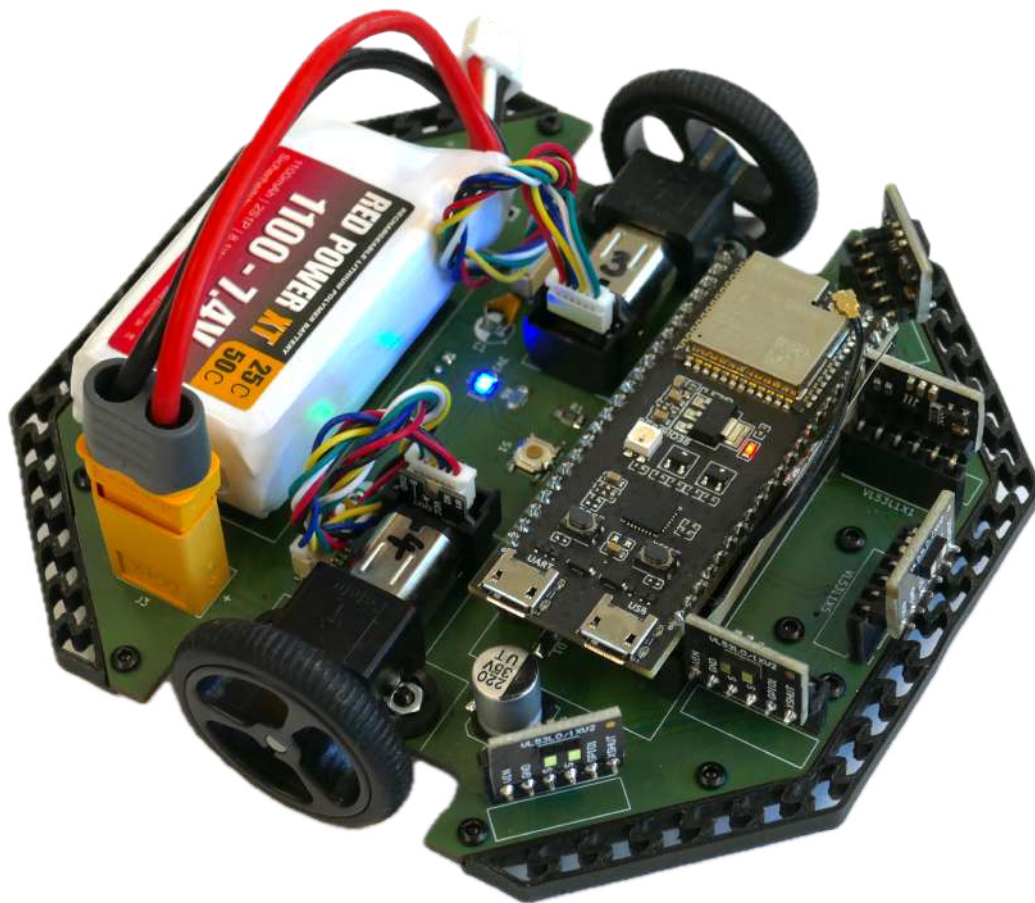


Algernon: Design of a Micromouse Robot With a Remote Debug Interface

Bachelor Thesis in Micro-Engineering



Regensburg, August 19, 2024

Océane Patiny

Supervisors

Prof. Dr. Martin Weiss (OTH)

M. Yves Chevallier (HEIG-VD)

Erklärung zur Bachelorarbeit

1. Mir ist bekannt, dass dieses Exemplar der Bachelorarbeit als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.
2. Ich erkläre hiermit, dass ich diese Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Regensburg, den 10. August 2024

Grammar and syntax

The grammar and syntax of this report were improved using the free version of the Elphas (<https://elephas.app/>) software v. 9.35. The chat model used is OpenAI's gpt-4.

Abstract

In this report, the comprehensive steps necessary for the design, assembly, and testing of a custom autonomous robot are detailed. The project drew inspiration from micromouse competitions, where small, autonomous, maze-solving robots compete to solve an unknown maze in the least amount of time. The primary focus of this project was the design of a micromouse from the ground up. This encompassed both electrical and mechanical design, along with the creation of a custom PCB. A physical prototype of the robot was successfully assembled and tested.

The robot is equipped with five Time-of-Flight (ToF) distance sensors, an accelerometer, and two brushed DC motors with magnetic encoders. The microcontroller utilized is an ESP32-S3, and a 2P1S LiPo battery serves as the power supply. On the software front, each sensor was independently tested and found to be functional. A rudimentary odometry task was implemented, allowing for the tracking of the robot's position. Speed regulation of the wheels was also incorporated. The maximal speed the robot can reach is 1.05 m/s, and its maximal acceleration is 1.5 m/s². Finally, a remote debug interface utilizing WiFi was developed to enable real-time visualization of the robot's status.

Acknowledgements

First and foremost, I would like to express my gratitude to Prof. Martin Weiss, who kindly agreed to be my supervisor at Ostbayerische Technische Hochschule (OTH) in Regensburg, Germany, despite never having met me before. I wish to thank him for his warm welcome to this school, which was brand new to me as an exchange student. I could not have brought this project to fruition without Prof. Weiss's Swiss counterpart, Mr. Yves Chevallier, who kindly agreed to be my supervisor at the School of Business and Engineering Vaud (the HEIG-VD), my home university.

I am also deeply grateful to all the professors of the HEIG-VD who made themselves available to answer my technical questions in their respective areas of expertise: Mr. Michel Etique, Mr. Michel Girardin, and Mr. Mikaël Krummen. Another person whom I would like to thank for always lending an attentive ear to my various struggles is my dad, who never seems to get bored of my robotic shenanigans. In addition, I would like to extend my sincere thanks to my friend Bo, who considered redoing some of my figures as an entertaining graphic design exercise.

Finally, special mention goes to all of my friends, old and new, who showed interest in my project, continuously asked for updates about my little robot, and never failed to encourage me when I felt like I would never manage to finish this project. Working on such an ambitious project in an unknown environment, without most of the tools I am accustomed to, indeed felt like jumping off a cliff... without a parachute.

Contents

Acknowledgements

1	Introduction	13
1.1	Outline of the Contests Rules	14
1.2	Motivations	15
1.3	Objectives	15
1.4	Planning	16
1.5	Documentation and Project Files	16
2	State of the Art	17
2.1	Price	17
2.2	Types of Motors	18
2.3	Configuration of the Motors and Wheels	21
2.4	Encoders	26
2.5	Distance Sensor Technologies	27
2.6	Inertial Measurement Unit (IMU)	31
2.7	Batteries	32
2.8	Microcontroller	34
2.9	Breakthroughs of Micromouse Contests	37
3	Solutions	41
3.1	Custom PCB	41
3.2	Robot's Size and Shape	42
3.3	Wheels Layout	42
3.4	Distance Sensors Layout	43
3.5	Distance Sensors Technology	44
3.6	Inertial Measurement Unit (IMU)	44
3.7	Motors, Drivers, Encoders, and Wheels	45
3.8	Architecture	46
4	Theory	47
4.1	Robot's Dimensions	47
4.2	Brushed DC Motor Fundamental Equations	48
4.3	Motor Dimensioning	50
4.4	Differential Drive Kinematics	57
4.5	Rotary Encoders	59
4.6	Wheels' Speed Measurement	60
4.7	Odometry	61
4.8	Speed Calibration for Feedforward Controller	64

4.9	PID Regulator	68
4.10	Robot's Speed Regulation	70
4.11	Wall Following	70
5	Electronics	73
5.1	Microcontroller	73
5.2	Motors	74
5.3	Encoders	75
5.4	Motor Driver	76
5.5	Distance Sensors	77
5.6	Buzzer	79
5.7	Battery	79
5.8	IMU	80
6	PCB Design	81
6.1	Design Principles	82
6.2	Voltage Regulation	83
6.3	Voltage Measurement	83
6.4	Distance Sensors' Pinout	84
6.5	Motor Driver Pinout	84
6.6	Reducing the Motors' Electrical Noise	85
6.7	Protection of the Power Lines	85
6.8	Debug Components	86
7	Mechanical Design	87
7.1	Custom Caster Wheels	87
7.2	Wheels	88
7.3	Battery Mounting	88
7.4	Bumpers	89
8	A Custom Maze Design	91
8.1	Contest Maze Dimensions	92
8.2	Walls	92
8.3	Corners	94
8.4	Bridges	95
8.5	Possible Improvements	96
9	Software	97
9.1	Programming Environment	97
9.2	Base Project	97
9.3	Multitasking and Real Time	97
9.4	State	98
9.5	Quadrature Encoding and Speed Measurement	99
9.6	Odometry	100
9.7	Distance sensors	100
9.8	Motors' Speed Calibration	100
9.9	Motors' Control	102
9.10	Robot's Speed Control	104
9.11	Serial Interface	105

CONTENTS

9.12 WiFi Debug Interface	105
10 Results	109
10.1 Hardware	109
10.2 Software	110
10.3 Maximal Speed and Acceleration	111
10.4 Price	112
10.5 Autonomy	112
10.6 Review of the Project Planning	113
10.7 Improvements	114
10.8 Perspectives	114
11 Conclusions	115
Bibliography	121
Appendix	123
1 Experiment: Maximal Speed and Acceleration	123
2 Experiment: Measurement of wheels speed	124
3 Experiment: Motor's speed vs voltage	125
4 PCB schematics and project planning	128

Chapter 1

Introduction

*They called the mouse Algernon.
Algernon was in a box with a lot of
twists and turns like all-kinds of
walls and they gave me a pencil
and a paper with lines and lots of
boxes. On one side it said START
and on the other end it said
FINISH.*

Flowers for Algernon
- Daniel Keyes

The Amazing MicroMouse contest is a robotics contest first launched in the IEEE Spectrum magazine in 1977. The principle of the contest can be explained very easily: you have to create an autonomous robot (a mouse) that must solve an unknown maze as quickly as possible [1].

Since the first edition, the contests have continually gained more attention, spreading worldwide. By 2014, there were already around 100 contests held that year, and their popularity is always increasing [1]. Throughout the years, hundreds of micromice have been built, featuring a vast range of inventive solutions to various challenges. Research is also conducted in the field, often leading to scientific publications.

Whether university students, makers, or graduated engineers, anyone can have fun working on a micromouse project. Depending on the individual's skills, it is possible to buy a ready-to-program kit, or alternatively, fully build the micromouse from scratch. It is, therefore, an extremely versatile project idea, allowing people with more or less previous knowledge to engage with robotics in a fun way [69]. Indeed, there is something very satisfying about making a small robot that can navigate by itself. It is for these various reasons that I decided to tackle it myself, using my bachelor thesis as a perfect opportunity to dive deep into the micromouse world. The Amazing MicroMouse contest, a robotics competition, was initially introduced in the IEEE Spectrum magazine in 1977. The premise of the contest is straightforward: participants are required to construct an autonomous robot (termed a mouse) that can navigate an unfamiliar maze in the shortest time possible [1].

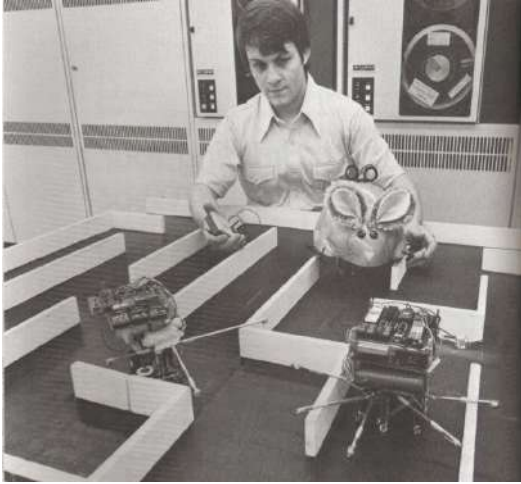


Figure 1.1: Some of micromice that participated in the very first contest (1977-1979) [30]

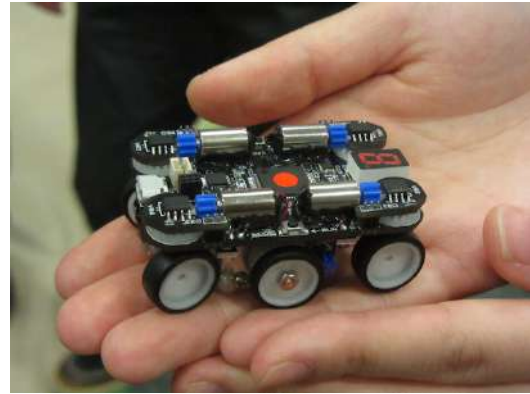


Figure 1.2: A mouse from the All Japan Half-Size micromouse contest 2017 [34]

1.1 Outline of the Contests Rules

The exact rules for micromouse contests vary slightly from one contest to another, but the principle remains the same. The robot must be fully autonomous and should therefore not communicate with the outside. The maze layout is kept secret until the beginning of the competition. It is therefore not possible to program the maze beforehand. The maze is composed of 16×16 square cells with 18 cm sides. The walls are 12 mm thick and 50 mm high. The start of the maze is always in one of the corners, and the goal is a 2×2 cells space in the center of the maze. Figure 1.3 shows a typical maze layout. Each team has 10 min of access to the maze and the fastest run from start to finish during this time is registered [51].

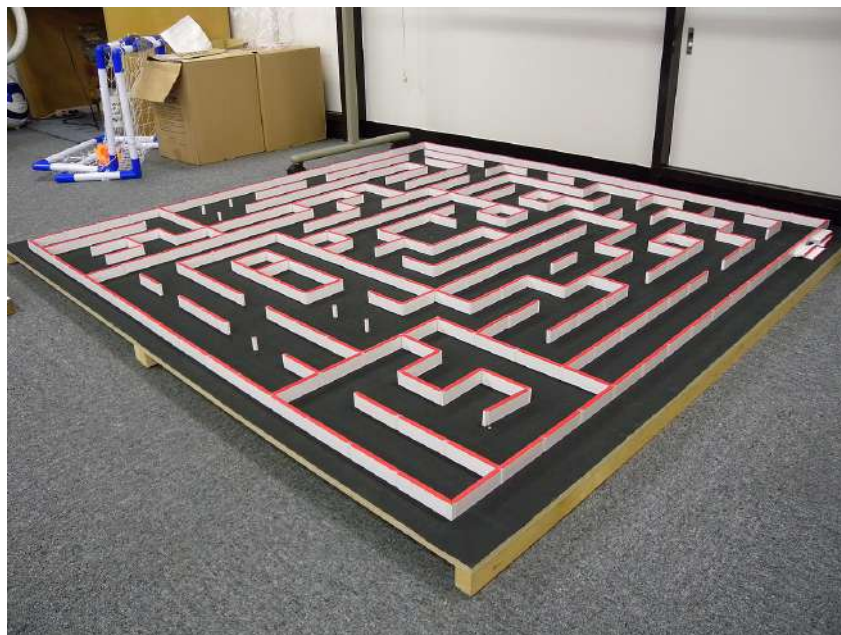


Figure 1.3: Example of a full-size micromouse maze [15]

1.2 Motivations

For me, the discovery of the amazing micromouse contests happened through Veritasium's excellent video "The Fastest Maze-Solving Competition On Earth," released on 24.05.2023 on YouTube [29]. Like all of Derek Muller's videos, the content is well-researched, interesting, and entertaining, and it immediately intrigued me. Conveniently, I was searching for a thesis subject at the same time, and it became obvious to me that I wanted to work on the design of my own micromouse.

Micromice are interdisciplinary projects that encompass all fields of micro-engineering: electronics, mechanics, programming, and robotics. For this reason, it appears as the perfect project for my thesis, even though it is very ambitious. To me, the bachelor thesis is an opportunity to experiment and step out of my comfort zone, even if I risk not achieving all I want to do. Indeed, it is probably one of the last situations where we can "fail" without it mattering too much.

1.3 Objectives

The goal of this project is to design a micromouse from scratch in 6 months. To frame the project a little bit more, the goal is to get as close as possible to having a robot that can solve a standard micromouse maze autonomously. It is likely that this objective is too ambitious considering the time available, however, it can serve as a clear guideline throughout the project. Since the time span we have to realize a project of this complexity is relatively limited, we will focus on the electronic and mechanical design, as well as the communication with the different sensors and the control of the robot movements. The main aspects of the project will hence be the following:

- **State of the art:** See what already exists, what technologies are commonly used, and what the well-established standards are.
- **Choice of the components:** List the different electronics needed and find the commercially available components.
- **PCB design and assembly:** Firstly, realize the PCB schematic with all the required components and then create the layout of the PCB. Order the custom PCB, assemble it, and test it.
- **Retrieve sensors data:** The micromouse will integrate a variety of sensors. We must be able to communicate with all of them and retrieve the data.
- **Control of the motors:** Be able to control the motors to move at a desired speed. This step will require the implementation of PID regulators. We will also need to implement an odometry algorithm, which allows us to know the robot's position, orientation, and speed.
- **Wall following:** The robot should be able to navigate in a corridor without bumping into walls. This implies a regulator on the robot's position in the corridor.

Additionally, we will try to develop good debugging tools as we go. Indeed, being able to easily see the current state of the robot, the different commands that are processed, and the sensor data will allow us to better understand the robot's behavior and therefore find potential bugs.

1.4 Planning

The bachelor thesis is a 450 h work project led during the last semester of the bachelor studies. This project was realized during an exchange semester at the Ostbayerische Technische Hochschule, in Regensburg, Germany. It officially started on the 18th of March and ended on the 18th of August 2024 (even though in practice we started one month earlier). In order to ensure the feasibility of the project, all of the tasks to be realized were listed and a plan spreading them over the duration of the project was created. In this process, ambitious decisions were made (short deadlines, little time for errors), while still trying to make honest estimations of the time each step would take. The resulting plan can be found in the annexes at the end of this document (see Figures 11.6, 11.7 and 11.8). The core milestones are, however, summarized in Table 1.1.

Dates	Nb. weeks	Type	Focus
19.02 - 17.03	4	Doc.	State of the art, documentation and architecture choice, custom maze design and manufacturing
18.03 - 12.05	8	Hardware	Electronic design, PCB design and manufacturing, assembly of robot v1
13.05 - 02.06	3	Hardware	Bug fixes in the hardware, final robot hardware done
03.06 - 30.06	4	Software	Test of all the sensors and programming of motor control task, driving straight in a corridor, create WiFi debug system
01.07-04.08	5	Software	Turning, following a predefined path in maze, implement floodfill algorithm
05.08-18.08	2	Doc.	Report

Table 1.1: Summary of the project planning (Doc. stands for documentation).

1.5 Documentation and Project Files

Throughout this project, GitHub has been used as a project management tool, as well as a version control tool. All of the data related to this project can be found in the three following repositories:

- Hardware related files, documentation and MatLab scripts:
<https://github.com/opatiny/micromouse>
 (commit a043f868b5adddb9797485b194d83862c804f196)
- C++ software for the micromouse:
<https://github.com/opatiny/ms-software>
 (commit d6f4fc6eaca07a7fbef5fb862d9489f111c2e57a)
- TypeScript code for the debug web page:
<https://github.com/opatiny/ms-webpage>
 (commit becbb41689782ab37acf18b62f3899d26b75d5da)

All of our work is open-source: the hardware repository is under CERN-OHL-P-2.0, whereas the software is under MIT.

Chapter 2

State of the Art

To be as exhaustive as possible in this state of the art, 10 different micromouse projects have been analysed. This allowed us to see the solutions used for the different parts of the robot, as well as establish which technologies are the most widespread. The list of projects studied, their characteristics, as well as links to the web pages where the information has been found can be accessed using the link in the footnote.²

2.1 Price

As the idea for this project was a personal proposition, there are no fixed requirements on any aspect of it. We do, however, need to define an approximate price target for the robot, since that will limit the technologies we can use for the different components. In order to set a price range, various existing mobile robotics and micromouse kits were analyzed. Table 2.1 summarizes some of these projects, as well as their respective prices. Based on this table, we could deduce that a reasonable price for our micromouse would be around €150, similar to the Project Futura robot. Indeed, in terms of features, it is the robot that is most similar to what we would like to realize in this project.

Name	Description	Price [€]
Thymio II	Open-source educational differential drive mobile robot developed in Switzerland	250
PICAXE 'PICone' Micromouse	Self-assembly micromouse kit	125
Micromouse T01 Kit	Self-assembly micromouse kit based on the UKMARBOT platform	75
Project Futura	Pre-made high-performance low-cost Mouse kit used as a teaching tool	150

Table 2.1: Breakdown of the robot's price, excluding all shipping costs.

²You can find the recapitulative state of the art here: <https://tinyurl.com/ms6mrj4p>

2.2 Types of Motors

The following sections describe some of the motor technologies that are used in existing micromouse robots. Their characteristics are outlined, and their advantages and drawbacks are summarized.

2.2.1 Brushed DC Motors

Brushed DC motors (BDC motors) are some of the most widespread motors used in robotics applications because of their low price and how easily they can be controlled [33]. Indeed, simply applying a DC voltage to the motor's terminals allows it to spin. Moreover, since the motors use direct current, they are particularly suitable for battery-powered devices.

The speed of a DC motor is directly proportional to the effective voltage applied to it. A pulse width modulation (PWM) signal can therefore be applied to the motor in order to control its speed [57]. Since the power required for the motors cannot be directly output by the microcontroller, a full-bridge motor driver has to be used. This component allows the low-power circuit of the microcontroller to be linked with the high-power circuit of the battery.

A disadvantage of these motors, however, is that there is no way to know at what speed the motor is actually spinning without using additional sensors. For this reason, encoders are added to the motor's shaft. The encoders allow the rotation of the shaft to be measured. The existing types of encoders are discussed in Section 2.4.

Based on the projects that we studied, we can see that BDC motors are the most widespread in micromouse projects (they are used in 7 out of 10 projects).

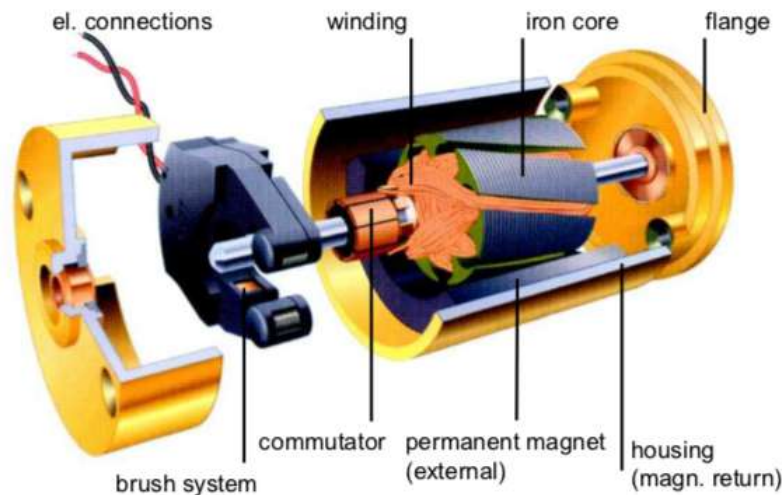


Figure 2.1: Schematic of a brushed DC motor [9]

Advantages	Drawbacks
High power to weight ratio	Absolute position unknown, requiring encoders to monitor motor movement
Mechanical commutation (no electronic controller needed)	Motor brushes tend to wear down, limiting motor lifetime
Often found pre-assembled with a gearbox	Adding a gearbox to a motor increases angular play
Wide variety of models varying in size, torque, speed, etc.	

Table 2.2: Advantages and disadvantages of brushed DC motors.

2.2.2 Coreless DC Motors

Coreless DC motors closely resemble standard BDC motors, with the exception that the rotor is self-supporting and not wound around an iron core. This is illustrated in Figure 2.2. The absence of an iron core significantly reduces the rotor's weight and its inertia, thus improving the motor's dynamics.

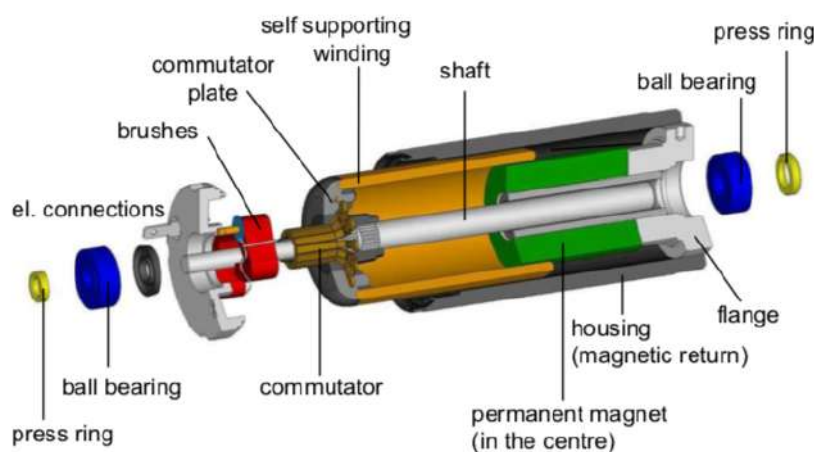


Figure 2.2: Schematic of a coreless DC motor [26]

Advantages	Drawbacks
Large torque to weight ratio compared to other BDC motors	Uses brushes, which limit the motor's lifespan
Can be controlled using PWM like other BDC motors	Requires encoders
Smaller than conventional DC motors	

Table 2.3: Advantages and disadvantages of coreless DC motors.

2.2.3 Stepper Motors

Stepper motors are constructed with pairs of coils on the stator and a toothed, magnetised rotor, as illustrated in Figure 2.3. By activating the pairs of coils in sequence, the teeth of the rotor align with the coils, resulting in a movement of one step. Typical stepper motors have 200 incremental steps per full rotation, implying that each step is 1.8° [36]. Therefore, unlike brushed DC motors, the movement of a stepper motor is not continuous but discrete. The motor is controlled using electrical pulses, the frequency of which determines the motor's speed [49]. The control is open-loop, meaning there is no need for encoders. However, if the motor misses steps, which can occur if the torque is too large, there is no way to determine the motor's absolute position anymore.

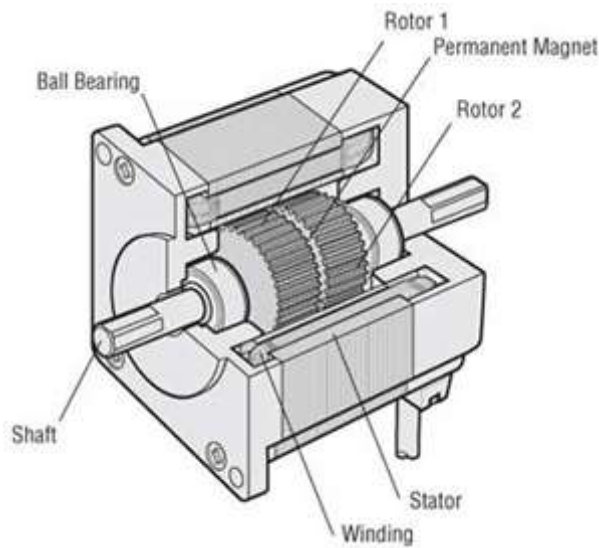


Figure 2.3: Schematic of a stepper motor [14]

Advantages	Drawbacks
No accumulation of error on the motors' position as long as no steps are missed	Full-step control of the stepper limits the minimal angular displacement to 1 step (typically 1.8°)
No need for encoders	No feedback to indicate missed steps
Overload safe: the motor can't be damaged by applying high torque	Prone to vibration and noise
No brushes, which are prone to wear	High power consumption even when the motor is not moving

Table 2.4: Advantages and disadvantages of stepper motors. This table is largely based on [37].

2.3 Configuration of the Motors and Wheels

All ten projects that we have studied are based on a differential drive. Differential wheeled robots are equipped with two wheels, each connected to their own motor. The wheels are controlled separately. The combination of the motors' speeds can result in forward motion, turning, and backward motion of the robot, as illustrated in Figure 2.4. According to our research, most competition-winning micromice use a variation of a differential drive [53]. However, we will discuss one interesting project using a belt-based drive in Section 2.3.4.

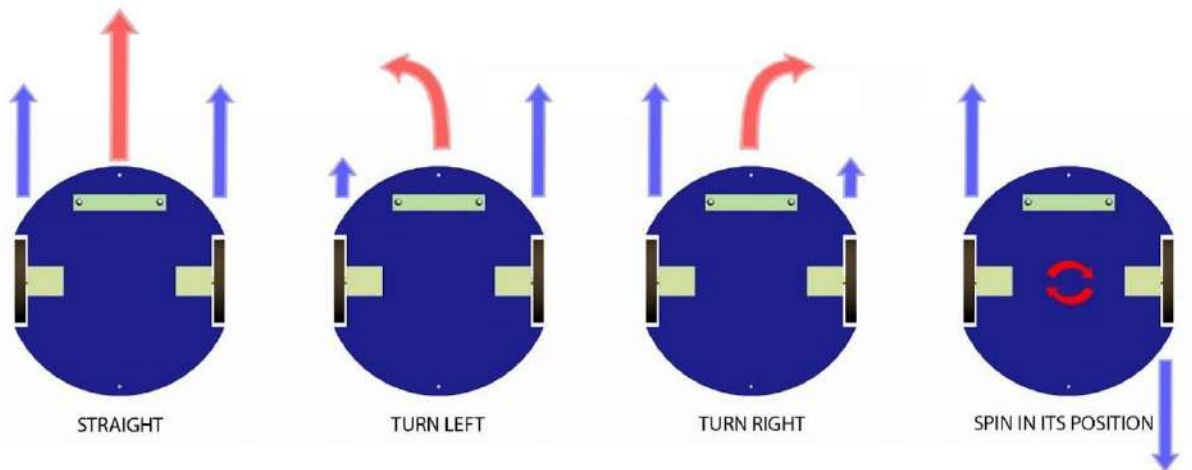


Figure 2.4: Differential drive robot principle: The combined speeds of the wheels allow the robot to drive straight, turn, and rotate on the spot [23]

Despite the drive principle being relatively similar among most of the micromice, there is still a wide variety of motor configurations, which will be detailed in the following sections.

2.3.1 Wheels Directly on Motors' Shafts

The simplest way to mount the wheels is to connect them directly to the motors' shafts. This approach is most commonly used in beginner micromouse projects, as well as in some of the open-source kits that can be found online, such as **ukmarsbot** by the UK Micromouse and Robotics Society (see Figure 2.5). In the case of this robot, the motors are equipped with a D-shaped output shaft. By using a wheel with a D-shaped bore, the wheel can be mounted without any screws, simply by using the friction between the shaft and the bore (this is also visible in Figure 2.5).

Advantages	Drawbacks
No additional transmission means fewer parts that can break Since no transmission is used, there is more space on the PCB for other components	Motors' length limited to half of the robot's width

Table 2.5: Advantages and disadvantages of wheels directly on the motors' shafts.

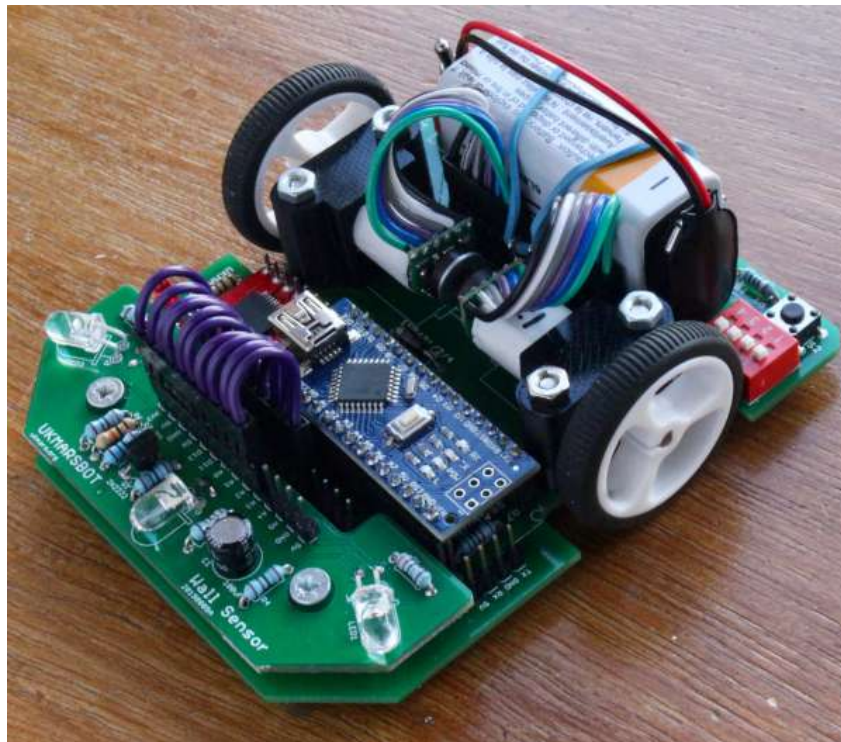


Figure 2.5: Ukmarsbot's wheels are directly attached to the motors' shafts. [25]

2.3.2 Wheels on Axes Parallel to Motors' Shaft

Instead of being directly on the motors' shaft, the wheels can be mounted on a parallel axis. There are two possibilities: either the two motors are mounted on the same side of the wheels, like in Figure 2.6, or they are placed on the opposite side of the wheels. The second possibility allows for longer motors, spanning almost the entire width of the robot. The motors can thus be more powerful, resulting in better performance of the final robot. Since the motors are no longer coaxial with the wheels, a transmission must be used to transfer the torque to the wheels. The transmission would typically use spur gears or belts. This design also has another advantage, which is that a reduction can be added at the level of the transmission, allowing optimization of the wheels' speed and torque for the micromouse.

Advantages	Drawbacks
Less restriction on motors' length	Additional mechanical play if gears are used
Possible additional reduction with gears or belt	Requires the design of more mechanical components
	Takes up more space on the PCB

Table 2.6: Advantages and drawbacks of wheels on axes parallel to the motors

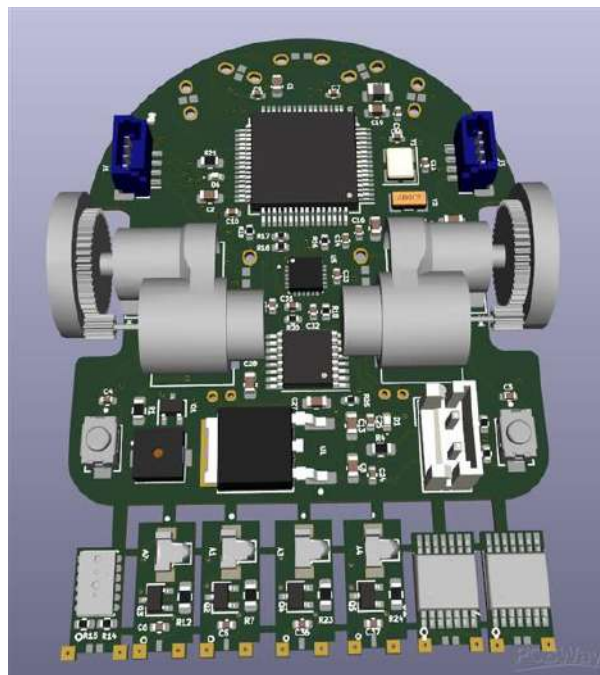


Figure 2.6: Mushak, by Chimnay Lonkar, uses wheels on axes parallel to the motors' shafts. [16]

2.3.3 Wheels Perpendicular to Motors' Shaft

Another interesting approach is to have the motors perpendicular to the wheels' axes. There are two different variants: the motors can either be placed vertically (see Figure 2.8), or along the longitudinal axis of the mouse (as shown in Figure 2.7). In both cases, a right-angle transmission is required (typically conical gears). This approach is interesting because it allows for the use of longer motors. In the case of the vertical motors, it also leads to a smaller footprint of the motors on the PCB, leaving more space for other components.

Advantages	Drawbacks
Less restriction on motors' length	Conical gears required (additional mechanical play)
Possible additional reduction at the level of the conical gears	The wheels can't be placed in the middle of the robot

Table 2.7: Advantages and drawbacks of longitudinal motors

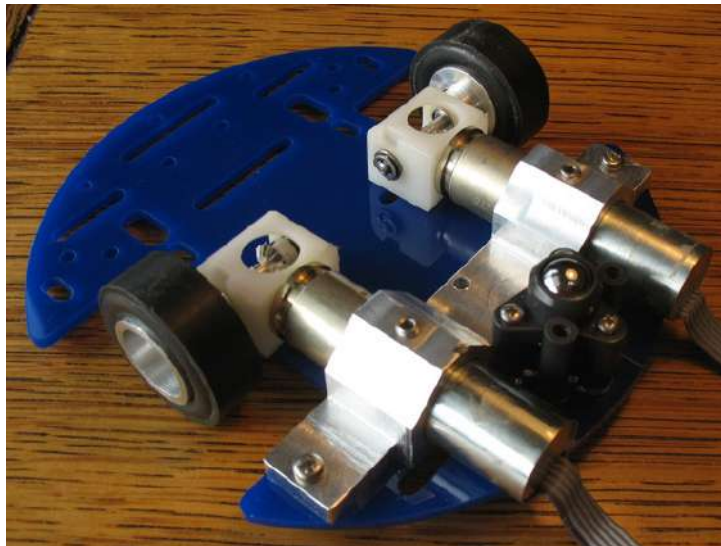


Figure 2.7: Jim Remington's micromouse has motors along the longitudinal axis of the robot and uses conical gears for the transmission of the torque to the wheels. [22]

Advantages	Drawbacks
Motors take less space on the PCB	Conical gears required (additional mechanical play)
No physical limitation on motors' length	The motors being vertical raise the robot's center of mass
Possible additional reduction at the level of the gears	

Table 2.8: Advantages and drawbacks of vertical motors



Figure 2.8: The Picaxe Microbot kit uses vertical motors and conical gears to transmit the movement to the wheels [10]

2.3.4 Belt Based Systems

Instead of using gears as in all the previous mice, it is also possible to use belts for transmission. Belts have multiple advantages:

- They are very lightweight.
- They allow for the addition of a reduction in the same way as with gears.
- The motor's shaft and the wheel axis can be further apart than with spur gears.
- Thanks to the tension in the belt, there is little additional mechanical play.

These systems, however, require an additional tensioning mechanism, which can make them more complex to design. The Shi On micromouse project (see Figure 2.9) is one of the few that actually implements belts in their design. In the case of this mouse, three pairs of wheels are used, and the front and back ones are oriented by a vertical motor through a belt transmission. This is one of the rare projects not using a differential drive approach. Instead, the robot has "drive motors", responsible for acceleration, and "steering motors", used to make the robot turn [18].

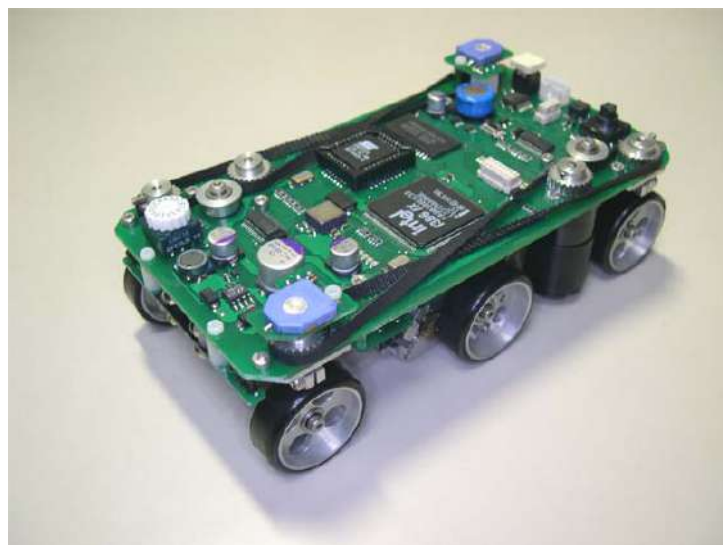


Figure 2.9: The Shi On micromouse uses 3 pairs of wheels. The motors are vertical and the motion is transmitted using belts that actuate the front and the back wheels. [18]

2.4 Encoders

Rotary encoders are necessary to measure the displacement of the DC motors. Knowing the movement of each wheel is essential to determine the robot's speed and orientation. Two main types of encoders used: magnetic and optic. More recently, capacitive encoders have also been developed, but they are not as widespread. Encoders generally output a digital signal with a certain number of pulses per full rotation of the motor. This is referred to as the "counts per revolution" (CPR). The higher the CPR, the better the resolution of the encoder.

2.4.1 Magnetic Encoders

Magnetic encoders are based on Hall effect sensors. They consist of a rotary magnetic disk with alternating south and north poles, and a magnetic sensor. When the disk rotates, the sensor detects variations in the magnetic field and outputs a digital signal with the transitions from positive to negative magnetic field. Counting the number of transitions allows to determine the rotary displacement of the motor. Some magnetic encoders, like the RM08 Miniature Rotary Magnetic Encoder [61], can also output a sinusoidal signal, which would need to be processed to deduce the rotary displacement.

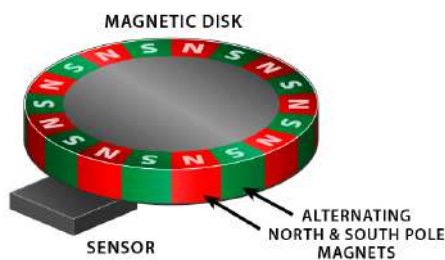


Figure 2.10: Principle schematic of a magnetic encoder [11]

2.4.2 Optical Encoders

Optical encoders are based on light instead of magnetic fields. In this case, the rotating disk is perforated with thin radial slits. As the light passes through the slits, it is detected on the other side by a light sensor, thus allowing to determine the incremental displacement. Unlike magnetic encoders, optical encoders are sensitive to ambient light and contamination (dust, oil) than magnetic encoders [11].

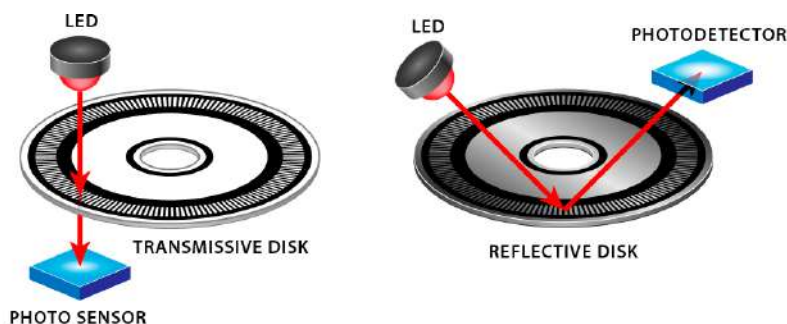


Figure 2.11: Principle schematic of an optical encoder [11]

2.5 Distance Sensor Technologies

For an autonomous robot, sensing its environment is crucial to deduce its location and eventually build a map of its surroundings. Of course, not all autonomous mobile robots (AMRs) have the same requirements. Depending on the application, different technologies should be used, each providing varying degrees of information about the robot's environment. If we venture beyond the scope of micromice, where the maze is a relatively standardized environment that is easy to navigate, the task of simultaneous localization and mapping (SLAM) becomes significantly more challenging. Robots need to process vast amounts of data and extract useful information from it. SLAM is a well-known challenge in mobile robotics, as presented by Thrun et al. in their book *Probabilistic Robotics*.

"In SLAM, the robot acquires a map of its environment while simultaneously localizing itself relative to this map. SLAM is significantly more difficult than all robotics problems discussed thus far: It is more difficult than localization in that the map is unknown and has to be estimated along the way. It is more difficult than mapping with known poses, since the poses are unknown and have to be estimated along the way." [8]

The typical sensors used on these more generic robots are the following [4]:

- 2D or 3D LiDAR (Light Detection and Ranging)
- Camera
- Stereo camera: uses left and right images to get an idea of the distance
- Infrared sensor
- Ultrasound sensor

In the case of micromice, we must consider that the microcontroller is relatively computationally limited, and therefore sensors generating a lot of data to process are not ideal. This is why the vast majority of micromouse projects use some sort of distance sensors. These generally fall into one of three categories: ultrasound sensors, infrared sensors, or time of flight sensors.

2.5.1 Ultrasound Sensors

Ultrasound sensors consist of an emitter (transducer) that generates ultrasonic waves (over 20 kHz), and a sensor, which acts as a microphone. The device emits short bursts of sound and measures the time required for the sensor to detect the reflected wave. Since the duration, as well as the speed of sound, are known, the distance to the obstacle can be calculated [50]. The output signal of the sensor is typically an analog voltage that is directly proportional to the distance to the object. It is therefore easy to use with a microcontroller. Ultrasonic sensors are an interesting technology because they are independent of ambient light, presence of dirt, target color, and material (generally). However, they can be affected by soft material which would absorb the sonic wave. These sensors would not be ideal when used with targets that are not perpendicular to the sound wave, as it would not bounce back towards the receiver. In addition, if many sensors are

used, there can be crosstalk issues, which means that the signal of one sensor will be detected by another one, leading to aberrant readings [50]. In the case of micromice, these sensors are rarely used due to their bulky size. Out of the ten projects on our list, only one used ultrasound sensors.

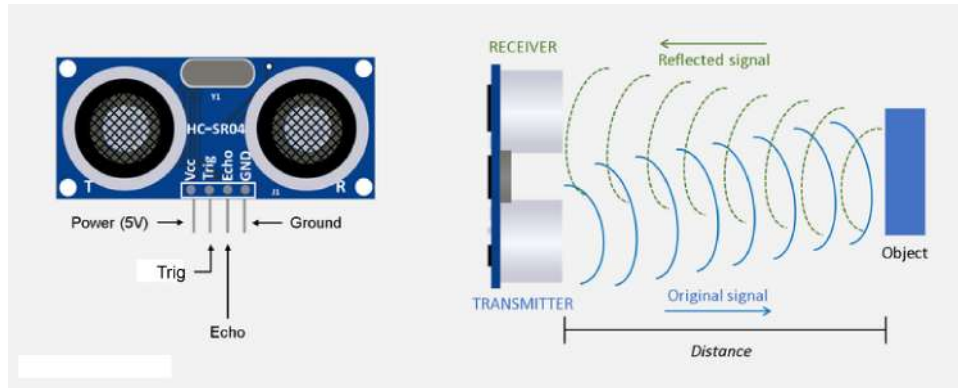


Figure 2.12: Schematic of an ultrasonic sensor [17]

Advantages	Drawbacks
Unaffected by dust, color, target material	Target orientation can impact readings
Relatively inexpensive (around 4 euros)	Dimensions: Typical board size is 45x20x15 mm
Longer range than infrared sensors (up to 4 m)	Possible crosstalk issues when multiple sensors are used

Table 2.9: Advantages and disadvantages of ultrasound sensors.

2.5.2 Infrared Sensors

Infrared distance sensors consist of an infrared (IR) LED and a photo-transistor that acts as a light detector. Similar to ultrasound sensors, the reflected light is measured. The intensity of the light returned by the sensors holds the information about the distance to the target. These sensors can be custom-built from an IR LED and a photo-transistor. They are also compact and inexpensive: both the LED and the photo-transistor cost a few tens of cents. A voltage divider allows the reading of the analog voltage returned by the sensor and deduces the intensity of the reflected light and therefore the distance. However, this technology has several issues. For instance, it is very sensitive to ambient light, dust, and the color of the target, all of which will impact the amount of light returning to the receiver. Camera auto-focus systems, which are also infrared, can interfere with the sensors. Furthermore, the orientation of the target is critical, as the amount of reflected light varies greatly with the angle of the surface [43].

Infrared distance sensors are the most widely used in micromouse projects (8 out of 10).

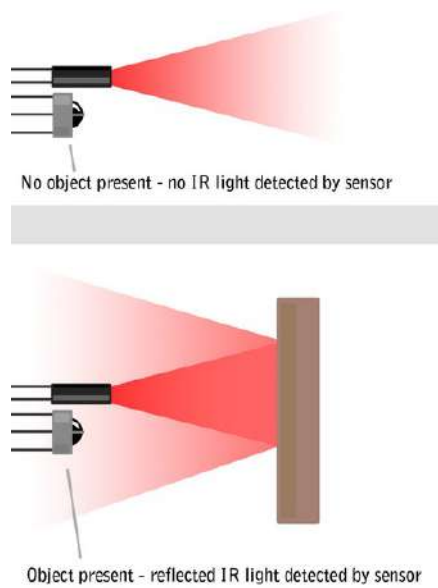


Figure 2.13: Principle of an infrared distance sensor [12]

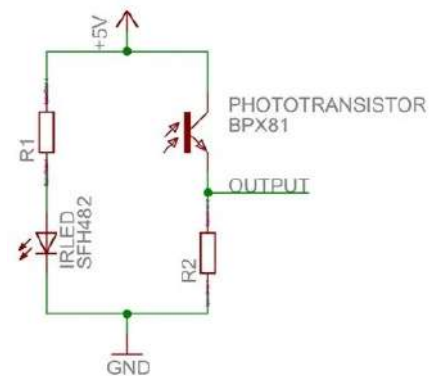


Figure 2.14: Typical electrical circuit of an IR sensor [12]

Advantages	Drawbacks
No additional PCBs required: the LED and photoresistor can be bent at 90°	Target orientation impacts readings
Inexpensive (around 20 cents for emitter + receiver)	Sensitive to ambient light, dust, target color
Fast readings	Possible crosstalk issues with multiple sensors

Table 2.10: Advantages and disadvantages of infrared sensors.

2.5.3 Time of Flight Sensors

Time of Flight (ToF) sensors is a term used for IR distance sensors. Instead of measuring the amount of light reflected, the time for the light to go from the emitter to the receiver is measured. There are two main ToF sensor technologies. The first variant emits light with modulated intensity, and the intensity of the reflected light is measured. By measuring the phase difference between the emitted and detected signals, the distance to the target can be deduced. These sensors are called indirect ToF. The second type of sensors is called direct ToF sensors. In this case, the time for the emitted light to bounce back to the receiver is measured [31]. The fact that this technology works for distances of only a few centimeters is really impressive, since a round trip for a 1 cm distance would only be 67 ps [63]. The distance d is then simply derived from the following formula, where c is the speed of light and Δt the time for the light to bounce back:

$$d = \frac{\Delta t \cdot c}{2} \quad (2.1)$$

Time of flight sensors can directly output the measured distance. Many ToF sensors can be interfaced using the I²C protocol [63], which is supported by most microcontrollers.

There are two possibilities: either only buy the distance sensor chip, or buy a shield integrating it. The issue is that the component itself is a very compact SMD package and can only be soldered with an oven (difficult to do without appropriate materials). Moreover, the sensors have to point perpendicularly to the robot's main PCB, which cannot be achieved with the SMD chips. Buying a fully integrated board allows the sensors to be added to the robot using standard pin headers. One issue, however, is finding ToF with the smallest possible PCB, since the space on the robot is limited.

This type of sensor is used in some beginner and intermediate micromouse projects, whereas competition-winning projects prefer infrared sensors. Of all the projects we analyzed, only 1/10 used ToF sensors.

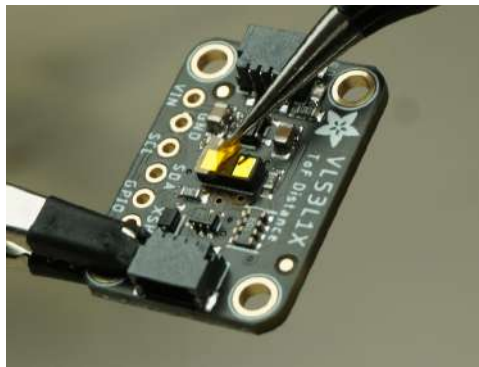


Figure 2.15: The VL53L1X time of flight distance sensing board by Adafruit (board around 25x20 mm) [19]

Advantages	Drawbacks
Fully integrated device with I ² C communication	Expensive compared to IR and ultrasound alternatives (around €10)
Readings are independent of the surface orientation	I ² C communication is slower than a simple ADC with an infrared detector. Can limit the robot's reactivity.
Range from 30 to 4000 mm	Depending on the model, the devices have a minimum range of a few centimeters!
Relatively small (compared to ultrasound)	

Table 2.11: Advantages and disadvantages of ToF sensors.

2.6 Inertial Measurement Unit (IMU)

Inertial measurement units are devices that are designed to measure an objects motion. They typically integrate an array of different sensors, which allow to measure the acceleration, rotation and velocity [52]. IMUs are often integrated in mobile robots as a relative positioning technology, which means that the current position depends on the previous ones. Indeed, the position and orientation of the robot can actually be deduced from the IMUs data by integrating it. Double integration of the acceleration results in the robot's position, whereas integrating the angular rate yields to the angle of the robot. This positioning method is known as Inertial Navigation System (INS) [3]. One issue with this positioning method, however, is that the errors accumulate over time, meaning that the accuracy of the position gets worse as the robot moves. On the other hand, it has multiple advantages:

- The IMU can sense movements in three dimensions, unlike odometry, which requires a contact with the floor at all times.
- IMUs are compact sensors, which do not require external connection for the positioning (on the contrary of GPS, for instance). This allows to use them indoors.
- Unlike odometry, the IMU is not impacted by the wheels slipping on the floor.

Adding an IMU to a micromouse hence provides useful data on the robot's state. Especially, it allows to detect when the wheels are slipping, and correct the estimated position accordingly. Out of the 10 project that we studied, 5 used an accelerometer.

2.7 Batteries

Since the robot must be autonomous, it requires its own energy source, in this case, batteries. Below is a summary of the most significant characteristics of a battery:

- Capacity: The amount of stored energy
- Nominal voltage: The voltage of the battery when charged
- Maximum continuous discharge current: The maximum continuous current that the battery can provide
- Maximum intermittent discharge current: The maximum current that can be drawn from the battery for a brief period

Choosing the most suitable battery technology can be challenging, regardless of the application. This is why the paper *Energy Sources of Mobile Robot Power Systems: A Systematic Review and Comparison of Efficiency* by Mikolajczyk is particularly intriguing, as it offers quantitative data about different battery chemistries, as well as a guide to selecting the appropriate battery chemistry [5]. Figures 2.12 and 2.16 display some of the most relevant images from the paper.

In the case of micromice, the weight-to-capacity ratio must be as small as possible. Indeed, a lower overall mass will allow greater acceleration of the robot. According to 2.16, the most optimal chemistry is therefore lithium polymer. From the projects we studied, 9 out of 10 used LiPo batteries. The final project used a nickel-metal-hydride (NiMH) battery.

	Lead-Acid	AGM	Gel	NiMH	LiPo	LiFePO ₄
Specific energy [Wh/kg]	35–40	35–40	35–40	60–120	100–265	90–160
Specific power [W/kg]	180	180	180	250–1000	245–430	2000–4500
Charge current [C]	0.2 C	0.25 C	0.25 C	1 C	1 C	1 C
Discharge current [C]	0.2 C	0.25 C	0.25 C	5 C–15 C	5 C	30 C
Self-discharge per month [%]	10–15	3–4	3–4	0.08–2.9	0.3	0.3
Max cell voltage [V]	2.15	2.15	2.15	1.4	4.2	3.65
Nominal cell voltage [V]	2.1	2.1	2.1	1.2	3.7	3.7
Min cell voltage [V]	1.8	1.8	1.8	0.9	2.7	2
Cycle durability [cycles]	350	500	500	180–2000	500	1200–2000
Max discharge capacity [%]	50	20	20	0	3	3
Operating temperatures [°C]	–35 to +50	–40 to +49	–20 to +45	–20 to +45	–20 to +60	–30 to +80
Charge temp range [°C]	–20 to +50	–20 to +50	–20 to +50	0 to +45	0 to +45	+5 to +45
Price [EUR/Wh]	0.14	0.21	0.26	0.63	1.04	2.96

where: AGM—Absorbent Glass Mat battery, Gel—Gel battery, NiMH—Nickel-metal-hydride battery, LiPo—Lithium-ion-polymer battery, LiFePO₄—Lithium iron phosphate battery.

Table 2.12: Battery properties depending on the chemistry [5]

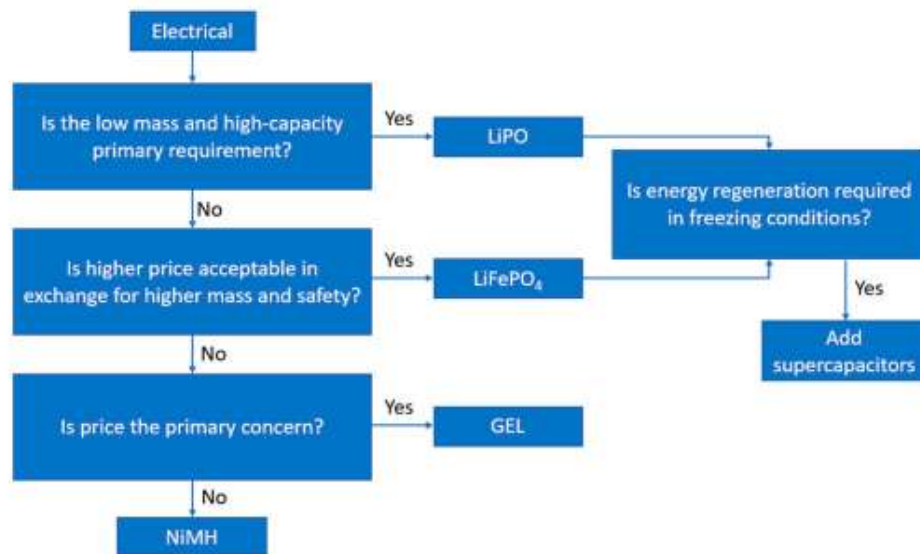


Figure 2.16: Procedure to choose the most suitable battery chemistry [5]

2.8 Microcontroller

The choice of the microcontroller unit (MCU) for the micromouse will impact the programming language used for the project, its ease of debugging, and the computational power of the mouse. The selection of the MCU involves a trade-off between several criteria, such as memory, power, number and type of pins, and the ability to connect to the internet. For micromice, there are additional constraints to consider:

- The device is battery-powered, so it's essential to minimize overall power consumption.
- The robot incorporates numerous sensors, the data from which must be processed simultaneously.
- The robot must be capable of processing real-time data to ensure accurate position control.
- The robots must make decisions quickly, as the best run times for full-size micromice are currently around 8 seconds (winner of the all Japan classic micromouse contest 2017) [54].
- The MCU requires sufficient interrupt pins to allow quadrature encoding.
- The device needs enough memory to store the map of the maze (at least 1k of RAM, according to [46]).

2.8.1 Raspberry Pi

Raspberry Pis are single-board computers (SBCs) that can function as regular PCs. They are compact and inexpensive, yet offer a range of features such as a graphic driver. A mouse, keyboard, and screen can be connected to the device, and a Linux operating system can be installed on it [62]. Raspberry Pis can be programmed using various languages like Java, Python, or C++. Additionally, they incorporate WiFi and Bluetooth, enabling remote debugging of a device.

Advantages	Drawbacks
Fast: 700 MHz – 1.8 GHz processor	The processor is on a separate board and cannot be soldered directly onto the PCB (bulky)
Supports multiple programming languages	Requires proper shutdown before power off
Incorporates WiFi and Bluetooth	More costly than other microcontrollers
Can execute complex calculations and applications	Consumes more power than regular MCUs
Can run the Robot Operating System (ROS)	

Table 2.13: Advantages and drawbacks of Raspberry Pi, largely based on [62].

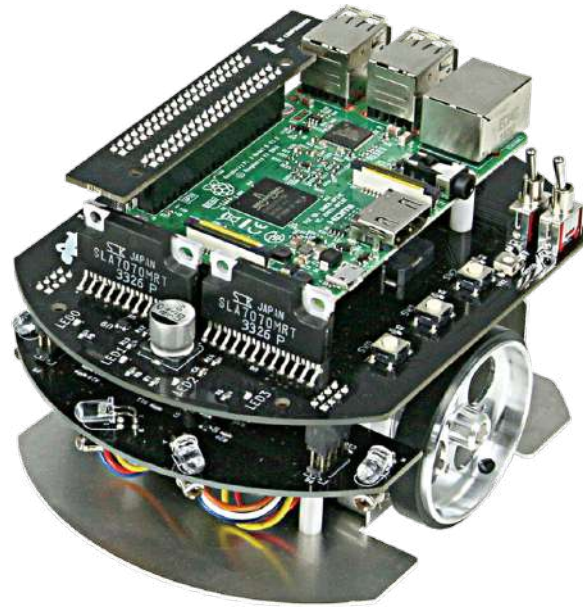


Figure 2.17: The Raspberry Pi Mouse v3 is based on standard Raspberry Pi boards [5]

2.8.2 Arduino

Arduino boards are widespread MCUs in the maker's community. They are inexpensive and easy to program through the dedicated Integrated Development Environment (IDE). These boards are ideal for quick prototypes and hacking projects. However, for more advanced projects, one can be limited by the IDE. Moreover, the devices do not integrate Bluetooth or WiFi. The microcontroller is a lot slower than on Raspberry Pi, with a common speed of 16 MHz [62]. From our analysis, the micromouse projects using Arduino are either beginner projects or ones meant for educational purposes.

Advantages	Drawbacks
Very inexpensive	Processor is on a separate board and cannot be soldered directly on the PCB (voluminous)
Easy to program through the IDE (in C or C++)	The IDE can be limiting
Low power	No WiFi or Bluetooth directly integrated

Table 2.14: Advantages and drawbacks of Arduino boards, widely based on [62].

2.8.3 STM32

STM32 microcontrollers appear to be the most widely used in intermediate and advanced micromouse projects (6 out of 10 projects studied). They are also equipped with ARM processors, like the Raspberry Pi boards. These microcontrollers have a faster clock than most Arduino boards, which means that they will perform the same calculations a lot quicker. Generally, the MCU will be directly soldered on the micromouse PCB, which allows for a significant gain of space. STM32 microcontrollers can also be programmed through the Arduino IDE, if desired [55]. This allows for a more powerful microcontroller, while still being able to program it very easily.

Advantages	Drawbacks
Powerful and cheap	Can be more difficult to program and understand, because it is more complex than Arduino
Faster clock than Arduino	
Component can be soldered directly on PCB (small)	

Table 2.15: Advantages and drawbacks of STM32 chips.

2.8.4 ESP32

ESP32 microcontrollers have emerged in the past few years and have become increasingly popular because of their versatility. The chips directly integrate WiFi and Bluetooth, which makes them ideal for IoT devices, or remotely controlled robots, for example. Moreover, ESP32 microcontrollers provide a lot more stack and flash than Arduinos, while still being compatible with the Arduino environment, which provides access to a multitude of libraries [64]. ESP32s have a large number of GPIO pins, and support multiple ADCs. All GPIO pins support interrupts, which is really useful in a micromouse project [28].

Advantages	Drawbacks
Powerful and cheap	Processor cannot be directly soldered on the micromouse PCB
Supported by the Arduino framework	
Integrates WiFi and Bluetooth, some modules have a removable antenna	
Dual core	

Table 2.16: Advantages and drawbacks of ESP32 chips.

2.9 Breakthroughs of Micromouse Contests

Throughout the years of the micromouse contests, some inventions stood out and spread among most of the competition winning robots. In the following sections, we present some of these revolutionary designs.

2.9.1 Handling Diagonals

Some sections of the micromouse mazes can be composed of multiple very short turns one after the other. A micromouse with a basic software would deal with these turns by moving forward, then stopping, then turning 90° , before finally re-accelerating for the next short segment. This approach is really inefficient, because the mouse never accelerates to its full speed. However, by building the mouse small enough, it can run down this part of the maze along a diagonal, hence hugely reducing the numbers of turns, as well as the total length of the path. This allows to solve the maze a lot faster. An example of a maze solution using diagonals is shown in Figure 2.18.

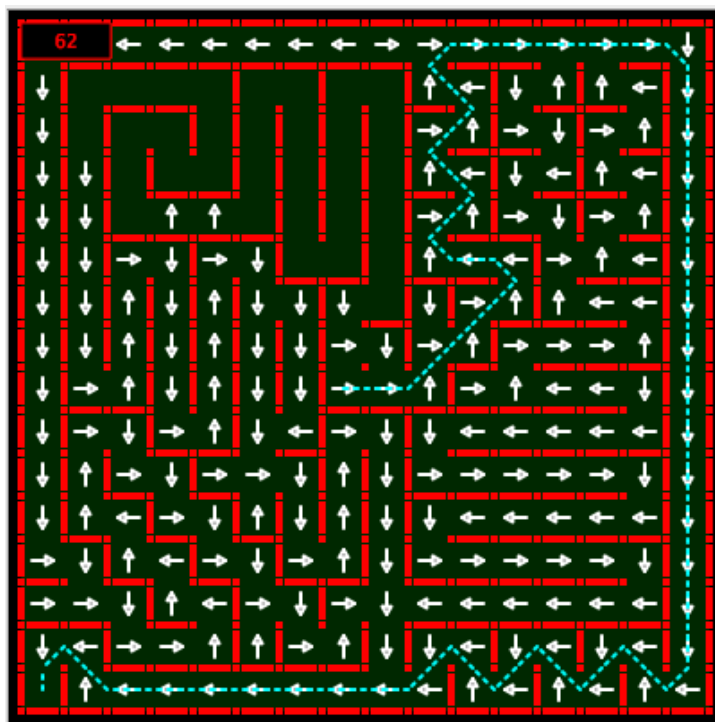


Figure 2.18: Example of a resolution of a maze using diagonals [41]

2.9.2 Double Pair of Wheels

Kato Yusuke, one of the winning micromouse builders in Japan, is the inventor of the four wheel design for micromice. He implemented it for the first time on his mouse Tetra in 2009 [53]. The idea is to have two pairs of wheels that are powered by one motor on each side of the mouse. The center of mass of the mouse must be relatively close to the front wheels. This design allows to turn in place nearly like a standard differential drive, but at the same time it allows to reach higher accelerations. Indeed, as the robot accelerates, the weight shifts to the back wheels, enhancing the friction with the floor, which provides a higher possible acceleration before slipping. Since its invention in 2009, this wheels configuration has widely spread among competition winning mice[42].



Figure 2.19: Decimus 4 by Peter Harrison has a 4 wheels design, allowing greater linear acceleration. Precise rotation can however be more difficult. [42]

2.9.3 Minimizing Turns

In 2017, the micromouse Red Comet was the first to take a path that was not the shortest one while still winning the competition. Indeed, to everyone's surprise, the micromouse did not take the path that minimized the distance, but the one minimizing the number of turns! The path was approximately 5 meters longer than the shortest one, but it could be run faster because it was composed of longer streaks and fewer turns, which allowed the robot to get to higher speeds [40]. This was a huge paradigm change, opening the doors to a multitude of new path finding algorithms.

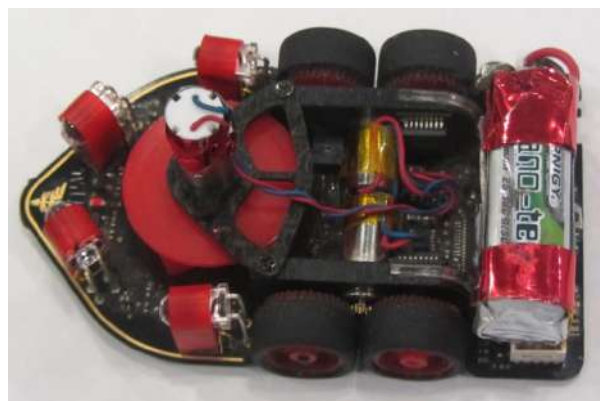


Figure 2.20: Red Comet is the first micromouse to have won a competition while not taking the shortest path from start to finish. [40]

2.9.4 Using Fans

Another paradigm change that impacted the history of micromouse is the use of fans to increase the grip of the mouse on the floor. Indeed, the fan adds a downwards force on the mouse without increasing its weight. One of the most known designs of micromouse using a fan is Sainen developed by Utsunomiya Masakazu in 2012, that has been ranking first in micromouse competitions in Japan. Nowadays, the best mice systematically include a fan [44]. The fans are so strong that the mouse could actually drive upside down.

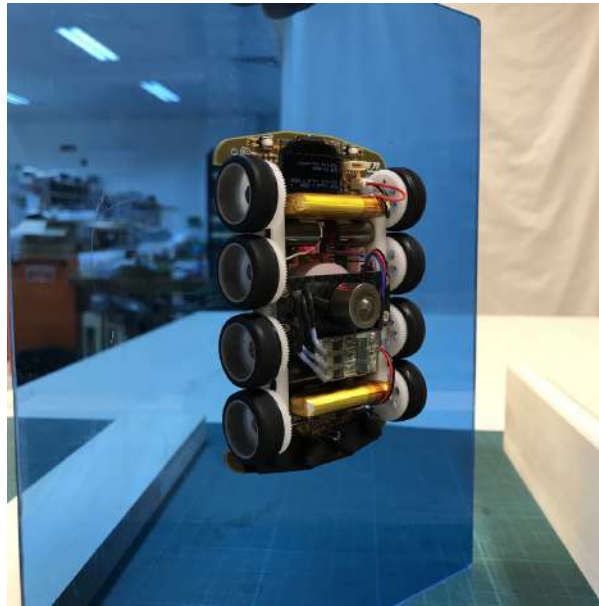


Figure 2.21: Excel 9a by Khiew Tzong Yong integrates a very powerful fan to increase the grip on the floor and therefore the maximal acceleration of the robot[66]

Chapter 3

Solutions

In this chapter, we present the primary design choices for the micromouse hardware and explain the reasons behind these decisions. The hardware specifications, however, will be introduced in Chapter 5. Throughout the state of the art, the major components of a micromouse robot have been identified. Based on our analysis of existing projects, we decided to add the following components to our robot:

- 1 ESP32-S3 micro-controller
- 2 BDC motors
- 1 dual motor driver
- 2 magnetic encoders
- 1 LiPo battery
- 1 buzzer
- 5 ToF distance sensors
- 1 accelerometer
- 1 push button
- 2 caster wheels
- 1 custom PCB

The main features that we did not include are as follows: multiple pairs of wheels, and an on-board fan to increase the friction between the robot and the floor. Indeed, these features are very advanced and would not be of any advantage in this relatively beginner’s robot.

3.1 Custom PCB

It is common to use a custom-designed PCB as the main structural element of micromice robots. This design eliminates the need for any sort of chassis, thereby reducing the overall weight. Moreover, the base of the PCB is a fiberglass and resin composite, which has high tensile strength and resistance to impacts, making it a suitable material to withstand the shocks that the micromouse might experience. We, therefore, decided to create our own PCB. The different components are then either hand-soldered or screwed onto the PCB.

3.2 Robot's Size and Shape

The micromouse contest rules limit the size of the robots to a maximum of 25 x 25 cm, without restrictions on the height. However, we want the robot to be small enough to navigate diagonally through the maze, which restricts the width to around 11 cm. Moreover, a smaller robot will likely be proportionally lighter, which allows for greater acceleration. We, therefore, aim to reduce the robot's weight overall, as well as placing the weight as low as possible (low center of mass) to increase stability.

Regarding the robot's shape, we decided to opt for an **octagonal design** (see Figure 3.1). This shape is interesting for several reasons:

- **It has multiple symmetries.** For instance, the back and front parts of the robot are symmetrical, which means that the robot will behave in similar ways when driving forwards and backwards. Symmetry also generally implies an easier design, since the same parts can be reused, for example, bumpers.
- **The back of the robot is flat.** This is really important for the start of a run in a maze. The robot can be pressed against the wall of the start cell, which ensures that it is well-oriented from the start.
- **It is a shape close to a circle.** This is practical because it means that there are no parts of the robot that stick out, which facilitates rotations on the spot in restricted spaces.

3.3 Wheels Layout

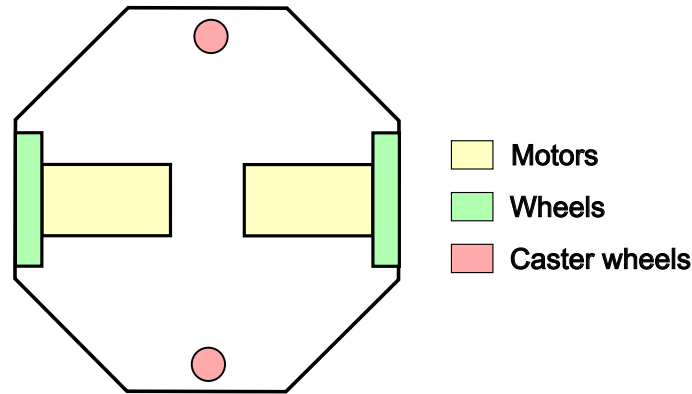


Figure 3.1: Schematic of the robot's wheels layout; the robot's dimensions are 11x11 cm

Most micromice use a differential drive design because the speed of the robot can be easily deduced from the motors' speeds. Additionally, this layout has the advantage of having a zero turning radius, meaning the robot can turn on the spot. This is a significant advantage in confined spaces such as a maze. In our design, we decided to use a standard differential drive with one motor on each side of the robot (see Figure 3.1). Due to our octagonal shape, the motors are in the middle of two opposite sides. We also decided to place the wheels directly on the motors without additional transmission. This limits the maximum length of the motors to approximately half of the robot's width (50 mm).

The length will therefore be a major criterion in the motors' choice. On the other hand, having direct transmission reduces the play and the complexity of the drive. Each motor is equipped with 32 mm wheels with silicone tires for improved adherence. To ensure the robot's stability, we integrated two additional caster wheels at the front and back of the robot. Indeed, since the motorized wheels are in the middle of the robot, the robot's weight will shift to the front or the back depending on the acceleration. This design also ensures a very stable robot, since the contact points with the floor are far apart.

3.4 Distance Sensors Layout

The number of distance sensors used in micromice varies between a minimum of 3 and up to 7 sensors. When only 3 sensors are used, two are pointed to the sides to measure the robot's distance to the walls, and one is pointed forward to detect obstacles and stop before them. Often, micromice also include two additional sensors oriented at 45° , which allow the mouse to see in advance whether the next cell has walls on the right or the left, hence enabling the mouse to react faster while navigating through the maze. Other designs also integrate a sensor oriented downwards, which allows the mouse to stop before falling if it reaches the edge of a table, for instance. Finally, some designs include sensors at the back of the robot, which allows it to avoid obstacles when driving backward.

In our case, we opted for the fairly standard 5 sensor design (see Figure 3.2). The robot has the three main sensors for the front and side walls, as well as two more sensors with 45° angles. These two additional sensors allow the detection in advance of whether the next maze cell has walls on the right and left sides. Having this information enables faster decision-making for the robot's next movement.

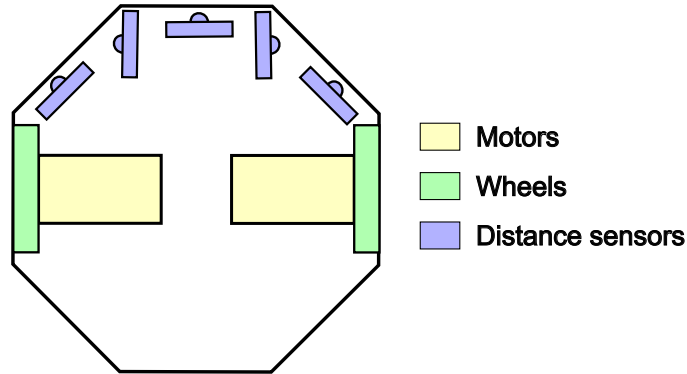


Figure 3.2: Schematic of the robot's distance sensors layout

3.5 Distance Sensors Technology

From the technologies for sensing distances that we presented in the state of the art (see Section 2.5), ultrasound sensors are only rarely used in micromouse projects because of their size. The two main options to chose from are IR sensors and ToF sensors. We have already stated that the most commonly used sensors are infrared sensors. Their main advantages are: they are very small, inexpensive and the readings are fast. The range of these sensors is however only a few tens of centimeters. Moreover, since they are reflective sensors, the relationship between the distance and the intensity of light detected is non-linear (proportional to $1/d^2$). Getting an actual measurement of the distance can therefore requires a good calibration procedure, as well as processing of analog signals. Obtaining a precise distance measurement is especially difficult if one wants the sensors to be robust to various ambient lighting conditions and target surface colors. On the other hand, ToF sensors are SMD chips, which have to be integrated on a carrier board, but some very small boards exist. What is more, the reading of the distance is straightforward, since they are compatible with I2C communication. An I2 request then directly returns the distance in millimeters, for a range to up to 4m in the case of the STMicroelectronics sensors.

In order to facilitate the software, we decided to use ToF sensors, despite the fact that they are pricier (about €10 for the ToF, vs €0.2 for an IR sensor). Since five sensors are used, this is a significant increase in the overall price of the robot, compared to the case where IR sensors were used. The total cost of the distance sensors covers around 1/3 of our target price, defined in section 2.1. This technology, however, ensures a good robustness in the distance readings regardless of the target color, the distance and the ambient light, which is a big advantage.

3.6 Inertial Measurement Unit (IMU)

The IMU (generally referred to as accelerometer + gyroscope) allows to know the robot's accelerations in each dimension, as well as it's angular speeds. This can be very useful to determine whether the wheels are slipping or not. Indeed, if the wheels are slipping on the floor, the odometry will continue to change, whereas the accelerometer would show that the robot is not actually moving. The accelerometer data can therefore be used to correct the odometry by using sensor fusion, the combination of data of multiple sensors in order to obtain a measurement with a smaller error [2]. Sensor fusion, however, is a relatively advanced technique. In this project, we decided to include an I2C IMU. Indeed, this addition to the PCB just requires to connect the sensor to the I2C bus. Moreover, the data provided by the sensor could be useful in the future to enhance the estimation of the robot's position, even though it will probably not be used it in the scope of this project.

3.7 Motors, Drivers, Encoders, and Wheels

When deciding on the type of motors to use for this project, two technologies stood out: DC motors and stepper motors. As presented in the state of the art, DC motors are the most widely used in micromouse projects. Depending on the quality of the motors, they can be quite inexpensive, very small, and come with a variety of reductions. The control of the motors can be achieved by applying a PWM signal to them, which varies the effective voltage value of the motors and, therefore, their speed. For precise control of speed and position, however, encoders are required. The encoder's signal must be processed to deduce the displacement of the motor shaft. With stepper motors, on the other hand, the control is very different. Electrical pulses are sent to the motor, which makes it move step by step (this can be seen as discrete control of the position). In this case, the displacement of the motor is accurately controlled by the pulses sent to it. Indeed, it is possible to control stepper motors to move an exact number of steps, resulting in a typical angular resolution of 1.8° . Therefore, there is no need for encoders, which simplifies control as it is open-loop. Moreover, stepper motors can be controlled at a very wide range of speeds and are good at turning at low speeds. If stepper motors have so many advantages, why aren't they used more in micromouse robots? According to Peter Harrison, steppers have multiple drawbacks as motors for micromouse robots [47]:

- It can be challenging to spin the motor very quickly (a high-frequency pulse train needs to be generated).
- It can be difficult to find a motor with high enough torque at the desired speed.
- Steppers are bulky, heavy, and consume a lot of power.

According to Harrison, steppers are a valid alternative to DC motors and are used in some micromice, but overall, DC motors have more advantages. For all these reasons, we decided to implement brushed DC motors in our mouse. Moreover, we had never used BDC motors with encoders before, and it seemed like a good opportunity to learn how they work.

We also have to add some encoders. It is possible to buy brushed DC motors that come directly assembled with a gearbox and a magnetic encoder for a very reasonable price. The price range for a motor can vary greatly depending on its quality and the manufacturer. Throughout the projects that we studied, one used Faulhaber motors, which cost around €200 per motor. According to the target price we set in section 2.1, this would be significantly above our budget. On the other hand, three projects used N20 micro gearmotors, which can be purchased for around €30 (including the encoder). These motors would therefore also be a suitable alternative for our robot.

Additionally, we cannot control the motors directly from the microcontroller because it cannot output enough power on the pins. To solve this issue, an H-bridge dual motor driver is added: the driver is connected to the battery and consists of multiple transistors that are driven by the microcontroller pins (see Figure 3.3). Finally, the wheels that are mounted on the motors are commercially available, 32 mm diameter wheels with silicone tires. They are very easy to mount on the motor's axles and have a perimeter of nearly exactly 10 cm, which makes estimations of speeds and displacements much easier.

3.8 Architecture

Now that the technologies being used have been defined, it is necessary to sketch out how the different components are connected together. This is illustrated in Figure 3.3, which depicts the architecture of our robot. The figure presents all the main electronic components, as well as how they interact with each other. Communication with the distance sensors and the IMU is facilitated using the I2C communication protocol. The encoders' data is processed with interrupts on GPIO pins of the microcontroller. The motor driver is controlled by PWM signals. The push button is also wired to a digital pin with an interrupt. The battery is connected to the driver, which powers the motors, and to the microcontroller, whose on-board voltage regulator generates the 3.3 V that powers all the sensors. Finally, the buzzer is controlled by a PWM signal on a GPIO pin of the microcontroller. The buzzer helps inform the user about the state of the robot, while the push button allows switching between control modes of the robot (stop, move on a straight line, avoid obstacles, etc).

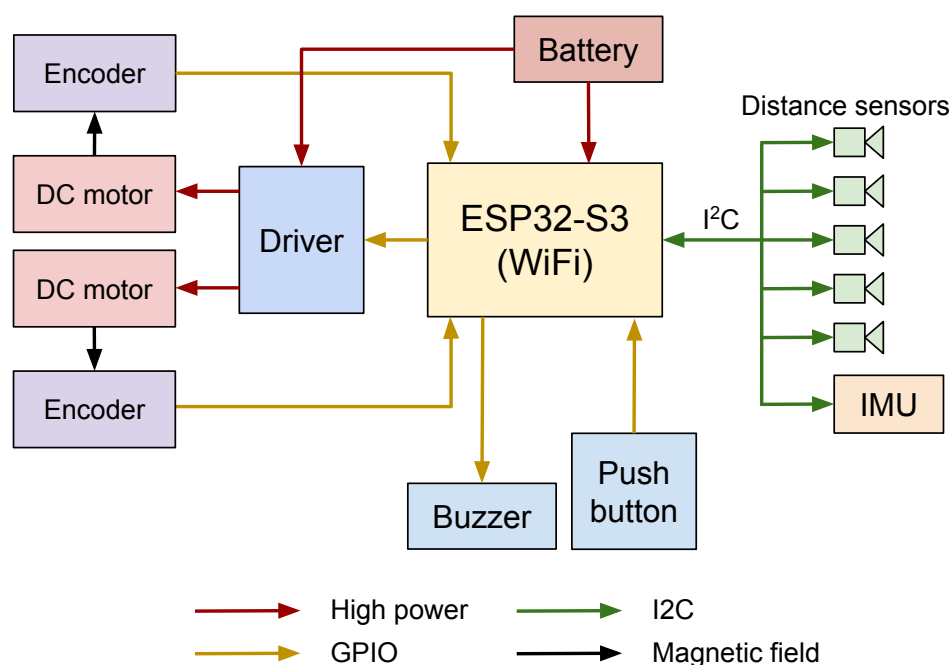


Figure 3.3: Schematic of the architecture of our robot. The red arrows indicate high power connections. The yellow arrows represent GPIO connections with the microcontroller. Green arrows are for I2C.

Chapter 4

Theory

4.1 Robot's Dimensions

As stated in chapter 3, we aspire to have a robot small enough to run diagonally through the maze. The maze consists of square cells with a side $a = 18$ cm (see Figure 4.1). Based on this measurement and some fundamental geometry, we can deduce the width d of the maze's diagonal:

$$d = \frac{a}{\sqrt{2}}$$

Since our robot needs to be smaller than this width, we consider a margin of $m = 1$ cm on each side of the robot. Hence, we get the following equation for the width w of the robot:

$$w = \frac{a}{\sqrt{2}} - 2m$$

This results in a width of approximately 10.7 cm. In practice, we rounded this width to 11 cm.

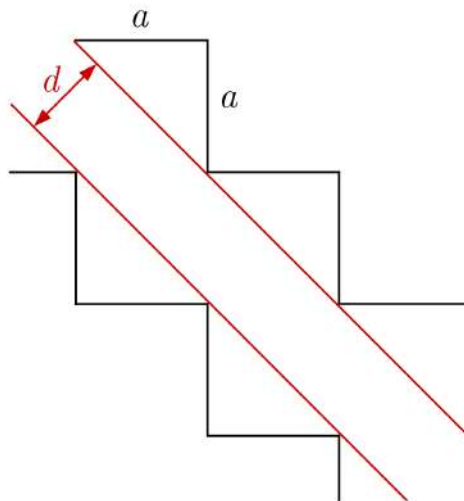


Figure 4.1: Sketch for the computation of the diagonal width of the maze.

4.2 Brushed DC Motor Fundamental Equations

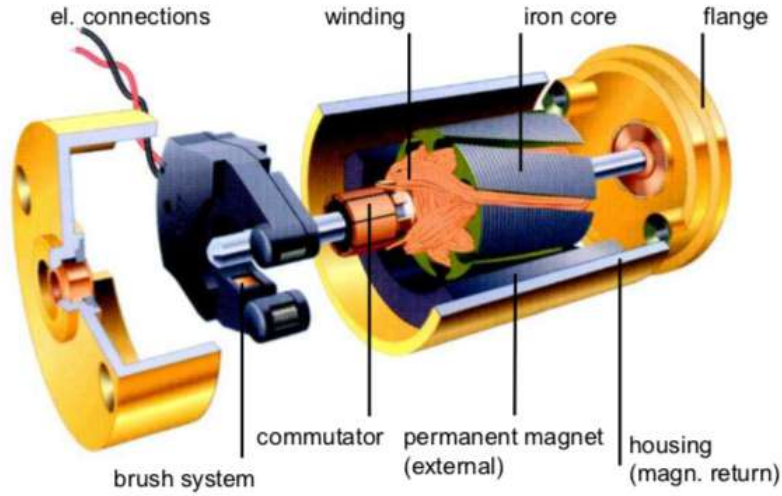


Figure 4.2: Schematic of a brushed DC motor [9]

Brushed DC motors are constructed with a stator composed of an outer permanent magnet and a rotor integrating the winding as well as an iron core, as shown in Figure 4.2. To understand the physics behind this type of motor, we can first consider what happens to a single loop of conductor inside a magnetic field. This layout is sketched in Figure 4.3. According to Lorentz' law, a charge q with a given speed \vec{v} will experience a force \vec{F} when immersed in a magnetic field \vec{B} . The analytical expression of the Lorentz force is as follows:

$$\vec{F} = q\vec{v} \wedge \vec{B}$$

However, in our case, we want to express the force as a function of the current I applied to the loop of wire. By modifying the previous equation, we get

$$d\vec{F} = Id\vec{l} \wedge \vec{B}$$

Where $d\vec{l}$ is a small section of wire, and $d\vec{F}$ the force applied to this small section. To obtain the force on the whole loop, one should integrate this expression over the entire length of the conductor. This force F is represented in Figure 4.3. In this figure, we can see that the loop of current is aligned with the magnetic field, and that the force is both perpendicular to the current and the magnetic field (due to the vector product). This is the configuration where the force applied on the loop is maximal. Since the loop of conductor can rotate freely, it will start to turn due to the Lorentz force. Once in motion, the commutator will reverse the direction of the current in the loop every half turn, which will make the loop spin continuously thanks to its inertia. This is the basic principle of how brushed DC motors work.

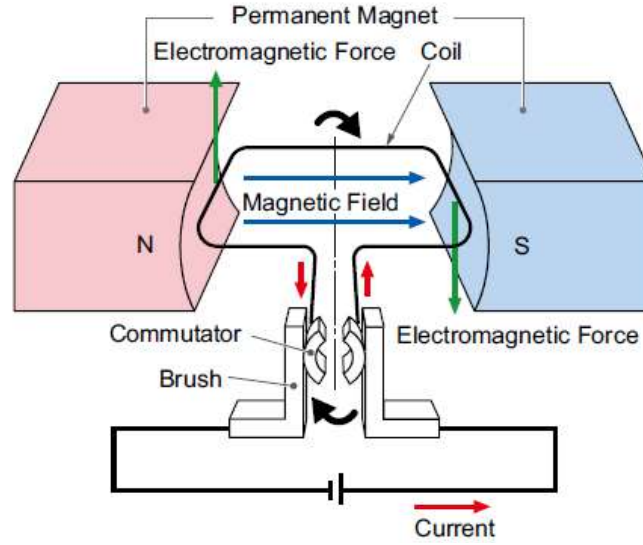


Figure 4.3: According to Lorentz' law, a loop of wire supplied with a current will perceive a force when a magnetic field is applied to it [24]

Next, we will express the basic equations governing BDC motors. To do that, we first need to find the electrical model of the BDC motor's rotor. The rotor is basically a winding with multiple coils of wire, which can be represented as an inductor and a resistor in series. There is one more component to add to this circuit: a voltage source, that will model the induced voltage $u_i(t)$ in the winding. This induced voltage is a consequence of Lenz' law, which states that a variation of magnetic flux generates an electromagnetic force that opposes the said variation. Figure 4.4 shows the electric model of a BDC motor. Based on this model, we can express Ohm's law applied to the motor:

$$u_a(t) = R_a i_a(t) + L_a \frac{di_a(t)}{dt} + u_i(t) \quad (4.1)$$

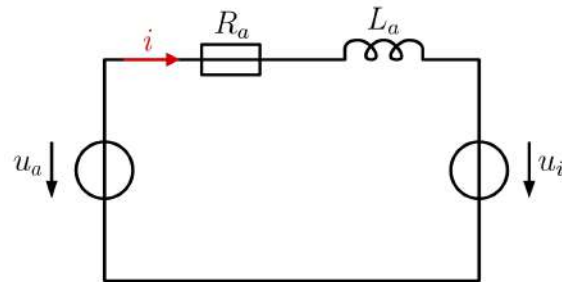


Figure 4.4: Electric model of a BDC motor

Furthermore, DC motors possess two intriguing linear characteristics. First, the induced voltage u_i in the winding is directly proportional to the motor's speed ω . The constant connecting the two is the speed constant k_u .

$$u_i(t) = k_u \omega(t) \quad (4.2)$$

Secondly, the motor's torque T_{em} is directly proportional to the current i_a applied to it, and both are connected by the torque constant k_T .

$$T_{em}(t) = k_T i_a(t) \quad (4.3)$$

This suggests that to achieve a specific output torque, we need only control the current applied to the motor. Conversely, Equation 4.2 is extremely beneficial because it indicates that by measuring the induced voltage on the motors, we can obtain an accurate measurement of the motor's speed. This is vital for effective regulation of the motor's speed. Lastly, BDC motors have an additional advantageous property, which is that

$$k_T = k_u$$

4.3 Motor Dimensioning

The selection of the motor directly impacts the maximum speed the robot can achieve, as well as its acceleration. In this section, we will analytically compute the maximum speed and acceleration of our robot based on our final motor selection, as well as the robot's properties, such as weight, wheel diameter, etc. There will be multiple approximations throughout the calculations, because many variables are not quantitatively known. For example, it is very difficult to estimate the efficiency of the gearbox, because we use inexpensive motors, and the datasheet does not contain this information. The friction coefficient of the wheels on the ground can also only be roughly approximated.

In order to choose a commercially available motor suitable for our application in terms of power, we first have to establish the following properties:

- n_0 : Ideal no-load speed
- T_{max} : Maximum required torque

Other parameters will also further limit our choice: the motor's dimensions, its nominal voltage, etc.

4.3.1 Required Speed and Acceleration

In 2011, the fastest micromice could reach up to 3.3 m/s, solving a standard maze in about 4 s [32]. According to Peter Harrison, a finalist of multiple micro-mouse competitions, micromice should maintain an average speed of around 2.3 m/s to be competitive (as of 2009) [45]. Similarly, Harrison states that a maximum acceleration of 5 m/s² (around $0.5 \cdot g$) is sufficient for most micromice [39]. Indeed, this acceleration will mainly be limited by the friction coefficient of the tires on the floor and the weight of the mouse, in the absence of a suction system.

We therefore take the following theoretical values for the calculations:

- $a_{max,th} = 5 \text{ m/s}^2$
- $v_{max,th} = 1.5 \text{ m/s}$

4.3.2 Variables

We begin by defining all the variables used in further calculations. Firstly, Table 4.1 shows all of the variables with a known numerical value. Secondly, Table 4.2 summarizes all the important properties of our chosen motor. Finally, the speed of the motor depending on the torque is displayed in ??.

Symbol	Value	Unit	Definition
g	9.81	m/s ²	Earth acceleration
$a_{max,th}$	$0.5 \cdot g$	m/s ²	Max. desired linear acceleration
$v_{max,th}$	1.5	m/s	Max. desired linear speed
m_{wheel}	3	g	Mass of the wheels of the motors
r	16	mm	Radius of the wheels
M	150	g	Total mass of the robot

Table 4.1: All variables which value is known

Symbol	Value	Unit	Definition
I_0	0.07	A	Motor's no-load current
n_0	1100	rpm	Motor's no-load speed
T_s	38	mNm	Motor's stall torque
n_{nom}	840	rpm	Motor's nominal speed
T_{nom}	9.8	mNm	Motor's nominal torque
P_{max}	1.1	W	Maximum motor power

Table 4.2: Properties of the gearmotors used in our robot, the Polulu HPCB 12V 30:1 micromotor [27]

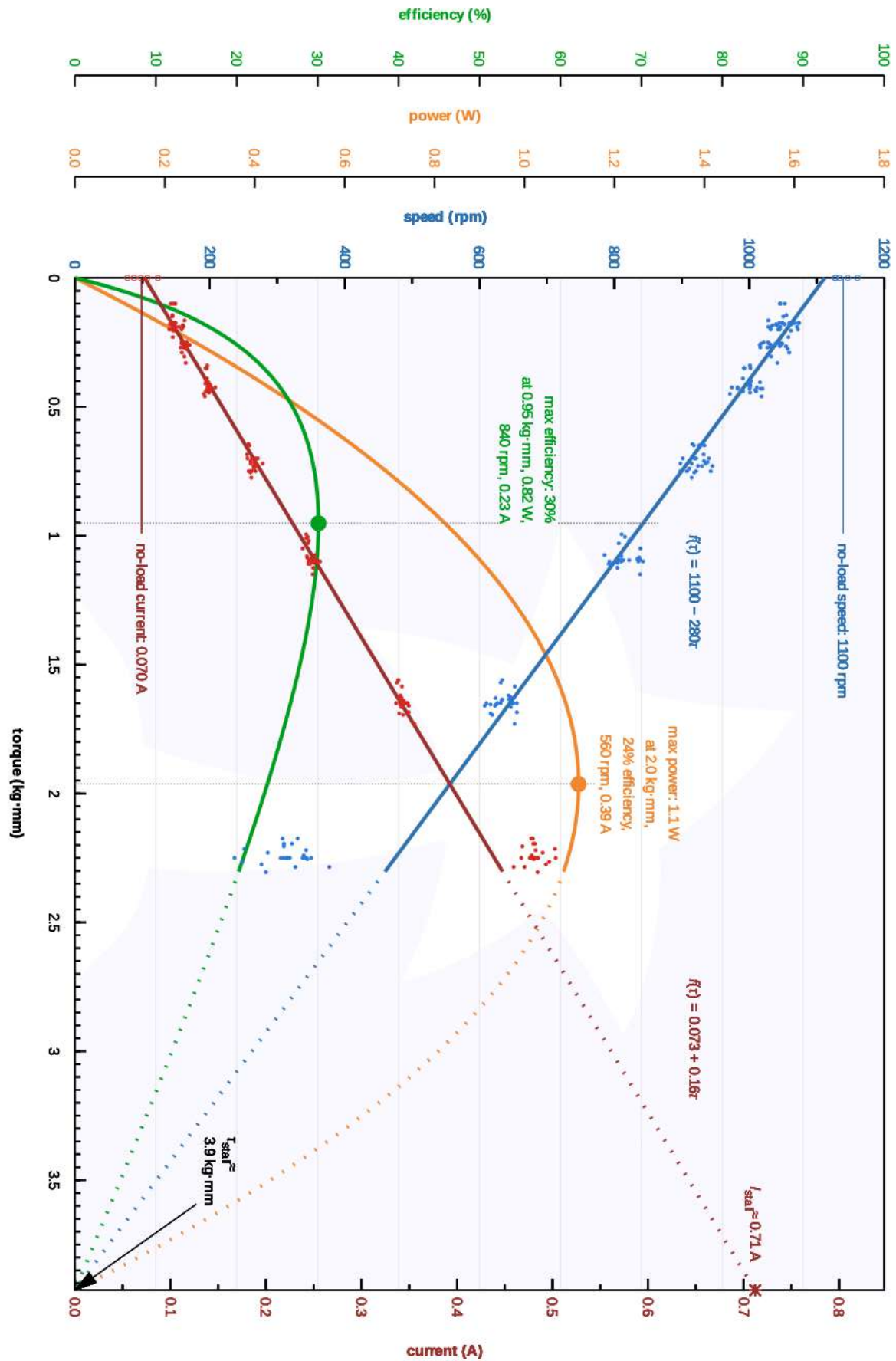


Figure 4.5: Speed, current and efficiency of the gearmotor depending on the torque, for a supply voltage of 12 V. Note that $1 \text{ kg} \cdot \text{mm} \approx 9.8 \text{ N} \cdot \text{mm}$.

4.3.3 Motor's Maximal Theoretical Speed and Acceleration

The maximum speed of the motors can be easily computed from the desired maximum linear speed of the robot and the perimeter of the wheels. In our case, the wheels have a diameter of $d = 32$ mm. We can compute the perimeter p with $p = \pi \cdot d$, which in our case is conveniently around 10 cm. The ideal maximum speed of the gearmotor $n_{max,th}$ in [rpm] is then simply

$$n_{max,th} = \frac{v_{max,th}}{60 \cdot p}$$

Numerically, we get a speed of 895 rpm, or expressed as an angular speed

$$\omega_{max,th} = \frac{v_{max}}{r}$$

Which corresponds to 94 rad/s. It should be noted that this is the speed that the motor should have when the mouse moves at its maximum speed. In our case, we have chosen a motor with a no-load speed of 1100 rpm, this means that the loaded motor will be slower than this value (especially since we power it at a lower voltage than the nominal one). In the end, our motors might be a little undersized to reach the desired performance of 895 rpm, but we consider it sufficient for this project.

Similarly, the maximum desired angular acceleration $\alpha_{max,th}$ is

$$\alpha_{max,th} = \frac{a_{max}}{r}$$

Which yields approximately 307 rad/s².

4.3.4 Maximal Theoretical Motor Torque

To compute the torque of the motor, we will use Newton's second law for rotation, which states that the sum of the torque equals the inertia times the angular acceleration.

$$\sum \vec{T}_O = J_{tot,O} \cdot \vec{\alpha} \quad (4.4)$$

We therefore first have to compute the total inertia "seen" by each of the motors of the active wheels. Indeed, in our design, we have two active wheels equipped with motors, and two passive wheels, that are just caster wheels. In these calculations, we will consider the case of the gearmotor as a whole, and do not include the yield of the gearbox in the calculation. There are hence three different contributions to the inertia of each gearmotor:

- J_{mot} : Inertia of the gearbox and motor shaft
- J_{wheel} : Wheel's inertia
- J_M : Inertia of the robot's mass

The inertia of the motor is generally provided in the motor's datasheet. However, this information is unavailable for our motors, so we estimated it based on other similar motors. We use $J_{mot} = 1 \cdot 10^{-7}$ [kg·m²]. For the wheel's inertia, we approximate it as a full cylinder, using the wheel dimensions and mass:

$$J_{wheel} = \frac{1}{2}m_{wheel}r^2$$

Finally, the inertia of the robot mass is evenly distributed on the two wheels and is therefore given by:

$$J_M = \frac{1}{2}Mr^2$$

Since we compute the torque directly at the output of the gearbox, we do not need to consider the gear ratio or the gearbox yield in our calculations. The total inertia is then simply

$$J_{tot} = J_{mot} + J_{wheel} + J_M$$

This leads to a numerical value of $J_{tot} = 2.2 \cdot 10^5 \text{ kg} \cdot \text{m}^2$. Next, we have to enumerate all torques on the gearbox axis. The main torque is that of the motor T_{mot} . Finally, we consider a load torque T_{load} in the calculations. However, as long as the robot is moving on a flat surface, this load torque will be null. One final torque must be considered: the one caused by the friction on the ground and the rolling resistance of the wheels. The rolling resistance torque depends on the material of the wheels and the pressure between the two surfaces in contact. As the robot is relatively light, we neglect its contribution in the numerical calculation. Indeed, it would also be really difficult to estimate. On the other hand, the friction torque T_{fr} is due to the inner friction inside the motor and the gearbox. It is typically a viscous friction, which implies that it is proportional to the motor's speed:

$$T_{fr} = C\omega \quad (4.5)$$

At this stage, we only work in one dimension and can therefore use scalars instead of vectors. We hence obtain

$$\sum T_O = T_{mot} + T_{fr}$$

By replacing the total torque in Equation 4.4, we can finally isolate the motor's torque

$$T_{mot} = J_{tot} \cdot \alpha - T_{fr} - T_{load}$$

For the maximal acceleration that was calculated in the previous section $\alpha_{max,th} = 307 \text{ rad/s}^2$, we get a desired motor torque of $M_{mot} = 6.7 \text{ mNm}$ to ensure the acceleration of $0.5 \cdot g \text{ m/s}^2$. As a reminder, this is an approximation in which all friction is ignored. By referring to Figure 4.5, we can see that for a torque of approximately 7 mNm , the motor's maximal speed is around 900 rpm (when the motor is powered at 12 V). This implies that the motor can have the desired acceleration over the range from 0 to 900 rpm without saturating. Since our input voltage is lower than 12 V , however, this maximal speed for a given torque will be lowered. It is therefore possible that the motor will not manage to provide the torque required for a given acceleration until the desired theoretical speed of 1.5 m/s is reached.

4.3.5 Friction Coefficient of the Wheels

In order to compute the static friction coefficient of the wheels on the floor, a special setup was designed. A pendulum is attached to a rigid board at a distance L from its border. The robot is placed on the board, which is then tilted slowly, increasing the angle θ between the board and the table. A ruler is used on the table to measure the distance Δx from the point where the board touches the table to the pendulum. This ensures that we are working with a right-angle triangle, represented in Figure 4.6.

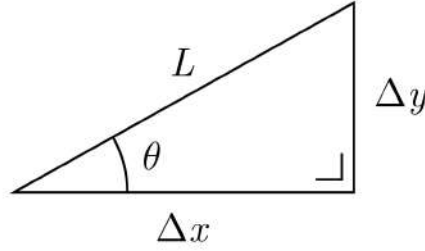


Figure 4.6: Right angle triangle used to compute the friction coefficient μ_0

The friction coefficient is then simply

$$\mu_0 = \tan(\theta)$$

However, since we did not measure the angle directly, it is easier to express μ_0 in terms of L and Δx :

$$\mu_0 = \sqrt{\left(\frac{L}{\Delta x}\right)^2 - 1}$$

To obtain a more reliable result, we took five measurements for Δx and used the average in the calculation. We obtained a static friction coefficient of $\mu_0 = 1.05 [-]$, which is excellent. The board we used was covered with a vinyl-type plastic sheet, so this measurement is for silicone on plastic.

4.3.6 Maximal Acceleration Before Slipping

In micromouse competitions, the friction between the tires and the floor is crucial, as it is the factor that will limit the robot's acceleration. In this section, we aim to express the maximal acceleration of the robot α_{max} depending on the static friction coefficient μ_0 between the wheels and the floor. First, we need to find the maximal force between the floor and the wheel before slipping. The maximal friction force $F_{fr,max}$ is simply

$$F_{fr,max} = \mu_0 \cdot N$$

where N is the force the wheel applies perpendicularly on the floor. In the case where the robot runs on a flat surface, this is simply the gravity force divided by the number of contact points with the ground (in our case 4, because we have 4 wheels).

$$N = \frac{Mg}{4}$$

Then, to compute the torque on the wheel's axis, we multiply the force by the wheel's radius

$$T_{mot,max} = F_{fr,max} \cdot r$$

Which, by substituting, leads to

$$T_{mot,max} = \frac{Mgr\mu_0}{4}$$

The resulting maximal acceleration of the motor before slipping is then simply

$$\alpha_{max} = \frac{T_{mot,max}}{J_{tot}}$$

By plotting the maximal acceleration for various friction coefficients, as we have done in Figure 4.7, we can clearly visualize the importance of a good friction coefficient between the wheels and the floor. In our case, we will have silicone tires, and we could roughly estimate $\mu_0 = 0.8 [-]$, which would limit our acceleration to around 240 rad/s^2 . This is significantly less than our desired 307 rad/s^2 . There is no other way to increase this theoretical maximal acceleration than either increasing the friction coefficient, or the robot's mass (but that would actually be counter-productive).

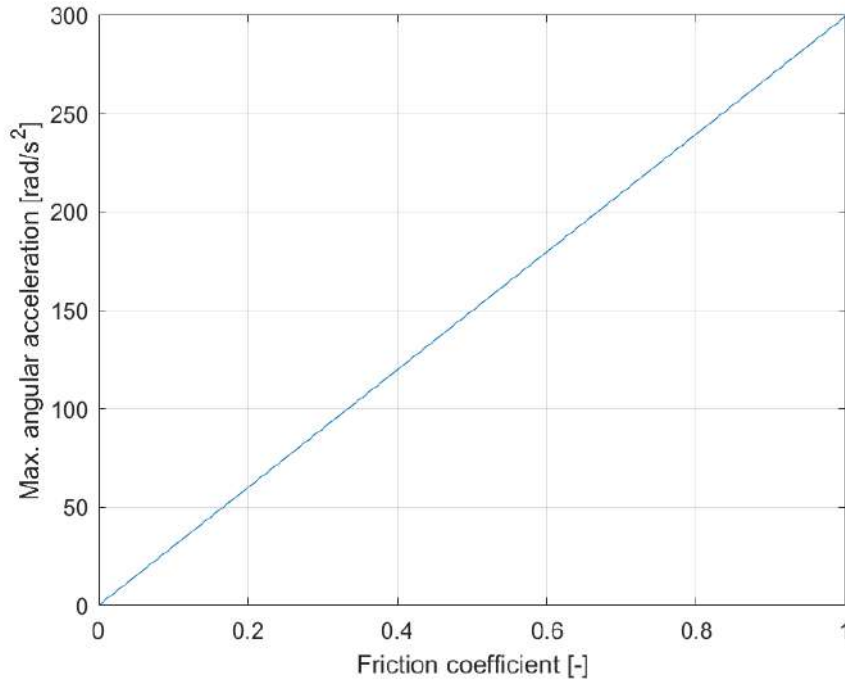


Figure 4.7: Maximum acceleration of the motors before slipping for different friction coefficients of the wheels on the floor.

4.4 Differential Drive Kinematics

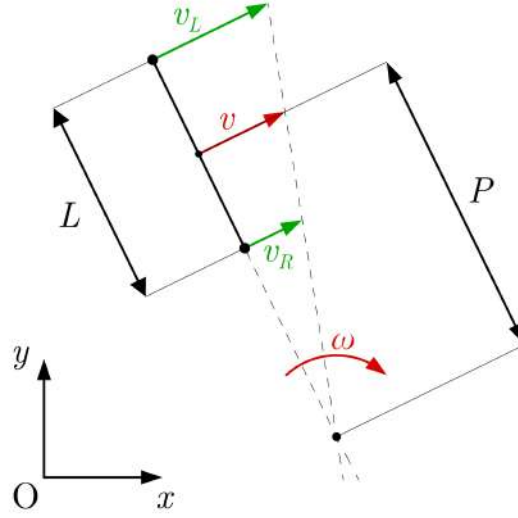


Figure 4.8: Schematic of a differential drive robot's kinematics

In this section, we want to find the relationship between the wheel's angular speed (Ω_L , Ω_R) and the robot's speed (v and ω). To begin, we have to set a few basic assumptions:

1. The robot is symmetric along its longitudinal axis.
2. The robot's chassis is a rigid body.
3. The wheels do not slip or skid.

Based on the hypothesis that the wheels do not slip, we can deduce that the angular speed Ω of the wheels is directly proportional to their linear speed:

$$v_L = \Omega_L r \quad \text{and} \quad v_R = \Omega_R r$$

Figure 4.8 shows an illustration of the problem. In this figure, we can see that the linear speed of the robot is just the average of the linear speeds of the wheels. This is only true because of our first hypothesis: since the robot is symmetrical, the center of the robot is the midpoint between the two wheels.

$$v = \frac{v_L + v_R}{2} \tag{4.6}$$

On the other hand, the angular speed of the robot is the rate at which it rotates around its instantaneous center of curvature (ICC). This rate is determined by the difference between the wheel speeds and the distance L between the two wheels:

$$\omega = \frac{v_R - v_L}{L}$$

Equation 4.4 implies that clockwise rotation of the robot corresponds to a negative speed and counter-clockwise rotation to a positive speed. By substituting the linear wheel speed by the angular wheel speed and writing the equations in a matrix form, we get the forward kinematics of a differential drive robot:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = r \begin{pmatrix} 1/2 & 1/2 \\ 1/L & -1/L \end{pmatrix} \begin{pmatrix} \Omega_R \\ \Omega_L \end{pmatrix}$$

To find the inverse kinematics, we simply isolate the angular wheel speeds in the previous equation which, thanks to a little linear algebra, leads to

$$\begin{pmatrix} \Omega_R \\ \Omega_L \end{pmatrix} = \frac{1}{r} \begin{pmatrix} 1 & L/2 \\ 1 & -L/2 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}$$

This equation gives us the wheel's angular speed for any desired speed of the robot. Another useful equation is the relationship between the wheel speed and the curvature radius ρ . Indeed, if we want the robot to turn around an obstacle, we need to turn with a known radius. The curvature radius can be found easily by using similar triangles, which leads us to the following relationship (see Figure 4.9):

$$\frac{|v_R - v_L|}{L} = \frac{v}{\rho}$$

By substituting the robot's speed v using Equation 4.6, we can isolate ρ :

$$\rho = L \frac{v_L + v_R}{2|v_R - v_L|}$$

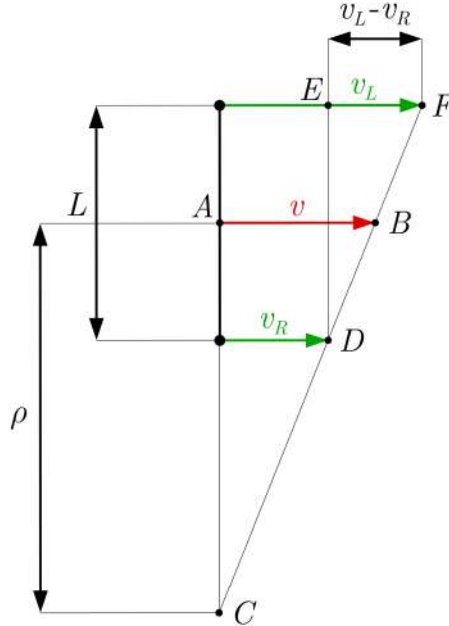


Figure 4.9: Using the $\triangle ABC$ and $\triangle DEF$ similar triangles allows us to find the curvature radius ρ

4.5 Rotary Encoders

One issue with encoders is determining in which direction the motor is rotating. To solve that, two sensors with a 90° phase difference are used simultaneously. For instance, our encoders are built with 6 magnets, and two hall effect sensors that are 90° apart, as illustrated in Figure 4.10. Depending on which of the sensor is triggered before the other, the direction can be determined. This is called quadrature encoding [21]. The output signals of the two sensors can be seen in Figure 4.11. Typically, the transitions from one state to another are detected using interrupts. Since they are two data channels per encoder and we have two encoders, our micro-controller requires at least 4 interrupt pins.

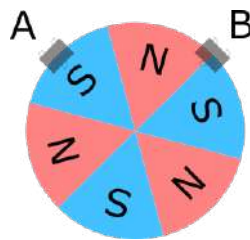


Figure 4.10: Schematic of the Polulu magnetic encoder: the spinning disk is composed of 6 magnets with alternating polarities, and two hall effect sensors [13]

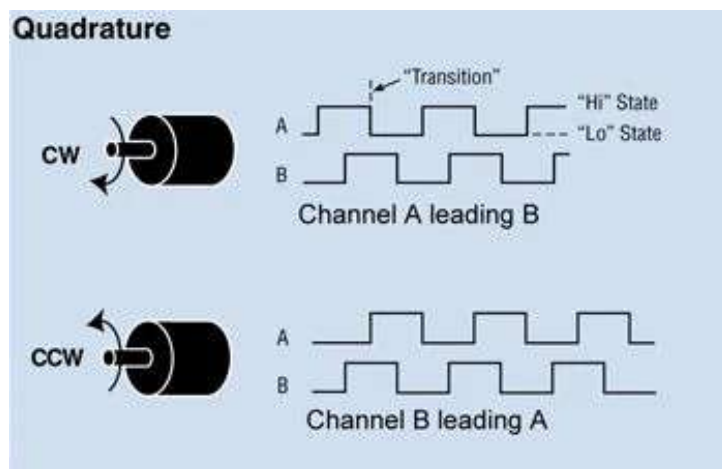


Figure 4.11: Principle of quadrature encoding: depending on the direction in which the motor spins, either channel A or channel B is leading [21]

There are three different ways to count the pulses of the encoders, which will result in different resolutions. They are called X1, X2 and X4 encoding [56]:

- X1 encoding: Only increment/decrement the counter every time there is a rising edge on channel A.
- X2 encoding: Increment/decrement the counter every time there is a rising and a falling edge on channel A. This results in twice as many counts per revolution.
- X4 encoding: Increment/decrement every time there is a rising or falling edge on either channel A or channel B. This results in four times as many counts per revolution as in X1 encoding.

Every time an interrupt event is detected, the counter of the encoder is incremented or decremented depending on the state of the other channel. These three types of encoding are illustrated in Figure 4.12.

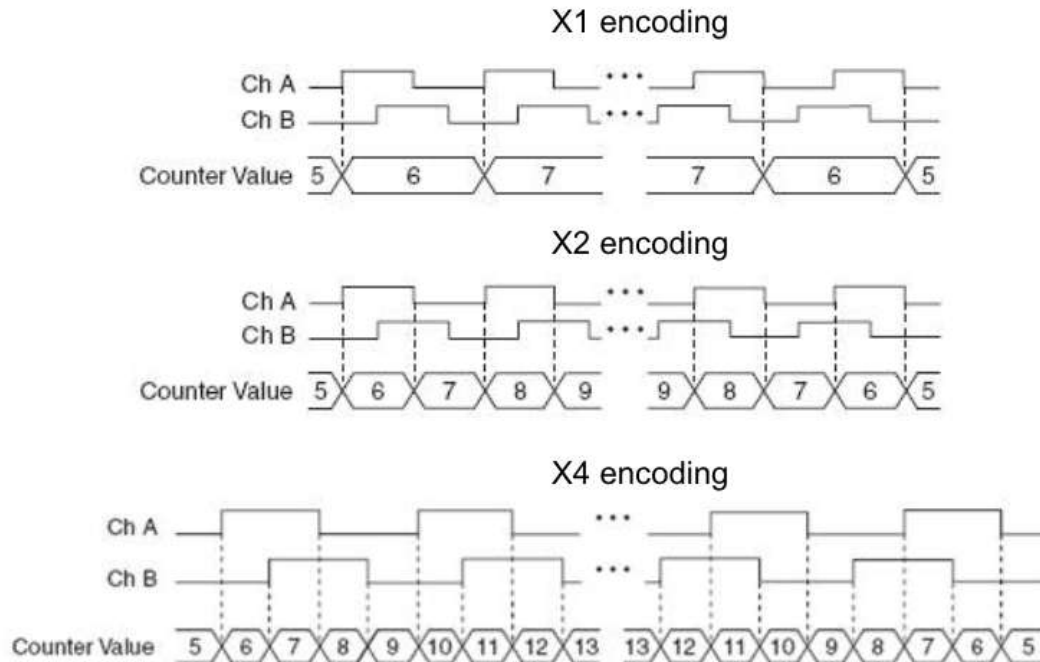


Figure 4.12: Signals on channels A and B for each type of encoding, as well as the corresponding counter value [56]

4.6 Wheels' Speed Measurement

Measuring the speed of our robot precisely is crucial for a good regulation of the robot's speed. The goal is to have a high update of the speed measurement as well as a good accuracy and resolution. In our case, the speed of the wheels is deduced from the encoders data. Indeed the basic principle is just to divide a measured number of counts by the corresponding elapsed time. They are however two distinct ways of achieving that, that will result in different resolutions depending on the speed.

The first method is the most intuitive one: we call a function at a given frequency. Every time the function is called, we count the number of pulses Δc since the last call and divide it by the elapsed time T :

$$v_{high} = \frac{\Delta c}{T}$$

The problem with this approach is that the measurement will have a really bad resolution for low speed. A better approach for low speeds is actually to use the interrupts on the encoders as well as a timer. We define a fixed number of counts C and use a timer to measure precisely the time elapsed Δt for the desired movement to be done. For instance, on the ESP32, the time resolution of the timer is of one microsecond, which will lead to a good final measurement of the speed.

$$v_{low} = \frac{C}{\Delta t}$$

One issue with this method is that often the number of counts that a timer can measure is limited, and there will be an overflow if the duration measured is too long. The number of counts C should therefore be chosen accordingly, so that the counts are reached before a counter overflow. Moreover, if we only use interrupts, we will never know if the speed is null, since the interrupt service routine (ISR) will never be called. This case has to be handled separately.

4.7 Odometry

Odometry is a crucial part of mobile robots navigation: indeed it is one of the ways the robot can determine its position in space and its speed. Formally, odometry can be defined as follows:

"A method to do position estimation for a wheeled vehicle during navigation by counting the number of revolutions taken by the wheels that are in contact with the ground." [68]

This implies that the robot needs to integrate a sensor that will measure the displacement of the wheels that are touching the ground. These are typically some sort of encoders, that generate pulses as the wheels are rotated. These pulses are counted, and the counter value is directly proportional to the distance traveled by each wheel. It is important to note that since we are counting pulses, we are dealing with discrete values, which will impact the precision of all the values that are deduced from the encoder counts.

Symbol	Value	Unit	Definition
N	12	CPR	Encoder counts per revolution
i	30	[-]	Gearbox reduction
L	103	mm	Distance between the wheels contact point with the ground
r	16	mm	Radius of the wheels

Table 4.3: All variables which value is known

Firstly, we want to compute the distance dx traveled by the wheels for each encoder count:

$$\Delta x_{step} = \frac{2\pi r}{iN}$$

In our case this is around 0.3 mm, which is relatively good considering the quality of our motors and encoders. On the other hand, it is useful to know what rotation of the wheel $d\theta$ one step of the encoders corresponds to. This is simply

$$\Delta\theta_{step} = \frac{360}{Ni} \quad [^\circ]$$

In our case, we conveniently get $\Delta\theta = 1^\circ$. Next, we want to find how to compute the current robot's position.

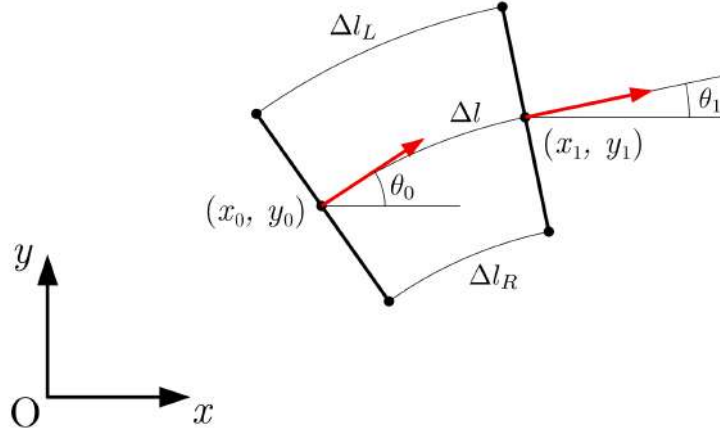


Figure 4.13: General sketch of the odometry problem: the red arrows indicate the orientation of the robot.

The robot position is defined thanks to three variables: the coordinates x and y of the robot's center, as well as the robot's orientation θ (see Figure 4.13). These variables are expressed with regards to a fixed cartesian frame of reference. Moreover, the odometry algorithm will be called every fixed delay dt . We will use the indices 0 for the previous values, and the indices 1 for the current values. At each call of the function, we first calculate the number of counts Δc of each encoder wheel since the last call. We then convert these counts into a distance traveled by each wheel:

$$\Delta l_L = \Delta c_L \cdot \Delta x_{step} \quad \text{and} \quad \Delta l_R = \Delta c_R \cdot \Delta x_{step}$$

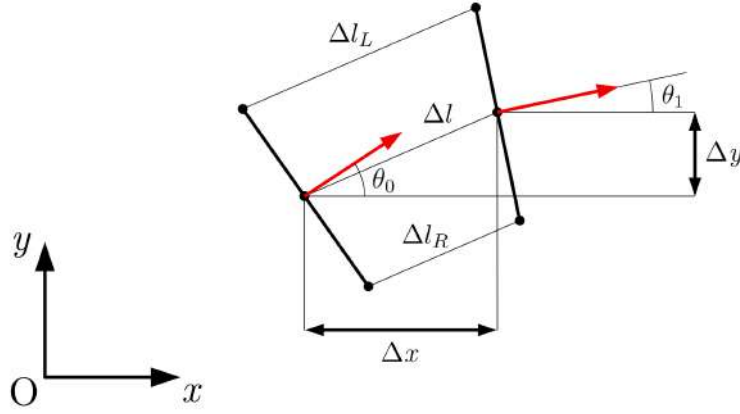


Figure 4.14: Similar triangles allow to compute the linear displacement of the robot Δl based on the distance traveled by each wheel (Δl_L and Δl_R). The red arrows indicate the orientation of the robot.

We can now compute the distance Δl traveled by the center of the robot. Figure 4.14 shows two similar triangles: $\triangle ABD$ and $\triangle ECD$. Based on the properties of similar triangles, we deduce the following equation:

$$\frac{\overline{AB}}{\overline{DB}} = \frac{\overline{EC}}{\overline{CD}}$$

Which leads to

$$\frac{\Delta l_R - \Delta l_L}{L} = \frac{\Delta l - \Delta l_R}{L/2}$$

By isolating the distance Δl traveled by the center of the robot in this equation, we find

$$\Delta l = \frac{\Delta l_L + \Delta l_R}{2}$$

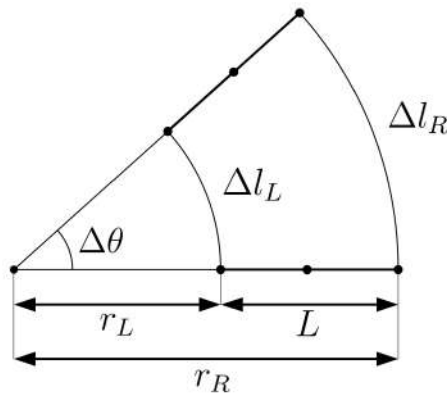


Figure 4.15: Schematic for the computation of the variation of the robot's angle

On the other hand, the variation in the orientation $\Delta\theta$ can be found by using arcs, this time, as shown in Figure 4.15. Again, we imagine that the robot moves on a circular trajectory between two calls of the odometry task. By using basic geometry knowledge,

we know that the length of an arc is given by the product of the angle and the radius. This leads to the two following equations

$$\Delta l_R = \Delta\theta \cdot r_R$$

$$\Delta l_L = \Delta\theta \cdot r_L$$

By subtracting these two equations, we get

$$\Delta l_R - \Delta l_L = \Delta\theta \cdot (r_R - r_L)$$

However, the distance between the two radii is simply the distance L .

$$\Delta l_R - \Delta l_L = \Delta\theta \cdot L$$

At this stage, the angle variation can be isolated

$$\Delta\theta = \frac{\Delta l_R - \Delta l_L}{L}$$

The new orientation can then be deduced:

$$\theta_1 = \theta_0 + \Delta\theta$$

This orientation is then clamped in order to remain in the range $[-\pi, \pi]$. Finally, the new coordinates of the center can be calculated:

$$x_1 = x_0 + \Delta l \cdot \cos(\theta_1)$$

$$y_1 = y_0 + \Delta l \cdot \sin(\theta_1)$$

Remark 1 *It should be noted that this odometry calculation is based on multiple approximations. This implies that the more often the odometry algorithm will be run, the more precise the estimation of the position and orientation will be over time. Moreover, errors will accumulate as the robot moves, and will become considerable after some time, if no method is used to correct these errors (for instance by using the maze posts to recalibrate the position).*

4.8 Speed Calibration for Feedforward Controller

The speed of our motors is controlled by sending a PWM signal to the motor driver. By varying the duty cycle δ of the signal, we can directly control the effective value of the voltage U_{eff} applied to the motors. The relationship is as follows:

$$U_{eff} = U_a \cdot \delta \tag{4.7}$$

Here, U_a is the voltage of the battery. In practice, the PWM command ranges from 0 to 255. A command of 0 results in a duty cycle of 0, and a command of 255 results in a duty cycle of 1.

At first, we aim to visualize the relationship between the PWM command sent to the motors and the actual wheel speed. To measure this, the robot is placed on a support so

that the wheels do not touch the floor. The robot is powered by the LiPo battery, which has a voltage of 7.4 V. Commands spanning the whole range are then sent to the motor, starting with a maximally negative command (motor spinning counter-clockwise) to a maximally positive command (motor spinning clockwise). The command is increased by a given step every few seconds. The result from this experiment is shown in Figure 4.16. One notable observation is that the relationship between the PWM command and speed is highly non-linear.

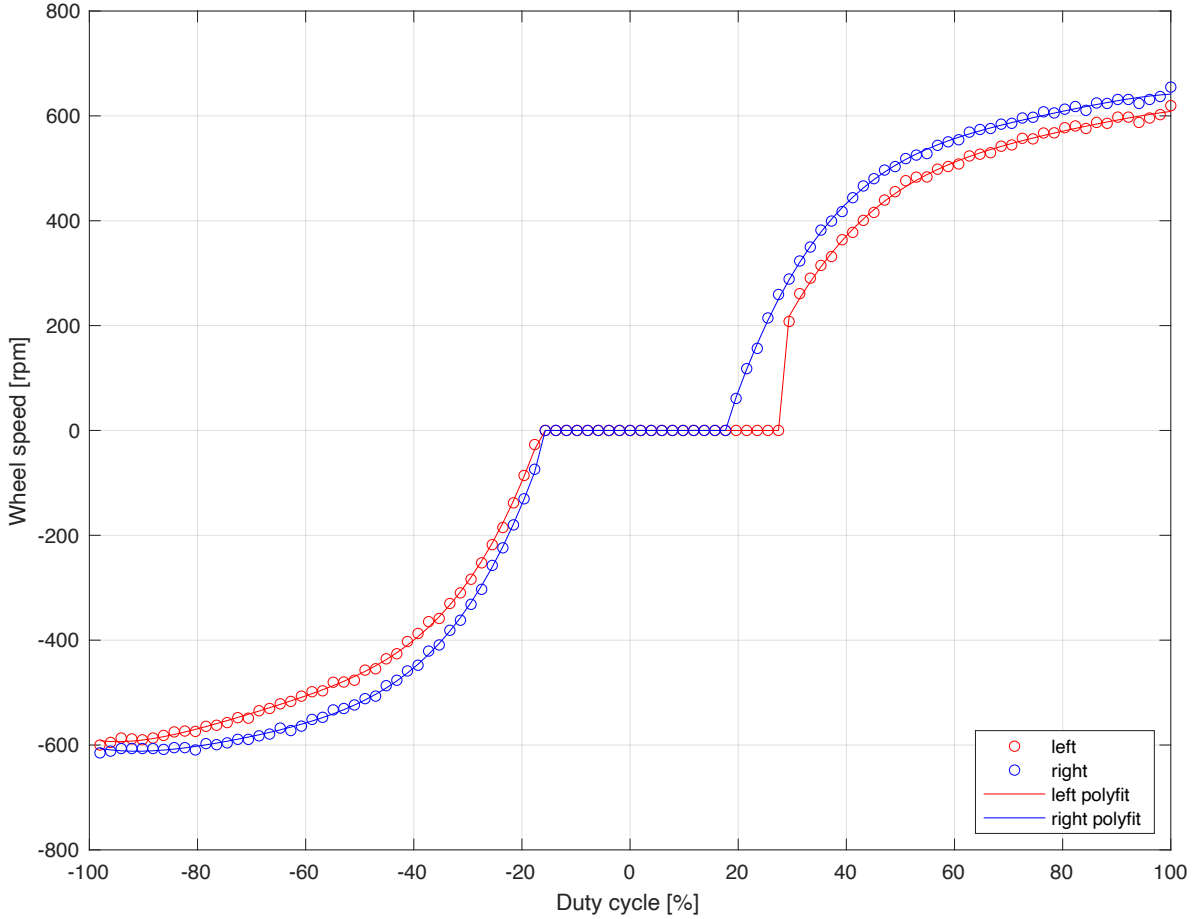


Figure 4.16: Speed of the wheels depending on the duty cycle, with degree 4 polynomial fits. Time between measurements: 2 s

This is surprising because the motor is not loaded, implying that the torque is zero. A direct consequence is that the current in the motor is also zero (see Equation 4.3). Since the current is zero, we can deduce the following from Equation 4.1):

$$u_a(t) = u_i(t) = k_u \omega(t)$$

This demonstrates that in this specific case, we would expect the motor's speed to be directly proportional to the applied voltage. In order to better understand this behavior, we conducted a test setup in which a DC power supply was directly connected to one of our motors, and an oscilloscope was used to measure the encoder's frequency. This experiment is explained in detail in the annex, in section 3. The resulting motor's speed depending on the input voltage is shown in Figure 4.17. On this figure, we can see that the response is indeed linear, as expected. We also measured the response for the motor

with and without the wheel, and observed that the wheel has very little influence. This eliminates the hypothesis that air friction is responsible for the non-linearity. Since the motor is not the source of the non-linearity, there are two possibilities left:

- The PWM commands sent to the motor are not directly proportional to the duty cycle. However, the documentation of the library used to generate the PWM signal states that the command is equivalent to the duty cycle.
- The motor driver influences the signal that is actually sent to the motors and is responsible for the odd behavior.

These are two hypotheses that would need to be tested, but we did not have the time to do so.

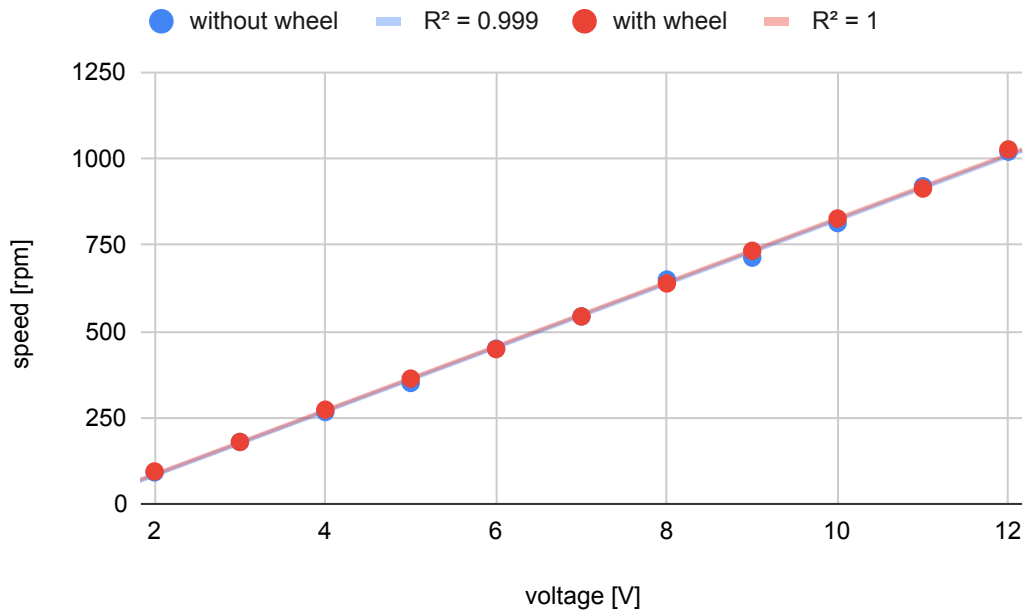


Figure 4.17: Motor speed with no load depending on input voltage, with or without wheel.

Despite the unexpected curve, we can still draw multiple observations from the speed vs. duty cycle plot shown in Figure 4.16:

- Both motors are different, which means that they will not spin at the same rate if the same command is applied to them. In other words, the robot will be slightly turning if we just apply the same command to both motors. To fix this issue and actually have the robot driving straight, we will have to regularize the robot's speed. This is further explained in the following sections.
- For the smallest commands, the motor does not start at all (when only the feedforward controller is used), due to the inertia of the robot.
- Since the motor is directly connected to the battery, the motors will not turn at the same speed for a given command depending on how charged the battery is, as the battery voltage drops when it discharges.

We now want to find a model that allows us to know which duty cycle should be applied to each of the wheels to achieve a certain speed. Indeed, this model can be used as a feedforward controller in the regulation of the robot's wheel speed. To find this model, we first use MatLab to find the type of function that best fits the data. In practice, the data is separated into three parts, and a model is computed for each. This is how the data is split:

1. The part where the speed is negative.
2. The part where the speed is zero.
3. The part where the speed is positive.

After experimenting for a bit, we realize that a fourth-degree polynomial yields a good approximation, as shown in Figure 4.18. In this figure, we can see that especially for the left motor, the polynomial is non-monotonous. In practice, this would imply that with this model, the command that we would apply to the motor to have an increasing speed would go down before increasing again. Since this dip in the speed remains relatively small, we decide to keep this model, because it is relatively simple and can be implemented on a micro-controller. Another problem we might have is due to the motors not being powered at their nominal voltage of 12 V, but at around 8 V. We can see that absolute speeds of 50 rpm and lower cannot be reached at all since the motor is not strong enough to overcome the inertia. We thus have poor control at low speeds.

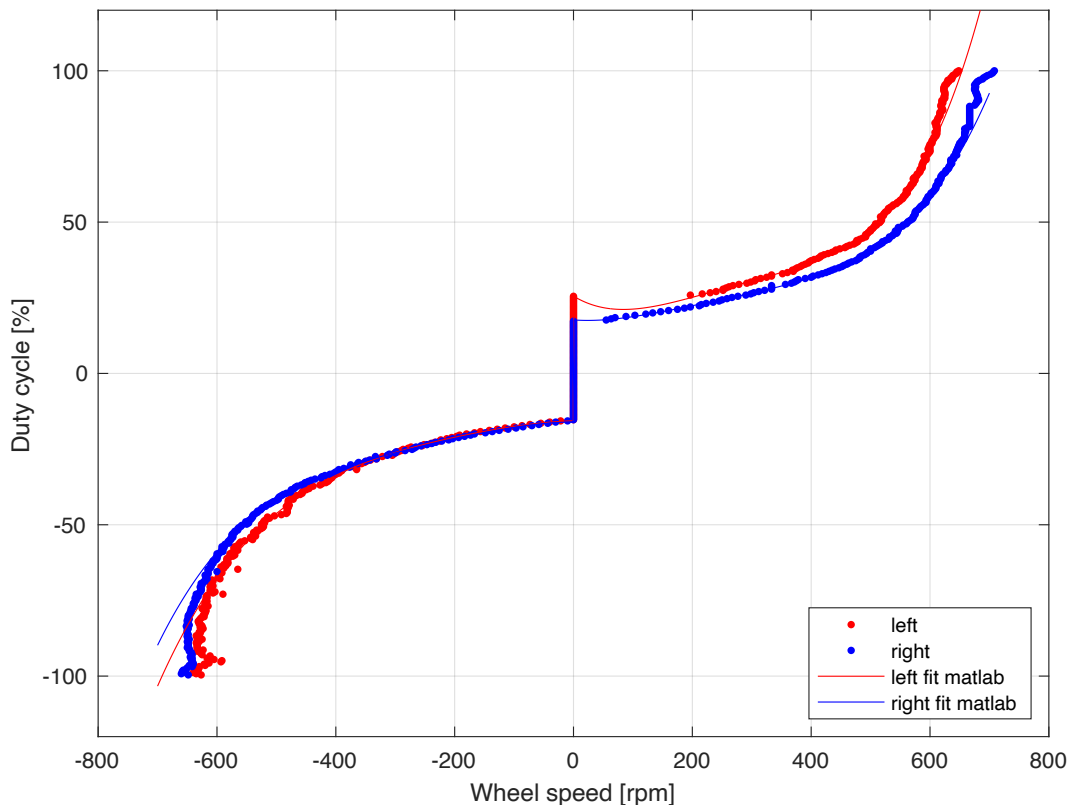


Figure 4.18: Duty cycle sent to the motors versus speed of the wheels, with 4th degree polynomial fits. Time between measurements: 1 s

The implementation of the speed calibration on the micro-controller is further discussed in section 9.8.

4.9 PID Regulator

Regulation is one of the key aspects for smooth navigation of a micromouse. Indeed, a micromouse will commonly integrate at least one regulator, either on the robot's speed or its position. In this section, we explain the theory behind proportional integral derivative (PID) regulators. As the name suggests, this type of regulator is a superposition of three terms, each of which has its own coefficient:

- The proportional term is directly proportional to the error. The proportional term will typically help to adjust the reactivity of the regulator. The corresponding coefficient is called K_p .
- The integral term is the integral of the error since the regulator was started. The integral term will compensate for a constant offset between the output and the setpoint. The corresponding coefficient is called K_i .
- The derivative term is the derivative of the error since the last iteration of the algorithm. The derivative term will ensure that there are no variations in the error over time (that a constant value is reached). The corresponding coefficient is called K_d .

A PID regulator can therefore be analytically expressed with the following equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt} \quad (4.8)$$

Where $u(t)$ is the PID control variable, and $e(t)$ is the error, defined as the difference between the setpoint and the output. A block diagram of the regulator is shown in Figure 4.19.

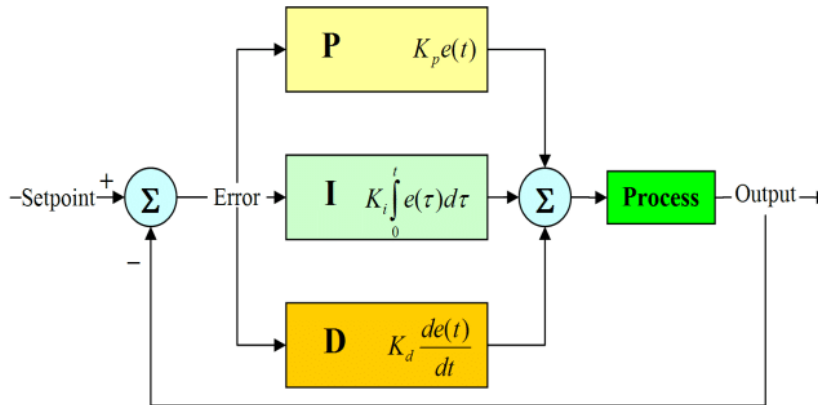


Figure 4.19: Block diagram of a PID regulator [6]

In practice, however, we are dealing with discrete values, which implies that the integral and the derivative terms have to be approximated. The integral is replaced by a sum of small rectangles, whose width is the time interval, and whose height is the current error. On the other hand, the derivative is the variation of the error during that interval. Listing 4.1 shows code to compute the new values for the integral and derivative terms at each iteration of the algorithm. In this snippet, `dt` is the time elapsed since the last call of the function.

```

1 | integral += error * dt;
2 | derivative = (error - previousError) / dt;

```

Listing 4.1: Code to update the derivative and integral terms in a discrete PID controller.

4.9.1 Filtering the Derivative Term

One issue with real-life data is that it is intrinsically noisy. In a PID controller, the D term uses the derivative of the error, which is extremely influenced by noise. To enhance the contribution of the derivative term, the sensor data can be smoothed before the calculation using a first-order filter. It is analytically expressed as follows:

$$x_{filt,i} = \alpha \cdot x_i + (1 - \alpha) \cdot x_{filt,i-1}$$

Where x is the sensor data, x_{filt} is the filtered data, and α is a smoothing parameter with a typical value of 0.1 [48]. We can see that the smaller α is, the stronger the smoothing.

To visualize the impact of the filter, we applied it to some test speed data that was recorded using the encoders of our motors. The results are displayed in Figure 4.20. The plots lead the following observations:

- The filtering adds a delay to the signal: the larger α is, the more the new curve is shifted towards the right.
- The filtering process also reduces the signal's amplitude.
- The close-up shows that the noise is effectively reduced using the filter.

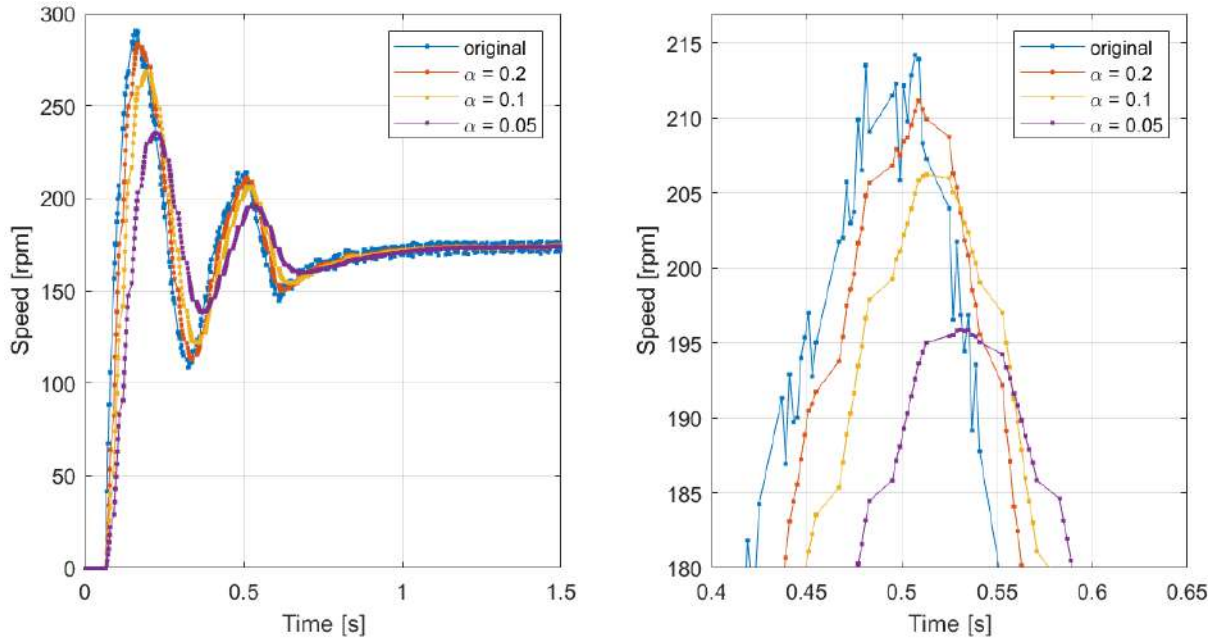


Figure 4.20: Effect of a first-order filter on the wheels' speed data. The right figure is a close-up showing that the noise is effectively reduced.

4.10 Robot's Speed Regulation

In the previous section, we saw that PID regulators are linear, which leads to a very interesting property: multiple regulators can be summed to regulate multiple properties simultaneously, while only actually controlling one output. This principle is used to regulate the robot's speed. Intuitively, we can imagine two solutions to control the speed. The first solution is to regulate the speed of each wheel separately. In that case, setting different targets allows the robot to turn, whereas setting the same target will make the robot move straight. The other solution is to directly regulate the robot's linear and angular speeds, which eliminates the need for the conversion from wheels to robot's speed. This is the approach that will be favored in the software implementation, as it allows for very easy control of the robot. The 1992 paper "Wall following Control of a Mobile Robot" by van Turenout *et al.* shows how the robot's angular and linear target speeds (respectively ω_{set} and v_{set}) can be achieved, as shown in Figure 4.21 [7].

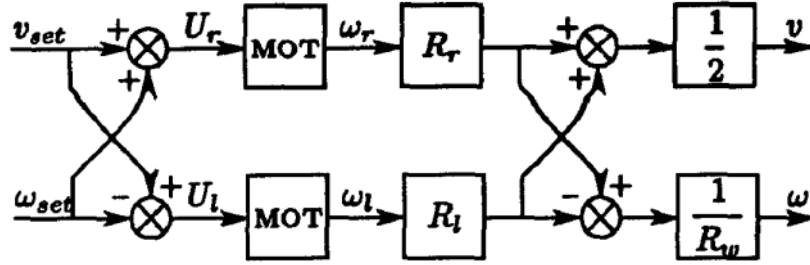


Figure 4.21: Block diagram modeling the control of the robot's speed [7]

4.11 Wall Following

Wall following in robotics is a common problem, in which a mobile robot must move in a desired direction along a wall, maintaining a constant distance from the wall [7]. Wall following is incredibly useful for navigating unknown environments, or for following an obstacle before resuming a desired trajectory. In the case of micromice, it is a major challenge, as we want the robot to explore an unknown maze and always remain in the middle of the corridors. A main issue with wall following is that the walls are often not continuous: they can be doors, lateral corridors, etc. This means that the robot requires both a closed-loop controller and a dead reckoning navigation system. Moreover, the robot has to dynamically switch between both modes depending on the presence or absence of walls.

The remaining question is how to concretely implement wall following for our application. In his presentation "Wall and Line Tracking for Micromouse and Line Followers" given at the 2023 MINOS conference [48], Peter Harrison simply proposes to consider the difference between the left and right distances as a form of angular speed error. A third controller is added to adjust the response to a distance difference, and the result of this controller is summed with the angular speed. This idea is shown in the block diagram in Figure 4.22.

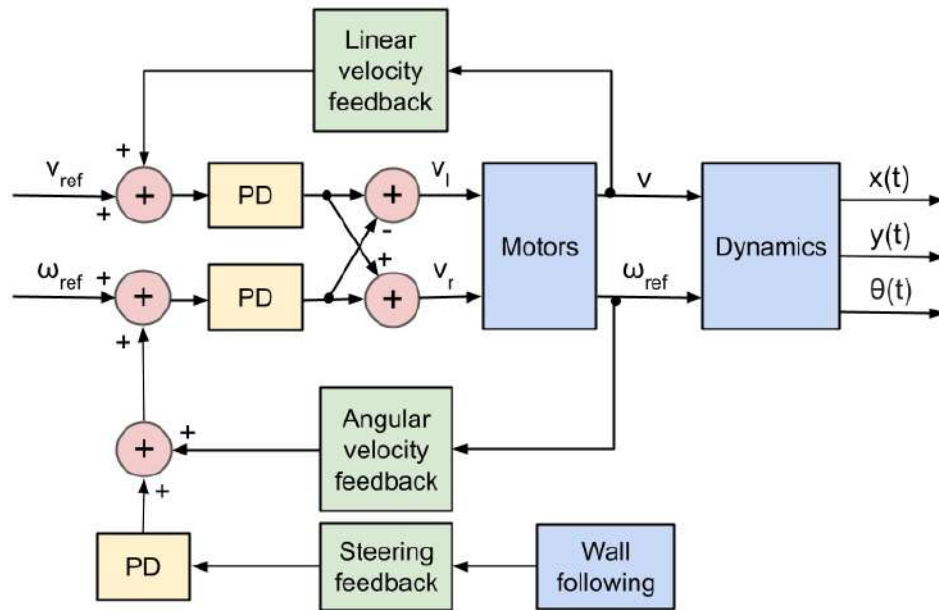


Figure 4.22: Block diagram of the robot speed control and wall following.

To summarize, the control of the robot is achieved with three separate controllers: one for the linear speed, one for the angular speed, and one for remaining in the middle of a corridor. One last interesting aspect highlighted by Harrison is that there is no need for a PID controller, which is too complex for the application. Harrison argues that PD regulators (with a first-order filter for the derivative term) are more than sufficient. Not having the integral term in the controller helps reduce the number of coefficients to optimize and therefore simplifies the control.

Chapter 5

Electronics

In the following sections, all the different parts of the robot's electronics are presented in detail.

5.1 Microcontroller

Our micromouse integrates an ESP32-S3 microcontroller board with a removable WiFi antenna. The board we chose, the ESP32-S3 DevKit by Espressif, has 45 GPIO pins, supports I²C, and directly provides WiFi and Bluetooth. The pin layout of the board is presented in Figure 5.1 and the main characteristics of the microcontroller are summarized in Table 5.1. The board's dimensions are 63 x 25 mm. Smaller boards integrating the same microcontroller were considered (like the ESP32-S3 board by Xiao), but they had an insufficient number of GPIO pins for our use case.

The ESP32 microcontrollers are widely used for IoT applications, which means that there is a lot of documentation on how to use them. This helped us significantly with the programming. Moreover, these microcontrollers are compatible with the Arduino environment, which gives us access to a multitude of libraries. Another significant advantage of this microcontroller is that it can handle interrupts on all pins, which gives us a lot of flexibility in the PCB design.

Reference	ESP32-S3-DevKitC-1U-N8R8
Manufacturer	Espressif
Flash	8 Mb
RAM	8 Mb
CPU frequency	Up to 240MHz
SPI clock frequency	Max 80 MHz by default
Wireless	WiFi and Bluetooth, removable antenna
Operating voltage	2.7 to 10.8 V
Board dimensions	63 x 25 mm

Table 5.1: Microcontroller main characteristics [28]

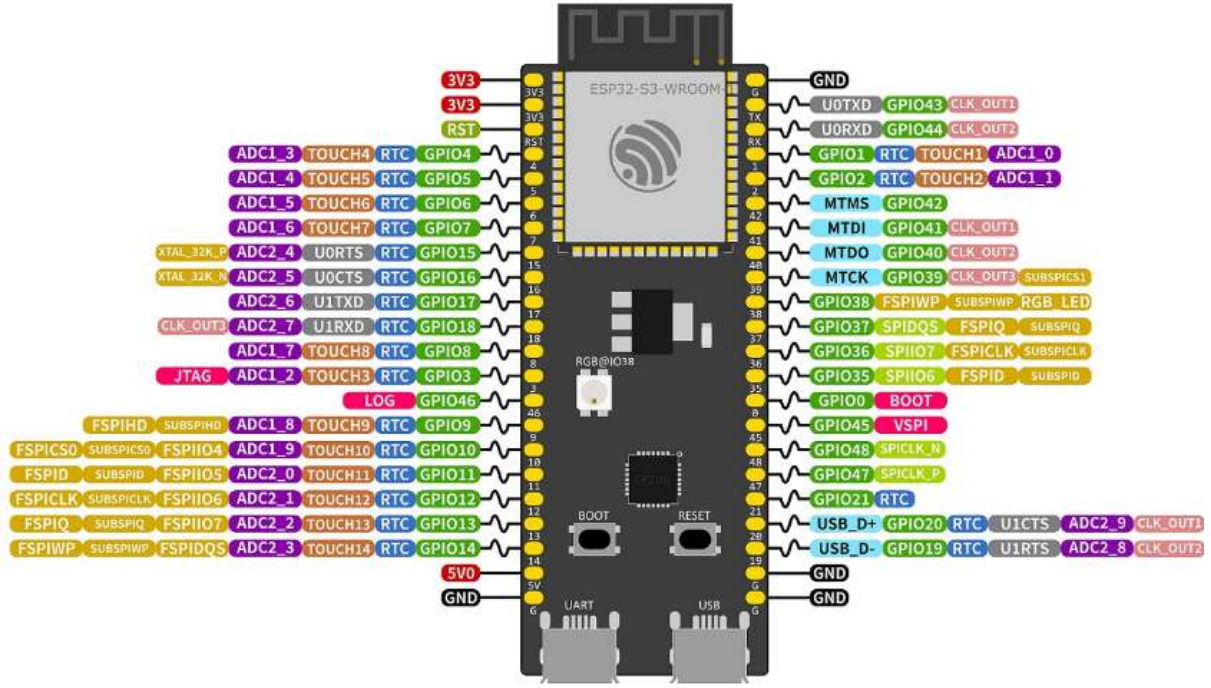


Figure 5.1: Pin mapping of the ESP32-S3-DevKit [28]

5.2 Motors

As discussed in Chapter 3, we selected brushed DC motors for our robot. The primary considerations when selecting our motors were the dimensions, power, nominal speed, input voltage, and cost. In our design, both motors are directly connected to the wheels. This means that the length of the motors should be less than half the width of the robot, corresponding to a maximum of 50 mm. Moreover, according to the calculations in Section 4.3, we determined that the maximum speed of the wheels should be approximately 900 rpm (for a robot's speed of 1.5 m/s), and the torque necessary to achieve the desired acceleration is 7 mNm (for a maximum robot acceleration of 5 m/s²). These two criteria, along with the maximum length of the motors, guided us in our search for suitable motors. We decided to use the Pololu micro metal gearmotor with a 30:1 reduction. The 3D model of the motor is depicted in Figure 5.2, and the motor's main characteristics are summarized in Table 5.2. The motors' nominal input voltage is 12 V. Thus, our 7.4 V battery can power the motors, but they will rotate slower than their nominal speed. A significant advantage of these motors is that they come in a variety of gear ratios while maintaining the same external dimensions. There are 13 different models, with gear ratios ranging from 5 to 986. This means that, if necessary, the motors could be replaced with another gear ratio without any hardware modifications to the robot.

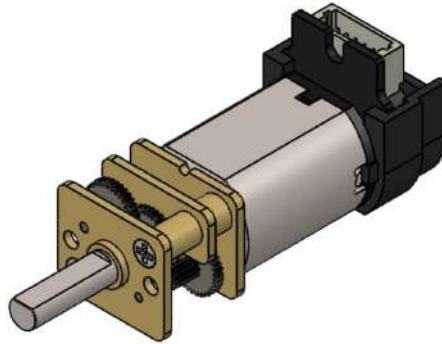


Figure 5.2: Schematic of the DC motors (with gearbox and encoder) used in the robot, total length: 32 mm [27]

Reference	30:1 Micro Metal Gearmotor HPCB 12V with Extended Motor Shaft
Manufacturer	Pololu
Nominal voltage	12 V
Gearbox ratio	30:1
No-load speed	1100 rpm
No-load current	80 mA
Max. power torque	20 mNm
Stall current	0.75 A
Max. power	1.1 W
Dimensions	10x12x32 mm

Table 5.2: Motors' main characteristics

5.3 Encoders

In our case, the choice of encoders was relatively straightforward once the motors were selected. Indeed, the manufacturer of the motors allows us to purchase the micro-motors with the encoders directly assembled on the secondary shaft. These are quadratic magnetic encoders with a resolution of 12 CPR. The encoder, with a side entry JST connector, is visible on Figure 5.2.

Reference	Magnetic Encoder Pair Kit with Side-Entry
Manufacturer	Pololu
Input voltage	2.7 to 18 V
Counts per revolution	12
Connector	6-pin female JST SH-type

Table 5.3: Main characteristics of the encoders

5.4 Motor Driver

The motor driver is required to convert the micro-controller's PWM signal into a high-power signal that will be the motor's input. The structure of the driver is relatively straightforward, as it is a dual H-bridge motor driver. This means that the driver can control two independent motors, and it does so using some NMOS transistors, which are driven by the microcontroller. Each H-bridge is composed of 4 transistors, as shown in Figure 5.3. By turning on two of the transistors on a diagonal, the motor can be rotated in a clockwise or counter-clockwise direction. The speed of the motor, on the other hand, is changed by modifying the effective voltage applied to the motor, by sending a PWM signal to the driver.

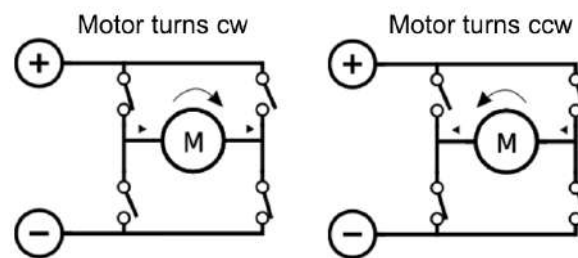


Figure 5.3: Schematic of an H-bridge: turning on the transistors on a diagonal allows the robot to spin clockwise or counter-clockwise [38]

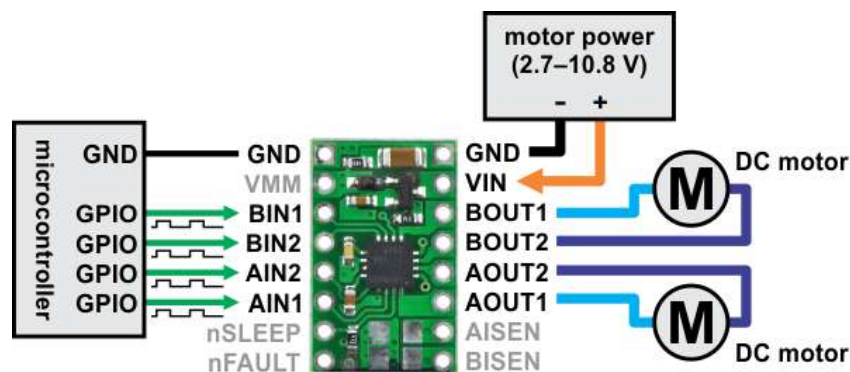


Figure 5.4: Minimal wiring of the motor driver [20]

Reference	DRV8833 Dual Motor Driver Carrier
Manufacturer	Polulu
Current	Max. 1.5 Arms output per motor
Operating voltage	2.7 to 10.8 V
Board dimensions	20 x 13 mm

Table 5.4: Main characteristics of the motor driver

5.5 Distance Sensors

In Chapter 3, we present the reasons for using time of flight distance sensors in our micromouse. We decided to use one of the ToF sensors manufactured by STMicroelectronics. Different manufacturers have carrier boards for various models, each with their own characteristics. After some research, we settled on the VL51L1X carrier board by TinyTronics, as it was the smallest board we could find. It is also the most inexpensive by a significant margin (6.5 EUR vs 19 EUR for the equivalent Polulu carrier board). In order to reduce the size of the sensors, we decided to remove the two semicircles on the sides of the PCB with holes for the screws (see Figure 5.5). The sensors are soldered perpendicularly to the main PCB using standard male-male 90° headers, therefore, no screws are required to mount the sensors. To remove the extra part of the sensors' PCB, cutting pliers were used. The edges were then smoothed using files. This modification reduces the sensors width from 25 mm to 14.5 mm, thus reducing the sensors' footprint on our custom PCB. The original and modified distance sensors are shown in Figure 5.5.

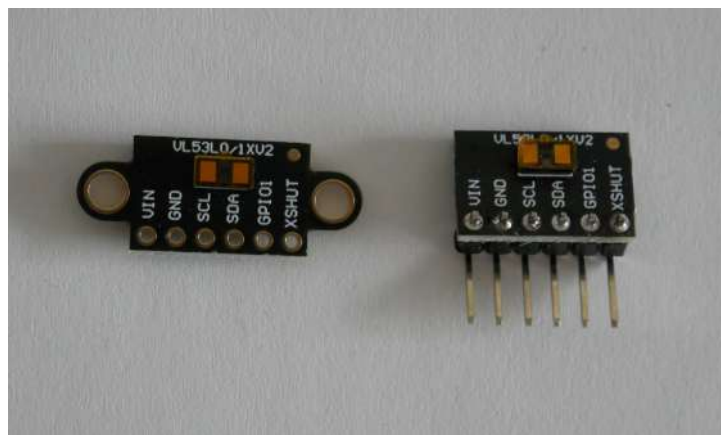


Figure 5.5: The original VL51L1X TinyTronics carrier board on the left, the modified board on the right.

Reference	VL53L1X ToF Distance Sensor
Manufacturer	TinyTronics
Range	30-4000 mm
Supply voltage	3.3 to 5 V
Board dimensions	25 x 10.7 mm
Communication	I ² C
Wavelength	940 nm

Table 5.5: Main characteristics of the distance sensors

Inspecting the sensors' datasheet provides some interesting insight into how they work and how to use them. When using this type of sensor, the "timing budget" for the measurement is a crucial parameter: it is the time allocated to a measurement. Figure 5.6 shows that the timing budget affects the maximum distance that can be measured, as well as the repeatability of the measurements. As explained in the sensors' datasheet:

"Increasing the timing budget increases the maximum distance the device can range and improves the repeatability error. However, average power consumption increases accordingly."

Another disadvantage of a large timing budget is that the distance is updated less frequently, which can impact the quality of our wall following, as the distance measurements are used in the controller. Indeed, the more frequently the data is updated, the better the control.

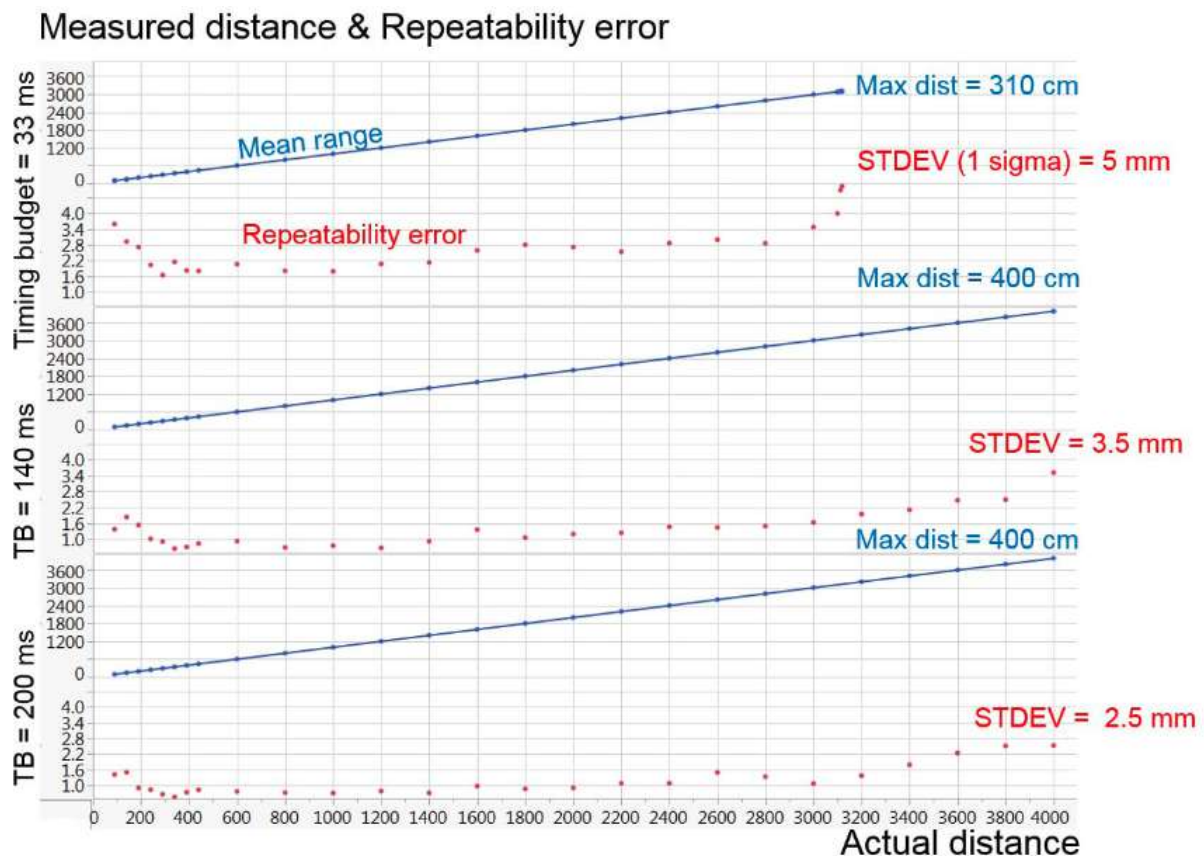


Figure 5.6: VL51L1X sensor measured distance and accuracy for various timing budgets. This figure is from the sensor's datasheet.

5.6 Buzzer

The only constraint for the buzzer was that it needed to be as small as possible to minimize space usage on the PCB. We chose one of Murata's buzzers due to their small footprint and SMD component status. The buzzer functions by applying a PWM signal at the desired sound frequency to the component's pins.

Reference	PKLCS1212E20A0-R1
Manufacturer	Murata
Technology	Piezo-electric
Maximum operating voltage	25 V
Dimensions	12 x 12 x 3 mm

Table 5.6: Main characteristics of the buzzer

5.7 Battery

Having already decided to use a LiPo battery in Chapter 3, we are left with the question of how many cells the battery should have and how they should be connected. LiPo batteries consist of elementary cells with a nominal voltage of 3.7 V, which can reach up to 4.1 V when fully charged. These cells can be connected in parallel to sum their voltages, or in series to increase the maximum output discharge current. We chose a 2P1S battery, meaning there are two single cells in parallel. This battery has a nominal voltage of 7.4 V and a capacity of 1100 mAh. We did not choose batteries with 3 cells in parallel, even though the nominal voltage would then have been closer to the motors' nominal voltage (11.1 V), for the following reasons:

- 3S1P batteries are larger and would be difficult to fit on our micromouse.
- The maximum input voltage of our motor driver is 10.8 V, so we would be out of the specifications.

Given the current consumption of our various components, we estimate the robot's autonomy to be approximately 1 h with this battery. This estimate assumes that the WiFi is activated on the micro-controller, all sensors are functioning, and the motors are operating at full speed 70 % of the time. By examining the battery's characteristics, we also find that the maximum continuous output current is far more than what we will need (see Table 5.7).

Reference	RD XT 1100 S2
Manufacturer	Red Power
Chemistry	Lithium polymer
Capacity	1100 mAh
Configuration	2S1P
Nominal voltage	7.4 V
Max. continuous output current	27 A
Dimensions	55 x 30 x 17 mm

Table 5.7: Main characteristics of the battery

5.8 IMU

The IMU integrated into the micromouse is the MPU6050 3-axis accelerometer and 3-axis gyroscope. This component is one of the most affordable IMUs on the market (around €3), and it is often used in low-budget projects that do not require excessive precision. We decided to use the breakout board by InveSense, which will come as a shield on our own PCB, making it much easier to solder than the accelerometer's SMD chip itself. Moreover, it is also an I²C device, which means that it can be added to the I²C bus and the data can be directly read in [m/s²] and [deg/s].

Reference	MPU6050
Manufacturer	InveSense
Features	3-axis accelerometer and 3-axis gyroscope
Supply voltage	2.4 to 3.5 V
Board dimensions	21 x 15.5 mm
Communication	I ² C

Table 5.8: IMU's main characteristics

Chapter 6

PCB Design

As mentioned in Chapter 3, a custom PCB was designed. Indeed, creating our own PCB is the simplest way of connecting all our components reliably. Often when prototyping, the various electronics are connected using a breadboard and cables. However, this is very bulky and impractical because some wires may become unplugged, which takes a lot of time to debug. On the other hand, having all the components on a single PCB makes for a robust and compact prototype. In our case, the various components were first tested individually using breadboards with jumper cables to ensure the correct connections would be realized on the PCB. Then, the schematics of the PCB were outlined, providing a better overview of how the different components interact with each other. The complete schematics of the final version of our robot (Algernon v1.1.0) can be found in the annexes of this report (see Figure 11.5). Finally, the layout of the PCB was created, defining the actual board dimensions and shape, and positioning and physically connecting all the components with copper traces. We opted for a standard two-layer PCB, which allows for component placement on both sides of the board. The final PCB is shown in Figure 6.1. The PCBs were sent for manufacturing to the German company Aisler.² The boards cost around 17 euros each and were produced within a week.³

All of the design was created in KiCad EDA, an open-source electronics design automation suite.⁴ KiCad may not be the most advanced PCB software available, but it is easy to understand and use, even for people with little electronics background (like me), and is perfectly suited for a project of this complexity. Moreover, it is free and open-source, which allows anyone to download the robot's files and reuse some of them in personal projects. Since our robot is open-hardware (under the CERN-OHL-P license), it makes sense to use open-source alternatives whenever possible. Furthermore, a useful feature of KiCad is the 3D viewer tool which allows for a 3D model view of the PCB with all the components. This is helpful for verifying component positions and getting a better idea of the relative component sizes. It also makes it easy to check where the different connectors are and whether they will be accessible on the final board. Figure 6.2 shows the final 3D view of our robot.

²Access Aisler's web page here: <https://aisler.net/>

³It was the first time we had PCBs produced in Europe, and we were very pleased with the quality, delivery time, and customer support.

⁴The main web page of KiCad EDA is available here: <https://www.kicad.org/>

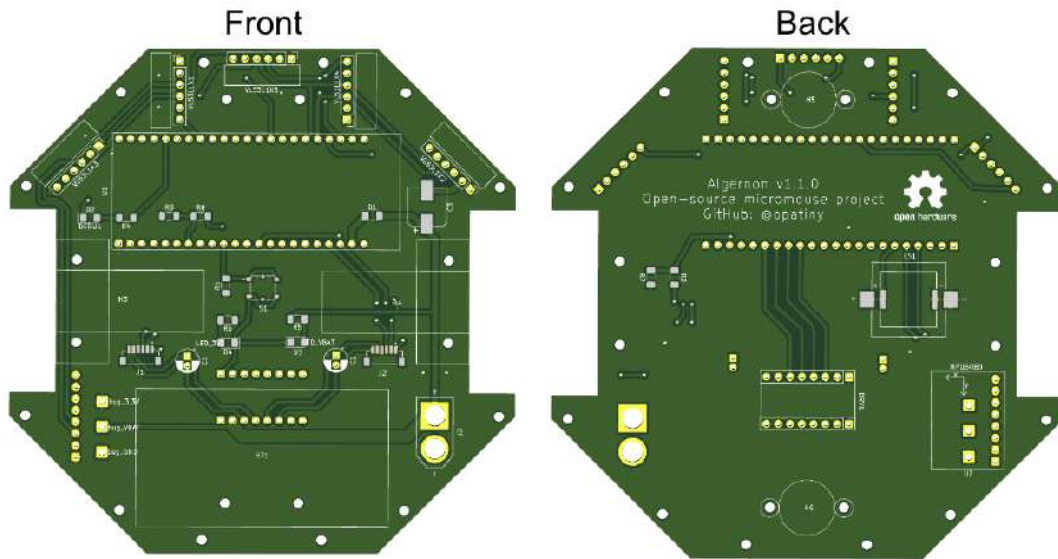


Figure 6.1: The final custom PCB front and back views

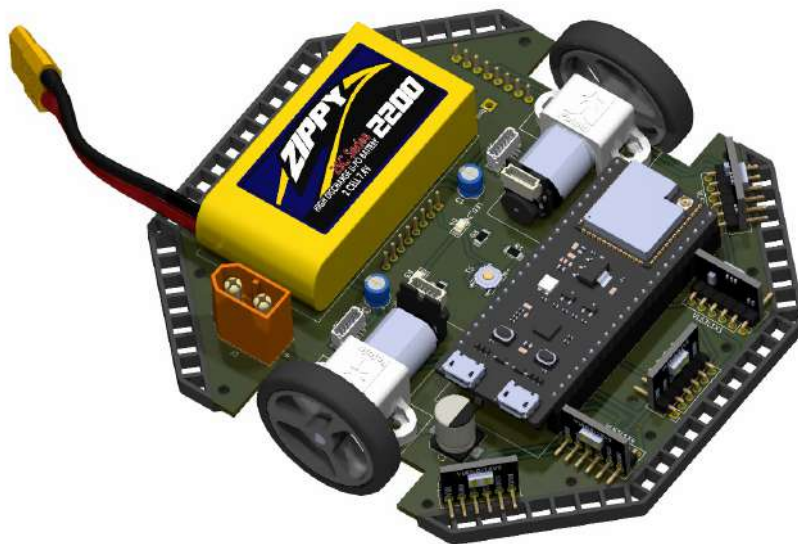


Figure 6.2: Final 3D view of the PCB with all the components

6.1 Design Principles

Our overall goal with this PCB is to create a compact and robust robot prototype. To this end, we strive to minimize the complexity of the board and the number of components. We favor fully integrated circuits over surface mount devices (SMD). The various sensors, for example, come as shields that can be easily soldered onto the main board using standard 2.54mm headers. For the remaining components, we predominantly use SMD components, which optimize the usage of both sides of the PCB. In this case, "hand-solder" footprints are employed, facilitating easier hand soldering of components. The smallest components on the board, the 12x06 resistors and LEDs, can be easily soldered by hand without the need for binoculars, using a standard soldering iron.

6.2 Voltage Regulation

Voltage regulation is a fundamental challenge in micromouse projects. Multiple components often require different input voltages, necessitating the use of several voltage regulators. This is particularly true for more advanced mice, such as Green Ye's micromouse Green Giant, which uses four distinct voltage regulators [65]. In our case, we only need two different voltages:

- 3.3 V: Used by the micro-controller, the accelerometer, the encoders, and the distance sensors
- 7.4 V: The battery voltage, used as the motors' supply.

Upon reviewing the ESP32 datasheet, we find that the on-board voltage regulator, an SGM2212, can withstand an input voltage from 2.7 to 20 V and outputs 3.3 V. Thus, we can connect the ESP32 board directly to the battery and use its internal regulator to generate the voltage for all other components. This eliminates the need for an additional voltage regulator on our PCB, simplifying the design.

6.3 Voltage Measurement

We use simple voltage dividers to measure the battery voltage and the microcontroller's output 3.3 V voltage, as depicted in Figure 6.3. The voltage is measured using analog to digital converters (ADCs) on the microcontroller. Voltage dividers are necessary because the maximum voltage that the ADC can measure is 1.1 V. The original voltage V_{in} can be easily derived from the measured voltage V_{out} using the following formula:

$$V_{in} = \frac{R_1 + R_2}{R_2} V_{out} \quad (6.1)$$

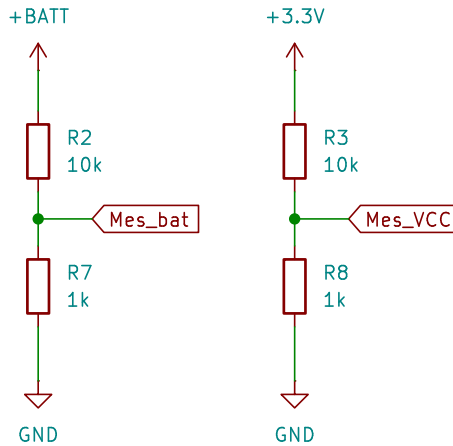


Figure 6.3: The voltage dividers for the measurement of the battery and 3.3 V voltages

6.4 Distance Sensors' Pinout

Multiple distance sensors are used on the same I²C bus, necessitating the modification of their addresses as they all have the same factory address. The XSHUT pin is utilized for this purpose, allowing the devices to be turned on and off. The sensors' pinout is displayed in Figure 6.4. The procedure to modify the devices' addresses is detailed in section 9.7.

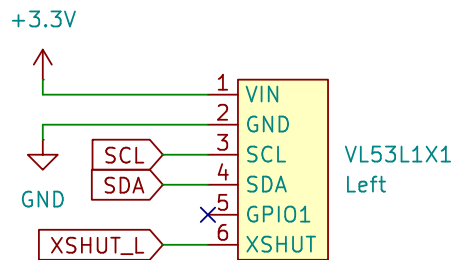


Figure 6.4: Pinout of one of the distance sensors

6.5 Motor Driver Pinout

A dual motor driver is used in our setup, requiring only one component for the control of both motors. On one side, the driver is connected to the four GPIO pins of the microcontroller. On the other side, the four high power pins are connected to the motors' terminals. This is depicted in Figure 6.5.

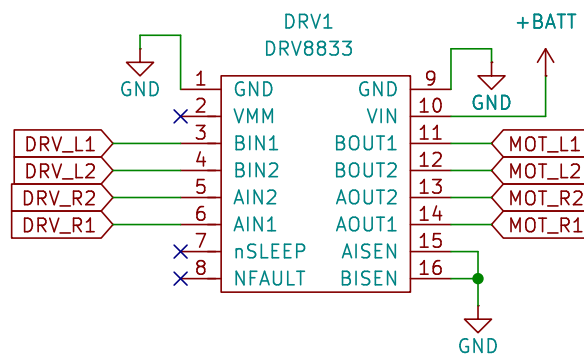


Figure 6.5: Pinout of the motors' driver

6.6 Reducing the Motors' Electrical Noise

In our design, both the motors and the microcontroller are directly connected to the battery. The issue with BDC motors is that the commutator brushes generate significant electrical noise, which can cause multiple problems, as described by the manufacturer of our motors:

"One major drawback to working with motors is the large amounts of electrical noise they produce. This noise can interfere with your sensors and can even impair your microcontroller by causing voltage dips on your regulated power line. Large enough voltage dips can corrupt the data in microcontroller registers or cause the microcontroller to reset." [35]

A solution to significantly reduce the motors' noise is to add capacitors between the terminals of the motor. In this case, it is critical to use ceramic capacitors, which are not polarized, since the current can flow in both directions. Based on the recommendations of the manufacturer of our motors, we added a $0.1\ \mu\text{F}$ capacitor between the terminals of our motors. This is visible in Figure 6.6. The same source also recommends using twisted cables to power the motors, which we also do on our robot. Finally, we decided to decouple the microcontroller using a $220\ \text{mF}$ electrolytic capacitor, which should smooth the $3.3\ \text{V}$ generated by the microcontroller's voltage regulator.

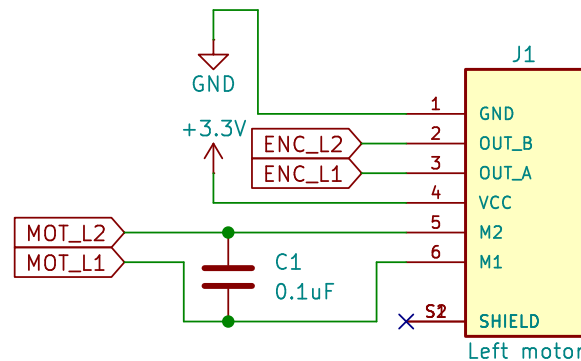


Figure 6.6: The JST connector for the motor and the encoder. A capacitor is placed between the two inputs of the motors to reduce the motor's electrical noise.

6.7 Protection of the Power Lines

Our microcontroller can be powered in two different ways: either by using the micro USB connector or by plugging in our battery. It is therefore essential to protect the various power sources. We wouldn't want to recharge the battery through the USB (the LiPo battery requires a specific charger), or to inject some current from the battery to the computer connected to the robot. By analyzing the ESP32 schematic, we can see that the USB connector is already equipped with a diode that prevents the current from flowing from the microcontroller to the USB cable. We still had to prevent the battery from being recharged by the USB, however. To achieve this, we added a Schottky diode on the track of the battery's positive voltage. We used a Schottky diode instead of a regular diode because its forward voltage drop is lower than the $0.7\ \text{V}$ for a regular diode.

In our case, we use a Schottky diode with a reverse voltage of 20 V and a continuous forward current of 1 A, which forward voltage drop is only 0.23 V. The diode is visible in Figure 6.7.

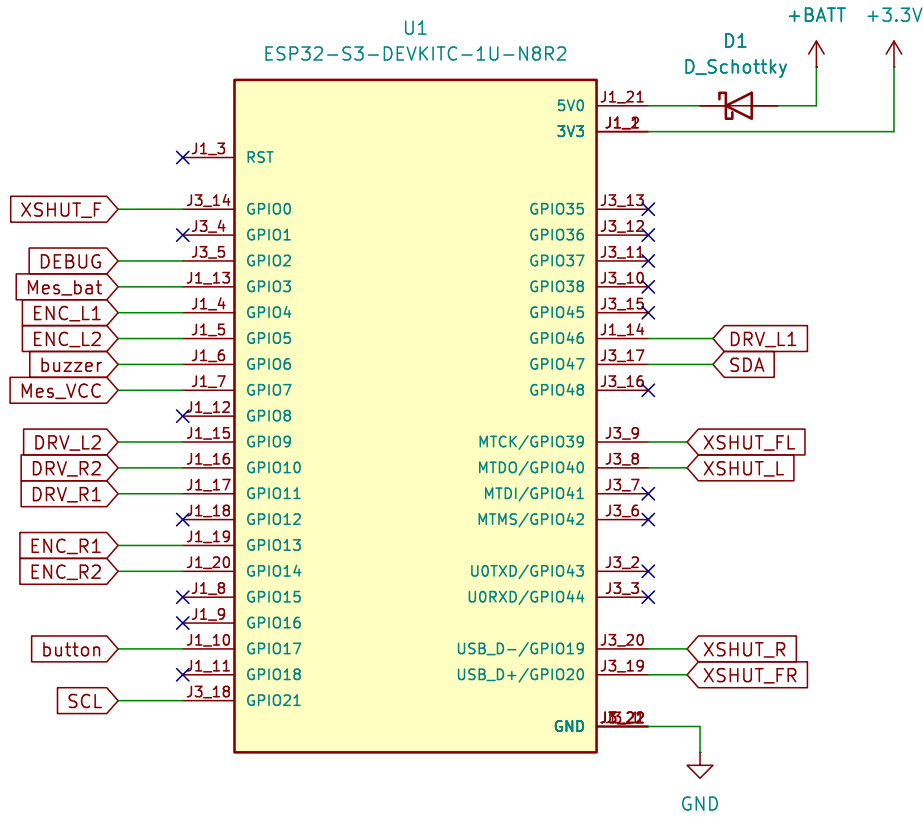


Figure 6.7: Layout of the microcontroller's connections. The Schottky diode prevents the battery from being recharged when a USB cable is connected to the MCU.

6.8 Debug Components

Multiple debug elements were added to the mouse to allow better human-robot interaction, as well as to assist with debugging throughout the development of the software. The following components were implemented for this purpose:

- **Debug LEDs:** Three debug LEDs allow for a quick view of the robot's status. One LED is connected to the battery and is turned on when the battery is correctly plugged in. The second one is on the microcontroller's 3.3 V output, which powers all of the sensors. Finally, an LED is configured to blink in the lowest priority thread on our robot. This allows us to directly see whether the threads are all running properly or not.
- **Buzzer:** The robot is equipped with a buzzer, which can generate various sounds depending on the status of the mouse. For instance, the mouse will warn the user after booting or when the push button is pressed.
- **RGB LED:** The microcontroller board that we chose directly integrates a programmable RGB LED. We use it to display the robot mode, as well as to warn the user that the battery is empty (blinks in red).

Chapter 7

Mechanical Design

7.1 Custom Caster Wheels

Overall, the robot's center of mass should be as low as possible to increase stability. In our case, the height is essentially defined by the diameter of the wheels. Since the motors are fixed directly on the top face of the PCB, the distance from the PCB to the floor is:

$$z = \frac{d_{wheel} - h_{motor}}{2} = \frac{32 - 10}{2} = 11 \text{ mm} \quad (7.1)$$

This implies that the maximum height of our caster wheels is 11 mm. If the caster wheel is smaller, the height can easily be adjusted using a spacer. We had some commercial caster wheels available; however, these were too high (15 mm) [60]. We therefore decided to design our own caster wheels. The body of the wheel is 3D printed from PETG, and the rolling element is an 8 mm stainless steel ball bearing. The assembly of the caster wheel is straightforward: the ball is simply pushed inside the body. It will snap into place thanks to the slight elasticity of the plastic. The friction between the plastic and the ball is reduced using a lubricant, in our case, graphite powder. The caster wheel is then fixed on the mouse using two M2x6mm screws and bolts.



Figure 7.1: Comparison of the commercial (left) and custom (right) caster wheels. Our caster wheel is 11 mm high, and the commercial one is 14.7 mm high.

7.2 Wheels

Algernon uses commercially available wheels from the same brand as the motors: Polulu. The wheels are easy to mount on the motor's axis thanks to the D shape of the connector matching the shape of the motor axis. The wheels just have to be pushed onto the axis and are held in place thanks to friction. The wheels have a diameter of 32 mm and are equipped with silicone tires, which ensure a high friction coefficient with the floor.



Figure 7.2: The wheels used for the robot (32 mm diameter)

7.3 Battery Mounting

The battery must be securely fastened to the PCB to prevent movement while the robot is in operation. Additionally, the battery should be easily replaceable, as the robot only has a run time of approximately one hour. A simple and elegant solution is to utilize a hook-and-loop fastener on both the PCB and the battery. We used a fastener with adhesive on one side, which allows it to be easily affixed to any surface. To mount the battery, it merely needs to be lightly pressed against the hook and loop fastener on the PCB. The fastener is durable enough to keep the battery in place even during impacts.

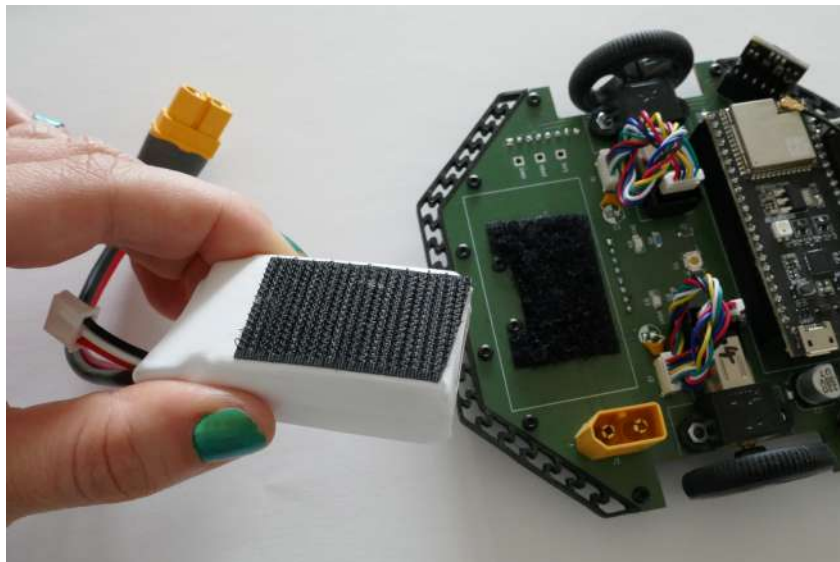


Figure 7.3: Hook and loop fastener is used for the mounting of the LiPo battery.

7.4 Bumpers

To safeguard our robot from anticipated impacts, we decided to outfit it with bumpers at the front and back. The concept is to have a mildly elastic component that will absorb the energy in the event of an impact, thereby protecting the PCB. We opted to use 3D printing to produce the bumpers, selecting a flexible filament. The filament used is FiberFlex 40D by Fiberlogy, which has a shore D hardness of 40, akin to a shoe sole. We designed the bumper to cover three edges of the robot and created two distinct versions. The initial version resembles a grid with straight spacers, while the second version is more advanced, comprised of an array of Z-shaped spacers. This design allows the plastic to bend more (see Figure 7.5), like miniature springs, and decreases the overall rigidity of the bumpers. The design of the bumper minimizes the materials used, thus reducing the bumper weight to only 2.5 g. The bumpers are identical for the front and back and are mounted using four M2x6mm screws and bolts per bumper.



Figure 7.4: The two versions of the robot bumpers: version 1 on top, final version on bottom

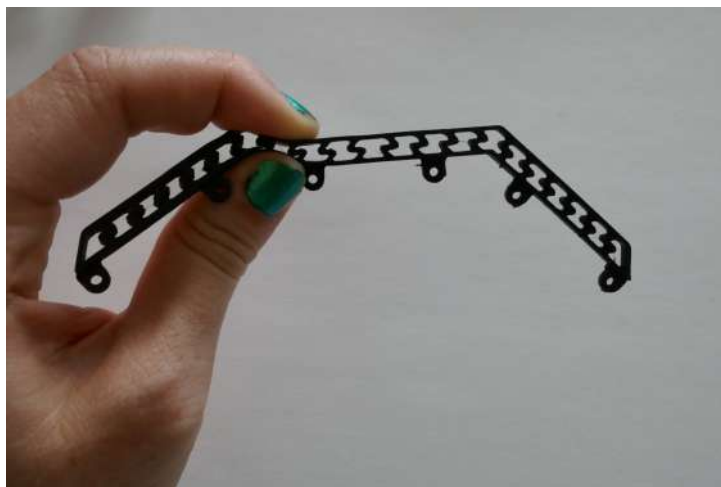


Figure 7.5: The final version of the bumper is easily deformed when a force is applied to it, allowing it to absorb shocks more effectively.

Chapter 8

A Custom Maze Design



Figure 8.1: Example of a small maze section

To facilitate the testing and debugging of our robot, it was deemed suitable to create a simple and reliable system for modular mazes. All files required to manufacture the different parts of the maze are available in the `opatiny/micromouse` repository.² This is primarily useful when the robot can be controlled reliably, and the maze-solving strategies are implemented. However, it's also necessary to implement wall-following for the robot. For these reasons, we decided to design our own maze system. Since we are not tackling maze-solving in the scope of our project, we are not interested in creating a full 16x16 cell maze, but will limit ourselves to an 8x8 cells prototype. The design should adhere to the following constraints:

- The dimensions of the maze cells' interior should closely match those of the contest mazes.
- The maze should be easily assembled and disassembled without any tools.
- Aim to minimize the weight, cost, and storage volume of the maze.
- Facilitate manufacturing: The process should be as quick as possible and minimize manual steps. Moreover, we prefer to use machines commonly found in a makerspace.

²The maze files are available in this folder <https://github.com/opatiny/micromouse/mechanics/maze>

8.1 Contest Maze Dimensions

A maze for a full-size micromouse robot competition consists of 16x16 square cells. Each cell has a width of 18cm, excluding the wall thickness, resulting in an inner cell side of 16.8cm. All the walls are 12mm thick and 5cm high. Additionally, the rules state that the outer wall must be standalone, and there should be a post at each cell corner. The walls are painted white. Another important aspect is that the posts of the contest mazes are fixed directly onto a large support board. This means a massive, 3x3m board is required as a base for the maze. In our case, we aim to design a "stand-alone" maze that doesn't require this base board.

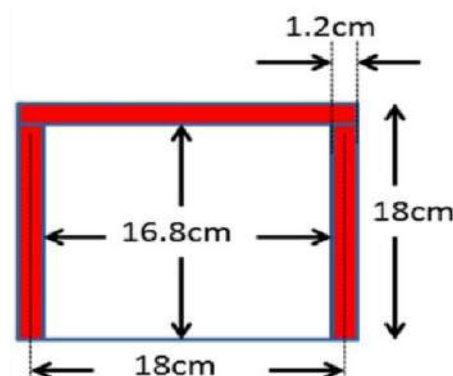


Figure 8.2: Dimensions of a micromouse contest maze cell [67]

8.2 Walls

We reduced the wall thickness to only 3mm, a quarter of the original thickness. This reduction lowers both cost and weight. Additionally, thinner walls can be cut using a laser cutter, a quick and easily learned manufacturing method. We chose medium density fiberboard (MDF), an affordable material that is well-suited for laser cutting. MDF with a 3mm thickness typically has a surface density of around 2.3 kgm^2 , which is still relatively heavy once all pieces are cut out. For an 8x8 maze, we estimate the weight to be around 3kg. Raw MDF has a light brown color. Dyed MDF, or MDF with a colored surface finish, are available but harder to find and more expensive. Thus, we chose to use basic MDF, painting it with multiple layers of white spray paint. Figure 8.3 illustrates the painting process. This figure also displays the varying lengths of the wall segments. We cut parts that are 1, 2, or 4 cells long, simplifying maze assembly and reducing play along a long wall due to fewer joints. Figure 8.4 shows two finished wall parts with visible slits for the corner pieces.



Figure 8.3: Painting the walls with white spray paint: 0, 1, and 3 layers of paint



Figure 8.4: Close-up of one cell long wall segments

8.3 Corners

In our design, we used plastic corner pieces as connectors between the walls. These parts are 3D printed from white PETG filament, facilitating mass production. As shown in Figure 8.5, there are four different types of connectors, chosen based on the number of walls meeting at a single corner. The connectors tightly fit with the wall parts. However, this system is sensitive to the MDF width, which can vary by a few tenths of a centimeter from board to board.

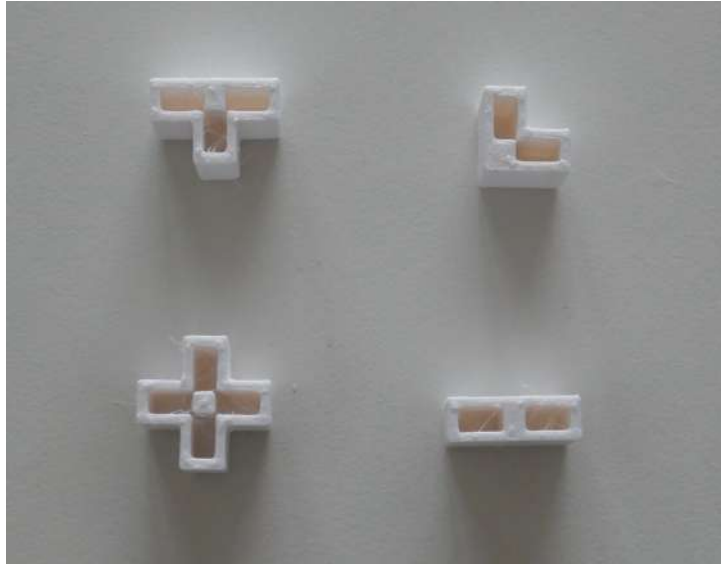


Figure 8.5: The four different types of connectors: straight, corner, T, and cross



Figure 8.6: Close-up of a corner of the maze

8.4 Bridges

Since we designed a standalone maze without a base board with fixed posts, we encountered issues with walls not being perfectly straight when multiple small segments are connected. To improve this, we designed a "bridge" part that connects two parallel walls, ensuring they are at the correct distance from each other. It also improves wall parallelism and adds rigidity to the overall structure. The bridge consists of two elements: the main part, laser-cut from 3 mm MDF, and a connector part, which is 3D printed (see Figure 8.7). Figure 8.8 shows how the bridge part is used in a maze.



Figure 8.7: The bridge part ensures that the walls are parallel.

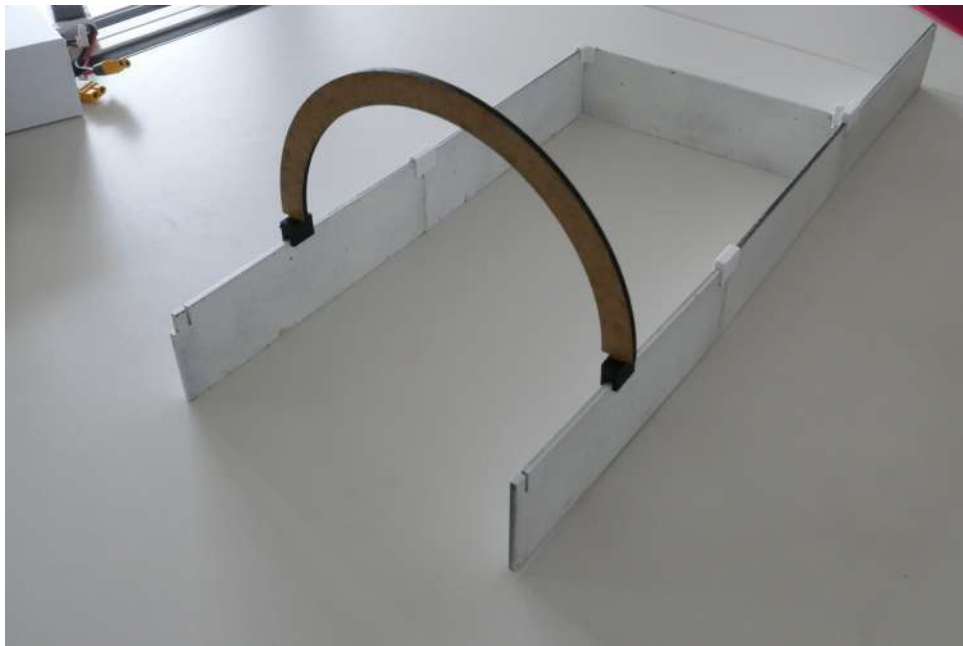


Figure 8.8: A maze section with a bridge part

8.5 Possible Improvements

The current design could be improved in several ways:

- The current inner cell width is incorrect. We designed the maze for the inner side to be 18 cm instead of 16.8 cm.
- When stored for a long time, the wall parts tend to stick together due to the paint. When the parts are separated, some of the paint flakes off, leaving little brown spots. We might want to change the paint used.
- The mechanical play along a wall composed of multiple segments depends on the fit between the connectors and the walls. This is problematic because it is very sensitive to any variation of the MDF width, as well as the number of layers of paint.

Chapter 9

Software

9.1 Programming Environment

The ESP32-S3 is programmed in C++, using the PlatformIO extension in Visual Studio Code. PlatformIO is a very powerful and open-source IDE that allows writing, debugging, and compiling code easily for hundreds of different hardware boards [58]. Furthermore, it enables the inclusion of a multitude of libraries in the project. Typically, since the ESP32-S3 is Arduino compatible, the Arduino library has been used throughout the project to set pin modes, handle interrupts, set pin outputs, etc. One of the main advantages of PlatformIO is the integrated serial monitor, which allows logging variables on the serial port. This tool has been used extensively throughout the software development for debugging purposes.

9.2 Base Project

The software for our robot was based on an already existing repository for the ESP32-C3 microcontroller.² This provided us with a code base that could be reused, thus allowing us to have a working environment much faster. The repository is structured to store software that can be reused from one project to another. For instance, it has some base tasks for multiple types of sensors. On the other hand, it contains tools for setting up the micro-controller's WiFi. Finally, it offers some memory management tools, as well as a very practical system allowing to read and write parameters through the serial interface. All the code that was reused from this repository is in the `lib/Hack` folder.³

9.3 Multitasking and Real Time

A micromouse has multiple sensors and actuators, which have to be handled in parallel. Moreover, the execution of specific actions at precise times is essential. In order to achieve these two goals, we use FreeRTOS⁴, a real-time operating system (RTOS) for micro-controllers. This allows us to have multiple "tasks" running concurrently. Each task is a program with a given priority. Depending on the priority, the task is given

²The Hackuarium/esp32-c3 repository can be found here: <https://github.com/Hackuarium/esp32-c3>

³Link to the code reused from the Hackuarium/esp32-c3 repository:
<https://github.com/opatiny/ms-software/lib/Hack>

⁴The FreeRTOS home page: <https://www.freertos.org/index.html>

more or less processing time. The tasks are handled by a part of the kernel called the "scheduler". The scheduler is responsible for switching between the different tasks. The tasks appear to run at the same time, because the switching is done very rapidly, but in reality, only one task is running at a time. The scheduler also handles other types of events, like interrupts and timers [59]. This is what allows executing given functions at very precise times, and therefore have real-time code execution.

The FreeRTOS documentation describes real-time programs in the following way:

"Real-time embedded systems are designed to provide a timely response to real-world events. Events occurring in the real world can have deadlines before which the real-time embedded system must respond and the RTOS scheduling policy must ensure these deadlines are met." [59]

A typical example in the case of a micromouse would be that if an obstacle is detected in front of the robot, it should stop. The robot has to react as fast as possible to prevent crashing into the obstacle. To achieve that, the navigation task and the distance sensing tasks have to run in parallel and interact with each other.

In our case, we have 15 different tasks handled by our micro-controller. Since the ESP32-S3 is dual-core, the tasks were distributed between both cores, which allows having two tasks running simultaneously. The very time-sensitive tasks (odometry, encoders), which are also run very frequently, are set on core 1, whereas the other tasks are on core 0. In this way, we ensure that the less important tasks will not add unexpected delays in the main tasks execution. A detailed description of all the tasks can be found in the documentation of the software repository.¹

9.4 State

The project relies on object-oriented programming. All the different components of the robot are abstracted in the software using objects. These objects are then nested into a top-level structure called `Robot`, which essentially represents the state of the robot. The definition of this structure is shown in Listing 9.1. An instance of this structure is defined globally, and is passed to all the functions that need to modify the state. In this way, the state is always up-to-date with the latest status of the robot.

```

1 | struct Robot {
2 |     Motor leftMotor;
3 |     Motor rightMotor;
4 |     Encoder leftEncoder;
5 |     Encoder rightEncoder;
6 |     int distances[NB_DISTANCE_SENSORS];
7 |     ImuData imuData;
8 |     RobotNavigation navigation;
9 |     Odometry odometry;
10 |    VoltageMeasurement battery;
11 |    VoltageMeasurement vcc;
12 | };

```

Listing 9.1: The Robot structure contains all of the information about the robot's state

¹The tasks' documentation can be found here: <https://github.com/opatiny/ms-software/README.md>

9.5 Quadrature Encoding and Speed Measurement

To achieve the highest possible resolution with our low-cost magnetic encoder, we implemented X4 encoding (see Section 4.5). This allows us to have 360 counts per revolution of the wheel. The encoding is achieved by using a counter for each of the encoder pins. Whenever an edge (rising or falling) is detected on either of the pins, the state of the other pin is checked, and the encoder's counter is increased or decreased accordingly. Indeed, it is the state of the second pin that indicates the direction of the rotation.

Regarding the measurement of the wheels' speed, both methods presented in Section 4.6 were implemented. Initially, we used the dc/T method, which involves dividing the number of counts measured by a fixed time period. However, since we want to update the speed as frequently as possible, this time is really short (1 ms), which means that only a few counts are measured in that interval, resulting in very low resolution. To improve our speed measurement, we used the C/dt method in which an interrupt is called every C counts. The time elapsed since the last ISR call is then used for the computation of the speed. This new method led to significantly better results, as shown in Figure 9.1. The detailed description of this experiment can be found in the annex, in section 2.

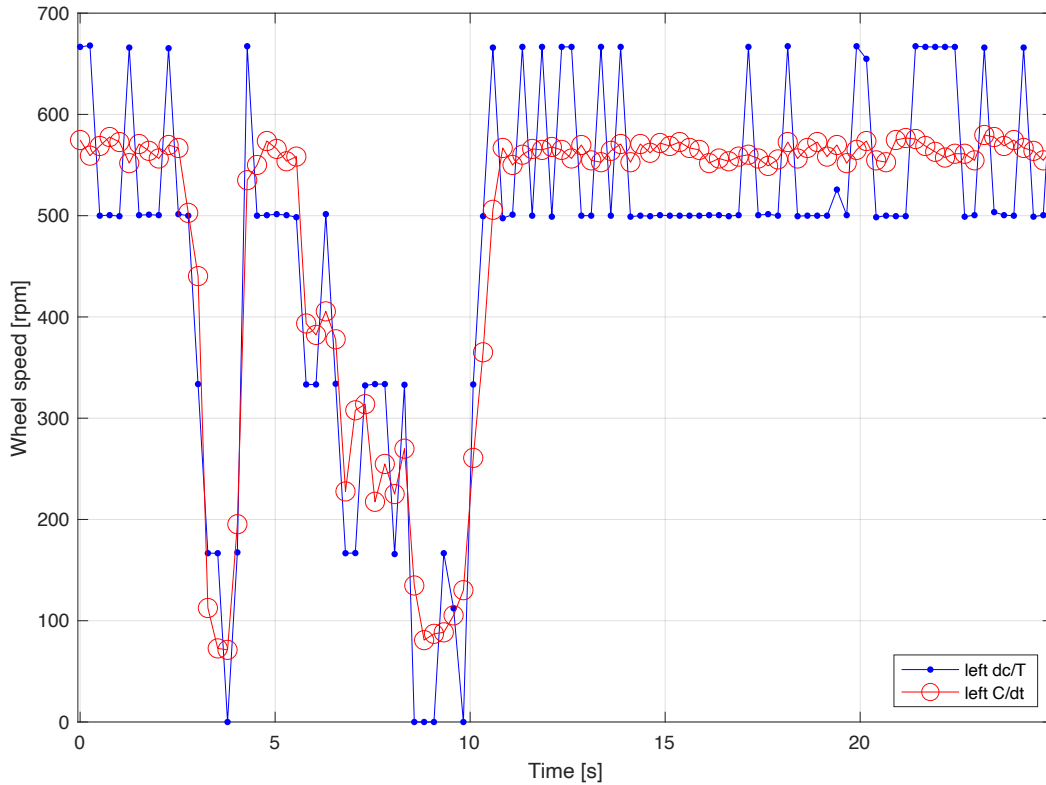


Figure 9.1: Comparison of the two methods to measure the wheels' speed. We can see that the C/dt method has better resolution. Speeds are logged every 250 ms, but are updated every 1 ms in the encoders' task.

9.6 Odometry

The odometry of our robot was implemented as described in Section 4.7. To have the best estimation of the position and orientation, the odometry algorithm needs to be run as frequently as possible. The smallest non-blocking delay that can be used with FreeRTOS is 1 ms, which is the delay we used in the odometry task.

The quality of the odometry has not been quantitatively measured. By moving the robot on the table by hand, however, we were able to observe that the position and orientation are plausible (as shown in Figure 9.2). It would be really interesting to conduct tests to measure the error accumulation over time.

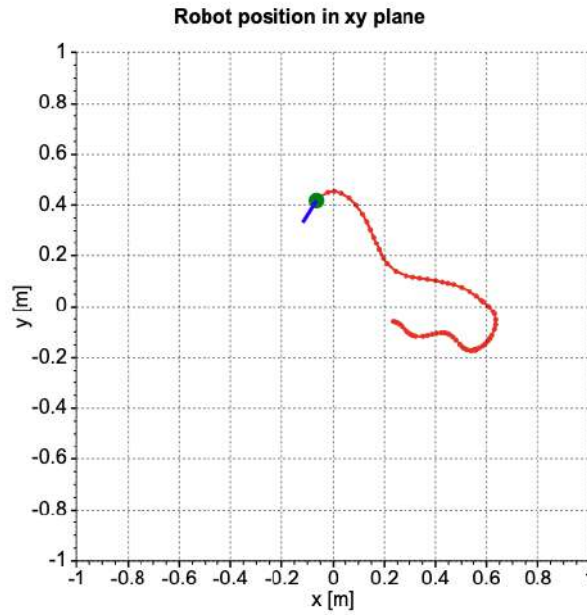


Figure 9.2: Plot of the robot's position and orientation generated by our remote debug interface. The robot was moved by hand on the table. The trace of the trajectory is displayed in red.

9.7 Distance sensors

The distance sensors come with a default I2C address, which cannot be changed permanently. Therefore, to use multiple distance sensors simultaneously, the sensors' addresses have to be modified after every shutdown of the robot. The sensors will be re-initialized after every reboot. This is done by turning only one of the devices on at a time using a special pin (called XSHUT) and changing its address. Once all the addresses are changed, normal I2C communication can be used with the sensors.

9.8 Motors' Speed Calibration

As previously discussed in Section 4.8, a fourth-degree polynomial is a suitable model for the PWM command depending on the wheel speed. It is also a simple model that can be easily implemented on a microcontroller. We used an existing polynomial regression library called `curveFitting`. The calibration can be run at any time. Once the

speed calibration starts, the motors begin spinning from the maximum backward speed to the maximum forward speed, incrementing the PWM command at regular intervals. This data is stored and once all the values are measured, the polynomial regression is computed. The resulting coefficients for each motor are stored in the `Robot` structure and can be used to compute the necessary command to achieve a given wheel speed. Figure 9.3 shows the comparison of the Matlab and C++ polynomials, which shows that the software implementation on our robot is correct. The polynomials are stored on the EEPROM, which means that they will be saved even after a device reboot. The current coefficients can be inspected at any time using the `pr` serial command. Figure 9.4 shows the motors' speed versus the expected wheel speed using the previously computed forward controller. We can see that the measured wheel speed is now fairly close to the desired speed, even by only using an open-loop controller. This implies that if both a feed-forward and a feedback controller are used, we might considerably enhance the system's response time. However, there is still a problem with the lower speeds, which are difficult to control.

Remark 2 *This entire calibration is done when the only load the motor perceives is caused by the wheels' inertia. This is therefore not representative of what will actually happen when the motor is rolling on the floor. This feedforward model might be highly improved by attaching wheels to the motors that would simulate the robot's inertia. However, we did not have the time to experiment with that.*

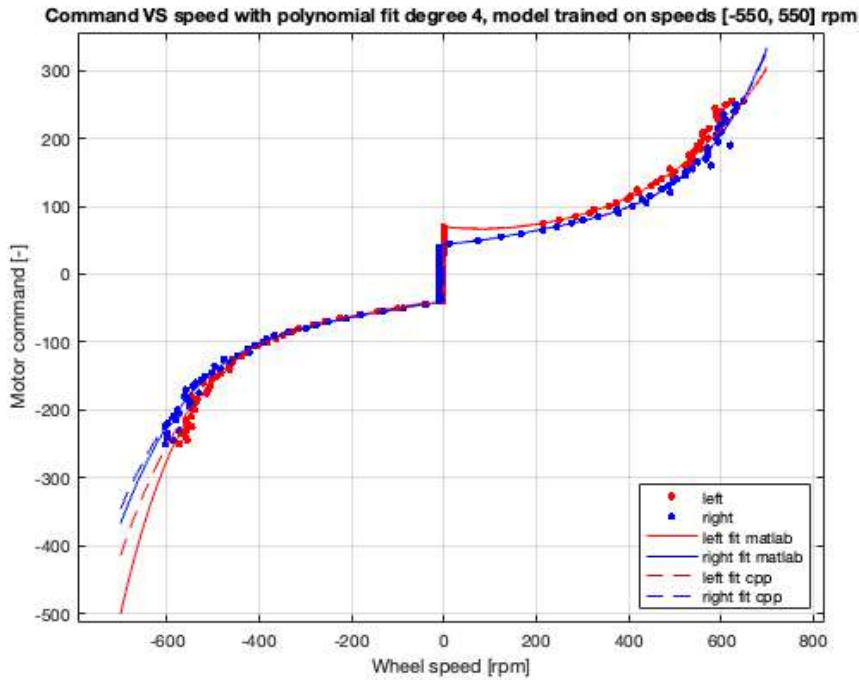


Figure 9.3: Comparison of the MatLab and C++ polynomial fits for the commands versus wheel speed. Command step: 5, time between measurements: 1 s

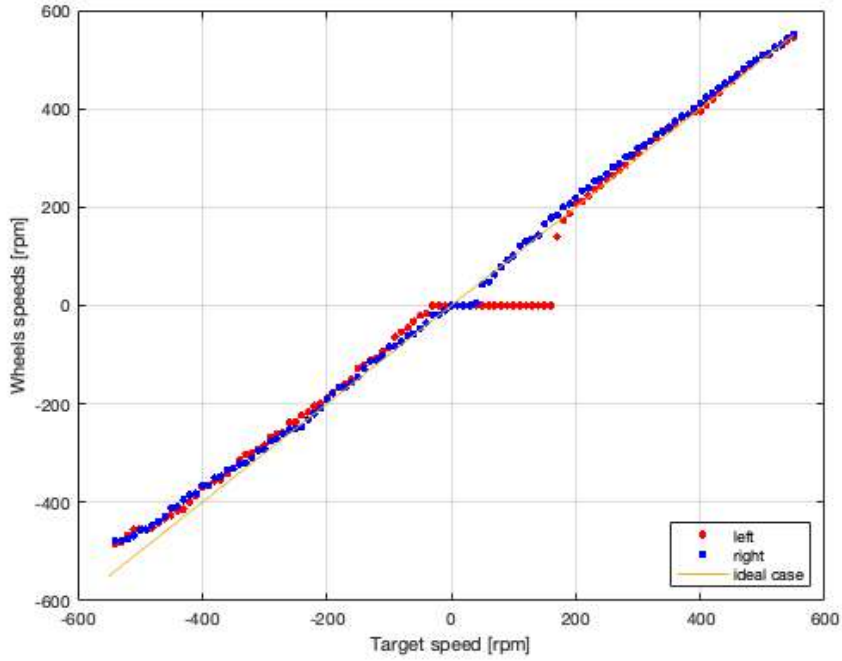


Figure 9.4: Expected VS actual wheel speed for the motors with no load, after speed calibration was done.

9.9 Motors' Control

One issue with the control of the motors is that they have to be accelerated simultaneously. Indeed, at the beginning, we would have one motor accelerate from zero to the desired speed, then the other one. This obviously results in the robot spinning on itself at the beginning and end of a movement. In order to fix that, we had to find a non-blocking approach, which allows us to apply small variations of the motors' speed at the same time, so that the two motors change speeds synchronously. This is slightly tricky because the motors don't necessarily have the same final speeds, which means that the speed "step" applied to each of them is different.

For all speed variations of our motors, we use a constant acceleration, which results in a trapezoidal speed profile. In practice, we define a desired duration T of the acceleration. This duration is constant, whatever the speed variation is. From this acceleration time, we then deduce a speed (PWM command) step s from the initial and final speeds:

$$s = \lfloor \frac{v_f - v_i}{T} \rfloor$$

Remark 3 If T is too long and results in a step of 0, a step of 1 is used.

There is a different step for each motor. Then, at every call of the motors' control function, the time Δt_{i-1} elapsed since the last call is measured. This is illustrated in Figure 9.5. This then allows us to compute the required speed Δv_i increment required for each motor:

$$\Delta v_i = s \Delta t_{i-1}$$

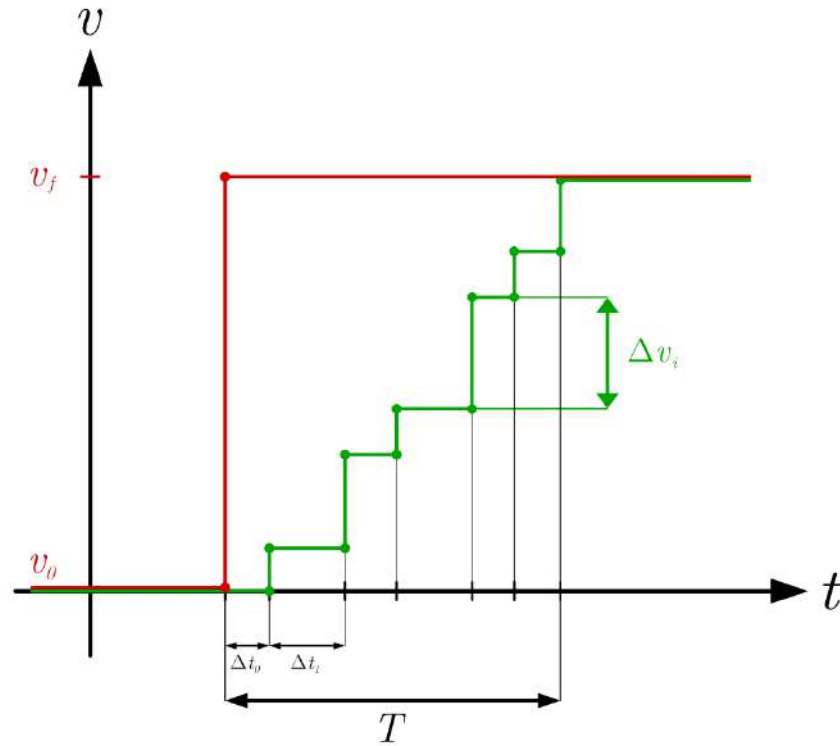


Figure 9.5: Schematic of the actual profile of the PWM command sent to the motor: the increase in speed at every function call is proportional to the time elapsed since the last call.

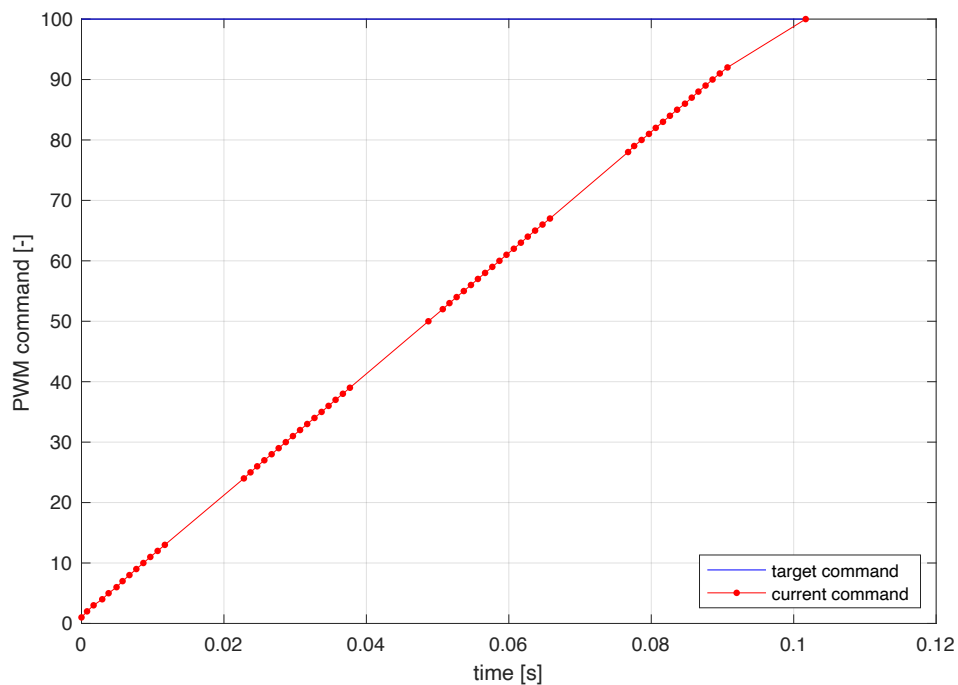


Figure 9.6: The actual PWM command sent to the motor when the target command changes from 0 to 100. We can see that despite unexpected delays between the function calls, the overall movement is executed in approximately the desired time (100 ms)

9.10 Robot's Speed Control

To implement the speed control of the robot, we first had to implement a generic PID regulator function, which can be used to handle all of our different controllers. In practice, we implemented two different speed controls:

- **Wheels' speed control:** This controller was easier to implement because it only ensures that both wheels are spinning at the same speed in [rpm]. It allows the robot to move straight at a desired speed. The same PID parameters are used for both wheels.
- **Robot's speed control:** This controller simultaneously regulates the angular and the linear velocity of the robot, allowing the setting of any desired speed, and enabling the rotation of the robot.

In the end, both controllers were kept, and the user can switch from one to another by changing the robot control mode (using the serial parameters). All controller parameters can also be modified through the serial monitor, allowing direct observation of the impact of the different parameters on the robot's behavior. To analyze the response of our system to abrupt variations of the command, we plotted the target speed, the wheel speeds, and the PWM commands. Figure 9.7 shows one of these tests. In this figure, we can see that the controller's response is slow, there is a lot of overshoot, and some oscillation. These issues could be improved by enhancing the PID parameters.

Remark 4 *The performance of the controllers was not optimized at all, and no analytical approach was used to find appropriate PID parameters. For now, the parameters were modified qualitatively to achieve seemingly good behavior.*

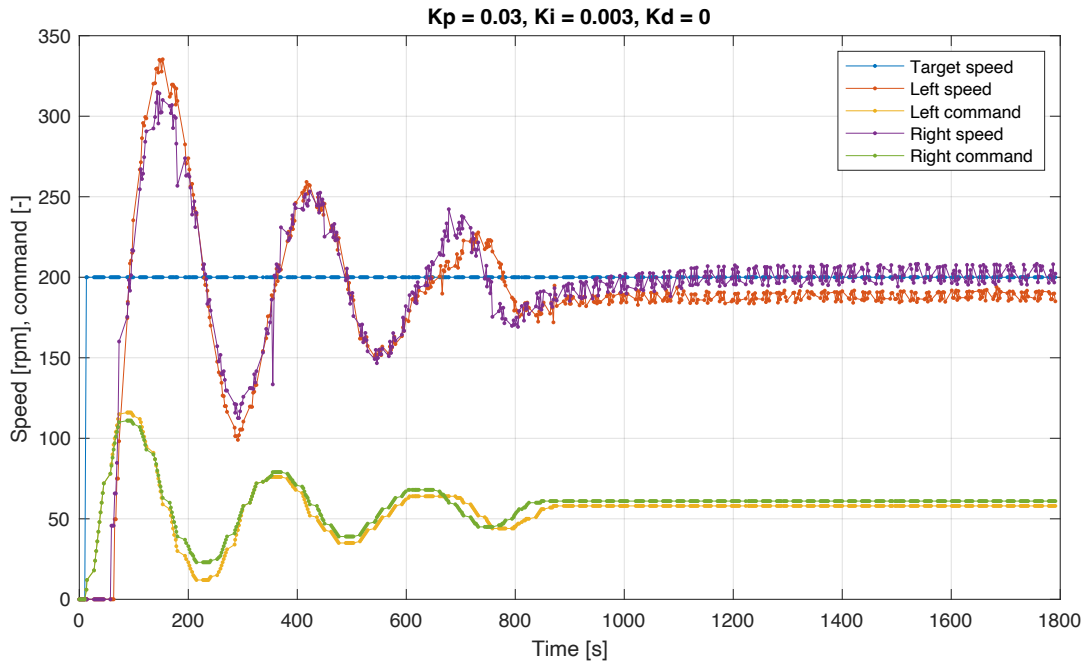


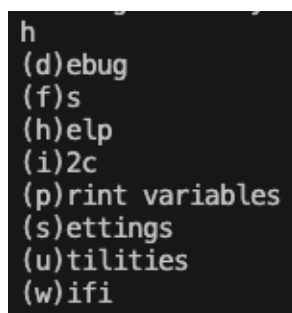
Figure 9.7: Wheel speeds over time following a sudden variation of target speed (0 to 200 rpm). In this case, the same PID parameters are used for both wheels, and the robot is in wheel speed control mode.

9.11 Serial Interface

We developed multiple debug tools that use the serial monitor and allow the user to modify parameters of the robot. The basic principle for this interface was reused from the `Hackuarium/esp32-c3` project, as mentioned in Section 9.2. This system allows us to store variables permanently in the EEPROM, as well as read and modify them through the serial monitor, which is extremely practical while developing software. Most of the important variables of our robot are stored in these parameters, and all tasks can read them and write to them as well by using the corresponding uppercase letter.

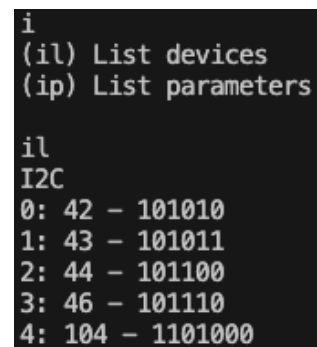
Apart from the parameters, a menu was implemented, which can be accessed using lowercase commands (an example is shown in Figures 9.8 and 9.9). This menu allows a quick overview of the robot's status. For instance, the current WiFi status can be checked, as well as all of the I2C devices currently connected. A detailed description of the serial interface can be found in the software repository: `opatiny/ms-software`.¹

Finally, a debug system was developed, which allows logging data for each individual task. For instance, all of the distances measured by the distance sensors can be printed at regular intervals to the serial monitor, thus allowing verification that everything works as expected.



```
h
(d)ebug
(f)s
(h)elp
(i)2c
(p)rint variables
(s)ettings
(u)tilities
(w)ifi
```

Figure 9.8: Result of running the command "h" in the serial monitor



```
i
(il) List devices
(ip) List parameters

il
I2C
0: 42 - 101010
1: 43 - 101011
2: 44 - 101100
3: 46 - 101110
4: 104 - 1101000
```

Figure 9.9: Details of the I2C tools, "il" can be used to list all devices that are detected

9.12 WiFi Debug Interface

To facilitate the debug of the robot's software, a remote WiFi interface was developed. The page was initially programmed using React and TypeScript, which can be served locally. Furthermore, the code can be recompiled on the fly whenever a modification is made, which is very practical. All of the original code for the debug page is contained in its own repository.² Once the web page was finished, the software was built into a plain JavaScript file and uploaded to the microcontroller. The microcontroller is configured to connect automatically to an existing WiFi, the name and password of which can be set up using the serial parameters. Once the connection is established, the microcontroller serves the debug page on its IP address. To access the page, one can simply type the

¹List of all the serial parameters available on the robot: <https://github.com/opatiny/ms-software/main/docs/serialParameters.md>

²Repository for the web page development: <https://github.com/opatiny/ms-webpage>

device's IP address in a web browser (as long as they are connected to the same WiFi). The microcontroller publishes events regularly with the new robot's state. Currently, the delay between each event is set to 100 ms. Indeed, the processing and sending of the data require a lot of processing power, so we cannot do it too frequently. Finally, the data is then stored and processed on the client's side, and finally displayed in tables and plots. Figures 9.10 to 9.14 are screenshots of the final web page.

Algernon debug page



Figure 9.10: At the top of the page, an input field allows typing in serial commands, which will be sent to the robot. This allows modification of all of the robot's variables. The result of the command is displayed in the textbox.

Distance sensors

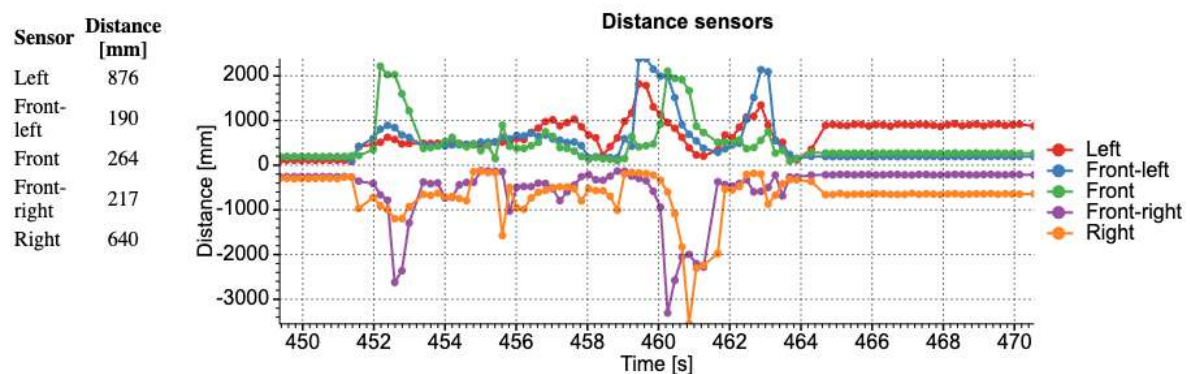


Figure 9.11: Data and plot of the distance sensors measurements.

Robot control mode (T)

Current robot mode: Stop

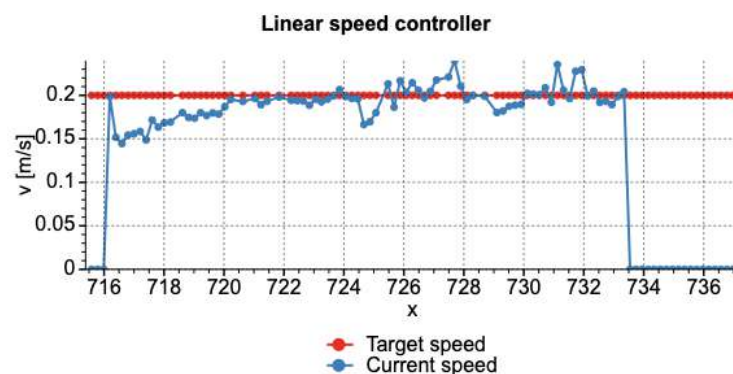
Select mode	Set speeds and other params	Unit	Serial param
Stop			
Same PWM	0 50 100 150	[-]	P
Wheels speed control	0 100 300 500 1200	[rpm]	Q
Robot speed control	0 0.1 0.3 0.5	[m/s]	R
	0 45 90 180	[deg/s]	S
Stop when obstacle	0 100 300 500	[rpm]	Q
	0 100 200 300	[mm]	V (distance)

Figure 9.12: Buttons allowing to set the robot's control mode and the values of the corresponding variables.

Robot speed controllers

Linear speed

Parameter	Value	Set value
k _p	10.000	0 10 20 30
k _i	4.000	0 2 4 10
k _d	1.000	0 1 5 10
Target value	0.200	0 0.1 0.2 0.3 0.4
Current value	0.000	
Mode	Enabled	Off On



Angular speed

Parameter	Value	Set value
k _p	3.000	0 1 3 10
k _i	0.100	0 0.1 0.2 1.0
k _d	0.050	0 0.01 0.05 0.1
Target value	0.000	0 45 90 180
Current value	0.000	
Mode	Enabled	Off On

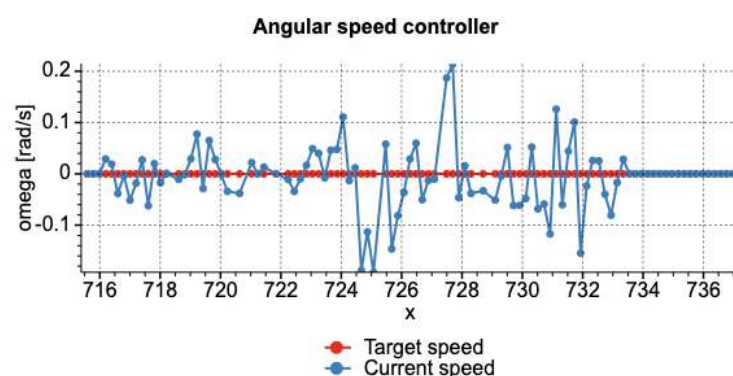


Figure 9.13: Plots of the robot's linear and angular speeds when the robot is in "robot speed control" mode, and the target speed is $v = 0.2 \text{ m/s}$ and $\omega = 0 \text{ rad/s}$.

Odometry

<input type="button" value="Reset odometry"/>	
Variable	Value
x [m]	-0.385
y [m]	0.472
theta [rad]	1.770

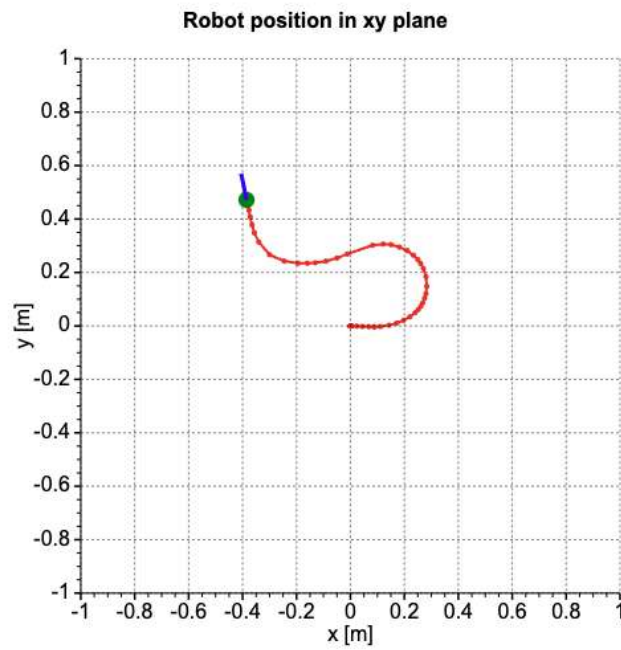


Figure 9.14: The odometry plot, when the robot is moved by hand on the table.

Chapter 10

Results

The following sections summarize all the results that have been achieved throughout this project. Firstly, the hardware and software of the robot are discussed. The maximum speed and acceleration of the robot are then analyzed. Finally, we present the robot's price, as well as its autonomy.

10.1 Hardware

Throughout this project, functional micromouse hardware has been successfully designed, assembled, and tested. The final robot is based on an ESP32-S3 microcontroller. It uses a differential drive with two BDC micromotors with magnetic encoders. The motors are controlled using a dual motor H-bridge driver. Moreover, the robot is equipped with five ToF distance sensors, as well as an IMU. The robot also integrates a buzzer for sound feedback. Finally, it is powered using a 2S1P LiPo battery. The robot's shape is an octagon of 11x11cm, and has an overall height of 32mm. The micromouse has four points of contact with the ground: it has two wheels with silicone tires (diameter of 32mm), and two small caster wheels. It was additionally equipped with soft, 3D printed bumpers in the front and the back, which protect the robot in case of impacts. Figures 10.1 and 10.2 show pictures of the final version of the robot.

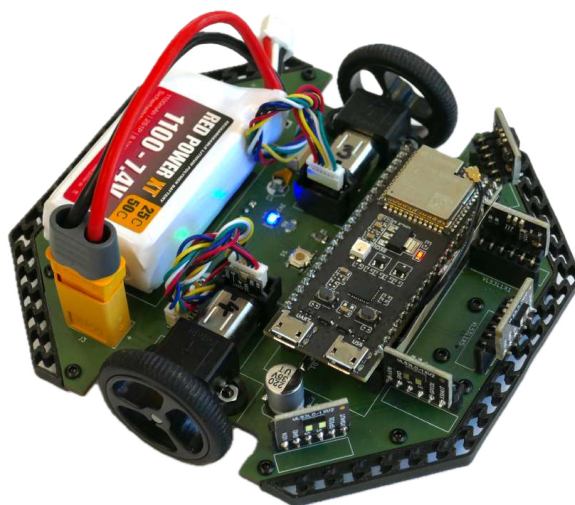


Figure 10.1: The final Algernon micromouse robot v. 1.1.0

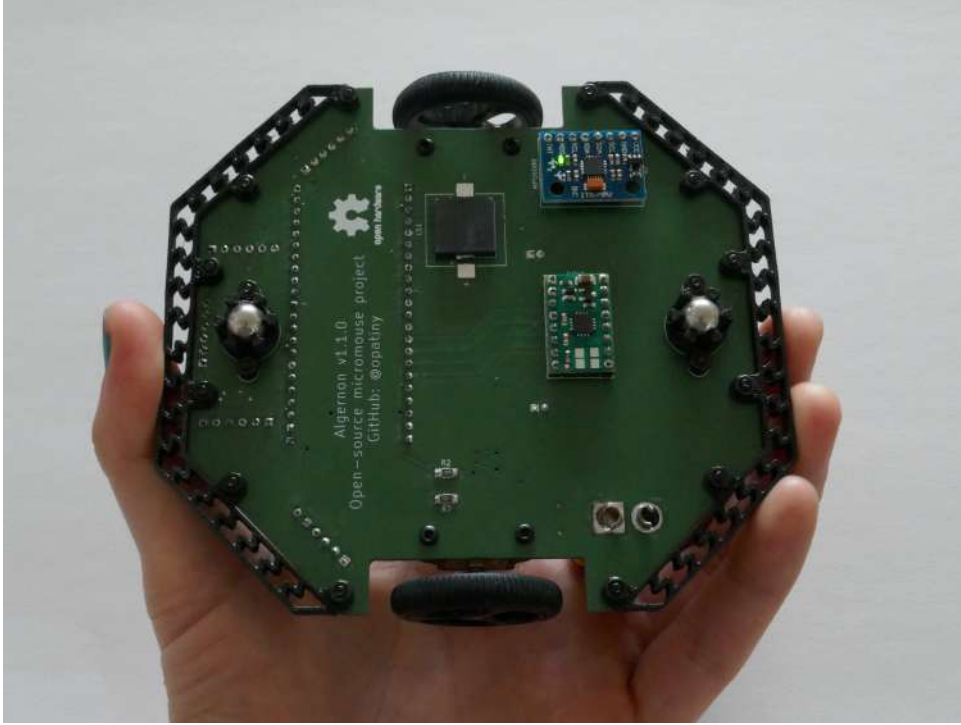


Figure 10.2: Bottom view of the final robot

10.2 Software

The programming of our robot was composed of two main parts: the software of the microcontroller itself and the development of the debug web page.

Firstly, we tested all of the individual components of the PCB and showed that all of the sensors and actuators work as expected, demonstrating that our PCB design does not have any major bugs. We used a multitasking approach based on FreeRTOS, which allows us to have multiple programs run in parallel on the microcontroller. In total, we have 15 tasks running simultaneously. The final robot's software has the following features:

- **Distance sensors:** The data of the five distance sensors is retrieved through I2C and stored in the serial parameters.
- **IMU:** Allows us to retrieve both the accelerations and the angular velocities for the three spatial dimensions.
- **Speed measurement:** The speed of the wheels in [rpm], as well as the angular and linear speeds of the robot are stored in the serial parameters.
- **Odometry:** X4 encoding was implemented to process the encoders' data, which allows us to have 360 counts per revolution of the wheels. This corresponds to a resolution of 0.3 mm. The encoders data is then used to compute the robot's position and orientation.
- **Speed control:** PID controllers, whose parameters can be easily modified through the serial interface, were implemented in order to control the linear and angular speeds of the robot.

- **RGB LED:** The on-board RGB LED of the ESP32-S3 is used to display information about the robot's state. The color and brightness can also be changed manually.
- **Buzzer:** The buzzer can be activated at any time through the serial monitor. It is also used to inform the user when the robot boots.

The user can interact with the robot to read and write variables through the serial monitor, which allows for easy testing of functionalities.

Unfortunately, we did not have the time to go as far as planned with the software of the robot. Indeed, the wall following could not be implemented, and we were very far from actually solving a maze. The perspectives for the software are, however, presented in detail in Section 10.8.

The second part of the software was the development of a custom debug web page for our robot. The final page allows for the display of real-time plots of the most important robot's data, as well as to modify the robot's state remotely. This is very practical for tests where the robot can't be connected to the computer and the serial monitor cannot be used.

10.3 Maximal Speed and Acceleration

The details of this experiment are included in the appendix, in Section 1.

The robot rolling on the floor with both motors' duty cycles set to 100 % reaches a maximal speed of about 1.05 m/s (equivalent to 630 rpm). On the other hand, the robot rolling on the floor has an average maximal acceleration of 1.5 m/s². This performance can now be compared to the initial requirements that we wanted for our robot. As presented in Section 4.3, we desired a maximal robot's speed of 1.5 m/s, and an acceleration of $0.5 \cdot g$ m/s² (4.9 m/s²). Starting with the speed, we can see that the maximal speed that can be reached by our robot is about a third less than what was required. On the other hand, the acceleration of our robot is only a quarter of our initial target.

Finally, the robot requires 35 cm to accelerate from zero to its maximal speed. This means that the robot will only have reached its maximal speed after 35 cm, which corresponds to a bit less than two maze cells. Increasing the robot's acceleration would allow us to reduce this distance and therefore reduce the time for the robot to solve the maze. Indeed, a small acceleration implies that if the maze has lots of short segments, the robot will hardly ever reach its maximal speed.

10.4 Price

Our micromouse can be built for around €170. This is slightly over our target price of €150 that we set in section 2.1.

We can see that the most expensive component of the robot is the motors with the encoders. This is expected, and would be a cost that would be difficult to cut down. On the other hand, the second biggest expense is due to the distance sensors, accounting for 27% of the robot's price. This is due to our choice of using ToF distance sensors, which are tens of times more expensive than basic IR sensors. The overall price of the robot could therefore be highly reduced by changing the technology for IR sensors, which are the most commonly used in micromouse projects. Since this project is a research project, we found it interesting to see how a micromouse would perform with this type of sensors, knowing that they are probably not the most optimal solution. However, when compared to IR sensors, as stated in Chapter 3, ToF sensors have the advantages of a bigger range, independence from ambient lighting conditions, and linearity.

Item	Price per unit [€]	Nb. units [-]	Total [€]
Motor + encoder	28.5	2	57.0
Micro-controller	14.0	1	14.0
Motor bracket pair	3.0	1	3.0
Battery	11.5	1	11.5
Wheel pair	4.0	1	4.0
PCB	16.6	1	16.6
Motor driver	10.0	1	10.0
3D printed parts	1.0	1	1.0
Distance sensor	9.1	5	45.5
Accelerometer	3.0	1	3.0
Buzzer	1.2	1	1.2
Other small components	0.05	15	0.8
Total price			168

Table 10.1: Breakout of the robot's price, this excludes all shipping costs.

10.5 Autonomy

In order to estimate the robot's autonomy, we first researched the current consumption of all the different components. These values are listed in Table 10.2. For each component, we looked for the consumption in standby, as well as the maximal current that could be drawn. This allows us to calculate a minimum and a maximum autonomy for the robot. We get a minimal current consumption of 350 mA and a maximal consumption of 1.8 A. Since we have a battery with a capacity of 1100 mAh, this corresponds respectively to an autonomy of 3 h and 40 min. This is amply sufficient for a micromouse, since the access time to the maze in competition is only 10 min per team [51].

Component	Standby current [mA]	Max. current [mA]	Nb. items [-]
ESP32	100	300	1
Motors	70	710	2
Distance sensors	18	18	5
Accelerometer	4	4	1
Encoders sensor	3.5	3.5	4
Motors driver	1.7	3	1

Table 10.2: Current consumption of the different components of the robot in standby mode and at maximal power.

10.6 Review of the Project Planning

The initial planning for this project was introduced in Section 1.4. In this section, we will review how the project has actually progressed. We can distinguish two main phases throughout the duration of the project. The first part is from the start of the project until the milestone of having finished the hardware, at the beginning of June. During all this time, the generic advancement of the project was relatively in line with the planning, despite that the real progress was much less linear than planned, and multiple tasks were actually done in parallel. During this phase, we were overall in a range of one week too early or too late on the planning depending on the tasks. The main objective of finishing the hardware by the end of June has been successfully reached.

This can be explained by the fact that we already had some previous experience in robot conception and in PCB design, which made it easier to approximate the time that would be required by the different tasks.

During the second part of the project, however, delay started accumulating. Indeed, this was the software part, which presented challenges that we had never tackled before, such as the encoders, the odometry, and the speed regulation. We still managed to successfully make each independent element of the robot work, but there was no time left to put it all together, and we weren't even close to navigating autonomously through a maze. One of the tasks that has been hugely underestimated has been how difficult it would be to just make the robot drive straight, because this implied the implementation of PID regulators, which took us quite some time. Despite that, we still managed to implement a good prototype for the debug of the robot, which would make further development easier. The fact that the thesis has been done abroad also resulted in unexpected delays. Indeed, it was a lot more difficult to access laboratory equipment, or ask specialists in certain fields, which we would have known back home.

All in all, we have been highly too ambitious for the software part of the project, because multiple aspects weren't known very well, which led to their complexity being underestimated. It was, however, interesting to actually realize that some things that may be straightforward to the inexperienced eye actually require a lot of work and expertise to work properly.

10.7 Improvements

Overall, the robot's hardware functions as expected and does not require any significant modifications. The primary need for improvement lies in the software.

- **Fix software errors:** After rebooting the robot, we frequently encounter errors that hinder its proper functioning. The first issue pertains to the I2C sensors, and the second is an "integer division by zero" error. Despite considerable efforts to identify the sources of these errors, we have not been successful in resolving them. Addressing these issues would enhance the robot's reliability.
- **Distance sensors' measurements:** Verify the accuracy and consistency of the distance sensors' measurements, and determine the frequency at which these measurements are taken. Does this present a problem for the wall-following regulation?
- **Precision of the odometry:** Develop an effective method for measuring the accuracy of our odometry algorithm.

10.8 Perspectives

Now that we have functional micromouse hardware, the possibilities for software are virtually limitless. Here is a list of features that would be interesting to implement:

- Incorporate the wall-following controller into the current control loop.
- Learn to rotate a specified angle.
- Navigate in an unknown maze.
- Map an unknown maze.
- Implement algorithms to identify the most optimized path in a maze.
- Add the feedforward controller to our control loop to decrease the robot's response time and enhance its reactivity.
- **Sensors fusion:** Integrate distance sensors, accelerometer, and odometry data using Kalman filters to minimize the overall error in the robot's position and speed.

Chapter 11

Conclusions

Within the scope of this thesis, custom micromouse hardware was successfully developed. The robot is equipped with an ESP32-S3 microcontroller, a 2S1P LiPo battery, two BDC motors with magnetic encoders, five ToF distance sensors, an IMU, a push button, and a buzzer. All the different components were tested individually and are functioning as expected. The distance sensors return the distances in millimeters, and the IMU measures the accelerations and angular speed in the three space dimensions. The encoders' data is processed using X4 encoding and is used to compute the wheel speeds, as well as the robot's position and orientation. Moreover, a closed-loop feedback controller was implemented on the robot's speed, which allows the setting of any desired angular and linear speed. The maximum speed of our robot is around 1.05 m/s, and its maximum acceleration is 1.5 m/s². This implies that the robot travels 35 cm to accelerate from zero to its maximum speed. All of the robot's parameters can be displayed in a custom debug web interface, which features real-time plots of the robot's data. Among others, the distances, wheel speeds, and odometry are plotted, with the data being updated every 100 ms. Despite the fact that maze-solving couldn't be implemented due to a lack of time, our final hardware is a flexible and reliable platform for the development of more advanced software, which might be pursued in future projects.

Bibliography

Articles

- [1] Donald Christiansen. “The Amazing MicroMouse Contest - IEEE Spectrum”. In: *IEEE Spectrum (online)* (2014). URL: <https://spectrum.ieee.org/the-amazing-micromouse-contest> (visited on 07/13/2024).
- [2] Man Lok Fung, Michael Z.Q. Chen, and Yong Hua Chen. “Sensor fusion: A review of methods and applications”. In: *Proceedings of the 29th Chinese Control and Decision Conference, CCDC 2017* (July 2017), pp. 3853–3860. DOI: 10.1109/CCDC.2017.7979175.
- [3] Hugh H.S. Liu and Grantham K.H. Pang. “Accelerometer for mobile robot positioning”. In: *IEEE Transactions on Industry Applications* 37 (3 May 2001), pp. 812–819. ISSN: 00939994. DOI: 10.1109/28.924763.
- [4] Yu Liu et al. “A Review of Sensing Technologies for Indoor Autonomous Mobile Robots”. In: *Sensors 2024, Vol. 24, Page 1222* 24.4 (Feb. 2024), p. 1222. ISSN: 1424-8220. DOI: 10.3390/S24041222. URL: <https://www.mdpi.com/1424-8220/24/4/1222/htm%20https://www.mdpi.com/1424-8220/24/4/1222>.
- [5] Tadeusz Mikołajczyk et al. “Energy Sources of Mobile Robot Power Systems: A Systematic Review and Comparison of Efficiency”. In: *Applied Sciences 2023, Vol. 13, Page 7547* 13.13 (June 2023), p. 7547. ISSN: 2076-3417. DOI: 10.3390/APP13137547. URL: <https://www.mdpi.com/2076-3417/13/13/7547/htm%20https://www.mdpi.com/2076-3417/13/13/7547>.
- [6] Nikunj Mehta et al. “Design of HMI Based on PID Control of Temperature”. In: *International Journal of Engineering Research and V6.05* (May 2017). DOI: 10.17577/IJERTV6IS050074. URL: https://www.researchgate.net/publication/316709017_Design_of_HMI_Based_on_PID_Control_of_Temperature.
- [7] P. van Turenout, G. Honderd, and L. J. van Schelven. “Wall-following control of a mobile robot”. In: *Proceedings 1992 IEEE International Conference on Robotics and Automation* 1 (Jan. 1992), pp. 280–285. DOI: 10.1109/ROBOT.1992.220250.

Books

- [8] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005, p. 245.

Images

- [9] *Brushless Vs Brushed DC Motors: When and Why to Choose One Over the Other / Article / MPS*. URL: <https://www.monolithicpower.com/en/brushless-vs-brushed-dc-motors> (visited on 03/29/2024).
- [10] *Buy PICAXE-20X2 Microbot in India / Fab.to.Lab*. URL: <https://www.fabtolab.com/picaxe-20X2-microbot> (visited on 04/01/2024).
- [11] *Difference between optical, magnetic and capacitive encoders? / US Digital*. 2019. URL: <https://www.usdigital.com/blog/difference-between-optical-magnetic-and-capacitive-encoders/> (visited on 04/01/2024).
- [12] gycka (Instructables Username). *Micromouse IR Distance Sensor Board*. URL: <https://www.instructables.com/Micromousemobile-robot-IR-distance-sensor-board-ti/> (visited on 04/02/2024).
- [13] Peter Harrison. *Improving the Pololu Magnetic Encoder - more magnets*. Sept. 2020. URL: <https://ukmars.org/2020/09/improving-the-pololu-magnetic-encoder-more-magnets/> (visited on 07/18/2024).
- [14] *How Stepper Motors Work*. URL: <https://www.orientalmotor.com/stepper-motors/technology/stepper-motor-overview.html> (visited on 03/29/2024).
- [15] Osamu Iwasaki. *Micromouse maze image*. 2011. URL: https://en.wikipedia.org/wiki/Micromouse#/media/File:Micromouse_maze.jpg (visited on 07/30/2024).
- [16] Chinmay Lonkar. *Mushak - A half-size micromouse*. 2022. URL: https://www.pcbway.com/project/shareproject/Mushak_A_half_size_micromouse_f1889e61.html (visited on 04/01/2024).
- [17] *Micro bit Lesson: Using the Ultrasonic Module*. 2018. URL: <https://osoyoo.com/2018/09/18/micro-bit-lesson-using-the-ultrasonic-module/> (visited on 04/02/2024).
- [18] *My MicroMouse: Shi On 2005*. URL: <http://4th-laboratory.la.coocan.jp/mymouse.html> (visited on 04/01/2024).
- [19] *Overview / Adafruit VL53L1X Time of Flight Distance Sensor / Adafruit Learning System*. URL: <https://learn.adafruit.com/adafruit-vl53l1x/overview> (visited on 04/02/2024).
- [20] *Pololu - DRV8833 Dual Motor Driver Carrier*. URL: <https://www.pololu.com/product/2130> (visited on 03/20/2024).
- [21] *Quadrature Encoders - The Ultimate Guide*. URL: https://www.dynapar.com/technology/encoder_basics/quadrature_encoder/ (visited on 04/01/2024).
- [22] Jim Remington. *Micromouse/line maze platform with high res wheel encoders*. 2007. URL: <https://forum.pololu.com/t/micromouse-line-maze-platform-with-high-res-wheel-encoders/536> (visited on 04/01/2024).
- [23] Ruthu S Sanketh. *Drives. Differential Drive*. 2020. URL: <https://medium.com/manual-robotics/drives-76c2b2dac97c> (visited on 04/01/2024).
- [24] Hohann Tang. *Technical Manual Series: Brushed DC Motor Structure and Principles*. 2021. URL: <https://blog.orientalmotor.com/technical-manual-series-brushless-motor-structure-and-principles> (visited on 07/13/2024).

- [25] UK Micromouse and Robotics Society. *ukmarsbot: A simple beginners multi purpose robot platform*. URL: <https://github.com/ukmars/ukmarsbot/tree/master> (visited on 04/01/2024).
- [26] *What are coreless DC motors?* URL: <https://www.motioncontroltips.com/what-are-coreless-dc-motors/> (visited on 03/18/2024).

Datasheets

- [27] Polulu Corporation. *Micro Metal Gearmotors*. Rev 6.1. Apr. 2024. URL: www.pololu.com (visited on 07/06/2024).

Web references

- [28] Espressif Systems (Shanghai). *ESP32-S3-DevKitC-1 v1.1 latest documentation*. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html> (visited on 07/06/2024).
- [29] Derek Muller (Veritasium). *The Fastest Maze-Solving Competition On Earth*. May 2023. URL: <https://www.youtube.com/watch?v=ZMQbHMgK2rw&t=12s> (visited on 07/02/2024).
- [30] *1977-79 - "Moonlight Special" Battelle Inst. (American) - cyberneticzoo.com*. URL: <https://cyberneticzoo.com/mazesolvers/1977-79-moonlight-special-battelle-inst-american/> (visited on 05/08/2024).
- [31] *A brief introduction to Time-of-Flight sensing. Part 2*. URL: <https://www.terabee.com/a-brief-introduction-to-time-of-flight-sensing-part-2-indirect-tof-sensors/> (visited on 08/12/2024).
- [32] Evan Ackerman. *Meet the New World's Fastest Micromouse Robot - IEEE Spectrum*. 2011. URL: <https://spectrum.ieee.org/meet-the-new-worlds-fastest-micromouse> (visited on 06/07/2024).
- [33] *Benefits of DC Motors for Robotics | automate.org*. URL: <https://www.automate.org/motion-control/blogs/benefits-of-dc-motors-and-why-they-re-great-for-robotics> (visited on 03/29/2024).
- [34] *Classic Micromouse*. 2020. URL: <https://ukmars.org/contests/micromouse/> (visited on 05/08/2024).
- [35] *Dealing with Motor Noise*. URL: <https://www.pololu.com/docs/0J15/9> (visited on 07/24/2024).
- [36] *Everything You Need To Know About Stepper Motors | RS*. URL: <https://uk.rs-online.com/web/content/discovery/ideas-and-advice/stepper-motors-guide> (visited on 03/29/2024).
- [37] Wally Gastreich. *Stepper Motors Advantages and Disadvantages*. 2018. URL: <https://www.realpars.com/blog/stepper-motors-advantages> (visited on 03/29/2024).
- [38] *H-Bridge*. Jan. 2013. URL: <https://www.uni-weimar.de/kunst-und-gestaltung/wiki/H-Bridge> (visited on 07/05/2024).

- [39] Peter Harrison. *Acceleration*. URL: <https://micromouseonline.com/micromouse-book/robot-dynamics/acceleration/> (visited on 06/07/2024).
- [40] Peter Harrison. *All Japan Micromouse 2017 - Micromouse Online*. URL: <https://micromouseonline.com/2017/11/26/japan-micromouse-2017/> (visited on 04/01/2024).
- [41] Peter Harrison. *Diagonal paths for a micromouse using a state machine*. 2014. URL: <https://micromouseonline.com/2014/05/05/diagonal-paths-micromouse-using-state-machine/> (visited on 02/23/2024).
- [42] Peter Harrison. *Four wheel micromouse is difficult to turn accurately*. 2013. URL: <https://micromouseonline.com/2013/02/24/four-wheel-micromouse-difficult-to-turn/> (visited on 04/02/2024).
- [43] Peter Harrison. *InfraRed Sensors*. URL: <https://micromouseonline.com/micromouse-book/sensors/infrared-sensors/> (visited on 04/02/2024).
- [44] Peter Harrison. *More suck, less slip*. 2018. URL: <https://micromouseonline.com/2018/02/18/more-suck-less-slip/> (visited on 04/02/2024).
- [45] Peter Harrison. *Navigation*. URL: <https://micromouseonline.com/micromouse-book/navigation/> (visited on 06/07/2024).
- [46] Peter Harrison. *Processor*. URL: <https://micromouseonline.com/micromouse-book/command-and-control/processor/> (visited on 04/17/2024).
- [47] Peter Harrison. *Stepper Motors - Micromouse Online*. URL: <https://micromouseonline.com/micromouse-book/motors/stepper-motors/> (visited on 07/04/2024).
- [48] Peter Harrison. *Wall and Line Tracking for Micromouse and Linefollowers - Minos 2023*. 2023. URL: <https://www.youtube.com/watch?v=22l6MrAwN-o&t=2393s> (visited on 07/28/2024).
- [49] *How are stepper motors controlled? - Speed control of stepper motors*. Feb. 2021. URL: <https://eu.aspina-group.com/en/learning-zone/columns/what-is/017/> (visited on 07/04/2024).
- [50] *How Ultrasonic Sensors Work - MaxBotix*. URL: <https://maxbotix.com/blogs/blog/how-ultrasonic-sensors-work> (visited on 04/02/2024).
- [51] IEEE. *MicroMouse Competition Rules*. URL: https://attend.ieee.org/r2sac-2020/wp-content/uploads/sites/175/2020/01/MicroMouse_Rules_2020.pdf (visited on 07/28/2024).
- [52] *Inertial Measurement Unit (IMU) / An Introduction*. June 2024. URL: <https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/>.
- [53] *Links / Micromouse USA*. URL: http://micromouseusa.com/?page_id=75 (visited on 04/01/2024).
- [54] *Micromouse - UKMARS*. URL: <https://ukmars.org/contests/micromouse/> (visited on 04/16/2024).
- [55] B. Montanari. *How to program and debug the STM32 using the Arduino IDE*. 2023. URL: <https://community.st.com/t5/stm32-mcus/how-to-program-and-debug-the-stm32-using-the-arduino-ide/ta-p/608514> (visited on 04/17/2024).

- [56] *NI Hardware Encoder Measurements: How-To Guide*. May 2024. URL: <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000x1riCAA&l=en-US> (visited on 07/18/2024).
- [57] Peter Harrison. *DC Motors - Micromouse Online*. URL: <https://micromouseonline.com/micromouse-book/motors/dc-motors/> (visited on 03/29/2024).
- [58] *PlatformIO*. URL: <https://platformio.org/> (visited on 08/05/2024).
- [59] *RTOS Fundamentals*. Aug. 2024. URL: <https://www.freertos.org/Documentation/01-FreeRTOS-quick-start/01-Beginners-guide/01-RTOS-fundamentals> (visited on 08/06/2024).
- [60] *Small Ball Caster Wheel - 12mm Diameter Metal - BC Robotics*. URL: <https://bc-robotics.com/shop/small-ball-caster-wheel-metal/> (visited on 08/13/2024).
- [61] *Super small non-contact rotary encoder RM08 - www.rls.si*. URL: <https://www.rls.si/eng/rm08-super-small-non-contact-rotary-encoder> (visited on 04/01/2024).
- [62] Aleksandra Szczepaniak. *Raspberry Pi or Arduino - when to choose which?* 2023. URL: <https://www.leorover.tech/post/raspberry-pi-or-arduino-when-to-choose-which> (visited on 04/16/2024).
- [63] *Time of Flight (ToF) Sensors - FlightSense - STMicroelectronics*. URL: <https://www.st.com/en/imaging-and-photonics-solutions/time-of-flight-sensors.html> (visited on 04/02/2024).
- [64] Nick (DeepSea username). *The ESP32 Chip explained: Advantages and Applications*. URL: <https://www.deepseadev.com/en/blog/esp32-chip-explained-and-advantages/> (visited on 06/07/2024).
- [65] Green Ye. *Micromouse 2016-2017*. URL: <https://greenye.net/> (visited on 07/24/2024).
- [66] Khiew Tzong Yong. *My ProjectQ - Excel-9a*. URL: <https://sites.google.com/site/myprojectq/robotic/classic-micromouse/excel-9a> (visited on 04/02/2024).
- [67] Gan Zhen, Dae-Ki Ye, and Kang. *Autonomous Maze Solving Robot*. 2011. URL: <https://api.semanticscholar.org/CorpusID:219964670>.

Other references

- [68] Lee Gim Hee and Marcelo H. Ang Jr. “Mobile Robots Navigation, Mapping, and Localization Part II”. In: IGI Global, May 2011, pp. 1080–1088. DOI: 10.4018/978-1-59904-849-9.CH159.
- [69] Antonio Valente, Salviano Soares, and F.S. Pereira. “Micromouse portuguese contest: A didactic robotic project”. In: *International Conference on Innovative Technologies 2014*. 2014. URL: https://www.academia.edu/10289832/MICROMOUSE_PORTUGUESE_CONTEST_A_DIDACTIC_ROBOTIC_PROJECT.

Appendix

1 Experiment: Maximal Speed and Acceleration

1.1 Objectives

Measure the maximal speed of the robot, the maximal acceleration of the robot to reach that speed, and the distance traveled by the robot to accelerate from an initial null speed to its maximal speed.

1.2 Procedure

Speed

In order to measure the maximal speed of our motors, we used the "wheel speed control" mode, which allows to set a target speed in rpm for both wheels. That mode used a PID regulation of the wheels speed. The robot is placed on the floor, and the target speed is then set to 1200 rpm, which is highly above the actual maximal speed that the robot can reach. This implies that the duty cycle of both wheels is saturated to 100 %. We then used the remote web interface to record the maximal speed that the robot reached.

Acceleration

In order to estimate the maximal acceleration of our robot, we programmed a new feature directly onto the microcontroller. The function measures the time dt elapsed for the robot to reach 95 % of the maximal speed v_{max} that we computed previously. Since the speed is noisy, we additionally use a first order filter to smooth it. The theory for this filter is described in Equation 4.9.1. For this experiment, we implemented $\alpha = 0.9$. The average acceleration a can then be computed with the following formula:

$$a = \frac{v_{max}}{dt}$$

Since the measurement of the time interval highly influences the acceleration, we realized eight measurements, and took the average.

Distance

With the maximal acceleration a of the robot, we can compute the minimum distance required to get from a speed of zero to the maximal speed. In order to find that distance Δx , we simply isolate it in the formula below:

$$v_{max}^2 = v_0^2 + 2a\Delta x$$

Which leads to

$$\Delta x = \frac{v_{max}^2 - v_0^2}{2a}$$

1.3 Results

The following results were obtained:

- Maximal speed: 1.05 m/s
- Maximal average acceleration: 1.5 m/s²
- Distance to accelerate from zero to maximal speed: 35 cm

2 Experiment: Measurement of wheels speed

2.1 Objectives

Compare the two methods that were presented in section 4.6 to measure the wheels' speed.

2.2 Procedure

As described in section 4.6, they are two possibilities to deduce the motor's speed from the encoders data:

- Count the number of ticks of the encoder since the last function call. Then divide this number of counts by the time elapsed. We call this the $\Delta c/T$ method.
- Measure the time for a given number of counts to be reached. Divide that fixed number of counts by the time elapsed. We call this the $C/\Delta t$ method.

Both methods were implemented in the software. The $\Delta c/T$ method is implemented directly in the odometry task, and the speed is updated every time the task is called. The second method is implemented in the encoders' task, inside of the interrupt routines. We use a default number of counts of 2. A debug tool was implemented which allows to print both speeds at given time intervals. The data can be visualized using the **A15** serial command.¹ The serial monitor's data is then saved in a `.csv` file, and plotted in MatLab.

2.3 Results

Figure 11.1 compares both methods to measure the speed of the wheel. We can see that the $C/\Delta t$ method has a higher resolution than the $\Delta c/T$ method. Indeed, the $\Delta c/T$ speed varies with steps of 167rpm, whereas the other speed has a resolution of a few rpm. We therefore use the $C/\Delta t$ speed in all of the other tasks.

¹Using the following version of the software: <https://github.com/opatiny/ms-software> (commit d6f4fc6eaca07a7fbeb5fb862d9489f111c2e57a)

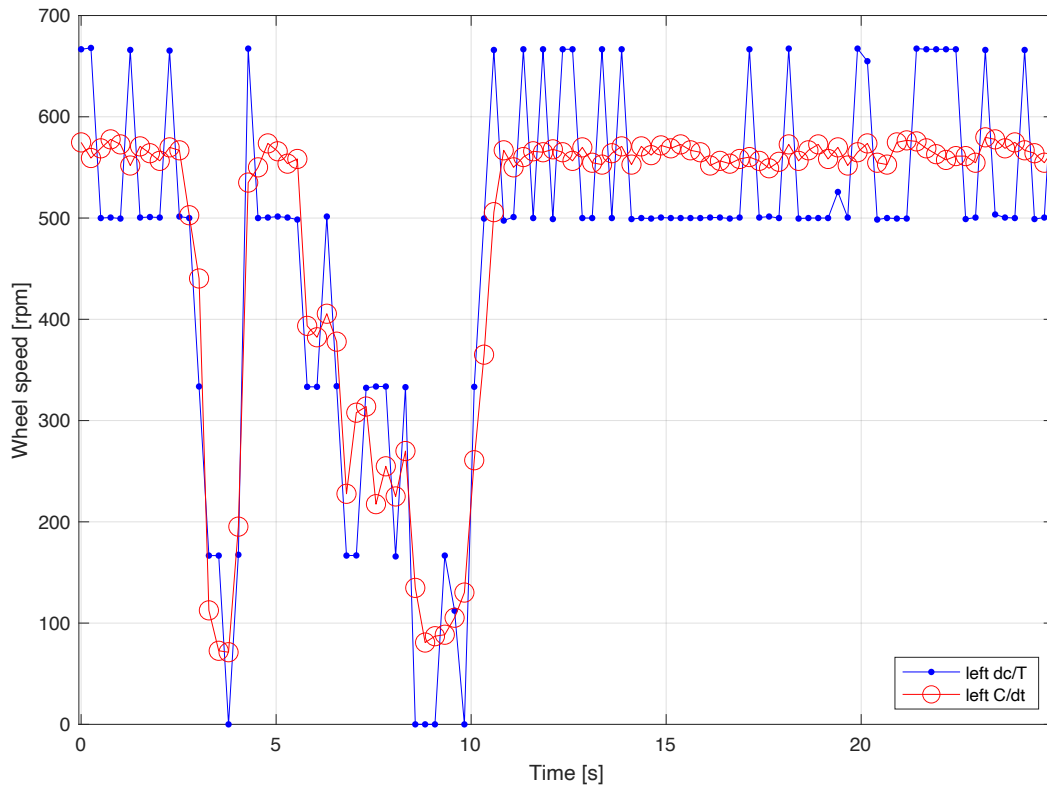


Figure 11.1: Comparison of the two methods to measure the wheels speed. We can see that the C/dt method has a better resolution. Speeds are logged every 250 ms, but are updated every 1 ms in the encoders' task.

3 Experiment: Motor's speed vs voltage

3.1 Objectives

Measure the angular speed of the motor for DC voltages in the range 0 to 12 V. Do the measurements with and without a wheel on the motor's shaft, in order to verify whether the air friction has an impact on the motor's speed. Show that for a BDC with no load, the speed is directly proportional to the applied DC voltage.

3.2 Materials

The following materials were used for this experiment:

- 1 breadboard and jumper cables
- 1 ESP32-S3 Xiao microcontroller
- 1 PeakTech DC power supply 6080 + banana cables
- 1 30:1 Micro Metal Gearmotor HPCB 12V with encoder

3.3 Procedure

Figure 11.3 shows how the different elements are connected. In this experiment, a DC power supply is used in order to apply a constant voltage on the DC motor. The power supply is connected between the two terminals of the motors, and the voltage is varied from 0 to 12 V with 1 V steps. On the other hand, the frequency of the encoder of the motor is measured using a small oscilloscope (oscilloscope mode of the MT8208 multimeter). In order to measure the frequency, the oscilloscope is connected to the ground (GND) and to one of the output channels of the encoder (Data). The encoders are powered by the 3.3 V generated by an ESP32-S3. Another DC power supply could have been used instead, but we didn't have one. A picture of the setup is shown in Figure 11.2. Finally, the frequency f measured on the oscilloscope is converted to the motor's speed using the following formula:

$$n = \frac{f \cdot 60}{N_{X1}} \quad [\text{rpm}] \quad (11.1)$$

where N_{X1} is the number of pulses of the encoder per rotation of the wheel in X1 mode. In our case, we have $N_{X1} = 90 \text{ CPR}$.

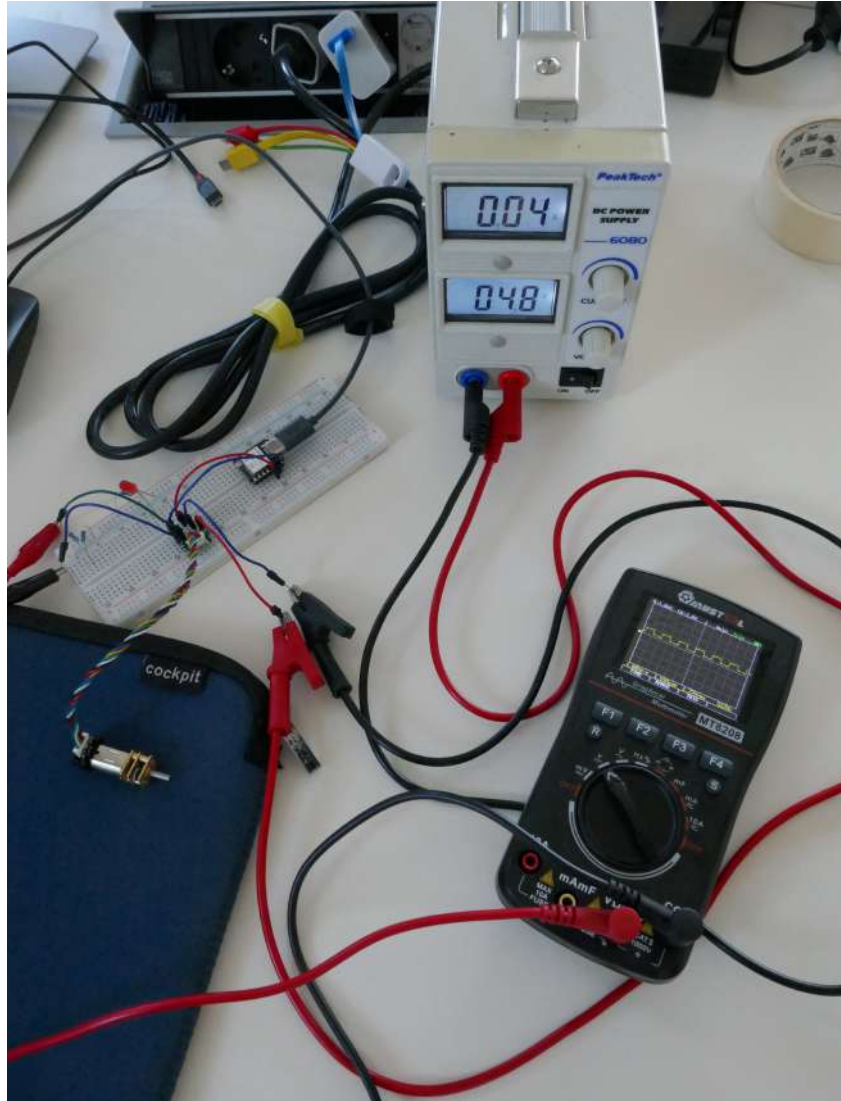


Figure 11.2: Test setup to measure the motor's speed with no load depending on input voltage

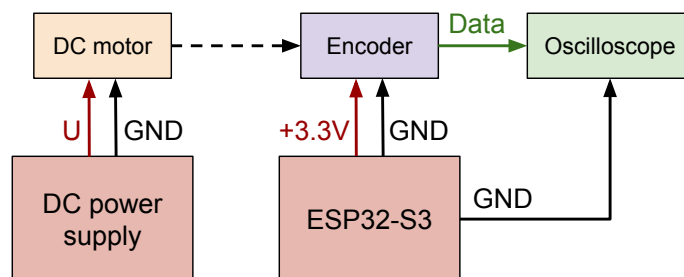


Figure 11.3: Connections diagram for the experiment.

3.4 Results

Tables 11.2 and 11.1 show all of the measured frequencies for the different input voltages for a motor without a wheel and with a wheel. The tables also show the corresponding wheel speeds, computed with Equation 11.1. The speed depending on the input voltage were plotted in Figure 11.4. The plot clearly shows that the motor's speed is linear with the input voltage, as expected. Moreover, the presence of the wheel on the motor has little to no impact on the final speed.

Voltage [V]	Frequency [Hz]	Speed [rpm]
0.0	0	0
1.0	0	0
2.0	138	92
3.0	270	180
4.0	400	267
5.0	526	351
6.0	675	450
7.0	815	543
8.0	975	650
9.0	1070	713
10.0	1220	813
11.0	1380	920
12.0	1530	1020

Table 11.1: Motor without wheel: Frequency measured on the oscilloscope and corresponding wheel speed depending on the input voltage

Voltage [V]	Frequency [Hz]	Speed [rpm]
0.0	0	0
1.0	0	0
2.0	142	95
3.0	270	180
4.0	410	273
5.0	545	363
6.0	673	449
7.0	815	543
8.0	958	639
9.0	1100	733
10.0	1240	827
11.0	1370	913
12.0	1540	1027

Table 11.2: Motor with wheel: Frequency measured on the oscilloscope and corresponding wheel speed depending on the input voltage

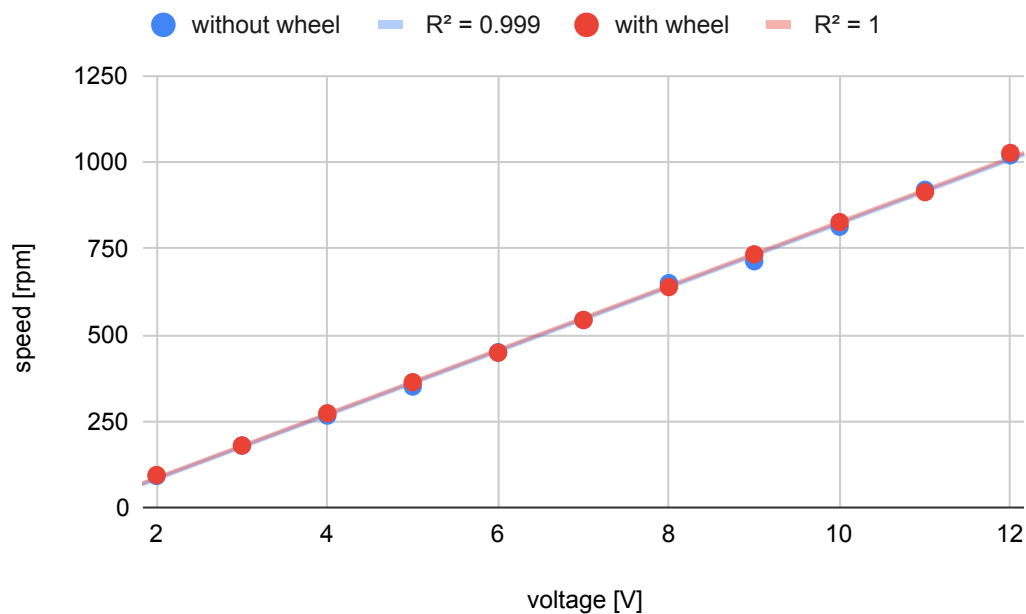


Figure 11.4: Motor speed with no load depending on input voltage, with and without wheel.

4 PCB schematics and project planning

The subsequent pages first showcase the electrical schematic of our micromouse's PCB (Figure 11.5), followed by the initial project planning (see Figures 11.6, 11.7 and 11.8), which was realized in February 2024.

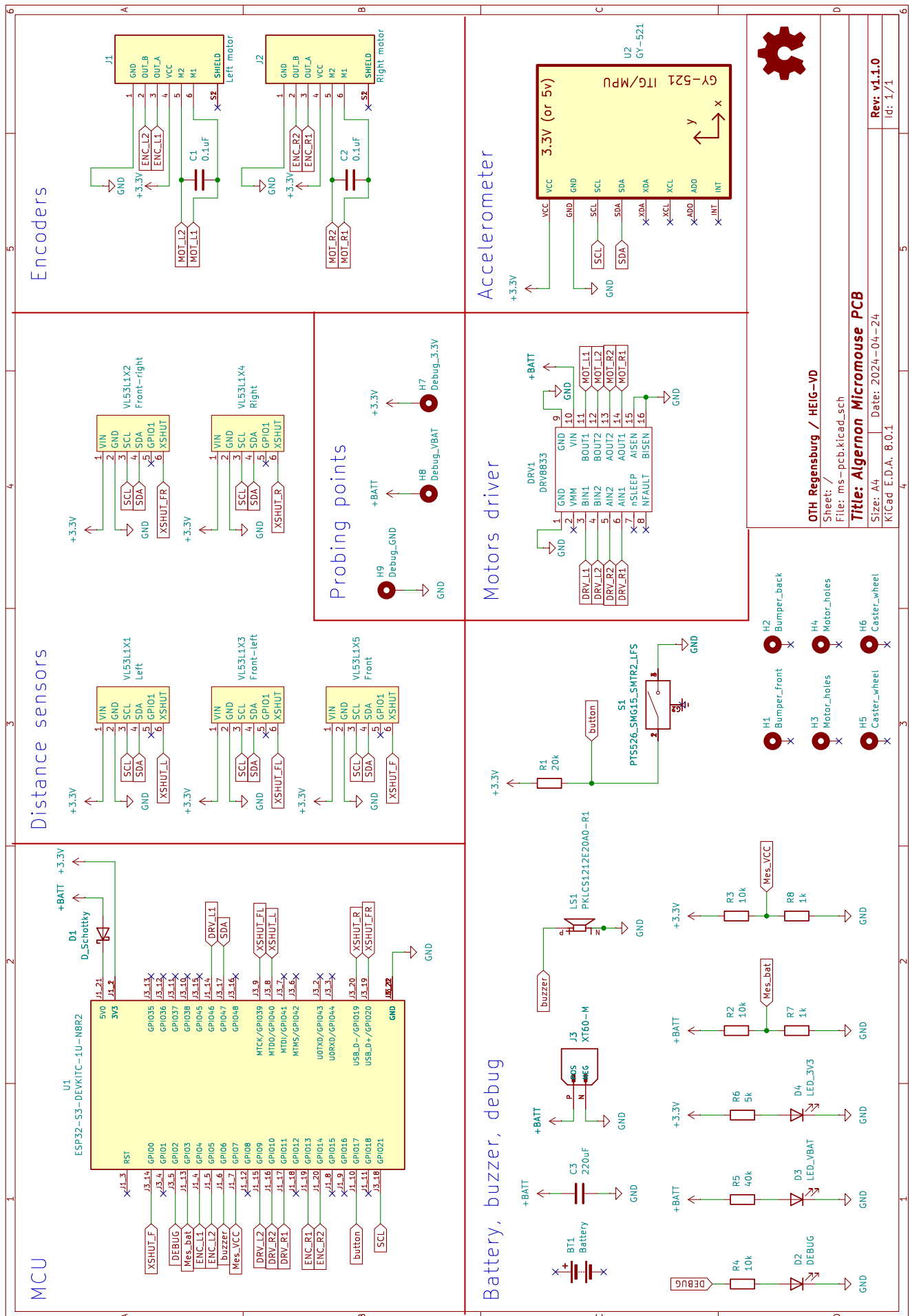


Figure 11.5: Final electric schematic of our micromouse

Tasks			February		March			
	Rough estimation nb hours	Total hours per task	19 - 25	26 - 3	4 - 10	11 - 17	18 - 24	25 - 31
Full micromouse hardware and software								
Comments / Milestones								
Project start	70							
State of the art	40	40	20	10		10		
Specifications	10	10		8			2	
Planning	5	5		5				
Architecture choice	15	15	5	5			5	
Hardware	154							
Components choice	15	15		10			2	3
Order components	5	4		1			1	1
PCB v1 design v1	30	30						
PCB v1 order	2	2						
PCB v1 assembly	15	15						
PCB v1 test	15	15						
PCB correction for v2	15	15						
PCB v2 order	2	2						
PCB v2 assembly	5	5						
PCB v2 test	5	5						
CAD design of mechanical parts	15	15						
3D printing	10	9						
Full mouse assembly	20	20						
Software	215							
Micro-controller choice	5	5					5	
IDE setup	15	15					5	10
First programs on MCU	15	15						8
Test distance sensors	10	10						
Test motors	15	15						
Drive in straight corridor	30	30						
Bluetooth/wifi setup and tests	10	10						
Debug system (retrieve params by wifi)	15	14						
Turn 90 and 180°	20	20						
Follow predefined path in maze	30	30						
Implement flood fill to find good path	30	30						
Final tests and enhancements	20	20						
Documentation	60							
Report	50	50	2					
Oral presentation	10	10						
Unexpected events (10% more time)	50							
Rough expected nb hours each week		500	40	40	0	8	20	20
Total nb hours per week		496	27	39	0	10	20	22
Rough estimation total nb hours	549							

Figure 11.6: Project planning, part 1/3

Tasks	April					May			
Full micromouse hardware and software	1 - 7	8 - 14	15 - 21	22 - 28	29 - 5	6 - 12	13 - 19	20 - 26	27 - 2
Comments / Milestones						PCB v1 done!	Intermediate report?	PCB v2 done!	Hardware done!
Project start									
State of the art									
Specifications									
Planning									
Architecture choice									
Hardware									
Components choice									
Order components	1								
PCB v1 design v1	12	16	2						
PCB v1 order			2						
PCB v1 assembly				15	0				
PCB v1 test					15				
PCB correction for v2					3	12			
PCB v2 order						2			
PCB v2 assembly								5	
PCB v2 test								5	
CAD design of mechanical parts			10	1			2	2	
3D printing			3	3			3		
Full mouse assembly						4	5	5	6
Software									
Micro-controller choice									
IDE setup									
First programs on MCU	5	2							
Test distance sensors									5
Test motors									5
Drive in straight corridor									
Bluetooth/wifi setup and tests									
Debug system (retrieve params by wifi)									
Turn 90 and 180°									
Follow predefined path in maze									
Implement flood fill to find good path									
Final tests and enhancements									
Documentation									
Report							6		
Oral presentation									
Unexpected events (10% more time)									
Rough expected nb hours each week	16	16	16	16	16	16	16	16	16
Total nb hours per week	18	18	17	19	18	18	16	17	16
Rough estimation total nb hours									

Figure 11.7: Project planning, part 2/3

Tasks	June				July				August		
Full micromouse hardware and software	3 - 9	10 - 16	17 - 23	24 - 30	1 - 7	8 - 14	15 - 21	22 - 28	29 - 4	5 - 11	12 - 18
Comments / Milestones									Software done!		
Project start											
State of the art											
Specifications											
Planning											
Architecture choice											
Hardware											
Components choice											
Order components											
PCB v1 design v1											
PCB v1 order											
PCB v1 assembly											
PCB v1 test											
PCB correction for v2											
PCB v2 order											
PCB v2 assembly											
PCB v2 test											
CAD design of mechanical parts											
3D printing											
Full mouse assembly											
Software											
Micro-controller choice											
IDE setup											
First programs on MCU											
Test distance sensors	5										
Test motors	8	2	0								
Drive in straight corridor	5	5	10	10							
Bluetooth/wifi setup and tests		3	4	3							
Debug system (retrieve params by wifi)		6	3	5							
Turn 90 and 180°					10	10					
Follow predefined path in maze							15	5	10		
Implement flood fill to find good path								10	20		
Final tests and enhancements									10	10	
Documentation											
Report										32	10
Oral presentation											10
Unexpected events (10% more time)											
Rough expected nb hours each week	16	16	16	16	16	16	16	16	40	40	20
Total nb hours per week	18	16	17	18	10	10	15	15	40	42	20
Rough estimation total nb hours											

Figure 11.8: Project planning, part 3/3