expected that the data are UTF-8 Ascii strings.
        The method returns the string available in the queue
        after the port has been opened.
        The CR/LF/TAB characters are removed, according to the
        flag stripCRLF
        '''
        global ser_port
        ''' Serial port instance '''

        ser_port.open()
```

```
        return self.wait_string(stripCRLF)

    def config(self):
        '''
        Initialize the serial port and open the port.
        '''

        global serial_settings
        ''' Predefined serial settings '''
        global ser_port
        ''' Serial port instance '''

        ser_port = serial.Serial()
        ''' Get the serial object instance. '''

        ''' Configure the serial '''
        ser_port.baudrate = serial_settings.baudrate
        ser_port.bytesize = serial_settings.databits
        ser_port.stopbits = serial_settings.stopbits
        ser_port.parity = serial_settings.parity
        ser_port.port = serial_settings.port

    def send(self, string_data:str):
        '''
        Send a string to the serial port and return the number
        of bytes sent.

        The function returns the number of bytes written.
        '''
        global ser_port
        ''' Serial port instance '''

        return ser_port.write(string_data)

    def send_line(self, string_data:str):
        '''
```

```
        Send a text line to the serial port and return the
        number of bytes sent.

        The function returns the number of bytes written.
        '''
        global ser_port
        ''' Serial port instance '''

        sending = str(string_data).encode('utf-8') + "\n".
        encode('utf-8')

        return ser_port.write(sending)

    def get_port(self):
        '''
        Return the selected port name, or None.
        '''
        global serial_settings
        ''' Serial configuration object '''

        return serial_settings.port

    def available_ports(self):
        '''
        Return a list of the available COM ports in the system.
        '''
        global ports
        ''' Serial ports list '''

        ports = list_ports.comports()
        return ports

    def config(self, speed=9600, data=8, stop=1, parity=serial.
    PARITY_NONE, timeout = 0.25):
        '''
```

```
    Set up the serial port configuration parameters. The
    settings can be used for the
    serial port opening and configuration.
    '''
    global serial_settings
    ''' Serial configuration object '''

    serial_settings = SerialParameters(speed, stop, parity,
    data, timeout)
```

12.5. Class: Logger

The Logger class is an independent piece of code that can be used anywhere in a Python program.

This class implements the Python logging library, and an enum defines five log levels. See Listing 12-11.

Listing 12-11. The Logger Class

```
class LogLevel(Enum):
    '''
    Defines the various log levels. Used by the logger classes
    and by the calling application to set the loglevel.
    '''
    DEBUG = 1
    INFO = 2
    WARNING = 3
    ERROR = 4
    CRITICAL = 5
    def __init__(self,
        loggerName = "Generic logger",
```

```
        logLevel = LogLevel.CRITICAL,
        logFile = "logfile.log",
        outFile = True):
        '''
        Class constructor. Initializes the logger with the
        lowest verbosity.
        '''
        global logger
        ''' Logger instance '''
        global logFileName
        ''' Logger file name '''
        global logToFile
        ''' File logger flag '''

        logToFile = outFile
        logger = logging.getLogger(loggerName)
        ''' Console logger handler '''
        logFileName = logFile

        for level in LogLevel:
            if level.value == logLevel.value:
                logger.setLevel(level.value)
```

By default (if no `logLevel` parameter is passed), the global log level is set to `critical`: All the less-than-critical log messages are automatically disabled.

The global log level can be set only when instantiating the class; the caller program, instead, will set the log level to the console handler before assigning it to the logger. It is strongly suggested that you set the highest logging level if the highest verbosity log file is desired.

When instantiated, this class creates the console and the file logger. If no log file is specified, the name of the log file is `logfile.log`. The same log level is assumed to be applied to the console and to the file handler (if set).

For more details on how the Python logging library works, see the documentation at https://docs.python.org/2/howto/logging-cookbook.html#multiple-handlers-and-formatters.

For every log level, a simple method accepting an input message to be sent to the log file is available. The `logger` adds the time stamp when the message is sent to the terminal or written to the log file:

```
def debug(self, message):
    global logger
    logger.debug(message)

def info(self, message):
    global logger
    logger.info(message)

def warning(self, message):
    global logger
    logger.warning(message)

def error(self, message):
    global logger
    logger.error(message)

def critical(self, message):
    global logger
    logger.critical(message)
```

12.6. Class: driverGCode

The `driverGCode` class defines all the methods to control the Sand Machine behavior through macro commands. Some class methods are designed specifically for the Sand Machine, but these can easily be converted to generic macro-commands to control a three-axis CNC engine.

Like the `SerialControl` class, the `driverGCode` also implements the logger feature for debugging and keeping track of every method's functionality.

The class expects a serial communication protocol controlled by the `SerialControl` class: it is sufficient to know the serial port ID; the low-level hardware communication is instantiated externally.

The `__init__` class initialization function starts the logging library, instantiates the `serialControl` class, and loads the JSON configuration files in dictionaries used by the class methods (see Listing 12-12).

Listing 12-12. The __init__ class Initialization Function

```
def __init__(self, comm_port, logLevel = logger.
LogLevel.CRITICAL):
'''
Class constructor. Initializes the dictionaries and
the logger.

Parameters:
comm_port : Communication port ID
Log level (*)default CRITICAL)
'''

global serial_comm
''' Serial communication class instance '''
global GCODE_preset
```

```
''' Predefined G-Code command strings dictionary. '''
global GRBL_settings
''' Dictionary for GRBL (firmware configuration)
initialization. '''
global GRBL_info
''' Firmware information commands dictionary '''
global protocol_log
''' Logger instance '''
global sand_plot_X;
''' Keep history of the x-axis coordinates to return to
home '''
global sand_plot_Y;
''' Keep history of the y-axis coordinates to return to
home '''

with open("GRBLcontrol.json") as file:
    GRBL_info = json.load(file)

with open("GRBLsettings.json") as file:
    GRBL_settings = json.load(file)

with open("GCODE.json") as file:
    GCODE_preset = json.load(file)

''' Serial communication initialization '''
serial_comm = comm.SerialControl()
serial_comm.create_name(comm_port)
serial_comm.config()

''' Configure the serial and open the connection '''

protocol_log = logger.Logger(
            'G-Code Event log',
            logger.LogLevel.DEBUG,
```

```
                'g-code.log',
                True)
        ''' Create the logger object'''

        for level in logger.LogLevel:
            if level.value == logLevel.value:
                protocol_log.initLog(logger.LogLevel.INFO)
                ''' Initialize the logger level '''

        sand_plot_X = 0
        sand_plot_Y = 0
```

The paramsGRBL(), GRBL_set_param(), and GCODE_exec() methods interface with the hardware, executing the firmware initialization and sending custom G-Code strings to the device. See Listing 12-13.

Listing 12-13. The paramsGRBL(), GRBL_set_param(), and GCODE_exec() Methods

```
def paramsGRBL(self):
        '''
        List all the GRBL firmware parameters
        '''
        global GRBL_info
        ''' GRBL Control commands dictionary '''
        global serial_comm
        ''' The pre-configured serial port '''

        str(serial_comm.send_line(GRBL_info["Settings"]))
        serial_comm.print_multiline()

    def GRBL_set_param(self, param_name:str):
        '''
        Decode the GRBL firmware parameter name from the
        dictionary and send the corresponding command to
        the serial.
```

```
        '''
        global serial_comm
        ''' The pre-configured serial port '''
        global GRBL_settings
        ''' Dictionary for GRBL (firmware configuration)
        initialization. '''

        print(param_name + " bytes sent: " + str(serial_comm.
        send_line(GRBL_settings[param_name])))
        r = serial_comm.wait_string(False)
        print("Response :" + r)
```

The soft_homing() macro command repositions the magnetic head to the home position (see Listing 12-14). Due to the particular kind of drawing and the shape of the Sand Machine's working surface, the home of the axes is the center of the circular area.

Listing 12-14. The soft_homing() Macro Command

```
def soft_homing(self, sand_home_x = 200, sand_home_y = 200,
AxisXCorrectionFactor = 30, AxisYCorrectionFactor = 10):
        '''
        Reposition the x-y axis to the sand home position
        (center of the dome)
        '''
        global serial_comm
        ''' The pre-configured serial port '''
        global sand_plot_X
        global sand_plot_Y

        print("sand_home() -sand_plot_X ", -sand_plot_X, "
        -sand_plot_Y ", -sand_plot_Y)
```

```
        commmand_string = "G00 X " + str(-sand_plot_X + sand_
        home_x + AxisXCorrectionFactor) + " Y " + str(-sand_
        plot_Y + sand_home_y + AxisYCorrectionFactor)

        self.GCODE_exec(commmand_string)
```

The `init_magnet()` macro command positions the working head to the lower z-axis position, goes to the center of the working area, and positions the head to the upper position, nearest to the iron sphere. See Listing 12-15. After this initialization procedure is complete, the program can start the mandala design.

Listing 12-15. The init_magnet() Macro Command

```
def init_magnet(self):
        '''
        Initializes the axis in the right position
        '''
        global GCODE_preset
        ''' Dictionary with the preset commands '''

        # Initializes the GCODE
        self.GCODE_exec(GCODE_preset["init"])
        self.GCODE_exec(GCODE_preset["units"])
        # Set magnet far from top
        self.GCODE_exec(GCODE_preset["zdown"])
        # Move to the work point
        self.GCODE_exec(GCODE_preset["sand"])
        # Move magnet up to the work point
        self.GCODE_exec(GCODE_preset["zup"])
```

The `run_command()` macro command is a helper method that executes the G-Code sequence from one of the GCODE JSON files in the dictionary. See Listing 12-16.

Listing 12-16. The _command() Macro Command

```
def run_command(self, cmd:str):
        '''
        Executes one of the predefined commands in the GCODE
        Json dictionary.
        '''
        global GCODE_preset
        ''' Dictionary with the preset commands '''

        self.GCODE_exec(GCODE_preset[cmd])
```

12.7. Class: MathCircularFunctions

This class defines the mathematical methods to generate circular objects on the sand surface, according to the theory behind the mathematics of the mandala.

The class methods only focus on the x-y 2D coordinates system; in fact, after the magnetic head of the CNC frame has linked the iron sphere (initially positioned in the center of the Sand Machine), it moves the ball following a single path.

The constraints and global parameters refer to the physical coordinates dimensions of the surface, while the z-axis movement is called when the drawing cycle starts and ends.

The two helper methods—`p2c()` and `distance2D()`—are used to calculate the motion steps and to convert the polar coordinates of every point to cartesian coordinates to drive the x-y axes. See Listing 12-17.

Listing 12-17. Two Helper Methods—p2c() and distance2D()

```
    def p2c(self, r, phi):
        '''
        Convert polar to cartesian coordinates. To reach the
        right absolute position on the virtual cartesian
        coordinates system where the axes origin is in the
        center of the sand arena, the calculated absolute
        coordinates are added to the real origin. Doing so, the
        return value is ready for plotting.
        '''
        return int(r * math.cos(phi)) + self.xArenaCenter,
        int((r * math.sin(phi)) + self.yArenaCenter)

    def distance2D(self, po, pd):
        '''
        Calculate the distance between the two points
        po and pd.
        Both points are a bidimensional array with the point
        coordinates in the format [x, y]
        '''
        return math.dist(po, pd)
```

12.7.1. The Mandala Curve Methods

For every kind of mandala design available—spiral, epicycloid, cardioid, nephroid, and ranunculoid—there is a corresponding method that runs a loop following the design path. See Listing 12-18.

Listing 12-18. The Five Methods of Developing the Mathematics of the Mandala Figures Converted Into G-Code Steps

```
    def spiral(self, arc, separation):
        '''
        Plot on the sand an Archimedes' spiral based on the
        equation r = b * phi

        The system should start from the initial setup of the
        x-y axes.
        The center (starting) point is hardcoded.

        arc: Arc length between two points
        separation: Distance between consecutive turnings
        '''
        gcode_driver = gcode.DriverGCode(self.serialID)
        gcode_driver.start_serial()
        gcode_driver.initGRBL()
        gcode_driver.run_command("sand")
        gcode_driver.run_command("zup")

        r = arc
        ''' r is the radius r in the original equation '''
        b = separation / (2 * math.pi)
        '''' b is the "b" constant value in the original
        equation'''

        # find the first phi to satisfy distance of `arc` to
          the second point
        phi = float(r) / b

        xCoord = self.xArenaCenter
        yCoord = self.yArenaCenter

        # Plot the figure
```

```
        while   (
                (xCoord <= (self.xArenaCenter + self.
                xArenaSize)) and
                (xCoord >= (self.xArenaCenter - self.
                xArenaSize)) and
                (yCoord <= (self.yArenaCenter + self.
                yArenaSize)) and
                (yCoord >= (self.yArenaCenter - self.
                yArenaSize))
                ):
            '''
            Calculate a new couple of coordinates
            '''
            xCoord, yCoord = self.p2c(r, phi)
            # print(str(xCoord), str(yCoord))
            gcode_driver.GCODE_exec("G01 X " + str(xCoord) +
                                    " Y " + str(yCoord) +
                                    " F" + str(self.feedRate))
            # Updated the homing coordinates
            gcode_driver.update_axes_
            coordinates(xCoord, yCoord)

            #Increment the variables
            phi = phi + float(arc) / r
            r = b * phi

        # Close the connection and exit
        gcode_driver.run_command("zdown")
        gcode_driver.soft_homing(self.xArenaCenter, self.
        yArenaCenter)
        # gcode_driver.run_command("home")
        gcode_driver.quit()
```

```
def epicycloid(self, radius = 50, fixed_circle_
radius = 10):
    '''
    Draw a parametric circular epicycloid.

    ref: https://mse.redwoods.edu/darnold/math50c/CalcProj/
    Sp99/LindaL/epicycloid.html

    For better understanding, the control parameters
    and the variable names inside the functions are
    referred to the above documentation link.
    '''
    # Initialize the connection and GCode driver
    gcode_driver = gcode.DriverGCode(self.serialID)
    gcode_driver.start_serial()
    gcode_driver.initGRBL()
    gcode_driver.run_command("sand")
    gcode_driver.run_command("zup")

    b = radius
    ''' The point P moves along the radius b. '''
    t = 10
    ''' And is translated every step by the angle t
    (deg) '''
    a = fixed_circle_radius
    ''' The circumference the circle rotates along. '''

    x = self.xArenaCenter
    y = self.yArenaCenter

    translation_angle = 1
```

```
        while (translation_angle < 360):
            x = ( (a + b) * math.cos(translation_angle) ) + (b
            * math.cos( ( (a + b) / b) * translation_angle ) )
            + self.xArenaCenter
            '''
            A more readable version of the x formula:
            x = (a + b) cos t + b cos {[(a + b)/b] t}
            '''
            y = ( (a + b) * math.sin(translation_angle) )
            + (b * math.sin( ( (a + b) / b) * translation_
            angle) ) + self.xArenaCenter
            '''
            A more readable version of the y formula:
            y = (a + b) sin t +  b sin {[(a + b)/b] t}
            '''

            gcode_driver.GCODE_exec("G01 X " + str(x) +
                                    " Y " + str(y) +
                                    " F" + str(self.feedRate))
            # Updated the homing coordinates
            gcode_driver.update_axes_coordinates(x, y)

            translation_angle = translation_angle + t

            print("x ", x, " y ", y, " t ", translation_angle)

        # Close the connection and exit
        gcode_driver.run_command("zdown")
        gcode_driver.soft_homing(self.xArenaCenter, self.
        yArenaCenter)
        gcode_driver.run_command("home")
        gcode_driver.quit()
```

```
def cardioid(self, radius = 30, fixed_circle_radius = 30):
    '''
    Draw a parametric circular cardioid.

    ref: https://mse.redwoods.edu/darnold/math50c/CalcProj/
    Sp99/LindaL/epicycloid.html

    For better understanding, the control parameters and
    the variable names inside the functions are referred
    to in the
    above documentation link.
    '''
    # Initialize the connection and G-Code driver
    gcode_driver = gcode.DriverGCode(self.serialID)
    gcode_driver.start_serial()
    gcode_driver.initGRBL()
    gcode_driver.run_command("sand")
    gcode_driver.run_command("zup")

    b = radius
    ''' The point P moves along the radius b. '''
    t = 10
    ''' And is translated every step by the angle t
    (deg) '''
    a = fixed_circle_radius
    ''' The circumference the circle rotates along. '''

    x = self.xArenaCenter
    y = self.yArenaCenter

    translation_angle = 1
```

```
        while (translation_angle < 360):
            x = ( (a + b) * math.cos(translation_angle) ) + (b
            * math.cos( ( (a + b) / b) * translation_angle ) )
            + self.xArenaCenter
            '''
            A more readable version of the x formula:
            x = (a + b) cos t + b cos {[(a + b)/b] t}
            '''
            y = ( (a + b) * math.sin(translation_angle) )
            + (b * math.sin( ( (a + b) / b) * translation_
            angle) ) + self.xArenaCenter
            '''
            A more readable version of the y formula:
            y = (a + b) sin t +  b sin {[(a + b)/b] t}
            '''

            gcode_driver.GCODE_exec("G01 X " + str(x) +
                                    " Y " + str(y) +
                                    " F" + str(self.feedRate))
            # Update the homing coordinates
            gcode_driver.update_axes_coordinates(x, y)

            translation_angle = translation_angle + t


            print("x ", x, " y ", y, " t ", translation_angle)

        # Close the connection and exit
        gcode_driver.run_command("zdown")
        gcode_driver.soft_homing(self.xArenaCenter, self.
        yArenaCenter)
        gcode_driver.run_command("home")
        gcode_driver.quit()
```

```
def nephroid(self, radius = 20, fixed_circle_radius = 40):
    '''
    Draw a parametric circular cardioid.

    ref: https://mse.redwoods.edu/darnold/math50c/CalcProj/
    Sp99/LindaL/epicycloid.html

    For better understanding, the control parameters and
    the variable names inside the functions are referred
    to in the
    above documentation link.
    '''
    # Initialize the connection and G-Code driver
    gcode_driver = gcode.DriverGCode(self.serialID)
    gcode_driver.start_serial()
    gcode_driver.initGRBL()
    gcode_driver.run_command("sand")
    gcode_driver.run_command("zup")

    b = radius
    ''' The point P moves along the radius b. '''
    t = 10
    ''' And is translated every step by the angle t
    (deg) '''
    a = fixed_circle_radius
    ''' The circumference the circle rotates along. '''

    x = self.xArenaCenter
    y = self.yArenaCenter

    translation_angle = 1
```

```
        while (translation_angle < 360):
            x = ( (a + b) * math.cos(translation_angle) )
            + (b * math.cos( ( (a + b) / b) * translation_
            angle ) )  + self.xArenaCenter
            '''
            A more readable version of the x formula:
            x = (a + b) cos t + b cos {[(a + b)/b] t}
            '''
            y = ( (a + b) * math.sin(translation_angle) ) + (b
            * math.sin( ( (a + b) / b) * translation_angle) ) +
            self.xArenaCenter
            '''
            A more readable version of the y formula:
            y = (a + b) sin t +  b sin {[(a + b)/b] t}
            '''

            gcode_driver.GCODE_exec("G01 X " + str(x) +
                                    " Y " + str(y) +
                                    " F" + str(self.feedRate))
            # Updated the homing coordinates
            gcode_driver.update_axes_coordinates(x, y)

            translation_angle = translation_angle + t

            print("x ", x, " y ", y, " t ", translation_angle)

        # Close the connection and exit
        gcode_driver.run_command("zdown")
        gcode_driver.soft_homing(self.xArenaCenter, self.
        yArenaCenter)
        gcode_driver.run_command("home")
        gcode_driver.quit()
```

```
def ranunculoid(self, radius = 10, fixed_circle_
radius = 50):
    '''
    Draw a parametric circular cardioid.

    ref: https://mse.redwoods.edu/darnold/math50c/CalcProj/
    Sp99/LindaL/epicycloid.html

    For better understanding, the control parameters and
    the variable names inside the functions are referred to
    in the above documentation link.
    '''
    # Initialize the connection and GCode driver
    gcode_driver = gcode.DriverGCode(self.serialID)
    gcode_driver.start_serial()
    gcode_driver.initGRBL()
    gcode_driver.run_command("sand")
    gcode_driver.run_command("zup")

    b = radius
    ''' The point P moves along the radius b. '''
    t = 10
    ''' And is translated every step by the angle t
    (deg) '''
    a = fixed_circle_radius
    ''' The circumference the circle rotates along. '''

    x = self.xArenaCenter
    y = self.yArenaCenter

    translation_angle = 1
```

```
        while (translation_angle < 360):
            x = ( (a + b) * math.cos(translation_angle) )
            + (b * math.cos( ( (a + b) / b) * translation_
            angle ) )  + self.xArenaCenter
            '''
            A more readable version of the x formula:
            x = (a + b) cos t + b cos {[(a + b)/b] t}
            '''
            y = ( (a + b) * math.sin(translation_angle) )
            + (b * math.sin( ( (a + b) / b) * translation_
            angle) ) + self.xArenaCenter
            '''
            A more readable version of the y formula:
            y = (a + b) sin t +  b sin {[(a + b)/b] t}
            '''

            gcode_driver.GCODE_exec("G01 X " + str(x) +
                                    " Y " + str(y) +
                                    " F" + str(self.feedRate))
            # Updated the homing coordinates
            gcode_driver.update_axes_coordinates(x, y)

            translation_angle = translation_angle + t


            print("x ", x, " y ", y, " t ", translation_angle)

        # Close the connection and exit
        gcode_driver.run_command("zdown")
        gcode_driver.soft_homing(self.xArenaCenter, self.
        yArenaCenter)
        gcode_driver.run_command("home")
        gcode_driver.quit()
```

CHAPTER 13

Upcycling a Rotary Phone

Figure 13-1. *The vintage rotary phone that hosted this chapter's project, before the transformation*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_13

I can't imagine Ray's surprise when the appliance caught his eye; he hadn't such a device in at least 30 years.

Hello? Is there anybody *in there?*
Just nod if you can hear me. Is there anyone home?
*(Pink Floyd, "*Comfortably *Numb")*

He was expecting anyone to answer from the rotary phone—he randomly composed a three-cipher number—and surprisingly, the phone answered him. It took two or three minutes to realize he was speaking to anyone else but the phone.

When Ray mentioned this episode during one of our calls some months after his incredible adventure, the story sounded like a challenge: I want one, too; I want to make one, I thought! After some research and analysis, the "Rotary Phone" project became a reality.

Another fact from my past played a crucial role in designing this project. When I was a kid, the national phone company in Italy, for some years, provided a fairy tale service. You called a short number to hear a three-minute tale, which differed every day. I still remember the feel of these stories and won't soon forget.

13.1. Investigating the Parts

Starting from the idea of making an interactive new kind of device, before advancing any hypothesis of how to actually do it, I deeply analyzed the phone to determine which components were reusable for interfacing with a computer and which had to be removed.

I also decided that to achieve the features I had in mind in this upcycling project, using an Arduino board—or any other microcontroller—was not an option. I decided to move to a Raspberry Pi B3. After the experience of making this project, I suggest you consider this

a minimal requirement if you want to try a similar, challenging adventure. Of course, the next-generation models are also acceptable. The Pi3 B4 is a good-performing embedded Linux machine, and I have used it in many other projects.

After carefully disassembling the rotary phone, you must check the components, determine their role, and consider if you can reuse or remove them from the upcycled device:

- **Hang-up switch**: Detects when the caller picks up the phone receiver microphone; it can be converted to the power on/off switch.
- **Handset**: Used by the caller to hear the voice and speak in the microphone; perfect for hosting a couple of small speakers.
- **Rotary dialer**: This is the most interesting part. It can be converted into a numeric interface used as a numeric keyboard. The interfacing process was more challenging than expected.
- **Ringer**: Do we need to use it? As the upcycled device will include audio features, this is a part that can be removed. Removing the ring bell and the internal ring solenoid is the only way to make space inside the phone. See Figure 13-2.

Figure 13-2. *The two most important parts of the phone are the rotary dialer and the ring bell. The ring bell became redundant for the project and was removed with the solenoid circuit that excites the bell during an ongoing call*

13.1.1. Upcycling, Not Restoring

Do not confuse the meaning of upcycling with restoring. Restoring means regenerating an old device—sometimes one that's broken—and preserving its original features and aspects as much as possible.

Not to offend vintage restoration enthusiasts, but upcycling moves to the next step. This means transforming an appliance, preferably electronic, electric, or electromechanical, into something different and perhaps entirely innovative.

In this case, when added to an embedded Linux board and a bunch of discrete components (mostly resistors and capacitors), this analog rotary telephone becomes an interactive appliance with entirely new features. See Figure 13-3.

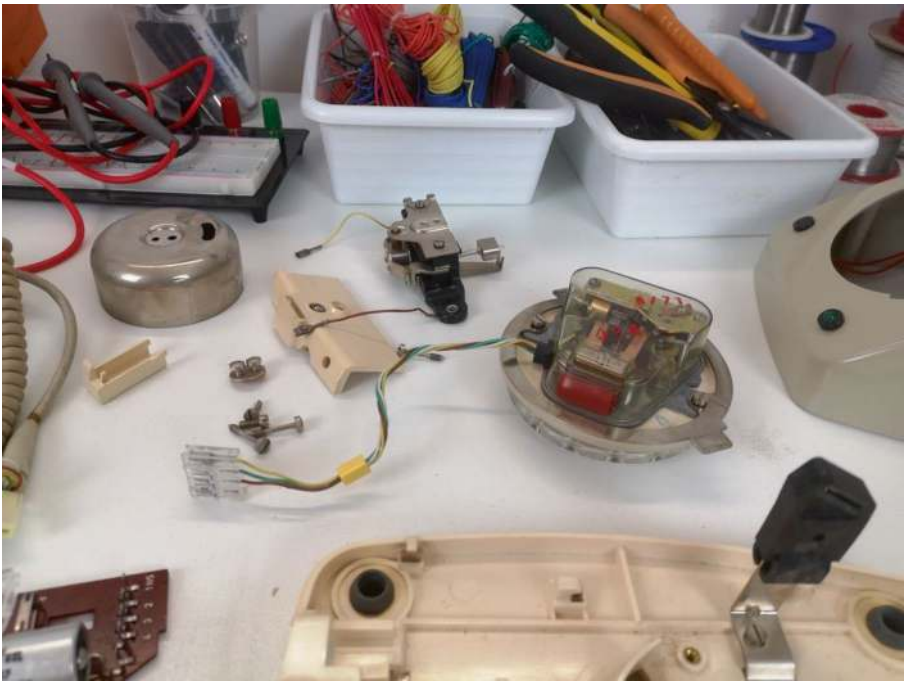

Figure 13-3. *I fully disassembled the telephone to understand the space distribution inside the device and how to use some parts. The rotary numbers generator is a relatively modern model, compact and closed inside a plastic box*

13.1.2. Removing the Ring Bell

I decided to remove the bell to free up space inside of the device. This means the ring must be controlled with digital low-voltage signals, which can be more complex than you imagine.

The bell rings by the vibration of a small metal hammer connected to a solenoid, and this circuit works at a relatively high voltage. On the telephone board, an analog circuit with a transformer, provides the needed voltage and power from the phone line cable.

It would be possible to reverse engineer this circuit to adapt it to 5V, but this solution risks creating a lot of extra circuitry for a task that's easier to solve in other ways.

The ringing circuitry components are soldered on a single-side thick PCB that also hosts the pick-up switch of the telephone. After desoldering these components and deriving a couple of wires in correspondence to the switch, I screwed the bare PCB inside the phone case. At this point, I had sufficient space inside the phone to fit the Raspberry Pi and other components required by the audio session. See Figure 13-4.

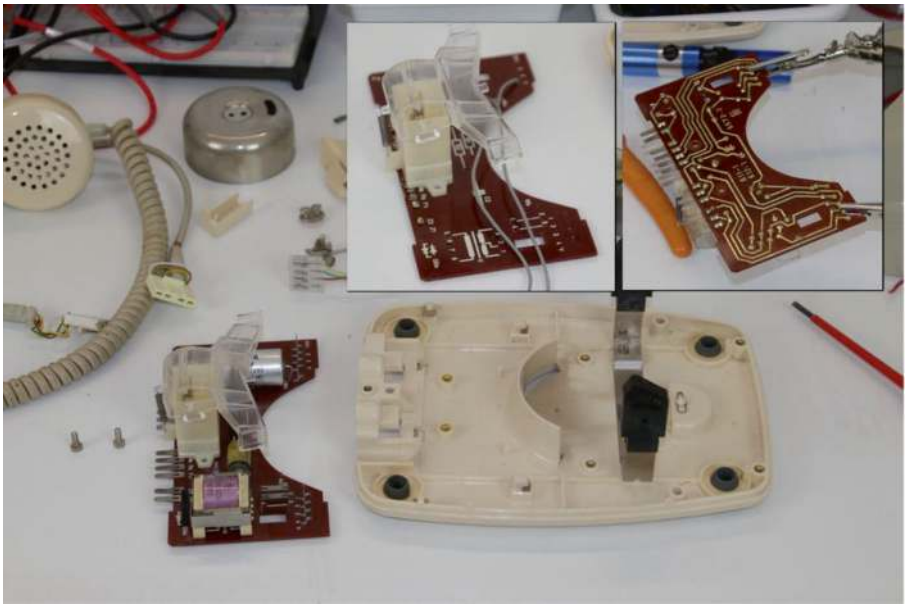

Figure 13-4. *After the phone was fully disassembled, the PCB components used for the ringer were desoldered. After removing the elements, the board, screwed to the phone base, is the support for the phone switch that will become the on/off switch of the project*

I also removed some plastic components to maximize the space inside the phone; these were finalized to support the ringer bell and the hammer.

While making this change, I also improved the air circulation. If you have experienced working with the Raspberry Pi, an essential factor to consider is providing an efficient cooling system, especially when the device is pushed to its maximum performance. See Figure 13-5.

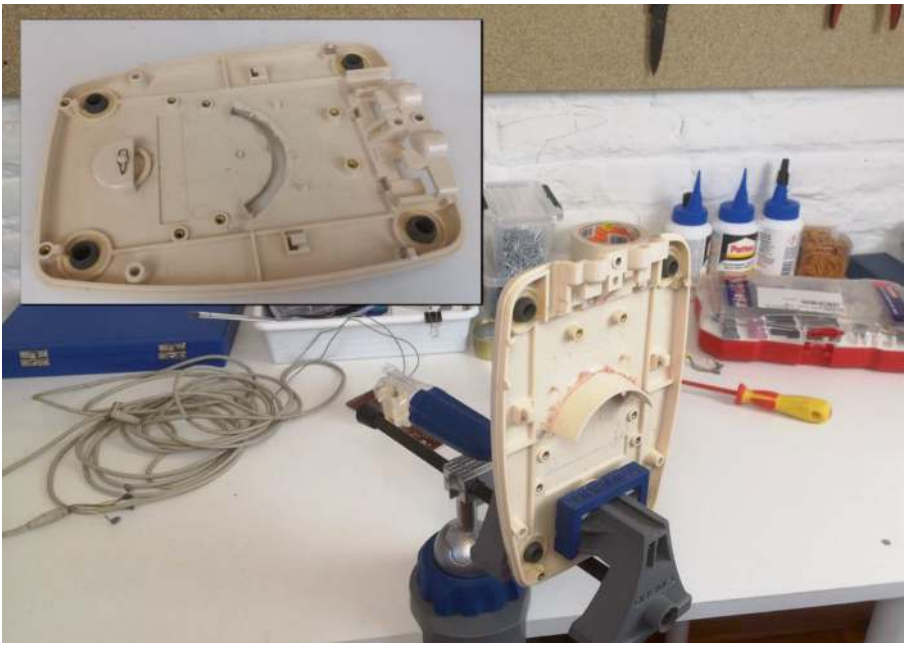

Figure 13-5. *To recover as much space as possible inside the phone, part of the plastic support of the ringer bell was cut. The curring has also been enlarged to improve the air circulation for better cooling of the Raspberry Pi*

13.2. The Rotary Dialer

Regardless of their external aspects, not all the rotary phones available on the market were identical. The common characteristic of these components is that they are essentially *pulse generators.* How they worked and provided the pulse counts associated with the numbers depended on the brand and the telephone line they were connected to. For example, the number of pulses differs between Europe and the United States.

Also, knowing the common general principle they are based on, I had to investigate how this specific model worked. Using a tester capable of producing a signal graph, I verified the functioning of the rotary dial when a digit is dialed. In my case, each digit on the rotary dial corresponds to an equal number of pulses—one pulse for 1, two for 2, and so on, except for zero, which generates ten pulses. In addition, the pulses are generated by an analog mechanism—nothing digital! See Figure 13-6.

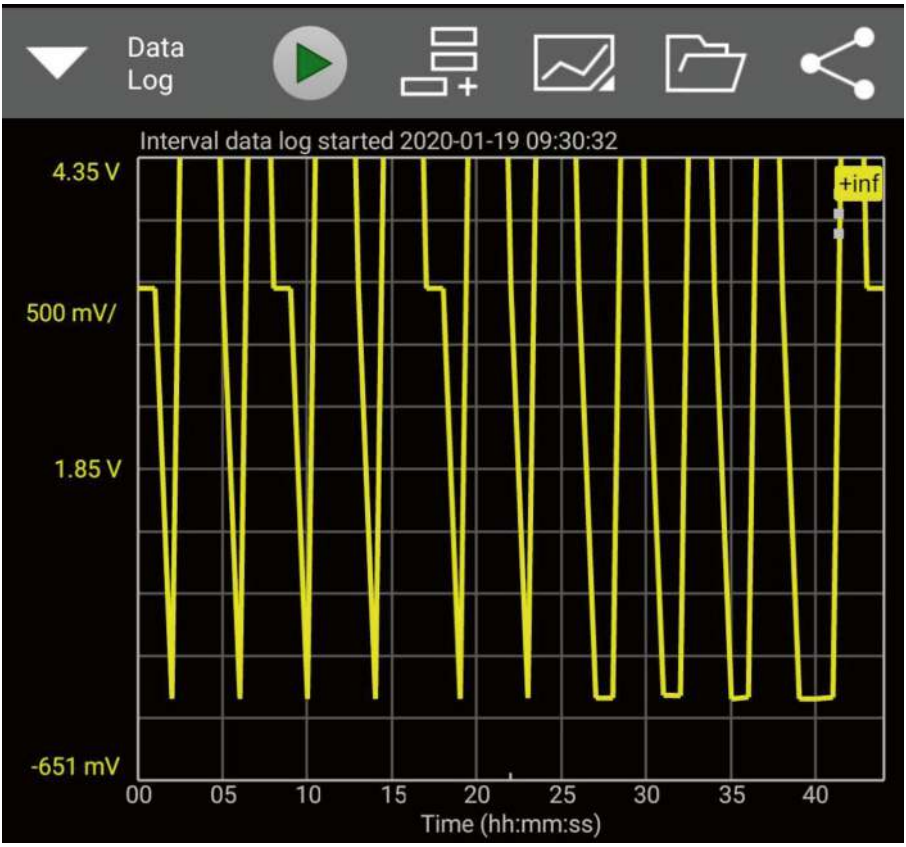

Figure 13-6. *Data log of a series of pulses from the rotary dial. As the pulse generator is electromechanical, the duration of the pulses is not precise but average. I count a pulse based on its minimal duration, and the systematic error should be a software concern*

The rotary dial has four wires. Two wires close a contact when the dial is rotated clockwise; the contact between these wires remains closed until the dial is released to return automatically to the original position through a circular spring by turning counterclockwise. The other two wires open and close a contact multiple times while the dial is released to dial a digit, producing a series of pulses corresponding to the dialed number—the digit 0 corresponds to ten pulses, 1 corresponds to one pulse, and so on.

To input the pulses and detect when the dial rotates clockwise—this is the "start" signal—for every couple of contacts, I connected one of the two wires to the 5Vcc to read the contact closing on the other wires.

13.3. Embedding Audio and Controls

Starting with the Raspberry Pi model 3, the platform improved its audio capabilities compared to the previous models, which used lower-quality PWM audio. Model 3 introduced a more powerful digital audio output, including an audio out 3mm stereo plug.

Hearing the audio directly by connecting a pair of earbuds to the audio out of the Raspberry Pi was not a very interesting solution. I decided to add an amplifier and a cheap Chinese portable Bluetooth amplified stereo speaker was perfect.

I got and fully disassembled one of these devices (you can find one in any bazaar for $10 or less) to reverse-engineer the control circuit and isolate the components out of the box. See Figure 13-7.

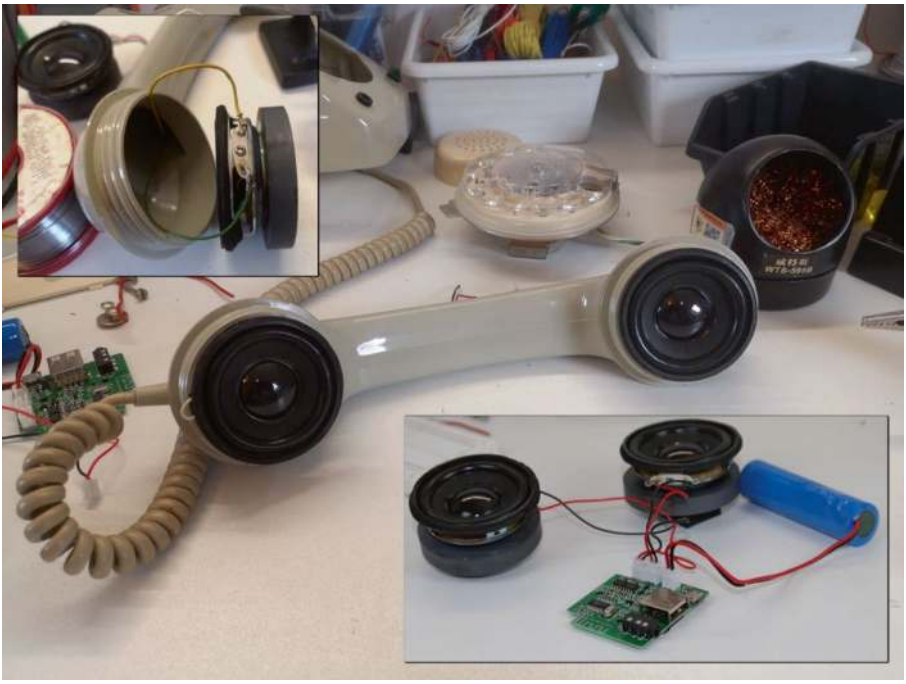

Figure 13-7. *The two speakers of the amplifier fit in the handset, soldered to the small battery-operated amplifier using the original handset cable*

The problem I faced was with the amplifier board. The device operates in three modes: Bluetooth, microSD, and line-in (which is what I needed) using a Setting pushbutton. The default operating mode when it is powered on—through a second pushbutton—is Bluetooth. The battery is charged by the on-board USB connector.

To make it usable, I needed to power on the board when the handset telephone switch was activated and power it off when the handset was in place, closing the communication. In addition, on every power-on cycle, the board should be set to the line-in mode by pressing the Setting button twice.

The Raspberry Pi's software logic should perform all these operations and a circuit is needed. See Figure 13-8.

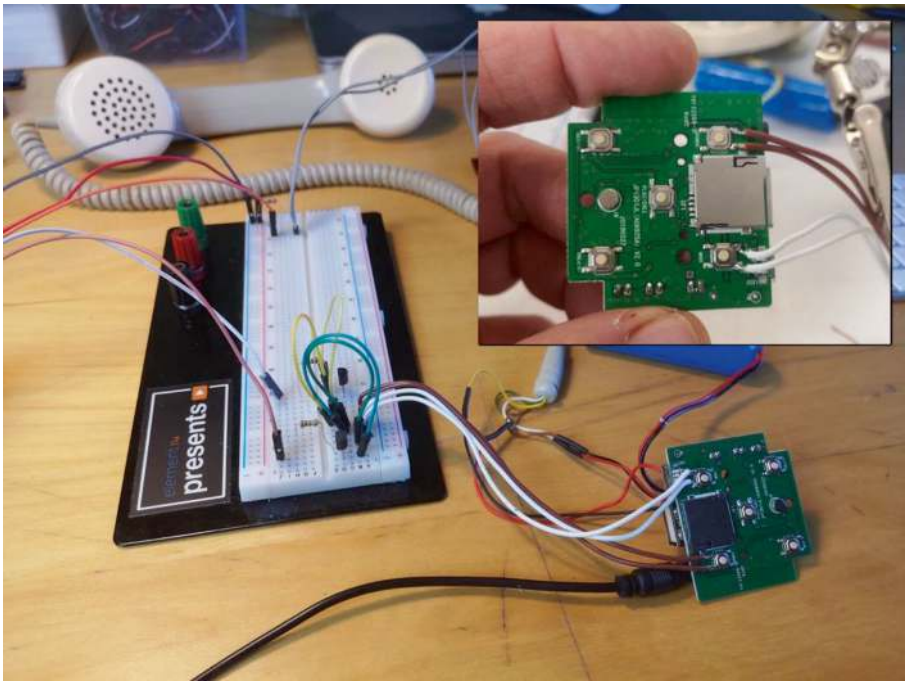

Figure 13-8. *The small amplifier board of the speakers. I soldered two couples of wires to the Power-on and Mode pushbuttons, controlled by the Raspberry Pi*

13.3.1. A Circuit to Control All

The Raspberry Pi exposes most of the hardware protocol ports (I2C, UART, PWM, etc.), bare GPIO pins, and the 3.3Vcc, 5Vcc, and GND signals through a 26-pin connector. The signals can be controlled via software, including Python, with the Pi I/O library. See Figure 13-9.

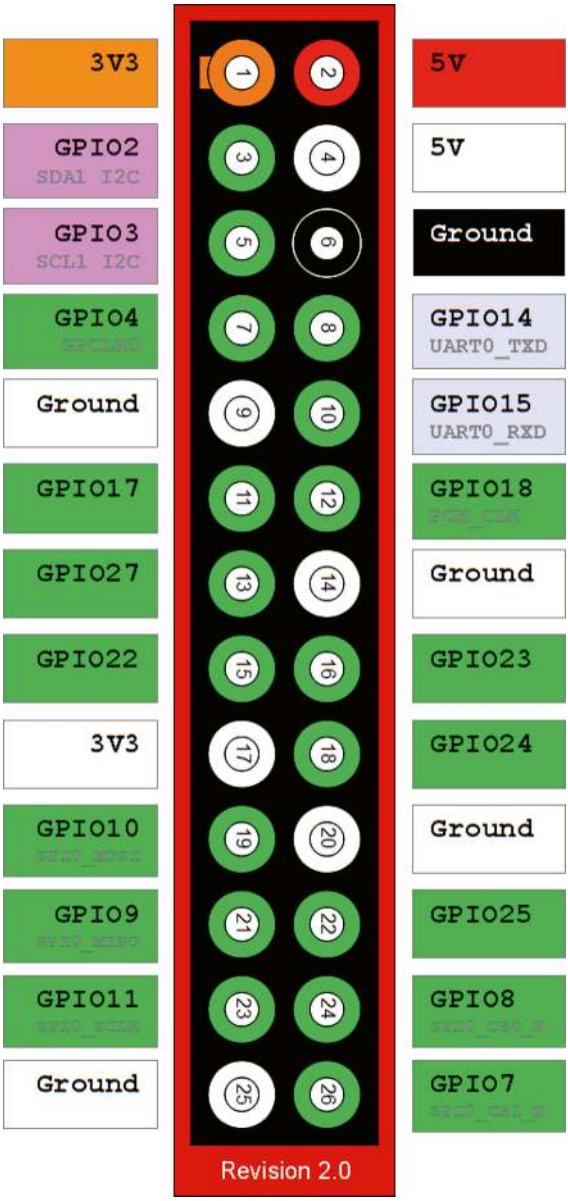

Figure 13-9. *The Raspberry Pi 3B GPIO connector with the signals assigned to every pin*

As mentioned, I planned to reuse the telephone components, as well as the control of the required circuitry, to interface them to the embedded Linux machine. Regardless of their original use, you can assume that after the transformation, all the events to detect and generate are just temporary switches.

Note The GPIO signals assigned to the Raspberry Pi can be subject to changes between models. For this reason, I always refer to the GPIO number instead of the connector PINs. To use a different Linux-embedded platform, you must check the connector's data sheet and the corresponding pin assignment.

Of course, the two pushbuttons of the amplifier (Power and Mode setting) are digital. To operate the buttons without pressing them, you can use a simple NPN transistor to close the circuit the same way as when the pushbuttons are pressed.

In all the other cases, you need to simulate a pushbutton as the contacts are passive; these are like special temporary switches without any electronics. See Figure 13-10.

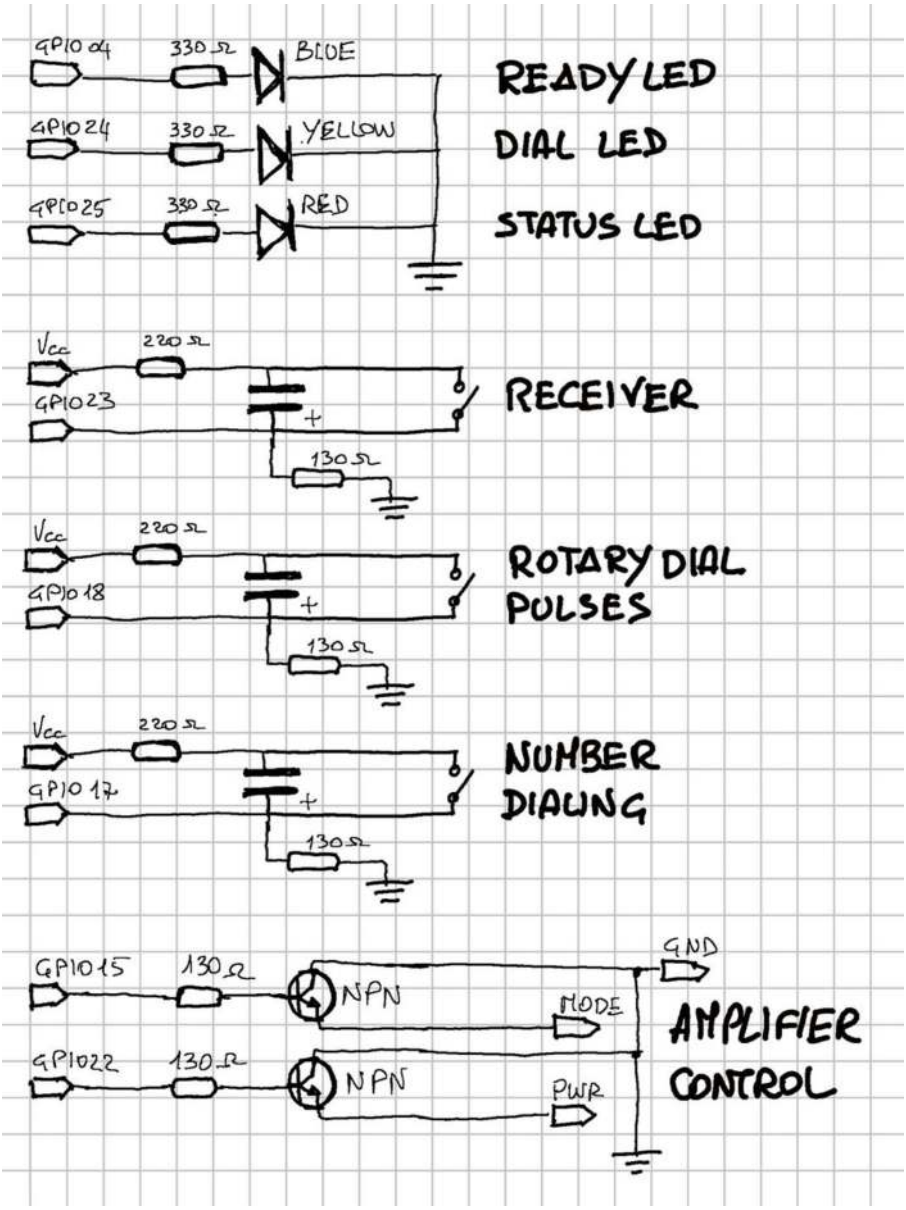

Figure 13-10. *Schematic of the simple circuit associated with the GPIO pins of the Raspberry Pi to control all the functions of the upcycled phone*

13.3.2. The Breadboard Shield

According to the drafted circuit, I made a breadboard PCB to which I soldered a female connector to fit the Raspberry Pi's male GPIO connector.

As there are only a few components—resistors, capacitors, and two transistors—it was not worth designing a PCB circuit for a single piece. In most cases, upcycling projects do not need a dedicated PCB for one piece only. On the contrary, a separate bench test of the circuit parts is always good practice, regardless of whether they are straightforward.

Still considering the concerns of the reduced available space, I excluded adopting connectors and soldered the wires directly on the PCB. See Figure 13-11.

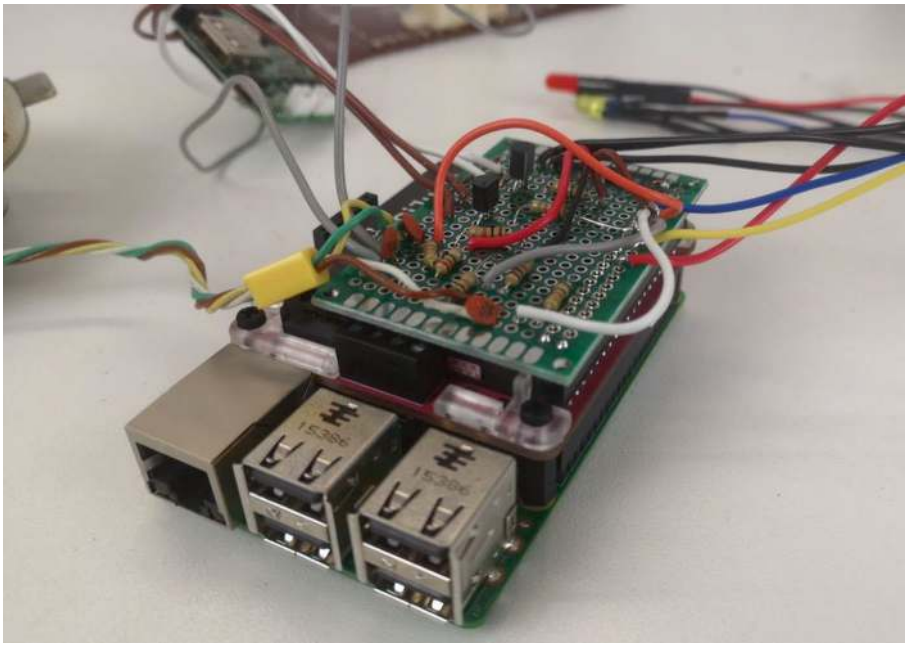

Figure 13-11. *The PCB breadboard connected to the Raspberry GPIO with the components and the upcycled telephone wires directly soldered on the circuit*

13.3.3. A Minimal Interface

This project's user interface is based on audio feedback. All the logic is based on dialing a short code number of three ciphers to get help or enable the associated feature. For this reason, there is no display or other screen-like visual interface.

Three states of the machine instead need a small visual aid: When the system is powering up—and the audio is still to be powered and set—during the composition of a number and a ready state indicator. For this reason, I included three Raspberry GPIO pins dedicated to driving three LEDs.

To avoid altering the device's design, I added two holes in front of the cover, resembling the multiline telephones used in offices in the same period. The three LEDs are inset in two circular covers I designed and 3D printed with transparent resin. See Figure 13-12.

Figure 13-12. *The front side of the telephone cover case has two holes to host the status LEDs. The LEDs are covered by cover elements 3D printed with transparent resin*

On the right side, the blue LED indicates the telephone is ready, and on the left, a red LED indicates the system is powered on. A third orange LED blinks when a number is dialed.

After the device was completed and assembled, and the functional tests were positive (see Figure 13-13), I started to develop the logic of the upcycled rotary dial telephone.

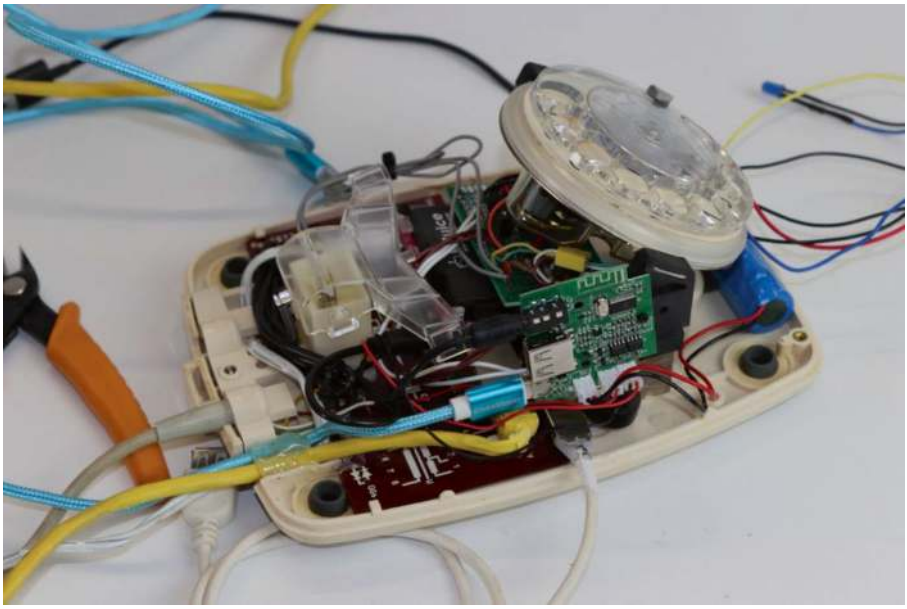

Figure 13-13. *The complete assembly of the upcycled rotary telephone before closing the case to start developing the logic*

CHAPTER 14

The Rotary Phone Software

The Rotary Phone's core is a Raspberry Pi 4B; the software consists of a series of modules running several tasks involving hardware and business logic. The components run on the Raspbian Linux distribution for the Raspberry Pi (Debian-derived).

The high-level application, developed in Python for its convenience, seamlessly manages the hardware control modules developed in C++. The Python main application's role is to trigger the events generated by user interaction and schedule the execution of independent processes, ensuring a user-friendly interaction with the Rotary Phone.

The program's business logic and the tasks it triggers are governed by JSON files. These files, easily configurable, provide a high degree of flexibility in customizing the behavior of the Rotary Phone, adapting it to various user needs and preferences.

Some extra application components are directly managed as Linux subprocesses controlled by the Python main program.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_14

Note According to the JSON configuration files, the main application can launch external processes running Linux commands or applications executing Bash commands. The software's available features can be customized and replaced by any other application that can run from the Bash terminal without impacting the application workflow.

14.1. The Python Application

The application uses the `pigpio` library to interface with the Raspberry Pi GPIO hardware and the `subprocess` library to manage the external processes triggered by user interaction. According to the `pigpio` pin identification, every pin of the Raspberry Pi GPIO connector used to control the hardware is defined as an initial constant.

14.1.1. Constants and Control Parameters

The pin numbers should be those mentioned in the Broadcom CPU (the Raspberry Pi processor). This numbering is not the GPIO connector number but the effective microprocessor pin.

```
import time
import pigpio
import subprocess
import json

PI_HIGH = 1
PI_LOW = 0
```

```
pin_hangout_led = 4          # Hangout LED indicator (pin 7)
pin_ampli_mode = 15          # Amplifier mode button (pin 10)
pin_dial_counter = 18        # Count the dialed number (pin 12)
pin_dial_detect = 27         # Detect number dialing (pin 13)
pin_ampli_power = 22         # Amplifier power button (pin 15)
pin_phone_hangout = 23       # Phone hangup/hangout input
                               (pin 16)
pin_dialer_led = 24          # Dialer counter LED (pin 19)
pin_dial_counter_led = 25    # On when the dialer is ready
                               (pin 22)
```

The global variables in the following code snippet change their status depending on the hardware status of the rotary dialer and the state of the signaling LEDs. Using the rotary dialer, only numeric sequences with the characters 0-9 are available. The `Pi Rotary` command codes are limited to three characters, making it virtually possible to define up to 1,000 different commands, from 000 to 999.

```
cb_hangout_handler = 0       # Callback handler for the
                               hangout switch
cb_dialer_handler = 0        # Callback handler for the
                               rotary dialer
cb_counter_handler = 0       # Callback handler for the rotary
                               pulse counter

'''
What is the status of the amplifier, or at least what is it
expected to be? If the amplifier status is not corresponding,
there is a number to dial to reset the status accordingly with
the pick-up switch detector.
'''
ampli_status = False     # Become true when the amplifier is in
                           a powering On/Off state
```

```
# Initially set to false it is True when the user starts
dialing a number with the
# rotary dialer. The status remains True until the rotary
dialer has not completed the
# counterclockwise rotation emitting all the impulses
corresponding to the dialed
# number.
dialer_status = False   # Becomes true when the user starts
dialing a number

# Pulse counter of the rotary dialer. When the dialer_status is
high the
# counter is incremented while, when the dialer_status goes low
the dialed
# number sequence is updated with the last number dialed.
pulses = 0

# Compound dialed number in string format. Until a number is
not recognized as a command
# the further dialed numbers are queued to the string. If the
number of characters of the
# dialed number reach the max length and have no meaning the
number is reset and the counter
# restarts to a new number.
dialed_number = ''

# Maximum number of characters of the dialed number.
# This depends on the numeric command structures decided by the
program. A three-number
# command code is sufficient for 999 different commands, maybe
sufficient!
max_numbers = 3
```

TTS, PLAYER, REBOOT, and WEATHER define the Linux commands launched as external processes; these can also run concurrently. In fact, a separate PID is retrieved for every external process without blocking user interaction through the Python application via the pigpio library.

At this point, it is possible to customize the commands, each corresponding to a different number, or to add more commands.

```
# Text-to-speech command and parameters
# Parameters: -sp = speak, -n = narrator voice (not used)
TTS = [ '/home/pi/smartphone/trans', '-sp' ]

# Mp3 play command and parameters. Volume can be a parameter of
the command but in
# this case we only use the bare call to the player. The volume
is set globally and is
# used by default.
PLAYER = ['mplayer']

# Cold reset command
REBOOT = ['sudo', 'reboot', 'now']

# Weather command
WEATHER = ['weather']
```

The sentences_file and music_file files are the two configurable JSON files defining the parameters of the program and the messages. As there is not a visual interface, all the messages are converted to voice and spoken by the TTS process:

```
# Voice comments file
sentences_file = "/home/pi/smartphone/comments.json"
# Playlist file
music_file = "/home/pi/smartphone/playlist.json"
```

14.1.2. The JSON Configuration Files

Two JSON files can be customized to change the application's behavior without modifying the code. The first contains all the strings used for the audio help, and the second defines the MP3 files that can be played with the corresponding command.

Note In this version of the project, the external commands are hardcoded. It is not difficult to use the same structure as the JSON files, including different commands. In this case, a third JSON file should be created to parametrically define the external commands and their parameters.

The comments.json File

The `phrases` and `helpsentences` parameters define the dictionary strings, while the `ICAO` parameter is related to the weather news. This is an example of how every command can be parametrized instead of hardcoding it.

```
{
  "phrases": 9,
  "list": [
    "Pi-Rotary is ready, dial commands when the red light is
    on. Dial 1 1 1 for help",
    "Closing interactive station.",
    "Starting music player",
    "Play the entire playlist.",
    "Now playing",
    "System booted",
    "Playlist content:",
```

```
    "titles",
    "Wrong command. Please, redial."
  ],
  "helpsentences": 10,
  "help": [
    "Usage information",
    "Wait for the red light before dialing a command",
    "Dial 1 2 3: play all the playlist in order",
    "Dial 1 2 4: list all the playlist titles",
    "Dial 3 2 1: play the next track in the playlist",
    "Dial numbers from 4 0 1 to play the corresponding track in
    the playlist",
    "Dial 6 6 6: hot reset the Pi Rotary",
    "Dial 9 9 9: cold reset the Pi Rotary",
    "Dial 1 0 0: hear last weather report",
    "Dial 1 1 1: these help notes"
  ],
  "ICAO": "EBBR",
  "airport": "Weather from airport station of Bruxelles.
Please wait."
}
```

The playlist.json File

This file defines four parameters: `tracks`, `folder`, `files`, and `songs`. The MP3 song files should be three-character numbers from 001 to the maximum number of tracks. For every filename in the `files` dictionary, the song title is defined in the same position as the `songs` dictionary. The `folder` key parametrizes the relocation of the MP3 files folder in the system.

```
{
  "tracks": 20,
  "folder": "/home/pi/Music/",
  "files": [
    "001",
    "002",
    "003",
    "004",
    "005",
    "006",
    "007",
    "008",
    "009",
    "010",
    "011",
    "012",
    "013",
    "014",
    "015",
    "016",
    "017",
    "018",
    "019",
    "020"
  ],
  "songs": [
    "Precious illusions",
    "Hang on to me tonight",
    "Voyager",
    "Chiquitita",
    "Elephant gun",
    "Suzanne",
```

```
    "Tired of sleeping",
    "The Wall",
    "Is there any way out of this dream?",
    "It's all over now, baby blue",
    "The wall",
    "Heroes",
    "Mother",
    "The boxer",
    "Tonight",
    "One of these days",
    "Knockin' on Heaven's door",
    "Missis Robinson",
    "Little bird",
    "Angel in my heart"
  ]
}
```

14.1.3. Event-driven Application

The user interaction controls the application business logic through the switches and the rotary dialer. In this way, all the application functions run asynchronously through callbacks.

The `main` function, launched at startup, only initializes the hardware interface and enables the events callbacks. See Listing 14-1.

Listing 14-1. The Main Function

```
if __name__ == '__main__':
    # Main application
    initGPIO()

    # looping infinitely
    while True:
        pass
```

The initGPIO() Function

Initialize the GPIO library and set the callback for the interested pins using the corresponding function. To avoid hardware problems, the Broadcom GPIO pin 28 should not be set to a callback.

The `initGPIO()` function is composed of three parts: hardware GPIO pin initialization, starting the callbacks functions, and loading the music JSON dictionary. See Listing 14-2.

Listing 14-2. The initGPIO() Function

```
# Set the output pins
pi.set_mode(pin_hangout_led, pigpio.OUTPUT)
pi.set_mode(pin_ampli_mode, pigpio.OUTPUT)
pi.set_mode(pin_ampli_power, pigpio.OUTPUT)
pi.set_mode(pin_dialer_led, pigpio.OUTPUT)
pi.set_mode(pin_dial_counter_led, pigpio.OUTPUT)

# Set the input pins
pi.set_mode(pin_dial_counter, pigpio.INPUT)
pi.set_pull_up_down(pin_dial_counter, pigpio.PUD_DOWN)
pi.set_mode(pin_dial_detect, pigpio.INPUT)
pi.set_pull_up_down(pin_dial_detect, pigpio.PUD_DOWN)
pi.set_mode(pin_phone_hangout, pigpio.INPUT)
pi.set_pull_up_down(pin_phone_hangout, pigpio.PUD_DOWN)

# Set the callback for the interested pins and get the
handlers
set_callbacks()

# Load the music lists
with open(music_file) as file:
    dictionary = json.load(file)
```

```
    track_list = dictionary['files']
    track_titles = dictionary['songs']
    tracks = int(dictionary['tracks'])
    music_path = dictionary['folder']

    # Load the message tracks
    with open(sentences_file) as file:
        dictionary = json.load(file)

    num_messages = dictionary['phrases']
    help_strings = dictionary['helpsentences']
    text_messages = dictionary['list']
    help_messages = dictionary['help']
    weather_airport = dictionary['airport']
    weather_ICAO = dictionary['ICAO']

    reinit()
```

14.1.4. Callback Functions

For every independent event triggered by the user interaction, there is the `set_callback` function and the corresponding reset to disable the callback. The callback reset functions are needed because not all the triggered events can run concurrently. See Listing 14-3.

Listing 14-3. The Callback Functions

```
def set_callbacks():
    '''
    Enable the callback functions associated to the GPIO pins
    and save the callback handlers
    '''
    global pi
```

```
    global cb_counter_handler
    global cb_dialer_handler
    global cb_hangout_handler

    cb_hangout_handler = pi.callback(pin_phone_hangout, pigpio.
                         EITHER_EDGE, hangout)
    cb_dialer_handler = pi.callback(pin_dial_detect, pigpio.
                        EITHER_EDGE, dial_detect)
    cb_counter_handler = pi.callback(pin_dial_counter, pigpio.
                         EITHER_EDGE, pulse_count)

    # LED high when the rotary is accepting numbers
    pi.write(pin_dial_counter_led, PI_HIGH)

def release_callbacks():
    '''
    Disable the callback functions
    '''
    global pi
    global cb_counter_handler
    global cb_dialer_handler
    global cb_hangout_handler

    cb_hangout_handler.cancel()
    cb_dialer_handler.cancel()
    cb_counter_handler.cancel()

    # LED high when the rotary is accepting numbers
    pi.write(pin_dial_counter_led, PI_LOW)

def cb_release_rotary():
    '''
    Cancel the specific callback functions (disable the
    interrupt)
```

```
    associated to the rotary dialer
    '''
    global pi
    global cb_counter_handler
    global cb_dialer_handler

    cb_counter_handler.cancel()
    cb_dialer_handler.cancel()

    # LED high when the rotary is accepting numbers
    pi.write(pin_dial_counter_led, PI_LOW)

def cb_set_rotary():
    '''
    Enable the specific callback functions (disable the
    interrupt) associated to the rotary dialer
    '''
    global pi
    global cb_counter_handler
    global cb_dialer_handler

    cb_dialer_handler = pi.callback(pin_dial_detect, pigpio.
                        EITHER_EDGE, dial_detect)
    cb_counter_handler = pi.callback(pin_dial_counter, pigpio.
                         EITHER_EDGE, pulse_count)

    # LED high when the rotary is accepting numbers
    pi.write(pin_dial_counter_led, PI_HIGH)

def cb_set_hangout():
    '''
    Set the hangout callback
    '''
```

```
    global pi
    global cb_hangout_handler

    cb_hangout_handler = pi.callback(pin_phone_hangout, pigpio.
    EITHER_EDGE, hangout)
```

14.1.5. Triggered Events

Every event function triggered by a callback has a set/reset sequence of the callback functions that should not be intercepted, as it can't run concurrently. This prevents, for example, launching the same command twice. See Listing 14-4.

Listing 14-4. The Triggered Events

```
def play_all_tracks():
    '''
    Play all the tracks on the playlist.json file
    after powering the amplifier and announcing every
    track title.
    Then power off the amplifier.
    '''
    global track_position
    global is_playing
    global music_path
    global tracks

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()

    # Start from the first track of the playlist
    track_position = 0
```

```
    # Announce the initial message
    txt = text_messages[3] + ' ' + text_messages[2]
    runCmd([TTS[0], TTS[1], txt])

    # Play tracks until the playlist ends
    while track_position < tracks:

        debug_message('pos ' + str(track_position) + ' tracks '
        + str(tracks))

        # Announce the name of the track
        txt = text_messages[4] + ' ' + track_titles[track_
              position]
        runCmd([TTS[0], TTS[1], txt])

        # Create the track file name and play it
        txt = music_path + track_list[track_position] + '.mp3'
        runCmd([PLAYER[0], txt])

        # Update the track number
        track_position += 1

        # Stop the playing loop if the user hangout
        if pi.read(pin_phone_hangout) is PI_LOW:
            break;

    # Disable the amplifier, if it has not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False

def list_all_tracks():
    '''
    Tell all the tracks on the playlist.json file
```

```
    after powering the amplifier and saying every track title.
    Then power off the amplifier.
    '''
    global is_playing
    global tracks
    global track_titles

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()

    # Announce the initial message
    txt = text_messages[6] + ' ' + str(tracks) + ' ' + text_
          messages[7]
    runCmd([TTS[0], TTS[1], txt])

    counter = 0     # Playlist title counter

    # Play tracks until the playlist ends
    while counter < tracks:

        # Announce the name of the track
        txt = ' ' + str(counter + 1) + ': ' + track_
              titles[counter] + ", "
        runCmd([TTS[0], TTS[1], txt])

        # Update the title number
        counter += 1

    # Disable the amplifier, if it has not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False
```

```
def say_help():
    '''
    Tell the help notes.
    '''
    global is_playing
    global help_strings
    global help_messages

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()

    counter = 0     # Playlist title counter

    # Play messages
    while counter < help_strings:

        txt = help_messages[counter]
        runCmd([TTS[0], TTS[1], txt])

        # Update the title number
        counter += 1

    # Disable the amplifier, if it is not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False

def play_track():
    '''
    Play a track based on the parameters of the playlist.
    json file after powering the amplifier and announcing the
    track title.
```

```
    Then power off the amplifier.
    '''
    global track_position
    global is_playing
    global music_path
    global tracks

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()

    # Announce the name of the track
    txt = text_messages[4] + ' ' + track_titles[track_position]
    runCmd([TTS[0], TTS[1], txt])

    # Create the track file name and play it
    txt = music_path + track_list[track_position] + '.mp3'
    runCmd([PLAYER[0], txt])

    # Increment the number of the track and if the value
      is bigger
    # than  the max number of tracks it is reset.
    track_position += 1
    if track_position == tracks:
        track_position = 0

    # Disable the amplifier, if it has not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False
```

The check_number() function acts as a menu according to the recognized three-digit commands executed with the rotary dialer. When a command is recognized, the set/reset callback pattern is configured accordingly. See Listing 14-5.

Listing 14-5. The check_number() Function

```
def check_number():
    '''
    Check if the current number corresponds to a valid command.
    '''
    global dialed_number
    global track_position
    global pi

    debug_message(str(dialed_number))

    # Numeric commands and related functions
    if dialed_number is not '':
        if int(dialed_number) == 666:
            # Restart the application to initial conditions
            dialed_number = ''
            reinit()

        # Play the next track
        elif int(dialed_number) == 321:
            # Disable the interrupts until finished
            release_callbacks()
            # Start playing the next track
            play_track()
            # Enable the interrupts
            cb_set_hangout()
            # If the hangout is still active, enable the
              dialer too
```

```
            if pi.read(pin_hangout_led) is PI_HIGH:
                cb_set_rotary()

        # Play all tracks in sequence
        elif int(dialed_number) == 123:
            # Disable the interrupts until finished
            release_callbacks()
            # Play the entire playlist
            play_all_tracks()
            # Enable the interrupts
            cb_set_hangout()
            # If the hangout is still active, enable the
              dialer too
            if pi.read(pin_hangout_led) is PI_HIGH:
                cb_set_rotary()

        # Tell the playlist titles
        elif int(dialed_number) == 124:
            # Disable the interrupts until finished
            release_callbacks()
            # Tell the playlist titles
            list_all_tracks()
            # Enable the interrupts
            cb_set_hangout()
            # If the hangout is still active, enable the
              dialer too
            if pi.read(pin_hangout_led) is PI_HIGH:
                cb_set_rotary()

        # Play the desired track
        elif (int(dialed_number) <= tracks + 400) and
        (int(dialed_number) > 400):
            track_position = int(dialed_number) - 401
```

```
        # Disable the interrupts until finished
        release_callbacks()
        # Start playing the next track
        play_track()
        # Enable the interrupts
        cb_set_hangout()
        # If the hangout is still active, enable the
          dialer too
        if pi.read(pin_hangout_led) is PI_HIGH:
            cb_set_rotary()

    # Tell the help notes
    elif int(dialed_number) == 111:
        # Disable the interrupts until finished
        release_callbacks()
        # Tell the help messages
        say_help()
        # Enable the interrupts
        cb_set_hangout()
        # If the hangout is still active, enable the
          dialer too
        if pi.read(pin_hangout_led) is PI_HIGH:
            cb_set_rotary()

    elif int(dialed_number) == 999:
        # Reboot the system
        runCmd([REBOOT[0], REBOOT[1], REBOOT[2]])

    # Tell the weather
    elif int(dialed_number) == 100:
        # Disable the interrupts until finished
        release_callbacks()
```

```
            # Retrieve and say the weather message
            get_weather()
            # Enable the interrupts
            cb_set_hangout()
            # If the hangout is still active, enable the
              dialer too
            if pi.read(pin_hangout_led) is PI_HIGH:
                cb_set_rotary()

        elif int(dialed_number) == 999:
            # Reboot the system
            runCmd([REBOOT[0], REBOOT[1], REBOOT[2]])
```

14.1.6. Low-level Functions

The low-level functions are a small set of methods that launch the hardcoded commands as external processes. See Listing 14-6.

Listing 14-6. The Low-Level Functions

```
def list_all_tracks():
    '''
    Tell all the tracks on the playlist.json file
    after powering the amplifier and saying every track title.
    Then power off the amplifier.
    '''
    global is_playing
    global tracks
    global track_titles

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()
```

```
    # Announce the initial message
    txt = text_messages[6] + ' ' + str(tracks) + ' ' + text_
          messages[7]
    runCmd([TTS[0], TTS[1], txt])

    counter = 0     # Playlist title counter

    # Play tracks until the playlist ends
    while counter < tracks:

        # Announce the name of the track
        txt = ' ' + str(counter + 1) + ': ' + track_
              titles[counter] + ", "
        runCmd([TTS[0], TTS[1], txt])

        # Update the title number
        counter += 1

    # Disable the amplifier, if it has not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False

def say_help():
    '''
    Tell the help notes.
    '''
    global is_playing
    global help_strings
    global help_messages

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()
```

```
    counter = 0     # Playlist title counter

    # Play messages
    while counter < help_strings:

        txt = help_messages[counter]
        runCmd([TTS[0], TTS[1], txt])

        # Update the title number
        counter += 1

    # Disable the amplifier, if it is not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False

def play_track():
    '''
    Play a track based on the parameters of the playlist.
    json file after powering the amplifier and announcing the
    track title.
    Then power off the amplifier.
    '''
    global track_position
    global is_playing
    global music_path
    global tracks

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()
```

```
    # Announce the name of the track
    txt = text_messages[4] + ' ' + track_titles[track_position]
    runCmd([TTS[0], TTS[1], txt])

    # Create the track file name and play it
    txt = music_path + track_list[track_position] + '.mp3'
    runCmd([PLAYER[0], txt])

    # Increment the number of the track and if the value
      is bigger
    # than the max number of tracks it is reset.
    track_position += 1
    if track_position == tracks:
        track_position = 0

    # Disable the amplifier, if it has not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False

def tts_message(msg):
    '''
    Prepare the message msg to be played as an audio command
    :param msg: The message code accordingly with the list in
    the json file
    :return: 0 or the tts bash command execution error code
    '''

    # Create the full text message
    tText = text_messages[msg]

    return runCmd([TTS[0], TTS[1], tText])
```

```
def runCmd(cmd, extra_info = False):
    '''
    Execute a subprocess command managing the return value,
    stdout, stderr and the return code (0 or not 0 if error
    occurred)
    :param cmd: The bash command with the parameters
    :return: 0 or the error returncode
    '''

    proc = subprocess.Popen(cmd,
                            stdout=subprocess.PIPE,
                            stderr=subprocess.PIPE,
                            )
    stdout, stderr = proc.communicate()

    return proc.returncode # , stdout, stderr

def get_weather():
    '''
    Retrieve the weather from the nearest airport. The desired
    international airport weather station ICAO four character
    code should be set in the comments.json file.
    The command speaks the weather data returned from the call.
    '''
    global is_playing
    global weather_airport
    global weather_ICAO

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()
```

```
# Announce the weather retrieval
runCmd([TTS[0], TTS[1], weather_airport])

# Execute the weather command
cmd = [WEATHER[0], weather_ICAO]

proc = subprocess.Popen(cmd,
                        stdout=subprocess.PIPE,
                        stderr=subprocess.PIPE,
                        )
stdout, stderr = proc.communicate()
# Divide the weather message in a list of single lines
# removing the newline characters
forecast = stdout.splitlines()

w = 4   # First useful line of the weather forecast

# Say the forecast meaningful strings (starting from 4)
# Note that forecast is a list of bytes so every text line
# should be decoded to the corresponding ASCII string
while w < len(forecast):
    text = forecast[w].decode('ascii')
    runCmd([TTS[0], TTS[1], text])
    w += 1

# Disable the amplifier, if it has not yet disabled by
  the user
if is_playing is True:
    ampli_on_off()
    is_playing = False
```

```
def wrong_command():
    '''
    Says wrong command if the dialed number is invalid
    '''
    global is_playing

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()

    # Announce the weather retrieval
    runCmd([TTS[0], TTS[1], text_messages[8]])

    # Disable the amplifier, if it has not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False

def play_tts_sentence(msg):
    '''
    Play a text message. Call this command only outside of the
    playing track as it enables and disables the amplifier
    by itself.
    :param msg: The message code accordingly with the list in
    the json file
    '''
    global is_playing

    # Enable the amplifier and set the playing flag
    is_playing = True
    ampli_on_off()

    tts_message(msg)
```

```
    # Disable the amplifier, if it has not yet disabled by
      the user
    if is_playing is True:
        ampli_on_off()
        is_playing = False

def hangout(self, event, tick):
    '''
    Callback function.
    Detect the hangon/hangoff switch of the pickup.
    The status of the switch will power the amplifier
    accordingly when needed. When the function is called the
    LED is set accordingly to the status of the pin
    '''
    global pi
    global is_playing

    if pi.read(pin_phone_hangout) is PI_HIGH:
        # Disable the interrupts until finished
        release_callbacks()
        # Show the ready LED
        pi.write(pin_hangout_led, PI_HIGH)
        # Activation message
        play_tts_sentence(0)
        # Enable the interrupts
        set_callbacks()

    else:
        # Disable the interrupts until finished
        release_callbacks()
        # End message
```

```
        play_tts_sentence(1)
        # Disable the ready LED
        pi.write(pin_hangout_led, PI_LOW)
        # Enable only the hangout interrupt
        reinit()
        cb_set_hangout()
```

PART VI

The Process

"Another place I will never figure out until I see it," said Ray, stepping inside. Tommy felt guilty and responsible for the situation. Until that moment, he did not speak, walking head down and looking at the floor.

"Tommy, this place is not as bad as we imagined," continued Ray. Tommy looked around. Yes was the only answer he could give.

The prison cell was comfortable, similar to a small apartment. The most curious aspect was the simple, doorless rectangular entrance. The room had no windows. It included a small kitchen corner equipped with a microwave and a half-sized fridge, a small toilet on the opposite side, and two beds, one in front of the other immediately beside the entrance. Ray noted a digital watch over the entrance wall. The digits showed 11:37.

"Tommy, look, it is a countdown."

Tommy sat on the right bed, raising his head blankly. Ray was trying to appear less worried than he was. He started a compulsory exploration of the room.

"Tommy, there is good food in the fridge! We can heat up something if you are hungry. There is also a coffee machine..." – Tommy showed no interest in their surroundings.

"Tommy, please," said Ray, sitting close to his son.

"Stop with this deadly expression. This will not help." Ray put his arm around Tommy's shoulder.

"Beating yourself up over a mistake is useless." Tommy looked at his father, drafting a weak smile. "That is the worst smile I have ever seen on your face," Ray said. At this point, Tommy smiled at his father and relaxed slightly.

"Come on, stand up and come with me," Ray suggested.

"Let's sit at that table and catch each other up on this incredible adventure. Maybe we can come up with a plan." Tommy nodded, unconvinced, and slowly went to the table. After Ray carried some food from the fridge and put it on the table, they realized that the last time they had eaten was at least 12 hours ago.

Ray prepared sandwiches with peanut butter. They also had bacon, chicken salad, some pre-cooked pancakes heated in the microwave, and orange juice. Ray made black coffee for himself. Tommy wasn't a coffee drinker like Ray.

They devoured the first two sandwiches, and then Tommy showed a brighter expression and started talking about his journey.

"Look," said Tommy, pulling out of his pocket the amusement park leaflet with a suggested path marked on the map on the back.

"I saw you here, at the main visitors' entrance!" Ray exclaimed.

"I suppose yes," Tommy answered. "I found a weird robot that welcomed me and handed me this leaflet," said Tommy, showing the leaflet.

"Strangely, only its upper body moved. It was dressed in a curious, striking uniform but was just a machine impersonating the ticket controller. Then, I passed through a rotating door and started exploring around," concluded Tommy.

"Instead, I entered from the opposite side, a long corridor used for maintenance or something like that," Ray added. "Now I can understand Sonya's first words..."

"Who is Sonya?" Tommy asked.

"I am confused about her," Ray said. "She is... someone who appears whenever I am lost while searching for you. It is a complex answer, Tommy. She acts as my guide, a friend, maybe more than a friend... It's complicated." Tommy was listening with rapt attention.

"The first time I met her, she asked why I entered from the thousand doors side. As far as I know, Sonya is the only other human living in the park."

"There are a thousand doors? Really?" Tommy interrupted.

"Yes," Ray continued. "When you suddenly disappeared, I spent hours searching for a combination to activate the cryptex again. Then, when I wrote the number 6736, I was allowed to enter. Maybe every time the access to the portal changes," he concluded.

"But I entered another number!" Tommy said.

"I entered 2384. I remember very well."

"And why did you try that combination?" Ray asked.

"I have no idea. I was just playing with some random numbers."

"Let me see…," Ray said, taking his Palm out of his pocket. He showed Tommy an image reproducing an old rotary dialer.

"Look," he said to his son, showing the tiny black and white screen. "Every number also represents a group of characters. After trying every possible numeric sequence I knew, I figured this was my last chance, and it worked. I entered "OPEN," corresponding to the number 6736."

Tommy looked up at his father, full of admiration.

"Are you sure you entered 2384?," asked Ray. Tommy nodded.

"Due to an incredible coincidence, you entered the word BDTH, which is the name of this place. I never imagined something like this."

"But the name of the amusement park is longer than that!," Tommy said.

"That's true," Ray answered. "The full name is BDTH6159. But all the robots I spoke with during my journey always mentioned BDTH only. They never added the other four numbers." Ray started playing on the Palm screen.

"There should be a reason," Ray murmured. "The four ciphers should be related to the name. Let me try…"

"This place is full of enigmas!" exclaimed Tommy.

"Wait, there should be an explanation, "Ray stated, focusing on the Palm screen and writing something with the small pen on the capacitive surface of the device.

Tommy counted 17 minutes on the wall display before Ray finally looked at him with a satisfied expression.

"I figured it out!" Solving challenging enigmas has always been one of Ray's preferred activities.

"Look," Ray said, showing the solution to his son. "The number 6159 is related to the name, and if I am right, every word that activates the cryptex is related to a number that has meaning inside this world. Unfortunately, I can't imagine what it is."

Tommy looked at his father with admiration.

"Look," Ray followed.

"Reducing BDTH to numbers you already know, the result is 2584. The trick is finding the generator key. It is so simple that I missed finding it at first. The generator key is the number four!," he continued.

"How do obtain the number 6154 from 2384 (BDTH)? To every cipher of the first sequence of four, respectively, you:

- Add four
- Subtract four
- Sum the first and second cipher
- Sum the third and fourth cipher

"You're a genius, dad!" Tommy exclaimed with a shining expression on his face.

"Ray, I am impressed!" Ray immediately recognized Sonya's voice at the room's entrance. "You decoded and found the algorithm of the four ciphers," she concluded by crossing the room to sit at the table.

"You must be Tommy," She said after sitting down.

"Yes...," Tommy answered.

"Tommy, this is Sonya. I told you about her before," Ray stated, a bit embarrassed by the unexpected visit.

"You are very smart, Ray. This will be an advantage during the process," Sonya continued. Tommy looked proudly at his father.

"But now, you two, follow my advice: go to sleep. Tomorrow will be a long day." Ray smiled at her silently. "Have a good night. I will see you tomorrow." Then she left the room.

"Tommy, she is correct; we need a rest." Ray said, approaching the left bed. Tommy nodded and followed his father. The lights diminished as both were in their beds, leaving the room in a comfortable darkness.

"Dad, are you already asleep?" "No," Ray answered.

"These beds remind me of when we crossed a whole state on the night train. Do you remember?"

"Yes, I remember. Now try to sleep, Tommy," Ray concluded.

"You and Sonya seem so close," Tommy said. Then, they both fell into a deep sleep.

Father and son were fully awake when the light in the room progressively increased. Ray made black coffee while Tommy prepared a couple of peanut butter sandwiches. While Ray was drinking his coffee, standing near the small kitchen, a man in uniform appeared at the room's entrance.

"When you are ready, you are invited to go out. Beside the exit, you will find some important notes regarding this process. Take your time." He then went out.

"What a strange accent," said Tommy.

"He is a robot, Tommy," Ray answered. "What resembles an accent may be a defect of his voice processor. Many things in this place need some serious repairing," Ray said, eating his sandwich.

The room was connected to the exit door by a long corridor similar to the one Ray walked through when his journey started, but with better lighting. A couple of meters before the exit door, on a small desk, a screen showed a text loop and a big, red button labeled "Press to open the door."

Both read the text loop carefully a couple of times. The first part concerned the accusation. The text was formal, with redundant expressions typical of legal language.

The second part was more interesting—it contained instructions on what to do before the process. Ray found the indications inconsistent and vague. The only sure thing was they should go to the "Seven Bells Columns." One of their duties was choosing their lawyer at the "Judgment Box." Maybe it was near the column bells.

The process would start at noon. Concerning the process and judgment procedure, there were only a few words at the end. It would develop in three main phases—the audition, where the judge hears the witnesses; the debate between the prosecutor and the defense; and the verdict.

The verdict would be dictated the next day when the process ended.

"Not so much," said Tommy. "I see," answered Ray with a defeated expression. As he pressed the button on the desk, the screen went black, and the exit door slid open, so they stepped out into a hot, sunny day.

"It is not difficult to find the columns," said Tommy as they were outside. The magnificent structure of seven columns was visible a few hundred meters from the exit.

Ray estimated they were at least 50 meters high, and the mechanisms on the top were difficult to see.

"The diameter of these columns is impressive," commented Ray. "We should start from there." They started walking along the wide gravel lane.

"This place seems different today. All the attractions have stopped," Tommy commented.

"True," confirmed Ray. "Maybe this trial is a major event for the amusement park population. Remember that no one has visited this place for decades." Tommy nodded.

The seven columns were incredible from the inside of the circle. Both remained silent for a while, intimidated by the building. The diameter of the column circle was at least 200 meters.

In the center, emerging from a big, squared granite block, was a statue of a scientist bent over a bench soldering a circuit.

"That must be a tribute to the creator," said Ray. Under the statue, text engraved on a dark, shiny metal tag stated the three fundamental laws of robotics.

1. A robot cannot injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Reading them, Ray felt a bit more comfortable. "We should search around; the Judgment Box might be nearby," said Tommy.

They found the box behind a granite block. It resembled a snack dispenser from the late 60s. Inside the machine, a neon light was flickering. Inside, three rows were labeled 1, 3, and 4 with eight horizontal labels, from A to G.

Due to the lack of regular maintenance, row number 1's labels were no longer legible. Row number 3's eight labels had been replaced by an "Out of Order" white sticker. The only cell on row number 4 was labeled D.

The glass protecting the cells was dirty and opaque at several points, while the metallic machine case was rusty.

"It'll be a miracle if it still works," commented Ray, examining the device. On the side, there was a small keyboard with the keys A-G and 1-8 aligned in four rows, and the # and * keys on the fifth row to the bottom. The two central spaces had a single, double key with the word "ENTE " white on red."

"It was ENTER a long time ago," commented Tommy.

A small metal label on top of the keyboard showed text that was difficult to read due to the rust.

Only people charged with crimes are allowed to use this machine. Make your choice of the preferred defense lawyer.

PRESS # TO START

PRESS ENTER TO CONFIRM

*PRESS * TO ABORT*

"It seems there are not many choices," said Ray while pressing the keys "D, 4, ENTER". Tommy and Ray watched. For a minute or so, nothing happened, then the machine started with a metallic ticketing for a while, then silence again.

Finally, after another minute, there was a clang and the sound of rotating gears. Then, the machine spat out a long paper ticket.

Please give this ticket to the doorman when entering the judgment hall.

The line below, in bold text, mentioned the chosen lawyer:

You chose the defense lawyer D-4. Good luck with your trial.

On the second half of the ticket, a map showed the path to the judgment hall. "I think we should move, Tommy. We don't have much time," Ray said.

Tommy took the ticket and then moved, following the map. The judgment hall was a medium-sized wooden construction with a large entrance. The building was on top of a small hill and opened to a long stair. The building was not as majestic as the court buildings that Ray and Tommy were used to seeing on TV. It was a smaller-scale building that mimicked those more significant courthouses.

When Ray and Tommy reached the middle of the stairs, the powerful sound of bells announced it was time for the process to start.

"It's time," said Tommy. "I see. Don't be afraid, Tommy," Ray answered, seeing his son's worried expression.

Ray handed the ticket to the doorman, who, after a quick look, invited them to follow him. The doorman opened a large wooden gate to a wide semicircular room that could hold hundreds of people. The shape and the distribution of the rows of seats resembled an arena.

In the lower part, the judge was waiting behind a dark mahogany table.

The doorman instructed them to sit in two seats in the first row, the farthest from the judge. After they sat there, the doorman stood near the door entrance, maybe waiting for orders.

Ray felt relieved when he saw Sonya come inside and sit beside him. She didn't speak but smiled at Ray and Tommy. There was a palpable sense of anticipation. Ray held Sonya's hand while the judge hit the wooden gavel three times on the table, declaring the start of the process.

"Another journey is starting…," Ray thought, then he focused his attention on the judge.

CHAPTER 15

Chess with Arduino UNO R4

Contribution of Luis Garcia.

Figure 15-1. *While Ray and Tommy anxiously wait to see how the destiny wheel will drive their lives, hung to the result of a chess game between the prosecutor and their lawyer, you'll see how it is possible to create a tiny chess engine using a microcontroller and a sketch*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_15

The "classic" version of Arduino UNO R3, based on the AVR 328p microprocessor, has survived for years. It is the icon of every maker, and I bet that the iconic UNO will survive for years to come.

Nowadays, the most recent Arduino boards implement powerful Arm-based microcontrollers oriented to IoT and embedded Machine Learning (ML) applications.

The recent Arduino UNO R4, part of this family of innovative Arduino boards, does not aim to replace the previous model, but completes the coverage of the wide range of "Arduino" low-cost boards.

This chess project was developed in cooperation with the maker and friend Luis Garcia—yes he is the technical reviewer of this book—who worked on the MINIMA model of the Arduino UNO R4. At the same time, I developed the same project on the Arduino UNO R4 WiFi model. See Figure 15-1.

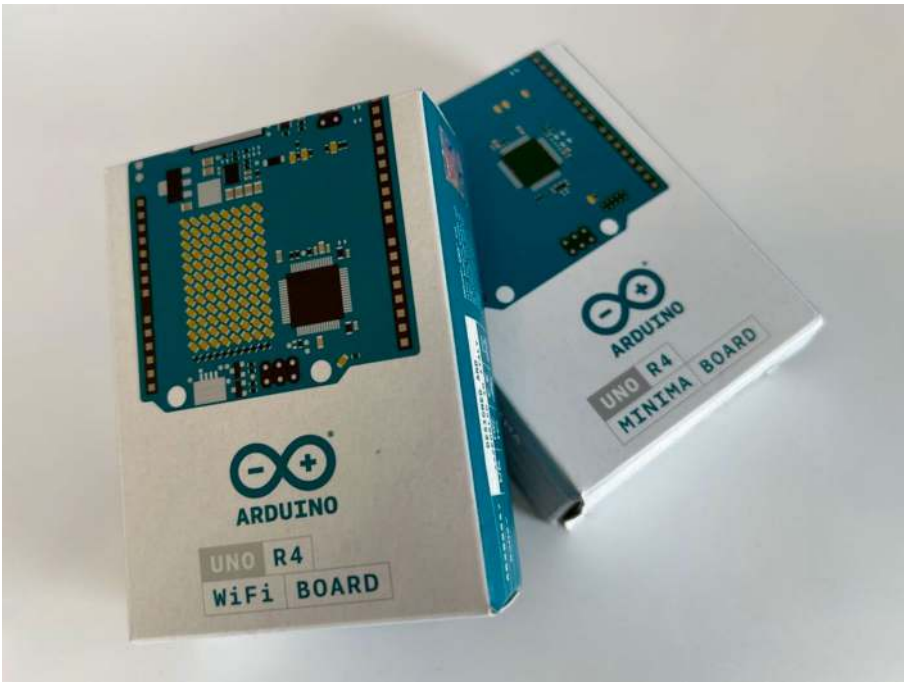

Figure 15-2. *The two Arduino UNO R4 versions (on the market by 2024): the UNO R4 WiFi (left), and the MINIMA (right), the cheaper of the two models*

We developed a project that required some small changes to be compatible on both board models; the goal was to see if it is possible to use the Arduino UNO R4 models to play chess against a human in an appreciably way.

Spoiler: The answer is yes.

15.1. The R4 WiFi and MINIMA Boards

Arduino UNO R4 models host a powerful 32-bit ARM MCU (Microcontroller Unit) and—only with the WiFi model—an ESP-32s included on the same board.

The two microcontrollers can work together in different configurations; the advanced WiFi version of the board also includes an LED matrix that can be programmed to show short messages and icons. Both boards have the same form factor as the classic Arduino UNO boards.

15.1.1. UNO R4 WiFi Specifications

The Arduino UNO R4 WiFi board is shown in Figure 15-3. Its specifications are as follows:

- Microcontroller Renesas RA4M1 (Arm® Cortex®-M4) 48MHz
- ESP-32 S3 secondary core up to 240MHz
- RA4M1 (main memory): 256KB Flash, 32KB RAM
- ESP-32 S3 memory: 384KB ROM, 512KB SRAM
- USB: USB C power and programming port
- Fourteen digital I/O pins
- Six analog inputs

- One DAC
- Six PWM pins
- UART, I2C, SPI, and CAN bus
- Voltage: 5Vcc, ESP-32 S3 3.3Vcc
- Input voltage (from the power supply): 6-24 V

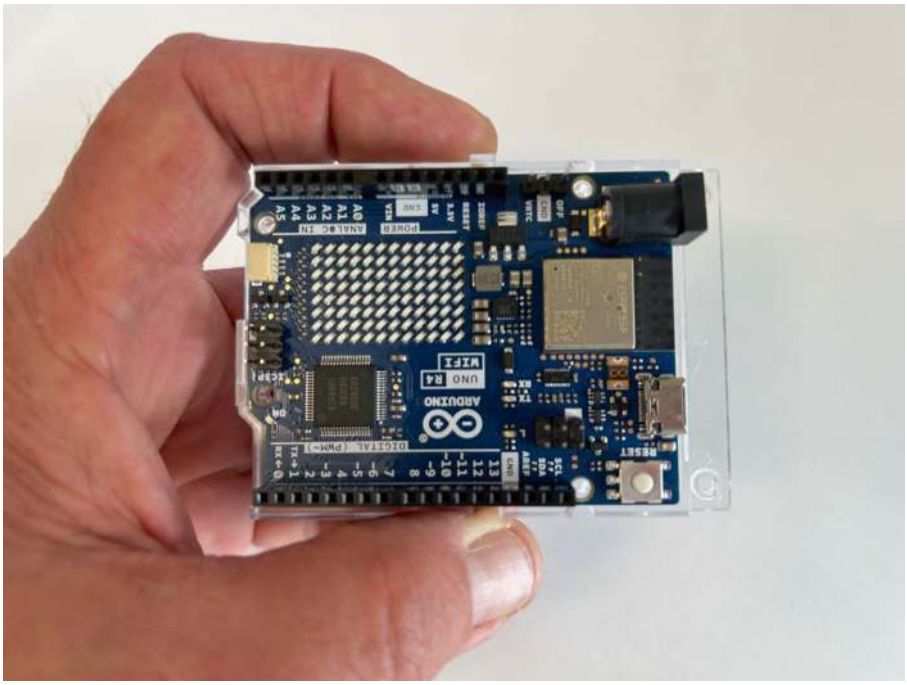

Figure 15-3. *The Arduino UNO R4 WiFi board. The bottom-left side hosts the ARM microcontroller (MCU) software and hardware connected to the ESP32-S3 visible to the right of the board. The architectures support WiFi connection from the sketch through the ESP secondary microcontroller. On the top left, the programmable LED matrix.*

15.1.2. UNO R4 MINIMA Specifications

The UNO R4 MINIMA is shown in Figure 15-4. Its specifications are as follows:

- Microcontroller Renesas RA4M1 (Arm® Cortex®-M4) 48MHz
- RA4M1 (main memory): 256KB Flash, 32KB RAM
- USB: USB C power and programming port
- Fourteen digital I/O pins
- Six analog inputs
- One DAC
- Six PWM pins
- UART, I2C, SPI, and CAN bus
- Voltage: 5Vcc, ESP-32 S3 3.3Vcc
- Input voltage (from the power supply): 6-24 V

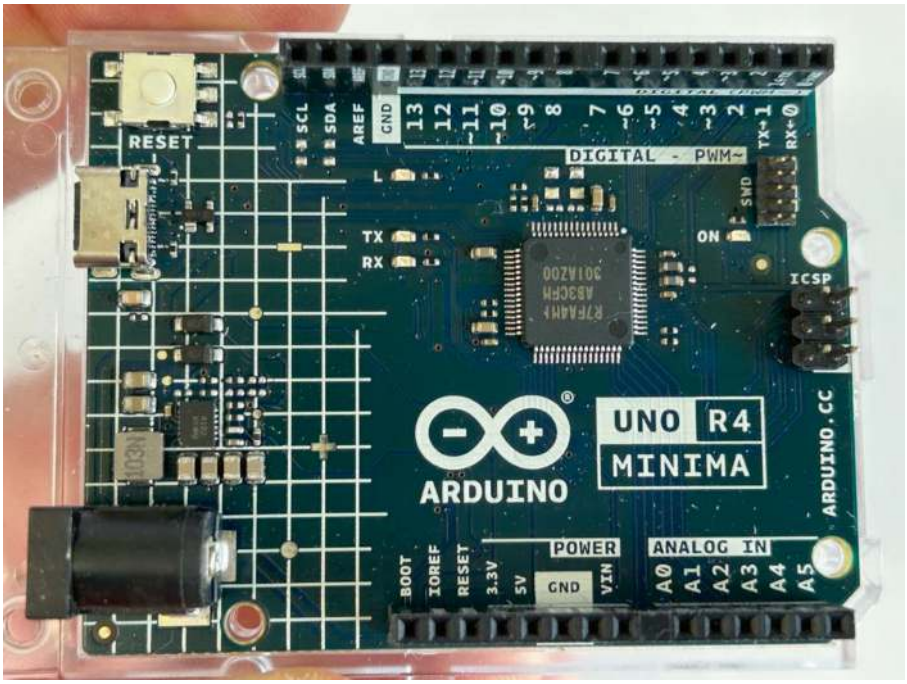

Figure 15-4. *The MINIMA version of the Arduino UNO R4 board. The ARM MCU is the same, but the LED matrix and the ESP32-S3 are absent. Indeed, it is possible to connect an ESP32-S3 secondary board without difficulty through the GPIO.*

15.2. Computers Playing Chess

In 1957, Alex Bernstein developed a program to play a complete chess game on an IBM 704. The details of this first computer program were published in *Scientific American* issue 158. I noticed this historical event only in 1988—I was born later—when the article was reprinted. I am not a great chess player, but I am fascinated by the algorithms behind computer chess.

Between the end of 1970 and the first decades of the 1980s, modern microprocessors, such as the Rockwell 6502 and the Z80 by Zilog, gave new life to developing computer chess programs. In 1977, Fidelity Electronics produced the "Chess Challenger," the first commercial chess computer.

Although not based on a microcontroller, IBM's Deep Blue represented a new milestone between 1996 and 1997. This program became famous for defeating the world chess champion Garry Kasparov in 1997 using the power of parallel computing.

The mix of scientific and mathematical interest in chess algorithms and a continuously growing community of open source developers interested in this topic sparked development.

The algorithms developed over the years remain part of the public domain heritage. Consider this list of the most popular, starting in the 90s:

- **Fritz** (1990s - Present):
  - Algorithm: Alpha-beta pruning, sophisticated evaluation functions.
  - A commercial chess program that has remained strong through continuous updates, incorporating various search optimizations and heuristics.
- **Rybka** (2005)
  - Algorithm: Improved evaluation functions and efficient search techniques.
  - It has dominated computer chess tournaments with innovative evaluation methods and search optimizations for several years.

- **Stockfish** (2008 - Present)
  - Algorithm: Alpha-beta pruning, bitboards, advanced evaluation functions, and multithreading.
  - Open source and consistently one of the strongest engines, thanks to community contributions and cutting-edge techniques in search and evaluation.
- **Houdini** (2010)
  - Algorithm: Based on Stockfish and other engines, with unique evaluation and search improvements.
  - Known for its tactical strength and practical playing style, briefly became the top engine before Stockfish regained the lead.
- **Komodo** (2010)
  - Algorithm: Focus on evaluation function accuracy using alpha-beta pruning.
  - This strong engine is known for its positional play and gradual improvements in evaluation techniques and search strategies.
- **AlphaZero** (2017)
  - Algorithm: Deep neural networks, Monte Carlo Tree Search (MCTS), and self-play reinforcement learning.
  - Developed by DeepMind, AlphaZero revolutionized chess engines by learning from scratch through self-play, outperforming traditional engines like Stockfish with innovative, human-like strategies.

Note For those interested, the GitHub repository https://github.com/mdoege/PyTuroChamp contains the Python implementation of Alan Turing's TUROCHAMP (1950), John Maynard Smith's SOMA (1961), The Bernstein Chess Program (1957), Leonardo Torres y Quevedo's El Ajedrecista (1912), and some other related engines.

15.2.1. A Note on the Chess Algorithms

The general goal of a chess algorithm is to evaluate the possible moves starting from a certain game position. This evaluation (analysis) aims to identify the best next move in the game.

Chess algorithms are not strictly related to the whole game evolution; at any game position, the process can analyze the best move, regardless of the previous game strategy. Thanks to this approach, chess algorithms can be used in several ways:

1. Play an entire game, giving the best response to any opponent's move. It doesn't matter if the opponent is a human player, another algorithm, or the algorithm itself called alternatively from the white and black positions.
2. Based on a certain game position, analyze the game to find or suggest the best next move. It is also possible to understand the process made by humans.
3. Analyze a game strategy to find all alternative moves and identify the weak points of the losing side.

Algorithms approach the Moves Tree analysis considering every move as a node, to which they assign a rating depending on how well the node move drives the player to victory or gains a better position on the chessboard.

Alpha-Beta Pruning

This algorithm is based on the Min-Max algorithm, which aims to reduce the number of nodes in the search tree of all possible moves and eliminate inefficient nodes (pruning the tree).

This algorithm reduces the number of nodes evaluated in the search tree, allowing for a deeper search and improving the quality of the decision to make the best move. It is used in many chess algorithms; in fact, the complete chess algorithm usually applies multiple basic algorithms for the parts of the game evaluation.

Bitboard Algorithm

Alpha-beta pruning efficiently searches the tree for all possible next moves; it is frequently used in conjunction with the Bitboard algorithm.

It is a technique for representing the chessboard and its pieces using bitwise operations on binary numbers. This method allows for efficient computation and manipulation of the board state for faster move generation and evaluation.

Evaluation Algorithms

While tree search and optimization algorithms speed up the selection of the next possible moves, the element that differentiates computer chess players is how the possible moves are rated and how efficiently a certain computer chess player can apply the game strategy.

On the contrary, the evaluation algorithms control the tree search and the optimization algorithms according to the specific strategy adopted by the chess engines.

Chess Engines

A chess engine is the ensemble of algorithms that enable a chess program to play the game.

GitHub hosts many open source chess engines. Here is a comprehensive list of the most popular engines:

- **Stockfish**: One of the strongest and most popular open source chess engines. (https://github.com/official-stockfish/Stockfish)
- **Leela Chess Zero** (LCZero): An open source engine that uses neural networks and reinforcement learning. (https://github.com/LeelaChessZero/lc0)
- **Fairy-Stockfish**: A variant of Stockfish that supports many chess variants, making it versatile for non-standard chess games. (https://github.com/ianfab/Fairy-Stockfish)
- **Ethereal**: A strong, open source chess engine known for its competitive strength and efficient coding. (https://github.com/AndyGrant/Ethereal)
- **Cfish**: A fast and lightweight version of Stockfish written in C, suitable for systems where performance is critical. (https://github.com/syzygy1/Cfish)
- **Texel**: An open source chess engine focusing on evaluation function research and development. (https://github.com/peterosterlund2/texel)

- **Xiphos**: An open source UCI chess engine that is competitive in strength and written in C. (https://github.com/michaeldiggs/xiphos)
- **GNU Chess**: One of the oldest open source chess engines, serving as a base for many other projects. (https://github.com/gnu-chess/gnuchess)
- **ChessEngine:** A simple, open source chess engine written in Python, suitable for educational purposes and development. (https://github.com/Zeyu-Li/ChessEngine)

15.2.2. Interfacing Chess Computers and Humans

The engines mentioned in the previous section, mostly developed in C++, can run as a terminal command—better when using Linux platforms—passing a series of parameters.

To make the chess engines "playable" by humans, an interface is needed. For example, humans need to control the validity of moves, represent the game chessboard after every move, or simply log the history of the game move after move.

For this reason, the engines evolved to be controlled similarly, accepting moves and control parameters through a standardized protocol supported by many open source interface applications.

Indeed, this application model offers plenty of possibilities; the same chess player interface can also work with different engines, which can be installed separately, like plugins.

Here is a list of the most interesting chess player user interfaces available on GitHub as open source projects:

- **Arena Chess GUI**: A powerful chess interface that supports multiple engines and allows analysis and play. (http://www.playwitharena.de/)
- **XBoard/WinBoard**: A versatile graphical user interface for chess that supports many engines and chess variants. (https://github.com/gnome-terminator/xboard)
- **PyChess**: A GTK chess client for Linux with a clean interface that supports various chess engines. (https://github.com/pychess/pychess)
- **Cute Chess**: A cross-platform chess interface and tool for running engine tournaments and matches. (https://github.com/cutechess/cutechess)
- **ChessX**: A cross-platform chess database application for managing and analyzing chess games. (https://github.com/chessx/chessx)
- **Lucas Chess**: A chess training program with various difficulty levels and features to play against different engines. (https://github.com/lukasmonk/lucaschessR)
- **Tarrasch Chess GUI**: A simple and user-friendly chess interface that supports UCI engines. (https://github.com/billforsternz/tarrasch-chess-gui)
- **LiChess** (`Lichess.org`): An open source online chess platform that provides a full suite of chess-related activities. (https://github.com/ornicar/lila)
- **Chess Trainer**: An educational chess GUI with training features that support various chess engines. (https://github.com/marcusbuffett/chess-trainer)

The Universal Chess Interface (UCI) Notation

The Universal Chess Interface (UCI) is a protocol defined to simplify the communication between chess engines and graphical user interfaces. This protocol includes more than the simple moves representation; it also supports a series of parameters to instruct the chess engine how to operate, when to start and stop the analysis of the moves nodes tree, and more.

UCI provides a standardized way for engines and interfaces to interact, ensuring compatibility across different platforms and software. This approach makes it possible, as mentioned, to use the chess engines as plugins by the game-playing chess interfaces.

It employs a simple command-response structure to set up positions, manage game statuses, and communicate moves.

The UCI protocol provides efficient and flexible interaction between chess engines and interfaces, facilitating features like move analysis, game playing, and engine configuration (see Listing 15-1_.

Here is a list of the key commands of the UCI protocol:

- **uci:** Initializes the engine, prompting it to identify itself and its capabilities.
- **setoption name [option] value [value]:** Sets engine options.
- **ucinewgame:** Signals the start of a new game.
- **position [fen <FEN-string> | startpos] moves [move1 move2 ...]:** Sets the board position.
- **go [searchoptions]:** Starts the search for the best move.
- **stop**: Stops the search.
- **bestmove [move] [ponder move]:** Returns the best move found.

Listing 15-1. A UCI protocol Used to Exchange Information Between a Chess Engine and a Graphical User Interface

```
GUI: uci
Engine: id name MyEngine
Engine: id author MyName
Engine: uciok

GUI: setoption name Hash value 128
Engine: info option name Hash type spin default 1 min 1
max 1024

GUI: ucinewgame
GUI: position startpos moves e2e4 e7e5

GUI: go depth 20
Engine: info depth 1 score cp 20 nodes 10 nps 1000 time 10 pv
e2e4 e7e5
Engine: info depth 2 score cp 34 nodes 30 nps 1500 time 20 pv
e2e4 e7e5 g1f3

GUI: stop
Engine: bestmove g1f3
```

Note A bare metal definition of the UCI protocol can be found in the GitHub Gist link at https://gist.github.com/DOBRO/2592c6dad754ba67e6dcaec8c90165bf, while a complete description can be found in the Chess Programming Wiki at https://www.chessprogramming.org/UCI. The Python `python-chess` library also includes the UCI definition, which is easy to use for Python-based user interfaces with chess engines (https://python-chess.readthedocs.io/en/v0.25.0/uci.html).

15.2.3. A Move Representation Method

You need a notation that represents the game moves regardless of how the physical chess engine communicates.

In chess, every move should be represented in a single line of text describing the “from" and “to” positions of the moved piece and some extra information like mate, *en-passant*, and so on.

The need to note the standardized chess moves and track the moves of a game dates almost two centuries before the advent of the chess computer. A structured notation appeared for the first time in 1737 in the book, *Essay on the Game of Chess* by Philip Stamma.

The representation of every tile on the board uses a system of coordinates. Every row is numbered from one to eight, where the first row corresponds to the first left white rook, and every column has the letters from a to h from bottom to top—where the bottom is the first row of the white pieces.

According to this board representation, several move notations have been developed in the following decades. The evolution of computer chess engines required a formal model that was easily managed by computer systems and understandable by human players.

This is a comprehensive list of the most popular chess game notations still in use:

- **Forsyth-Edwards Notation (FEN):** Describes a specific board position at any point in a game on a single line of text with six fields separated by spaces. The piece placement defines the location of all pieces from the 8th rank to the 1st rank, using characters (e.g., r for black rook, P for white pawn, etc.). This notation also includes in the single line the active color (w for white and b for black), castling availability, and all the other chess information that fully describes a position of the game (en-passant target square, move number).

- **Portable Game Notation (PGN):** A standard plain-text format for recording chess games, including move sequences and game metadata. At the end of the move, the game result is shown (e.g., 1-0 for a white win, 0-1 for a black win, 1/2-1/2 for a draw).
- **Algebraic Notation:** This is the standard method for recording and describing the moves in a chess game. Letters like K for King, Q for Queen, and P for Pawn identify the pieces. Uppercase letters are for white pieces, and lowercase letters are for black pieces.
- **Standard Algebraic Notation (SAN):** Another standard for recording chess moves, and it is used mainly for game annotations. It includes details like check, checkmate, capture, and piece promotion.
- **Long Algebraic Notation:** A more detailed version of algebraic notation, including starting and ending squares for each move.

15.2.4. The Arduino Chess Moves

Theoretically, the best way to represent the game on the Arduino UNO R4 boards is through the UCI protocol. However, considering the limitations of the Arduino board, adopting the complete protocol will be redundant. In fact, this software version does not consider the majority of the features of the UCI protocol.

Indeed, you should be able to manage the whole game in which the board chess engine interacts with a human player. The PGN notation is sufficient to achieve this task for both sides—machine and human players.

In the PGN notation, the moves are represented in the format

{piece name}{ coordinates from} {piece name}{coordinates to}

Example:

1. *e4 e5*
2. *Nf3 Nc6*
3. *Bb5 Ba6*

(Ruy Lopez opening)

This representation is ideal for saving string space: only the current move is described. The game always starts from the initial chess position, and you need to keep track of every move to reach any point of the game until the last move.

If you want the program to load a saved game to see a certain position, the moves should be replayed from the start to that point. Also, saving a game in simple text format is more compact. In addition, there are some game global parameters stored in the form of metadata on the top of the moves list for a better contextualization of the game.

Metadata:

[Event "Event name"]

[Site "Place of the game"]

[Date "yyyy.mm.dd"]

[Round "Number of moves"]

[White "Player name"]

[Black "Black name"]

[Result "Game result"]

Moves list

1 ...

2 ...

3 ...

15.2.5. The Arduino Chess Engine

This section starts with a short chronology of the chess engine.

1975: Rockwell 6502

In 1975, the Rockwell MOS technology 6502 was released; it was a revolutionary 8-bit microprocessor.

Its low cost and high performance made it the heart of many early home computers and gaming consoles, like Apple I and II, Commodore 64, Atari 2600, and the Nintendo Entertainment System (NES).

The 6502's affordability and simplicity made it a pivotal component in the personal computer revolution.

1976: Zilog Z80

The Z80 was another influential 8-bit microprocessor compatible with the Intel 8080.

It offered enhanced features and performance, and it was used in home computers like the Sinclair ZX Spectrum, the Tandy TRS-80, and many early arcade systems.

The Z80 also became popular in embedded systems and peripherals, where its robust instructions-set and ease of use were highly valued.

The First Personal Computers Age

Among the iconic computers, mostly oriented to the end-user offering and the ability to program with the popular BASIC language, a couple of 6502-based platforms went to the market dedicated to those we can call the first makers.

I refer to the Rockwell AIM-65 and the KIM-1: Two bare metal computer platforms open to learning and developing hardware interfacing and low-level computer architecture, including the machine code and assembler.

KIM-1 was the first piece of hardware available at a very low price, and it was ideal for deep programming. Regardless of the difficulty, programming in assembler and machine code was a way to get the highest performance and best usage of the—still limited—processing power and memory resources. This was when software was saved on audio tape, and many user interfaces we are used to today were yet to be developed.

What during the 70s represented technological advances, today we call retro-computing. The popularity gained by the R-6502 survived the evolution: A simple Arduino UNO has sufficient computational power to be able to support a full emulation of this microcontroller (https://forum.arduino.cc/t/arduino-6502-emulator-basic-interpreter/188328). Another complete 6502 processor emulation developed for both Arduino UNO and any other computer is available on GitHub at https://github.com/goncrust/arduino-6502.

Due to its simple interface, limited to a hexadecimal keyboard and a seven-segment LED display, a full working version of KIM-1 is available as an open source project. This project (https://obsolescence.wixsite.com/obsolescence/kim-uno-details) includes the components to design a PCB emulating the original KIM-1 keyboard and display. It is originally based on an Arduino Pro Mini but can be easily ported to other Arduino board (UNO, Mega, and others).

The Work of Peter Jennings

Peter Jennings's notable work started when he bought a KIM-1 board, probably the cheapest microcomputer for personal use at the time.

When a chess lover meets a computer, the prediction is almost obvious. Most of his knowledge of chess strategy derived from reading *My System* by Aron Nimzovich. Jennings started developing Microchess until, on December 18, 1976, the first copy of the complete chess game on the KIM-1 was shipped to his first customer.

Microchess was also the first computer game sold for home computers. It was a full chess program fitted in 924 bytes of 6502 code! You can find the full story on the author's site at https://www.benlo.com/microchess/index.html.

Thinking of playing chess on an Arduino UNO R4 Microchess is an obliged choice.

The Microchess Porting

There are several open source projects for Arduino platforms based on the porting of Microchess; some are complete projects, and others are works in progress. I started from the Diego Cueva project for Arduino Mega (www.diego-cueva.com/projects/chessuino/), originally created to work with a beeper and an alphanumeric LCD display.

The Microchess engine has been separated from the rest of the code, so I arranged for it to work with the Arduino UNO R4 integrated with a terminal TTY interface and created the web server for remote connection.

This way, you can play chess with Arduino from any remote Telnet connection via WiFi. I also arranged for a better, text-only representation of all the moves. After every move—passed in PGN notation—a small chessboard is redrawn on the terminal for a more comfortable user interface.

15.3. Arduino Chess Software

This version of Arduino Chess is tailored to work using all the available resources of the UNO R4 WiFi. The LED matrix—not present on the MINIMA version—can be enabled or not.

Of course, the availability of a more powerful microcontroller gives developers space to add a more structured architecture and extend the capabilities of the original Microchess (without changing the min-max algorithm and the alpha-beta pruning), thus increasing the number of moves to explore (Moves Tree exploration).

To make the application as adaptable as possible, including running a version on the MINIMA board, a series of header files have been defined to keep the code blocks reusable without impacting the application architecture.

15.3.1. The Header Files

When running on the WiFi model, an LED matrix is available. Using the simple HTML editor available on the Arduino site (which can also run locally), the from-to moves of the board are shown in coordinates format in Figure 15-5.

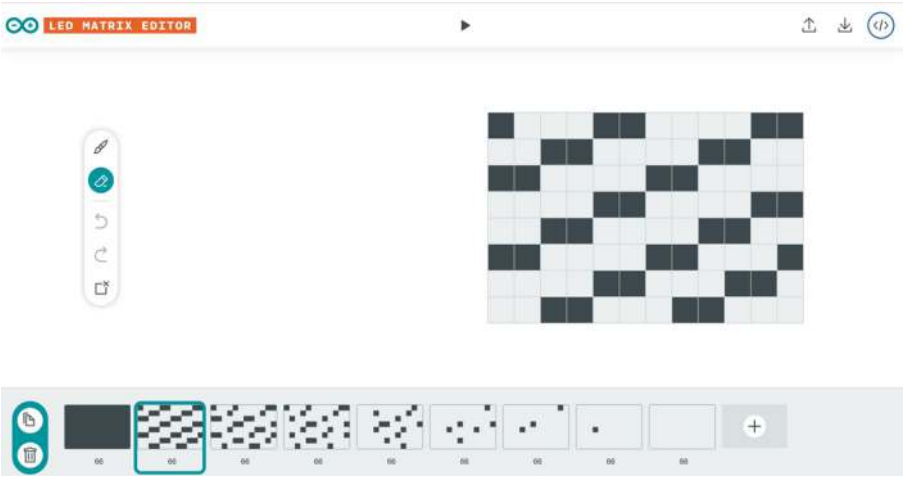

Figure 15-5. *The HTML screen of the LED matrix editor. The patterns designed on the rectangle can be exported in "mpj" JSON format, which is easy to convert to an array of values. Multiple patterns can be designed and exported together to create simple animations. Exported projects can be reloaded in the editor for further changes.*

The MatrixChars.h File

The patterns for the chessboard coordinates were converted to arrays in the `MatrixChars.h` header file.

```
/**
\file MatrixChars.h
\brief Preprocessor definitions and byte arrays to manage the
UI on the Led Matrix (Arduino Uno R4 only)

\author Enrico Miglino <enrico.miglino@gmail.com>
\version 1.0
\date July 2023
*/
#include <memory>
```

```
//! Number of characters in a move
#define MOVE_CHARACTERS 2

//! Arduino Uno R4 LED matrix array width
#define MATRIX_ARRAY_WIDTH 12
//! Arduino Uno R4 LED matrix array height
#define MATRIX_ARRAY_HEIGHT 8

// Offset from char 'a'
#define ASCII_A_LC 97
// Offset from char '0'
#define ASCII_ZERO_LC 49
// Character size
#define ASCII_CHAR_WIDTH 5
// "to" string width
#define ASCII_TO_WIDTH 8

#define DISPLAY_DELAY 1000

// LED matrix starting column for the four move characters
// The first two positions are for the move characters and the
// third is the "to" string starting position.
int START_MOVE_CHARACTER_FROM[3] = {0, 6, 2};

byte frameLogo[8][12] = {
  {0,0,0,0,1,0,1,0,0,0,0,0},
  {0,0,1,0,0,1,0,0,1,0,0,0},
  {0,1,0,1,0,1,0,1,0,1,0,0},
  {0,1,0,0,1,1,1,0,0,1,0,0},
  {0,0,1,0,0,1,0,0,1,0,0,0},
  {0,0,0,1,1,1,1,1,0,0,0,0},
  {0,0,0,0,1,1,1,0,0,0,0,0},
  {0,0,0,1,1,1,1,1,0,0,0,0}
};
```

```
// LED matrix frame definition
byte frame[8][12] = {
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
};

// definition of the frame to animate the piece move (piece
from - arrow - piece to)
byte frameArrow[8][8] = {
  {0,0,0,0,0,0,0,0},
  {1,0,0,0,0,0,0,0},
  {1,1,1,0,0,0,0,0},
  {1,0,0,0,0,1,1,0},
  {1,0,0,0,1,0,0,1},
  {1,0,0,0,1,0,0,1},
  {0,1,1,0,0,1,1,0},
  {0,0,0,0,0,0,0,0}
};

// Characters definition
byte frameLowercase[8][8][5] = {
  { // a
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {0, 0, 1, 1, 0},
    {0, 1, 0, 0, 1},
```

```
    {0, 1, 1, 1, 1},
    {0, 1, 0, 0, 1},
    {0, 1, 0, 0, 1},
    {0, 0, 0, 0, 0}
  },
```

Next, definitions of the lowercase characters a-h

```
byte frameNumbers[8][8][5] = {
  { // 1
    {0, 0, 0, 0, 0},
    {0, 0, 0, 1, 0},
    {0, 0, 1, 1, 0},
    {0, 0, 0, 1, 0},
    {0, 0, 0, 1, 0},
    {0, 0, 0, 1, 0},
    {0, 0, 1, 1, 1},
    {0, 0, 0, 0, 0}
  },
```

Next, definitions of the numbers 1-8, then the uppercase characters A-H

The ChessMessages.h File

This header contains all the string definitions shown on the terminal during the game. Every string character consumes a byte of memory. It is strongly suggested to keep the string definitions as short as possible, eventually improving the application verbosity as the last task after testing and debugging.

```
/**
\file ChessMessages.h
\brief Preprocessor definitions of the application errors and
message strings.
```

```
Defining the messages and error Strings makes it easy to
localize the version of the program, and eventually reduce the
string length if memory space is needed.

\note The SSID and WPA passwords are hardcoded. When you change
the network access to the program should be recompiled.

\author Enrico Miglino <enrico.miglino@gmail.com>
\version 1.0
\date July 2023
*/

// --------------
// Messages
// --------------
#define MSG_TITLE "*** CHESS ! ***"
#define MSG_SSID "Connecting to SSID: "

// --------------
// Errors
// --------------
#define ERR_001 "E001 Communication with WiFi module failed!"
#define ERR_002 "E002 Please upgrade the firmware"

// --------------
// Debug
// --------------
#define DBG_CLIENT_CHARS "(debug) Chars available from the
client: "
#define DBG_GOT_CLIENT_MOVE "(debug) Received the move from
the client"
```

The WiFiAccess.h File

The `WiFiAccess.h` header defines the default access point and WPA password to connect the board to the WiFi. These `defines` should be configured according to your WiFi settings.

Thinking about a version running standalone, to avoid recompiling when the WiFi parameters change, you can add a serial terminal menu where the two values can be changed and saved on the Arduino board Eprom. If this value is present, the custom settings will overwrite the default defined in this header.

```
/**
\file WiFiAccess.h
\brief Preprocessor definitions for WiFi connection parameters.

\note The SSID and WPA passwords are hardcoded. When you change
the network access to the program should be recompiled.

\author Enrico Miglino <enrico.miglino@gmail.com>
\version 1.0
\date July 2023
*/

//! Network WPA Access parameters
// #define SECRET_SSID "EnricoA71"
//! WPA connection password
// #define SECRET_PASS "cagliostro"

//! Network WPA Access parameters
#define SECRET_SSID "AccessPoint"
//! WPA connection password
#define SECRET_PASS "WPA Password"
```

```
//! Telnet server port
#define SERVER_PORT 23

//! Connection timeout
#define WIFI_CONN_TIMEOUT 10000
```

The ChessEngine.h File

The `ChessEngine.h` header defines all the parameters that control the game engine to make it more readable. It is possible to customize the engine game side, the randomization seed, and other control parameters.

```
/**
\file ChessEngine.h
\brief Preprocessor definitions of the chess engine constants.

\author Enrico Miglino <enrico.miglino@gmail.com>
\version 1.0
\date July 2023
*/

//! The chess symbols used by the text-only board
#define CHESS_PIECES ".?pnkbrq?P?NKBRQ"
//! Number of pieces symbols
#define CHESS_PIECES_NUMBER 17

//! 16bit pseudo-random generator
#define RANDOMIZER_MAX 65535

//! Computer engine moving side (usually black=16)
#define MOVING_SIDE 16

//! bytes (chars) size of the arrays storing the move
//! Moves are in the format from_xy to_xy where xy are
//! the row coordinates on the chessboard.\n
```

```
//! a >= x <= h and 1 >= y <= 8
//! The fifth character of the array is the Cr endstring
    character.
#define MOVE_SIZE 5

// Engine macros
#define W while
#define M 0x88
#define S 128
#define I 8000
```

The messages defined in this header are the hardcoded strings to draw the game status chessboard on the terminal:

```
//! Telnet welcome message
#define TLN_WELCOME "Welcome to ChessTelnet"

// Strings to build the text chessboard
#define BOARD_HOR_LINE "+-----------------+"
#define BOARD_SPACING " "
#define BOARD_SPACING2 "  "
#define BOARD_SEPARATOR "|"
#define BOARD_COLUMNS "  a b c d e f g h"

// -------------- Telnet messages
#define CHESS_LOSE "Engine lose"
#define CHESS_PLAYER_LOSE "You lost!"
#define CHESS_MOVE_INVALID "Invalid move!"
```

15.3.2. The Application Functions

The program requires a WiFi connection to start the game. I split the utility functions from the chess engine, and the same engine is now a function of the main source.

Here is an example of the terminal output of the first three moves of a game:

```
Welcome to ChessTelnet
  +-----------------+
 8| r n b q k b n r |
 7| p p p p p p p p |
 6| . . . . . . . . |
 5| . . . . . . . . |
 4| . . . . . . . . |
 3| . . . . . . . . |
 2| P P P P P P P P |
 1| R N B Q K B N R |
  +-----------------+
    a b c d e f g h
d2d4
1. d2d4b8c6
  +-----------------+
 8| r . b q k b n r |
 7| p p p p p p p p |
 6| . . n . . . . . |
 5| . . . . . . . . |
 4| . . . P . . . . |
 3| . . . . . . . . |
 2| P P P . P P P P |
 1| R N B Q K B N R |
  +-----------------+
    a b c d e f g h
b1c3
```

```
2. b1c3e7e5
  +-----------------+
 8| r . b q k b n r |
 7| p p p p . p p p |
 6| . . n . . . . . |
 5| . . . . p . . . |
 4| . . . P . . . . |
 3| . . N . . . . . |
 2| P P P . P P P P |
 1| R . B Q K B N R |
  +-----------------+
    a b c d e f g h
g1f3
3. g1f3f7f6
  +-----------------+
 8| r . b q k b n r |
 7| p p p p . . p p |
 6| . . n . . p . . |
 5| . . . . p . . . |
 4| . . . P . . . . |
 3| . . N . . N . . |
 2| P P P . P P P P |
 1| R . B Q K B . R |
  +-----------------+
    a b c d e f g h
```

The Setup() File

```
// Initialization
void setup() {
  matrix.begin();
```

```
Serial.begin(9600);

serialMessage(MSG_TITLE, true);
lastH[0] = 0;

// ----------------------- Network connection
// check for the WiFi module:
if (WiFi.status() == WL_NO_MODULE) {
  serialMessage(ERR_001, true);
  // don't continue
  while (true)
    ;
}  // Check for WiFi module

//! Get the firmware version and validate
String fv = WiFi.firmwareVersion();
if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
  serialMessage(ERR_002, true);
}

// Attempt to connect to WiFi network:
while (status != WL_CONNECTED) {
  serialMessage(stringJoinString(MSG_SSID, ssid), true);

  //! Connect to WPA/WPA2 network.
  //! Change this line if using open or WEP network:
  status = WiFi.begin(ssid, pass);

  // Wait for connection:
  delay(WIFI_CONN_TIMEOUT);
}  // Connection to server

// Start the server:
server.begin();
// Client is connected now, so print out the status:
```

```
  printWifiStatus();
  matrix.renderBitmap(frameLogo, 8, 12);
  // -------------------- Network connection
}
```

The Loop() File

```
// Main loop
void loop() {
  //! Wait for a new telnet client connected
  client = server.available();

  // When the client is connected the game can start
  if (client) {
    if (!alreadyConnected) {
      // clear out the input buffer:
      client.flush();
      serialMessage(MSG_TITLE, true);
      client.println(TLN_WELCOME);
      // Show the board for game start
      sendWiFiBoard();
      // Set the flat for the game continuing.
      alreadyConnected = true;
    }  // Already connected

    int r;

    // Take move from human (four characters + newline)
    while (stringComplete == false) {
      getClientChar();
    }  // Retrieve characters from the client
```

```
// Print the move
client.print(mn);
client.print(". ");
client.print(inputString.substring(0, 4));

// Build the move for process
c[0] = inputString.charAt(0);
c[1] = inputString.charAt(1);
c[2] = inputString.charAt(2);
c[3] = inputString.charAt(3);
c[4] = 0;

// Clear the input string:
inputString = "";

// Initialize the move string status
stringComplete = false;

// Parse the client move
K = *c - 16 * c[1] + 799, L = c[2] - 16 * c[3] + 799;
N = 0;
T = 0x3F;                 /* T=Computer Play strength */
bkp();                    /* Save the board just in case */
r = D(-I, I, Q, 0, 1, 3); /* Check & do the human
                             movement */

if (!(r > -I + 1)) {
  client.println(CHESS_LOSE);
  gameOver();
}  // Game ends

if (k == 0x10) { /* The flag turn must change to 0x08 */
  client.println(CHESS_MOVE_INVALID);
  return;
```

```
    }  // Wrong move

    strcpy(lastH, c); /* Valid human movement */

    mn++; /* Next move */

    K = I;
    N = 0;
    T = 0x3F;                  /* T=Computer Play strength */
    r = D(-I, I, Q, O, 1, 3); /* Think & do*/

    if (!(r > -I + 1)) {
      client.println(CHESS_PLAYER_LOSE);
      gameOver();
    }  // Last move, game ends

    strcpy(lastM, c); /* Valid ARDUINO movement */
    r = D(-I, I, Q, O, 1, 3);

    if (!(r > -I + 1)) {
      client.println(lastM);
      gameOver();
    }  // Last move, game ends

    client.println(lastM);
    showMoveOnLED();
    sendWiFiBoard();
  }  // If client
}  // End loop
```

The complete source code and other components part of the application are available in the chapter repository.

CHAPTER 16

Chess Player Interfaces

Contribution of Luis Garcia.

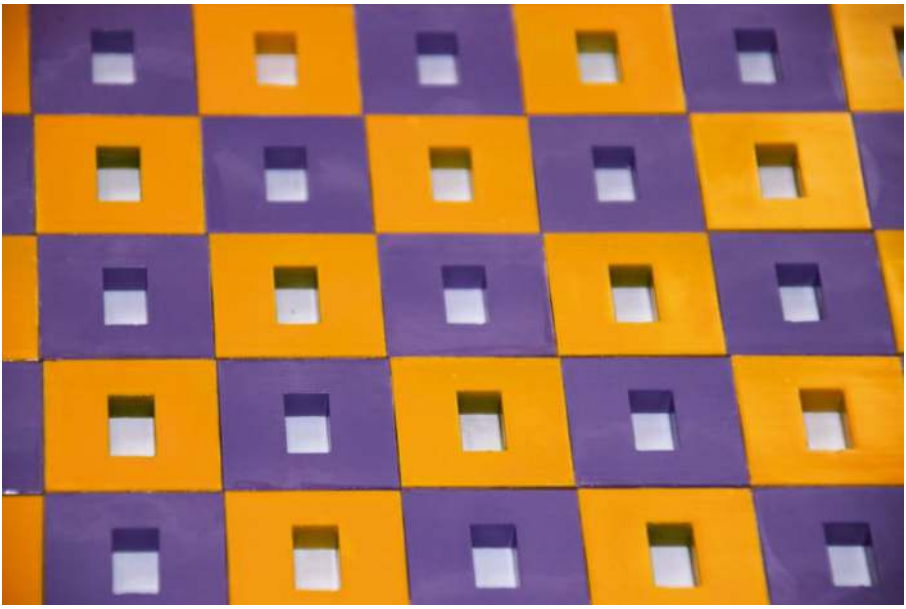

Figure 16-1. *A checkerboard with rectangular holes?*

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_16

I started thinking of a chess interface before designing a computer chess player based on the Arduino UNO R4. The original idea was to create a couple of chess boards connected through WiFi using two WiFi Arduino MKR 1010 to enable players to manage the game from different locations. Part of this idea was to implement a chess engine on a Raspberry Pi that would be able to replace one of the two players (or both with two Raspberry). See Figure 16-1.

With the introduction of the Arduino UNO R4, the board's computational power made it interesting to implement a full working engine on the new Arduino, as described in the previous chapter.

Even though the board can play against humans through a text-based Telnet terminal, the idea of an engaging interface remained an interesting addition.

In this chapter, while I was developing the chess engine and the basic software for the Arduino UNO R4 WiFi (the most advanced model), Luis started adding some external hardware to the MINIMA model to achieve the same result.

You'll see how he reached this goal and a couple of interface designs adaptable to the Arduino UNO R4 chess engine, with some parts still in progress.

16.1. The MINIMA Board and the ESP32-S3

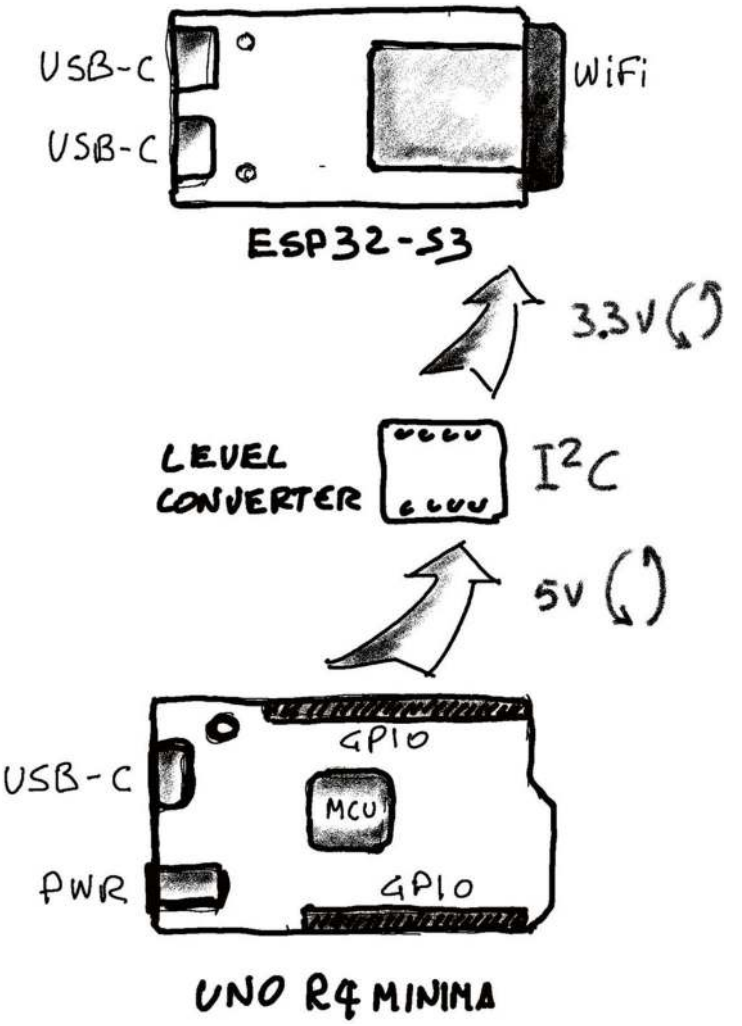

Figure 16-2. *Scheme of the connection between the Arduino UNO R4 MINIMA and the ESP32-S3*

The MINIMA model of the Arduino UNO R4 does not include the ESP 32 S3 (a powerful ARM microcontroller used in the WiFi model for connection and other internal features). We addressed this limitation by adding an external ESP32-S3 device facing a challenging issue on the power lines. See Figure 16-2.

The first—and easiest—way to exchange data between the MINIMA and an external ESP32-S3 is by using the UART. Everything went fine, but it was just for testing. We needed the UART to configure the board for the WiFi access point, WPA password, and dynamic IP address assigned by the router through the serial terminal.

The alternative and strategic solution for the inter-processor communication architecture is the I2C protocol.

I2C (Inter-Integrated Circuit), or IIC, is an extremely simple yet efficient way to share information between two boards or different components on the same board. It's a lightweight, high-speed protocol that requires minimal wiring.

Technically, it is not. It's not much different from how the Arduino UNO R4 WiFi board accomplishes this job. The main challenge was overcoming the different TTL voltage translations stably and reliably.

The I2C protocol is a bus that allows up to 127 slave devices to be connected to a master device. Any slave device can perform both input and output data transfers.

In our setup, the ESP32-S3 exposes a web server to which a terminal or interface application can connect, acting as the master, and the Arduino MINIMA is the slave.

We used breadboards and jumper cables instead of a custom PCB or a prototype soldering board to quickly prototype and test the system's viability. In this initial configuration, we faced instability issues, especially at a higher communication speed. Shorter cables and a lower transfer speed partially solved the problem. Also, when using a soldered prototyping board, we still met unstable performances in communication.

Not all voltage-level adapters work the same. We had to try several brands (using different ICs) until we found a good and cheap device that granted the requested reliability and performance (Teyleten Robot 4 Channels Logic Level Converter Bi-Directional Module Shifter I2C 3.3V—5V). See Figure 16-3.

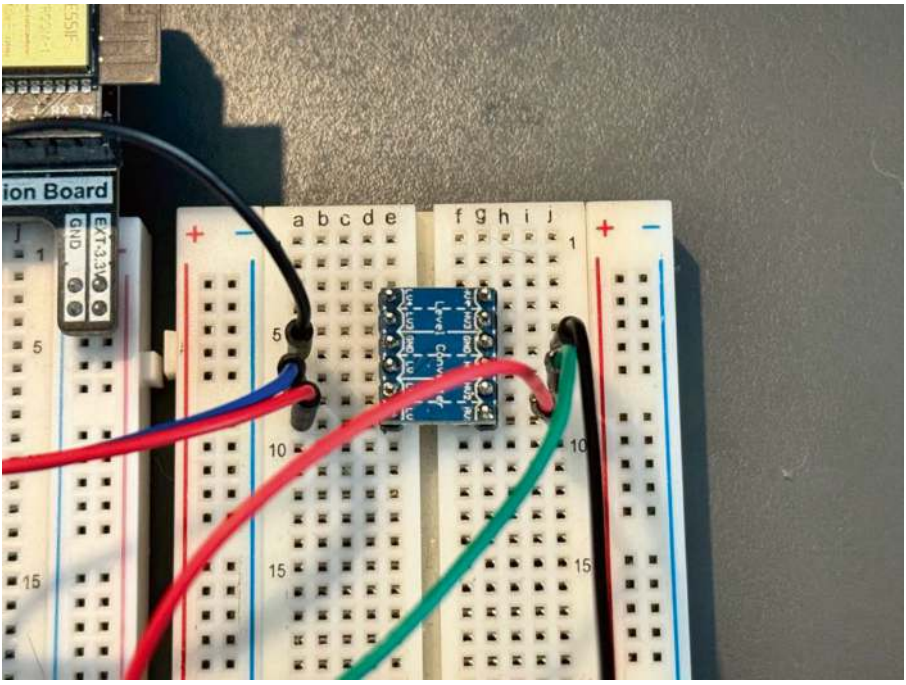

Figure 16-3. *The 5V-3.3V level shifter used to grant the boards compatibility in the I2C protocol*

16.1.1. Communication Software

The following code shows how, thanks to the libraries provided by the Arduino IDE, setting up the communication software is almost easy.

To connect a board as a slave to the I2C bus, you need to define the address that the I2C master device will use to communicate with this board. You also need to define two functions: one that will be called when data is available from the master and another one that will be called if the master is requesting data.

```
#include <Wire.h>
#define SLAVE_ADDRESS 0x20

void setup() {
  Wire.begin(SLAVE_ADDRESS); // Initialize as an I2C slave with
                                the defined address
  Wire.onReceive(receiveEvent); // Register an event handler
                                   for when data is received
  Wire.onRequest(requestEvent); // Register an event handler
                                   for when data is requested
                                   by a master

  Serial.begin(9600);

  // ... //
}
```

When set as master, the ESP32-S3 board does not connect to the I2C bus unless the used pins are not explicitly declared, even if they are the I2C pins hardware assigned, marked on the PCB board. Discovering this detail has been time-consuming and teaches a valuable lesson: thoughtfully test, test, and test before assuming something will work.

```
#include <Wire.h>
#define SDA_PIN 17
#define SCL_PIN 18

void setup() {
  // ... //

  Wire.begin(SDA_PIN, SCL_PIN); // Pins have to be specified on
                                   most boards

  // ... //
}
```

16.1.2. I2C Tasks Distribution

Arduino UNO R4 MINIMA is dedicated to running the chess engine, while the ESP32-S3 board—especially suited for IoT tasks—is focused on network scanning, storing credentials, and running the web server. See Figure 16-4.

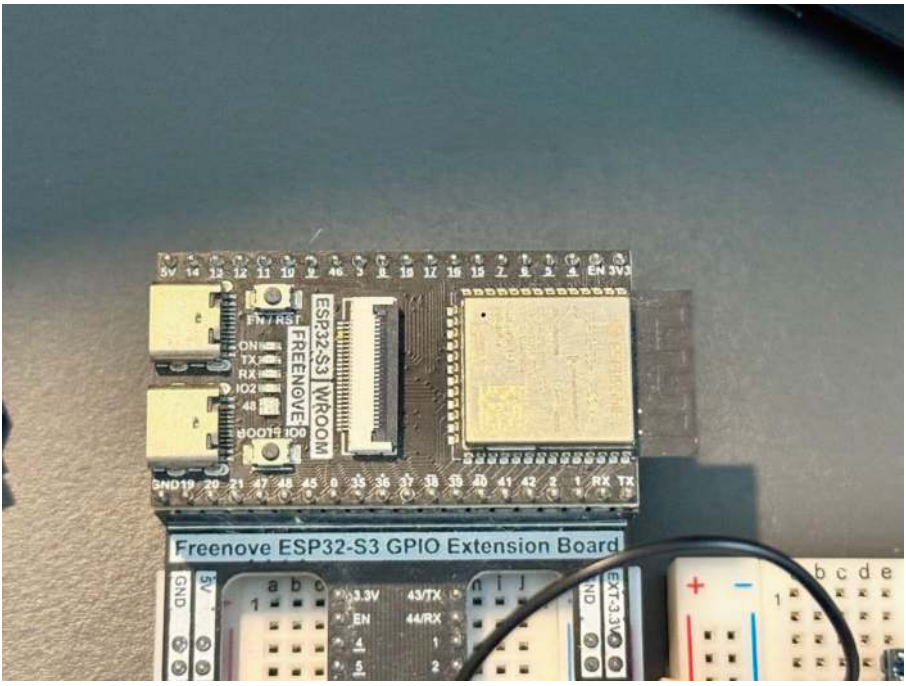

Figure 16-4. *The ESP32-S3 board connected to the MINIMA. For easier development and debugging, we used a development ESP32, including an on-board hardware debugger*

ESP32-S3 searches for stored network credentials at system boot and tries to connect to that network. If no saved credentials are found or the login is unsuccessful, it scans for available WiFi networks and presents a user interface for connection using Serial communication. See Figure 16-5.

Once the connection is successful, you get an URL with the IP address assigned to the ESP32-S3 through the Serial monitor, and the board starts up a web server.

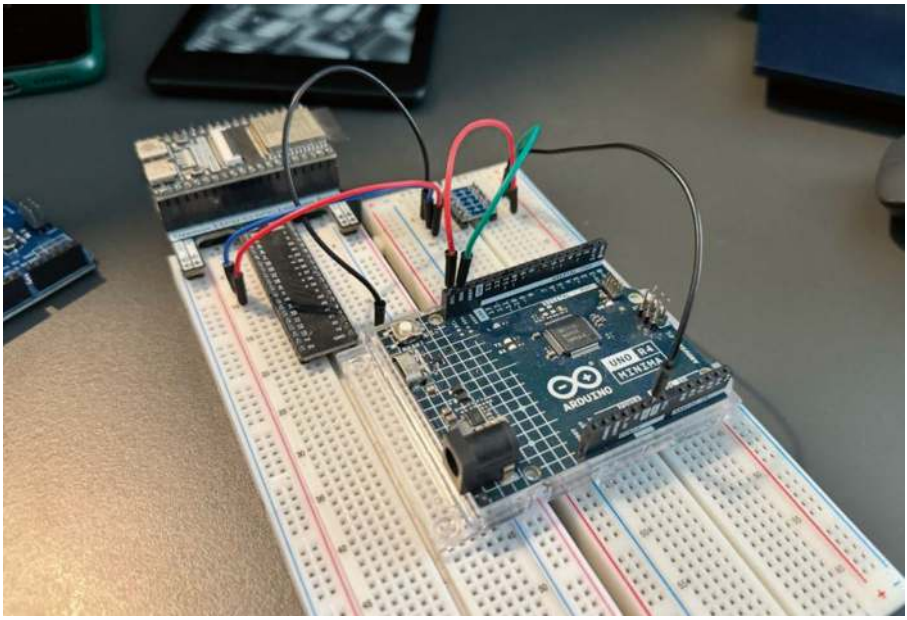

Figure 16-5. *The first breadboarded prototype of the Arduino UNO MINIMA linked to the ESP32-S3 via I2C protocol*

The board gets user input through the HTTP server, relays it to the Arduino MINIMA board, and processes the response. In this architecture, the ESP32-S3 board plays the role of a dumb terminal and offloads a computationally expensive task to another microcontroller.

At this point, the two boards wired together work the same way as the Arduino UNO R4 WiFi, excluding the LED matrix.

16.2. Physical Computing: the Distanced Pawn Project

This project is a general-purpose chess-playing interface. The chapter repository package includes all the project materials, including the STL files for 3D printing.

I was inspired by a detail that caught my attention the first time in the movie, "The Spy Who Come in from the Cold" (1965, from the John Le Carré novel), in a scene where spies meet in a public park amidst casual chess players, a typical Cold War espionage context.

By then, I had always desired to do something "techy." When I designed "The Distanced Pawn," I aimed to create a nice, general-purpose chess-playing interface that was easy for humans and machines to use.

16.2.1. Making the Chessboard

The Chess Pieces

The chess pieces have been 3D printed from the minimal STL files developed by FunFunBoy, which were published on Thingiverse (www.thingiverse.com/thing:2552392) and released under a CC 3.0 NC-ND-SA license. See Figure 16-6.

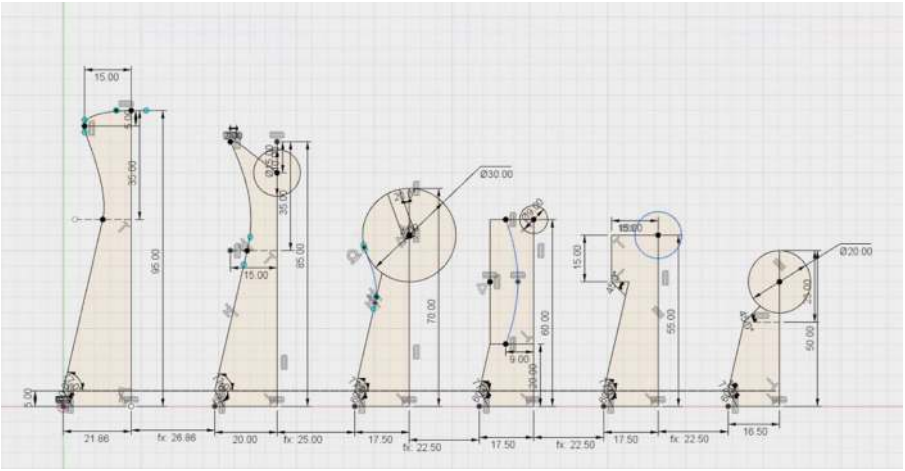

Figure 16-6. *The body design of the chess pieces developed by the author. To create the 3D models, the profile was converted to a rotation solid along the vertical axis.*

After properly scaling the models according to the size of the squares, which was about 4cm, I 3D-printed two sets of pieces and spray-painted them with the opponent's colors. See Figure 16-7.

Figure 16-7. *The pieces of 3D models were refined and painted with the two opponent colors. To achieve the best quality, I used a high-resolution resin 3D printer.*

The Checkerboard

Building the checkerboard was more complex because I needed special squares. An easy option was to buy and customize a commercial checkerboard, but then I decided to design every square and 3D print it. See Figure 16-8.

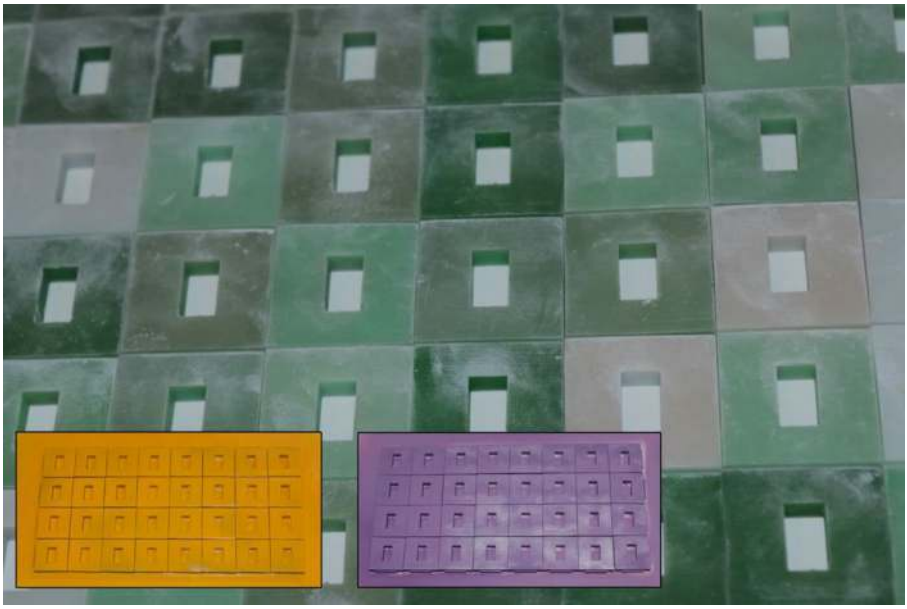

Figure 16-8. *The preassembled checkerboard is used to test that all the squared bricks fit well together, and the two series of 16 spray-painted squares in the opponents' color.*

The rectangular holes in every square will host a microswitch configured as a temporary pushbutton. To set a move, it is sufficient to press the piece on the foam square and push it on the destination square.

I designed every square with two extruded sides and two engraved sizes to improve stability when every square is glued together to compose the checkerboard. For this reason, I designed the square model myself. See Figure 16-9.

Figure 16-9. *The checkerboard tiles were glued after the final assembly. On the top left, the rendering of the tile 3D model shows the opposite sides with the extruded and engraved parts to keep them together. Of course, two identical checkerboards (and two full series of chess pieces) were required for the opponents.*

The design allows the tiles to be connected and glued on a larger surface, avoiding the risk of detaching during use. To avoid painting the wrong tiles, I assembled the board, separated it in the middle, and sprayed the right colors. After coloring, the board was assembled to glue the tiles with cyanoacrylate glue.

Before proceeding with the electronic components and the software, I glued a microswitch in the rectangular holes of the tiles. See Figure 16-10.

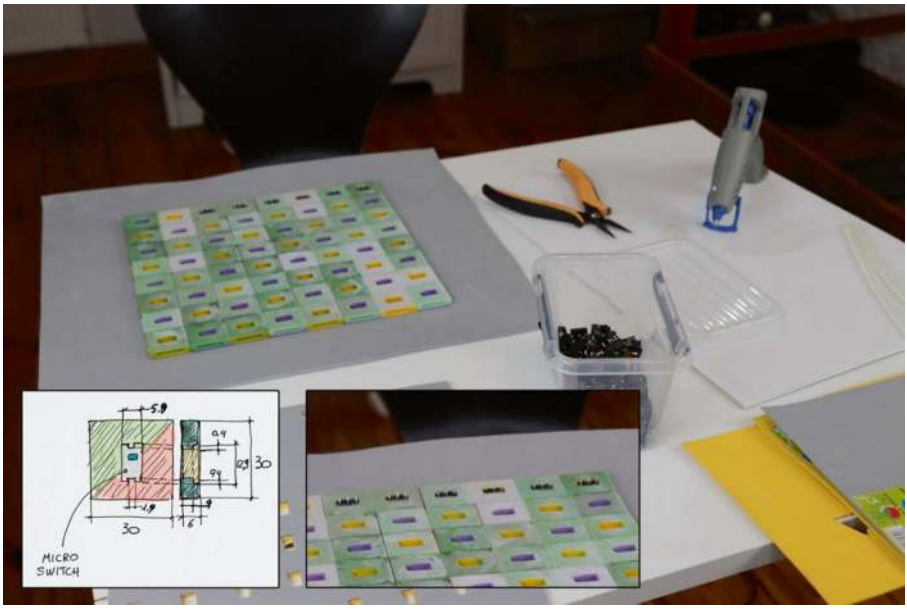

Figure 16-10. *The microswitch assembly to the bottom of the checkerboards. Every microswitch, when pressed, identifies the square with two signals corresponding to the row and column number.*

16.2.2. The Game Controller

Every opponent's chessboard has an 8x8 grid wiring, where every line-row cross identifies one of the 64 chessboard tiles. Every tile microswitch corresponds to the 64 tiles. These can be detected by 16 GPIO input pins, eight for every row and eight for every column.

The Circuit

The Arduino MKR 1010 does not have 16 GPIO available pins for this operation, so I added two I2C PCF8574 GPIO extenders chained together. See Figure 16-11.

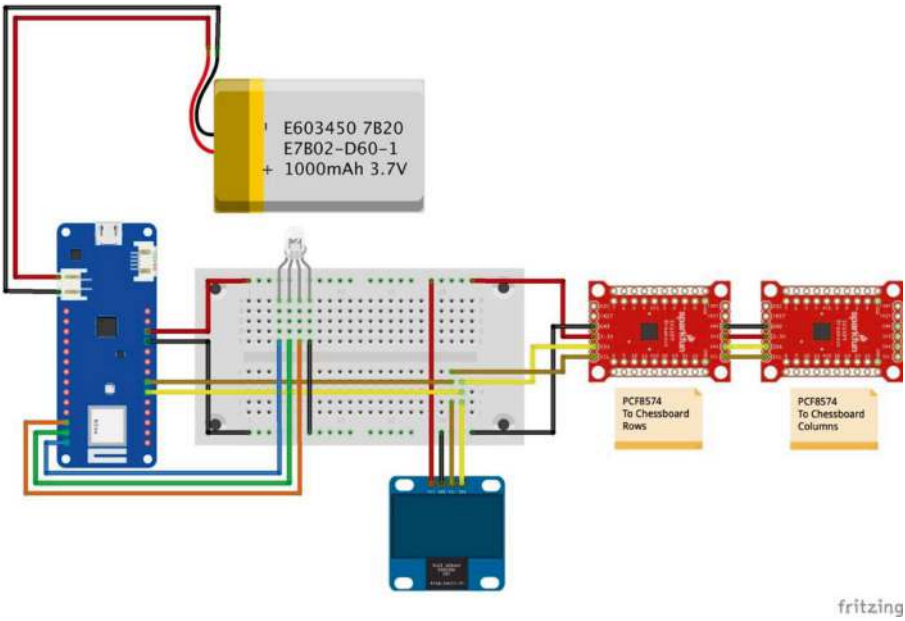

Figure 16-11. *Schematic of the game controller. I included an RGB LED to show the state of the game and a small monochrome OLED display to show the moves and the game messages (draw, chess, mate, etc.).*

The small GPIO expander board I2C has pins that can be hardwired to select the I2C bus address, supporting more than one expander chained together.

The board software can be configured to run as a web server or client with a hardcoded configuration to enable automatic communication. Adopting this solution, it is easy to configure the WiFi settings of the Arduino UNO R4 connected to one of the two board controllers, enabling a human-machine player.

According to the issues Luis previously experienced connecting the ESP32-S3 to the Arduino UNO R4 MINIMA via the I2C protocol, we tested the I2C prototype communication with an oscilloscope. See Figure 16-12.

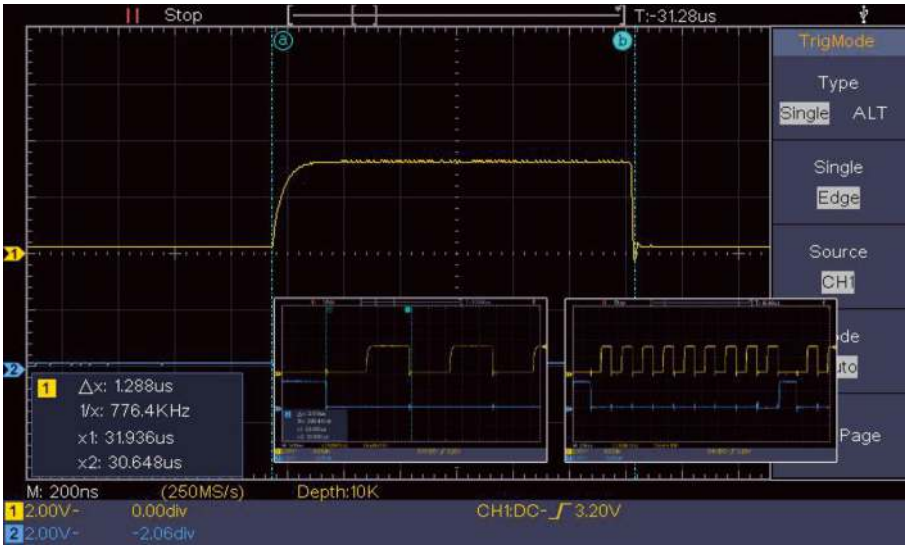

Figure 16-12. *Oscilloscope screenshots of the I2C signal stability tests*

In this case, both the MKR1010 and the PCF8574 GPIO expanders work at the same voltage. The critical issue in the I2C communication stability may depend on the relatively long wiring of the two expanders and the Arduino board.

With care in soldering the components, the circuit worked without problems. See Figure 16-13.

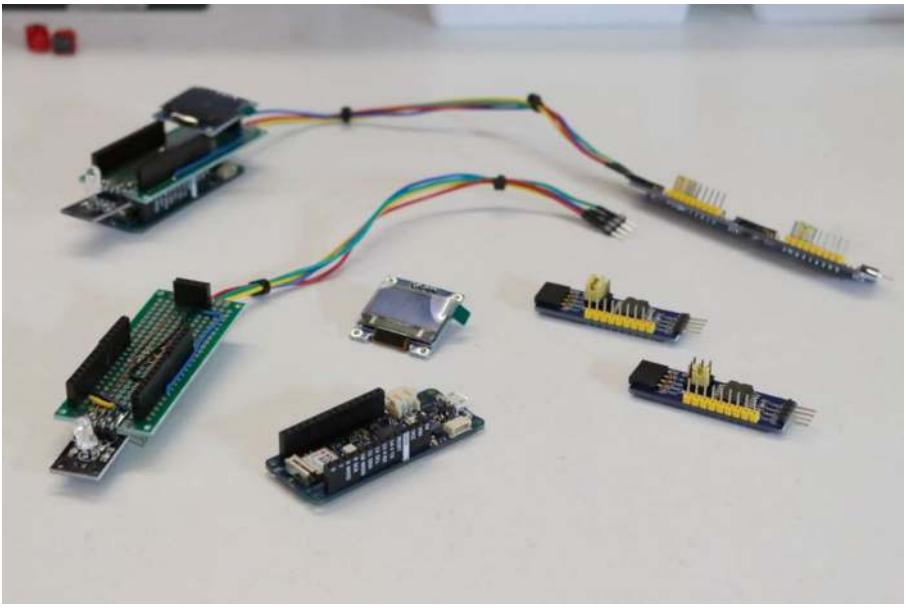

Figure 16-13. *The two completed board controllers are mounted over the Arduino MKR 1010 board GPIO connector*

16.2.3. The Controller Software

MKR 1010 Access Point

The first board that should be powered is the access point, to which the opponent player board will connected automatically.

Note The MKR 1010 access point is contacted by the client through three restful API: `/N` (new came), `/M` (move), and `/S` (game status). This makes it possible to also play with a single board and a remote web browser.

The server_params.h header file defines the WiFi connection constants. In this case, the board controllers connect one-to-one. There are no external parameters to be set and the IP address is hardcoded in the header definitions.

```
/**
 * \file server_params.h
 * \brief Global parameters header to implement the web server
   and the software AP
 */

//! AP SSID
#define SECRET_SSID "MKR1010"
//! Web password. Should be known by the remote client
#define SECRET_PASS "ChessMaster"

//! https custom server port
#define SERVER_PORT 8080

//! Delay before the AP can connect the WiFi (ms)
#define AP_DELAY 10000

//! Definition of the default AP IP address
inline int IP(int x) { int ip[] = {10, 0, 0, 1}; return
ip[x]; }

// HTTP GET commands
#define HTTPGET_NEWGAME     "/N"
#define HTTPGET_MOVE        "/M"
#define HTTPGET_STATUS      "/S"
```

The setup() function of the main program initializes the GPIO and RGB LED pins, then configures the local parameters to connect to the opposite player access point. This way, the controller client knows the IP address and port to connect to.

```
/**
 *  Initialization function.
 *
 *  In the setup function it is created the AP and assigned the
    default IP
 *  address, as well as the server creation. Only in debug mode
    (serial active)
 *  the setting operations are logged to the terminal.
 *
 *  \note To manage the status when the AP can't be initialized
    or there is a connection
 *  issue, the building LED goes not to On
*/
void setup() {

  pinMode(PIN_R, OUTPUT);
  pinMode(PIN_G, OUTPUT);
  pinMode(PIN_B, OUTPUT);

  digitalWrite(PIN_R,HIGH);
  digitalWrite(PIN_G,LOW);
  digitalWrite(PIN_B,LOW);
  delay(2000);
  digitalWrite(PIN_R,LOW);
  digitalWrite(PIN_G,HIGH);
  digitalWrite(PIN_B,LOW);
  delay(2000);
  digitalWrite(PIN_R,LOW);
  digitalWrite(PIN_G,LOW);
  digitalWrite(PIN_B,HIGH);
  delay(2000);
  digitalWrite(PIN_R,LOW);
```

```
  digitalWrite(PIN_G,LOW);
  digitalWrite(PIN_B,LOW);

  pinMode(LED_BUILTIN, OUTPUT);

#ifdef _DEBUG
  Serial1.begin(115200);
#endif
  sDebug("Access Point Web Server");

  // Check the firmware version and notify if it should
     be updated
  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
    sDebug("Please upgrade the firmware");
  }

   WiFi.config(IPAddress(IP(0), IP(1), IP(2), IP(3)) );

  // print the network name (SSID);
  sDebug("Creating access point named: ");
  sDebug(ssid);

  // Create open network. Change this line if you want to
     create an WEP network:
  status = WiFi.beginAP(ssid, pass);

  if (status != WL_AP_LISTENING) {
    sDebug("Creating access point failed");
    // don't continue
    while (true);
  }

  // wait for connection
  delay(AP_DELAY);
```

```
  // start the web server on the assigned port
  server.begin();

  //! System is ready
  digitalWrite(LED_BUILTIN, HIGH);
                // GET /H turns the LED on

  printWiFiStatus();

  chessBoard.setBoard();
  chessBoard.drawBoard(BOARD_SERIAL);

  // Initialize the display
  oled.begin(SSD1306_SWITCHCAPVCC, OLED_I2C);
  sDebug("OLED initialized");

  // Clear the buffer.
  oled.clearDisplay();
  oled.display();
  sDebug("Buffer cleared. Starting Fonts test");

  // Show title scrolling on the display
  initDisplay(&oled);
  textFont(SANS_BOLD, 9, &oled);
  showText("The", 45, 15, COL_WHITE, &oled);
  showText("Distanced", 20, 35, COL_WHITE, &oled);
  showText("Pawn", 35, 55, COL_WHITE, &oled);
}
```

OLED Controller

The OLED controller should be configured according to the resolution and device model used in the hardware. In this case, I used an Adafruit monochromatic-compatible OLED display.

The OLED functions set includes all the fonts and text styles available for the device. If memory is needed, the text display can be simplified, limiting the number of available font variations.

```
/**
 * Show the text string at the desired coordinates.
 *
 * The text is shown according to the current settings (color,
   font, etc.)
 *
 * \param text The string of text to display
 * \param x The x cursor coordinates
 * \param y The y cursor coordinates
 * \param color The color of the text
 * \param disp Pointer to the OLED display class
 */
void showText(char* text, int x, int y, int color, Adafruit_
SSD1306* disp) {
  // Set the desired color
  switch(color) {
    case COL_WHITE:
    oled.setTextColor(SSD1306_WHITE);
    break;
  }
  disp->setCursor(x, y);
  disp->print(text);
  disp->display();
}

/**
 * Initialize the display before showing a new screen.
 *
```

```
 * \note This method should be called for first before
   setting a new
 * string or when the screen setting changes.
 *
 * \param disp Pointer to the Oled display class
 */
void initDisplay(Adafruit_SSD1306* disp) {
  disp->clearDisplay();
  disp->display();
}

/**
 * Start the text scrolling in the desired direction.
 * The scrolled text is what has already been composed
 * and shown on the screen.
 *
 * Start this method after the text screen has been set
 *
 * \param Dir the scrolling direction
 * \param disp Pointer to the Oled display class
 */
void textScroll(int dir, Adafruit_SSD1306* disp) {
  switch(dir) {
    case OLED_SCROLL_LEFT_RIGHT:
      disp->startscrollright(0x00, 0x0F);
    break;
    case OLED_SCROLL_RIGHT_LEFT:
      disp->startscrollleft(0x00, 0x0F);
    break;
    case OLED_SCROLL_DIAG_RIGHT:
      disp->startscrolldiagright(0x00, 0x0F);
    break;
```

```
    case OLED_SCROLL_DIAG_LEFT:
      disp->startscrolldiagleft(0x00, 0x0F);
    break;
    case OLED_SCROLL_STOP:
      disp->stopscroll();
    break;
  }
}

/**
 * Set the desired font and size to the text.
 *
 * The fonts have four predefined sizes: 9, 12, 18, 24 points
 * Any different value is ignored and the function do nothing
 *
 * The selected font should be enabled in the fonts definition
 * else the method set the default font and returns -1
 *
 * \param fontName One of the following font families:\n
 * <ul>
 * <li>MONO
 * <li>MONO_BOLD
 * <li>MONO_OBLIQUE
 * <li>MONO_BOLD_OBLIQUE
 * <li>SERIF
 * <li>SERIF_BOLD
 * <li>SERIF_ITALIC
 * <li>SERIF_BOLD_ITALIC
 * <li>SANS
 * <li>SANS_OBLIQUE
 * <li>SANS_BOLD
 * <li>SANS_BOLD_OBLIQUE
```

```
 * </ul>
 * \param fontSize The size in point of the font chosen between
   one of the
 * following values: 9, 12, 18, 24
 * \param disp Pointer to the OLED display class
 *
 * \return 0 if the font is set else return -1
 */
int textFont(int fontName, int fontSize, Adafruit_
SSD1306* disp) {
  int j;

  // Check for the font size
  if( (fontSize != 9) && (fontSize != 12) &&
      (fontSize != 18) &&(fontSize != 24) ) {
        return -1;
      }

  // Switch the font family
  switch(fontName) {
    case MONO:
    #ifndef FREE_MONO
    disp->setFont();
    return -1;
    #else
    switch(fontSize) {
      case 9:
        disp->setFont(&FreeMono9pt7b);
      break;
      case 12:
        disp->setFont(&FreeMono12pt7b);
      break;
```

```
      case 18:
        disp->setFont(&FreeMono18pt7b);
      break;
      case 24:
        disp->setFont(&FreeMono24pt7b);
      break;
    }
    #endif
    break;

    case MONO_BOLD:
    #ifndef FREE_MONO_BOLD
    disp->setFont();
    return -1;
    #else
    switch(fontSize) {
      case 9:
        disp->setFont(&FreeMonoBold9pt7b);
      break;
      case 12:
        disp->setFont(&FreeMonoBold12pt7b);
      break;
      case 18:
        disp->setFont(&FreeMonoBold18pt7b);
      break;
      case 24:
        disp->setFont(&FreeMonoBold24pt7b);
      break;
    }
    #endif
    break;
```

```
case MONO_OBLIQUE:
#ifndef FREE_MONO_OBLIQUE
disp->setFont();
return -1;
#else
switch(fontSize) {
  case 9:
    disp->setFont(&FreeMonoOblique9pt7b);
  break;
  case 12:
    disp->setFont(&FreeMonoOblique12pt7b);
  break;
  case 18:
    disp->setFont(&FreeMonoOblique18pt7b);
  break;
  case 24:
    disp->setFont(&FreeMonoOblique24pt7b);
  break;
}
#endif
break;

case MONO_BOLD_OBLIQUE:
#ifndef FREE_MONO_BOLD_OBLIQUE
disp->setFont();
return -1;
#else
switch(fontSize) {
  case 9:
    disp->setFont(&FreeMonoBoldOblique9pt7b);
  break;
```

```
    case 12:
      disp->setFont(&FreeMonoBoldOblique12pt7b);
    break;
    case 18:
      disp->setFont(&FreeMonoBoldOblique18pt7b);
    break;
    case 24:
      disp->setFont(&FreeMonoBoldOblique24pt7b);
    break;
  }
  #endif
  break;

  case SERIF:
  #ifndef FREE_SERIF
  disp->setFont();
  return -1;
  #else
  switch(fontSize) {
    case 9:
      disp->setFont(&FreeSerif9pt7b);
    break;
    case 12:
      disp->setFont(&FreeSerif12pt7b);
    break;
    case 18:
      disp->setFont(&FreeSerif18pt7b);
    break;
    case 24:
      disp->setFont(&FreeSerif24pt7b);
    break;
  }
```

```
#endif
break;

case SERIF_BOLD:
#ifndef FREE_SERIF_BOLD
disp->setFont();
return -1;
#else
switch(fontSize) {
  case 9:
    disp->setFont(&FreeSerifBold9pt7b);
  break;
  case 12:
    disp->setFont(&FreeSerifBold12pt7b);
  break;
  case 18:
    disp->setFont(&FreeSerifBold18pt7b);
  break;
  case 24:
    disp->setFont(&FreeSerifBold24pt7b);
  break;
}
#endif
break;

case SERIF_ITALIC:
#ifndef FREE_SERIF_ITALIC
disp->setFont();
return -1;
#else
switch(fontSize) {
```

```
      case 9:
        disp->setFont(&FreeSerifItalic9pt7b);
      break;
      case 12:
        disp->setFont(&FreeSerifItalic12pt7b);
      break;
      case 18:
        disp->setFont(&FreeSerifItalic18pt7b);
      break;
      case 24:
        disp->setFont(&FreeSerifItalic24pt7b);
      break;
    }
    #endif
    break;

    case SERIF_BOLD_ITALIC:
    #ifndef FREE_SERIF_BOLD_ITALIC
    disp->setFont();
    return -1;
    #else
    switch(fontSize) {
      case 9:
        disp->setFont(&FreeSerifBoldItalic9pt7b);
      break;
      case 12:
        disp->setFont(&FreeSerifBoldItalic12pt7b);
      break;
      case 18:
        disp->setFont(&FreeSerifBoldItalic18pt7b);
      break;
```

```
  case 24:
    disp->setFont(&FreeSerifBoldItalic24pt7b);
  break;
}
#endif
break;

case SANS:
#ifndef FREE_SANS
disp->setFont();
return -1;
#else
switch(fontSize) {
  case 9:
    disp->setFont(&FreeSans9pt7b);
  break;
  case 12:
    disp->setFont(&FreeSans12pt7b);
  break;
  case 18:
    disp->setFont(&FreeSans18pt7b);
  break;
  case 24:
    disp->setFont(&FreeSans24pt7b);
  break;
}
#endif
break;

case SANS_OBLIQUE:
#ifndef FREE_SANS_OBLIQUE
disp->setFont();
```

```
    return -1;
    #else
    switch(fontSize) {
      case 9:
        disp->setFont(&FreeSansOblique9pt7b);
      break;
      case 12:
        disp->setFont(&FreeSansOblique12pt7b);
      break;
      case 18:
        disp->setFont(&FreeSansOblique18pt7b);
      break;
      case 24:
        disp->setFont(&FreeSansOblique24pt7b);
      break;
    }
    #endif
    break;

    case SANS_BOLD:
    #ifndef FREE_SANS_BOLD
    disp->setFont();
    return -1;
    #else
    switch(fontSize) {
      case 9:
        disp->setFont(&FreeSansBold9pt7b);
      break;
      case 12:
        disp->setFont(&FreeSansBold12pt7b);
      break;
```

```
    case 18:
      disp->setFont(&FreeSansBold18pt7b);
    break;
    case 24:
      disp->setFont(&FreeSansBold24pt7b);
    break;
  }
  #endif
  break;

  case SANS_BOLD_OBLIQUE:
  #ifndef FREE_SANS_BOLD_OBLIQUE
  disp->setFont();
  return -1;
  #else
  switch(fontSize) {
    case 9:
      disp->setFont(&FreeSansBoldOblique9pt7b);
    break;
    case 12:
      disp->setFont(&FreeSansBoldOblique12pt7b);
    break;
    case 18:
      disp->setFont(&FreeSansBoldOblique18pt7b);
    break;
    case 24:
      disp->setFont(&FreeSansBoldOblique24pt7b);
    break;
  }
  #endif
  break;
}
```

Client Connection

The MKR 1010 client connection to the opposite player access point is run during the `loop()` function as the access point becomes available. The program detects the availability of the access point in every loop cycle, as well as the WiFi connection status.

The client part of the program communicates to the access point through the three web server APIs.

```
void loop() {
  // compare the previous status to the current status
  if (status != WiFi.status()) {
    // it has changed update the variable
    status = WiFi.status();

    if (status == WL_AP_CONNECTED) {
      // a device has connected to the AP
      sDebug("Device connected to AP");
    } else {
      // a device has disconnected from the AP, and we are back
         in listening mode
      sDebug("Device disconnected from AP");
    }
  }

  WiFiClient client = server.available();   // listen for
                                               incoming clients

  if (client) {                             // if you get
                                               a client,
    Serial1.println("new client");           // print a message
                                               out the
                                               serial port
```

```
String currentLine = "";                // make a String to
                                           hold incoming
                                           data from
                                           the client
while (client.connected()) {            // loop while
                                           the client's
                                           connected
  if (client.available()) {             // if there's bytes
                                           to read from
                                           the client,
    char c = client.read();             // read a
                                           byte, then
    Serial1.write(c);                    // print it out
                                            the serial
                                            monitor
    if (c == '\n') {                    // if the byte
                                           is a newline
                                           character

      // if the current line is blank, you got two newline
         characters in a row.
      // that's the end of the client HTTP request, so send
         a response:
      if (currentLine.length() == 0) {
        // HTTP headers always start with a response code
           (e.g. HTTP/1.1 200 OK)
        // and a content-type so the client knows what's
           coming, then a blank line:
        client.println("HTTP/1.1 200 OK");
        client.println("Content-type:text/html");
        client.println();
```

```
            // the content of the HTTP response follows
               the header:
            client.print("Click <a href=\"/H\">here</a> turn
            the LED on<br>");
            client.print("Click <a href=\"/L\">here</a> turn
            the LED off<br>");

            // The HTTP response ends with another blank line:
            client.println();
            // break out of the while loop:
            break;
          }
          else {      // if you got a newline, then clear
                         currentLine:
            currentLine = "";
          }
        }
        else if (c != '\r') {    // if you got anything else
                                    but a carriage return
                                    character,
          currentLine += c;      // add it to the end of the
                                    currentLine
        }

        // Check to see if the client request was "GET /H" or
           "GET /L":
//        if (currentLine.endsWith("GET /H")) {
//          digitalWrite(led, HIGH);
                          // GET /H turns the LED on
//        }
//        if (currentLine.endsWith("GET /L")) {
```

```
//           digitalWrite(led, LOW);
                                // GET /L turns the LED off
//         }
       }
    }
    // close the connection:
    client.stop();
    sDebug("client disconnected");
  }
}
```

PART VII

Radio Amusement

For a few seconds, the courtroom was swept with a noisy buzz. Then, the familiar melody of the bells came from the outside. The hall echoed with the melody of a metallic accent.

Sonya leaned closer to Ray, her voice barely a whisper.

"The radio broadcast of the process starts. It is a bit creepy, but the echo effect only happens from inside the court." Ray nodded, "Maybe a delay in the streaming," he said.

"The trial BDTH6159 against Ray and Tommy Badmington can start," the judge announced with a solemn voice after the ritual three gavel bangs on the table.

Tommy's emotions swirled in a mix of fear for the future and a strange fascination with the situation. The courtroom's grandeur, with its towering columns and gleaming scales of justice, added to the surreal atmosphere.

"Let's enter the witnesses!," the judge announced.

"We were alone, I am pretty sure!," exclaimed Tommy, trying to keep his voice low, turning to Ray and Sonya. Ray nodded, but Sonya had a doubtful look.

Meanwhile, the door guard left his post, re-entering from a side door at the bottom of the court. He wheeled in a large flat screen, positioning it at the center of the arena. The man, also serving as the trial assistant, moved efficiently.

"That screen is about the size of a movie theater," Tommy commented, trying to mask his anxiety with curiosity. The judge lifted a smaller screen onto his table.

"We can proceed," said to the court assistant. The man in uniform, silently standing beside the screen, pressed a button on a remote controller.

A movie that neither Ray nor Tommy ever imagined to see started while the light softly diminished.

UNIT ID: C D R-M 0753

LOCATION: SANDCASTLE

ACTIVITY TIME: 05:00

ACTIVITY RECORD: MAINTENANCE

HEIGHT: 19.3 m

The strange stop-frame faded after a few seconds, replaced by the footage of those critical moments. The screen replayed the endless minutes from when Tommy crossed the dome fence to when the sandcastle began its unexpected self-destruction.

"The maintenance robots!," Tommy cried.

"Unfortunately, my worries were correct," Sonya replied. "The flying robots grabbed a loop of the last five minutes for security reasons. These are their last five minutes of maintenance activity."

The same endless scene repeated for every maintenance robot; Tommy and Ray lived again and again, like psychological torture, the same moments under countless different points of view, from different sides, and at different heights.

When the last short movie ended, the screen went black.

"The debate will start in 20 minutes," the judge said, hitting the gavel on the table. Then he disappeared through a small door near the right side of the table. The assistant wheeled the giant screen out of the court hall.

Tommy stood up, followed by Ray and Sonya.

"I need a quick walk outside," he said. Ray and Sonya nodded, following him.

When they were at the exit door, a doorman, identical to the judge's assistant but not the same, smiled at them.

"Please don't forget the radio," he said, giving Ray a portable transistor radio, a 70s model that Ray immediately recognized. As he got it in his hands, the radio started playing soft classical music.

"This radio is not working or has been modified," Ray said, examining the device. Tommy, intrigued, observed it in the hands of his father.

"The volume can be changed with this knob, but the power-off position is not working," Ray explained to his son. "The station wheel rotates, but nothing happens." He continued moving the radio to see all the sides.

"Are we obliged to carry this and listen to the music? I am not in the mood," Tommy asked.

"It seems yes," Sonya commented.

"At least, the volume can be set to zero," Ray said.

"I feel like living a sort of predestination," Ray continued while they walked silently, each one lost in their own thoughts.

"What do you exactly mean?," Tommy asked.

"This process seems the result of a perfect organization," Ray said. "This trial for us is a trap full of unknowns, but it is a show that someone already knows the end."

"To be honest," Sonya said, close to Ray, "I never heard of any trial in the past. I never read any mention of them in the historical log files."

"But you were here when the park was built," Ray commented. "It is weird. As far as I see, the park was built decades ago. You can see why I would be confused," he concluded.

Before Sonya could reply, the portable transistor radio buzzed. Ray took it in his hands; the volume knob moved automatically, raising the sound level. Then, the channel rotated to change the frequency. Tommy was speechless and admired it at the same time.

"The first debate of the trial will start in three minutes. All those involved are invited to return to the court hall as soon as possible."

When the message ended, the device stopped after another buzz. Masked by the distortion of the aged device, the tone and solemnity of the judge's voice were impossible to confuse.

Ray, Tommy, and Sonya went back quickly.

"I am curious to learn the name of our lawyer," Tommy said.

"Until now, we only know their code: D4," Ray replied. During the pause, the center of the court had been rearranged. A chessboard of at least five meters per side was on the ground before the judge's table. Two assistants were setting up big chess pieces on the board.

"Is this a joke?," Ray exclaimed. All three were speechless. Sonya's expression was surprised as well; no one could explain what was happening.

After the three gavel hits, the judge declared the first debate open. Ray's radio buzzed again, synchronizing automatically with the station on the trial broadcast channel.

"BOTH6159 against Tommy and Ray Badmington. In court session one, enter the prosecutor, Bent Larsen," the judge announced.

A dark-dressed man in a white shirt and black tie came out from the door, where the assistants came when setting up the chess pieces.

"Good day, your honor," said the man standing beside the chess board before the judge.

"The prosecutor Bent Larsen has entered the court," the radio buzzed.

After this ceremonial salutation, the judge introduced the defense lawyer.

"For the defense of Tommy and Ray Badmington, enter D4."

A woman, dressed like Bent Larsen, reached the middle of the chessboard from the same door.

"Good day, your honor," she said. The two lawyers silently shook hands, and each sat at one of the player's sides.

"The debate is ready to start," the radio buzzed.

"We will hear the chronicle of our debate on this radio," said Tommy. Ray nodded, getting his Palm out of his pocket.

"Sonya, do you play chess?," Ray asked.

"I know the rules, but it's been a long time since I played," she answered while a tinge of sadness appeared on her face. Tommy was observing his father without speaking a word.

"I hope you know what you are doing," Sonya said. Ray didn't answer, but focused on his Palm handheld.

"If I am right, our destiny is in the hands of one of the oldest games in the world," Ray said, moving his attention from the device.

"Do you think we can do something?," asked Tommy, visibly confused.

"Actually, I think no, son," Ray answered. "I was trying to set up my small computer to keep track of what seems to be the most probable option. Absurd and inconceivable, according to our idea of justice, but not for this population of robots."

"What do you think, Ray?," Sonya asked. "I feel so helpless."

"This is not the time to sulk. Let's watch the next moves," Ray said, holding Sonya's hand, who tried to smile.

The lights dimmed while a spot illuminated the chessboard, and another projected a bright light over the balance of justice on the wall over the judge's table.

The balance of justice oscillated in both directions and stabilized perfectly.

When all were ready, the prosecutor sat on the white side, stood up, and made his first move.

"From this point of view, it is almost impossible for us to see the moves in detail," Tommy commented.

"It is so sad that we have no options to influence the course of the game," Sonya added.

Ray was still focused on the Palm screen. Meanwhile, the radio buzzed, "White moves to D4." The silence in the hall was full of tension, interrupted only by the soft noise of the radio, which muted any comment after the move declaration. Ray wrote the first move on a chessboard on his tiny screen.

"Let me explain my hypothesis," Ray said. "This debate will follow a logic. Even though it is difficult for us to understand, it will be considered 'correct' for a robot."

"Chess is a game where nothing is up to chance; it all depends on the strategy and the capabilities of the two opponents," Ray concluded. His mind was trying to figure out an understandable behavior.

"But we had no choice but to decide on our defense lawyer," Tommy said.

"None of us knew the lawyer before. I am not a law expert, but this should be allowed," Sonya added. "We all know very well that when the creator built the park to grant the most independent and correct justice administration, he separated the judge, prosecutors, lawyers, and the whole process from any other aspect of the park engines and logic," Sonya explained. "I am not aware of how this system works either."

"I insist! This is not justice!," Tommy interrupted in a louder voice.

The doorman guard silently approached them from behind.

"Please exit the courthouse," the guard said in a formal tone. They looked at him with a questioning expression. "Loud noises are prohibited during court debates. You are no longer authorized to follow the debate in person. You can follow it on the radio," the guard concluded. Then he opened the entrance door and invited them to exit.

"I spoke too loud!," said Tommy. "I always create trouble," he commented sadly.

"It's okay to follow the debate outdoors," Ray answered, sitting on a wooden chair around a table in the middle of the lovely garden in front of the courthouse building.

Tommy followed him, shrugging his shoulders.

"Black moves Nf6," the radio buzzed. Hearing the moves from the radio was more than sufficient to follow the debate evolution.

"The mistake is ours! We were viewing the whole process from the wrong perspective," Ray said.

"What do you mean, Ray?," Sonya asked.

"As strange as it sounds, it's starting to make sense," Ray followed. "We always had all the information. But, we had been conditioned by a non-logic vision, which the machines totally ignore." He followed. Tommy and Sonya tried to understand what he was implying.

"What do you mean by following the pure logic of the machines?," Tommy asked. Sonya nodded; this was the same question she had.

"The prosecutor and defense lawyer debate in the virtual world of a chessboard, okay?," Ray followed. The two nodded silently. "Chess play can't be equivocated; every move exclusively depends on the player's ability, knowledge, and strategy. No tricks, no lies, no mistakes. Every move can decide the final result," Ray continued, fired up.

In the meantime, the radio buzzed "c4, g6, Nc3, Bg7"

Ray was diligently annotating every move on his Palm.

"Look. The chess pieces represent every character we recognize in human nature. And every move results from a strategy, exchanging attack and defense, and vice versa." Tommy and Sonya were starting to understand Ray's point.

"Please continue," Sonya said. Roy nodded and followed, expounding on his theory.

"Here we have the first move: the prosecutor puts his center pawn in position d4. This is the popular Queen opening. But what really counts is that our lawyer's name is D4. I think these two things are strongly related. Both of them speak the same language and debate on the same topic."

"This seems perfect!," Tommy exclaimed.

"I am impressed and fascinated," Sonya admitted.

"e4 d6," the radio buzzed again.

"These last two moves," Ray said, showing the last radio announcement on his palm screen, "Confirm my suspicion. Our lawyer, D4, is entering the King's Indian Defense," He said.

"We know that this amusement park was built between the end of the 50s and the first decade of the 60s," he continued.

"Everything pertains to a fascinating vintage world," Tommy commented.

"In fact, the King's Indian Defense was used by Bobby Fischer in 1958." While explaining, Ray spoke with a look of satisfaction.

"If I am right, we will not see more recent game openings or strategies. Those modern games did not exist when this place was built," Ray concluded.

"Maybe there is a way to train our lawyer after the debate," Tommy said hopefully.

"I don't know if these machines can follow a different pattern, but our knowledge is limited to the repair and maintenance operations. We can't change their original programs," Sonya said.

The King's Indian Defense, played aggressively by D4, was highly efficient. After 41 moves, the radio buzzed "Qf3 Bg3. D4 wins."

The victory filled all three with cautious optimism.

"If D4 wins, the second debate will be about whether our destiny will drastically change," Ray said.

"I hope. You can't imagine how much I hope, but unfortunately, Tommy remains guilty, according to our laws," Sonya said.

"I see," Ray answered. "But winning the debate may change his sentence. At least, it is what I hope."

In the meantime, the radio buzzed, "As D4 is the only available defense lawyer, no change is possible. The second debate is open." It was the unequivocal voice of the judge.

This time, D4 started from the white side.

"D4 moves to d4." When the radio announced the first move, Ray nodded.

"My supposition is confirmed," Ray said. "D4 will always play the Queen opening. Sonya, what do they mean—the judge's words when he started the second debate?" Ray asked.

"I think that it refers to the fact that the accused can choose up to three different lawyers, asking them to change every debate. I was not aware what you said was related to chess abilities," Sonya said.

After thinking about this detail, she added, "Maybe we can use it in the future."

Unfortunately, the game of the second debate was literally a disaster. D4 lost against an elegant defense strategy of the prosecutor.

"Mate for D4 in ten moves," the radio buzzed, then silenced.

No one commented. A cold silence enveloped the three, and none of them spoke, waiting for the start of the third and last debate.

When the radio buzzed again, the judge repeated the same formula, introducing D4 as the defense lawyer for the third time.

"This is our last chance," Ray said without enthusiasm. Sonya nodded.

"D4 is better in defense than attack," said Tommy, trying to convince himself that there was still hope.

"The judge repeated the same formula," Sonya commented.

"What worries me," Ray said, "is that the prosecutor, at this point, already knows the weak points of our lawyer. I am sure he will use this knowledge in the game."

"You are too pessimistic, Ray," Sonya commented, trying to smile. All three were aware that Ray was right.

The prosecutor, again, started with the Queen's opening. D4 played a nerve-wracking game, demonstrating incredible resistance. Regardless of the game brilliance and the tactical ability demonstrated by D4, she lasted 95 moves. Then, the radio announced

"Qb2. D4 lose for mate."

"She lost like Mikhail Botvinnik in 1960," Ray said, and they were enveloped in a shadow of sadness and despair.

CHAPTER 17

The Radio Magic Upcycling

Figure 17-1. *The past century Radio Magic*

The upcycling of this '60s Bush Radio (see Figure 17-1) marks the starting point of a path I used to make music with vintage technologies integrated into the most recent hardware and software devices available today.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_17

How can an old transistor radio be used to make music? Electronic music can be created with digital or analog synthesizers, which can generate waveforms in the audible range (from 20 to 20.000Hz). Many functions can be added to a bare waveform generator, applying effects. These consist of a "chain" of devices or software modules to change the original sound.

Indeed, this is not the only approach to electronic music; instead of artificially generated sounds, you can *sample* sounds.

A *sound sample* is the recording of a short sequence of sounds: a musical instrument, a noisy tool, the sound generated by a percussion, or—why not—a vintage radio.

Note One of the first music synthesizers was the Mellotron, an electro-mechanical musical instrument developed in Birmingham, England, in 1963. The instrument is played by pressing its keys, each of which pushes a length of magnetic tape against a capstan, which pulls it across a playback head. Then, as the key is released, the tape is retracted by a spring to its initial position. Different portions of the tape can be played to access different sounds (https://en.wikipedia.org/wiki/Mellotron).

17.1. Tuner Mechanical Upgrade

Like most radios from the same period, the Bush Blue Baby Limited Edition transistor radio I used in this project uses a mechanical tuner to find the radio stations.

Radio tuning is controlled by a variable capacitor, a widely used component in radios produced for many decades until the early 70s.

It is not easy to operate the proper selection by rotating the variable capacitor shaft; for this reason, the tuning mechanism has a pulley mechanism driven by a thin cotton thread to increase the rotation angle for a more precise selection of the desired band. See Figure 17-2.

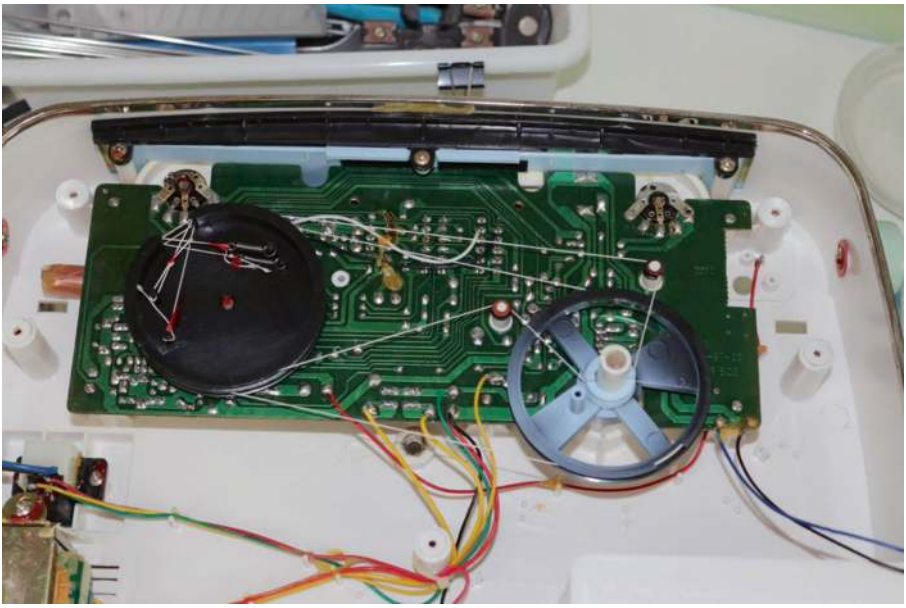

Figure 17-2. *Detail of the pulley-driven tuner mechanism. On the left side is the shaft connected to the variable capacitor, while on the right is the reduction pulley attached to the tuner shaft*

Note A *variable capacitor* is a component whose capacitance may be changed mechanically or electronically. Variable capacitors are used in L/C circuits to tune radio stations.

17.1.1. Making an Auto Tuning

Acquiring audio samples from a radio receiver is easy; recording on any device from the radio's audio-out plug is sufficient, but this method has a lot of limitations, and it is not what I want to create an original sound sample.

The exciting goal is to acquire short samples of audio noise while the tuner moves from one station to another. The transistor radios of the same age cover the three AM, FM, and LW bands, and my preferred effect is to generate recording sequences from the noise while scanning a portion of the AM band, as well as the other frequency ranges.

This effect can create samples added to a sequencer for later use or included real-time in a live execution; this second case is the most interesting because we will never retrieve identical sequences due to the radio broadcast characteristics.

I want to achieve continuous screening automation between a selected range of frequencies. To do this, I need to make mechanical changes to the radio tuning, including removing one of the original pieces to make space inside: the internal speaker. See Figure 17-3.

Figure 17-3. *The internal body of the radio is disassembled, removing the speaker. This makes sufficient space inside to host a low-speed stepper motor—and the motor controller—to enable the rotation of the variable capacitor.*

After some quick tests, I excluded using a DC motor because the tuner capacitor should be operated with high precision and very short increments. I used an inexpensive geared stepper motor (28BYJ-48, 5V stepper motor, four phases, and five wires).

Upgrading the Mechanics

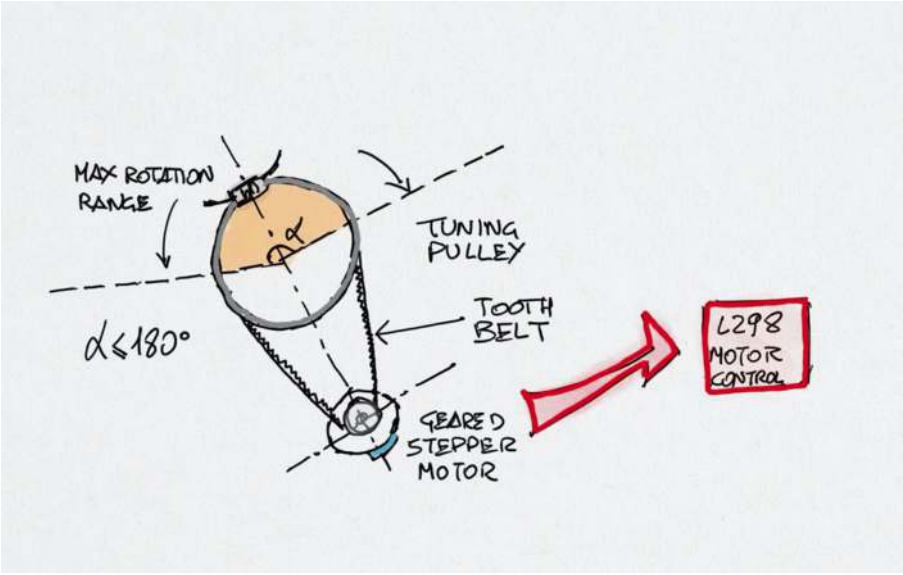

Figure 17-4. *Using a small pulley (almost the same diameter as the manual tuning shaft pulley) and a toothed belt connected to the large pully of the tuner capacitor, I can reach the same result of reducing the motion per step. This operation is necessary, as the radio tuner capacitor has a rotation angle of only 180 degrees.*

One of the "musts" I always impose on upcycling projects is to keep the device intact as much as possible, with minimal aesthetic changes.

In this case, the goal is to add a motor to control the tuner without changing the original pulley system; this also keeps the manual shaft moving after the automation. See Figure 17-4.

To occupy minimal space inside the radio body, I 3D printed two pieces: the motor support and the tooth belt gear of the same diameter as the tuner's large pulley. See Figure 17-5.

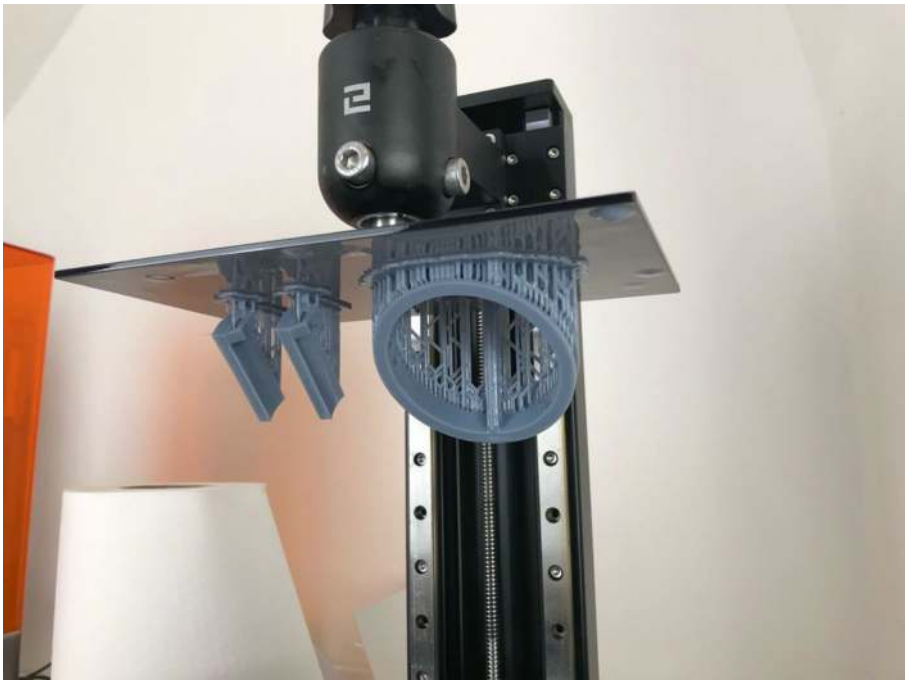

Figure 17-5. *The 3D printed supports and the tooth belt gear to move the tuner pulley with the stepper motor*

The ring gear was centered on top of the tuner pulley and glued. Because the tuner has a minimal rotation (less than 180 degrees), it was not difficult to connect it to the small stepper motor gear, reducing it to about 10:1.

After this operation, the manual tuner—the big white wheel in front of the radio—is blocked but will move following the motor. See Figure 17-6.

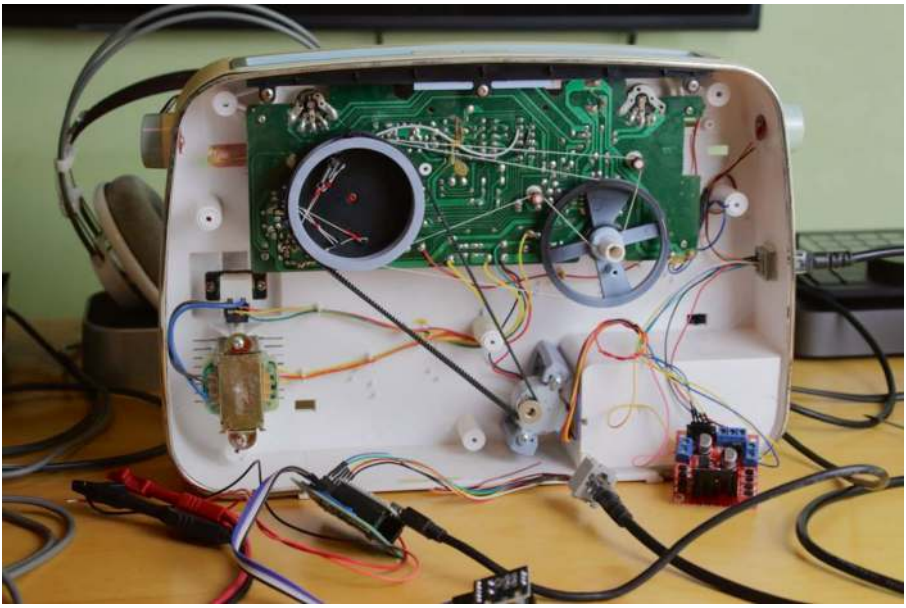

Figure 17-6. *The tuner pulley with the stepper motor is connected through a toothed belt. Thanks to the reduced size of the motor controller board, the parts fit easily inside the radio's body*

All Wires in One Connector

I faced a problem when I determined the number of wires that should go from the radio to the controller.

Even when upcycling a 60s device, using an old-style connector can't be the solution; it will ruin a considerable part of the plastic case, requiring a couple of screws and a large rectangular hole. To avoid this, I used a modern RJ-45 LAN connector instead, which had some attractive advantages:

- Minor changes to the case's aesthetic can be glued on one of the sides of the radio without excessive alterations.

- There are fewer than 12 wires to connect, so the connector will be sufficient.
- Special cables are not needed using the same solution on the controller side, so a straight network cable is sufficient.
- Signals and power supply for the motor controller and the radio are supported.
- The controller side takes advantage of the small size of the RJ-45 female plug.

When the work was finished, this proved to be the best solution. See Figure 17-7.

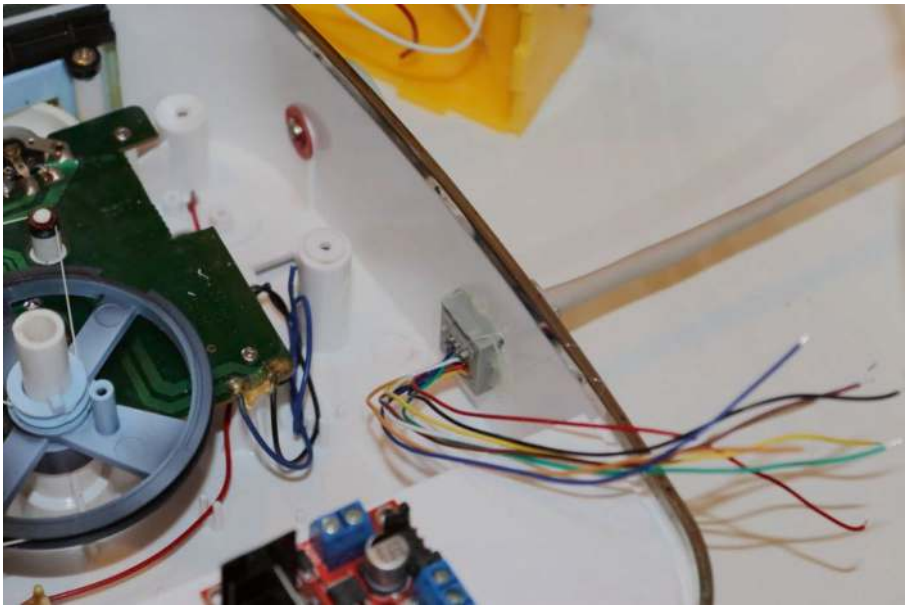

Figure 17-7. *The RJ-45 plug is glued to the radio case's left side device*

17.2. Auto Tuner Controller

I designed this prototype in two separate parts: the mechanics and the controller. After adding the automation to the tuner pulley and ensuring that all the signals and power supply were available on one side of a standard LAN cable, I determined that it would work no matter how the motor was controlled.

17.2.1. Requirements

The controller features do not need too many extra components, and an Arduino Nano microcontroller is sufficient to achieve the requirements.

- A rotary encoder moves the stepper motor in both directions.
- A temporary switch button is used to set the start and end of the tuner range and start the automation.
- A red LED signal indicates that the circuit is working.
- An RJ-45 female plug connects the signals and power lines to the radio.
- A small connector powers the controller, the motor, and the radio.
- A socket hosts the Arduino Nano board on the PCB; the microcontroller board can also be soldered directly on the board to reduce the height.

What We Get

According to the requirements, the controller should work as follows:

1. After powering on, the power LED shows that the circuit is working. To operate the radio tuning, the radio should also be powered using the rotating wheel on top (the volume wheel). To hear or record the audio, an audio mono cable should be plugged into the line-out plug of the radio, as there is no longer a speaker inside and the internal amplifier has been excluded.
2. Depending on the tuner gauge's current (last) position in front of the radio, the rotatory encoder is moved in a direction according to the device's audio feedback.
3. When the desired first position of the tuner reaches the first point, I press the temporary switch button to set the "start" tuner position.
4. I then do the same to position the tuner to the "end" position and press the button again.

Because the second position has been set with the temporary switch button, the tuner starts looping between the two positions until the temporary switch button is not pressed a third time (a reset command).

After resetting the loop, a new start-to-end loop position can be set again.

17.2.2. The Circuit and PCB

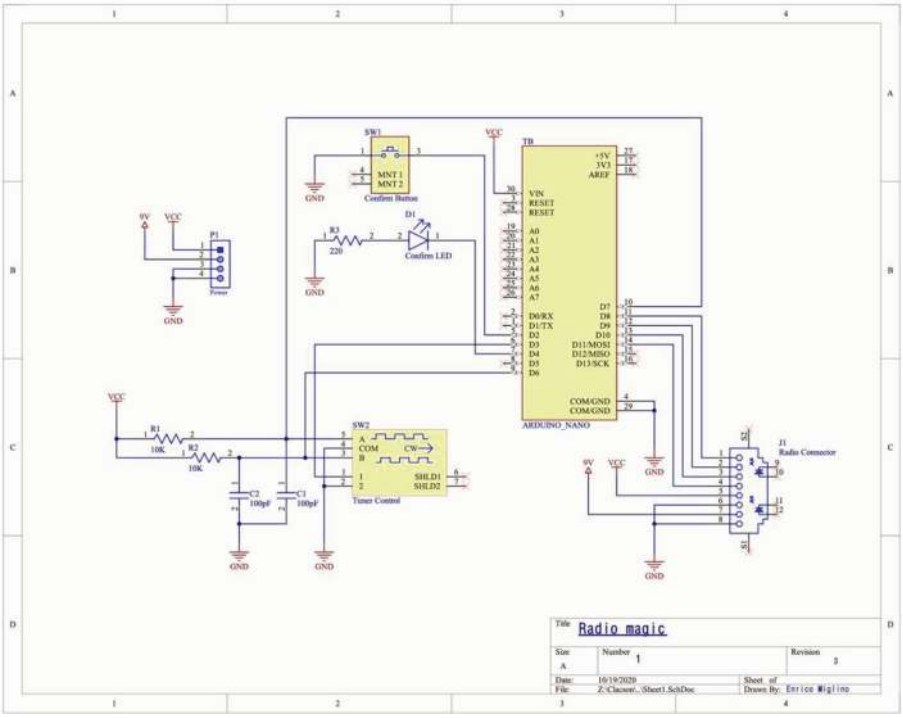

Figure 17-8. *The circuit schematics of the controller, as it was created to make the prototype*

Of course, the schematics were developed step by step, while testing the components on a breadboard. Then, after the prototype worked, I completed the schematics, from which I designed a decent PCB with Altium Designer. See Figures 17-8 and 17-9.

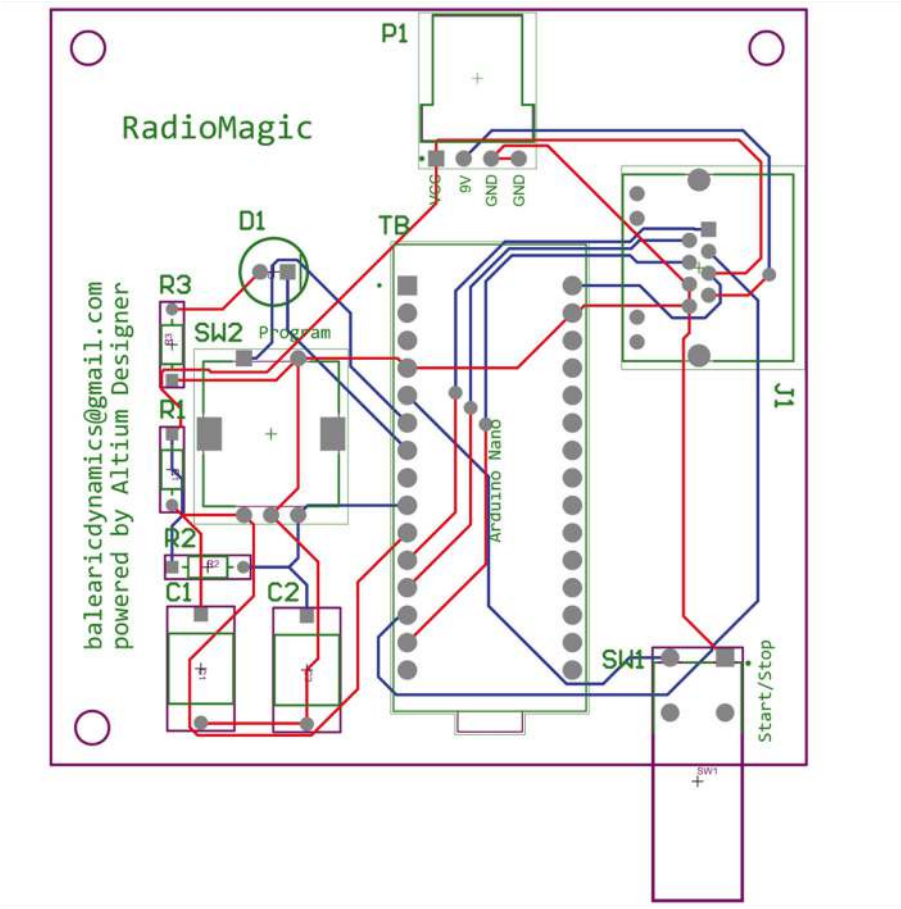

Figure 17-9. *The controller's two-side PCB routing. Disconnect the power line to the stepper motor and the radio to program the Arduino*

I made some PCB units to service, and after assembling them, they worked as expected. See Figure 17-10.

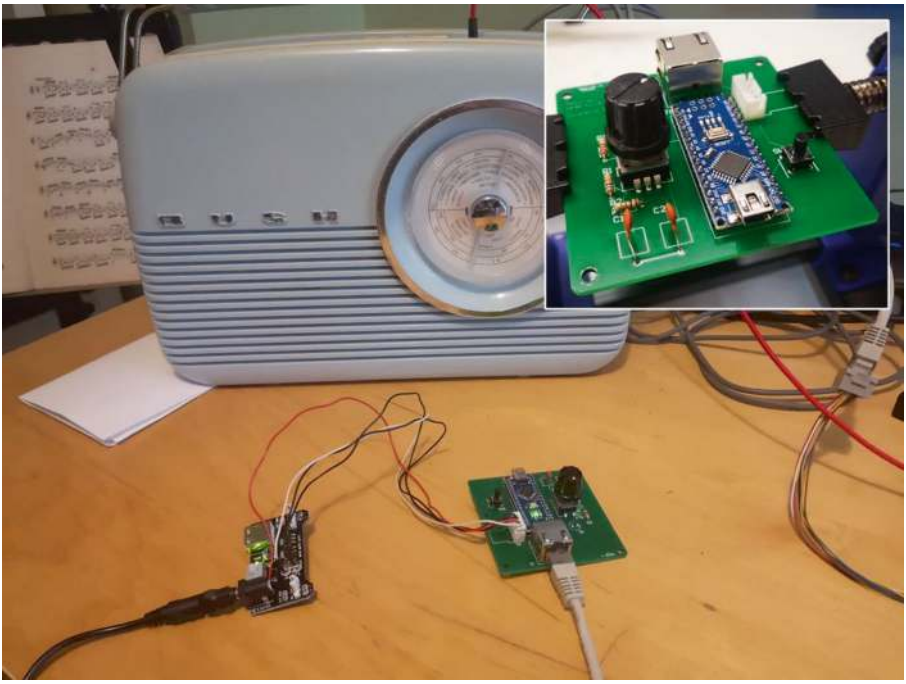

Figure 17-10. *The Radio Magic controller prototype, connected to the radio through a standard RJ-45 network cable*

17.3. The Controller Software

The software business logic runs the stepper motor according to the increments and direction of the rotary encoder. The temporary switch button drives the software events through an interrupt to handle the stepper loop.

17.3.1. Hardcoded Parameters

The `constants.h` header file defines all the control parameters of the application. These values have been tested empirically to get the best performance.

Constants concerning the stepper motor (steps and step increment) should be defined according to the kind of stepper motor and the demultiplication operated by the motor shaft on the tuner capacitor. Sizes and diameters can vary depending on the transistor radio model.

Note that the STEPPER_SPEED value corresponds to the stepper pulse frequency. In this case, the frequency is relatively high using a geared stepper motor. The reduction gears of the stepper move it more slowly than normal stepper motors (e.g., Nema 17) to increase its shaft torque, regardless of the low power of the stepper.

```
//! Steps per revolution, according to the motor specifications
#define STEPS_PER_REV 200
//! Preset tuner stepper speed
#define STEPPER_SPEED 90
//! Number of steps per increment, corresponding to a
rotary encoder
//! single step.
#define STEPPER_INCREMENT 100
//! Stepper direction clockwise. The number of increments is
multiplied
//! by the direction to get a positive or negative number
#define ROTATION_CW 1
//! Stepper direction counterclockwise. The number of
increments is multiplied
//! by the direction to get a positive or negative number
#define ROTATION_CCW -1

//! Clockwise calculated steps
#define ONE_MOVE_CLOCKWISE STEPPER_INCREMENT * ROTATION_CW
//! Counterclockwise calculated steps
#define ONE_MOVE_COUNTERCLOCKWISE STEPPER_INCREMENT *
ROTATION_CCW
```

```
//! L298 stepper motor controller pin 1
#define STEPPER_PIN_1 8
//! L298 stepper motor controller pin 2
#define STEPPER_PIN_2 9
//! L298 stepper motor controller pin 3
#define STEPPER_PIN_3 10
//! L298 stepper motor controller pin 4
#define STEPPER_PIN_4 11
```

The rotary encoder parameters are almost standard, as I count every single click. The only definition that can change concerns the pin assignments if you're using an Arduino board of a different microcontroller. The same is true for the looper button (the temporary switch).

```
//! Rotary encoder button pin (attached to Nano IRQ1)
#define ROTARY_BUTTON 3
//! Rotary encoder clock pin
#define ROTARY_CLK 6
//! Rotary encoder data pin
#define ROTARY_DATA 7
//! Rotary encoder value when rotating clockwise
#define ROTARY_CW -1
//! Rotary encoder value when rotating counterclockwise
#define ROTARY_CCW 1

//! Number of reading of the rotary encoder to take effect
//! Only the second reading is considered valid
#define ENCODER_READINGS 1

// ==============================
//       Looper Button &LED
// ==============================
```

```
//! Rotary encoder button pin (attached to Nano IRQ0)
#define LOOPER_BUTTON 2
//! Programming LED signal
#define PROG_LED 4
//! LED frequency while the stepper is looping
#define LED_FREQ 100
//! LED frequency when idling (looper button control has
temporary
//! stopped the loop)
#define LED_IDLE 1000
```

The RadioStepper structure defines all the dynamic values depicting the controller's state at any given moment in a single object.

```
/**
 * The RadioStepper structure contains the status of
 * all the parameters controlling the behavior of the radio
 */
struct RadioStepper {
  /**
   * The starting position has been selected
   *
   * This happens when the user presses for the first
   * time the rotary encoder button. From that point,
   * the number of effective steps is counted until
   * the button is not pressed for the second time.
   */
  bool isSelected = false;

  /**
   * When the button has been pressed for the second time, the
   * programmed status indicates that the system is ready to
   * loop the tuner.
```

```
 */
bool isProgrammed = false;
//! Status enabled when the tuner is looping
bool isLooping = false;
//! Current relative tuner position inside a loop
int tunerPosition = 0;
//! Looping direction. It is inverted when one of the
    two limits
//! is reached
int loopDirection = 0;

/**
 * Steps units expressed in number of rotary pulses
 *
 * The units are added algebraically to the counter until
 * the rotary encoder button is not pressed for the second
 * time. At this point the controller is programmed
   to execute
 * a loop.
 */
int loopSteps = 0;
//! Current rotary encoder position
int16_t encValue = 0;

//! Current LED non-stop blinking frequency
//! It is different when looping is stopped by the stepper is
//! programmed
int blinkLEDFrequency;
//! LED status, inverted during the non-blocking blinking
    mechanism
boolean isLEDOn = false;
```

```
  //! Starting millis to calculate the period for LED
      blinking in the
  //! non-blocking function
  unsigned long millisCounter;
};
```

17.3.2. The Program

Initial Setup

The program setup initializes all the states of the controller at boot and globally defines a RadioStepper structure.

```
RadioStepper radioStepper;

//! Initialization function
void setup() {
  #ifdef DEBUG
  Serial.begin(38400);
  #endif

  // -----------------------------------------------
  // Initialize the tuner stepper speed
  // -----------------------------------------------
  radioTuner.setSpeed(STEPPER_SPEED);

  // -----------------------------------------------
  // Initialize the rotary encoder
  // -----------------------------------------------
  encoder = new ClickEncoder(ROTARY_CLK, ROTARY_DATA, ROTARY_
            BUTTON);

  // Initialize the Rotary Encoder
  Timer1.initialize(1000);
  Timer1.attachInterrupt(timerIsr);
```

For obvious reasons, the temporary switch button interrupt is enabled as the last operation of the setup:

```
  pinMode(LOOPER_BUTTON, INPUT_PULLUP);
  pinMode(PROG_LED, OUTPUT);
  // Start showing the LED activity
  blinkLEDPeriod(1500);
  attachInterrupt(digitalPinToInterrupt(LOOPER_BUTTON),
  irqLoopButton, LOW);
}
```

The Main Loop

The primary action of the main loop is reading continuously—except when the temporary pushbutton generates an interrupt. The rotary encoder acts accordingly.

The logic of the loop's actions is driven by continuously checking the status of the `RadioStepper` structure and updating the concerned variables.

```
void loop() {
  // Read the encoder value. Maybe -1, 1 or 0
  radioStepper.encValue = encoder->getValue();

  // Check if the rotary position has changed (exclude the
     zero status
  if ( (radioStepper.encValue != 0 ) && (encoderCounter ==
  ENCODER_READINGS)) {
    if(radioStepper.isProgrammed == true) {
      // If the tuner is programmed and the user moves the
         rotary encoder
      // the programmed status is automatically reset
      setProgrammingStatus(false);
    }
```

```
    encoderCounter = 0; // Reset the counter readings
    // Check for the direction (clockwise of counterclockwise)
    if (radioStepper.encValue == ROTARY_CW) {
      radioTuner.step(ONE_MOVE_CLOCKWISE);
      // Update the loop counter (only if programming is set)
      updateLoopCount(ONE_MOVE_CLOCKWISE);
    } // Clockwise rotation
    else {
      radioTuner.step(ONE_MOVE_COUNTERCLOCKWISE);
      // Update the loop counter (only if programming is set)
      updateLoopCount(ONE_MOVE_COUNTERCLOCKWISE);
      } // Counterclockwise rotation
  } // Rotary encoder has been moved twice
  else {
    if(radioStepper.encValue != 0){
      encoderCounter++;
    }
  } // First encoder reading

  // Check for the rotary encoder button press. The 0 value
     shown on power-on can't be selected
  ClickEncoder::Button encButton = encoder->getButton();
  if(encButton == ClickEncoder::Clicked) {
  #ifdef DEBUG
      Serial << "Encoder button clicked" << endl;
  #endif
    if(radioStepper.isSelected == false){
      radioStepper.isSelected = true;
      setProgrammingStatus(false);
      // LED fixed on
      digitalWrite(PROG_LED, HIGH);
```

```
#ifdef DEBUG
      Serial << "isSelected true" << endl;
#endif
    } // Button pressed for the first time: start programming
        the range
    else {
  #ifdef DEBUG
      Serial << "Set prog status true" << endl;
  #endif
      setProgrammingStatus(true);
    } // Programming ended, start looping
  } // Encoder button clicked

  // Check for looping
  if(radioStepper.isLooping == true) {
    radioStepper.tunerPosition += (STEPPER_INCREMENT *
    radioStepper.loopDirection);
    // Check if the direction should be inverted
    if( (radioStepper.tunerPosition == 0) || (radioStepper.
    tunerPosition == radioStepper.loopSteps) ) {
      radioStepper.loopDirection *= -1; // Invert the loop
                                           direction
    }
    radioTuner.step(STEPPER_INCREMENT * radioStepper.
    loopDirection);
  }

  // Non-blocking LED blinking, if needed
  blinkLEDOnce();
}
```

The Interrupt Vector Function

When an interrupt occurs on the Arduino Nano IRQ 0, the control passes to the IRQ function that changes the looping status. Returning to the main loop, it changes the behavior as the scenario defined in the `RadioStepper` structure is changed.

The `irqLoopButton()` function is the callback that drives the status LED and changes the looping status in the `RadioStepper` global structure.

```
/**
 * IRQ Vector callback for Nano IRQ 0 (the loop control
   button pin)
 */
void irqLoopButton() {
  if(radioStepper.isProgrammed == true) {
    detachInterrupt(digitalPinToInterrupt(LOOPER_BUTTON));
     // If the tuner is programmed, change the status of the
        loop flag
    if(radioStepper.isLooping == true) {
      radioStepper.isLooping = false;
      radioStepper.blinkLEDFrequency = LED_IDLE;
#ifdef DEBUG
      Serial << "LED idle" << endl;
#endif
    } else {
      radioStepper.isLooping = true;
      radioStepper.blinkLEDFrequency = LED_FREQ;
#ifdef DEBUG
      Serial << "LED frequency" << endl;
#endif
    }
```

```
  attachInterrupt(digitalPinToInterrupt(LOOPER_BUTTON),
  irqLoopButton, LOW);
  delay(10);
  radioStepper.millisCounter = millis();
  }
}
```

The entire software, documentation, and components are included in the chapter package.

PART VIII

Life with a Borg

“Qb2. D4 lost mate,” Tommy slowly repeated one word at a time. Then, an oppressive silence fell over them,.

“Ray and Tommy Badmington are immediately summoned to present themselves before the judge in the courtroom,” The radio buzzed.

Without speaking a word, the three moved to the courthouse entrance. Ray felt a deep sense of discomfort. He was apprehensive about his son’s destiny, and his sense of helplessness grew as he entered the courtroom.

Sonya sat in one of the first rows while Ray and Tommy stepped down the central stair separating the two halves of the semicircular arena to reach the front of the judge’s table. At that exact moment, one of the attendants was entering the side door carrying the last two pawns. Without the chessboard, the judge’s front floor appeared to Ray like an infinite, desolate place.

“Ray and Tommy Badmington,” the judge began.

“Based on the witnesses’ testimony and the outcome of the three trial debates, you are hereby found guilty, with varying degrees of responsibility, for the destruction of the sandcastle, disruption of the peace of BDTH6159, and of irrevocable damage. You are required to return to this courtroom in 12 hours to hear your final sentence.”

The judge banged the gavel on the table and stood up while the radio speakers echoed his last words.

When the judge exited, the lights dimmed. Ray and Tommy walked to the exit where Sonya was waiting.

They stepped outside, their hearts heavy with sadness. For a while, no one was able to speak a word.

They were like strangers in an unknown world; any complaint or argument at this point was useless.

Ray and Tommy walked with their hands in their pockets, and Sonya walked silently beside Ray.

"This is the end of it all," Ray said, looking at Tommy, who nodded. Sonya linked her arm with him.

"Do you have any idea what the sentence will be?," Tommy asked Sonya.

"No idea, Tommy," she answered. "I reviewed all the text available on the laws and rules of the park, but found nothing about the sentences."

Ray sighed without commenting.

"How long should we wait for the sentence?," Tommy asked. "Two hours and forty-five minutes," Ray answered, checking his Palm.

"About three more hours of torture," Tommy commented.

"I see our dream breaking like a mirror, transformed into a nightmare," Tommy said. "All because of my stupidity," he continued. "I'll never see Lake Michigan, and I'll never be back in Wisconsin," he continued. Ray nodded, shrugging his shoulders.

"All that I still have from Wisconsin are these red socks," Tommy said, showing his socks with a sad smile. Ray smiled sadly, too. "They remind me of the ball signed by Tom Gordon, which we have at home, in the stand drawer next to your bed," Ray said. Tommy nodded with a nostalgic expression on his face.

Sonya continued walking close to Ray without intervening in the chat between the father and son.

The three continued chatting and remembering, waiting to pass the slowest hours of their lives.

"Ray and Tommy Badmington are immediately summoned to present themselves before the judge in the courtroom."

When their transistor radio buzzed the message, they breathed a sigh of relief. Finally, the endless waiting was nearing an end.

"What will happen now?," Tommy asked Sonya. "I have no idea, Tommy," she answered.

"As I told you before, I have not found any trace of sentences in all the documents I read," she continued. "The process has been followed literally because the BDTH6159 justice administration has no previous experience. This is the first trial they have managed," Sonya concluded.

They arrived some minutes later; the judge had not yet entered the courtroom. Ray and Tommy sat in front of the judge's mahogany table, where they found two chairs, while Sonya sat on the first row of benches, nearest to the judge.

"Please rise, the judge is entering the court," said a guard opening the judge's entrance door.

The judge went to his table and sat on the sizeable leather-upholstered chair.

"Please be seated," he said after hitting his gavel three times on the table.

"In the matter of BDTH6159 vs. Ray and Tommy Badmington, it has been determined that both defendants are guilty." Hearing these words for the second time, Ray was thankful to be seated.

After a short pause, the judge continued.

"This court hereby sentences Tommy Badmington to a lifetime residency within the confines of the park."

Suddenly, Tommy's face assumed an unnatural pale color. Ray thought he was on the verge of fainting. Tommy didn't change his expression and didn't move a single muscle.

"Ray Badmington is ordered to be permanently expelled from the premises within 48 hours of the issuance of this decree." The judge took a long breath and paused. Tommy felt a terrible sense of abandonment.

"All this effort to reach this point," Ray thought, wholly discouraged.

Then the judge continued, "Mitigating factors: This court acknowledges that Tommy Badmington's actions were the result of excessive enthusiasm and imprudence. Consequently, he is granted the freedom to roam and reside within the park as he chooses. However, should Tommy Badmington engage in any further misconduct, this court reserves the right to revise his sentence and impose incarceration without delay."

Ray, Tommy, and Sonia were speechless.

"The court hereby declares this trial closed and the sentence to be executed definitively unless new evidence or incontrovertible justification is presented to this court within the next 24 hours."

Then, the judge concluded, "The session is adjourned." He hit the gavel on the table and left. They were expecting something dramatic, anticipated by the previous audience to the court. Hearing the sentence, all three were shocked.

They remained frozen in place for a long time, unable to move a muscle. Then Sonya moved closer to Ray and Tommy, still sitting in front of the judge's table, which was now deserted. With a nod of the head, she asked them to stand up. Ray and Tommy followed Sonya to the stairs and courthouse exit without expression. They walked like two automatons.

Outdoors, under the sunlight, Sonya and Tommy's eyes were red, as if they were about to burst into tears at any moment. None of them could imagine a way to change the situation.

"A life-long sentence," Tommy murmured. "And you are forced to go back to home." Looking at Ray and hearing these words, Sonya had to look away.

"Our last day together," Ray said, holding Sonya's hand and putting an arm on Tommy's shoulder, who was walking beside him.

"This adventure has become the worst day of my life," Ray said. "Another terrible loss has just been announced," he continued. Sonya held his hand more firmly. She tried a smile, looking at Ray, betrayed by a tear on her cheek.

"How can I survive this nightmare?," Tommy asked, aware that his question was rhetorical. It was a question to which everyone already knew the answer.

"How can I survive here, Dad?," Tommy asked his father in tears. "Can you imagine my whole life in a world of machines? Damn cryptex!," he screamed. At that moment, Ray's faith in technology faltered. None of them could say anything that was not too obvious or stupid. Ray was struggling to do something to soothe his son's anguish and panic.

"Son, I would do anything to get you back to Michigan," Ray said. Sonya's eyes flashed for a second.

Sonya moved half a step in front of Ray; she put her arms around his neck and pulled his face close to her.

"Would you accept the hardest condemnation on behalf of Tommy?," she whispered in his ear.

"Yes, if he could come back home," answered Ray in a loud voice.

"No!," exclaimed Tommy, who imagined their idea.

"You can't do this!," he continued.

"Son," Ray said, but Sonya interrupted him with a gesture.

"Ray, wait. And Tommy, please, your father and I need to speak alone." Tommy was silenced with a puzzled expression.

Sonya sat on one of the benches. Ray followed her, almost confused.

"As I told you before, I read all the documents about the justice administration, searching for something mentioning previous trials or sentences." Ray nodded. He was not sure what Sonya was up to.

"As far as I know, this is the first judgment happening at BDTH6159," Ray nodded, curious and still confused.

"Thanks to these readings, I learned a lot about the laws, rules, and procedures the creator hardcoded in the justice administrators' memories. Unfortunately, nowadays, only D4, the prosecutor Bent Larsen, and the judge are operational after so long without maintenance," Sonya continued.

–"What does this mean? It is an unpredictable situation that probably caused our misfortune," Ray commented.

"Indeed, but this can also be our fortune," she followed. "We can appeal to the partial incorrectness of the procedure by asking for a kind of refund." Ray's eyes started shining at her words. In the meantime, Sonya continued explaining her theory.

"We can present opposition to the unfair procedure of the trial. At this point, there are two options. We can ask for a new process, waiting a long time until all the justice administration members, lawyers, and prosecutors are repaired."

"The second alternative, which, in my opinion, will be more acceptable by the judge, is to pretend, as a form of indemnity, the substitution of the defendants," Ray nodded, curious to know the conclusion.

"Could Tommy then come back to Michigan to our house?," Ray asked.

"Yes, this could happen," Sonya answered.

"And you will be condemned to a life-long existence in this world. The final decision is up to you. But it would be best if you decide soon, because the 24 hours are almost up," she concluded.

Tommy couldn't ignore the tone of their discussion, even though he hadn't heard the words.

"Undoubtedly, the second option is the best choice," Ray said. Sonya nodded with a shade of sadness in her eyes. Ray, worried, noted Sonya's mood. He took both her hands with his hands and leaned toward her.

"After my arrival, something changed in my inner self. Regardless of all the problems, these days have also been the most meaningful," Ray said. "The idea of never seeing again you would break my heart."

"I feel the same," she answered.

"It was the first time I discovered that human sentiments are impossible to confuse," Sonya said.

"So, I only repeat, the decision is done. Tommy can be free," Ray said.

"Ray, you are important to me. You can't imagine how much." A contagious smile appeared on Ray's face. Tommy, still apart and waiting patiently, also smiled.

"Ray, let me finish. There is something that you still don't know about me. I want you to know this before you make your decision," Sonya continued. Ray remained silent, still close to her, her hands in his.

"Ray, there is no easy way to say this. I admit I am scared of your reaction. I hope you don't blame me for keeping this secret until now." Ray looked into Sonya's eyes. He was serious and focused on her words.

"Ray, I am not human," Sonya took a long breath. Ray remained silent.

"When the creator was almost finished, the amusement park was already older, so she decided to create a guardian or a supervisor. An android that would be a perfect reproduction of herself at age 35," Sonya paused for a couple of minutes to give Ray the time to assimilate the revelation.

"I am identical to a human in all the minimal details. The only difference is that I recharge with the sunlight. I do not need to eat."

Ray was speechless. He needed a few minutes before talking again.

"I am the inheritance of the creator," Sonya concluded.

"But you are human!," Ray exclaimed. He spoke so loud that this was the only sentence that Tommy clearly distinguished. And this seemed so obvious to him.

"Until a few days ago, the only thing I was conscious of was that I am an artificial creature. Before dying, the creator told me that she put in my mind the full backup of the BDTH6159 project. And a gift in my heart."

"A gift?," Ray asked.

"Thanks to you, now I know what it is," she said. "The creator also gave me humanity. I can feel sentiment, and I can love," she concluded.

Ray never stopped looking at her eyes. Suddenly, he stood up, pulling Sonya by one hand.

"Rush!," he said, starting to run to the courthouse. Sonya followed him, holding his hand tightly.

"Hey! Where are you going?," Tommy yelled.

"To free you!," Ray answered without stopping.

As Sonya explained their request to the door guard, he kindly described the appeal procedure, giving them a four-page module he printed with the terminal on the small desk near the entrance.

Ray and Sonya needed almost one hour to answer all the questions and explain the reasons for their request.

Tommy, a bit anxious, was waiting outside.

As Ray and Sonya came back, Tommy began questioning Ray insistently.

"Tommy, tomorrow you can go back to Michigan," Ray said. "But now we should get some sleep. Tomorrow will be a great day," he concluded.

Tommy went to sleep immediately in the small apartment, that strong jail without doors. His questions could wait until the next day.

At nine in the morning, a guard woke them up, calling them by name and giving Ray a document. "The court has accepted your revision request." He left the paper signed by the judge on the small square table and left them alone.

When they went outdoors, Sonya was standing near the entry.

"I heard the news on the radio," she said with a smile.

Meanwhile, a couple of guards reached them.

"Good morning. We have the order to accompany Tommy Badmington to the nearest exit portal," they said, then the group moved.

"Dad, are you sure?," Tommy asked during the walk.

"Yes, I am. It is the right decision for all," Ray answered. They walked for 20 minutes while Ray reassured Tommy.

The two guards stopped in the center of a meadow with trimmed grass. One of the two pressed a button on a small device like a remote controller. In a few seconds, a metal rotating barrier appeared in front of them. On the other side, Marina del Rey's dock was visible. The second guard stepped close to Tommy.

"It's time to go," he said.

Tommy nodded, turned to his father, and hugged him tightly. It was difficult for Tommy to retain his tears. Then, he hugged Sonya, "Take care of him," Tommy said.

"Count on me," Sonya answered, smiling. Tommy moved toward the revolving door.

"Tommy," Ray yelled to him when he was some steps away. Tommy turned again, surprised.

"One moment!," Ray said, searching in his pocket. "Take this, and think of your father," giving his Palm handheld to Tommy.

"I'll never forget!," Tommy said, smiling. Ray and Sonya followed Tommy with their eyes, embracing each other.

They waited until Tommy passed the border of BDTH5159, and then the door disappeared.

It remains uncertain whether, in those final moments, Tommy turned back for one last farewell, witnessing Ray and Sonya share a kiss, just like two lovers.

CHAPTER 18

Life with a Borg

Figure 18-1. *A Borg*

We can't know if Ray's choice can be considered a happy ending, but I am sure his decision changed the rest of his life.

What makes the fantasy work in this scenario is that he had a unique opportunity, an option that remains in the BDTH6159 world for now.

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1_18

Nevertheless, we makers are used to approaching problems from uncommon perspectives. For this reason, something that seems impossible, like giving a 1960 mannequin a kind of artificial life, becomes possible—or at least we try it. See Figure 18-1.

18.1. The Inspiring Automaton

> *The word "automaton" comes from the Greek word "αὐτόματον" (automaton), which is derived from "αὐτόματος" (automatos). The Greek root "αὐτό" (auto) means "self," and "ματος" (matos) comes from the verb "μαω" (mao), which means "to think" or "to be inclined to do something." Together, "automatos" roughly translates to "acting of itself" or "self-moving."*
>
> *In ancient Greek, the word "automaton" referred to something that operates independently without needing external control, such as a machine or device that moves by itself. The Greeks found this concept fascinating, and they used it to describe both mechanical devices and natural phenomena that appeared to move or function independently.*
>
> *Over time, the term came to be associated with self-operating machines, often focusing on those designed to imitate living beings, such as early robots or mechanical figures.*
>
> *So, "automaton" in Greek emphasizes the idea of self-action or autonomous operation, reflected in its modern usage to describe machines or systems that operate independently.*
>
> (Source: explanation provided by AI.)

18.1.1. Moving the Mannequin

I received a 1960 mannequin in very good shape from a friend, the owner of Depto09, the biggest vintage shop in Gent (Belgium). When I got it, my first idea was to make an automaton. A fan of the *Star Trek* saga, I called this project "Seven of Nine."

Note According to the *Star Trek* fiction series, the Borg civilization is based on a hive or group mind known as *the collective*. Each Borg drone is linked to the collective by a sophisticated subspace network that ensures each member is given constant supervision and guidance. The collective consciousness gives them the ability not only to "share the same thoughts" but also to adapt quickly to new tactics (https://en.wikipedia.org/wiki/Borg).

This mannequin transformation was the first complex vintage upcycling I tried. When I got it to stand on its pedestal in my living room, I spent a couple of weeks thinking about what to do with this "creature."

I had never seen how a mannequin is built inside, and I bet that technology has changed considerably in the last decades (see Figure 18-2). The construction of this model involved a lot of artistic work; my concern has been to keep the vintage feeling intact.

For this reason, I decided to remake the mannequin, giving it new features and more functionality, but preserving its original fashion.

Figure 18-2. *The two main parts of the mannequin while working on the transformation*

The mannequin body is well articulated, making it possible to expose it in different positions while dressing—in its original use.

The ankles and legs are made in a single piece, while the shoulders, wrists, and torso are moveable. Due to its mechanical building, I realized that the only moveable part that could be motorized was the torso. Regardless of this limitation, I decided to proceed this way; this rotation, when working autonomously, would be sufficient to create a pleasant effect.

18.2. Torso Rotation

The concern with adding a motorized system to rotate the torso is the body's total height. To avoid this problem, I designed a mechanism using a stepper motor that would not increase the height of the mannequin by more than five centimeters. See Figure 18-3.

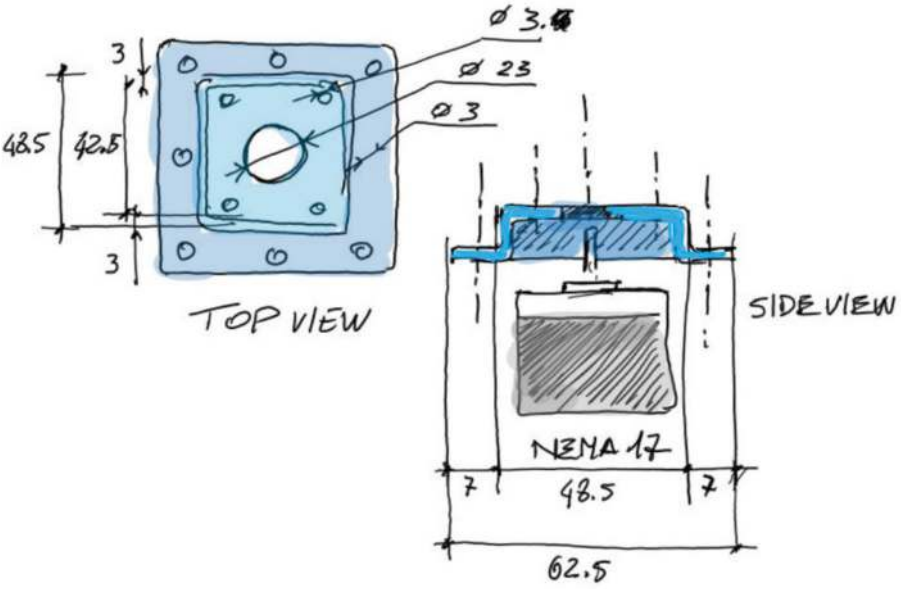

Figure 18-3. *The stepper motor used to rotate the torso was screwed to a 3D-printed support that kept the motor body inside the bottom side (ankle and legs) of the mannequin*

18.2.1. Mechanics

The rotation mechanism uses a Nema-17 stepper motor inserted inside the body's bottom. Thus, only the shaft emerges to the top surface. See Figure 18-3.

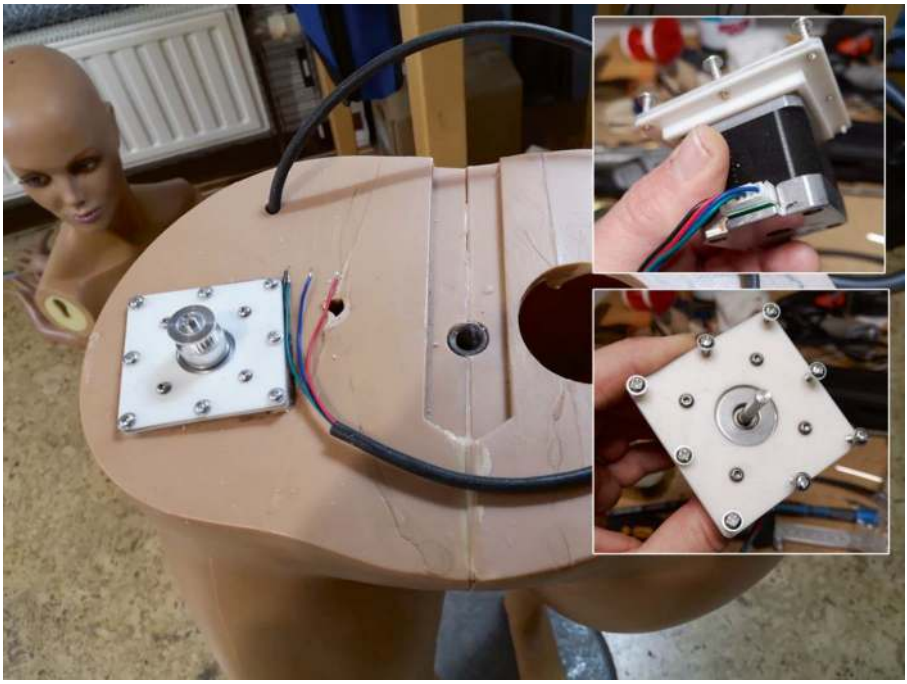

Figure 18-4. *The Nema-17 stepper motor, installed to the side of the bottom half of the mannequin's body. After the insertion, only the shaft with the tooth gear emerges for less than 4 cm*

The mannequin's top half was originally inserted into the bottom half using a swivel joint so that the torso could rotate to set the mannequin in different poses.

Figure 18-5. *The swivel joint is reused to motorize the torso rotation*

The swivel joint is fixed to the bottom of the body. Keeping the torso's original mechanic structure, I introduced a rotating platform controlled by the stepper motor between the bottom and the top parts of the mannequin. See Figure 18-5.

The empty torso weight is about five kilos; I expected a considerable increase in weight after the internal components and cables were inside at the end of the installation. To support this weight, the torso needed to lie on a consistent base and rotate with minimum attrition.

3D-printing a Lazy Susan bearing resulted in a very efficient and relatively simple solution that granted the required mechanical robustness.

The torso remained anchored to the bearing's upper side, connected to the stepper motor with a toothed belt. The 20-teeth shaft of the stepper motor, connected to the significant edge of the Lazy Susan bearing, applied a speed reduction factor and increased the torque applied to the rotation axis. See Figure 18-6.

Note A *Lazy Susan bearing* is a turntable mechanism that allows an object to rotate smoothly on a horizontal axis, typically used for rotating tables, displays, or shelving units. It consists of two concentric rings, with ball bearings sandwiched between them, allowing the top ring to spin while the bottom ring remains stationary. These bearings are commonly used in kitchen cabinets, serving trays, and rotating displays to provide easy access to items by simply rotating the surface. Lazy Susan bearings are available in various sizes and can support significant weight depending on their construction.

Figure 18-6. *The Lazy Susan bearing assembled on the top side of the bottom half of the mannequin*

The iron spheres of the bearing were easy to find online: I used 3mm diameter bicycle wheel spheres, bought as spare parts on Amazon.

The torso should not fully rotate, so the tooth-belt was tensioned and fixed on the moving part of the bearing; with this solution the torso could rotate about 120 degrees in both directions.

Following the same kind of solution adopted in CAM machines, I also 3D-printed an end-stop switch to find the point zero of the torso every time the system was powered on.

18.2.2. Motor Control

An Arduino UNO with a stepper motor controller based on an L298 IC achieved the torso rotation. To host the motor controller and the Arduino board, I opened a rectangular window on the back of the torso to access the inside of the body, where I introduced all the components. See Figure 18-7.

Figure 18-7. *The torso opening. I fixed all the components (including the power supplies and wiring) inside this window to control the torso rotation*

I determined the entire design of the Borg's features and characteristics when I started the upcycling. Nevertheless, many of the solutions required considerable improvisation and frequent updates to make everything work as desired.

When possible, I used all the processor power of the boards—an Arduino UNO for motion and direct feedback and a Linux-embedded Raspberry Pi. See Figure 18-8.

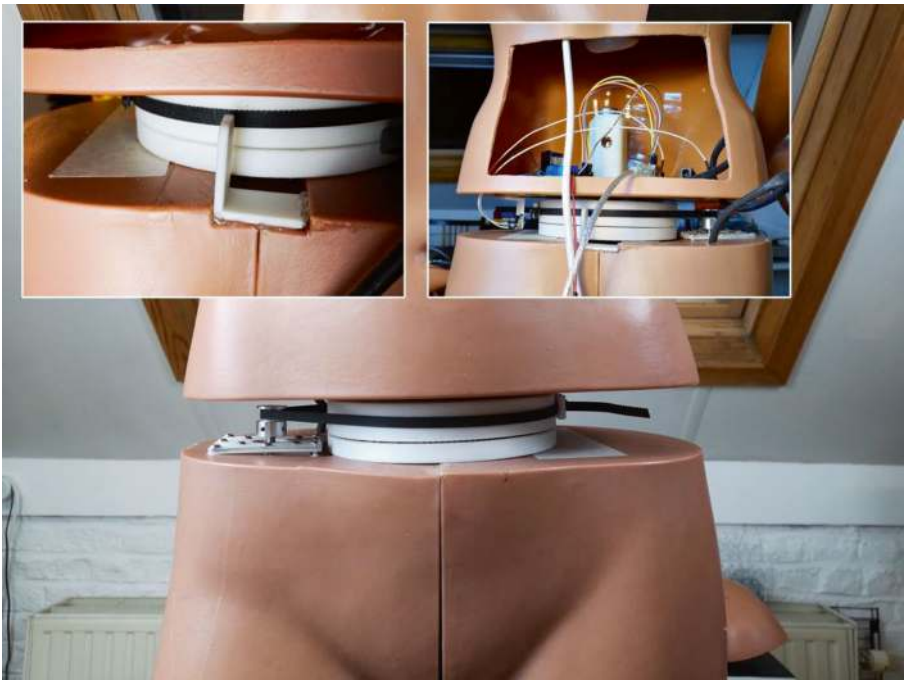

Figure 18-8. *The torso rotation control assembled for testing. The Arduino and L298 stepper controller are wired to the motor and the end-point switch inside the body*

18.2.3. Motion Feedback

The ability to turn in a specific direction when a sound catches our attention is a reaction that our brain should not necessarily mediate. Often, when someone calls us, we spontaneously turn toward the direction the sound is coming from without thinking and making a decision.

Similarly, the ability to rotate the torso to direct the gaze—the camera—to the sound must be an automatic reaction that the Arduino board can easily accomplish.

The Borg, capable of pointing out the direction from where a sound or a voice comes, is what I needed to make Seven of Nine reactive to sound-driven events. See Figure 18-9.

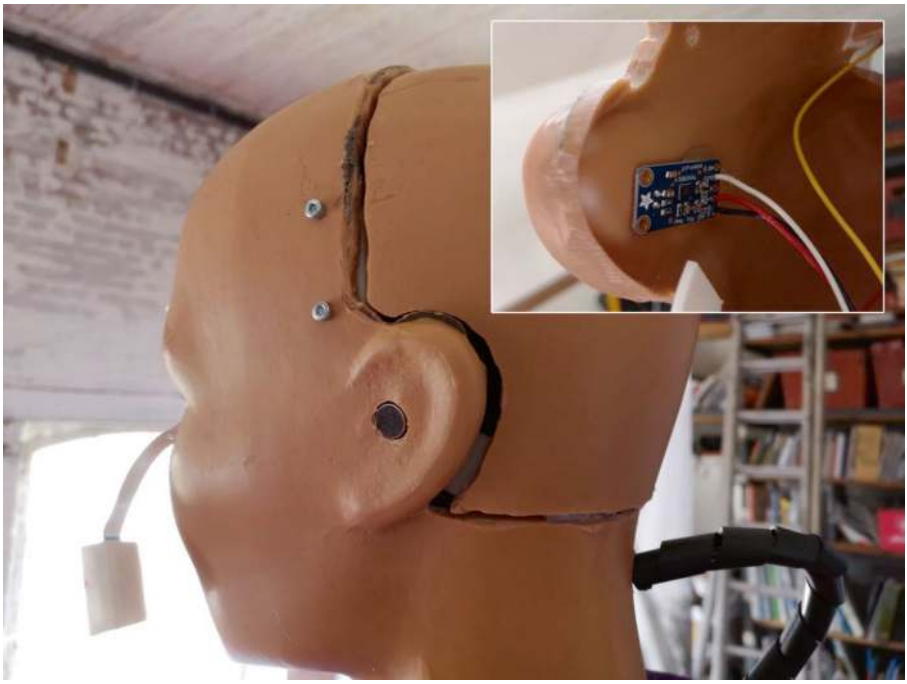

Figure 18-9. *The microphone installed on the head. Two microphones are added to the opposite sides in correspondence to the ears of the Borg*

The algorithm I developed on Arduino for the identification of sound in space—direction and distance estimated by the intensity—is based on *sound location.*

Sound Localization Algorithm

The sound localization methods are part of the science that deals with determining the direction and distance of a sound source with the sole help of the sound itself.

Acquiring the sound intensity from several microphones installed in different directions allows for accurate detection of sound sources, which is fundamental in a large number of industrial and multimedia applications.

There are sophisticated systems that can be found in many robotics applications, where a large set of microphones arranged circularly are used to obtain maximum accuracy in identifying the direction and origin of the sound source.

Even using only two microphones, a few simple mathematical formulas can be applied to acquire valuable data that is sufficiently precise for the scope.

Semi-random Feedback

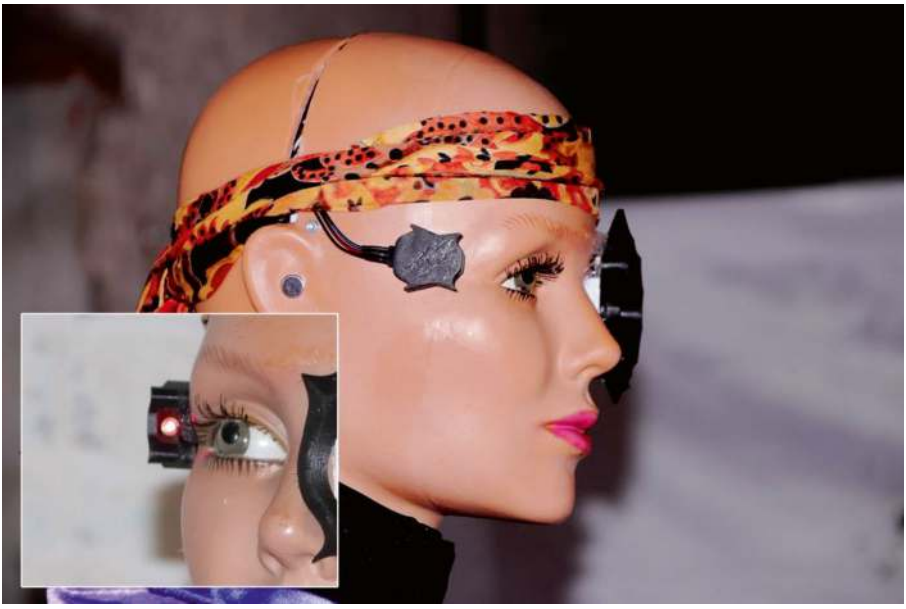

Figure 18-10. *The pointing laser was installed on the right side of the face. When armed, it points to the same direction of sight as the right eye*

Having space in the Arduino GPIO, I added a laser that reacts randomly with a red ray activated when the torso rotates, so that the Borg reacts to the sound.

I used a red laser pointer to mimic the Borg rays present in all the Collective units, including Seven of Nine. See Figure 18-11.

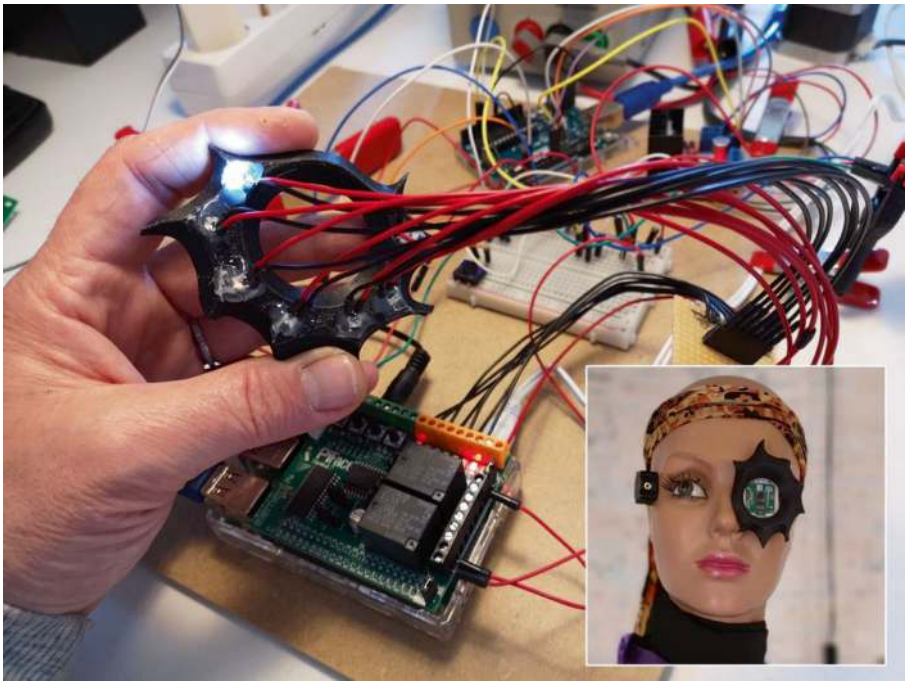

Figure 18-11. *The Borg implant in Seven of Nine's left eye is a 3D-printed model with a ring of high-intensity white LEDs flashing to the bottom*

Note Seven of Nine is a character in the *Star Trek* saga, first introduced in *Star Trek: Voyager*. Her designation is *Seven of Nine, Tertiary Adjunct of Unimatrix Zero One.* She is a former Borg drone freed from the Borg Collective by the Voyager crew. Played by actress Jeri Ryan, Seven of Nine initially struggles to regain her individuality and humanity after years of being assimilated by the Borg Collective.

Her character is known for her intelligence, logical thinking, and exceptional technical abilities, which often prove vital to the Voyager crew as they travel through the Delta Quadrant. She gradually evolves emotionally and learns to form relationships and rediscover her humanity.

18.2.4. Motion and Feedback Software

The Arduino application controls the torso rotation and activates the feedback behavior, which is parametrized through a series of header files. These headers allow customization of the peripheral devices' connections (microphones, stepper motor driver, laser pointer) and the behavioral control parameters.

The Parameters Header

```
/**
 * @file parameters.h
 * @brief Control parameters and structures for the mannequin
 *
 * @note As it is not used an end-stop switch we assume that
   when the
 * system is powered-on the torso position is in the middle.
 */
```

```
 //! Include the Arduino Stepper Library
#include <Stepper.h>

// ============================
// Stepper parameters
// ============================
#define STEP_PIN_A1 8     ///< STEPPER LIBRARY CONTROL PIN A+
#define STEP_PIN_A2 9     ///< STEPPER LIBRARY CONTROL PIN A-
#define STEP_PIN_B1 10    ///< STEPPER LIBRARY CONTROL PIN B+
#define STEP_PIN_B2 11    ///< STEPPER LIBRARY CONTROL PIN B-

#define ENDSWITCH_PIN 2   ///< End switch signal

// ============================
// Laser parameters
// ============================

#define LASER_PIN 3
```

The Motor Header

```
/**
 * @file motor.h
 * @brief Stepper motor global parameters and constants
 */

 typedef struct StepProfile {
  int torsoSpeed;
  int rotAngle;
  int lastAnglePos;
 };

//! Number of steps per output rotation.
//! Ref.: Nema 17 1.8 DEG/Step
#define STEPS_PER_REVOLUTION 200
```

The STEPS_PER_REVOLUTION is a fixed parameter, depending on the hardware characteristics (Nema-17 stepper motor). According to the two-toothed pulley diameter, the reduction factor is 170:14. This means that for a full rotation of the Lazy Susan bearing, there are theoretically 12 full rotations of the 20-teeth stepper shaft.

```
#define ANGLE_DEMULTIPLIER 12

//! Stepper predefined speed
#define SPEED_LOW 30
//! Stepper predefined speed
#define SPEED_MED 35
//! Stepper predefined speed
#define SPEED_HIGH 45
//! Search zero stepper end point speed
#define SPEED_ZERO 20

//! Increment in angles when searching the end stop point
//! corresponding to the leftmost position
#define SEARCH_ZERO_STEPS -1

//! Max angle in both sides respect to the middle torso
position
#define MAX_ANGLE 40
#define MIN_ANGLE 0
```

The Main Application

The initialization function configures the stepper, laser, and Arduino pins ready to start. The last call of the `setup()` is to `setTorsoZero()` function, which searches for the end-coarse switch to reset the torso position to a known starting point.

```
//! Initialization
void setup()
{
  Serial.begin(9600);

  pinMode(A0, INPUT);
  pinMode(A1, INPUT);

  pinMode(LASER_PIN, OUTPUT);

  // Initialize the laser
  laser.value = LASER_DEFAULT;
  laser.isOn = LASER_OFF;
  setLaser();

  // Initialize the endswitch input
  pinMode(ENDSWITCH_PIN, INPUT_PULLUP);
  // Initialize the stepper class
  torsoStepper.step(STEPS_PER_REVOLUTION);
  // Move to zero point
  setTorsoZero();
}
```

The main `loop()` function implements the hearing algorithm to rotate the torso according to the detected sound direction.

```
  peakToPeak[0] = 0;
  peakToPeak[1] = 0;
  signalMax[0] = 0;
  signalMax[1] = 0;
  signalMin[0] = MAX_SIGNAL;
  signalMin[1] = MAX_SIGNAL;
```

```
// collect data for 50 mS from both ears
while ( (millis() - startMillis) < sampleWindow ) {
  // Check left sample
   sample[EAR_LEFT] = analogRead(A0);
   if (sample[EAR_LEFT] < MAX_SIGNAL) {
      if (sample[EAR_LEFT] > signalMax[EAR_LEFT]) {
         signalMax[EAR_LEFT] = sample[EAR_LEFT];
      } else if (sample[EAR_LEFT] < signalMin[EAR_LEFT]) {
         signalMin[EAR_LEFT] = sample[EAR_LEFT];
      }
   }
}

delay(50);

while ( (millis() - startMillis) < sampleWindow ) {
  // Check right sample
   sample[EAR_RIGHT] = analogRead(A1);
   if (sample[EAR_RIGHT] < MAX_SIGNAL) {
      if (sample[EAR_RIGHT] > signalMax[EAR_RIGHT]) {
         signalMax[EAR_RIGHT] = sample[EAR_RIGHT];
      } else if (sample[EAR_RIGHT] < signalMin[EAR_RIGHT]) {
         signalMin[EAR_RIGHT] = sample[EAR_RIGHT];
      }
   }
}

// max - min = peak-peak amplitude
peakToPeak[EAR_LEFT] = signalMax[EAR_LEFT] -
signalMin[EAR_LEFT];
peakToPeak[EAR_RIGHT] = signalMax[EAR_RIGHT] -
signalMin[EAR_RIGHT];
```

```
volts[EAR_LEFT] = double(peakToPeak[EAR_LEFT] * 5.0) /
MAX_SIGNAL;
volts[EAR_RIGHT] = double(peakToPeak[EAR_RIGHT] * 5.0) /
MAX_SIGNAL;

double diff = abs(volts[EAR_LEFT] - volts[EAR_RIGHT]);

Serial << "Max left " << signalMax[EAR_LEFT] << " Min left "
<< signalMin[EAR_LEFT] << endl;
Serial << "Max right " << signalMax[EAR_RIGHT] << " Min
right " << signalMin[EAR_RIGHT] << endl;
Serial << "peakToPeak left " << peakToPeak[EAR_LEFT] << "
peakToPeak right " << peakToPeak[EAR_RIGHT] << endl;
Serial << "V left " << volts[EAR_LEFT] << " V right " <<
volts[EAR_RIGHT] << endl;
Serial << " V diff " << diff << endl ;
```

18.3. Preparing the Borg to Host the Brain

The “brain” of the Borg is implemented in a Raspberry Pi. It hosts the programs to simulate intelligent behavior and human reactions. See Figure 18-12.

Figure 18-12. *The Raspberry Pi installation in the Borg's head was achieved by 3D-printing flexible support to keep the embedded Linux board in place. The Raspberry Pi controls a small yet powerful speaker glued to the throat*

The Pi Camera and speaker—the eyes and voice of the Borg—are controlled by the Raspberry Pi installed in the head of the mannequin, on the opposite side where the Arduino and motion mechanism lie.

Fixing these components in the reduced space offered by the torso made this part a complex and time-consuming job.

To fix the Raspberry Pi inside the head, facilitating the access of the cables to the left eye of the mannequin, I designed and 3D-printed a PLA support and bolted it to the sides of the head. To the opposite side of the Raspberry Pi, I kept space for a small amplifier connected to a smartphone speaker fixed to the outside of the neck and hidden by a scarf. See Figure 18-13.

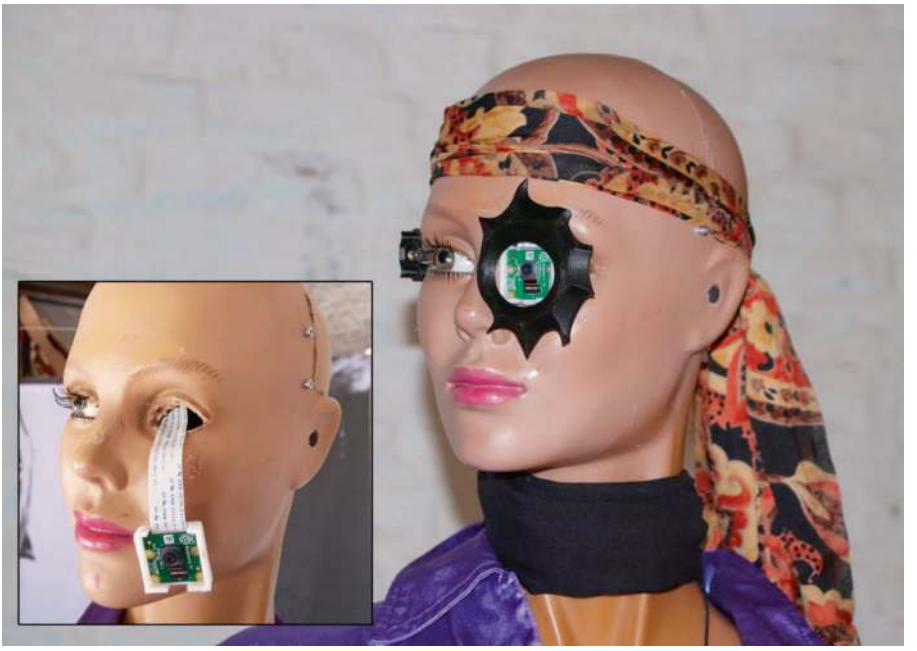

Figure 18-13. *The Pi Camera installed inside the Borg implant of Seven of Nine on her left eye*

18.3.1. Pi Camera Hosting

Note After Seven of Nine's humanization, she still retains some Borg implants, including the ocular implant over her left eye, because certain cybernetic enhancements were too integrated into her physiology to be safely removed. The ocular implant are from the extensive modifications the Borg made to her body, making its complete removal either too dangerous or impractical.

The Pi Camera is hosted in the center of Seven of Nine's left eye. Using a Raspberry Pi shield, the implant's white LED flashes and rotates the lights in sequence. In the project, I used a PiFace digital I/O board. This board is no longer available in the most recent Raspberry Pi versions (from Pi 4B), but it is not difficult to replace it with a simple relay circuit to power the LEDs through the Raspberry Pi GPIO. See Figure 18-14.

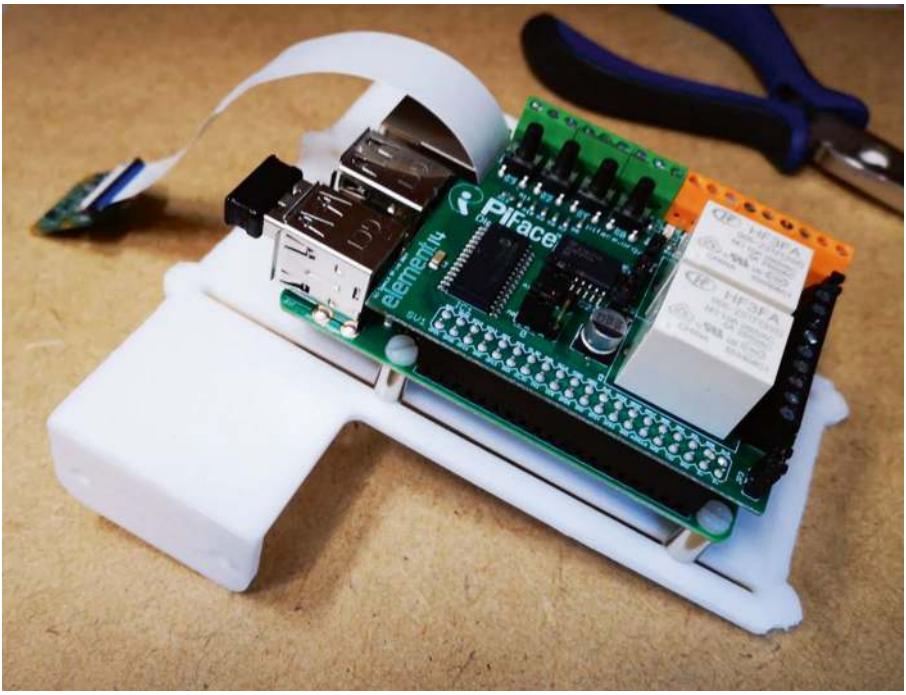

Figure 18-14. *The Raspberry Pi with the PiFace digital I/O installed on top and the Pi Camera connected through the flat cable*

18.4. The Raspberry Pi Modules

Thanks to the intensive use of separate processes—some of them running independently of human interaction—in the Linux Raspbian operating system and the processing power of the Raspberry Pi, I could make Seven of Nine mimic some human behaviors.

Indeed, as this project was my first prototype, many details and features required refactoring and optimization. But it worked!

Note The chapter repository contains all the components, STL files, documentation, details, and sources.

Figure 18-15. *The author and the Borg*

Index

A

E. Miglino, *Tales for Makers*, Maker Innovations Series,
https://doi.org/10.1007/979-8-8688-0080-1

B

C

D

E

F

G

H

I

J

K

L

M

N, O

P, Q

R

S

T

U

V

W, X, Y

Z