CHALMERS

Design and Development of a Robot Vacuum Cleaner

Bachelor's Thesis in Systems and Control

Robin Björklund, Georg Eriksson, Benjamin Jansson,
Gustav Knutson, Simon Rehn, Nidal Zeineddin

DEPARTMENT OF ELECRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg 2025
www.chalmers.se

Bachelor's Thesis 2025

Design and Development of a Robot Vacuum Cleaner

Robin Björklund, Georg Eriksson, Benjamin Jansson,
Gustav Knutson, Simon Rehn, Nidal Zeineddin

CHALMERS

Department of Electrical Engineering
Project group EENX16-VT25-12
Chalmers University of Technology
Gothenburg 2025

Design and Development
of a Robot Vacuum Cleaner

Robin Björklund, Georg Eriksson, Benjamin Jansson,
Gustav Knutson, Simon Rehn, Nidal Zeineddin

Examinor: Karinne Ramirez-Amaro, E2
Supervisor: Yingshuai Quan, E2

Bachelor's Thesis 2025
Department of Electrical engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telefon +46 31 772 1000

Cover Image: Visualization of the robot vacuum cleaner.

Typeset in LaTeX
Gothenburg 2025

Abstract

This bachelor's thesis presents the design, development, and evaluation of a robot vacuum cleaner. The system was developed as an integrated solution combining both hardware and software. The mechanical design was created using CAD software and then 3D-printed. The robot uses a differential drive configuration and employs Robot Operating System 2 (ROS 2) for control and communication. Key features include a brush and fan-based cleaning mechanism. The prototype was evaluated in terms of suction and brush performance, as well as its ability to follow predefined trajectories. Results demonstrated effective cleaning capability and reliable maneuverability within the design constraints, though some areas for improvement were identified. The thesis concludes with a discussion of the results, an analysis of the design, suggestions for future improvements, and a final conclusion.

Keywords: Robot vacuum cleaner, ROS 2, CAD, 3D-printing

Acknowledgments

We would like to thank our supervisor, Yingshuai Quan, for her continuous support during this project. We would also like to express our appreciation to our examiner, Karinne Ramirez-Amaro, for her valuable feedback and expert insight throughout the course of this project.

Special thanks goes to the staff at the CASE-Lab at Chalmers University of Technology, for providing guidance, access to essential tools and equipment and 3D-printing facilities, that enabled the construction of the robot vacuum cleaner.

Contents

List of Figures

List of Tables

List of Acronyms

ABS	Acrylonitrile Butadiene Styrene
CAD	Computer Aided Design
CAGR	Compound Annual Growth Rate
CAN	Controller Area Network
CFM	Cubic Feet per Minute
CLI	Command Line Interface
CPU	Central Processing Unit
FDM	Fused Deposition Modeling
IC	Integrated Circuit
ICR	Instantaneous Center of Rotation
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
LiPo	Lithium-ion Polymer
MCU	Microcontroller Unit
PC	Polycarbonate
PCB	Printed Circuit Board
PETG	Polyethylene Terephthalate Glycol
PLA	Polylactic Acid
ROS 2	Robot Operating System 2
SBC	Single Board Computer
SPI	Serial Peripheral Interface
TPU	Thermoplastic Polyurethane
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
vSLAM	Visual Simultaneous Localization and Mapping

1
Introduction

Robot vacuum cleaners have become an essential part of modern household cleaning. During recent years, great improvements have been made to these inventions, with features such as automated docking, wet mopping and effective cleaning performance. Alongside with an efficient navigation and mapping system, the robot vacuum cleaners have become a highly appreciated appliance. However, despite these benefits, robot vacuum cleaners still face challenges. This report presents the project of designing, developing, and implementing a robot vacuum cleaner. The project involves detailing the hardware components, designing the control systems, and implementing navigation strategies. Further improvements will also be discussed.

1.1 Background

The robot vacuum cleaner market is growing rapidly, with its size estimated at 5.67 billion USD in 2024 [1]. In the United States, it is projected to grow at a compound annual growth rate (CAGR) of 10.6% [2]. In comparison, more traditional vacuum cleaner, such as canister and upright models, have slightly lower CAGR rates of 8.7% and 8.6%, respectively, making the robot vacuum segment the fastest growing in the market.

Since the introduction of the robot vacuum cleaner, the devices have evolved with features such as advanced navigation systems, mapping algorithms and sensor technology. The most common method for the robot vacuum cleaner navigation systems are either Light Detection and Ranging (LiDAR) sensors or Visual Simultaneous Localization And Mapping (vSLAM) [3]. Together with their ability to map the environment in real time, these technologies enable the robot to efficiently navigate around obstacles, optimize cleaning patterns, and ensure complete coverage of the floor. In addition, their advanced sensors help to detect dust, allowing the vacuum cleaner to adjust its suction power and cleaning path accordingly. This combination of accurate navigation, adaptive cleaning, and intelligent obstacle avoidance makes robot vacuum cleaners highly effective in maintaining clean floors with minimal user intervention.

Despite these advanced features, robot vacuum cleaners still struggle to clean effectively in certain areas. Dust near corners or close to objects is often missed, and the robot vacuum cleaner frequently gets stuck. Both these result in need for manual assistance, which is undesirable from a user experience perspective.

1.2 Aim

The aim of the project is to design and develop a functional robot vacuum cleaner that integrates both hardware and software. The project will primarily focus on creating a robot capable of driving, turning, and following pre defined trajectories. The project will also focus on designing a cleaning mechanism that allows the robot to function as a vacuum cleaner.

1.3 Objectives

A list of objectives is formed to ensure a successful development of the robot vacuum cleaner. These objectives define important functions that need to be achieved to fulfill the projects aim.

1. **Mechanical and Electrical Design**
   Early in the project, a mechanical design for the robot vacuum cleaner will be developed. This will include the main chassis as well as various custom made components essential to the robot's functionality. While the chassis and some parts will be designed, the electronics and several other components will need to be purchased. Research will be carried out to identify the most suitable custom and pre manufactured components. Once selected, these parts will be integrated into the chassis and overall hardware assembly. This is an important step, since future decisions will be influenced by it, both hardware and software.
2. **Basic movement and control**
   This step focuses on developing the software to enable basic movement functionality. This includes moving forward, backward, turning and following pre defined trajectories using commands sent from the computer.
3. **Cleaning mechanism**
   The goal is to develop a cleaning mechanism, fulfilling the main task of a vacuum cleaner. A simple fan implementation will be made, which together with a filter and dust container will collect dirt and debris while moving around.
4. **Enhanced navigation with LiDAR**
   When previous steps have been fulfilled and if time allows, the final aim is to implement a LiDAR sensor, enhancing the navigation of the robot vacuum cleaner. This will map the environment and assist with obstacle avoidance.

1.4 Limitations

Several limitations affect the development of the robot vacuum cleaner, potentially impacting its overall functionality. The most significant constraint is the limited time frame, as the project runs from January to May. The fixed schedule limits what can be implemented, possibly making certain time consuming features impractical. Therefore, simpler solutions might be selected instead of more complex ones to ensure the project is completed on time

The project also has a set budget of 5000 SEK. Therefore, it is essential to make smart decisions and maximize the value of the purchases. There may be times where a cheaper solution has to be implemented rather then the most optimal, because of the budget. It is also important to keep some funds aside, in case something goes wrong.

The main focus in this project will be the robot vacuum cleaner's movement, navigation and cleaning performance. The aesthetics of the prototype will not be taken much into consideration. Because of both the time limit and set budget, spending time and money on aesthetics means spending less time on the other objectives, which is why minimal thoughts will be spent on this.

Developing efficient control algorithms and competent navigation system can be complex and time consuming. The main focus will be to implement basic movement and control, resulting in less time spent on things like smooth movement and effective mapping.

The choice of components such as the motors, sensors, and microcontrollers might limit the robot's performance in terms of speed, precision, or weight capacity. These aspects will not be highly prioritized, as the focus will be on basic functionality over optimized performance.

2
Theory

This chapter presents the theoretical foundations necessary for understanding the principles in designing a robot.

2.1 Control theory

Control theory is a branch of engineering and mathematics that deals with dynamic systems. The primary objective is to design controllers that can automate the regulation of a system's output to match a desired input or a reference value. In a typical feedback control system, the controller compares the desired value with the measured output and computes the error, seen in equation (2.1):

$$e(t) = r(t) - y(t), \tag{2.1}$$

where $e(t)$ is the error value, $r(t)$ is the reference value, and $y(t)$ is the output value. This error, $e(t)$, is then used by a controller to compute a new control signal, $u(t)$, that adjusts the system's behavior.

2.1.1 P-controller

A P-controller (Proportional-controller) is a type of feedback control system widely used in engineering to regulate physical processes. It continuously calculates an error value based on a desired value and the measured value. It applies a correction based on a proportional gain, shown in equation (2.2):

$$u(t) = K_p \cdot e(t), \tag{2.2}$$

where $u(t)$ is the control signal, K_p the proportional constant, and $e(t)$ the error value.

2.1.2 PD-controller

A PD-controller (Proportional-Derivative controller) is a type of feedback control mechanism commonly used in control systems, particularly in robotics and automation. It combines two components, proportional and derivative gain. The proportional gain reacts to the current error, similarly to a P-controller. In addition, the derivative gain reacts to the rate of change of the error, helping to dampen the system's response and reduce overshoot. This can be seen in equation (2.3):

$$u(t) = K_p \cdot e(t) + K_d \cdot \frac{d}{dt}e(t), \tag{2.3}$$

where $u(t)$ is the control signal, K_p the proportional constant, K_d the derivative gain, and $e(t)$ the error value.

2.2 Differential drive

A differential wheeled robot is a type of mobile robot that moves using two independently driven wheels mounted on either side of its body. Steering is achieved by varying the relative speeds of the wheels rather than using a traditional steering mechanism. This setup enables straightforward motion control and is widely adopted in robotics due to its low cost and simplicity. Typically, a caster wheel is used to balance the robot and prevent it from tipping.

The robot's orientation and movement depend on the tangential velocities of the left and right wheels. If both wheels move at the same speed and direction, the robot travels straight. If one of the wheels rotates faster than the other, the robot turns in a radius around the Instantaneous Center of Rotation (ICR). ICR refers to the point around which the robot is momentarily rotating at any given time. This point lies on the axis that connects the two wheels, and its location depends on the relative velocities of the left and right wheels. This is illustrated in figure 2.1. [4]

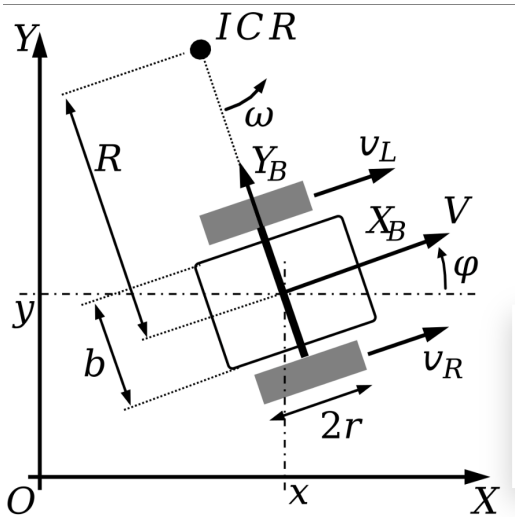

Figure 2.1: Differential drive kinematics

The angular velocity of the robot is calculated using equation (2.4):

$$\omega = \frac{v_R - v_L}{b} \tag{2.4}$$

The radius to the ICR is calculated using equation (2.5):

$$R = \frac{b}{2} \cdot \frac{v_R + v_L}{v_R - v_L} \tag{2.5}$$

The linear velocity of the robot's center is calculated using equation (2.6):

$$V = \frac{v_R + v_L}{2} \tag{2.6}$$

2.3 Odometry

Odometry is a way for a robot to estimate its position and orientation by keeping track of how much the wheels have turned. For the vacuum robot, which uses a differential drive system, this means measuring how far each wheel has moved and using that data to figure out where the robot is and which direction it is facing.

As shown in figure 2.2, the robot starts at the origin and moves to a new position by driving forward and possibly turning. The new odometry data (x', y', θ') is calculated based on the wheel encoder readings, which are translated into distance and rotation. It is essential for tasks such as following paths, avoiding obstacles, and knowing when to trigger sensor-based corrections.

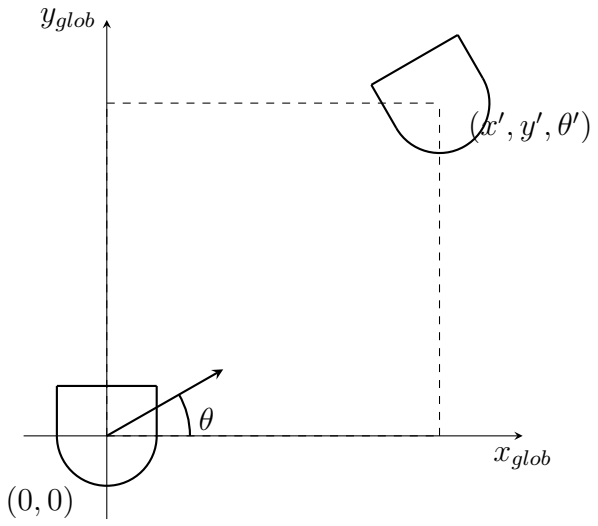

Figure 2.2: Odometry visualization showing initial and final poses of the robot.

2.4 Robot Operating System 2 (ROS 2)

Robot Operating System 2 (ROS 2) is an open source middleware framework for robotics development. It provides tools and libraries for communicating between different components using a publish-subscribe model and is widely used in industrial and research applications [5]. This project focuses on ROS 2 Humble.

2.4.1 Nodes

A node can be described as a ROS 2 element that serves a single modular purpose in a robot system. Each node runs as an independent process and typically handles one well defined task such as actuator control, sensor data acquisition, or decision making. A motor control node subscribes to velocity commands and translates them into control signals for actuators such as the drive motors. A filtering node improves the quality of raw sensor data, for example by applying a Kalman filter to encoder readings for more accurate pose estimation. A communication node manages data exchange with external systems, such as sending status updates or receiving commands over Wi-Fi. These nodes interact through the ROS 2 framework, supporting a modular and efficient robot architecture. The nodes communicate with other nodes through four different ways, topics, actions, services and parameters [6]. An illustration for this is shown in figure 2.3.

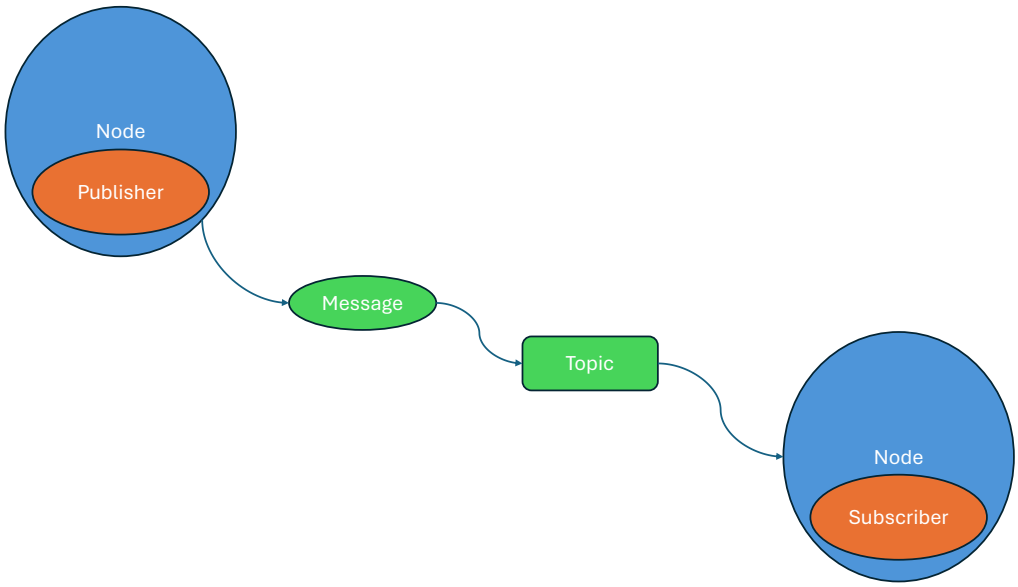

Figure 2.3: ROS 2 publisher-subscriber node

2.4.2 Topics

Topics are a way for nodes to share information with each other. It does this by allowing any number of nodes to fetch the same information by subscribing to the topic, and also allow each node to publish information through any number of topics,

allowing for effective transfer of information inside a robot system. It works much like a postbox. One node places a message in the postbox (topic) and one or multiple nodes can go and read the message from the postbox. Topics are used in continuous data streams and are therefore ideal in the case of sensor readings. Sensors have an asynchronous data stream where sometimes nothing happens and other times loads of data is supposed to be sent. The nodes interested in the sensor data can subscribe to the topic and receive the data whenever there is new information. [6]

2.4.3 Services

Services provide a structured way for nodes to communicate using a call and response model. This differs from the topic based publish–subscribe system in that the service client, a node, requests information or a specific action from the service server, another node, which replies with the requested information or the result of the action. This interaction is synchronous and blocking, meaning the client pauses and waits for the server's reply before continuing its operation. The service mechanism is particularly useful in scenarios where the client needs specific data or actions that are only used once, such as when a robot needs to be commanded to start or stop a task, request its current status, or store a map of its environment. Unlike topics, which are suitable for high frequency data, such as sensor readings or motor commands, services are intended for isolated or discrete actions that require confirmation or feedback. This distinction makes services more appropriate for implementing command interactions within a robotic system. [7]

2.5 Distributed Embedded System

A Distributed Embedded System is a system architecture in which multiple embedded processors or microcontrollers work together to perform a coordinated set of tasks. Rather than relying on a single processor to handle all functionality, the system is divided into discrete subsystems, each responsible for specific operations such as sensing, control or computation. These subsystems communicate with each other using well defined protocols, typically over interfaces such as UART, IC, SPI, CAN, or Ethernet. The primary justification for distributed embedded systems lies in their ability to provide real time performance and modularity. Certain tasks in embedded applications, such as precise motor control or collecting sensor data, require strict timing. That poses an issue in general purpose systems. Assigning such tasks to dedicated microcontrollers ensures that they can operate without interruption, maintaining consistent timing and responsiveness. [8]

Distributed systems also enable parallelism by allowing different processors to operate at the same time. This reduces computational load on each unit and enables the system as a whole to respond more quickly and efficiently [8]. Furthermore, distributing functionality supports a modular design approach, where components can be developed, tested, and modified independently. This improves maintainability and facilitates scalability when adapting the system to more complex tasks or environments. The use of distributed architectures allows designers to match hardware capabilities to task requirements. Simpler, low power microcontrollers can be used for repetitive or timing sensitive control tasks, while more powerful processors can be reserved for tasks requiring greater computational complexity, such as image processing or path planning. [8]

2.6 3D-printing

Fused Deposition Modeling (FDM), otherwise known as 3D-printing, is an additive process in which thermoplastic filament is heated and forced out of a hot nozzle and extruded layer by layer onto a build plate to form three dimensional parts. FDM is popular due to its low cost, time efficiency, availability, and flexibility with a broad range of applications ranging from prototyping to end-user parts. [9]

Material selection is a crucial element of effective 3D-printing since all materials have individual properties that impact their print-ability, strength, flexibility, and thermal resistance. The most widely used material is PLA (Polylactic Acid) because it is easy to print with, possesses good dimensional integrity, and degrades biologically [10]. However, PLA is brittle and not suitable for high heat. PETG (Polyethylene Terephthalate Glycol) is more heat- and flex-resistant than PLA and is therefore suitable for thicker parts, but potentially more prone to warping and stringing. Another possible material, ABS (Acrylonitrile Butadiene Styrene), is a rigid material but more difficult to print due to extreme thermal shrinkage, which can lead to warping. It must also be printed in a controlled environment. In perfor-

mance applications, PC (Polycarbonate) may be employed, with more strength and heat resistance, though with the requirements of precise printer settings and higher temperatures. TPU (Thermoplastic Polyurethane) introduces rubbery flexibility, with the capacity to print flexible pieces, but more difficult to print. Material selection depends on the functional requirements of the printed part, so understanding each filament's behavior, including the bed adhesion needed based on print sheet type, is essential for high-quality output.[11].

3

Technical Background and Components

This chapter provides an overview of all hardware components utilized in the system, along with an explanation of their respective functions.

3.1 Single Board Computer

A Single Board Computer (SBC) is a compact integrated circuit designed to execute specific tasks in embedded systems. These are usually Printed Circuit Boards (PCB) that contain components commonly found in a PC or laptop, such as a central processing unit (CPU), USB ports, and other essential components. However, while an SBC might not be as powerful as a laptop or PC, they offer cost effective solutions for a wide range of applications, including robotics, education, and prototyping. [12]. For this project, the Raspberry Pi 4B was chosen, see figure 3.1.

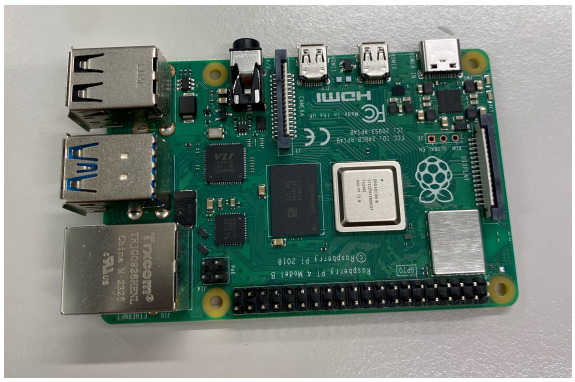

Figure 3.1: Raspberry Pi 4B

3.2 Microcontroller

A microcontroller (MCU) is a small computer without a complex operating system, used within embedded systems to perform specific tasks. Like a single board computer, an MCU also has components commonly found in a PC, such as a CPU, allocated memory, and other general components depending on the model. Due to their small size, low power consumption, low cost and ease of use, MCU's are widely used for battery powered systems [13]. A significant difference between a MCU and a SBC is that MCU's are usually based on the Harvard-architecture, which allows for the CPU to read instructions and access memory at the same time, allowing for faster operation of basic tasks. SBC's, on the other hand, uses the Von Neumann-architecture which has better scalability and reliability. Furthermore, the MCU lack a complex operating system, meaning that it will not have delays that a Raspberry Pi might experience due to multitasking. Therefore, using an MCU in combination with an SBC results in better performance in handling real time tasks [14]. This project uses an Arduino Nano, see figure 3.2.

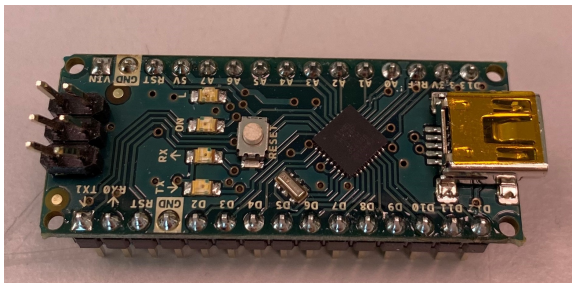

Figure 3.2: Arduino Nano developement board.

3.3 Drive motors

The motors for driving the robot vacuum cleaner needs to meet several general requirements. They have to provide sufficient torque to propel the robot across various surfaces, operate at a practical rotational speed, consume a reasonable amount of energy, and be compatible with the chosen motor controller. These requirements were derived by analyzing specifications and performance characteristics of similar commercial products. The exact torque values were not considered critical at the time of purchase, as torque can be adjusted through gear reduction if necessary. Based on the available information and the constraints of the project, a suitable brushed DC geared motor was selected that satisfied the practical needs of the prototype while aligning with the general expectations for robotic vacuum cleaner performance. The chosen motors are shown in figure 3.3, with specifications in table B.3 in Appendix.

Figure 3.3: Drive motor

3.3.1 Encoders

The chosen motors have built in encoders. Motor encoders are essential components in robot vacuum cleaners because they provide feedback on the rotation of the motors. This feedback allows the system to monitor and control the movement of the robot in a precise and predictable way. Encoders typically measure the number of revolutions or partial revolutions of a motor shaft, which can be used to calculate speed, direction, and distance traveled.

3.4 Brush motor

A motor powers a brush located under the chassis, whose purpose is to sweep dust at the edge of the robot closer to the suction inlet. The brush motor must consume minimal energy and spin at a moderately high rate. These were fulfilled by the RS PRO Brushed Geared DC Geared Motor, shown in figure 3.4. For detailed specifications of the brush motor, see table B.1 Appendix.

Figure 3.4: Brush motor

3.5 Motor driver

DC-motors require higher voltage and current than the microcontroller can provide. A motor driver is used as an interface between the motors and the microcontroller to be able to drive motors. A motor driver takes low power control signals from the micro controller and amplifies them. The motor driver chosen for this project is the L298 Dual Full-Bridge driver and is shown in figure 3.5. It includes H-bridge circuits which enables the motors to run in both directions which is crucial for a robot vacuum cleaner. An H-bridge consists of four transistors arranged in an H-shape. They control the current flow through the motor determining its direction. The motor driver also includes a 5V voltage regulator which supplies the microcontroller with the correct input voltage. More specifications can be found in table B.5 in the appendix. [15]

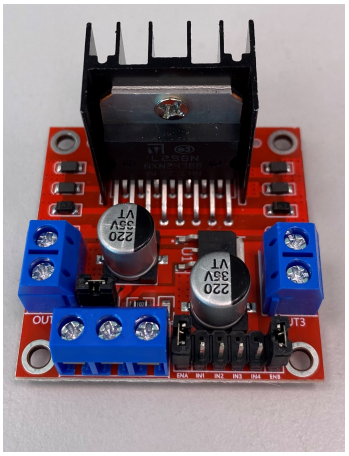

Figure 3.5: L298 motor driver

3.6 Buck Converter

A voltage regulator is required to step down the 12V from the battery to 5V, as the Raspberry Pi operates at a 5V input. For this, the LM2596 Buck Converter works well. See figure 3.6. Specifications can be found in table B.6 in Appendix.

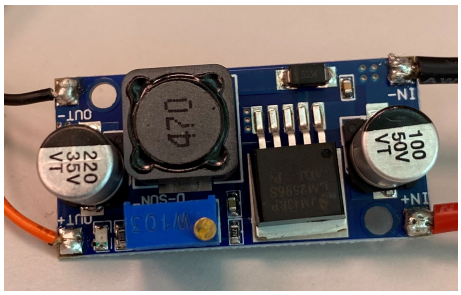

Figure 3.6: LM2596 Buck Converter

3.7 Wheels

Two wheels with a diameter of 65 mm are used as drive wheels. The wheels have deep treads, providing the robot vacuum cleaner with improved traction. The drive wheels are shown in figure 3.7. To stabilize the robot, a ball caster is placed at the back, shown in figure 3.8.

Figure 3.7: Drive Wheels

Figure 3.8: Ballcaster

3.8 Fan blower

The fan blower generates airflow to enable dust and debris collection in a robot vacuum cleaner. It produces a negative pressure, pulling air along with dust and debris into the vacuum chamber, creating the cleaning mechanism. The airflow in vacuum cleaners are measured in CFM. A normal vacuum cleaner typically have between 50-100 CFM [16]. The Sanyo Denki San Ace Blower, shown in figure 3.9, has a CFM of 56.8, which is enough for this project. More specifications is found in table B.2 in Appendix.

Figure 3.9: Fan blower

3.9 LiDAR

LiDAR is a remote sensing technology that determines the distance to an object by measuring the time it takes for a laser pulse to travel to the object and back. This is based on equation 3.1:

$$d = \frac{c \cdot t}{2}, \tag{3.1}$$

where d is the distance, c is the speed of light, and t is the measured time [17]. Because light travels extremely fast, this measurement must be incredibly precise, often requiring high speed electronics and finely tuned timing circuits. LiDAR sensors typically emit laser pulses in the near infrared range, although the specific wavelength can vary depending on the application, for instance, longer wavelengths may be used for better penetration through dust or fog. Modern LiDAR systems can generate thousands to millions of pulses per second, allowing them to create detailed 2D or even 3D maps of their surroundings in real time. Since a LiDAR sensor uses its own laser light, it does not rely on ambient light at all. It works just as well in complete darkness as in daylight, which can be highly useful when used in a robot vacuum cleaner that needs to navigate under furniture, in dim rooms, or at night[18]. The chosen LiDAR for this project is the LiDAR LD06 and is shown in figure 3.10.

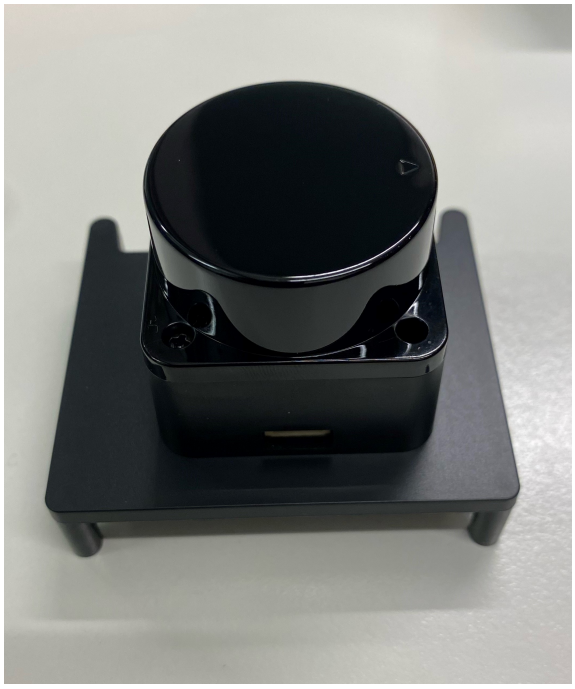

Figure 3.10: LiDAR LD06

3.10 Linux Gamepad (joystick)

Controlling the robot manually requires a controller capable of sending inputs to the Raspberry Pi wirelessly, and also that these inputs can be set to control the robot. Another requirement is that it has variable inputs so that the motors would not either be set to max or minimum as that would make maneuvering the robot difficult. All of these are fulfilled by any normal wireless controller used for gaming, such as the one chosen for the project, Microsoft Xbox Elite Wireless Controller Series 2, shown in figure 3.11.

Figure 3.11: Xbox Elite Wireless Controller Series 2

3.11 Battery

To power the robot vacuum cleaner, the 3 cell LiPo LemonRC battery is used, which is more than sufficient to power the robot, shown in figure 3.12. Specifications are shown in table B.4 in the appendix.

Figure 3.12: Battery

4
Design and Development Process

This chapter presents the design and development process of the robot vacuum cleaner, covering both hardware and software implementations

4.1 Project Resources

The team had access to the CASE-Lab at Chalmers, where system development, hardware integration and solution implementation were conducted. Catia V5 was used for the design and development of the chassis and other parts. They were then 3D-printed in the CASE-Lab. There are a few different types of printers with different nozzle sizes, varying from 0.4 mm - 0.8 mm. There are also different size printers, the default-sized with a build volume of 250 x 210 x 220 mm (width x depth x height) and the Prusa XL printer with the ability to use multiple colors and materials in one print and a build a volume of 700 x 900 x 720 mm. The team needed to use two different printers, which were the default-sized printer with a 0.4 mm nozzle and the Prusa XL. The usage of the Prusa XL printer was necessary for the printing of the chassis and cover. The other printer was used to print smaller parts that were used to assemble the robot.

The application Prusa Slicer was used to convert a CAD-model to G-code for the 3D-printer. In Prusa Slicer, the team adjusted several settings to achieve the desired print quality. Table 4.1 shows the settings used for all prints in this project. The only variation was the use of a brim, which was not required for some prints

Table 4.1: Print settings

Setting	**Value**
Material	addnorth E-PLA
Structural Layer Height	0.20 mm
Infill Percentage	15%
Infill Pattern	Grid
Supports	Everywhere
Adhesion Type	Brim
First Layer Height	0.2 mm
Layer Height	0.15 mm
Perimeters	3

The team also had access to a soldering station, where all wiring and soldering of the

electronic components was carried out. In addition, the CASE-Lab also provided a well equipped workspace with all the necessary tools and materials required for assembling the robot. This included a wide variety of wires, screws, glue, tape, and other essential components.

4.2 Component list

Listed in table 4.2 are the components and materials used for this project.

Table 4.2: List of components for the robot vacuum cleaner

Qty.	Name	Comment
1	Raspberry Pi 4B	Single-board computer (SBC)
1	Arduino Nano	Microcontroller, development board version
1	L298N Motor driver	Motor controller with H-bridge
2	12V DC motor with encoders	For wheel drive
2	Drive wheels	Treaded for better traction
1	Ball caster	For balance and smooth motion
1	Fan blower (Sanyo Denki San Ace)	Generates suction force
1	LiDAR LD06	For mapping and obstacle detection
1	Brush motor (RS PRO DC Motor)	Side brush motor for edge cleaning
1	Buck Converter (LM2596)	Steps down 12V to 5V for Raspberry Pi
1	LiPo Battery (11.1V, 1600mAh)	Main power supply
1	Xbox Elite Wireless Controller	Joystick for manual control
–	Various wires and connectors	For electrical connections
–	Various screws and bolts	For assembly
–	PLA	3D-printing filament
–	PETG	3D-printing filament

4.3 Hardware design and integration

This section describes the process of designing the robot vacuum cleaner, including the integration and fitting of purchased components.

4.3.1 Chassis

When deciding on the design of the chassis, the most common choice is a fully circular design. The reason for this is the ability to turn 180 degrees, which often can be useful in tight spaces, shown in figure 4.1. However, this limits the ability to fully clean corners. The team instead decided on a square front, with a rounded rear, figure 4.2. This will improve the ability to fully clean corners [19]. Instead of rotating when facing a wall, the robot vacuum cleaner will have to reverse and then turn.

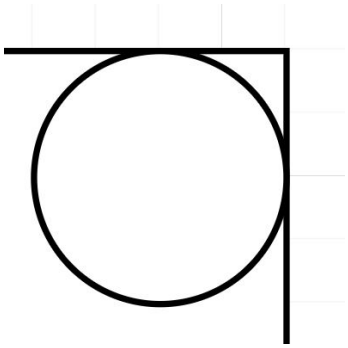

Figure 4.1: Overview of circular design

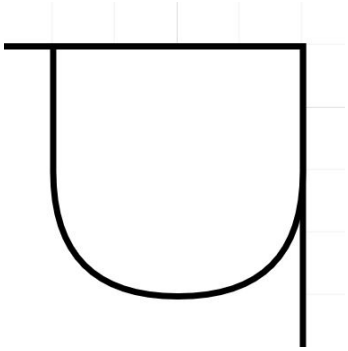

Figure 4.2: Overview of chosen design

A CAD-model of the bottom outer part of the chassis was made first. When designing this section, it was split into two parts to simplify the process, shown below.

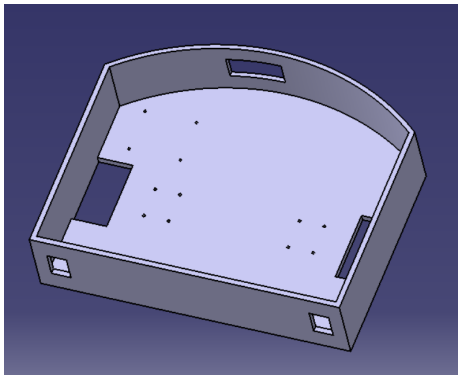

Figure 4.3: CAD-model of the bottom outside part

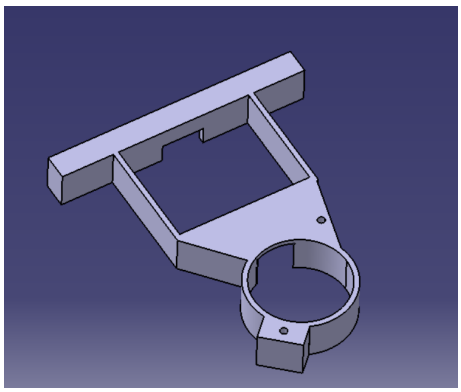

Figure 4.4: CAD-model the fan mount and airflow tunnel

Figure 4.3 shows the outside part for the bottom chassis. The half circle/rectangle design is used, with the rectangular part being the front. Holes were cut out in the front for distance sensors, labeled '1' in figure 4.5, and in the bottom for the wheels, labeled '2' in figure 4.5. Small holes were cut out to enable fastening of the motors and the ball-caster. The hole in the back is for outgoing flow from the fan, labeled '3' in figure 4.5, making sure the inside is not being overheated. The inside part, figure 4.4, will be mounted on the outside part and will be responsible for the cleaning function. The fan blower will be placed on the circular opening in the assembled chassis, '4' in figure 4.5, creating a suction through the tunnel leading to the suction inlet. These two parts were 3D-printed together, creating the main part of the chassis. The drawings that showcases the main chassis and the fan mount and airflow tunnel are illustrated in figures A.2, A.4 and A.3 in Appendix.

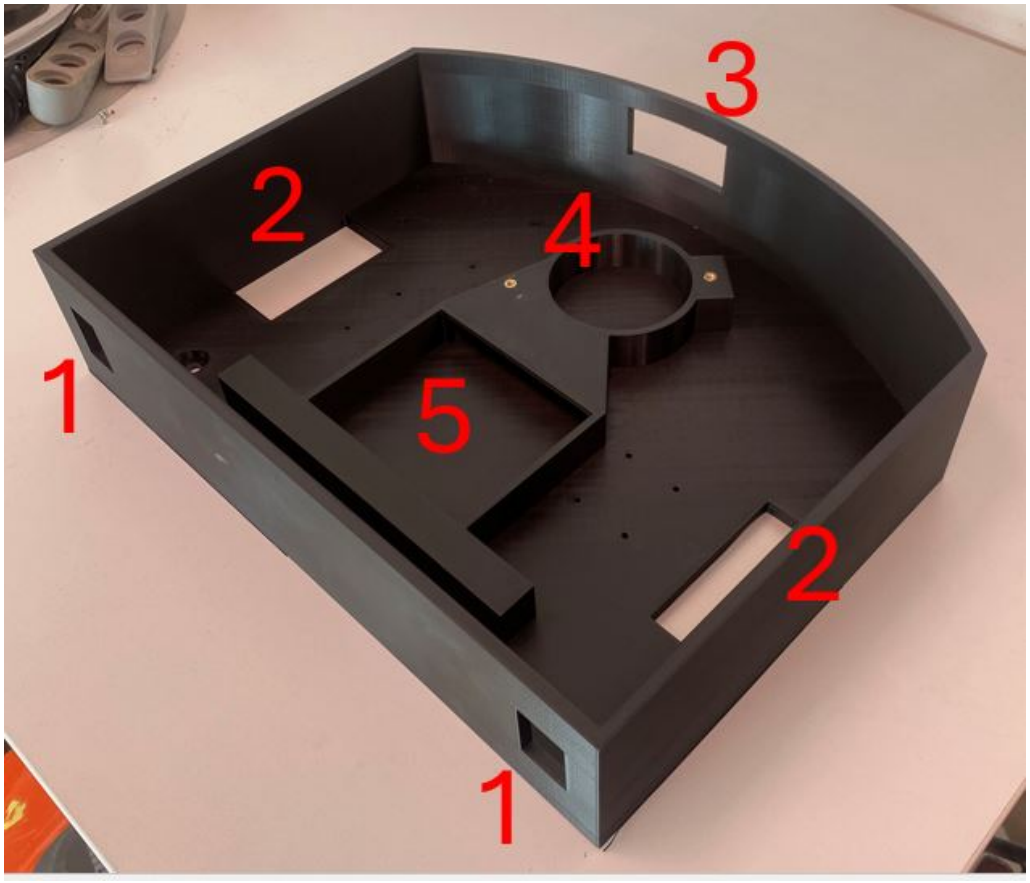

Figure 4.5: 3D-print of chassis

Figure 4.6 shows the complete chassis from underneath, figure 4.7 shows it from the top. Seen in figure 4.6, the suction inlet, labeled '6' is placed in the front part of the chassis and leads through the tunnel, labeled '5' in figure 4.5. The airflow will move through this tunnel, into the fan blower, and out through the hole in the back wall. However, the dust needs to be collected before being blown out with the air. The open rectangle, labeled '7' in figure 4.7, is for a container with a filter, collecting the dust and debris. This container will be removable from the top to empty the collected dust.

Figure 4.6: Chassis from bottom

Figure 4.7: Chassis from top

To seal the robot vacuum cleaner, a cover for the entire chassis was constructed. In the top view, figure 4.8, a cutout for the dust container can be seen. This makes it possible to remove the container without removing the cover. Half-circles were constructed to make removal easier. A lowering was made for the LiDAR sensor, placing it in the front of the robot. In this lowering, a hole was made to allow space for the wires. In the bottom view, figure 4.9, long structural beams can be seen. These make the cover more stiff, preventing any unwanted bending. Close to the edges, blocks were designed to make the cover slide on easily. These are placed in the corners and in the back, preventing any movement when the cover is placed. The drawings for the covers top and bottom are shown in figures A.5 and A.6 in Appendix.

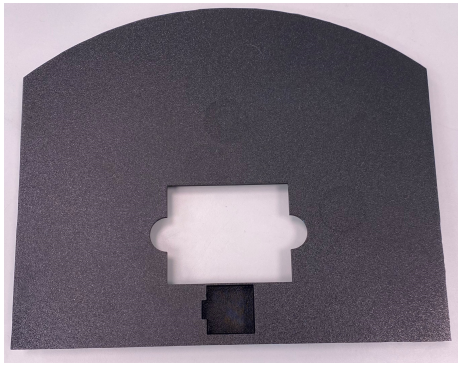

Figure 4.8: Top view of cover

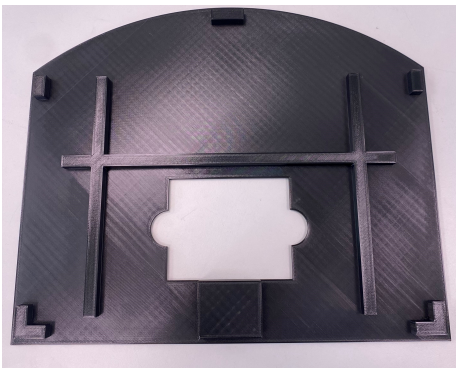

Figure 4.9: Bottom view of cover

4.3.2 Container and filter

The container, figure 4.10, was designed to smoothly slide in to the open rectangle in the chassis, preventing any air leakage. A lid was also constructed, figure 4.11, which easily attaches to the container. The drawings for these part are shown in figure A.7 in Appendix.

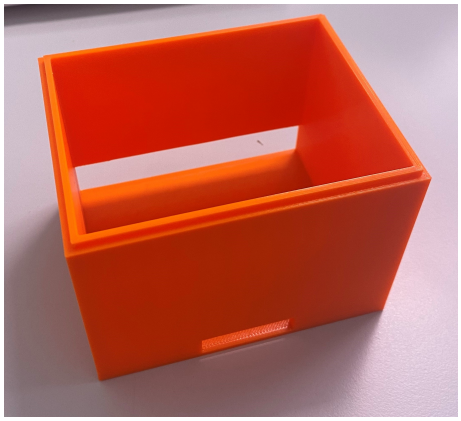

Figure 4.10: Container

Figure 4.11: Container lid

It is important to prevent any leakage, since the air pressure is vital for the entire cleaning mechanism. During the prototyping and testing of this mechanism, the airflow was directed towards the top of the container rather than passing straight through it. This created an unwanted pressure on the lid, and the wanted pressure at the suction inlet degraded. To prevent this, a part to direct the flow in the desired direction was created. This part guides the airflow from the suction inlet of the container to the fan. The part is also constructed with a much more narrow opening towards the suction inlet, creating higher pressure, resulting in an improved cleaning mechanism. The dust still needs to end up in the container, to solve this problem, small holes in the flow-direction-part was constructed. The purpose of these holes are to let dust through, without interrupting the airflow all too much. The part is shown in figure 4.12.

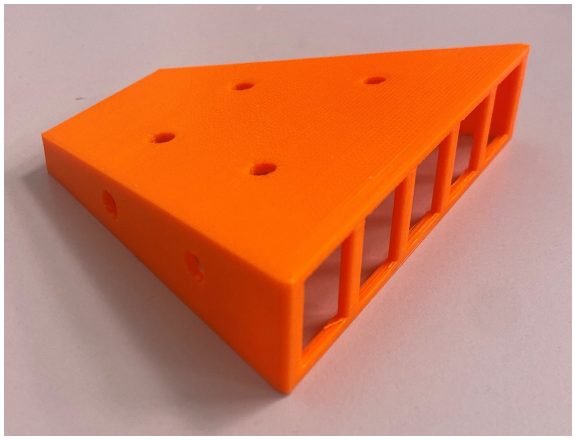

Figure 4.12: Part to direct flow

A grid was created at the larger opening in the flow-direction-part, to enable the attachment of a filter. The filter was created by cutting out a part of a paper, and fastened with tape. Ideally, this filter would be able to easily be removed and put back on, but this still simulates the filter function. Small holes were cut in the paper, making air flow easy through the filter, while still stopping dust. The filter is shown in figure 4.13.

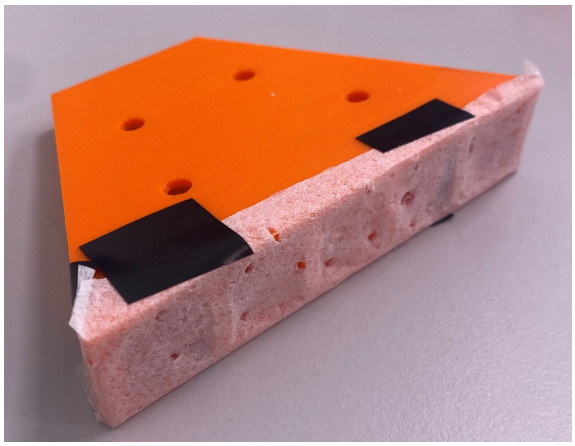

Figure 4.13: Filter

4.3.3 Motors and wheels

To attach the motors on the chassis, mounting brackets were designed and 3D-printed, shown in figures 4.14 and 4.15. These were designed to withstand bending and enables the motors to be placed close to the chassis floor, making the wheels extend the desired length on the other side.

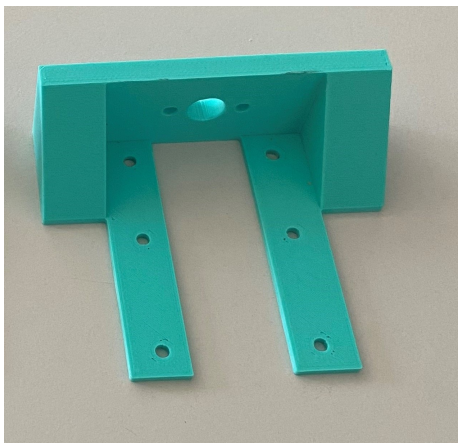

Figure 4.14: Mounting bracket

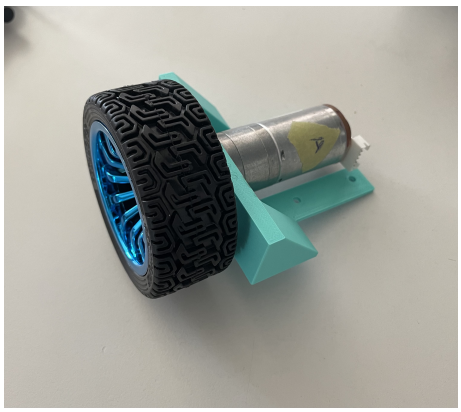

Figure 4.15: Motor and wheel mounted on bracket

The robot vacuum cleaner is equipped with two drive wheels and one ball caster for stabilization. The ball caster is mounted at the back on the bottom of the chassis, while the drive wheels are mounted inside the chassis, extending out through holes on each side. This results in a slight tilt, with the front of the robot positioned slightly closer to the ground than the back. Since the suction inlet is located at the front, this design improves the cleaning performance by keeping the suction point closer to the floor. A small recessed area was created in the front corner of the chassis to integrate the brush motor, shown in figure 4.16. This motor rotates a brush designed to sweep dust and debris towards the suction inlet. Even though the chosen chassis design improves the robot's ability to reach corners, the suction inlet is not positioned exactly at the edge. Therefore, the brush is a critical addition to ensure that dust from corners is effectively collected. An overview of the brush positioning relative to the suction inlet is shown in figure 4.17.

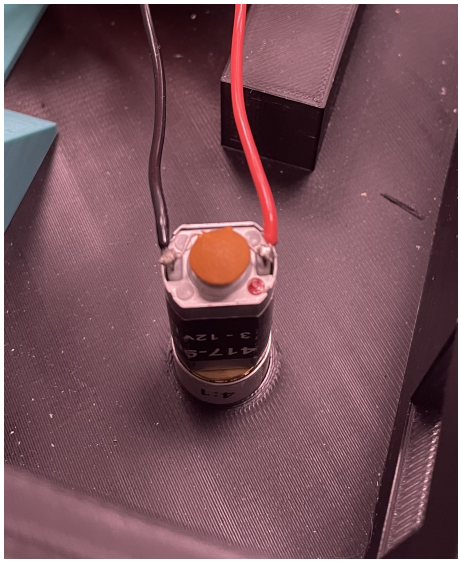

Figure 4.16: Brush motor mounted on chassis

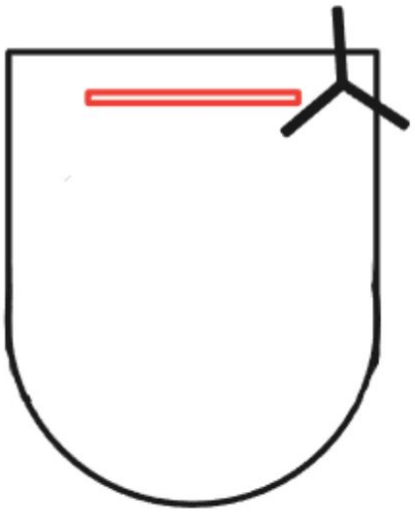

Figure 4.17: Overview with brush and suction inlet

The red rectangle represents the suction inlet, which does not reach either corner of the robot vacuum chassis. The brush, rotating anti clockwise, sweeps the dust in front of the inlet, making sure no area is missed. To attach the brush to the brush motor, a small part was printed, which the brushes are fastened to. Figure 4.18 shows the printed part with attached brushes, while figure 4.19 shows the part mounted on the brush motor.

Figure 4.18: Part to attach on brush motor

Figure 4.19: Mounted on motor

4.3.4 Mounting hardware

To mount the electronics on the chassis, several custom brackets were designed. Each bracket was engineered to allow the electronic components to be secured with screws. A small gap was included between the components and the mounts to accommodate pins and other protruding parts. To attach the mounts to the chassis, an extension was added. Figures 4.20 and 4.21 show the mounts for the Raspberry Pi and motor driver while figures 4.22 and 4.23 show the Raspberry Pi and buck converter mounted on their respective brackets.

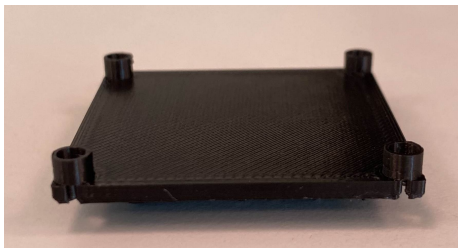

Figure 4.20: Motor driver holder

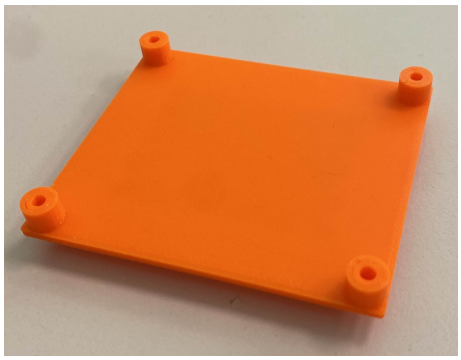

Figure 4.21: Raspberry Pi holder

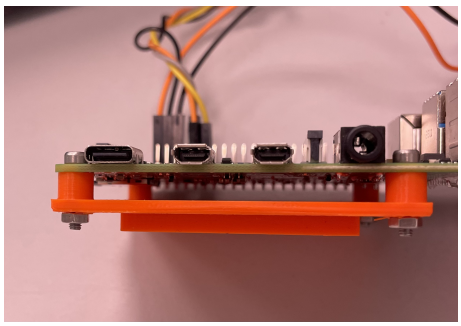

Figure 4.22: Raspberry pi mounted on holder

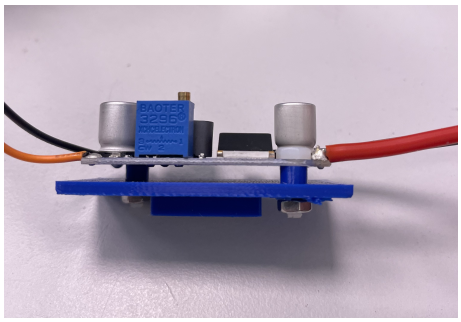

Figure 4.23: Buck converter mounted on holder

4.3.5 Final assembly

Once the chassis, motor brackets, container, and all necessary mounts were designed and 3D-printed, the final assembly of the robot could begin. The fan was placed in its designated mounting area and secured using M4 hex screws. Brass threaded inserts were used to hold the screws in place, allowing for easier and more secure installation. Care was taken to ensure there were no air leaks around the fan, as this could negatively impact the vacuum's cleaning performance.

The drive motors and wheels were mounted onto their respective brackets, which were then attached to the chassis using M3 hex screws through pre-drilled holes. The motor for the brush was installed in its assigned position using M2 screws and bolts. The electronics, including the Raspberry Pi, Arduino, motor driver, and buck converter, were mounted on their respective holders and fixed to the chassis using double sided tape. This method provided both ease of installation and flexibility for potential adjustments. Finally, the dust container with its filter could be inserted into its designated slot. An overview with all parts mounted on the chassis is shown in figure 4.24.

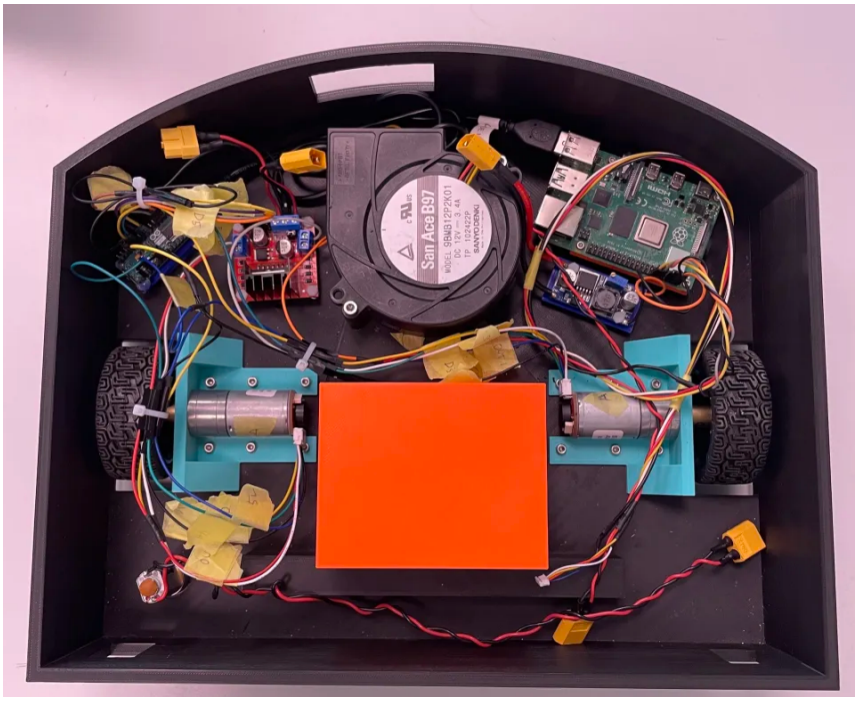

Figure 4.24: Overview of the assembled robot

Once all hardware components were mounted in place, the wiring process began. Extra care was taken to avoid mixing up any wires, as this could potentially cause short circuits. Zip ties were used to organize and tidy up the wiring for a cleaner and safer setup.

The 3 cell LiPo battery used to power the robot powers all systems of the robots. A power adapter cable, shown in figure 4.25 was made to be able to power all systems in parallel.

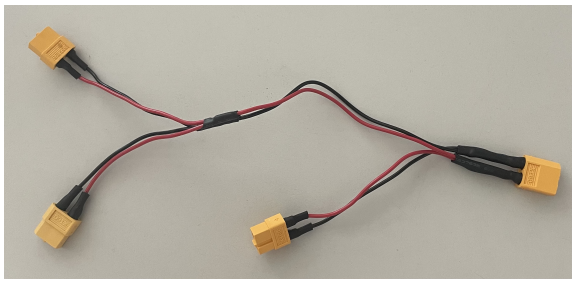

Figure 4.25: Power adapter cable

The power adapter cable has three extensions. One extension of the power adapter cable supplies voltage to the buck converter, which steps it down to 5V to power the Raspberry Pi via its GPIO pins. The Raspberry Pi, in turn, powers the LiDAR using one of its 5V output pins. The motor driver is powered directly from the battery through the second extension of the adapter cable. To ensure the brush motor operates at a suitable speed, the motor driver's built in 5V regulator is used to supply voltage to the brush motor. The Arduino is powered via a USB cable connected to the Raspberry Pi. The third extension of the power adapter is used to power the fan blower directly from the battery voltage.

The idea behind this power adapter cable was to be able to run all systems independently of each other. This proved especially useful during the software development phase, as it allowed the fan blower and brush motor to remain off when not needed, reducing noise and power consumption.

4.3.6 Circuit diagram

To gain a clear understanding of how the components interacted, a detailed circuit diagram was created. This process allowed visualization of the electric components and to ensure compatibility between the different parts. It also acted as a reference throughout the development of the design, since it is the center of the product. The diagram is shown in figure A.1 in Appendix.

4.4 Software development

This section focuses on the software part of the project.

4.4.1 Software based on hardware

The design of the software architecture in the system is directly shaped by the specifications of the hardware components. Specifically, the choice to distribute the system and divide responsibilities between the Raspberry Pi and the Arduino is driven by a need to optimize performance. This approach follows the principles outlined in the concept of distributed embedded systems, where specialized microcontrollers are assigned distinct roles to improve modularity, timing precision, and overall efficiency. As discussed in section 2.5, this type of system architecture is well suited for applications that combine real time control with computationally intensive tasks. To achieve reliable behavior in a mobile robot, it is essential to ensure that time sensitive control loops execute consistently, while at the same time also allowing for higher level functionality such as path planning, navigation, and decision making to operate in parallel without interference.

4.4.1.1 Raspberry Pi

The Raspberry Pi serves as the central computational unit of the system. It runs the ROS 2 framework and is responsible for executing complex and computationally demanding tasks. These include trajectory generation, future integration with SLAM or LiDAR-based localization, and high-level decision making. Given that the Raspberry Pi runs a operating system (Linux), it is not ideal for executing time critical control tasks that require consistent update frequencies. Interruptions from other processes and non deterministic scheduling can lead to latency in control signals which would not be optimal for this project. By offloading the low level motor control to the Arduino, the Raspberry Pi is freed from handling the real time demands of the differential drive system. This division of responsibilities ensures that the Raspberry Pi can focus entirely on higher level processes without being slowed down or made unstable by the need to manage motor feedback loops in real time.

4.4.1.2 Arduino

The Arduino microcontroller is responsible for executing the low level control logic of the robot, specifically the differential drive motion control. This includes receiving velocity commands, executing a PD controller, reading encoder values, and computing the appropriate motor signals via PWM outputs. Since the Arduino does not run a full operating system, it is capable of executing its control loop with high timing precision and no overhead from multitasking. This real time behavior is essential for smooth and accurate motor response.
Because the Arduino is dedicated solely to this task, it can handle computationally intensive control algorithms without being overburdened. Its simple control loop makes it well suited for tasks that require fast and precise execution. This configuration not only improves the responsiveness of the drive system but also makes the

robot architecture more modular.

4.4.1.3 Communication

The two components, Raspberry Pi and Arduino, communicate via a UART (Universal Asynchronous Receiver-Transmitter) serial interface over and USB-cord. UART is a widely used point to point communication protocol that is well suited for embedded systems due to its asynchronous and simple communication style [20]. In the system, UART is used to transmit velocity commands from the Raspberry Pi to the Arduino and receive encoder data in return. Given the low bandwidth of these messages, UART provides the speed needed for this communication. Therefore, the asynchronous style of the communication means that the Raspberry Pi will never have to stand idle and wait for a response from the Arduino. By using UART, the system maintains a clean separation between high level decision logic and low level actuation control, while keeping the latency low and the message protocol straightforward.

4.4.2 ROS 2

ROS 2 (Robot Operating System 2) was selected as the software framework due to its flexibility, modularity, and strong community support. It provides a solid foundation for building robotic systems, enabling an easy way to modularity add new functionality such as SLAM, LiDAR integration, or climbing capabilities without requiring changes to the rest of the system. Each part of the robot, such as motion control, sensor input, or decision making, can be handled in separate ROS nodes, making the system easier to manage. Another key advantage is the wide availability of documented packages and tools. Common features like joystick control, odometry, sensor drivers, and data visualization are readily available, which reduces development time and effort. ROS 2 also supports advanced hardware, including LiDAR sensors, cameras, and IMU's, and includes tools for filtering and processing sensor data. Overall, ROS 2 offers a reliable and scalable framework that supports both current functionality and future expansion.

4.4.3 Development stages

In this section, the development stages of the ROS 2 system is outlined, detailing the process from initial design to final implementation. See D for code, link to GitHub.

4.4.3.1 ROS 2 setup

To prepare the Raspberry Pi for deployment, the operating system Ubuntu 22.04.5 LTS (64-bit) was written to a SD card through a process commonly referred to as *flashing*. With the OS installed and working, the next step was making ROS 2 run on the Raspberry Pi. This was done by following the official ROS 2 documentation for the distribution Humble.

As part of the system setup, a main package was chosen, this is the `articubot_one`.

`articubot_one`: This repository was selected because a significant portion of its existing codebase could be effectively repurposed to fit the needs of the robot. Additionally, connections to other packages were already defined and ready to be used. The package is a template from GitHub [21].

4.4.3.2 ROS 2 and Arduino integration via serial interface

In order to ensure communication between the Raspberry Pi running ROS 2 and Arduino Nano the `serial` and `ros-arduino-bridge` were used.

`serial`: This package allows for serial communication with the Arduino through USB.

`ros-arduino-bridge`: The bridge receives wheel velocities from the ROS 2 system and can output sensor data via UART messages, over a reliable serial connection. The bridge was based on an open-source template found on GitHub [22] and was customized to fit the specific needs of the robot. The PID-controller was changed to a P-controller, and then a PD-controller. As a result, there was less complexity and tuning the controller in a later stage was made easier. This software module play a critical role in enabling the flow of information between the software stack and the hardware actuators.

To verify the communication setup, simple velocity commands were published using the ROS 2 CLI. These commands were recieved by the Arduino via the bridge and interpreted by the motor controller. Successful movement confirmed correct communication and also validated the firmware on the Arduino. Figure 4.26 illustrates the test setup. The practical wiring setup for the tests is shown in figure 4.27.

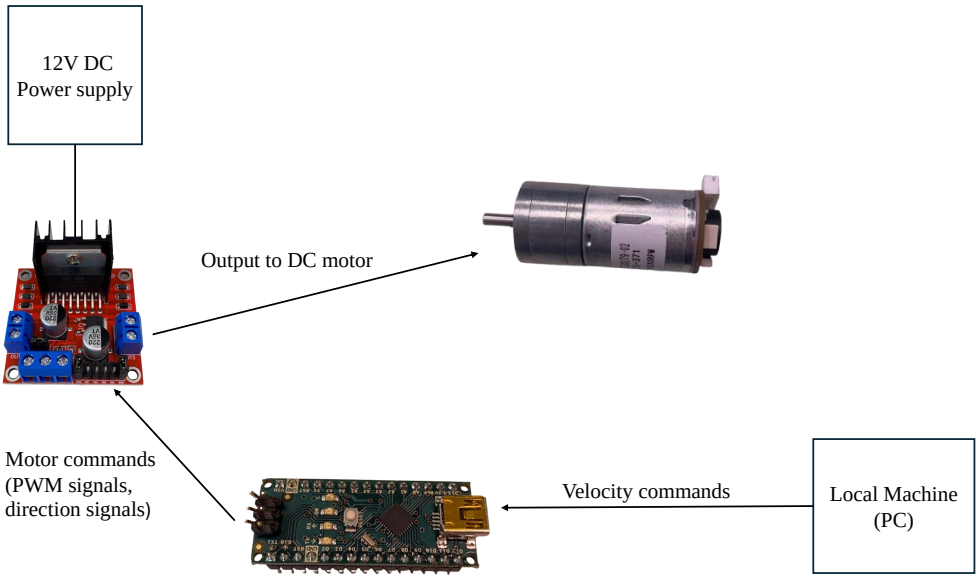

Figure 4.26: Test setup used for testing serial communication between ROS 2 and the Arduino

4.4.3.3 Encoder calibration for odometry

Accurate odometry is essential for autonomous control. To calibrate the encoders, the wheel was manually rotated one full revolution (2π rad), and the resulting encoder counts were read via the ROS 2 interface. Both wheels produced approximately 682 counts per revolution. This value was then set as `enc_counts_per_rev` in the hardware interface configuration.

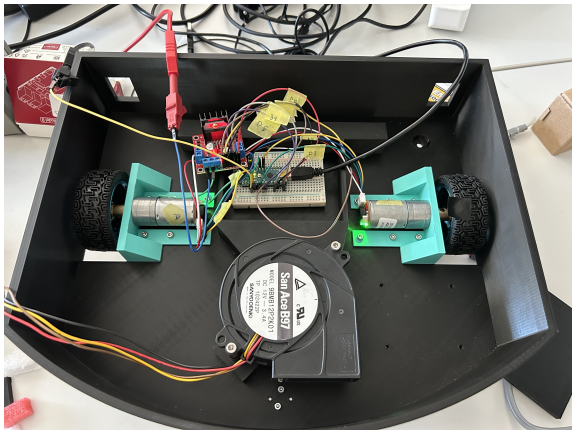

Figure 4.27: Wiring setup for encoder calibration.

4.4.3.4 Integration of `ros2_control`

Designing a multi-node architecture, where each action is handled by a dedicated node in ROS 2, may seem beneficial. This modular approach offers several benefits, particularly fault isolation, as errors or crashes are often contained within the

node where they originate. Additionally, it can improve maintainability, as individual nodes can be updated, debugged, or replaced independently. However, this design also introduces a lot of latency in the system [23]. As a consequence, the `ros2_control` framework, with `ros2_controllers` and `diffdrive_arduino`, was introduced and implemented in the system for real-time control.

`ros2_control`: This is a modular control framework within ROS 2 that provides a standardized way to connect to robot hardware with software controllers. It connects high-level commands to hardware interfaces within a single process, reducing latency and overhead. This improves timing consistency for motor control and simplifies controller management. Additionally, it integrates with ROS 2 tools seamlessly, enhancing performance, system structure, and maintainability [24].

`ros2_controllers`: The `diff_drive_controller` is a standard controller provided by the `ros2_controllers` package, designed for differential drive robots. Built on top of the `ros2_control` framework, it converts high-level velocity commands into target wheel velocities for the robot. These are passed to the hardware interface, which handles communication with the underlying motor control system. The controller also uses feedback from the hardware interface to publish odometry data, supporting localization and navigation.

Comment: The `diff_drive_controller` is named `diff_cont` in node structures and figures.

`diffdrive_arduino`: This package implements the hardware interface that connects the `diff_drive_controller` to an Arduino-based motor controller. It translates the wheel velocity targets from the controller into serial commands understood by the Arduino and processes encoder feedback in return. Originally adapted from a GitHub template [25], it simplifies the integration of ROS 2 with low-level microcontroller hardware, removing the need for custom command parsing.

4.4.3.5 Managing `cmd_vel` with `twist_mux`

To manage multiple sources of velocity commands in a predictable way, `twist_mux` was used in the ROS 2 control stack. Several input topics were defined in the `twist_mux.yaml` configuration file, including `cmd_vel_joy` and `cmd_vel` for the trajectory_generator function. Each velocity command was assigned a number indicating the priority, lower numbers translate to higher priority. The output of `twist_mux` was set to publish to `/diff_cont/cmd_vel_unstamped`, which is subscribed to by the `/diff_drive_controller`. This allows the system to switch between teleoperation and trajectory mode.

`twist_mux`: This node provides a way to prioritize certain topics from others, in this case prioritizing the topic our joystick sends its inputs to from other topics. It does this by accepting inputs from several topics, and by a priority list,

decides which to forward to our topic `diff_cont/cmd_vel_unstamped`, that `diff_drive_controller` uses to send velocity commands to the wheels.

4.4.3.6 Robot control using joystick

As part of the objective of robot control, focus was placed on achieving low-level control through the use of a joystick interface. This was accomplished by setting up a network in ROS 2 that publishes velocity commands via a topic to the `diff_drive_controller`. A test was first conducted to see if we could see the inputs from the controller, executed by the command line `ros2 topic echo /joy` in a terminal. This ensures that the input signals were correctly recieved on the `joy` topic. To convert these joystick inputs into the needed velocity command format, the `teleop_twist_joy` package was used. This package subscribes to the `joy` topic and publishes velocity commands in Twist message format to `cmd_vel_joy`. These commands are then routed through `twist_mux`, which outputs the result to `/diff_cont/cmd_vel_unstamped`. This setup allows control of the robot using the gamepad. Figure 4.28 illustrates the flow of the ROS 2 network.

`joy`: In order to control the robot with a joystick, ROS 2 provides a package `joy` that allows the operator to achieve this. The package includes a `joy_node` that reads the inputs of a gamepad and publishes a `Joy` message which contains the current state's of the buttons and axes [26]. This gives both the discrete values of the buttons and bumpers, and also the float values from -1 to +1 for both the joysticks and the triggers.

`teleop_twist_joy`: This package republishes the joystick inputs from the `joy` topic to a velocity command topic in the form of `geometry_msgs/msg/Twist` messages. Specifically, it subscribes to the `joy` topic and publishes to `cmd_vel_joy` [27]. This enables teleoperation of the robot without the need to manually convert raw axis values into linear and angular velocities. Since the joystick inputs are originally provided as axis values, and the `diff_drive_controller` expects velocity commands in Twist format, `teleop_twist_joy` serves as a convenient bridge between the two.

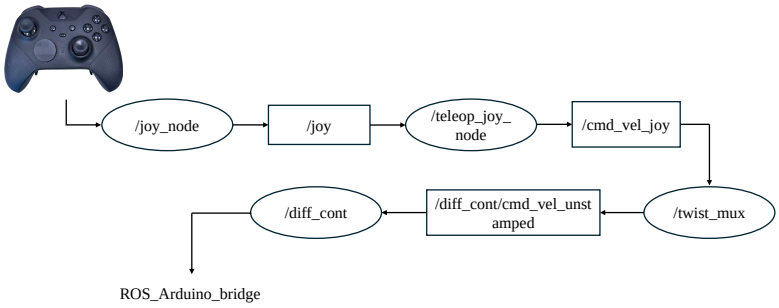

Figure 4.28: Communication path, from joystick to motor control

4.4.3.7 Robot control using `trajectory_generator`

In addition to teleoperation another control mode was implemented using coordinate-based navigation. This was handled through a custom ROS 2 package called `trajectory_generator`, which receives input from the waypoints topic. The input is a string of relative or absolute coordinates. Upon receiving the input, via the ROS 2 CLI using `ros2 topic pub`, the `trajectory_generator` node parses the coordinates and publishes goals sequentially. The robot's current position is tracked using encoder data, and each waypoint is approached using a combination of linear and angular motion.

`trajectory_generator`: This module enables robot control by specifying target positions of both absolute and relative coordinates. The `trajectory_genertor` node is executed by command lines in a terminal. For instance, the line to reach a absolute position ($x = 2.0, y = 1.0$), and then move an additional meter along the x-axis ($x = 3.0, y = 0.0$) would be: `ros2 topic pub /waypoints std_msgs/msg/String "data: 'absolute 2.0 1.0; relative 1.0 0.0'" -once`. Figure 4.29 shows the robots ideal movement. It also has a stop function, so by sending the command: `ros2 service call /stop_robot std_srvs/srv/SetBool` in the terminal the robot will stop.

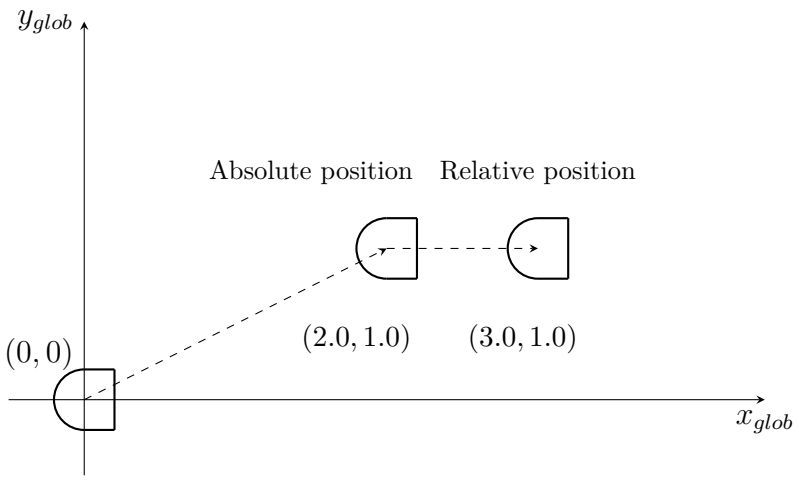

Figure 4.29: Visualization of robot's movement: first to an absolute position, then via a relative offset.

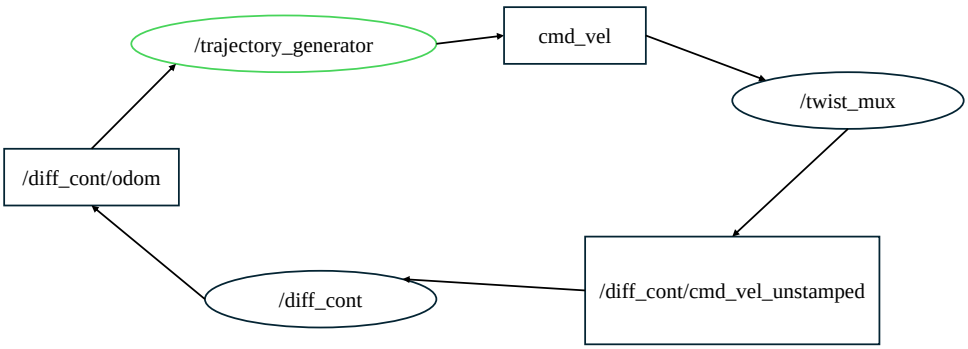

Figure 4.30: Node structure of the trajectory generator.

Comment: The green outline of the `/trajectory_generator` node indicates the start of the control loop.

4.4.4 System overview and node structure

The system was developed to control a differential-drive robot using ROS 2. First, hardware was interfaced via `ros2_control` and `diffdrive_arduino`. A `diff_drive_controller` was added to handle velocity commands and publish odometry. Encoder feedback was read using a `joint_state_broadcaster`. To support multiple command sources, `twist_mux` was integrated. Finally, a trajectory generator was connected to provide planned routes. Figures 4.31 and 4.32 highlights the ROS 2 node structure and a simplified view of the control system.

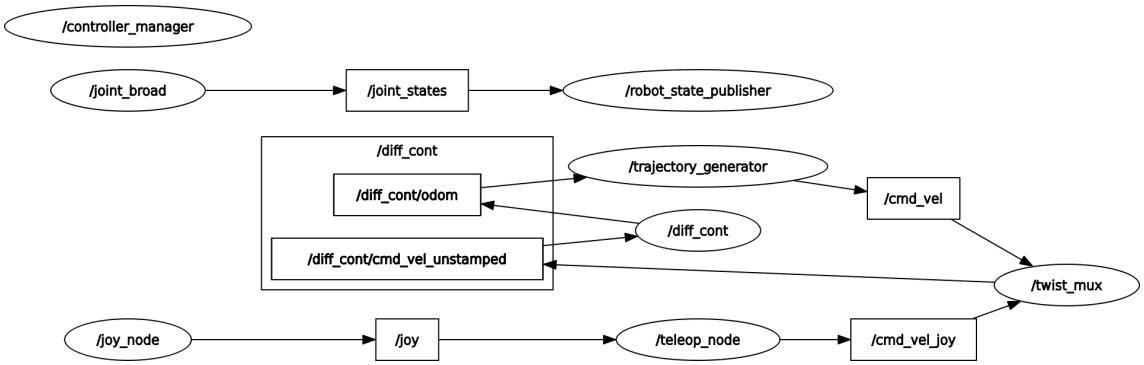

Figure 4.31: ROS rqt graph of entire system setup.

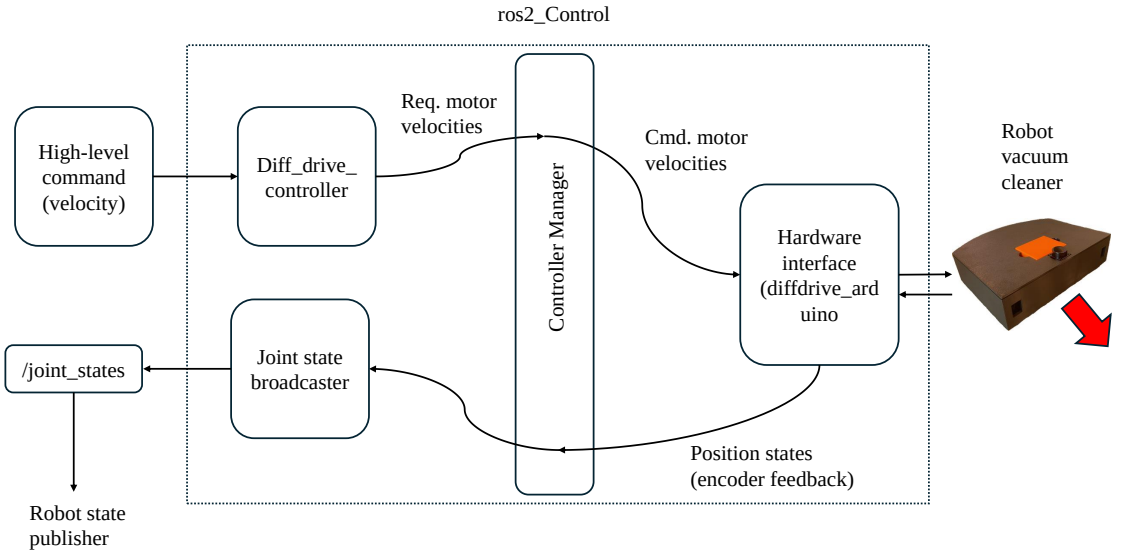

Figure 4.32: Entire ROS 2 control stack.

4.5 Validation

After the robot was designed and developed, a series of tests were conducted to validate the performance of the robot vacuum cleaner. The results of these tests are presented in Chapter 5.

4.5.1 Evaluation of cleaning mechanism

To evaluate the cleaning mechanism, two different tests were performed. For both tests, small paper pieces were placed at 5 cm intervals along a 1-meter distance. The paper pieces are heavier than the dust particles a robot vacuum is typically designed to suction, hence successful tests indicate that it will perform well under normal cleaning conditions. To determine whether the tests were successful, the percentage of paper pieces that were suctioned was calculated.

Suction performance

The first test was designed to evaluate the suction performance when the paper pieces were placed directly beneath the center of the suction inlet. A total of 20 pieces were placed along the 1-meter distance. This test was conducted three times, to ensure consistent results.

Brush performance

The second test evaluates the performance of the brush. It was conducted in the same way as the previous test, but with the paper pieces placed to the side, where the brush is located. This test was also conducted three times.

4.5.2 Trajectory following capabilities

This sections presents the tests of the robots ability to follow pre-defined trajectories.

Trajectory test

The robot vacuum cleaners capacity to follow a predesigned path was tested by driving in a straight line. Using the `trajectory_generator`, waypoints were placed to make the robot move to a point straight forward 1 meter away. Data was collected from the odometry readings and the robot was observed visually to determine how close it physically was to the goal.

Iterative controller parameter tuning

During the tuning process, it was noticed that the robot's turning behavior did not fully match the expected motion. To correct this, the wheel separation and wheel radius parameters were slightly adjusted based on observations from physical testing. These changes helped to improve the accuracy of the robot's turning and alignment with commanded trajectories. The value 0.29 m was optimal for the wheel separation and for the wheel radius the number 0.0325 m was deemed a good value

for optimal performance. To get the PD-controller working as smoothly as possible, different values for the controller parameters were tested. The tuning started by adjusting the proportional gain (K_P) to find a value that gave stable and responsive motion. After a suitable K_P was found, the derivative gain (K_D) was tested to reduce any overshooting or wobbling.

Star pattern test

A more complex path test was conducted by letting the robot move around in a star pattern to check its ability to turn accurately.

LiDAR readings

Since the LiDAR sensor is mounted on the robot vacuum cleaner and readings can be made, two LiDAR scans were performed under different conditions, to visualize how this could be implemented in the future.

5

Results

This chapter presents the results of the tests, regarding both hardware and software. The first part focuses on the cleaning mechanism of the robot vacuum cleaner. The second part covers the software results, including the robot's ability to follow a pre-determined trajectory. Lastly, the LiDAR readings are presented.

5.1 Cleaning performance tests

The suction performance test is shown in figure 5.1, with results presented in table 5.1.

Figure 5.1: Suction performance test

Table 5.1: Suction performance results

	Test 1	**Test 2**	**Test 3**
Initial pieces	20	20	20
Suctioned pieces	17	18	19
Percentage suctioned	85%	90%	95%

The brush performance test is shown in figure 5.2, with results presented in table 5.2.

Figure 5.2: Brush performance test

Table 5.2: Brush performance results

	Test 1	**Test 2**	**Test 3**
Initial pieces	20	20	20
Suctioned pieces	14	15	13
Percentage suctioned	70%	75%	65%

5.2 Trajectory following capabilities

This section presents the results of the robot vacuum cleaner's ability to follow trajectories.

5.2.1 Trajectory test

The collected odometry data is visualized in graphs shown in figures 5.3, 5.4, 5.5 and 5.6. Figure 5.3 shows the robot vacuum cleaners position during the first one meter test drive. Figure 5.4 shows the actual linear and angular speed compared to the computed. Figure 5.5 shows the robots position during the one meter test with corrective angular motion. Figure 5.6 shows the actual linear and angular speed compared to the computed, with corrective angular motion.

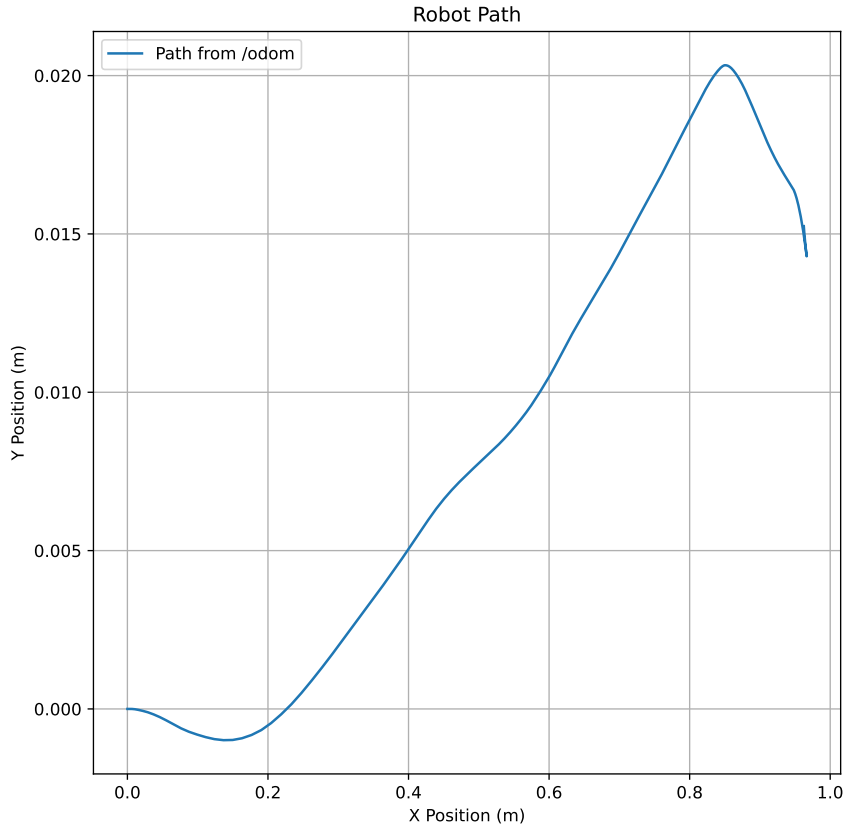

Figure 5.3: First path data collected during testing of the robot vacuum cleaner.

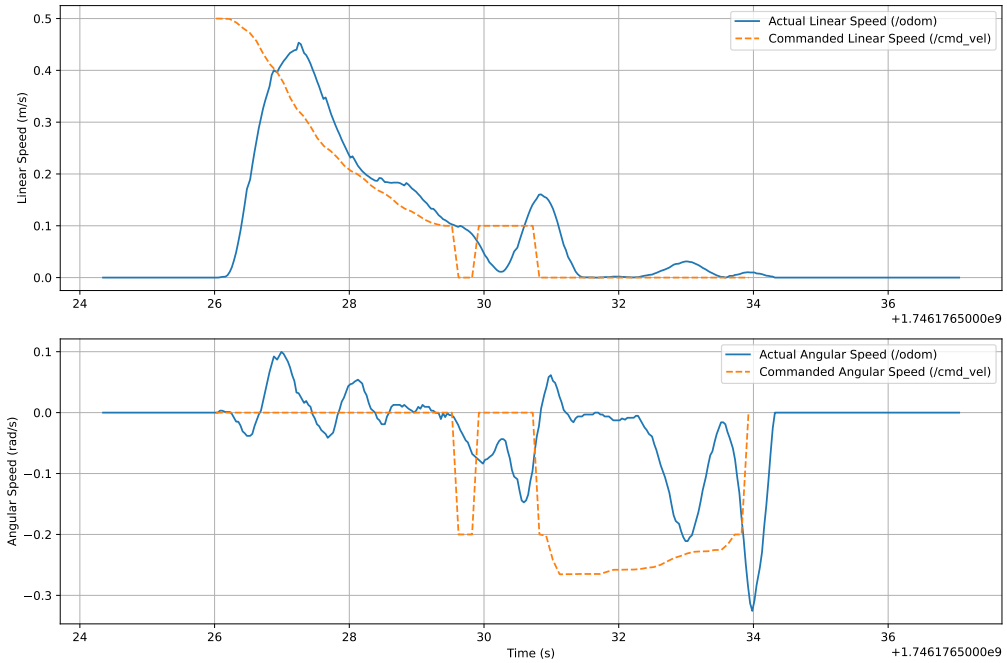

Figure 5.4: First speed data collected during testing of the robot vacuum cleaner.

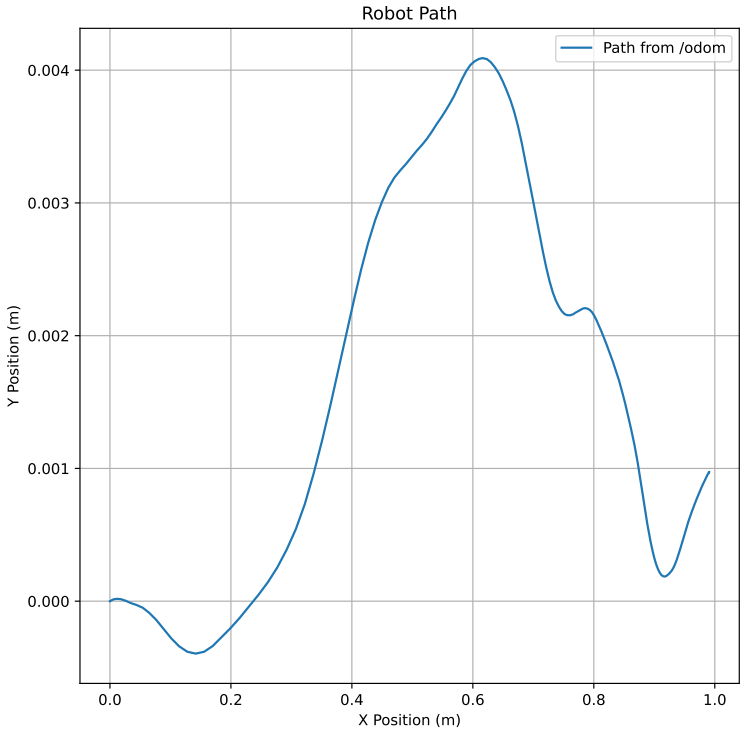

Figure 5.5: Robot path with corrective angular motion.

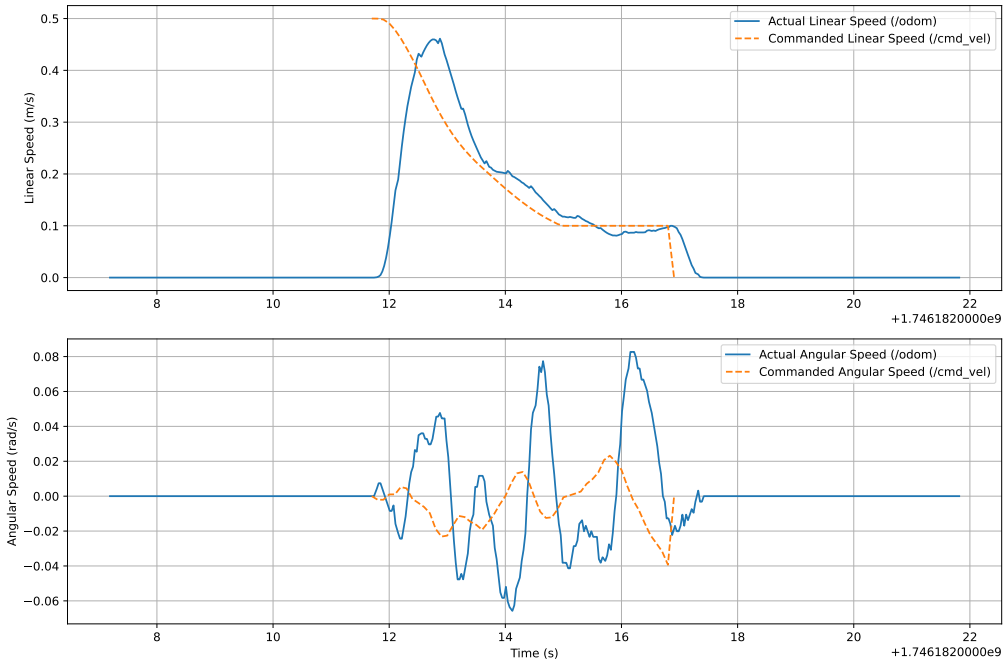

Figure 5.6: Linear and angular speed during testing of robot with corrective angular motion.

5.2.2 Iterative controller parameter tuning

Figures 5.7 and 5.8 show the best results when tuning the K_p value, at 0.6. The best results from the test with both K_p and K_d values can be seen in figures 5.9 and 5.10 with K_p=0.6 and K_d=0.4, which gave a good balance between quick response and smooth behavior. A full overview of the tests and graphs showing the performance with other values of K_p are shown in Appendix, figures C.1 to C.12. Additional graphs with other values of K_p and K_d are shown in Appendix, figures C.13 to C.20.

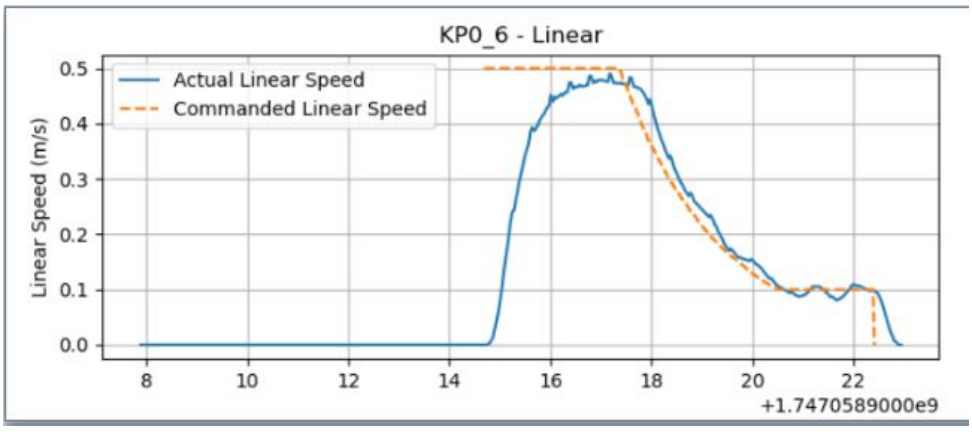

Figure 5.7: Graph with linear speed, with $K_p = 0.6$

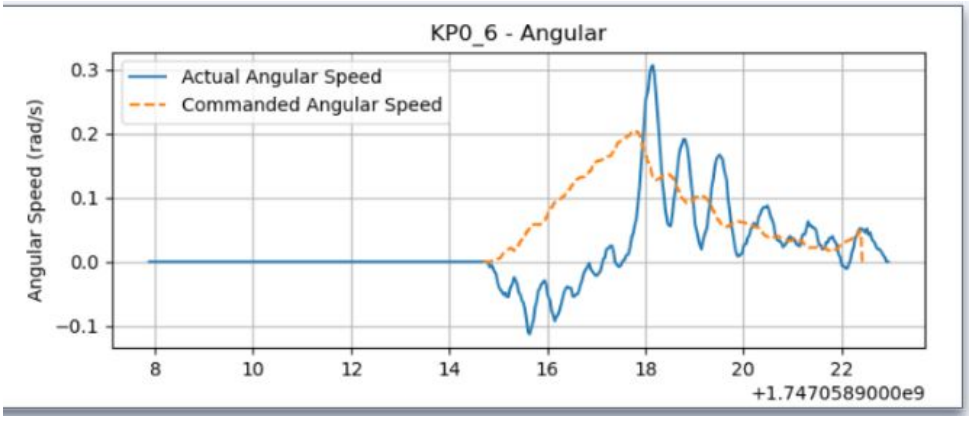

Figure 5.8: Graph with angular speed, with $K_p = 0.6$

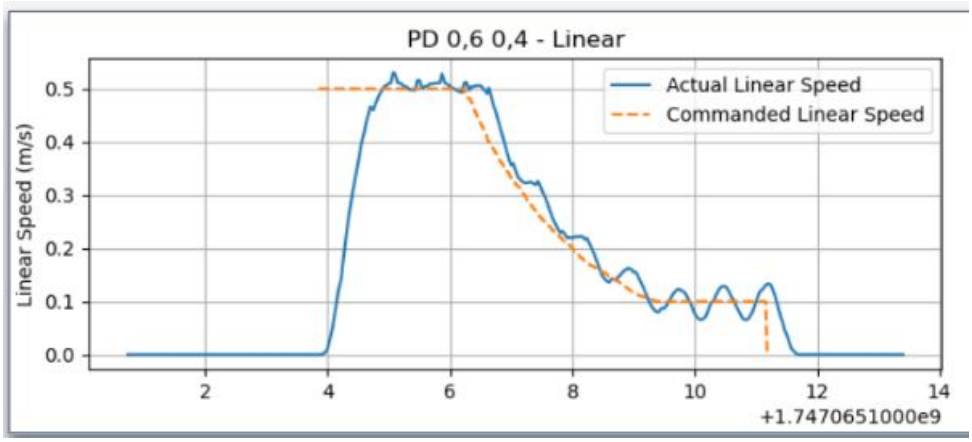

Figure 5.9: Graph with linear speed, with $K_p = 0.6$ and $K_d = 0.4$

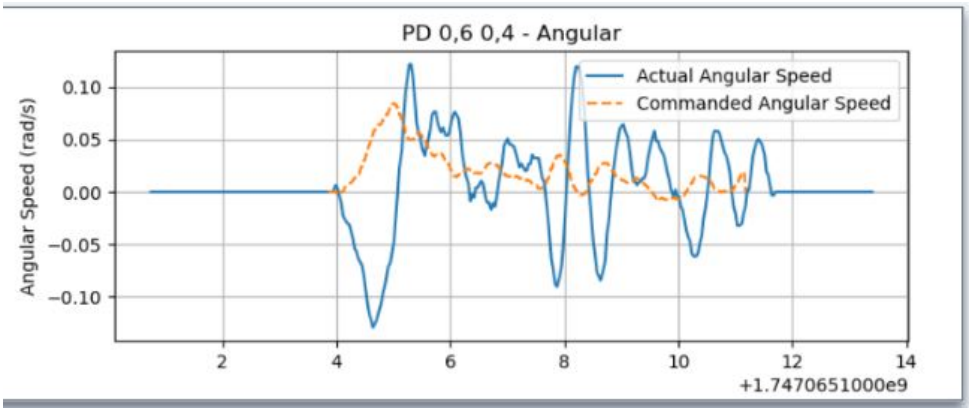

Figure 5.10: Graph with angular speed, with $K_p = 0.6$ and $K_d = 0.4$

5.2.3 Star pattern test

The data from the test can be seen in figure 5.11 and figure 5.12

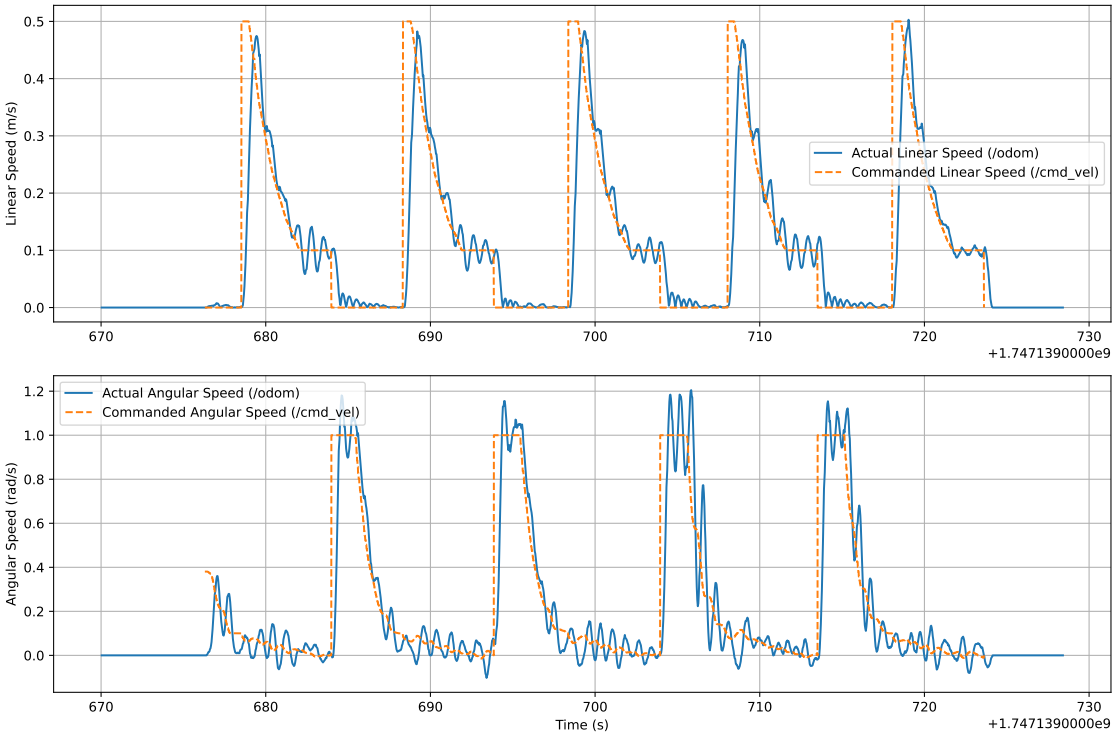

Figure 5.11: Speed data from the robot vacuum cleaner during star pattern test.

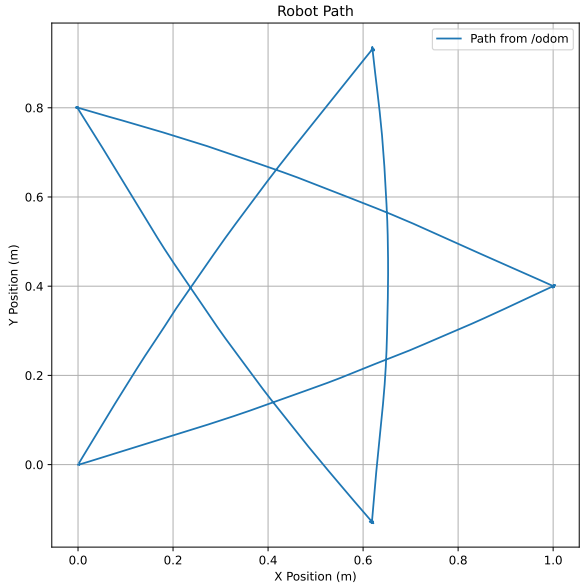

Figure 5.12: Path data from the robot vacuum cleaner during star pattern test.

5.2.4 Final parameters

From the tests, the most optimal values of K_p and K_d parameters are shown in table 5.3, alongside wheel separation, wheel radius and encoder counts per revolution. The encoder counts was not changed during the testing phase, it remained the same value from the initial calibration.

Table 5.3: Final parameters for the robot vacuum cleaner. K_p and K_d represents parameter values for the PD-regulator.

Wheel separation	Wheel radius	K_p	K_d	Encoder counts per revolution
0.29	0.0325	0.6	0.3	682

5.3 LiDAR readings

In figure 5.13, the robot vacuum cleaner was positioned near a wall with a nearby chair. Figure 5.14 shows a scan taken while the robot was placed on the floor surrounded by chair legs.

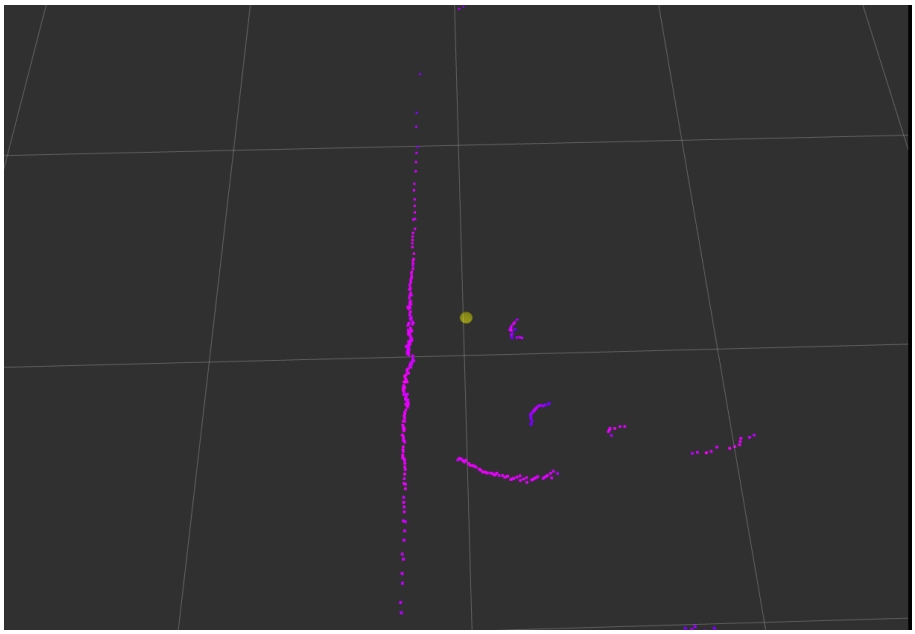

Figure 5.13: LiDAR reading with robot near a wall

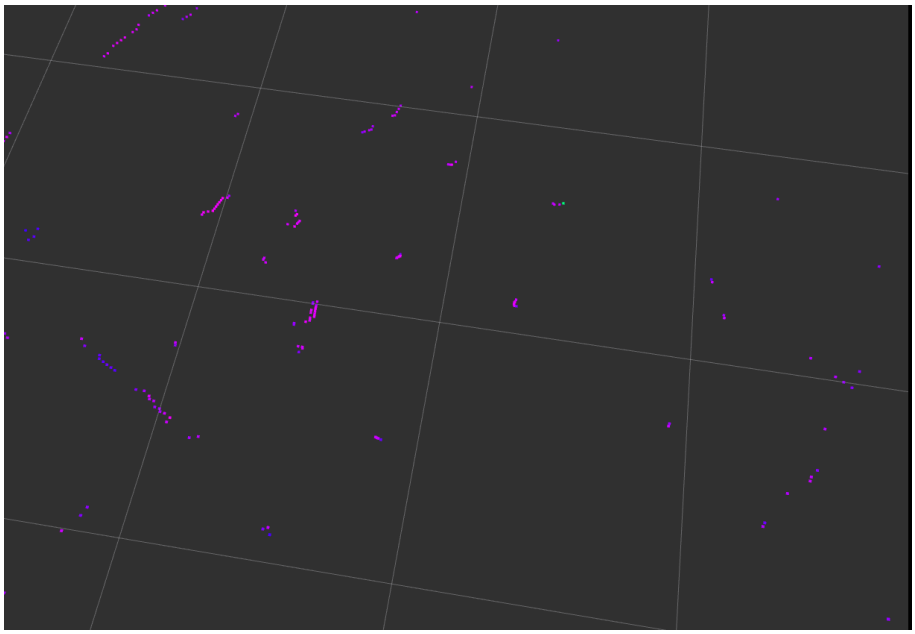

Figure 5.14: LiDAR reading with robot on the floor

6
Discussion

The following discussion explores the results form both hardware and software perspectives. Outcomes are analyzed in relation to project goals, highlighting success and challenges. Finally, possible improvements and future development paths are discussed.

6.1 Cleaning mechanism results

When testing the suction performance by placing paper pieces directly beneath the inlet, the three tests resulted in an average collection rate of 90%. This result is positive given that the paper pieces were larger than the typical particles the system is designed to collect, and that the robot vacuum cleaner passed over them only once. Most robot vacuum cleaners often revisits the same areas more than once, making sure that particles are not missed [28].

In the second test, where the paper pieces where placed directly in front of the brush, an average of 70% of the pieces were collected in the three tests. The reason for this slightly lower result, compared to test 1, is because of inconsistency in brush performance. The purpose of the brush is to sweep particles at the edge of the robot vacuum cleaner towards the suction inlet, located in the middle of the robots front. This functions in most cases, but occasionally the brush sweeps paper pieces either too short or too far, causing them to end up too far from the suction inlet.

Additionally, the suction is not evenly distributed across the entire inlet. This results in paper pieces being missed more often, since the actual inlet becomes much smaller, marked green in figure 6.1. It also creates a blind spot between the suction inlet and the brush, where the robot vacuum cleaner is less effective at collecting particles. The areas of the suction inlet that generate minimal suction are highlighted in red in figure 6.1.

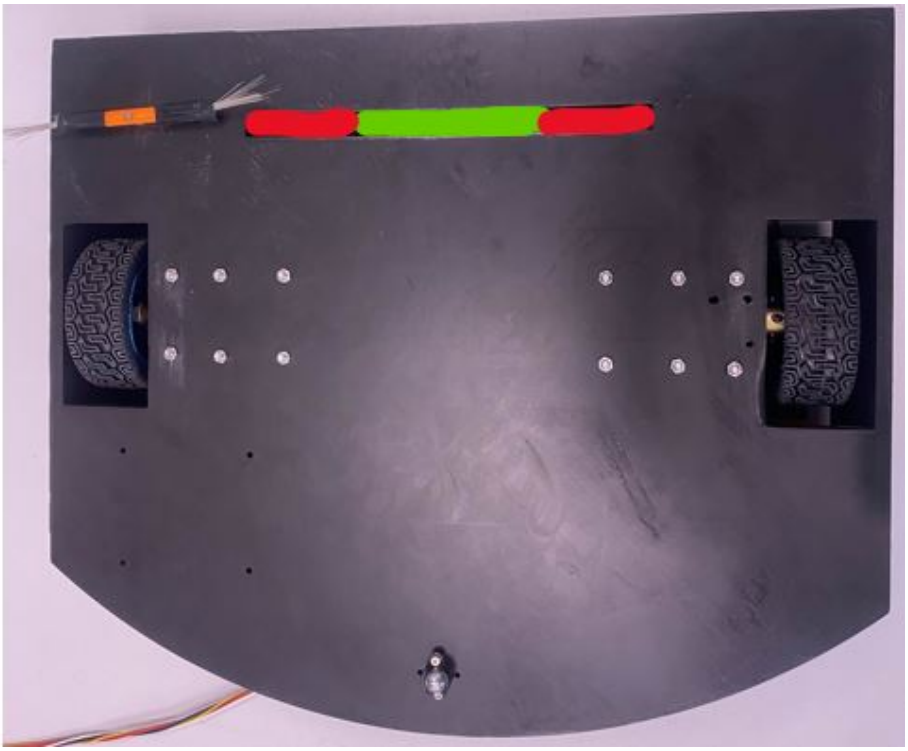

Figure 6.1: Showcasing flow in suction inlet

6.2 Flow analysis

When designing the chassis, much thought went into optimizing the flow of the fan. One of the considerations was how high up from the bottom of the chassis the fan's suction inlet should be. The number decided on was 30 mm. In hindsight, this distance could have been significantly smaller. The fan spins and generates a negative pressure. The fact that the distance from the spinning fan to the chassis bottom being that high can induce turbulence in the flow. This can result in loss of pressure which in turn worsen the suction performance. The 30 mm can be seen in figure 6.2 and could have been reduced to at least 15 mm.

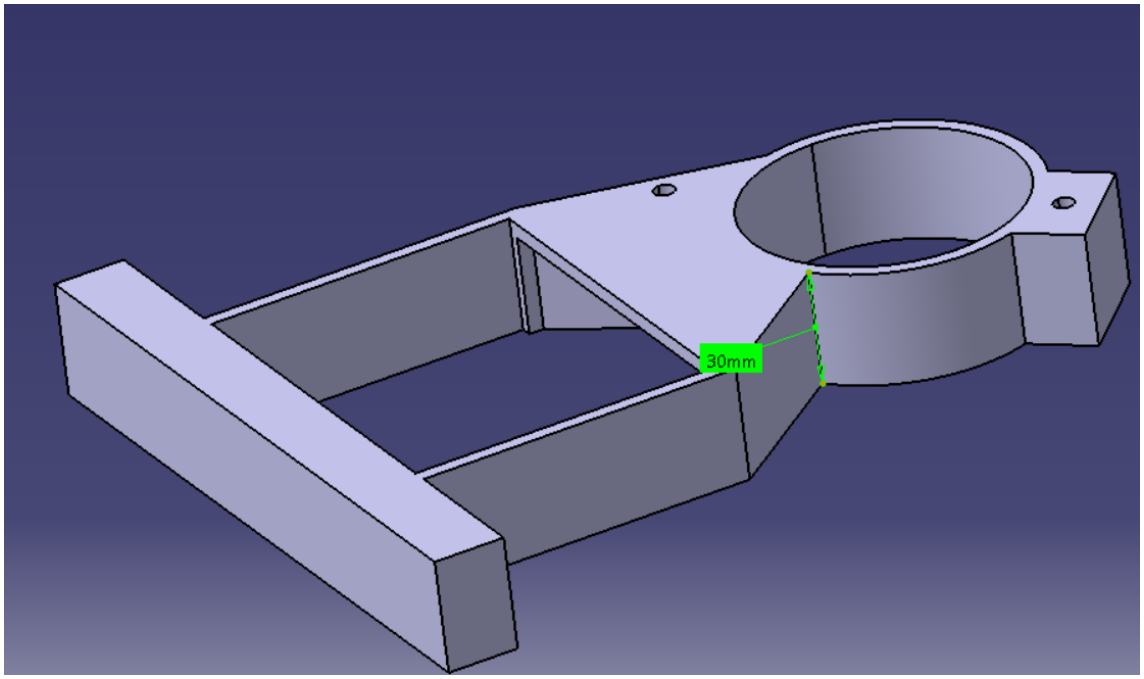

Figure 6.2: Fan mount and airflow tunnel

Another fundamental design issue is how big the different inlets and outlets are. In theory, when the flow passes from a large hole to a smaller hole, the velocity through the smaller hole will increase according to Bernoullis pinciple [29]. Figure 6.3 shows how air flows through the different compartments. When the air flows from compartment 1 to 2 the velocity of the air will decrease due to the air flowing to a bigger compartment. It will then increase when it flows from compartment 2 to 3. The fundamental issue is that after thereafter, air flow will decrease again when exiting compartment 3 which is the inlet for the suction mechanism of the dust. To compensate for that, additional flow parts were designed, see figures 6.4 and 6.5. These parts increased the flow velocity from compartment 3, leading to a slight improvement in suction performance. However, the improvement was not significant enough to justify reducing the inlet size, as this would compromise the robot's ability to pick up dust that is not directly underneath it.

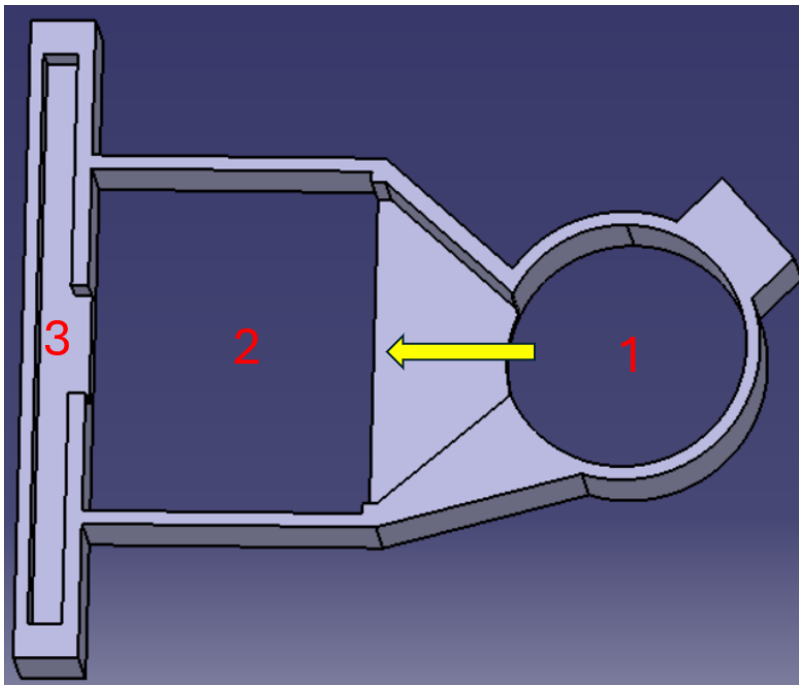

Figure 6.3: Overview of flow direction

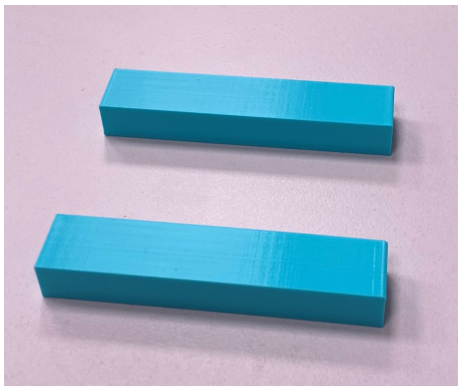

Figure 6.4: Flow parts

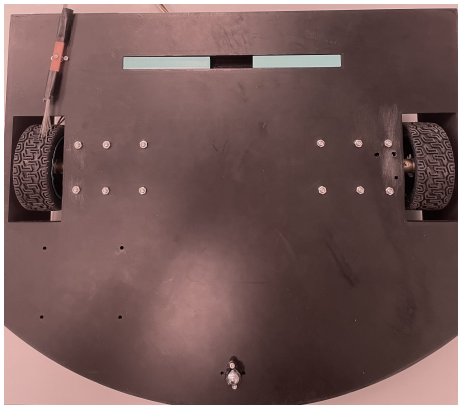

Figure 6.5: Flow parts mounted

The most efficient way to optimize airflow would have been to enlarge the opening between compartments 2 and 3 and match it with a similarly sized inlet. This approach would preserve the beneficial effect of increased velocity while avoiding the performance drop caused by an overly small inlet.

Another critical factor in optimizing the flow is to minimize any air leakage. Actions were taken, in specific to ensure that the fan was placed in such a way where no air could leak out and worsen the suction performance.

6.3 Design iteration

During the 3D-printing of the chassis, multiple challenges were encountered. One example is the cover of the whole robot vacuum cleaner, which failed once due to not using the brim setting in the Prusa Slicer, which made it lose grip, resulting in a slight bend in the cover. This led to the cover alignment being slightly off when attached to the chassis. The first prototype is seen in figures 6.6 and 6.7.

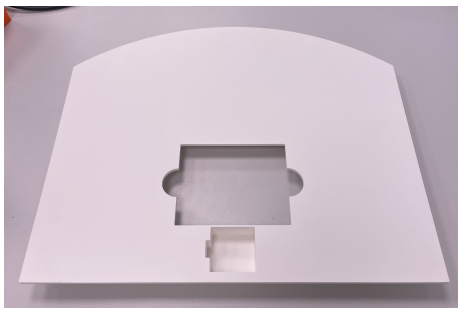

Figure 6.6: Cover front side prototype 1

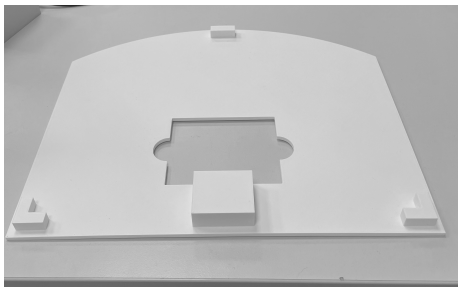

Figure 6.7: Cover rear side prototype 2

The second attempt on printing the cover failed as well. The print ran out of filament due to a shortage in the CASE-Lab, which resulted in an even more bent cover. This attempt is shown in figures 6.8 and 6.9.

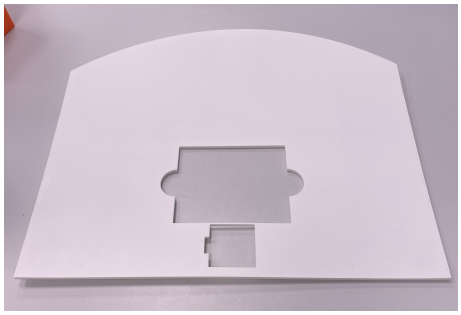

Figure 6.8: Cover front side prototype 2

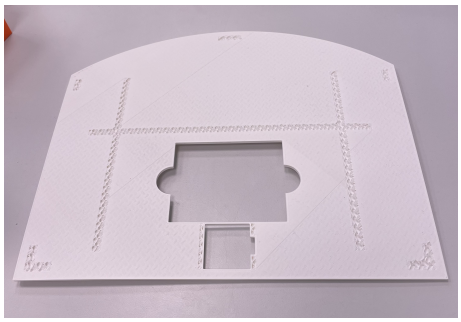

Figure 6.9: Cover rear side prototype 2

For the third and final attempt, structural beams were introduced in the design, making the cover more stiff. The infill setting was set to grid, which creates crossed lines, making it strong in both the x- and y direction. A brim was also used around the edges, making sure the print stuck to the printing board. Additionally, Addnorth PETG was used as material, instead of the previously used PLA. Even though PETG is the harder material, PLA was earlier used to print the chassis because of its lower cost. With all these changes, the print was successful and resulted in the final cover presented in chapter 4.

There are other examples of how the team managed to optimize the design of different parts in order to achieve the desired functions of the robot. A noticeable example of the optimization of such design are the motor mounts. Figure 6.10 shows how the design evolved through four iterations resulting in the final version all the way to the right.

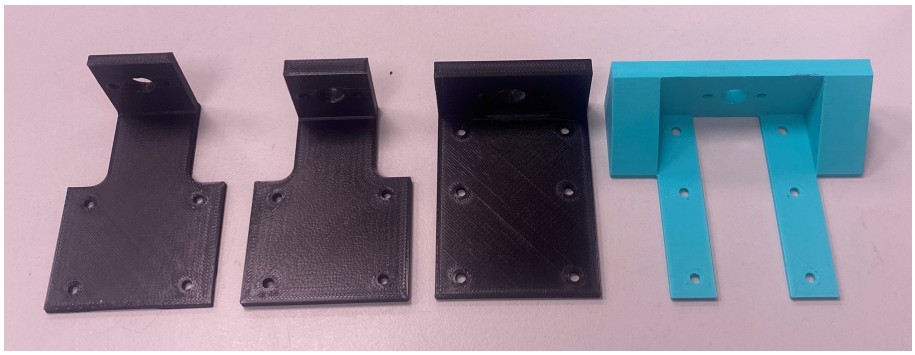

Figure 6.10: Different motor mounts

To determine the appropriate chassis size, infill, printing settings, and printer selection, an initial test chassis was printed. The first attempt failed, but it provided valuable insights that answered the key design questions. The failed chassis was too small and had wrong printing settings which resulted in defects, mainly the infill settings were wrong. Luckily only the bottom side of the chassis was printed when the defects were noticed. It was still enough to answer question on what went wrong. The test chassis is shown in figure 6.11

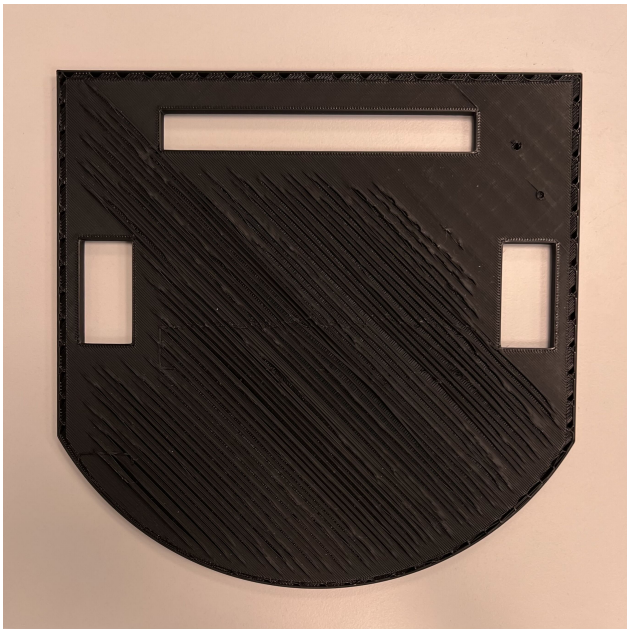

Figure 6.11: Print attempt of test chassis

The printing errors and some small design flaws resulted in wasted material which is normal when working on a project. Though the team had to redo a number of

parts in order for everything to be assembled in the best way, there was no need to redo the main chassis after the test chassis was tried. The chassis was by far the biggest part that had to be 3D-printed. It weighed around 700 g, and making the slightest mistake or error meant it had to be remade, but luckily everything went to plan. This was achieved by paying close attention to every detail from design to the printing process. The final chassis is presented in chapter 4.

6.4 Following a predefined trajectory

Initial tests revealed that the robot tended to drift off its intended path without self correcting its orientation. The problem was in the trajectory generator, which only issued angular corrections if the difference between the robot's current orientation and the target angle exceeded a predefined threshold. As a result, the robot executed either linear or angular motion, but rarely both at the same time. This is visible in the speed data, where linear and angular commands alternate, causing the robot to deviate from its path. The alternation of speed and angular compensation caused a jerking motion throughout the robots path where it needed to stop each time it found it self outside of the allowed angle toward the goal. The difference between the commanded and actual angular speed, especially at lower speeds, suggested either noise issues with the PD controller, or possibly some form of compensation from other components in the system. For larger command values, as seen in the star pattern test, figure 5.3), the deviations are not as noticeable. This suggests that the issue primarily witnessed at smaller speed levels.

To improve tracking accuracy, a modification was made to allow continuous corrective angular motion, even for small deviations, simultaneously as driving forward. This change led to noticeable improvement, where both linear and angular motion are coordinated more effectively. The resulting path shows reduced drift and a trajectory that more closely matches the expected motion. These improvements are still constrained by the limitations of using encoders as the feedback source. Since encoder based odometry depends on parameters such as wheel radius and wheel separation, any error in these values will affect the accuracy of position and velocity estimates. To minimize this, physical measurements were taken from the robot, and other tests were made by marking two square points on the ground and evaluating the robot's ability to move consistently between them.

Initial tests showed that while the distance traveled was fairly accurate, the robot tended to overshoot turns, indicating an overestimation in angular displacement. To correct this, the wheel separation parameter was slightly reduced, which improved angular tracking during testing. Once a more accurate motion model was achieved, PD controller tuning was carried out. A series of tests were conducted with varying K_p and K_d values. Starting with adjustments to the proportional gain (K_d), a value of 0.6 provided the best balance between responsiveness and stability. Following this, the derivative gain (K_d) was tuned to reduce overshoot, and a value of 0.4 was found to work effectively.

To validate the complete control and path execution pipeline, a star pattern test was performed, where the robot was given a list of way points to follow in a five pointed star configuration. The result demonstrates that the system is capable of accepting a sequence of goals and executing precise path following behavior with acceptable tracking error.

6.5 Future improvements

In this section, possible future improvements are discussed.

6.5.1 Sensors

While the current robot vacuum cleaner relies on movement by trajectory inputs, insightful steps have been made towards the implementation of integrating a more advanced navigation system, using both infrared sensors and a LiDAR sensor. In the chassis, holes were cut out for two infrared distance sensors. With the limited time and complexity, integrating these sensors was de-prioritized. A future improvement is to mount the sensors and incorporate them with the current navigation system, allowing the robot vacuum cleaner to recognize obstacles.

The current system relies solely on wheel encoders to estimate the robot's position and orientation. While sufficient for basic movement, this approach is prone to drift over time, especially due to wheel slip or uneven surfaces. As a result, the robot cannot reliably track its exact location during longer or more complex movements. A useful improvement would be to integrate an Inertial Measurement Unit (IMU) and apply sensor fusion techniques, such as a Kalman filter, to combine encoder and IMU data. This would help correct for individual sensor errors and significantly improve the accuracy and stability of the robot's motion tracking.

A LiDAR sensor has successfully been mounted on the robot vacuum cleaner, and initial tests confirm that reliable data can be read. The readings in figures 5.13 and 5.14 show promising results, detecting walls, chairs and other obstacles nearby. By implementing these results in the software, an advanced obstacle avoidance system can be developed. However, integration with the system is complex and requires a lot of time. Future developments would include the integration of LiDAR, as well as the development of a Simultaneous Localization and Mapping (SLAM) algorithm. This would enable the robot to create real time maps of its surroundings, significantly improving its efficiency and coverage. It would know where in the room it is located, as well as where its already been, resulting in no unnecessary overlap and advanced object detection.

6.5.2 Additional brushes

The robot vacuum cleaner has a brush located in one of its front corners. In test 2, when paper pieces were placed directly in front of the brush resulted in an average collection rate of 70 %. However, the side without a brush consistently fails to collect any particles, as the suction inlet does not extend that far. Adding a brush to this corner would help solve the issue of missed dust particles. It would also resolve the problem encountered in test 2, where paper pieces got swept a bit too far. A second brush would counteract this problem, sweeping the dust particle back towards the suction inlet.

6.5.3 Climbing mechanism

A potential future goal could be enabling the robot to overcome thresholds through targeted software and hardware solutions. The robot's software can be shared and modified as needed to support this functionality. Additionally, attachments may be added to the chassis, either by gluing or carefully drilling holes. When drilling, extra caution is required to avoid damaging the material, especially since the chassis is not printed with 100 % infill, making it more susceptible to structural damage.

6.5.4 Software improvement

There are a plenty of areas where the current software system could be improved. One of the limitations at this stage is battery monitoring and power management. During testing, there were multiple times where the robot began to behave unexpectedly due to low power levels. Right now the system lacks real time awareness of its battery status. Adding a module for battery level monitoring, as well as a low power warning, would prevent power-related failures.

Another important area for improvement is the system's inability to adapt its velocity to the situation. Currently the robot does not regulate its speed when approaching turns, obstacles, or performing precision maneuvers, however it is also not able to detect any of these situations as of right now.

When it comes to perception, the robot is already equipped with a functional LiDAR sensor, but time constraints during the project limited the development of features such as obstacle detection and avoidance. Adding features like real time processing of LiDAR data would enable the robot to navigate around obstacles, allowing for better path-planning and room navigation.

7
Conclusion

The aim with this project was to design and develop a functioning robot vacuum cleaner, equipped with basic movement functionality and a working cleaning mechanism. The strategy of developing the robot vacuum cleaner allowed high flexibility, allowing the implementation of both essential and wanted features. The prototype was successful, having both a functional cleaning mechanism and a working movement and control system. It also provides a solid foundation in both hardware and software, offering strong potential for future development in climbing mechanism and navigation.

During the project, the group learned valuable lessons in exploring the complexities of robotics. The project provided meaningful understanding of the challenges in robotics, combining both mechanical design and electronics, with software engineering to create a functional system.

References

[1] M. R. Future, *Robotic vacuum cleaner market research report*, Accessed: 2025-03-09, 2025. [Online]. Available: https://www.marketresearchfuture.com/reports/robotic-vacuum-cleaner-market-18855.

[2] G. V. Research, *U.s. household vacuum cleaner market report*, Accessed: 2025-03-09, 2025. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/us-household-vacuum-cleaner-market-report.

[3] P. Hartwell. "Sensor breakdown: How robot vacuums navigate and clean." Accessed: 2025-04-29. (Sep. 2022), [Online]. Available: https://www.therobotreport.com/sensor-breakdown-how-robot-vacuums-navigate-and-clean/.

[4] S. K. Malu and J. Majumdar, "Kinematics, localization and control of differential drive mobile robot," *Global Journal of Researches in Engineering: F Electrical and Electronics Engineering*, vol. 14, no. 1, pp. 1–7, 2014, Accessed: 2025-05-12. [Online]. Available: https://globaljournals.org/GJRE_Volume14/1-Kinematics-Localization-and-Control.pdf.

[5] Open Source Robotics Foundation. "Robot operating system (ros)." Accessed: 2025-05-20. (2024), [Online]. Available: https://www.ros.org/.

[6] Open Robotics. "Tutorials." Accessed: 2025-05-08. (2025), [Online]. Available: https://docs.ros.org/en/humble/Tutorials.html.

[7] Open Source Robotics Foundation, *ROS 2 Documentation: Humble Hawksbill*, Accessed: 2025-05-13, 2022. [Online]. Available: https://docs.ros.org/en/humble/index.html.

[8] T.-Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 11, pp. 1125–1136, 1998. DOI: 10.1109/71.736752.

[9] Protolabs Network. "What is fdm (fused deposition modeling) 3d printing?" Accessed: 2025-05-20. (2024), [Online]. Available: https://www.hubs.com/knowledge-base/what-is-fdm-3d-printing/.

[10] Dassault Systèmes. "What are the most common materials used for 3d printing?" Accessed: 2025-05-22. (2025), [Online]. Available: https://www.3ds.com/make/solutions/blog/most-common-materials-3d-printing.

[11] CASE Lab, *Preparation material - prusa 3d printers*, https://help.prusa3d.com/tag/mk4s, Last revised: 2025-02-11, 2025.

[12] M. Electronics. “Single board computers.” Accessed: 2025-03-26. (2025), [Online]. Available: https://www.mouser.se/c/embedded-solutions/computing/single-board-computers/.

[13] J. Schneider and I. Smalley. “What is a microcontroller?” Accessed: 2025-04-23. (2024), [Online]. Available: https://www.ibm.com/think/topics/microcontroller.

[14] J. Schneider. “Microcontrollers vs. microprocessors: What’s the difference?” Accessed: 2025-05-05. (2024), [Online]. Available: https://www.ibm.com/think/topics/microcontroller-vs-microprocessor.

[15] STMicroelectronics, *L298 h-bridge dual full bridge driver datasheet*, Accessed: 2025-03-11, 2013. [Online]. Available: https://www.electrokit.com/upload/product/41016/41016218/L298_H_Bridge.pdf.

[16] SEBO Canada. “Making sense of vacuum cleaner technical specs.” Accessed: 2025-05-13. (2025), [Online]. Available: https://sebocanada.ca/blogs/sebo-resource-centre/making-sense-of-vacuum-cleaner-technical-specs.

[17] Wikipedia, *Lidar*, fetched 2025-03-04. [Online]. Available: https://en.wikipedia.org/wiki/Lidar.

[18] L. Moussi, *Why does lidar have a specific wavelength? part.1*, fetched 2025-03-04, 2020. [Online]. Available: https://www.yellowscan.com/knowledge/why-does-lidar-have-a-specific-wavelength/.

[19] ECOVACS. “Round vs square robot vacuum: Which one is better?” Accessed: 2025-05-13. (2024), [Online]. Available: https://www.ecovacs.com/us/blog/round-vs-square-robot-vacuum.

[20] Analog Devices. “Uart: A hardware communication protocol.” Accessed: 2025-05-13, Analog Devices. (2020), [Online]. Available: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html.

[21] J. Newans, *Articubot$_o$ne*, Oct. 8, 2024. [Online]. Available: https://github.com/joshnewans/articubot_one.

[22] M. Ferguson, J. Newans, W. Gramlich, Fibird, and E. Nikulin, *Ros_arduino_bridge*, May 21, 2024. [Online]. Available: https://github.com/joshnewans/ros_arduino_bridge.

[23] Y. Ye and et al., “ROS2 real-time performance optimization and evaluation,” en, *Chin. J. Mech. Eng.*, vol. 36, no. 1, Dec. 2023.

[24] ros2_control Development Team. “Welcome to the ros2_control documentation - jazzy!” Accessed: 2025-04-23. (2025), [Online]. Available: https://control.ros.org/jazzy/index.html.

[25] J. Newans, *Joshnewans/diffdrive_arduino*, Jan. 28, 2024. [Online]. Available: https://github.com/joshnewans/diffdrive_arduino.

[26] M. Quigley, B. Gerkey, K. Watts, and B. Gassend, *Joy – ros package*, Available at: https://index.ros.org/p/joy/. Last updated: 2024-12-04. Accessed: 2025-05-13, 2024.

[27] A. Hendrix, M. Ferguson, D. Ash, and B. Gerkey, *Teleop_twist_joy – ros package*, Available at: https://index.ros.org/p/teleop_twist_joy/#humble. Last updated: 2024-09-09. Accessed: 2025-05-13, 2024.

[28] The Straits Times. "Hassle-free cleaning: Smart robot vacuum cleans own mop pads, revisits dirty areas." Accessed: 2025-05-08. (2023), [Online]. Available: https://www.straitstimes.com/life/home-design/hassle-free-cleaning-smart-robot-vacuum-cleans-own-mop-pads-revisits-dirty-areas-dreamebot-l20-ultra-dasher.

[29] Khan Academy. "What is bernoulli's equation?" Accessed: 2025-05-13. (2025), [Online]. Available: https://www.khanacademy.org/science/properties-of-matter-essentials/x56fba1647ecc4dbb:how-does-a-cricket-bowler-make-the-ball-swing/x56fba1647ecc4dbb:how-does-a-cricket-bowler-make-the-ball-swing-lesson/a/what-is-bernoullis-equation.

A

Hardware design

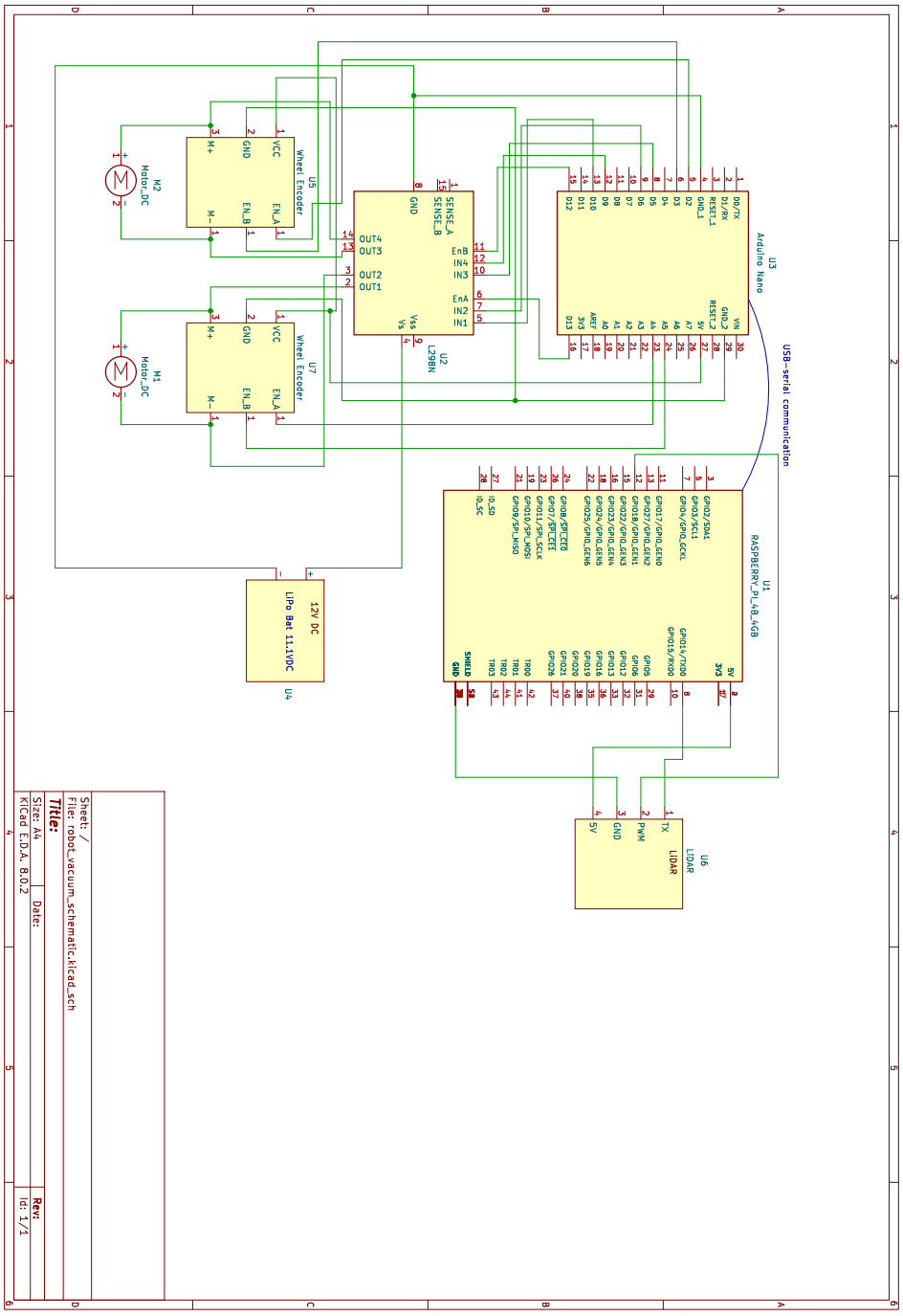

Figure A.1: Circuit diagram

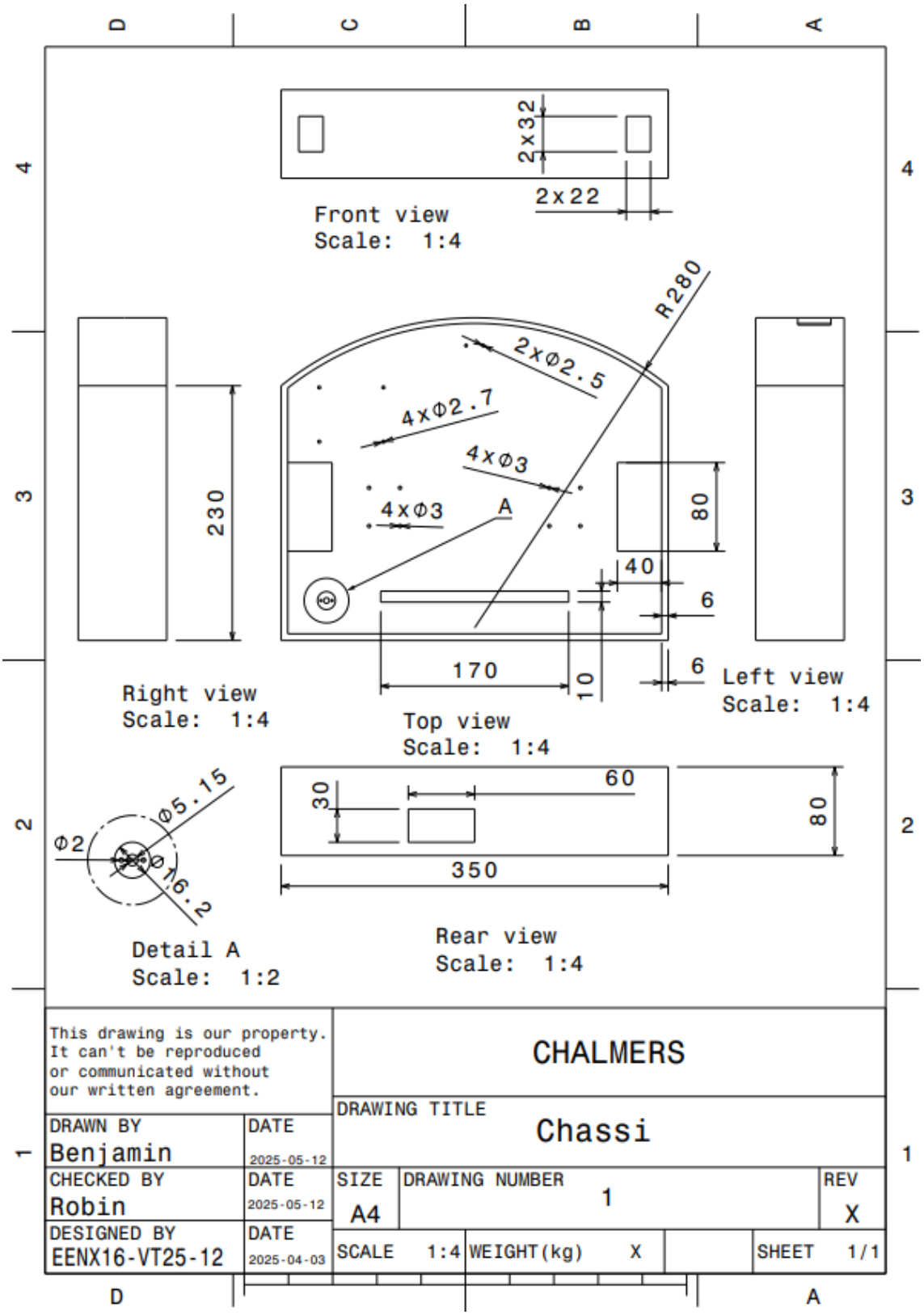

Figure A.2: Drawing of main chassis

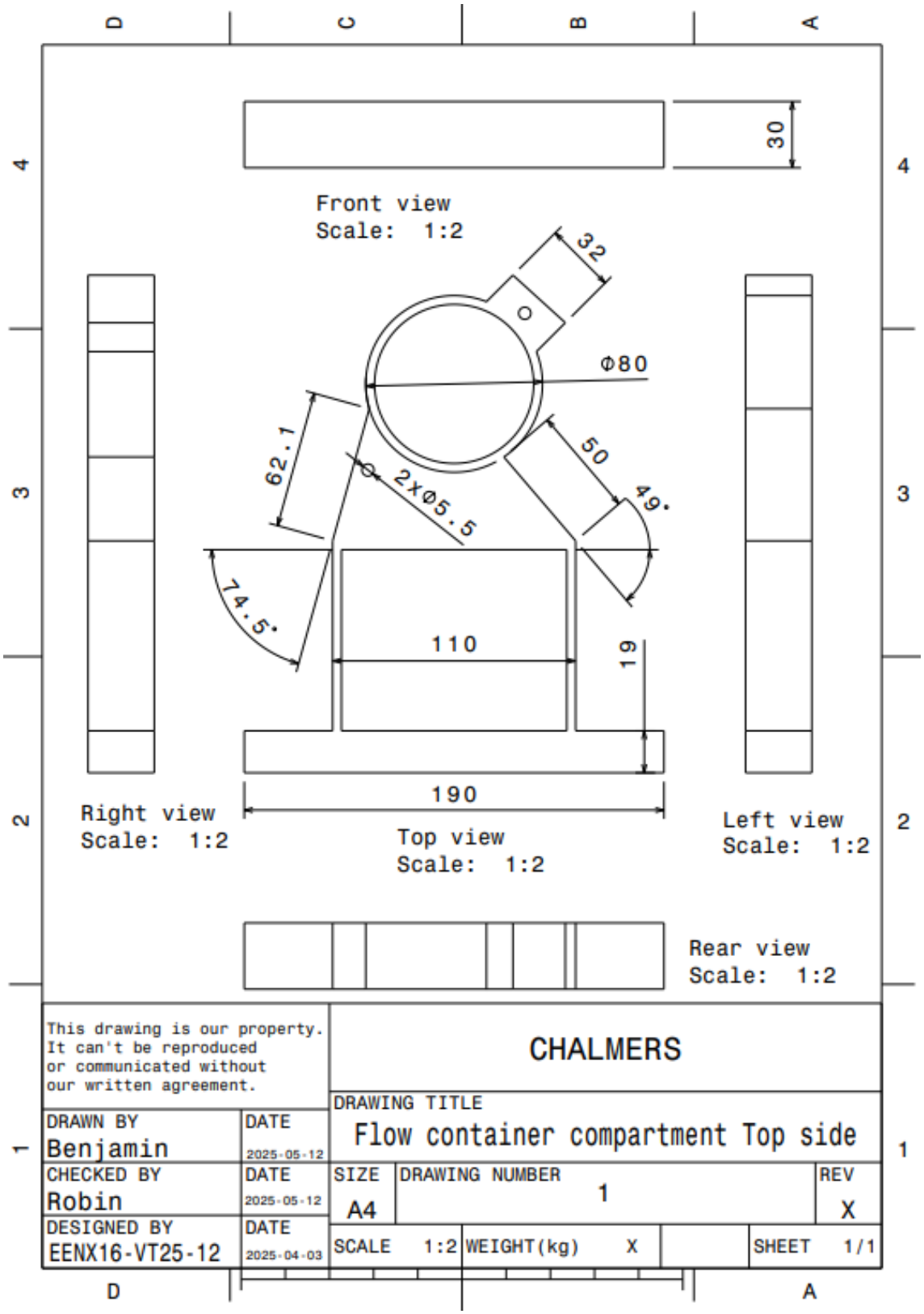

Figure A.3: Drawing of fan mount and airflow tunnel Top side

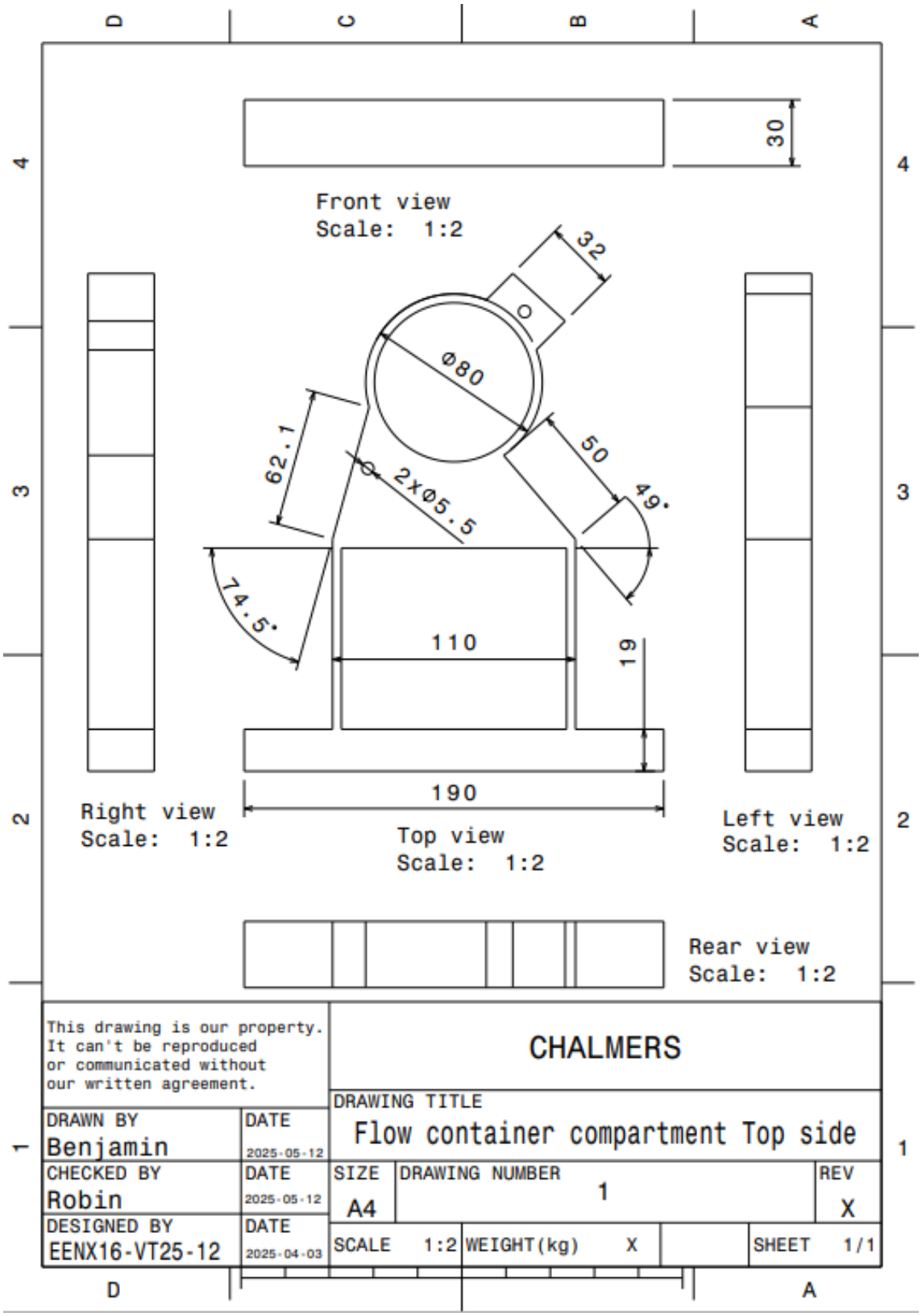

Figure A.4: Drawing of fan mount and airflow tunnel Bottom side

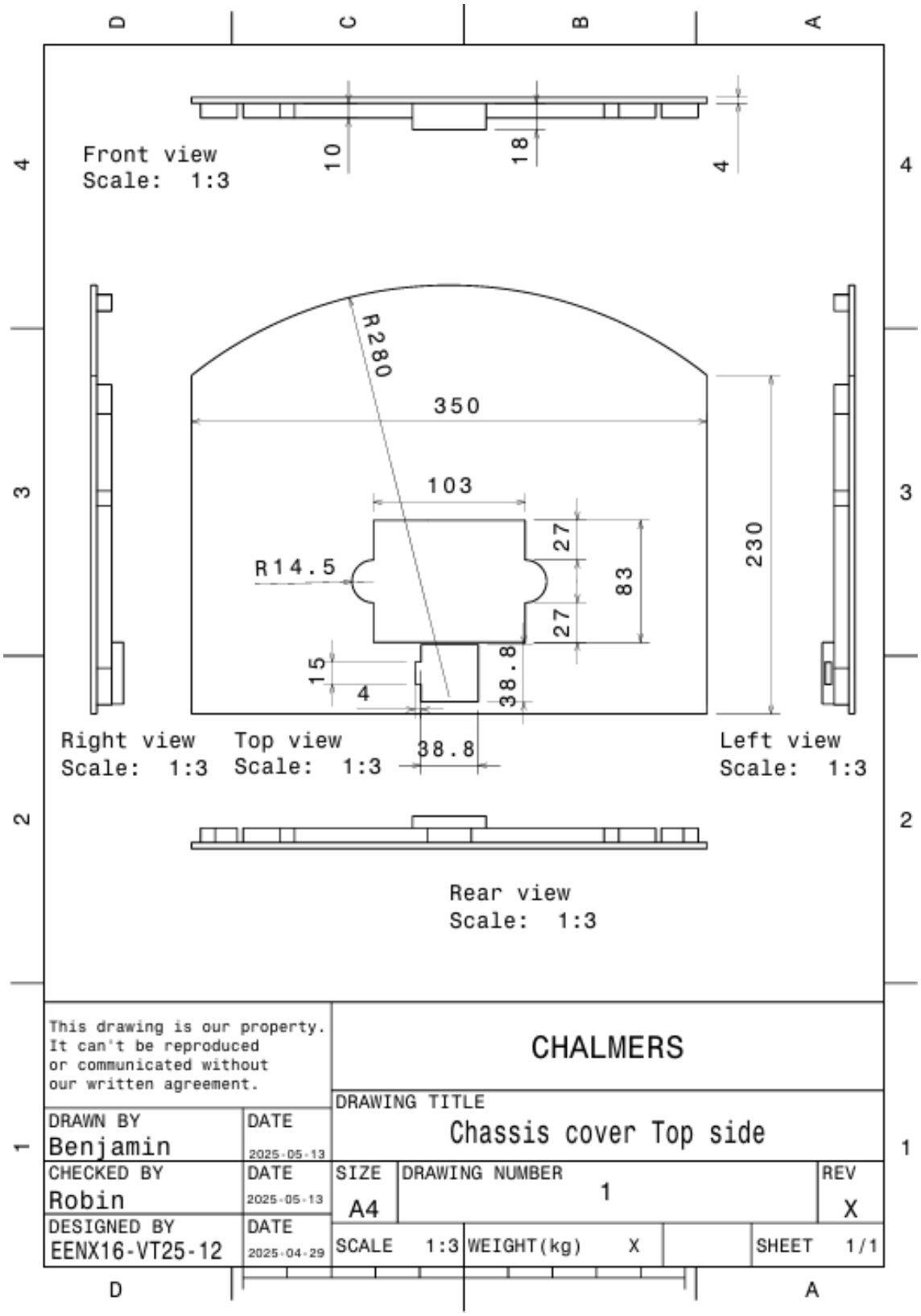

Figure A.5: Drawing of chassis cover Top side

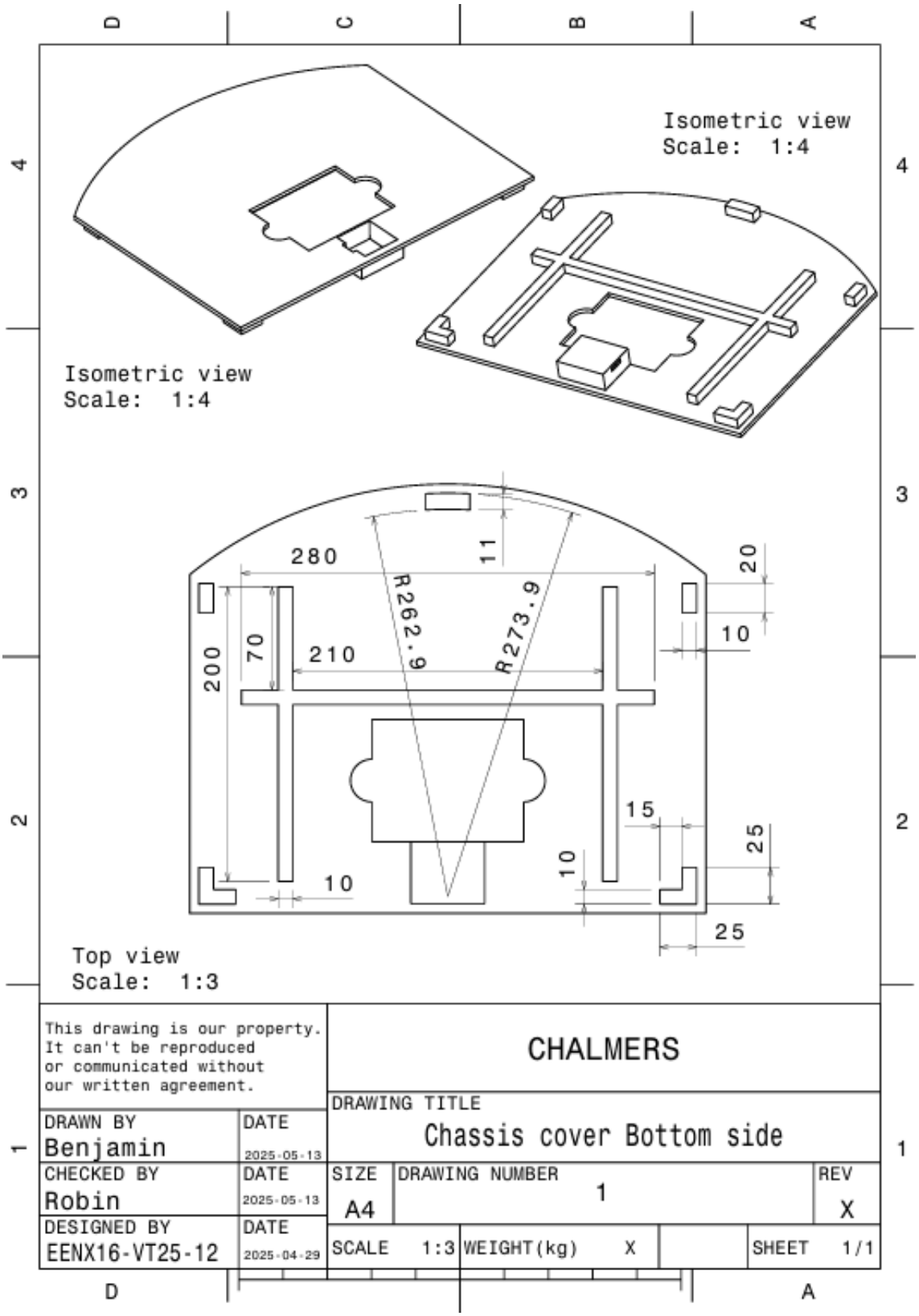

Figure A.6: Drawing of chassis cover Bottom side

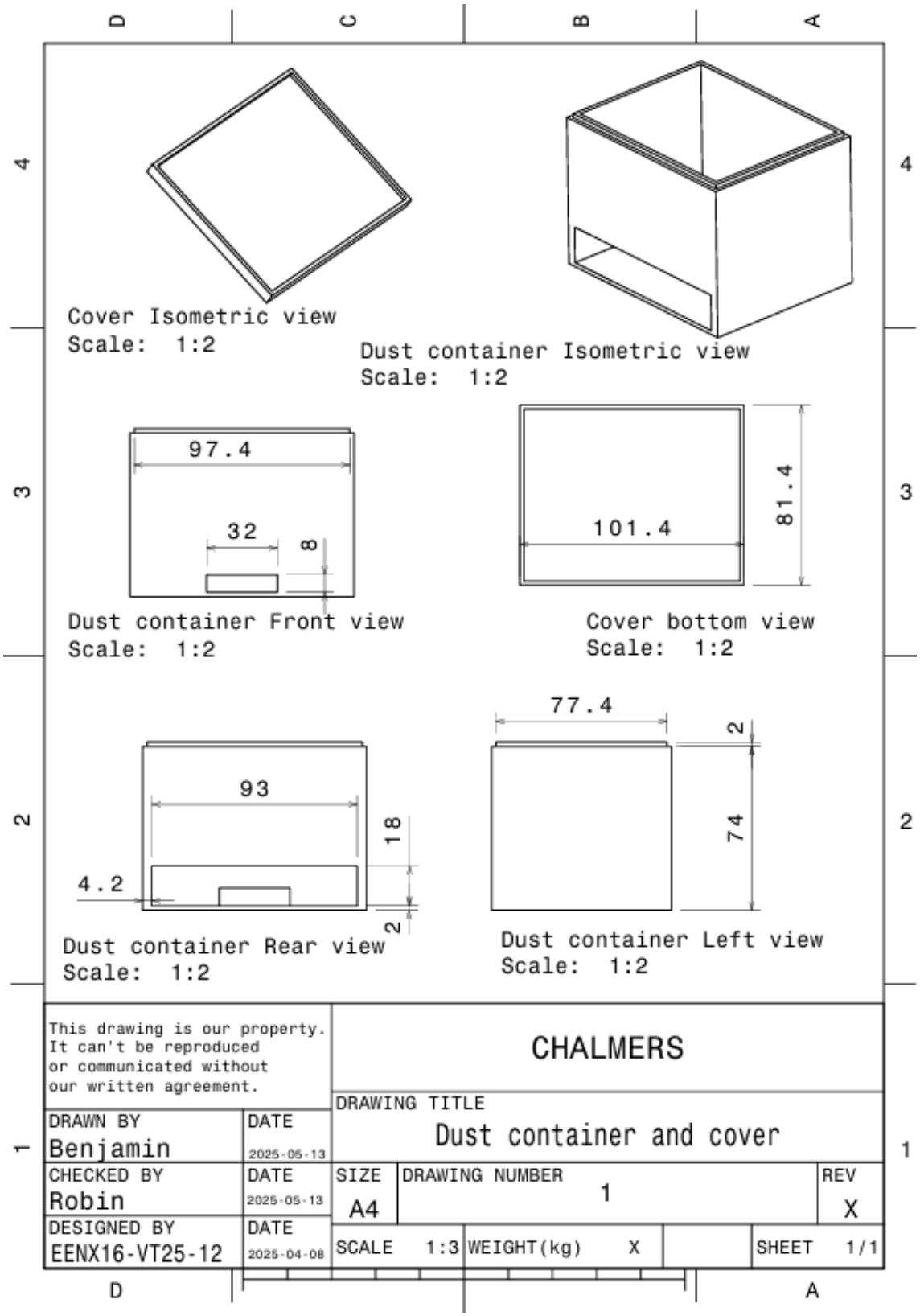

Figure A.7: Drawing of dust compartment and cover

B

Hardware specifications

Table B.1: Specifications of RS PRO Brushed Geared DC Motor

Specification	**Value**
Motor Type	Brushed Geared DC Motor
Power Consumption	0.394 W
Operating Voltage	12 V DC
Torque	7 mNm
Speed (No Load)	1550 rpm
Shaft Diameter	3 mm

Table B.2: Specifications of Sanyo Denki San Ace B97 Series Blower

Specification	**Value**
Supply voltage	12 V DC
Maximum current	3.4 A
Air Flow	56.8 cfm
Fan speed	6850 rpm

Table B.3: Specifications of the Garosa 12V Encoder Gear Motor

Specification	**Value**
Voltage	12 V
Speed	500 RPM
Encoder Type	Magnetic

Table B.4: Specifications of the LemonRC LiPo Battery

Specification	**Value**
Voltage	11.1 V
Capacity	1600 mAh

Table B.5: Specifications of the L298N Motor Driver

Specification	**Value**
Model	L298N
Voltage Range	5–35 V
Logic Level Voltage	5 V
Max Output Current	2 A per channel (peak)
Number of Channels	2 (Dual H-Bridge)
Integrated 5V Regulator	Yes

Table B.6: Specifications of the LM2596 Buck Converter

Specification	**Value**
Model	LM2596
Input Voltage Range	4–40 V
Output Voltage Range	1.25–37 V (adjustable)
Max Output Current	2 A (continuous), 3 A (peak)

C

Graphs from results

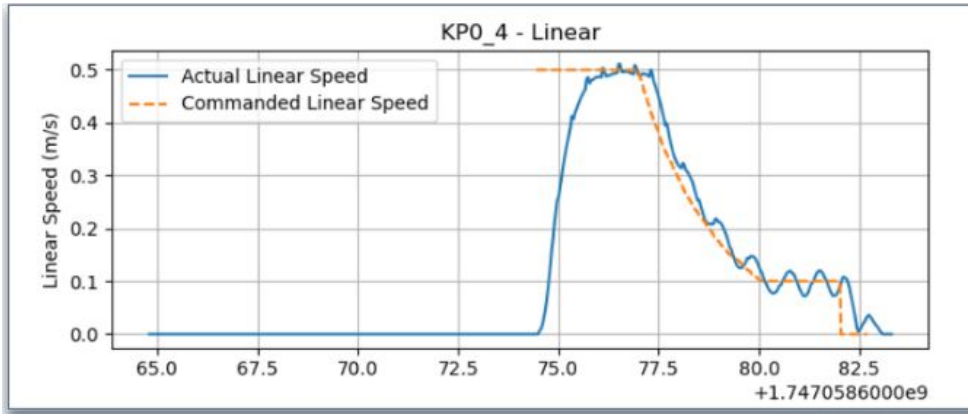

Figure C.1: Linear graph with $K_P = 0.4$

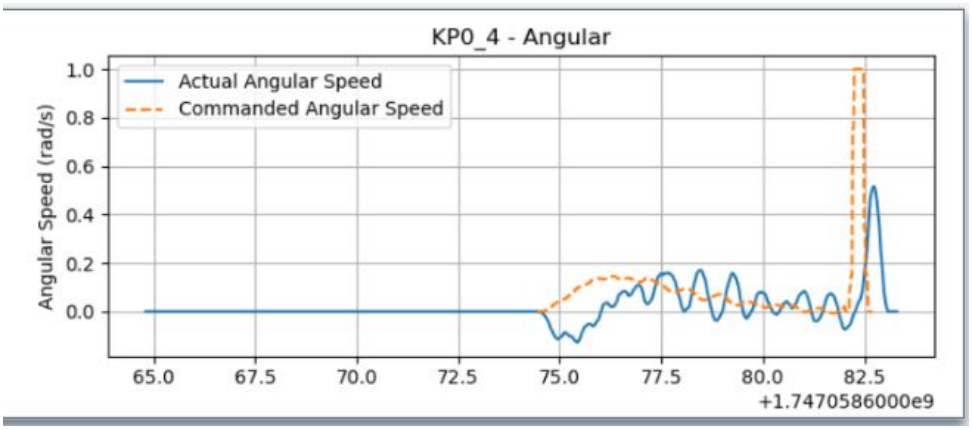

Figure C.2: Angular graph with $K_P = 0.4$

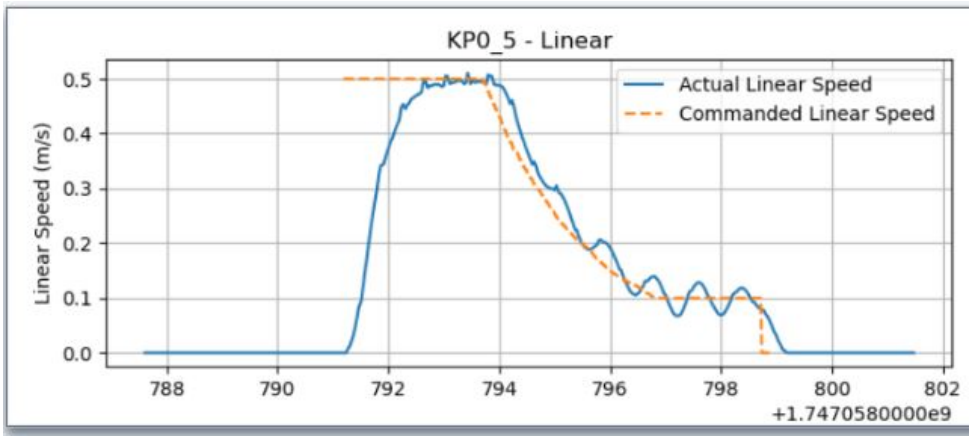

Figure C.3: Linear graph with $K_P = 0.5$

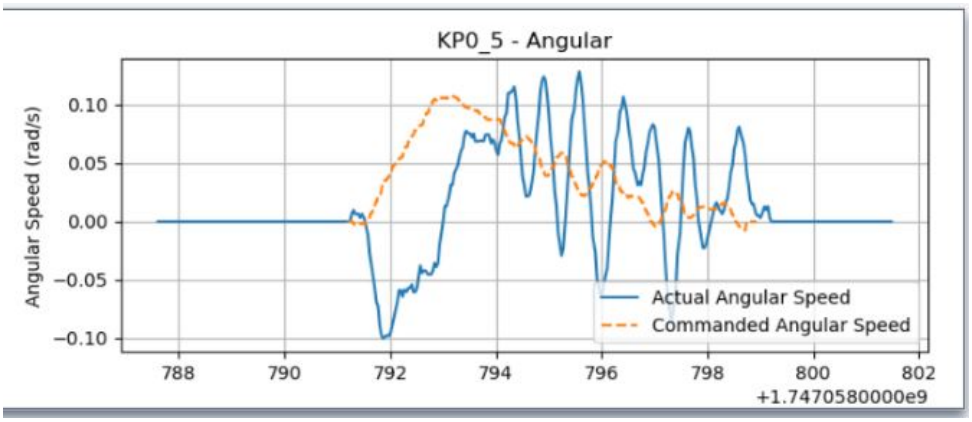

Figure C.4: Angular graph with $K_P = 0.5$

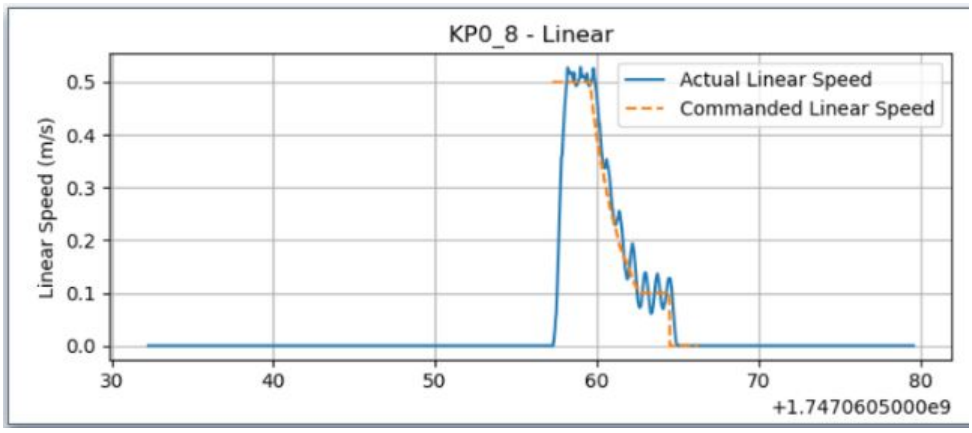

Figure C.5: Linear graph with $K_P = 0.8$

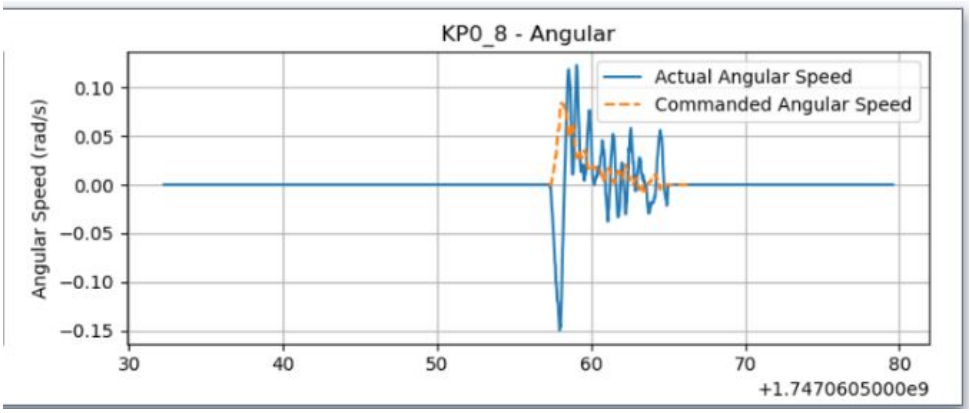

Figure C.6: Angular graph with $K_P = 0.8$

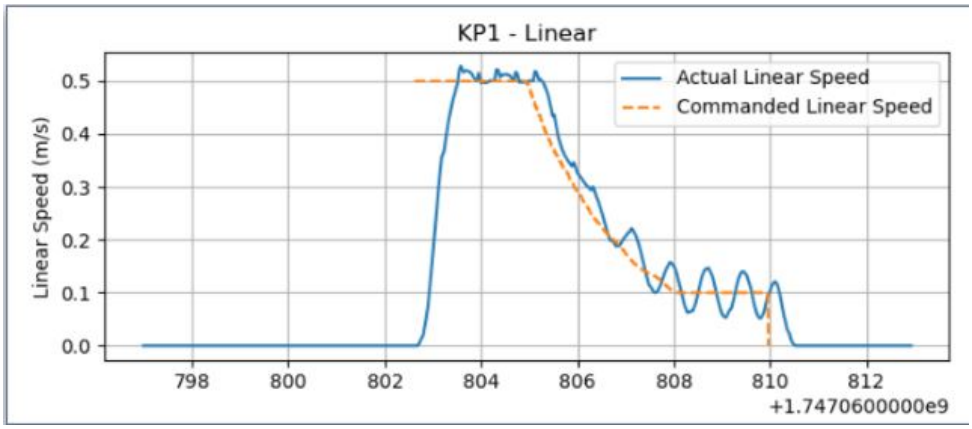

Figure C.7: Linear graph with $K_P = 1$

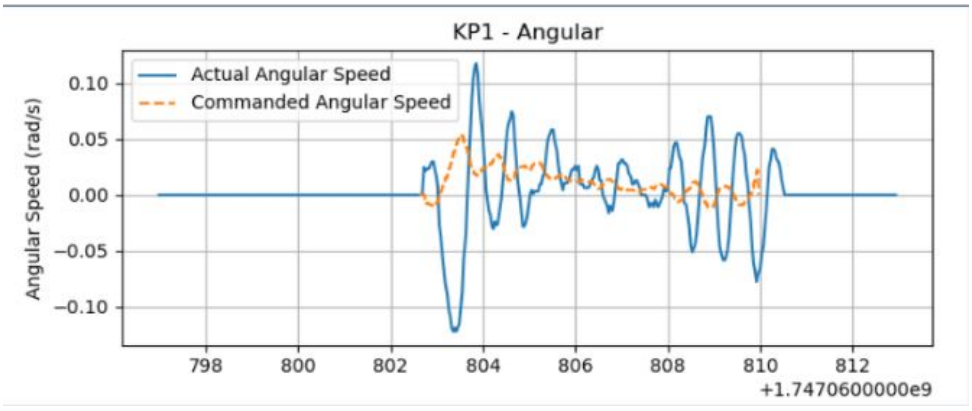

Figure C.8: Angular graph with $K_P = 1$

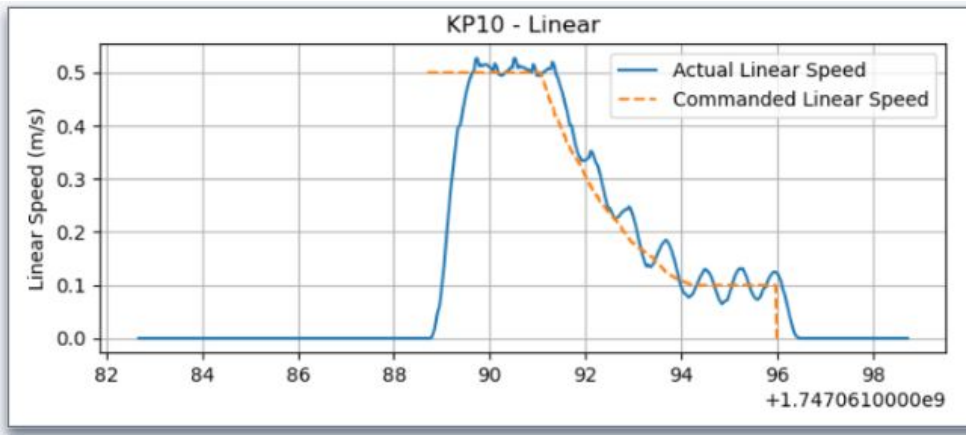

Figure C.9: Linear graph with $K_P = 10$

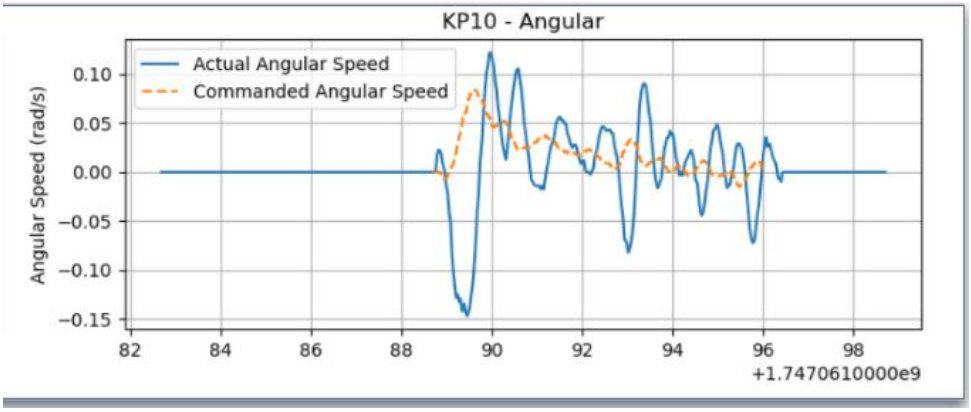

Figure C.10: Angular graph with $K_P = 10$

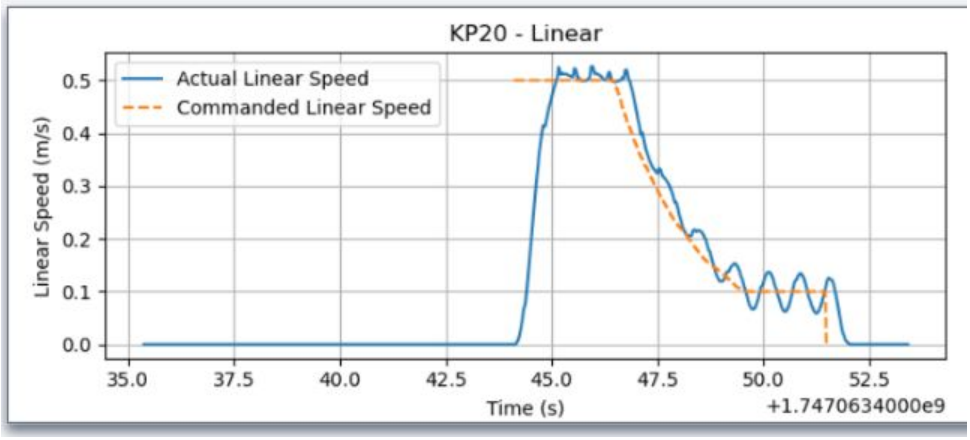

Figure C.11: Linear graph with $K_P = 20$

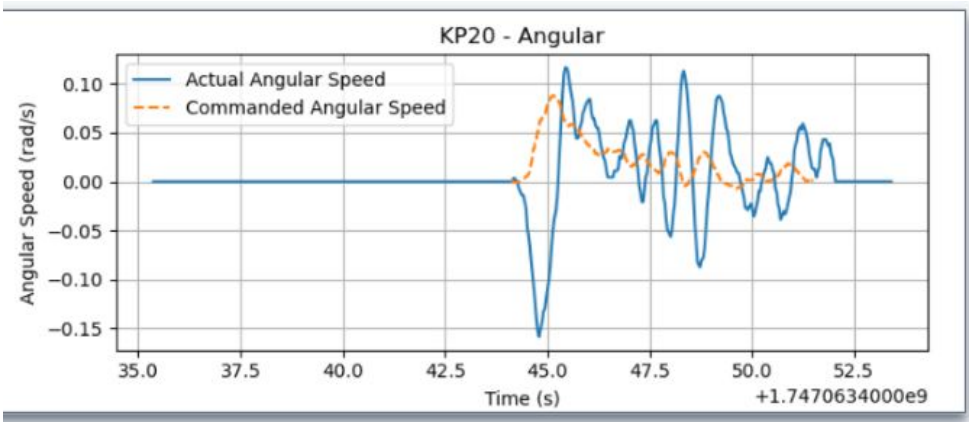

Figure C.12: Angular graph with $K_P = 20$

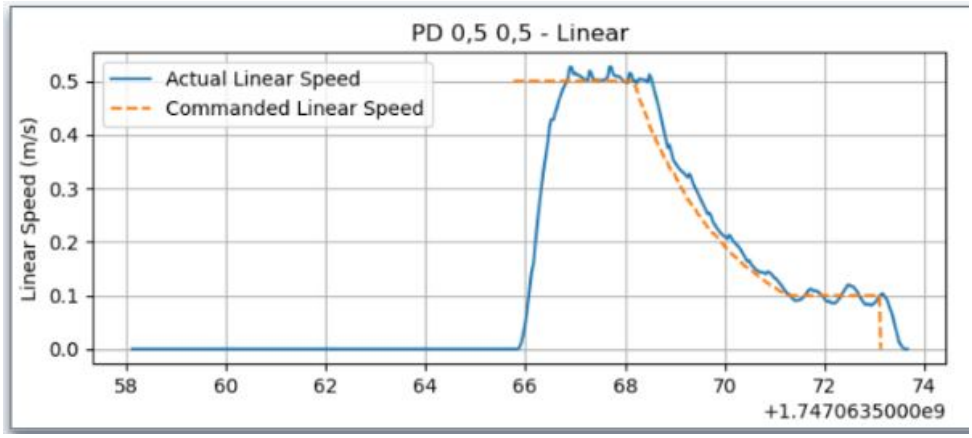

Figure C.13: Linear graph with $K_P = 0.5$ and $K_d = 0.5$

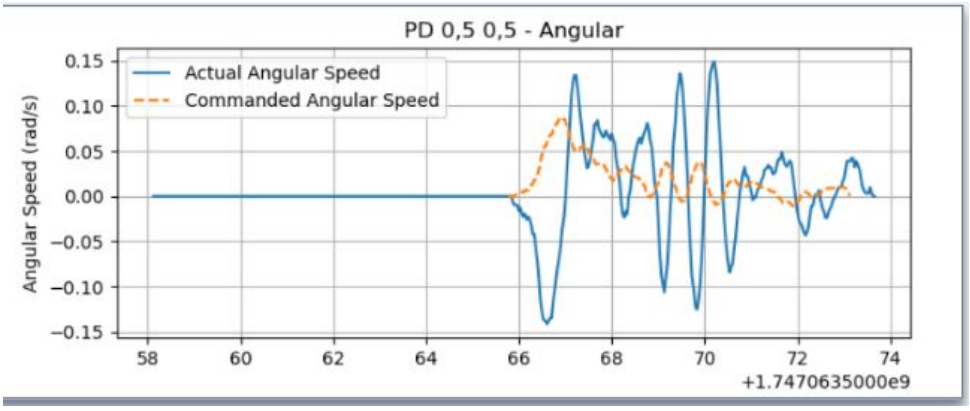

Figure C.14: Angular graph with $K_P = 0.5$ and $K_d = 0.5$

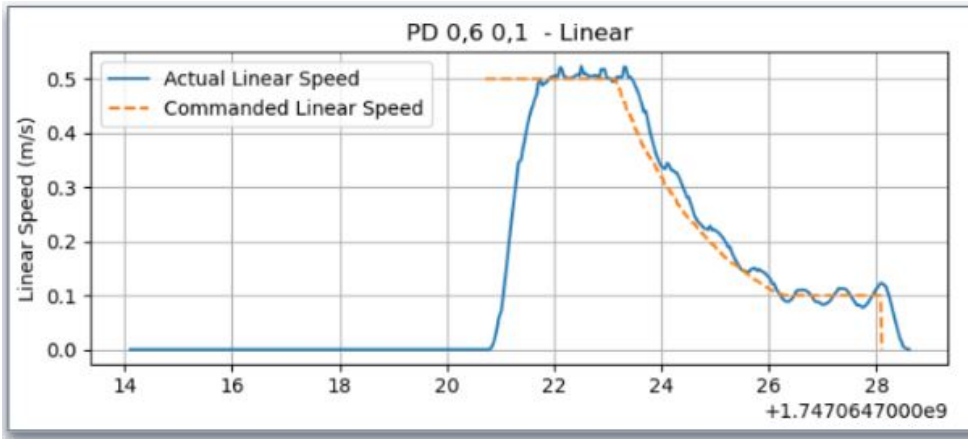

Figure C.15: Linear graph with $K_P = 0.6$ and $K_d = 0.1$

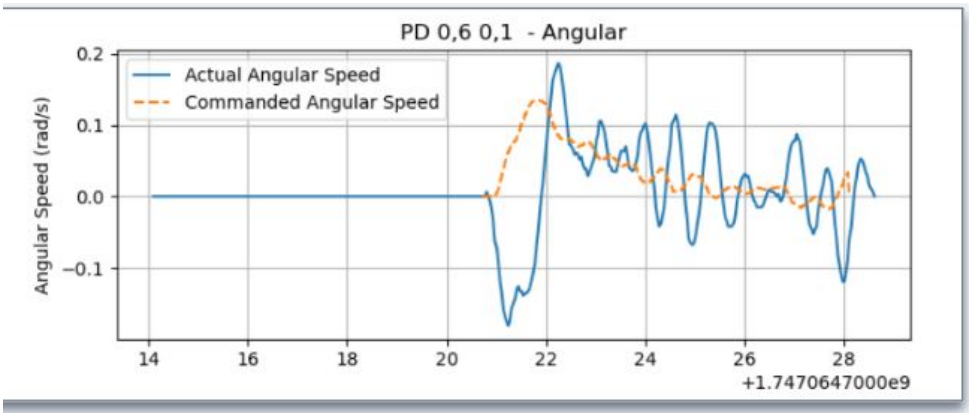

Figure C.16: Angular graph with $K_P = 0.6$ and $K_d = 0.1$

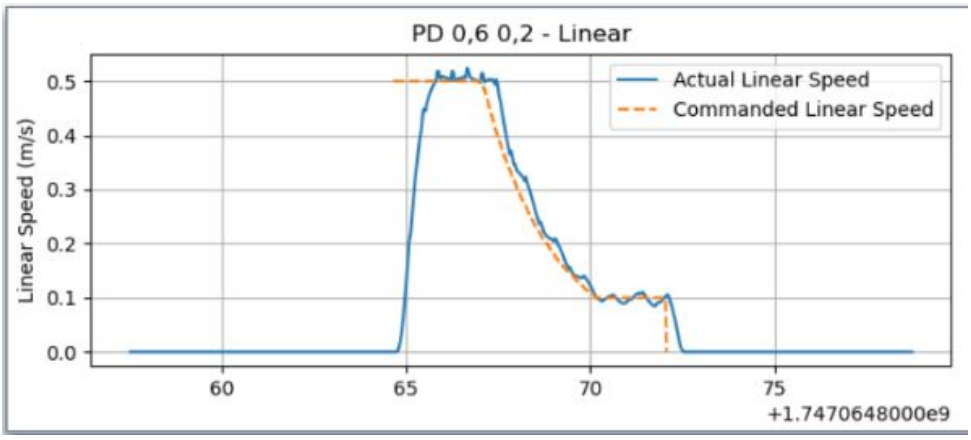

Figure C.17: Linear graph with $K_P = 0.6$ and $K_d = 0.2$

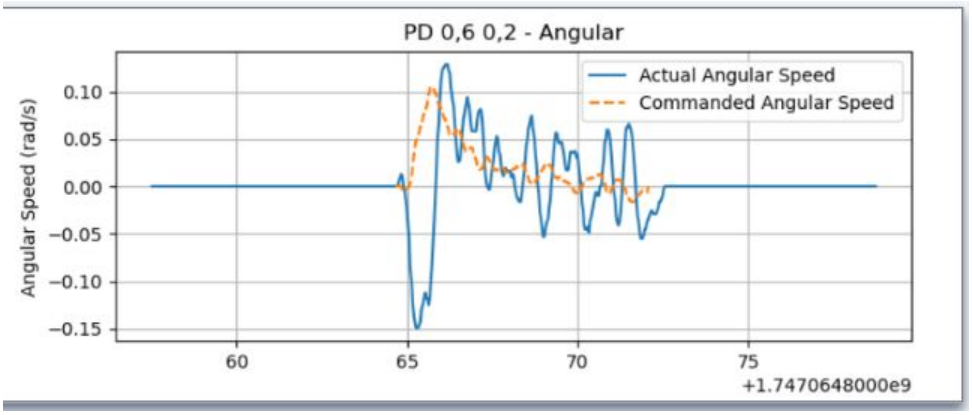

Figure C.18: Angular graph with $K_P = 0.6$ and $K_d = 0.2$

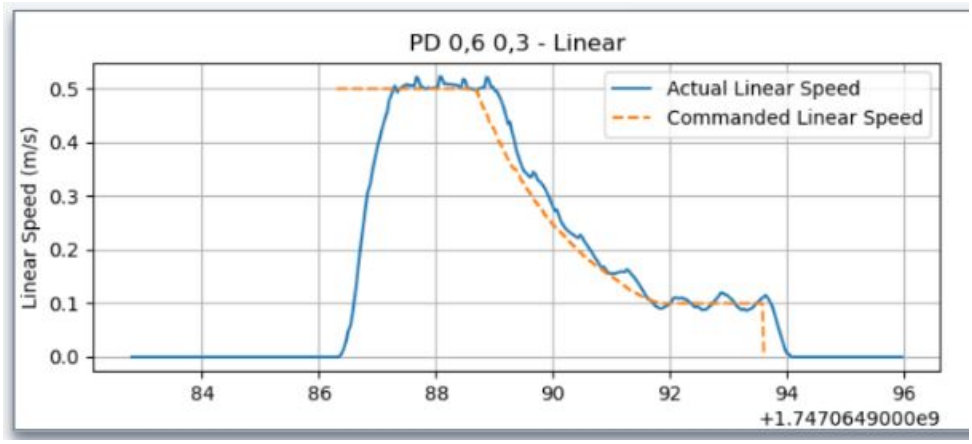

Figure C.19: Linear graph with $K_P = 0.6$ and $K_d = 0.3$

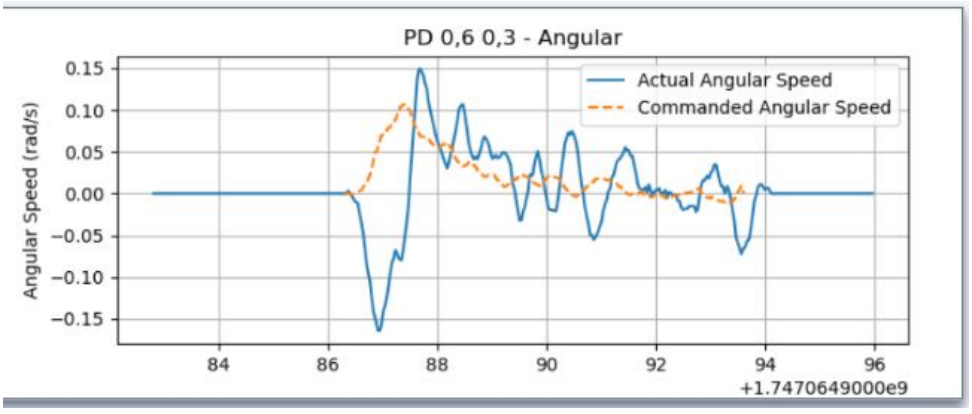

Figure C.20: Angular graph with $K_P = 0.6$ and $K_d = 0.3$

D

Code

Link to Github: `https://github.com/Trenaddictus/vaccum_robot`

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS