Autonomous Mobile Warehouse Robot

Graduation Project Prepared By

Student Name	Student ID
Ahmed Aly Gamal El-Din El-Ghannam	19015292
Ahmed Sherif Ahmed Mahmoud Ghanem	19015255
Ahmed Yossry Ahmed Arafa	17010296
Dina Hamdy Elsayed Mohamed	19015643
Menna Mohamed Anwar Ramadan Mohamed	19016729
Ragai Ahmed Abdelfattah Awad	19015655
Yahia Walid Mohamad Eldakhakhny	19016891
Youssef Abbas Mohamed Zaki	18012126
Youssef Mohammed Elsayed	19016941
Zyad Alaa Elsayed Goubashy	19015728

Supervisor:	Dr. Mohammed El-Shimmy
Academic Year:	2023 – 2024
Department:	Communications and Electronics Engineering
Faculty:	Faculty of Engineering
University:	Alexandria University

Abstract

Acknowledgement

Contents

List of Figures xi

List of Tables xiii

Listings xv

Nomenclature xvii

I. Project Overview 1

1. Introduction 3

1.1. Problem Statement . . . 3
1.2. Specifications . . . 3
1.3. Development Objectives . . . 4
1.4. Proposed System Architecture . . . 5
1.5. What To Expect . . . 5

II. Electrical & Mechanical Infrastructure 7

2. Mechanical Infrastructure 9

2.1. Introduction . . . 9
2.2. Design Process . . . 9
2.2.1. Conceptualization . . . 9
2.2.2. CAD Modeling Using SolidWorks . . . 9
2.3. Robot Chassis . . . 10
2.3.1. Material Selection . . . 10
2.3.2. Chassis Design . . . 10
2.3.3. Fabrication . . . 10
2.4. Mecanum Wheels . . . 10
2.4.1. Introduction to Mecanum Wheels . . . 10
2.4.2. Design and Selection . . . 11
2.4.3. Integration . . . 11
2.5. Camera Fixation . . . 11
2.5.1. Requirements . . . 11
2.5.2. Design . . . 11

2.5.3. Fabrication and Assembly . . . 12
2.6. Testing and Validation . . . 12
2.6.1. Mobility Testing . . . 12
2.6.2. Camera Stability Testing . . . 12
2.6.3. Durability Testing . . . 12
2.7. Output . . . 13

3. Electrical & Electronic Infrastructure – Power Management & Delivery 15
3.1. Introduction . . . 15
3.2. Development Objectives . . . 15
3.3. Design Approach . . . 15
3.4. Proposed Design . . . 16
3.4.1. Battery Pack . . . 16
3.4.2. Battery Management System . . . 17
3.4.3. Auxiliary Boards . . . 21
3.4.4. PCB Designs . . . 24

4. Electrical & Electronic Infrastructure – Sensing & Control 27
4.1. Introduction . . . 27
4.2. Development Objectives . . . 27
4.3. Used Components and Software Tools . . . 28
4.4. Design and Development Strategy . . . 29
4.4.1. Ultrasonic Implementation . . . 29
4.4.2. MPU6050 Implementation . . . 30
4.4.3. Encoder Implementation . . . 32
4.4.4. Publishing Data . . . 35
4.4.5. Interfacing PCB . . . 35
4.5. Output . . . 35

III. Central Processing Node 37

5. Communication Hub 39
5.1. Development Objectives . . . 39
5.2. Used Tools and Frameworks . . . 39
5.2.1. Tools for Upwards Communication . . . 40
5.2.2. Tools for Downwards Communication . . . 43

6. Central Processing Node – Reliable Localization 47
6.1. Development Objectives . . . 47
6.2. Methods of Localization . . . 47
6.2.1. Proposed Design . . . 48
6.2.2. Output . . . 51

6.2.3. Hardware Output: 52

IV. User Service Node **53**

7. Web Application **55**
7.1. Introduction 55
7.2. Development Objectives 55
7.3. Used Tools and Frameworks 57
7.3.1. Front-End Implementation 57
7.3.2. Back-End Implementation 66
7.4. Design and Development Strategy 68
7.4.1. System Architecture 69
7.4.2. User Interface Design 70
7.4.3. Implementation Plan 71
7.5. Output 71
7.5.1. Home Page 71
7.5.2. Robots Management 72
7.5.3. Maps Creation 73
7.5.4. Tasks Management 75

8. Server Infrastructure **77**
8.1. Development Objectives: Overview 77
8.2. System Design 77
8.2.1. Web Application Dependencies 78
8.2.2. Firewall Configuration 78

V. Results & Conclusion **79**

9. Setup For A Test Run **81**
9.1. Step 1: Install Dependencies on Server 81
9.2. Step 2: Install dependencies on the SBC 85
9.3. Step 3: Start Web Server Components 87
9.4. Step 4: Start SBC Components 88
9.4.1. Starting Dashboard Back-End 88
9.4.2. Starting ROS Environment 88
9.5. Step 5: Use The Application 90

10. Future Work & Recommendations **93**
10.1. Introduction 93
10.2. Areas For Improvements 93

Bibliography **95**

A. Bill of Materials **A1**
A.1. Boards A1
A.1.1. Sensors' Board A1

B. Theoretical Background – Sensors & PID **B1**
B.1. Encoders B1
B.1.1. Technologies used in Encoders B1
B.1.2. Basic types B2
B.1.3. Encoder Resolution Parameters B2
B.1.4. Encoder Working Principle B3
B.2. IMU (Inertial Measurement Units) B4
B.2.1. MPU6050 Module B4
B.3. Ultrasonic B6
B.4. PID Control B7
B.4.1. Introduction B7
B.4.2. Overview of PID Control B8
B.4.3. Discrete PID Controller B8
B.4.4. IMC Tuning Correlations B9
B.4.5. Optional Derivative Filter B9
B.4.6. Simple Tuning Rules B10
B.4.7. Anti-Reset Windup B10
B.4.8. Derivative kick B10

C. Me **C1**

D. Localization Scripts **D1**
D.1. Forward Kinematics – Dead Reckoning D1
D.2. Inverse Kinematics D3
D.3. Extended Kalman Filter D4

List of Figures

1.1. Block Diagram That Depicts System Architecture 5
1.2. Block Diagram That Depicts System Architecture (2) 5

2.1. Final Design – Full view 13
2.2. Final Design – Bottom view 13
2.3. Final Design – Side view 14
2.4. Final Design – Back view 14

3.1. The Block Diagram for Power Management. 16
3.2. Hybrid cell configurations in a pack. 17
3.3. The block diagram of the suggested BMS 18
3.4. Voltage Measurement Using Voltage Divider. 18
3.5. The voltage drop method of current measurement. 19
3.6. Improved Current Measurement Using Voltage Drop Method. 19
3.7. Temperature Measurement Using Op-Amp. [1] 20
3.8. Simulating the Cell-Balancing Circuit on LTSpice 20
3.9. The results of simulation. Vin_1 in green, Current through R_4 in red. . . . 21
3.10. Voltage Conversion for controller (Top) Measurement Modules and Protection Module (Bottom) 21
3.11. Cell-Balancing circuit and the ESP32 controlling the BMS. 21
3.12. Schematics for the Power Board. 22
3.13. First Part of Sensors' Board Schematics. 22
3.14. Second Part of Sensors' Board Schematics. 22
3.15. Schematics of Actuators' Board. 23
3.16. First Part of Motion Board Schematics. 23
3.17. Second Part of Motion Board Schematics. 23
3.18. PCB Output for the BMS Board. 24
3.19. PCB Output for the Power Board. 24
3.20. PCB Output for the Sensors' Board. 25
3.21. PCB Output for the Actuators' Board. 25
3.22. PCB Output for the Motion Board. 26

4.1. Used Microcontroller 28

5.1. Communication between ROS nodes 43

6.1. The Block Diagram for The Used VLC system 49
6.2. The used photodiode in the receiver. 49
6.3. The Transimpedance Amplifier Stage of The Implemented Receiver. 50
6.4. The second stage of the receiver: Non-inverting Amplifier (left) and inverting Schmitt Trigger (right). 51
6.5. Simulating The second phase of the receiver on LTSpice. 51
6.6. The results of simulation. 51
6.7. Schematics for The VLC Receiver. 52
6.8. PCB Output for The VLC Receiver. 52

7.1. Home page 71
7.2. Robots management dashboard 72
7.3. Creating new robot 72
7.4. Editing robot 72
7.5. Maps management dashboard 73
7.6. Node map editor 73
7.7. Updating node information. 74
7.8. Edit map 74
7.9. Tasks management dashboard 75
7.10. New task creation 75

9.1. Sign-up screen 90
9.2. Sign-in screen 90

B.1. Pulses per Revolution Diagram B3
B.2. Encoder Output Signals B3
B.3. MPU6050 Module B4
B.4. MPU-6050 3-Axis Gyroscope B5
B.5. MPU-6050 3-Axis Accelerometer B5
B.6. Ultrasonic Module B6
B.7. Ultrasonic sensor diagram B7

List of Tables

Listings

4.1. Initialization Function For Ultrasonic Sensors 29
4.2. Function to Detect Distance From Obstacle – Left Sensor 29
4.3. Function to Detect Distance From Obstacle – Right Sensor 30
4.4. Libraries Used For MPU6050 30
4.5. Defined Variables For MPU6050 30
4.6. Extracting Data From MPU6050 Buffer 30
4.7. Defined Class For Encoder Interfacing 32
4.8. Encoder Class Function For Speed Calculation 32
4.9. Encoder ISR Class 33
4.10. Encoder ISR Functions 33
4.11. Speed Control Function (1) 33
4.12. Speed Control Function (2) 34
4.13. Defined Class For PID 34
4.14. PID Class Function For Output Calculation 34

5.1. Sample json map file 40
5.2. Sample json task file 41
5.3. Synchronization script 42

7.1. React Router Dom Implementation (1) 58
7.2. React Router Dom Implementation (2) 59
7.3. Axios get request Implementation (1) 59
7.4. Axios get request Implementation (2) 60
7.5. Import the io function from the socket.io-client package 61
7.6. Initialize the socket connection 61
7.7. Initiate connection and listen for events 61
7.8. Cleanup on component unmount 62
7.9. Import the necessary libraries 62
7.10. Initialize the Flask application and sets up Socket.IO with cross-origin support. 62
7.11. Function to monitor file changes. 63
7.12. Run the Flask application with Socket.IO. 63
7.13. React Flow Implementation. 65
7.14. Middleware usage to protect routes in a Laravel application.. 67
7.15. Writing data to file in storage/app in laravel project. 67

8.1. Configure UFW to allow ports 78

9.1. Install general dependencies 81
9.2. Install Apache web server 81
9.3. Install and setup MySQL 81
9.4. Install and setup PHP 82
9.5. Install and setup composer 83
9.6. Install and setup laravel 83
9.7. Install NPM 83
9.8. Setup MySQL user 83
9.9. Install back-end 84
9.10. Install front-end 84
9.11. Install dashboard back-end 85
9.12. Install ROS Noetic 86
9.13. Get ROS scripts 86
9.14. Get ROS scripts 86
9.15. Start front-end 87
9.16. Start back-end 87
9.17. Start dashboard back-end 88
9.18. Start ROS environment 89
9.19. ROS launch 89

D.1. Dead reckoning script D1
D.2. Inverse Kinematics Function D3
D.3. EKF launch file D4

Nomenclature

If a nomenclature is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.

Abbreviations

Abbreviation	Definition
BMS	Battery Management Systme
ISA	International Standard Atmosphere
NTC	Negative Temperature Coefficient
VLC	Visible Light Communication
...	

Symbols

Symbol	Definition	Unit
V	Velocity	[m/s]
...		
ρ	Density	[kg/m^3]
...		

Part I

Project Overview

1 Introduction

1.1 Problem Statement

In a study made by the International Labor Organization (ILO), it was estimated that around 2.3 million men and women across the globe succumb to work-related accidents and/or diseases every year; this corresponds to over 6000 deaths every single day.

The consequences of these accidents are:

- Losses in manpower.
- Delayed production.
- Loss in capital—i.e.; money.

The proposed solution for this problem is using a fleet of autonomous robots inside factories: ones equipped with a multitude of sensors and state-of-the-art algorithms in order to navigate through challenging environments while being able to complete the desired tasks.

1.2 Specifications

In order for an autonomous robot to meet the expected work criteria, the following specifications are proposed:

1. Challenging environment traversal.
2. Battery-powered.
3. Ability to grab objects with a mechanical arm.
4. Ease of integration with existing environments.
5. Dashboard for monitoring robots.

1.3 Development Objectives

In order for the previously-mentioned list of specifications to be achieved, the system should be developed with the following objectives:

1. **From a Mechanical aspect, the robot should:**

    - Support the specified weight plus all the components of the robot itself—like boards, batteries, motors, etc—and be able to move normally.
    - Possess 3 Degrees of Freedom (DOF) to reliably traverse geometrically-challenging environments.
    - Contain a mechanical arm to hold shipments.

2. **From an Electrical and/or Electronic aspect, the robot should:**

    - Supply its own power via a battery pack that is able to both enable it to work for a specific time and ensure everything is powered correctly and safely.
    - Contain multiple sensors for two reasons: the first is to collect enough data in order to ensure reliable localization; the second is to check the state of the robot itself in order to trace back any fault that might arise.
    - Fine motors and arm control.

3. **From a Processing aspect, the central processing node inside the robot should:**

    - Act as a communication hub between Service Node and the rest of the robot's components: sending position data and sensors' readings; receiving the path that should be traversed from the user.
    - Perform accurate localization via odometry and a visible light communication (VLC) system prototyped for this project; in addition to employing sensor fusion to extract accurate readings.

4. **From a User Serving aspect, the user service node inside the robot should:**

    - Grant user-friendly GUI for task assignment.
    - Display system diagnostics and constantly monitor the robot's state and position.
    - Provide reliable communication between robot and user device via the implemented infrastructure.

1.4 Proposed System Architecture

The system consists of the main components shown in figure below. The arrows connecting the blocks indicate either communication between them or dependency of one over the other.

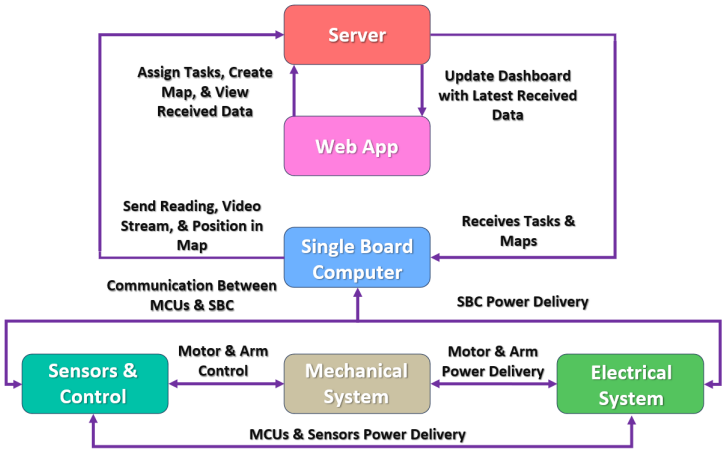

Figure 1.1.: Block Diagram That Depicts System Architecture

1.5 What To Expect

The next parts will include both the thought process behind each block implementation, used tools, as well as the expected outcome. The was the project is divided into section follows the depiction in the figure below.

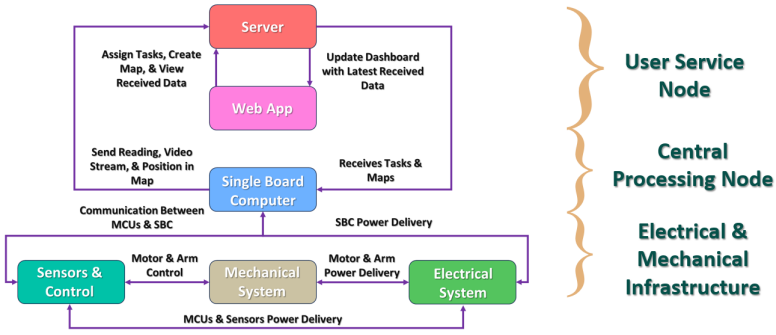

Figure 1.2.: Block Diagram That Depicts System Architecture (2)

Part II

Electrical & Mechanical Infrastructure

2 Mechanical Infrastructure

2.1 Introduction

The objective of our project was to design and build a prototype robot equipped with a mecanum wheel system and a camera fixation. This robot is intended to showcase advanced maneuverability and vision capabilities, making it suitable for various applications including surveillance, inspection, and autonomous navigation. The design and construction were carried out using SolidWorks, a powerful CAD software, to ensure precision and functionality.

2.2 Design Process

2.2.1 Conceptualization

The initial phase involved brainstorming and conceptualizing the robot's design. We focused on creating a compact and robust chassis that could support the mecanum wheels and the camera system. The key requirements identified were:

- Enhanced mobility with omnidirectional movement.
- Stable and secure mounting for the camera.
- Durability and ease of maintenance.

2.2.2 CAD Modeling Using SolidWorks

Using SolidWorks, we created detailed 3D models of the robot components. This phase included iterative design and simulation to optimize the robot's performance and ensure all parts fit together seamlessly. The major components designed were:

- Robot chassis.
- Mecanum wheels.
- Camera fixation mount.

2.3 Robot Chassis

2.3.1 Material Selection

The chassis was designed to be lightweight yet sturdy. We selected hard wood for the following reasons:

- High strength-to-weight ratio.
- Ease of machining and fabrication.
- Cost.

2.3.2 Chassis Design

The chassis was designed as a Circular frame with mounting points for the wheels and the camera system. The key features include:

- Central platform for mounting the control electronics and power supply.
- Four corner brackets for attaching the mecanum wheels.
- Slots and holes for cable management.

2.3.3 Fabrication

The chassis components were fabricated using CNC machining to ensure precision. The parts were then assembled using screws and bolts, providing a robust structure capable of withstanding operational stresses.

2.4 Mecanum Wheels

2.4.1 Introduction to Mecanum Wheels

Mecanum wheels are a type of omnidirectional wheel that allows a vehicle to move in any direction, including sideways, without changing its orientation. This capability is achieved through the unique design of the wheels, which feature rollers set at a 45-degree angle to the wheel's axis.

2.4.2 Design and Selection

For our robot, we selected commercially available mecanum wheels compatible with our chassis dimensions and load requirements. The key specifications considered were:

- Wheel diameter and width.
- Roller material and configuration.
- Load-bearing capacity.

2.4.3 Integration

The mecanum wheels were mounted on the chassis using custom-designed brackets. Each wheel was connected to an individual motor, enabling precise control of movement. The motors were chosen based on torque and speed requirements to ensure smooth and responsive operation.

2.5 Camera Fixation

2.5.1 Requirements

The camera fixation system needed to provide a stable and adjustable platform for mounting the camera. The key requirements were:

- Stability to avoid vibrations and ensure clear imaging.
- Adjustability to change the camera angle and orientation.
- Compatibility with various camera models.

2.5.2 Design

The camera fixation system was designed as a gimbal-like structure, allowing for both pan and tilt adjustments. The design features included:

- A base plate mounted on the chassis.
- Adjustable arms for changing the camera's position.
- Locking mechanisms to secure the camera in place.

2.5.3 Fabrication and Assembly

The camera fixation components were fabricated using a combination of 3D printing and CNC machining. This approach allowed for rapid prototyping and testing. The parts were assembled and tested to ensure stability and ease of adjustment.

2.6 Testing and Validation

2.6.1 Mobility Testing

The robot was tested for its mobility in various environments. The mecanum wheels provided excellent maneuverability, allowing the robot to navigate tight spaces and perform complex movements. The tests included:

- Forward and backward movement.
- Sideways movement.
- Diagonal movement.
- Rotational movement.

2.6.2 Camera Stability Testing

The camera fixation system was tested for stability and adjustability. The tests involved:

- Evaluating the ease of adjusting the camera angle.
- Assessing the stability of the camera during movement.
- Ensuring the camera remained secure in its fixed position.

2.6.3 Durability Testing

The overall durability of the robot was tested by subjecting it to various stress conditions. These included:

- Impact tests to assess the chassis' strength.
- Prolonged operation to test the endurance of the motors and wheels.
- Environmental tests to evaluate performance under different conditions.

2.7 Output

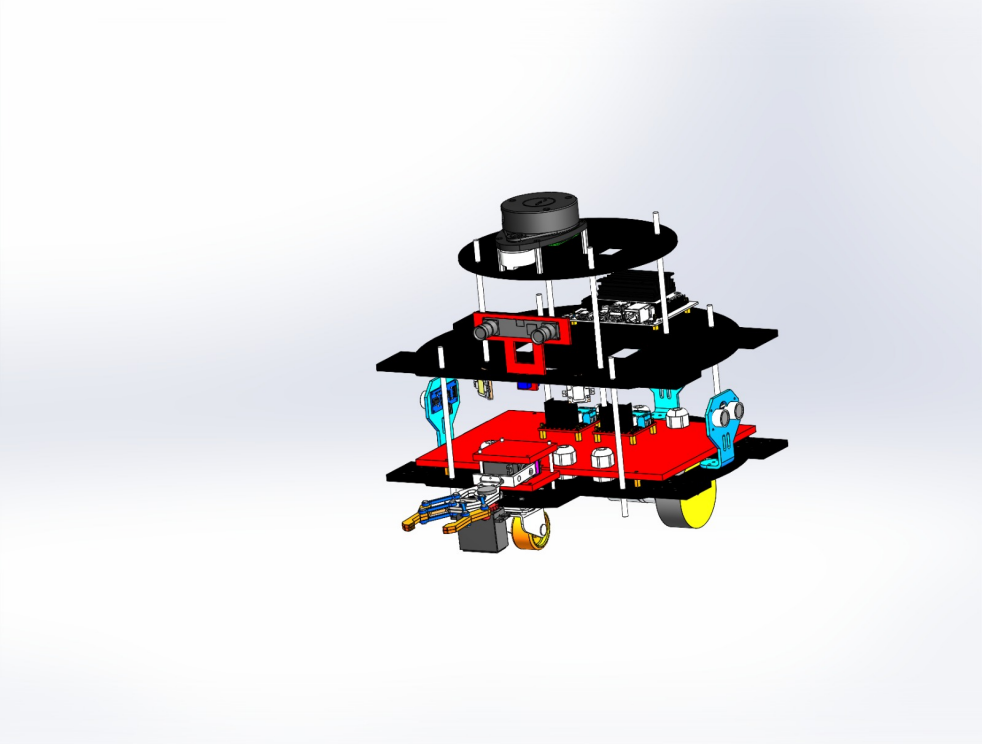

Figure 2.1.: Final Design – Full view

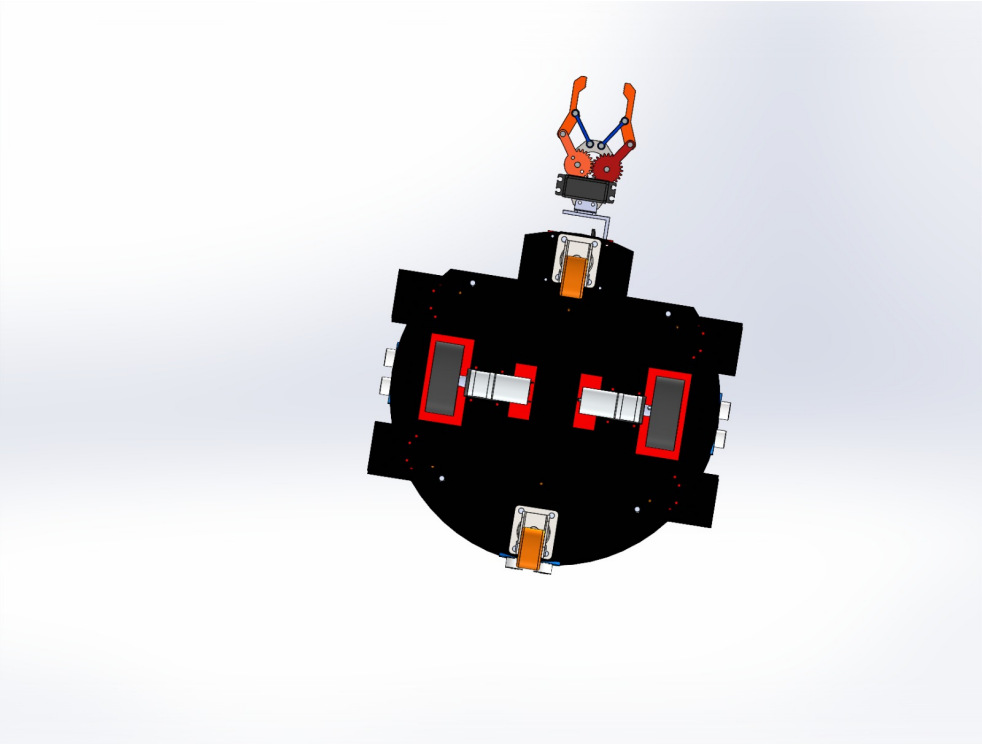

Figure 2.2.: Final Design – Bottom view

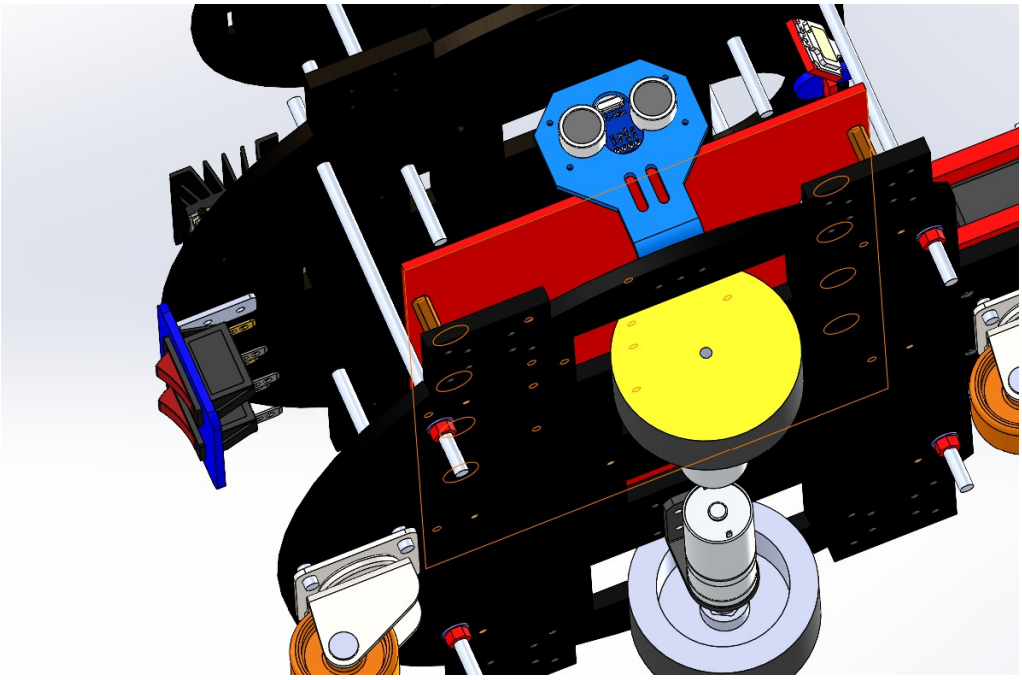

Figure 2.3.: Final Design – Side view

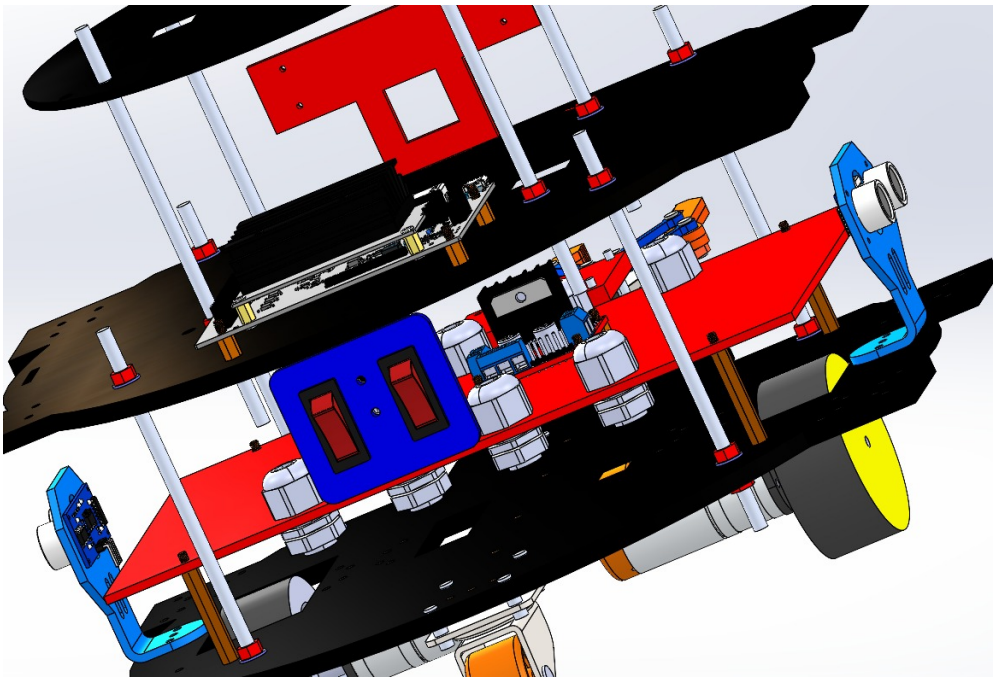

Figure 2.4.: Final Design – Back view

3 Electrical & Electronic Infrastructure – Power Management & Delivery

3.1 Introduction

In this chapter, the proposed electrical and electronic infrastructure model will be thoroughly examined—purely from a hardware/electronics perspective; along with the development objectives in mind to achieve the desired performance.

3.2 Development Objectives

The electrical infrastructure acts as the "brawn" for the robot, whether as energy source or in outside-world interactions. Providing the suitable power source is essential to ensure both the functionalities of the processing unit and the existing auxiliary units. Therefore, the main objectives for the infrastructure can be summarized as follows:

- Providing a stable power source to the whole system.
- Providing the means for data acquisition outside the robot.
- Providing the means for physical manipulation.

Based on the decided objectives, the infrastructure should capable of:

- Reliable Power Management and Delivery.

3.3 Design Approach

Connecting the battery pack directly to the system is not a plausible option, due to the need for different voltage levels for operation, mainly 12 and 5 volts. Moreover, monitoring the status of the battery pack and providing a controlled environment for its operation ensure a safer operation.
This approach can be represented as in diagram 3.1, where the selected battery pack is

connected to a BMS, which is connected to the processing unit for data acquisition. After being processed, the power goes to a distribution system to provide the suitable voltage levels to the system.

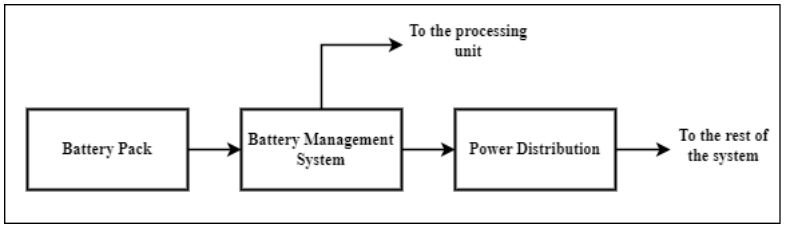

Figure 3.1.: The Block Diagram for Power Management.

For the **BMS**, a smart architecture is proposed . It should use a hardware-based protection system for over-voltage, under-voltage, over-current and short-circuit for faster response. A controller is present for measuring voltages, current, and temperature of the pack, and send it to the processing unit to be sent to the end-user. [2]**Power distribution** should provide 12V for motion, as well as 5V with high current capabilities for to power the processing unit and other subsystems.

3.4 Proposed Design

3.4.1 Battery Pack

Starting from the top, it is desired to build the battery pack with high gravimetric and volumetric energy densities for a compact pack running for a long time, so it was built using the 18650 Lithium-Ion (Li-ion) cells. [3]

Other important aspects of the pack are its nominal voltage and capacity, where they depend on the highest voltage-demand of the circuit and desired run-time. At full-load, the robot requires up to **51.25 W** of power, and for a runtime of about **1 hr**, according the relation 3.1:

$$Runtime = \frac{Energy\ Capacity\ at\ full\ charge[Wh]}{Consumed\ Power\ at\ Full-Load[W]} \tag{3.1}$$

A pack of energy capacity **51.25 Wh** is required. Since the energy capacity is a function of the voltage of the pack as in relation 3.2, we can determine the overall charge capacity, measured in **Ah**, where:

- n_s: number of cells or modules in series.
- n_p: number of cells or modules in parallel.
- C_{cell}: Charge capacity of one cell **[Ah]**

- V_{cell}: Voltage of one cell **[V]**

$$Energy\ Capacity[Wh] = n_s * n_p * C_{cell} * V_{cell} \tag{3.2}$$

V_{cell} of each 18650 Li-ion battery can be up to **4.2 V** at full-charge, and to satisfy the need of the most voltage demanding component: the motors; which are rated for **12 V**, the pack is built using **3** cells in series, to obtain **12.6 V** at full-charge. Therefore, the charge capacity of each cell is about **4.067 Ah**.

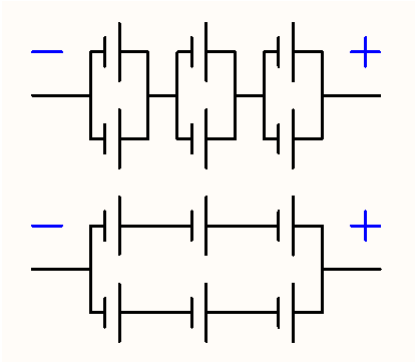

Figure 3.2.: Hybrid cell configurations in a pack.

To use a cell with a commercially-available capacity, parallel connections can be used. As shown in figure 3.2, cells are either connected in parallel; forming modules to be connected in series (as in the top figure), or modules of series-connected cells are connected in parallel (as in the bottom figure). These configurations hold the naming convention: { n_s (n_p) modules and n_p(n_s) cells}. In this case, these configurations are called **3S2P** and **2P3S** respectively. For the sake of simplicity in monitoring, the former configuration is used. From the local market, the best choice for Li-ion cells was the **BESTON** Li-ion cells with nominal capacity of **2600 mAh**. Connecting 2 cells in parallel formed a module of capacity **5200 mAh**, for a total energy capacity of **65.52 Wh** for about 1 hour and 15 minutes of runtime, which was enough for the prototype.

3.4.2 Battery Management System

Previously in section 3.3, the main objectives for the BMS are defined, thus each one can be achieved using abstracted subsystems, as shown in figure 3.3. Monitoring the pack status provides the user with information regarding its health, charge and discharge rate, and any possible problem with the pack.

For the choice of the controller, It was decided to use the ESP32.

Voltage measurement depends on the basic idea of **voltage dividers** as shown in figure 3.4 .Each module of the pack (refer to the upper schematic in figure 3.2) feeds two resistors in series, and the middle voltage is measured by the controller. Voltages of 4.2, 8.4, 12.6

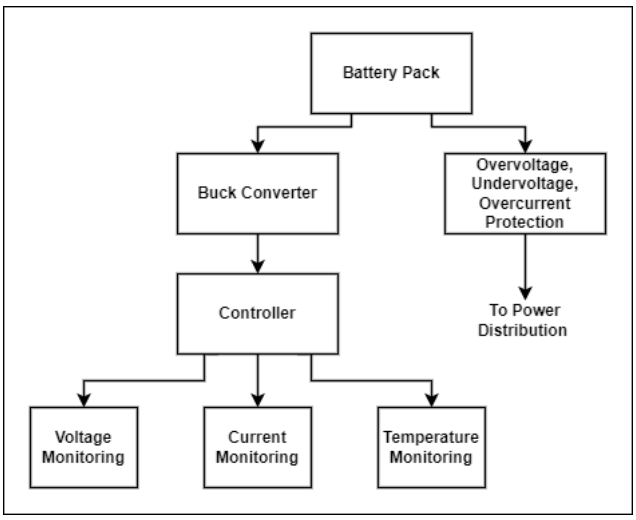

Figure 3.3.: The block diagram of the suggested BMS

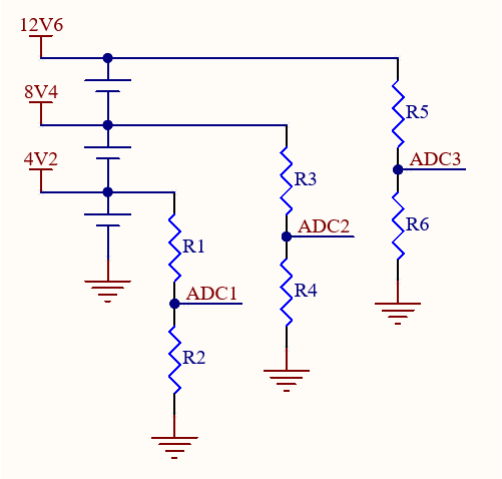

Figure 3.4.: Voltage Measurement Using Voltage Divider.

volts at full-charge are converted to voltages suitable for the ADC of the controller. Based on calculations in the **Appendix B**, the minimum values of $\frac{R_1}{R_2}$, $\frac{R_3}{R_4}$, and $\frac{R_5}{R_6}$ are $\frac{3}{11}$, $\frac{17}{11}$, and $\frac{31}{11}$ respectively.

Current measurement had two approaches: using a voltage-drop method or a hall-effect sensor. For the sake of simplicity and cost, the voltage-drop approach is chosen, as shown in figure 3.5. Its concept depends on measuring the voltage drop across R_{Shunt}, and calculating current knowing the value of the resistor. Some issues arise from this approach are:

- The voltage across R_{Shunt} may exceed the maximum voltage the ADC can handle.
- If R_{Shunt} is too high, the power loss across R_{Shunt} can be a considerable loss, and even exceed its power rating.

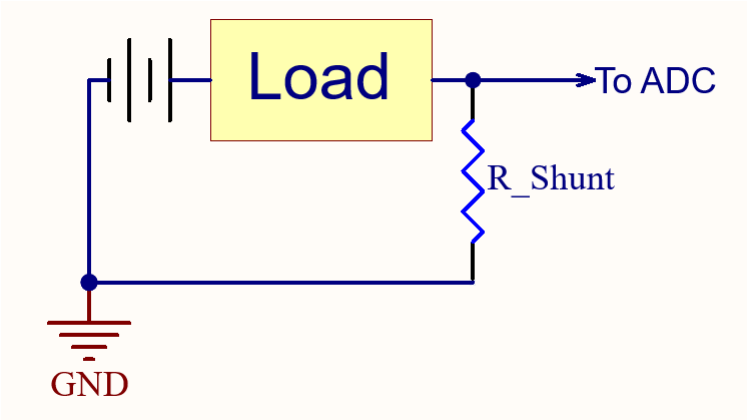

Figure 3.5.: The voltage drop method of current measurement.

- If R_{Shunt} is too low, the measured voltage range can be quite low, and inaccuracies might occur.

Alternate solution: Using a low resistance shunt of high power rating, and the voltage across it is fed to a rail-to-rail Op-Amp with a non-inverting amplifier configuration, as shown in figure 3.6. This will ensure a low voltage drop across R_{Shunt} and a wide range of measured voltage for better accuracy.

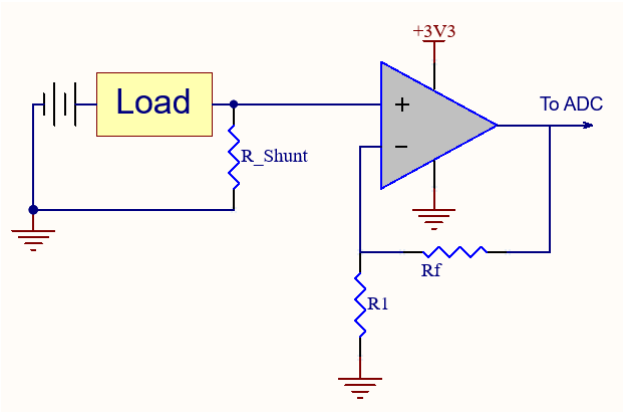

Figure 3.6.: Improved Current Measurement Using Voltage Drop Method.

Applying that design concept, the component values were chosen accordingly. R_{Shunt} = 0.05Ω for a voltage drop up to **0.2 V** and power loss of **0.8 W** at a current draw of **4 A**. The values of Op-Amp resistors R_1 and R_f are $15k\Omega$ and $150k\Omega$ for a voltage gain of **11** and a measured voltage of **2.2 V** which is fine for the ADC and provides a room for measuring higher current should it exist. The Op-Amp of choice is **MCP6002**: a rail-to-rail Op-Amp with V_{CC} = **3.3 V** should provide output voltages up to almost its V_{CC}.

Temperature measurement can be done using a voltage divider circuit with one of the resistors being an NTC resistor. One issue is the narrow range of the output voltage of the voltage divider. One way to remedy this is using an Op-Amp to widen its range, utilizing the full ADC resolution. [1]

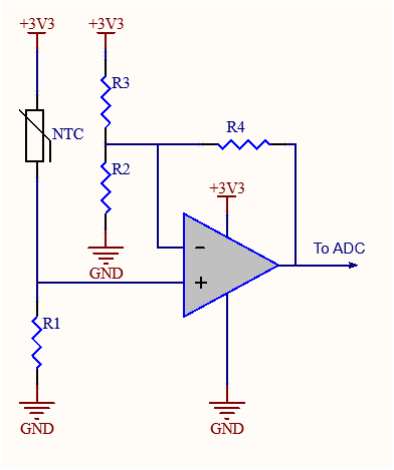

Figure 3.7.: Temperature Measurement Using Op-Amp. [1]

Using the calculations provided in document [1], the values of resistors are calculated to be: $R_1 = 78k\Omega$, $R_2 = 47k\Omega$, $R_3 = 56k\Omega$, and $R_4 = 68k\Omega$.

For achieving the protection features, a commercial 3S BMS module is used, capable of overvoltage, undervoltage, and over-current protection. However, one missing crucial feature of the BMS module is cell-balancing, which allows the cells of the pack itself to charge equally should there exist a difference in their capacity. This circuit uses information from voltage sensing subsystem, and open a path for each module to a dummy load if there exist an imbalance of charge.
The cell-balancing circuit is as shown in figure 3.8. All shown components are used in the actual circuit, except for the MOSFET, which is replaced by SI2302. The circuit simulates the pins of the ESP32, each pin controls an optocoupler to activate a MOSFET, which allows each module (the individual cell in the simulation) to discharge through a resistor The shown results of the simulation in figure 3.9.

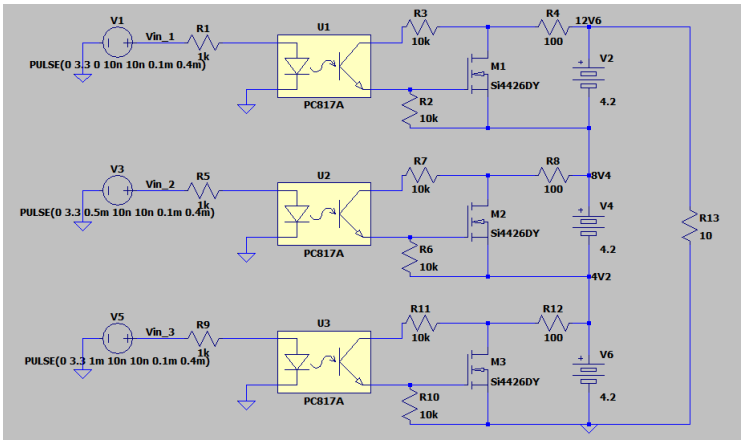

Figure 3.8.: Simulating the Cell-Balancing Circuit on LTSpice

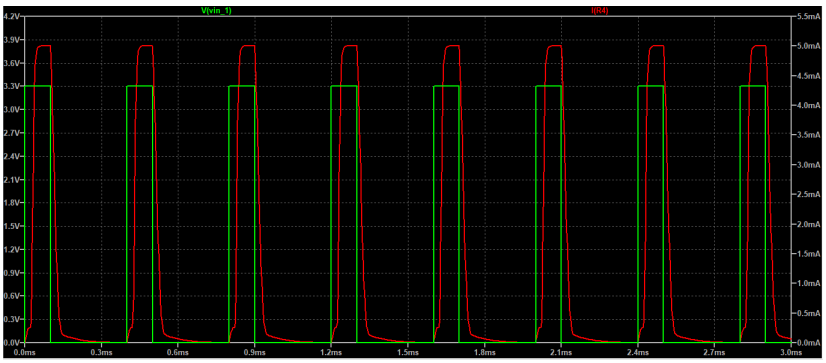

Figure 3.9.: The results of simulation. Vin_1 in green, Current through R_4 in red.

3.4.3 Auxiliary Boards

Battery Management System

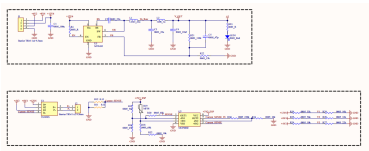

Figure 3.10.: Voltage Conversion for controller (Top) Measurement Modules and Protection Module (Bottom)

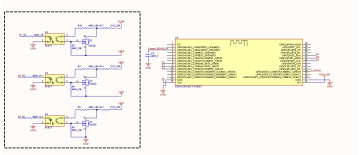

Figure 3.11.: Cell-Balancing circuit and the ESP32 controlling the BMS.

Power Distribution Board

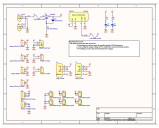

Figure 3.12.: Schematics for the Power Board.

Sensors' Board

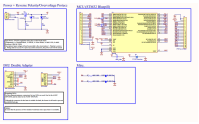

Figure 3.13.: First Part of Sensors' Board Schematics.

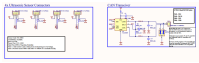

Figure 3.14.: Second Part of Sensors' Board Schematics.

Actuators' Board

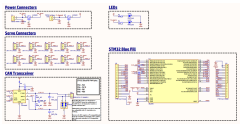

Figure 3.15.: Schematics of Actuators' Board.

Motion Board

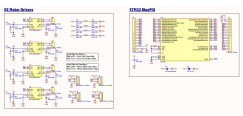

Figure 3.16.: First Part of Motion Board Schematics.

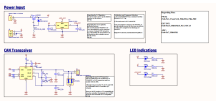

Figure 3.17.: Second Part of Motion Board Schematics.

3.4.4 PCB Designs

Battery Management System

Figure 3.18.: PCB Output for the BMS Board.

Power Distribution Board

Figure 3.19.: PCB Output for the Power Board.

Sensors' Board

Figure 3.20.: PCB Output for the Sensors' Board.

Actuators' Board

Figure 3.21.: PCB Output for the Actuators' Board.

Motion Board

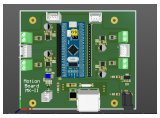

Figure 3.22.: PCB Output for the Motion Board.

4 Electrical & Electronic Infrastructure – Sensing & Control

4.1 Introduction

In this chapter, the proposed electrical and electronic infrastructure model will be thoroughly examined—purely from a sensors'/embedded systems' perspective; along with the development objectives in mind to achieve the desired performance.

4.2 Development Objectives

A robot that is able to do all the objectives listed in chapter 1 must need an incredible amount of data and precise control. Ergo, the robot must be equipped with a wide range of sensors that complements each other and provide the desired level of control. The list of following objectives will be framed specifically from a data gathering and fine control perspective:

1. There must be a way for the robot to detects rotation angle or linear displacement. For this, an **encoder** will be used at each motor.
2. As the robot is moving, it should be able to detect if there is an obstacle in its path. For that, there needs to be **ultrasonic** sensors at each of its sides.
3. The robot should be able to determine its speed and orientation in space. Therefore, it must be equipped with an **IMU** sensor.
4. The robot's path should be streamed in real-time for the operator. Hence, a **camera** is needed.
5. Fine motor control is needed and can be implemented via a **software PID controller**.

4.3 Used Components and Software Tools

The microcontroller-of-choice is the STM32F103C8T6—figure 4.1: a CORTEX-M3 ARM microcontroller equipped with an array of useful peripherals, is powerful enough to handle data gathering, and supports ROS [4].

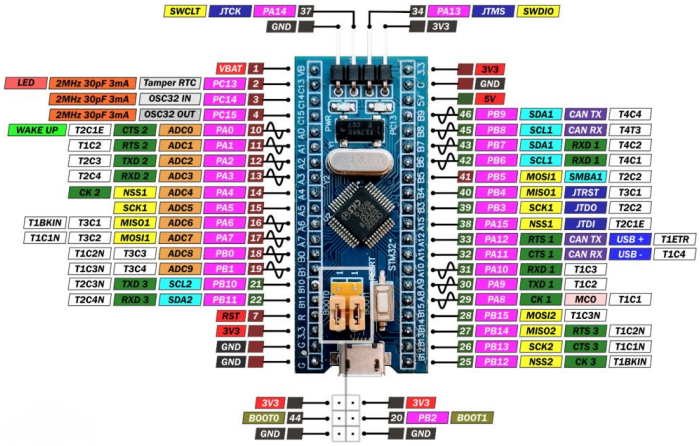

Figure 4.1.: Used Microcontroller

The used sensors are:

1. **Ultrasonic Sensors**
   - Function: detect obstacles in the robot's path in a 1.5m radius.
   - Used Peripheral: *GPIO*.
2. **MPU6050**
   - Function: detect changes in motion via the included 3-axis gyroscope and 3-axis accelerometer.
   - Peripheral: *I2C*.
3. **Magnetic Encoders**
   - Function: an encoder is connected to each motor to detect and calculate rotation angle and linear displacements; primarily used to implement **PID control**.
   - Used Peripheral: *Timer*, *Interrupts*, *ADC*, *PWM*.

It was decided to use the Arduino platform for handling data gathering and control. There are multiple reasons for this:

1. Open-source, widely-supported packages.
2. Ease of integration with ROS.

4.4 Design and Development Strategy

In this section, the design and development principles used will be tackled. For additional information regarding any of the used sensors, go to appendix B.

4.4.1 Ultrasonic Implementation

The principle of operation of ultrasonic sensors can be found in appendix B.3. Based on that, the proposed design is composed mainly of the functions in code snippets 4.1 to 4.3. The full code base can be found in the project's Github repository[1].

```
1  void ultrasonic_init()
2  {
3    pinMode(echo_front,INPUT);
4    pinMode(trig_front,OUTPUT);
5    pinMode(echo_back,INPUT);
6    pinMode(trig_back,OUTPUT);
7    pinMode(echo_left,INPUT);
8    pinMode(trig_left,OUTPUT);
9    pinMode(echo_right,INPUT);
10   pinMode(trig_right,OUTPUT);
11 }
```

Code Snippet 4.1: Initialization Function For Ultrasonic Sensors

```
1  float calc_distance_left()
2  {
3    t=0;
4    delayMicroseconds(2000);
5    digitalWrite(trig_left,LOW);
6    delayMicroseconds(2);
7    digitalWrite(trig_left,HIGH);
8    delayMicroseconds(10);
9    digitalWrite(trig_left,LOW);
10   t=pulseIn(echo_left,HIGH,timeout_ms);
11   float distance_left = t*Sspeed/2.0f;
12   return distance_left;
13 }
```

Code Snippet 4.2: Function to Detect Distance From Obstacle – Left Sensor

[1] Repository link: https://shorturl.at/Ln4UO

```
float calc_distance_right()
{
  t=0;
  delayMicroseconds(2000);
  digitalWrite(trig_right,LOW);
  delayMicroseconds(2);
  digitalWrite(trig_right,HIGH);
  delayMicroseconds(10);
  digitalWrite(trig_right,LOW);
  t=pulseIn(echo_right,HIGH,timeout_ms);
  float distance_right = t*Sspeed/2.0f;
  return distance_right;
}
```

Code Snippet 4.3: Function to Detect Distance From Obstacle – Right Sensor

4.4.2 MPU6050 Implementation

The principle of operation of IMU's can be found in appendix B.2. For convenience, an arduino library is used to operate the sensor and extract the necessary data from it. Based on that, the proposed design is composed mainly of the code listed in code snippets 4.4 to 4.6. The full code base can be found in the project's Github repository[2].

```
#include <Wire.h>
#include <I2Cdev.h>
#include <MPU6050_6Axis_MotionApps20.h>
```

Code Snippet 4.4: Libraries Used For MPU6050

```
float euler[3];         // [psi, theta, phi]    Euler angle
    container
uint8_t fifoBuffer[64]; // FIFO storage buffer
Quaternion q;           // [w, x, y, z]         quaternion
    container
MPU6050 mpu;
```

Code Snippet 4.5: Defined Variables For MPU6050

```
if(mpu.dmpGetCurrentFIFOPacket(fifoBuffer)){
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    Mpu_Values.data =euler[0] * 180/M_PI;
    Mpu_pub.publish(&Mpu_Values);
```

[2] Repository link: https://shorturl.at/Ln4UO

```
6 }
```

Code Snippet 4.6: Extracting Data From MPU6050 Buffer

4.4.3 Encoder Implementation

The principle of operation of encoders, as well as the idea behind PID can be found in appendix B.1 and B.4 respectively. The proposed design is composed mainly of the code listed in code snippets 4.7 to 4.12. Additionally, the code implementation for PID is in code snippets 4.13 to 4.14. The full code base can be found in the project's Github repository[3].

```
class Encoder{
  public:
    long counts; ///<encoder counts
    Encoder(float resolution); // constructor
    void setup();
    float calcspeed();

  private :
     int prevcount = 0;      ///<previous counts of the encoder
     float prevtime;         ///<previous time of the calculation
     float Espeed = 0 ;      ///<speed of the motor
     float  resolution = 0; ///<the resolution of the encoder
};
```

Code Snippet 4.7: Defined Class For Encoder Interfacing

```
float Encoder::calcspeed(){
  float current_time=millis();
  long newcount=counts;
  int dp=newcount-prevcount;
  float dt=current_time- prevtime;
  // calc speed
  Espeed=(60*1000/resolution)*(dp/dt);//in RPM,{speed=dp/dt in
      counts/ms}
  prevcount=counts; //update postition
  prevtime=current_time; //update time
  return Espeed;
}
```

Code Snippet 4.8: Encoder Class Function For Speed Calculation

[3] Repository link: https://shorturl.at/O9fj9

```
class ISR_Encoder : public Encoder{
    public:
        uint8_t PinA;   ///<the first pin of the encoder
        uint8_t PinB;   ///<the second pin if the encoder
    // constructor
    ISR_Encoder(float resolution, uint8_t pinA, uint8_t pinB);
    // method for interrupts on encoder pin A
    void update_ISR_A(void);
    // method for interrupts on encoder pin B
    void update_ISR_B(void);
    // method for encoder setup
    void setup();
};
```

Code Snippet 4.9: Encoder ISR Class

```
void ISR_Encoder::update_ISR_B(void)
{
    if (digitalRead(PinA) != digitalRead(PinB))
        counts++;
    else
        counts--;
}
void ISR_Encoder::update_ISR_A(void)
{
    if (digitalRead(PinA) == digitalRead(PinB))
        counts++;
    else
        counts--;
}
```

Code Snippet 4.10: Encoder ISR Functions

```
void speedControl()
{
  // Drive Motor FL
  digitalWrite(MOTOR_FL_IN1, wheel_pwm[0] > 0);
  digitalWrite(MOTOR_FL_IN2, wheel_pwm[0] < 0);
  analogWrite(MOTOR_FL_EN, constrain(abs(wheel_pwm[0]), MIN_VEL,
      MAX_VEL));
  // Drive Motor FR
  digitalWrite(MOTOR_FR_IN1, wheel_pwm[1] > 0);
  digitalWrite(MOTOR_FR_IN2, wheel_pwm[1] < 0);
  analogWrite(MOTOR_FR_EN, constrain(abs(wheel_pwm[1]), MIN_VEL,
      MAX_VEL));
```

Code Snippet 4.11: Speed Control Function (1)

```
  // Drive Motor BL
  digitalWrite(MOTOR_BL_IN1, wheel_pwm[2] > 0);
  digitalWrite(MOTOR_BL_IN2, wheel_pwm[2] < 0);
  analogWrite(MOTOR_BL_EN, constrain(abs(wheel_pwm[2]), MIN_VEL,
     MAX_VEL));
  // Drive Motor BR
  digitalWrite(MOTOR_BR_IN1, wheel_pwm[3] > 0);
  digitalWrite(MOTOR_BR_IN2, wheel_pwm[3] < 0);
  analogWrite(MOTOR_BR_EN, constrain(abs(wheel_pwm[3]), MIN_VEL,
     MAX_VEL));
}
```

Code Snippet 4.12: Speed Control Function (2)

```
class PIDControl{
    public:
        PIDControl(float Kp,float Ki,float Kd, float sample_time);
        float calculateOutput(float feedBack);
        void set_parameters(float Kp, float Ki, float Kd);
        void set_setpoint(float setpoint);
    private:
        float Kp = 0, Ki = 0, Kd = 0;
        float sample_time;
        float error, previous_error;
        float integralTerm, derivativeTerm;
        float output;
        float setpoint;
};
```

Code Snippet 4.13: Defined Class For PID

```
float  PIDControl::calculateOutput(float feedBack) {
    error = setpoint - feedBack;
    integralTerm += Ki * (error * sample_time);
    integralTerm = constrain(integralTerm,minIntegral,maxIntegral);
    derivativeTerm = Kd * (error - previous_error) / sample_time;
    output = Kp * error + integralTerm +  derivativeTerm;
    previous_error = error;
    if ((abs(setpoint-feedBack)<DEAD_ZONE)&& (0 == setpoint))
    {
      output = 0;
      integralTerm = 0;
    }
    return output;
}
```

Code Snippet 4.14: PID Class Function For Output Calculation

4.4.4 Publishing Data

REQUIRED BY ZYAD published via ROS messages + will be discussed more next chapter

4.4.5 Interfacing PCB

4.5 Output

Part III

Central Processing Node

5 Communication Hub

5.1 Development Objectives

The Single board computer (SBC) serves as the central processing node, i.e., it gathers all the data from the layers directly above and below it, then it performs the necessary computations on this data to provide control signals, feedback to electronic components, as well as feedback to the user. Thus, the SBC's development objectives can be summarized as follows:

- Receive data from the web GUI to perform instructions issued by the user.
- Send data and logs back to the GUI to ensure proper user feedback.
- Receive data from the underlying micro-controllers, to utilize it in the localization process.
- Use all gathered data to calculate the appropriate signals and feedback that should be sent back to the micro-controllers.

To achieve the aforementioned objectives, the SBC must be capable of two types of communication, which are:

1. Upwards communication: which is the type of communication concerned with receiving user input and providing feedback.
2. Downwards communication: which is the the type of communication that is concerned with receiving sensors' readings from micro-controllers and issuing commands to execute the assigned tasks.

In the subsequent sections we discuss the implementation of these objectives at greater length.

5.2 Used Tools and Frameworks

Different tools are used to accomplish the two desired types of communication.

5.2.1 Tools for Upwards Communication

There are two types of data that the SBC needs to receive from the server. The first type of data is the direct user request, this is what the SBC receives when the user requests that the robot moves with a specific speed, or when the user starts a given task. In short, the request data type is directly triggered by a user action.

Requests are sent from the server to the SBC using the Axios framework, which will be discussed in further detail in the next chapter, but in short, it is a tool used to send requests from the front-end of a web application to some endpoint. Axios uses XML and HTTP to accomplish this goal [5]. These requests are sent from the user-facing GUI to the PHP back-end that is running on the SBC.

The other data type that the SBC needs to receive from the server is files. Some user input is too complex to encapsulate in a single request, so these inputs are collected in files on the server, which are then sent to the SBC. The two main types of files are map files and task files, both of which are of the commonly used JavaScript Object Notation (JSON) format. Map json files contain details about the maps that the user creates in the GUI. For example it contains the identifiers and labels of every node in the map, the relation between nodes (which nodes are connected with wich nodes), and the absolute position of every node. A simple map json file is shown in code snippet 5.1.

```
{
  "name": "simple",
  "nodes": [
    {
      "id": "node_1",
      "type": "input",
      "position": {
        "x": 276.3011474609375,
        "y": -137.43181610107422
      },
      "data": {
        "label": "entrance",
        "X": 0,
        "Y": 0
      },
      "width": 150,
      "height": 40,
      "selected": false,
      "dragging": false
    },
    {
      "id": "node_2",
      "type": "output",
```

```
      "position": {
        "x": 268.8011474609375,
        "y": 23.56818389892578
      },
      "data": {
        "label": "exit",
        "X": 1.5,
        "Y": 0
      },
      "width": 150,
      "height": 40,
      "selected": true,
      "positionAbsolute": {
        "x": 268.8011474609375,
        "y": 23.56818389892578
      },
      "dragging": false
    }
  ],
  "edges": [
    {
      "source": "node_1",
      "sourceHandle": null,
      "target": "node_2",
      "targetHandle": null,
      "id": "reactflow__edge-node_1-node_2"
    }
  ]
}
```

Code Snippet 5.1: Sample json map file

Similarly, when the user creates a new task to be executed by a robot, this task is saved to a json file on the server. A sample task file is shown in code snippet 5.2.

```
{
  "id": 1,
  "name": "tst_simple",
  "map": "simple",
  "pickupNode": "entrance",
  "dropoffNode": "exit",
  "taskTime": "2024-06-15 11:41:00",
  "user_id": 1,
  "created_at": "2024-06-14T08:41:07.000000Z",
  "updated_at": "2024-06-14T08:41:07.000000Z"
}
```

Code Snippet 5.2: Sample json task file

The shown files are created on the server and need to be synchronized with the SBC. The desired feature is synchronization and not simple file transfer because these files represent maps and tasks that the user may edit or delete, so it is desirable to have these changes synchronized between the server and the SBC.

The tool that is used to accomplish this synchronization is RSYNC, which is a fast file synchronization tool for both remote and local files [6]. The bash script shown in code snippet 5.3 is used to synchronize the maps and tasks directories between the server and the SBC.

The script uses the server's IP address to communicate with the server. The SBC has a number of ssh keys used for authentication, one of which is used by RSYNC for authentication between the SBC and the server. But for this to work properly, the ssh-agent must be running on the SBC to enable automatic authentication. Lastly RSYNC is used to synchronize its target directories, i.e., the maps directory on the server and its corresponding directory on the SBC, and the same is true for the tasks directory.

The system administrator is free to change:

- IP address of the server.
- Which ssh keys are used for authentication (in script 5.3 all available ssh keys are added for good measure).
- The directories that RSYNC targets for synchronization.

```
1  #!/bin/bash
2  srv_ip=192.168.1.202
3  my_keys="$(ls ~/.ssh | grep .pub | sed 's/.pub//g')"
4  eval "$(ssh-agent -s)"
5
6  cd ~/.ssh
7  ssh-add $my_keys
8  # sync directories
9  rsync -avz yahia@$srv_ip:/home/yahia/GP_laravel/storage/app/maps
       /home/pi
10
11 rsync -avz yahia@$srv_ip:/home/yahia/GP_laravel/storage/app/tasks
       /home/pi
```

Code Snippet 5.3: Synchronization script

5.2.2 Tools for Downwards Communication

Downwards communication is mainly accomplished using the Robot Operating System (ROS) which is a set of software libraries and tools that are used to build robot applications [7]. When building an application using ROS, the application logic is broken into a number of ROS nodes, where these nodes communicate using ROS topics. When a node produces data it is then called a publisher, that is because this data is published on a specific topic. When a node receives data it is then called a subscriber, that is because it subscribes to the appropriate ROS topic that contains the data of interest. This data is called ROS messages, and each ROS topic supports a specific type of messages. For example, some ROS topics support only integer data, so all nodes publishing on this topic must only publish integer data, and all nodes subscribing to this topic must only expect to receive integer data. Figure 5.1 shows the relation between the different ROS nodes on the system as well as the ROS topics [7]. There have been many ROS releases over the years, here we use ROS Noetic, which is primarily targeted for machines that run Ubuntu 20.04 LTS as their operating system. In other words, newer versions of Ubuntu will not be compatible with ROS Noetic.

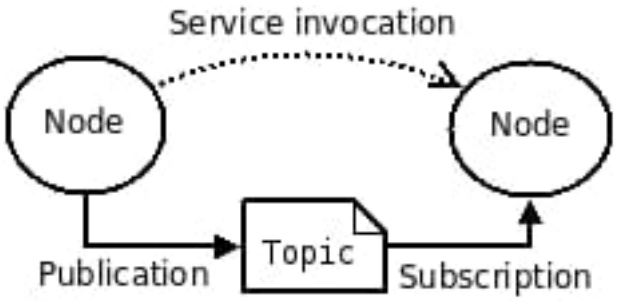

Figure 5.1.: Communication between ROS nodes

We will define some of the basic concepts needed to understand ROS and how to use it:

- Packages:

  They are the primary unit for organizing software in ROS. They can contain ROS runtime processes (nodes), ROS-dependent libraries, datasets, configuration files, or any other related components. In ROS, packages are the smallest units of build and release, meaning that the smallest entity you can build and release is a package.

- Package Manifests

  Package manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other metadata such as exported packages.

- Message (msg) Types:

  Message types, define the data structures for messages sent within ROS.

  There are several types of messages used to facilitate communication between nodes. These message types are defined by their content and structure. The main types of ROS messages are:

  1. Standard Messages:

     These are the predefined message types provided by ROS. They cover a wide range of common data structures and are grouped into different categories:

     - std_msgs: Contains simple message types like Bool, Int32, Float64, String, etc.
     - geometry_msgs: Includes messages for geometric primitives like Point, Vector3, Pose, Twist, etc.
     - sensor_msgs: Contains messages related to sensor data like Image, Imu, LaserScan, NavSatFix, etc.

  2. Custom Messages:

     Users can define their own message types to suit their specific application needs. Custom messages are defined in .msg files within a ROS package and can include various data types such as integers, floats, strings, arrays, and other message types.

- Nodes:

  Nodes are processes that perform computations. ROS is designed for fine-grained modularity, meaning a robot control system typically comprises many nodes. For example, one node might control a laser range-finder, another controls the wheel motors, another handles localization, another performs path planning, and yet another provides a graphical view of the system. Nodes in ROS are written using a ROS client library such as roscpp or rospy.

- Master:

  The ROS Master provides name registration and lookup services for the Computation Graph. Without the Master, nodes would be unable to locate each other, exchange messages, or invoke services.

- Topics:

  Messages are routed via a transport system using publish/subscribe semantics. A node sends a message by publishing it to a specific topic, a name identifying the message content. Nodes interested in certain data subscribe to the appropriate topic. Multiple publishers and subscribers can exist for a single topic, and a single node can publish and/or subscribe to multiple topics. Publishers and subscribers are generally unaware of each other's existence, which decouples information production from its consumption. Logically, a topic functions as a strongly typed message bus, with each bus having a name. Anyone can connect to the bus to send or receive messages, provided they use the correct type.

6 Central Processing Node – Reliable Localization

6.1 Development Objectives

Through the robot runtime, it will receive missions in which cargo shall be transmitted from one place to another. Each place has its own set of information, which can either be related to the place itself, related to adjacent places, or assigned by the infrastructure. In light of all of this, the robot should be able to:

- Determine the best path from current location to target destination.
- Localize itself using existing infrastructure.
- Integrate various sensor inputs to maintain accurate localization.

6.2 Methods of Localization

Through investigating different means of localization, an approach involving Dead Reckoning technique for position estimation accompanied with an Extended Kalman Filter (EKF) to improve estimation performance was selected. The output of Dead Reckoning algorithm will be processed by the EKF in addition to IMU data in order to provide precise position estimation. This method is labeled Odometry Localization for the rest of the chapter.

Additionally, Visible Light Communication (VLC) is used as an auxiliary means of localization; offering several advantages over other methods such as Radio Frequency (RF) localization:

- It uses the unlicensed visible light portion of the spectrum.
- It makes use of the existing LEDs in the facility for both illumination and data transmission, while consuming less power than other RF transmitters.
- It depends on the inherent need of Line-Of-Sight (LOS) and the inability to penetrate through walls, making it more secure.

Localization via VLC can be classified into two categories depending on the specific implementation details and use cases, which are: **Direct Positioning**; where the receiver uses all received information (i.e. time of arrival, angle of arrival) from transmitters to directly determine its position, and **Two-Step Positioning**, in which the receiver first extracts position-related parameters from the transmitter, then processes these parameters using algorithms to determine its location. For its convenience, the two-step positioning approach is utilized, and more specifically the **proximity method**.

6.2.1 Proposed Design

The localization system uses a combination of dead reckoning and sensor fusion to maintain an accurate estimate of the robot's position. The dead reckoning algorithm is implemented in the **dead_reckoning.py** script, which updates the robot's position based on wheel encoder data and publishes it as an odometry ROS message.

Forward Kinematics

Forward kinematics involves calculating the position and orientation of the robot based on the joint parameters (e.g., wheel rotations). The dead reckoning script shown in code snippet D.1 in Appendix **??** includes the implementation of forward kinematics for a differential drive robot.

The script calculates the changes in position (Δx, Δy) and orientation ($\Delta\theta$) based on the velocity commands received and updates the robot's position accordingly.

Inverse Kinematics

Inverse kinematics involves calculating the joint parameters needed to achieve a specific position and orientation of the robot. In the context of a differential drive robot, inverse kinematics can be derived from the desired linear and angular velocities to determine the individual wheel speeds. The implementation is shown in code snippet D.2 located in Appendix **??**.

Sensor Fusion using EKF

The EKF is configured to combine odometry data and IMU data to provide a more accurate and robust localization estimate. The EKF configuration is provided in code snippet D.3 situated in Appendix **??**.

Visible Light Communication:

In the proximity method, each transmitter (i.e. LED) continuously sends a code specific to its location. As the robot moves, the receiver receives information from the closest transmitter, then uses algorithms to determine the code, thus knows its location. This method is chosen as a trade-off between its simplicity and its inaccuracy in localization. However, the used applications for the robot may not require much accuracy. [**?**] [**?**]

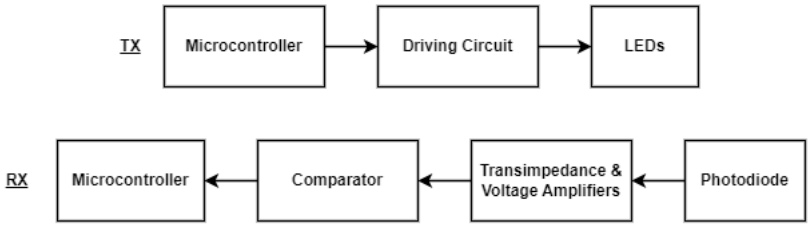

Figure 6.1.: The Block Diagram for The Used VLC system

The structure of the used VLC system is shown in figure 6.1. The code is assigned to every location and stored in a micro-controller, which periodically drives some LEDs. The signal transmitted by the LEDs are then received through a photodiode. The received signal is then amplified, then passes through a comparator to produce 0's and 1's, to be finally received by another microcontroller responsible for storing the received code, processing it and determining its location based on it.

Figure 6.2.: The used photodiode in the receiver.

The key element of the receiver system is the **photodiode**, in this case, a BPW34 shown in figure 6.2, which designed to work within wavelengths **[420:1100] nm**, produce open-circuit voltage up to **350 mV**, and short circuit current up to **70** μ**A** . It acts just like a solar-cell: when light falls on it, it generates current to its load. [**?**].
The issue is dealing with the BPW34 directly is quite difficult; since its current is in the μA. A solution this problem is to convert this insignificant current into more measurable voltage

through a **Transimpedance Amplifier** (or TI). It is designed to amplify the light-dependent current of the photodiode.

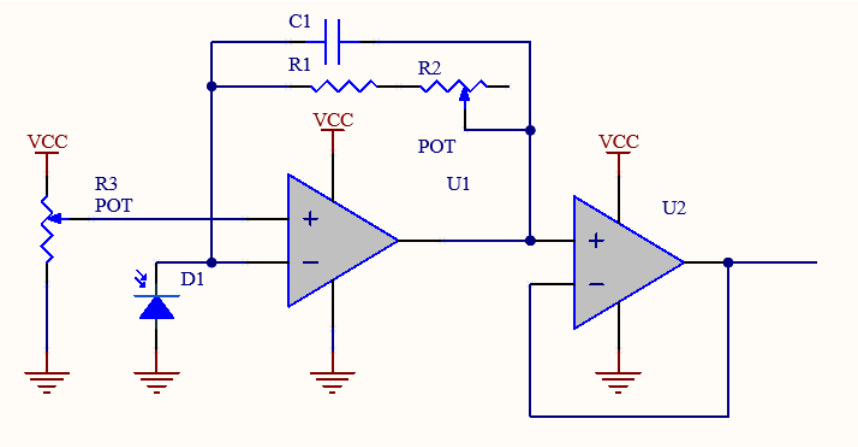

Figure 6.3.: The Transimpedance Amplifier Stage of The Implemented Receiver.

In the actual receiver shown in figure 6.3 the TI amplifier is designed to to be modified based on the transmitter and surrounding conditions, by modifying R_2, and the potentiometer R_3 controls its offset. The output of the TI is followed by a buffer to provide a low impedance output for the next stage. **[?]**

The output voltage the first stage has a range of about **0.7 V**, which may not be difficult to process. It is desired to:

- Increase the output voltages.
- Increase the voltage range, allowing for more accurate decision making.

Both of these objectives can be achieved by following the first stage with a non-inverting amplifier. A voltage gain ratio of 2 should be enough.

The final stage for the receiver is the decision making. Ideally, a comparator circuit with one threshold would work well, however, it comes with some issues:

- Unstable output when the input is very close to the threshold.
- The value for a '0' or a '1' can fluctuate causing misreadings in the middle of a single bit.

One way to optimize this behaviour is using a Schmitt Trigger for decision makinhg, which is basically a comparator with an upper and lower thresholds, where if and only if:

- $V_{in} > V_u$, V_{out} is HIGH.
- $V_{in} < V_l$, V_{out} is LOW.

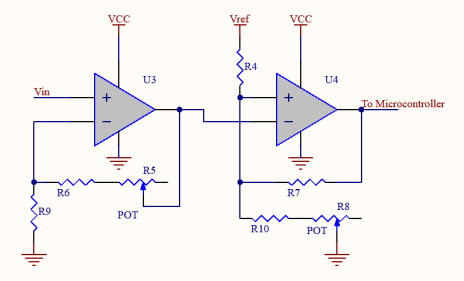

Figure 6.4.: The second stage of the receiver: Non-inverting Amplifier (left) and inverting Schmitt Trigger (right).

Where V_u and V_l are the Schmitt Trigger upper and lower thresholds respectively.

The used schmitt trigger in the actual circuit is an inverting one, so V_{out} inverted from what was mentioned above. To design the second stage with proper components, LTSpice was used for simulation, as found in figure **??**. The results can be found on figure **??**. It noted that $V_{amplified}$ is clamped from the negative side due to the presence of ground instead on negative voltage. On the other hand, to prevent clamping on the positive side for the amplifier, the 12V supply can be used to power it.

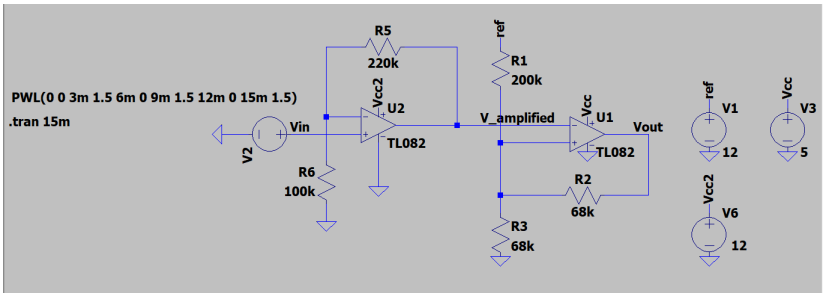

Figure 6.5.: Simulating The second phase of the receiver on LTSpice.

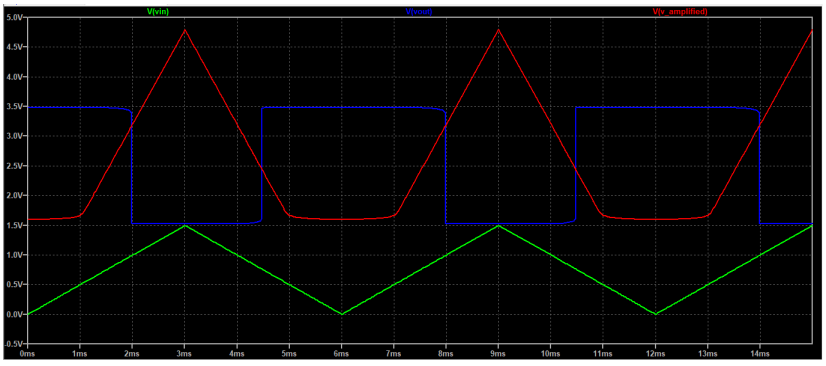

Figure 6.6.: The results of simulation.

6.2.2 Output

The localization system meets the following specifications:

- Accurate path determination from current location to target location.
- Real-time localization using existing infrastructure and sensor fusion.
- Robust integration of various sensor inputs to maintain accuracy.

6.2.3 Hardware Output:

Schematic

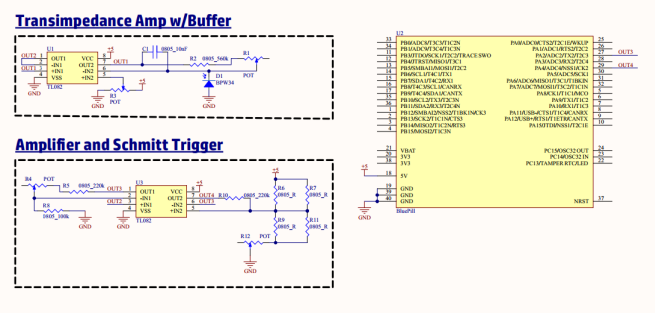

Figure 6.7.: Schematics for The VLC Receiver.

PCB

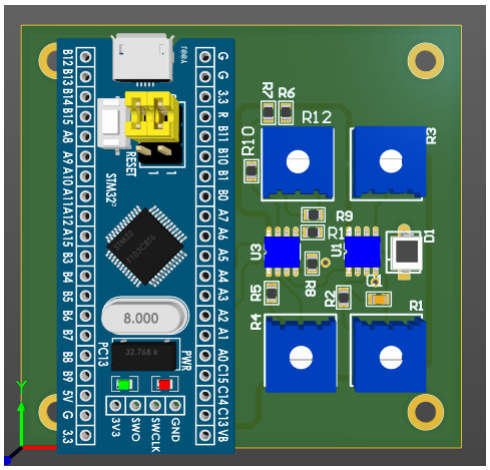

Figure 6.8.: PCB Output for The VLC Receiver.

Part IV

User Service Node

7 Web Application

7.1 Introduction

In this chapter, the proposed user service node model will be thoroughly examined—from the perspective of the web application; along with the development objectives in mind to achieve the desired performance and acceptable user experience.

7.2 Development Objectives

Among the development objectives stated in chapter 1, two in particular can be classified to be related to the user service node and one will be the main focus in this chapter: Graphical User Interface. The graphical user interface can be dissected into multiple sub-objectives as follows:

1. User Interface (UI) Design.

   Objective: Create an intuitive and user-friendly interface that allows users to easily navigate through the app.

   Details:

   - Develop a dashboard for quick access to main features.
   - Design task creation and map editing interfaces with drag-and-drop functionality.
   - Ensure responsive design for compatibility with various devices.

2. Task Management System.

   Objective: Develop a robust task management system for creating, scheduling, and monitoring tasks.

   Details:

   - Allow users to create, edit, and delete tasks.
   - Enable task scheduling.
   - Assign Task to a robot easily.

3. Reliable data storage and management.

   Objective: Implement a reliable data storage solution for tasks, maps, and user data.

   Details:

   - Use a relational database (e.g., MYSQL) for structured data storage.
   - Save data like tasks and maps in JSON files for easy access and modification.
   - Ensure data integrity and implement backup and recovery mechanisms.
   - Implement data encryption for sensitive information.

4. Map Creation and Management.

   Objective: Provide tools for users to create and manage maps for robot navigation.

   Details:

   - Implement a map editor with features for drawing and editing paths, obstacles, and waypoints.
   - Allow users to save and load maps.
   - Integrate map data with the robot's navigation system for real-time updates.

5. Real-Time Control and Monitoring

   Objective: Enable real-time control and monitoring of the robot.

   Details:

   - Implement controls for manual operation of the robot.
   - Display real-time data such as status, motors' speed and position.

6. Integration with Robot's API

   Objective: Ensure seamless integration with the robot's API for command execution and data retrieval.

   Details:

   - Implement API endpoints for sending commands to the robot.
   - Retrieve and display various data from the robot.
   - Handle API error responses and provide appropriate user feedback.

7. Python API Development

   Objective: Develop a Python-based API for enhanced functionality and interoperability.

   Details:

   - Design and implement a Python API for additional command execution and data retrieval.
   - Utilize web sockets for real-time communication and updates between the client and server.
   - Ensure the Python API integrates seamlessly with the existing system.
   - Provide detailed documentation and examples for using the Python API.

8. Performance Optimization

   Objective: Optimize the web app for performance and scalability.

   Details:

   - Use efficient algorithms for task scheduling and map processing.
   - Implement caching strategies to reduce load times.

7.3 Used Tools and Frameworks

A comprehensive robot control system is implemented using a combination of front-end and back-end technologies. The front-end [1] is developed with React.js and TypeScript, utilizing libraries such as React Router DOM, Axios, React Flow, and various hooks (`useState`, `useContext`, and `useEffect`). The back end is built with Laravel[2], while Python scripts are executed to perform specific tasks. This document details the frameworks and libraries used, the reasons behind their selection, and their roles in the overall system.

7.3.1 Front-End Implementation

A) React.js and TypeScript

React.js is a popular JavaScript library for building user interfaces, particularly single-page applications (SPAs). It allows developers to create reusable UI components, making the development process more efficient and manageable [8].

[1] Full code implementation can be viewed at: `https://github.com/meh-land/web_App`
[2] Full code implementation can be viewed at: `https://shorturl.at/VdtOP`, `https://shorturl.at/6a0cM`

TypeScript is a superset of JavaScript that adds static type definitions, enabling developers to catch errors early in the development process and improve code maintainability and readability [9].

- **Create a new React.js project**:

  ```
  npx create-react-app my-app
  ```

- **Create a new React.js project with TypeScript**:

  ```
  npx create-react-app my-app --template typescript
  ```

- **Navigate to the project directory**:

  ```
  cd my-app
  ```

- **Start the development server**:

  ```
  npm start
  ```

B) React Router DOM

React Router DOM is a routing library for React that enables the creation of dynamic and complex navigation structures in web applications. It manages the navigation between different components and pages in the application [10] [11].

- **Usage**: It allows the robot control system to have multiple views, such as robots, maps, tasks, and dashboard, while maintaining a seamless user experience. It ensures that users can easily switch between various functionalities of the robot control system. To install React Router DOM, you can use the following npm command [11]:

  ```
  npm install react-router-dom
  ```

  Once React Router DOM have been installed, import the needed components for the demo inside App.js/.tsx. Then add the following code snippets:

```
1  import { BrowserRouter, Route, Link, Routes }
2      from "react-router-dom";
3  // Import the pages
4  import Page1 from "./Components/page1"
5  function App() {
6      return (
7          <div className="App">
8              <BrowserRouter>
9                  <Routes>
10                     <Route exact path="/" element={<h1>Home
                           Page</h1>} />
```

Code Snippet 7.1: React Router Dom Implementation (1)

```
                    <Route exact path="page1" element={<Page1
                        />} />
                </Routes>
            </BrowserRouter>
        </div>
    );
}
export default App;
```

Code Snippet 7.2: React Router Dom Implementation (2)

C) Axios

Axios is a promise-based HTTP client for making requests to external APIs or back-end services. It simplifies the process of sending asynchronous HTTP requests and handling responses [12].

- **Usage**: Axios is used for communication between the front-end React application and the back-end Laravel server. It handles tasks such as fetching real-time robot data, sending control commands, and retrieving analytics information.To install axios, you can use the following npm command:

  ```
  npm install axios
  ```

  Once you have installed Axios, it can be used as follows:

```
import axios from 'axios';

// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .finally(function () {
    // always executed
  });
```

Code Snippet 7.3: Axios get request Implementation (1)

```
1
2  // Optionally the request above could also be done as
3  axios.get('/user', {
4      params: {
5        ID: 12345
6      }
7    })
8    .then(function (response) {
9      console.log(response);
10   })
11   .catch(function (error) {
12     console.log(error);
13   })
14   .finally(function () {
15     // always executed
16   });
17
18 // Want to use async/await? Add the 'async' keyword to your
     outer function/method.
19 async function getUser() {
20   try {
21     const response = await axios.get('/user?ID=12345');
22     console.log(response);
23   } catch (error) {
24     console.error(error);
25   }
26 }
```

Code Snippet 7.4: Axios get request Implementation (2)

D) socket.io-client

socket.io-client package is a library that enables real-time, bidirectional, and event-based communication between the client and server. It is particularly useful for applications that require instant updates, such as real-time notifications [13] [14].

- **Usage**: socket.io-client package is used to enable real-time updates by listening for changes in logs, status and Motors' speed from the server, and if there is change, new data is displayed.[3] To install socket.io-client package, you can use the following npm command:

```
npm install socket.io-client
```

[3] Full code implementation can be viewed at: https://github.com/meh-land/web_App

Once you have installed the Socket.IO client library, you can initialize the client as follows:

1. **Import the `io` function from the `socket.io-client` package**:

```
import io from "socket.io-client";
```

Code Snippet 7.5: Import the io function from the socket.io-client package

This imports the `io` function from the `socket.io-client` package, which is used to establish a connection with the Socket.IO server.

2. **Initialize the socket connection**:

```
// If your front-end is served on the same domain as your
   server, you can use the default initialization:
const socket = io();

// If your front-end is not served from the same domain as
   your server, pass the server URL to the io function:
const socket = io("https://server-domain.com");
```

Code Snippet 7.6: Initialize the socket connection

If the front-end and server are served from the same domain, the default `io()` initialization is sufficient. If they are served from different domains, you need to provide the server URL.

3. **Initiate connection and listen for events**:

```
// In this example, we listen for the "file_changed_logs"
   event and update the logs state accordingly
socket.on("file_changed_logs", (data) => {
  // Update the logs state with the new data
  setLogs((prevLogs) => [data.data, ...prevLogs]);
  // Log the new data to the console for debugging purposes
  console.log(data.data);
});
```

Code Snippet 7.7: Initiate connection and listen for events

Here, we set up a listener for the `file_changed_logs` event. When this event is received, the `data` is used to update the `logs` state and is also logged to the console.

4. **Cleanup on component unmount**:

```
return () => {
  // Remove the "file_changed_logs" event listener
  socket.off("file_changed_logs");
};
```

Code Snippet 7.8: Cleanup on component unmount

This ensures that the `file_changed_logs` event listener is removed when the component is unmounted, preventing potential memory leaks.

By following these steps, you can effectively manage real-time updates in your application using the Socket.IO client library.

Using Flask and Socket.IO for Real-Time File Monitoring [4]

The following steps demonstrate how to use Flask and Socket.IO to monitor file changes in real-time and notify clients of updates.

1. **Import the necessary libraries**:

```
from flask import Flask
from flask_socketio import SocketIO
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import threading
import time
```

Code Snippet 7.9: Import the necessary libraries

These imports bring in the necessary libraries for creating a Flask application, handling real-time communication with Socket.IO, and monitoring file changes with watchdog.

2. **Initialize the Flask app and Socket.IO**:

```
app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins="*")
```

Code Snippet 7.10: Initialize the Flask application and sets up Socket.IO with cross-origin support.

[4] Full code implementation can be viewed at: https://shorturl.at/M1oNT

3. **Function to monitor file changes**:

```
def monitor_file(file_path, socket_name):
    event_handler = FileChangeHandler(file_path,
        socket_name)
    observer = Observer()
    observer.schedule(event_handler,
        path='C:\\xampp\\htdocs\\dashboard_ros\\storage\\app',
        recursive=False)
    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

Code Snippet 7.11: Function to monitor file changes.

4. **Run the Flask application with Socket.IO**:

```
if __name__ == '__main__':
    threads = []
    for socket_name, path in file_paths.items():
        thread = threading.Thread(target=monitor_file,
            args=(path, socket_name))
        thread.daemon = True
        thread.start()
        threads.append(thread)
    socketio.run(app, debug=True)
```

Code Snippet 7.12: Run the Flask application with Socket.IO.

Creates and starts threads to monitor each file and runs the Flask application with Socket.IO in debug mode.

E) React Flow

React Flow is a library for building node-based graphical interfaces. It provides tools to create interactive flow diagrams, which are essential for visualizing complex systems and processes [15] [16].

- **Usage**: React Flow is utilized to design and visualize the path of the robot. It allows users to create, modify, and monitor the robot's movement through an intuitive graphical interface. To install reactflow, you can use the following npm command:

```
npm install reactflow
```

Once you have installed reactflow, you can use it as follows:

```
import { useCallback } from 'react';
import ReactFlow, {
  MiniMap,
  Controls,
  Background,
  useNodesState,
  useEdgesState,
  addEdge,
} from 'reactflow';
import 'reactflow/dist/style.css';
const initialNodes = [
  { id: '1', position: { x: 0, y: 0 }, data: { label: '1' } },
  { id: '2', position: { x: 0, y: 100 }, data: { label: '2' }
    },
];

const initialEdges = [{ id: 'e1-2', source: '1', target: '2'
  }];

function Flow() {
  const [nodes, setNodes, onNodesChange] =
    useNodesState(initialNodes);
  const [edges, setEdges, onEdgesChange] =
    useEdgesState(initialEdges);

  const onConnect = useCallback((params) => setEdges((eds) =>
    addEdge(params, eds)), [setEdges]);

  return (
    <ReactFlow
      nodes={nodes}
      edges={edges}
      onNodesChange={onNodesChange}
      onEdgesChange={onEdgesChange}
      onConnect={onConnect}
    >
      <MiniMap />
      <Controls />
      <Background />
    </ReactFlow>
  );
}
```

Code Snippet 7.13: React Flow Implementation.

F) React Hooks

React hooks are functions that let developers use state and other React features without writing a class [17] [18]. The primary hooks used in this project include:

- `useState`: Manages state within functional components [19].
- `useContext`: Shares state and data across components without prop drilling [20].
- `useEffect`: Handles side effects such as data fetching and updating the DOM [21].

The hooks mentioned above are employed to manage the application's state, handle data fetching, update the user interface in response to state changes, and share data between components efficiently.

7.3.2 Back-End Implementation

A) Laravel

Laravel is a PHP framework designed for web application development. It follows the Model-View-Controller (MVC) architectural pattern, providing a robust structure for building scalable and maintainable back-end systems [22].

- **Usage**: Laravel serves as the back-end framework for the robot control system. It handles user authentication, data storage, and communication with the front end. The framework's built-in features, such as routing, middleware, Passport, and caching, streamline the development process and ensure a secure and efficient server-side operation.
  - **Middleware**: Middleware in Laravel acts as a filtering mechanism for HTTP requests entering your application. It ensures that only authenticated and authorized users can access specific routes and resources [23].
    1. To create middleware, use the following command:

       ```
       php artisan make:middleware MiddlewareName
       ```

    2. Middleware in Laravel acts as a filtering mechanism for HTTP requests entering your application. It ensures that only authenticated users can access specific routes and resources.

```php
Route::middleware(['auth:api'])->group(function () {
    // define routes here
});
```

Code Snippet 7.14: Middleware usage to protect routes in a Laravel application..

- **Passport**: Laravel Passport provides a full OAuth2 server implementation for Laravel applications. It is used for API authentication, allowing the robot control system to securely authenticate users and manage tokens [24]. To install Passport, use the following commands:

```
composer require laravel/passport
php artisan passport:install
```

- **File Writing**: Laravel provides a robust file storage system that supports local file storage. This is used to store necessary data required by the robot control system [25].

  When using the local driver, all file operations are relative to the root directory defined in your filesystems configuration file. By default, this value is set to the storage/app directory. Therefore, the following method would write to storage/app/example.txt:

```php
use Illuminate\Support\Facades\Storage;

Storage::disk('local')->put('example.txt', 'Contents');
```

Code Snippet 7.15: Writing data to file in storage/app in laravel project.

B) Python API (Flask)

Flask is a micro web framework written in Python. It is designed to be simple and flexible, allowing developers to build web applications quickly and with minimal overhead. Flask provides the essentials needed to get a web server up and running but leaves many decisions about architecture and additional features up to the developer. This makes it a popular choice for both small applications and large-scale projects [26].

- **Usage**: Flask in this code serves as the foundation for creating a web server that can handle HTTP requests and integrate with SocketIO for real-time communication. It provides the necessary infrastructure to serve web pages and handle client connections, while the SocketIO integration allows for dynamic updates based on file changes detected by the watchdog observers.

- **Commands**:
  - **Install Flask and Flask-SocketIO**:

```
pip install flask flask-socketio
```

  - **Install Watchdog** (for file system monitoring):

```
pip install watchdog
```

C) ROS (rospy) Scripts

Python is a versatile programming language known for its simplicity and extensive library support. It is widely used in robotics for tasks such as data processing, machine learning, and control algorithms. In this context, **rospy** is the Python client library for ROS (Robot Operating System), which facilitates communication and coordination between various components in a robotic system.

- **Usage**: rospy scripts are executed within the Laravel framework to perform specific tasks that require complex computations or interactions with the robot hardware. These tasks include syncing tasks, and executing control algorithms to operate the robot. rospy scripts leverage ROS topics, services, and actions to communicate with the robot and control its behavior in real-time.

The combination of React.js, TypeScript, React Router DOM, Axios, React Flow, and React hooks on the front end, along with Laravel and Python scripts on the back end, provides a powerful and efficient architecture for the robot control system. Each framework and library was chosen for its specific strengths and capabilities, ensuring a robust, maintainable, and scalable solution for managing and controlling the robot. This comprehensive setup enables the development of an intuitive user interface, efficient data handling, and seamless integration with the robot hardware, making it an ideal choice for this graduation project.

7.4 Design and Development Strategy

The Robot Control System Web Application is designed to provide an intuitive and efficient interface for controlling and monitoring robots in various environments. This project aims to leverage modern web technologies to create a versatile platform that can be accessed from any device with internet connectivity.

7.4.1 System Architecture

The system architecture follows the Model-View-Controller (MVC) concept to ensure a clear separation of concerns and maintainability.

A) Client-Side Application

The client-side application is developed using React.js, a popular JavaScript library for building user interfaces. The application will feature a responsive design to ensure it functions well on both desktop and mobile devices. Key components include:

- **View (UI Components)**:
  - **Dashboard**: A central hub displaying the status and controls for connected robots.
  - **Control Panel**: An interface for sending commands to the robots, including movement, actions, and other functions.
  - **Maps Creation**: A feature to create maps/paths for robots to follow.
  - **Tasks Management**: A feature to assign, monitor, and manage tasks for the robots.

B) Server-Side Application

The server-side application will handle communication between the client-side application and the robots, manage user authentication, and store data. Key components include:

- **Controller**:
  - **REST API**: Endpoints for retrieving and updating robot data, user information, and application settings.
  - **Business Logic**: Processing commands from the client-side application and executing corresponding actions.
- **Model**:
  - **Database**: A MySQL database for storing persistent data such as user profiles, robot configurations, maps, and tasks.

C) Robot Control Integration

The Robot Control Integration ensures seamless communication and control between the web application and the physical robots. Key components include:

- **Command Execution (Controller)**: The control units receive commands from the server-side application's REST API and execute movement and action instructions sent from the client-side application.
- **Real-Time Feedback (Model)**: The control units send real-time data from sensors and motors back to the server-side application. This data is stored in the database and used to update the client-side application's Dashboard and other interfaces.
- **Connectivity (View)**: The control units maintain a reliable and secure connection to the server-side application via Wi-Fi or other wireless protocols, ensuring continuous operation and efficient communication.

By adhering to the MVC architecture, the system ensures a structured and scalable approach to development, enhancing maintainability and ease of integration.

7.4.2 User Interface Design

The user interface (UI) design will prioritize ease of use, clarity, and accessibility. It will follow a modular design approach, with reusable components for different parts of the application.

A) Dashboard

The dashboard will provide an overview of connected robot, displaying its current status, location, and speed. Users can quickly select a robot to view more detailed information or take control.

B) Control Panel

The control panel will offer an intuitive layout for sending commands to the selected robot.

C) Maps Creation

Tools for creating and managing maps or paths that robots can follow. This includes the ability to define way points, set navigation routes, and visualize the paths in a user-friendly manner.

D) Tasks Management

Functionality for creating, scheduling, and assigning tasks to robots. This includes setting task priorities, monitoring task progress, and receiving notifications upon task completion or failure.

7.4.3 Implementation Plan

The implementation of the Robot Control System Web Application follows an iterative development process, with key milestones including:

1. **Requirements Gathering**: Define detailed requirements and use cases for the application.
2. **Design Phase**: Create mockups, and a detailed system architecture.
3. **Development Phase**: Implement the client-side and server-side applications, followed by integration with the robot control units.
4. **Testing Phase**: Conduct thorough testing to ensure functionality, performance, and security.

7.5 Output

7.5.1 Home Page

The dashboard for managing robots, featuring sections for Robots, Maps, and Tasks. The sidebar allows users to navigate to different sections including Home, Robots, Maps, and Tasks. Figure 7.1 shows how the home page is presented.

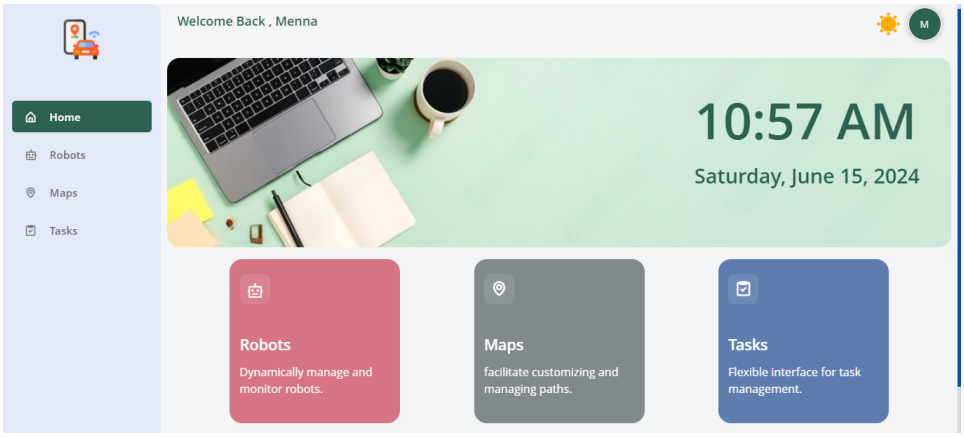

Figure 7.1.: Home page

7.5.2 Robots Management

The robots management dashboard—shown in figure 7.2—provides a comprehensive view of all robots. It includes a table displaying the robots and offers options to add new robots, edit existing robot details, or delete robots from the system.

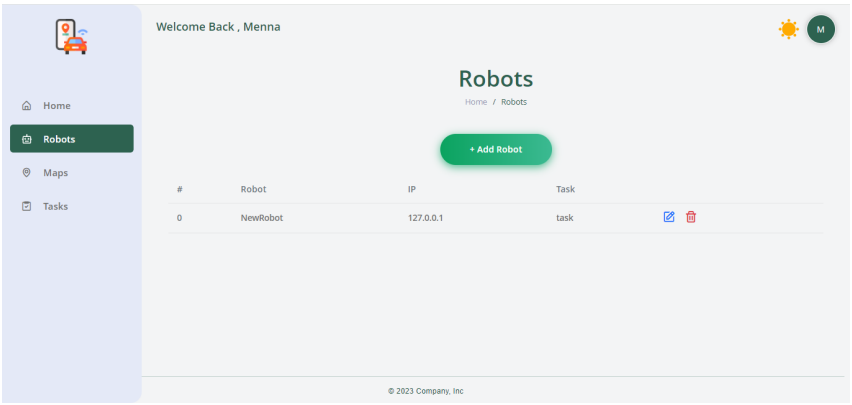

Figure 7.2.: Robots management dashboard

Figure 7.3 shows a popup form used for adding new robots to the management system. Users are prompted to enter the robot's name and IP address.

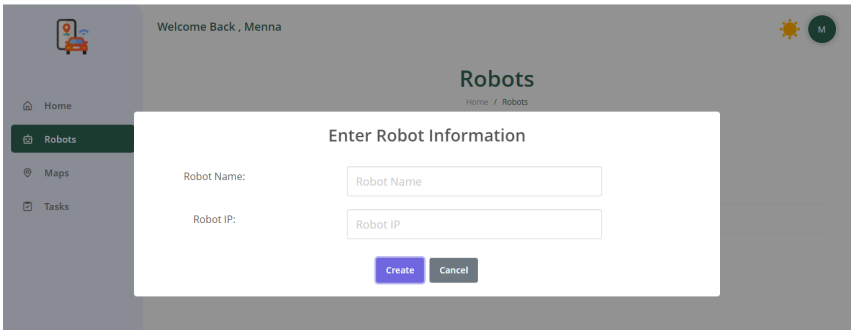

Figure 7.3.: Creating new robot

Figure 7.4 shows a popup form used for editing the information of existing robots. Users can update the robot's name, IP address, and assign task to the robot.

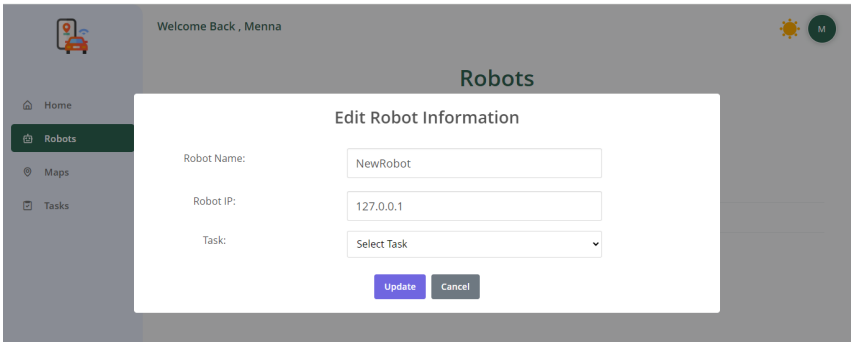

Figure 7.4.: Editing robot

7.5.3 Maps Creation

Figure 7.5 shows the maps management dashboard displays a table listing all maps in the system. It provides options to add new maps or delete existing ones, allowing users to manage their maps efficiently.

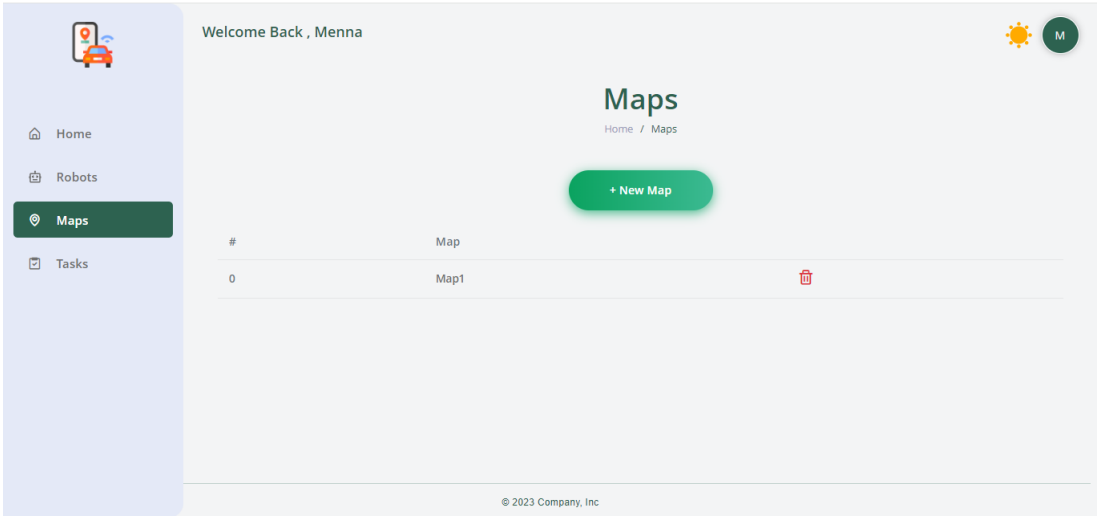

Figure 7.5.: Maps management dashboard

The node map editor interface that displays input, default, and output nodes is shown in figure 7.6. Users can add and name nodes, with guidance on the right side for editing and deleting nodes and connections. This tool facilitates the creation and customization of node maps.

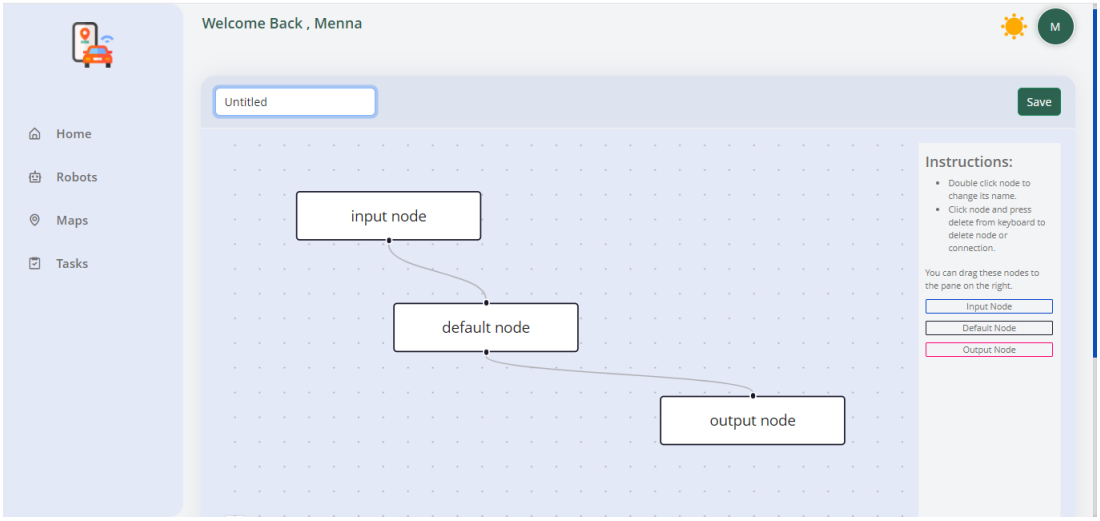

Figure 7.6.: Node map editor

The popup form in the node map editor—shown in figure 7.7—allows users to update node information. Users can change the node's label and coordinates, with fields provided for the new label and X and Y coordinates. Options to confirm or cancel the update are included.

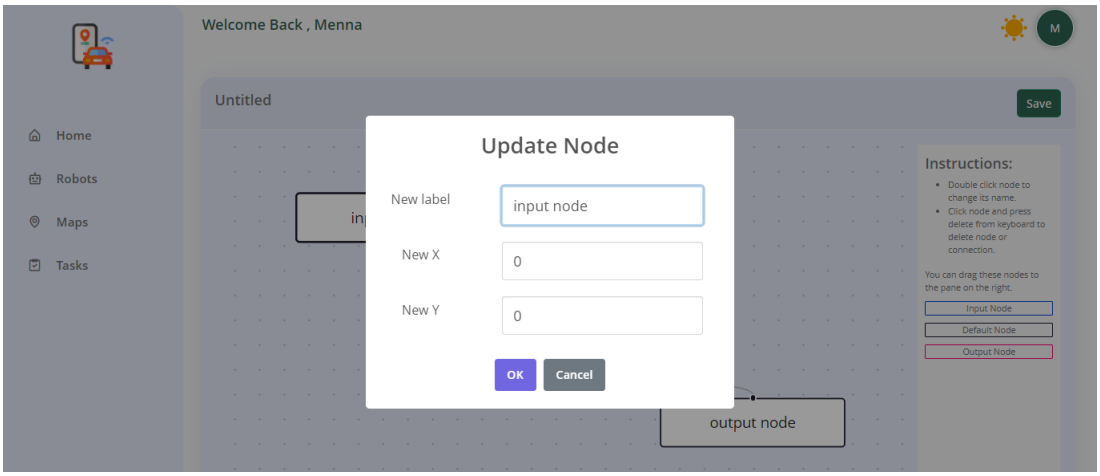

Figure 7.7.: Updating node information.

The node map display (figure 7.8) showcases a map titled 'Map1' with input, default, and output nodes connected sequentially. The interface provides an 'Edit Map' button, enabling users to modify the map as needed.

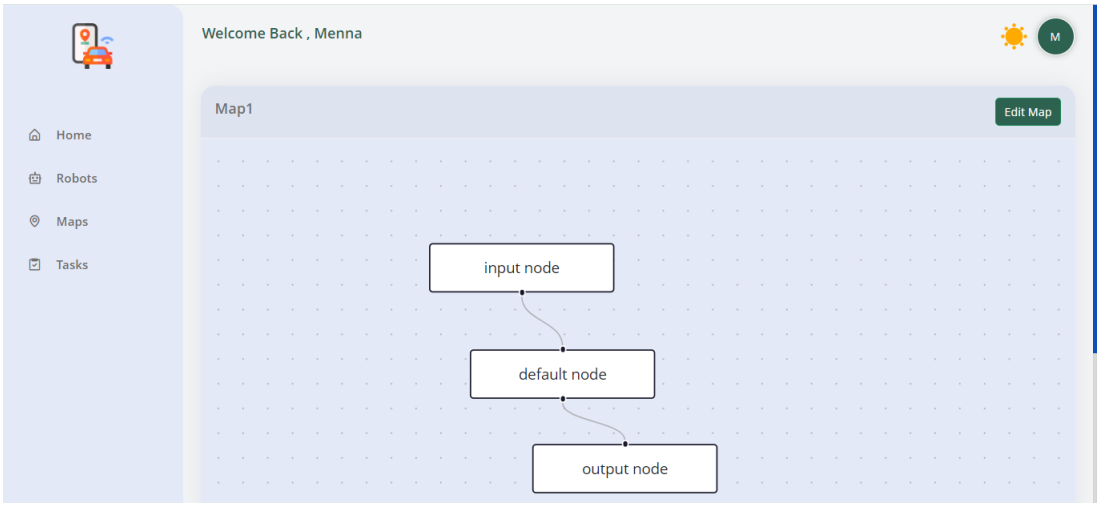

Figure 7.8.: Edit map

7.5.4 Tasks Management

The tasks management dashboard (figure 7.9) shows an empty state when no tasks have been added. An illustration and a message prompt users to add new tasks, with a button provided for this purpose.

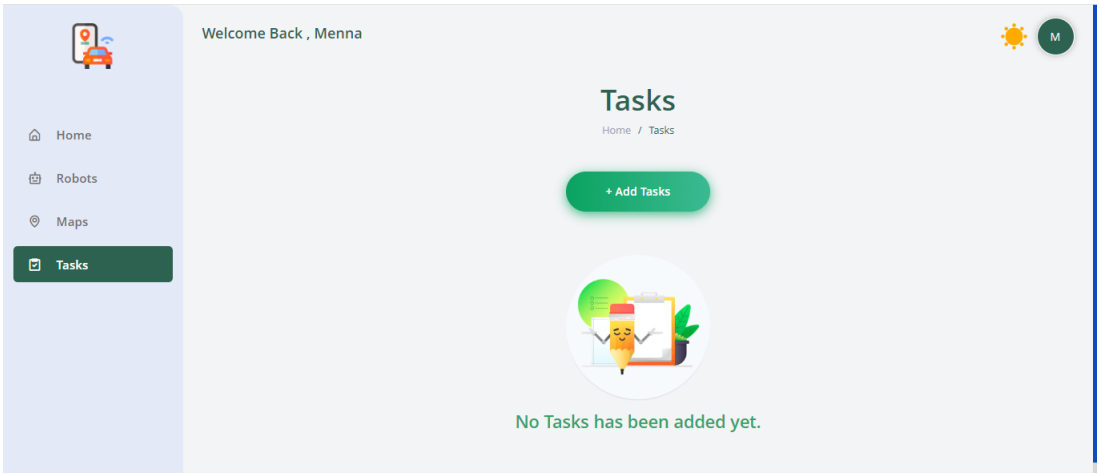

Figure 7.9.: Tasks management dashboard

The popup form in the tasks management dashboard—shown in figure 7.10—allows users to enter task information. Users can input details such as the task name, select a map, and specify pick-up and drop-off nodes along with the schedule time. Options to close or save the task are included.

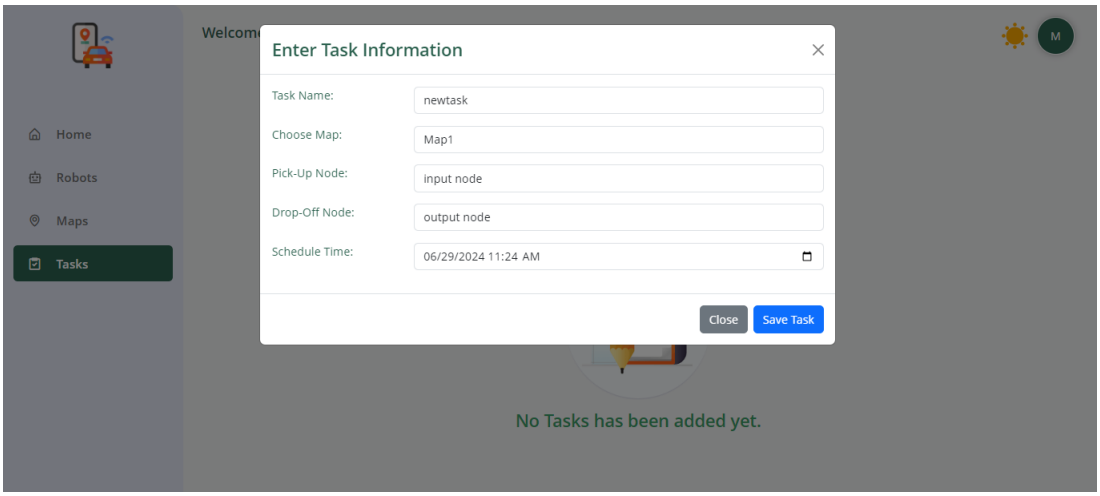

Figure 7.10.: New task creation

8 Server Infrastructure

In this chapter, the proposed user service node model will be thoroughly examined—from the perspective of the server infrastructure; along with the development objectives in mind to achieve the desired performance and acceptable user experience.

8.1 Development Objectives: Overview

Among the development objectives stated in chapter 1, two in particular can be classified to be related to the user service node; one was discussed in chapter 7 and the other will be the main focus in this chapter: Reliable Server Infrastructure. Making a reliable server infrastructure can be dissected into multiple sub-objectives as follows:

1. Creating a reliable server machine on which the web app can run.

2. Allow this server to communicate with the SBC.

3. Setup the server such that it has seamless integration withe rest of the network.

8.2 System Design

The server uses Ubuntu 20.04 LTS as its operating system, this is done to offer compatibility with ROS Noetic as mentioned in chapter 5. As it stands, the system does not run any ROS nodes on the server, but it could be useful for future improvements to have a high degree of compatibility between the server and the SBC.

Once the operating system is installed on the server, one must install all the packages required by the web app to operate correctly. Here is only a list of these packages and the utility of each one, the details of installation will be discussed in chapter 9.

8.2.1 Web Application Dependencies

Here is a list of all the dependencies needed for the correct and reliable operation of the web application. Note that the packages here are mentioned along their version numbers, this is to ensure the reliability of the setup. In other words, these versions are not strictly necessary but these are the only tested versions. Thus they are guaranteed to work coherently.

- Apache2: It is the web server of choice.
- MySQL: It is the used database.
- PHP 8.1: All backend is built using PHP.
- Composer: An application-level dependency manager for the PHP programming language that provides a standard format for managing dependencies of PHP software and required libraries.
- Laravel 10: The PHP framework used to build the backend.
- NPM 8.11: It is the package manager for front-end.
- nodejs 16.15: Provides the necessary JavaScript runtime environment and tooling that allows you to effectively use and manage packages through NPM.

8.2.2 Firewall Configuration

By default Ubuntu 20.04 uses the Uncomplicated Firewall (UFW) [27], which by default blocks all ports on the host. For the web application to be accessible, UFW needs to be configured in such a way that it allows the appropriate ports. The front-end runs on port 3000 and the back-end runs on port 8000. Thus, ports 3000 and 8000 need to be exposed to the external network. The bash script shown in code snippet 8.1 shows the commands needed to expose the needed ports and refresh UFW in order to make sure that UFW loads the most up-to-date configurations. Note script 8.1 needs sudo privilege to execute correctly.

```
1 #!/bin/bash
2 ufw allow 3000
3 ufw allow 8000
4 systemctl restart ufw
```

Code Snippet 8.1: Configure UFW to allow ports

Part V

Results & Conclusion

9 Setup For A Test Run

This chapter assumes that the user already has a working server with Ubuntu 20.04 LTS installed on it. The goal is to setup this server in a way that replicates our environment.

9.1 Step 1: Install Dependencies on Server

Code snippet 9.1 shows the commands needed to make sure that the server is up-to-date and has some basic dependencies installed.

```
1 #!/bin/bash -eu
2 sudo apt update
3
4 sudo apt install curl -y
5 sudo apt install git -y
6 sudo apt install net-tools -y
7 sudo apt upgrade -y
```

Code Snippet 9.1: Install general dependencies

Code snippet 9.2 installs the apache web server, and uses systemd to ensure that apache runs on system startup.

```
1 sudo apt install apache2 -y
2
3 sudo systemctl start apache2
4
5 sudo systemctl enable apache2
```

Code Snippet 9.2: Install Apache web server

Code snippet 9.3 shows the commands needed to install the mysql-server package and the command needed to set it up in a secure manner. Note that the second command will need some user input to complete, mainly setting up configurations in a way that is suitable for your specific use-case and setup.

```
1 sudo apt install mysql-server -y
2 sudo mysql_secure_installation
```

Code Snippet 9.3: Install and setup MySQL

We need to use PHP version 8.1.0 which is not readily available for Ubuntu 20.04, but luckily PHP is open source and we can download its source code and build it ourselves. Code snippet 9.4 shows the following steps:

1. Install dependencies needed for the build process.
2. Download the source code for PHP 8.1.0 as a compressed archive and extract it.
3. Configure the build environment.
4. Build PHP 8.1.0
5. Add PHP to the PATH environment variable to make sure the executable file is visible system-wide. This step is repeated every time we use an interactive shell, so we add it to the user's .bashrc file.

```
sudo apt install -y build-essential libxml2-dev libsqlite3-dev
    curl libcurl4-openssl-dev pkg-config libssl-dev libonig-dev
    libzip-dev zlib1g-dev libpng-dev libjpeg-dev libfreetype6-dev
    libgmp-dev libpq-dev libicu-dev libbz2-dev libxpm-dev
    libwebp-dev libtidy-dev libxslt1-dev zip

cd ~
wget https://www.php.net/distributions/php-8.1.0.tar.gz

tar -xzf php-8.1.0.tar.gz

cd php-8.1.0
./configure --prefix=/usr/local/php --with-openssl --with-zlib
    --enable-bcmath --with-bz2 --enable-calendar --with-curl
    --enable-exif --enable-ftp --with-gd --enable-mbstring
    --with-mysqli --with-pdo-mysql --enable-soap --enable-sockets
    --enable-sysvsem --enable-sysvshm --enable-shmop --with-tidy
    --with-xmlrpc --with-xsl --enable-zip --with-pear --with-webp
    --with-jpeg --with-xpm --with-freetype --enable-intl

make
sudo make install

echo 'export PATH="/usr/local/php/bin:$PATH"' >> ~/.bashrc
export PATH="/usr/local/php/bin:$PATH"
source ~/.bashrc
```

Code Snippet 9.4: Install and setup PHP

Code snippet 9.5 shows the process of installing composer by downloading its official installation script and piping it into php.

```
cd ~
curl -sS https://getcomposer.org/installer | php
sudo mv composer.phar /usr/local/bin/composer
```

Code Snippet 9.5: Install and setup composer

Code snippet 9.6 shows the commands needed to install Laravel.

```
cd ~
composer global require laravel/installer

export PATH="$PATH:$HOME/.config/composer/vendor/bin"
```

Code Snippet 9.6: Install and setup laravel

Code snippet 9.7 shows the commands needed to install NPM.

```
cd ~
curl
   https://raw.githubusercontent.com/creationix/nvm/master/install.sh
   | bash
source ~/.bashrc

nvm install 16.15.1
nvm use 16.15.1

npm install -g npm@8.11.0
```

Code Snippet 9.7: Install NPM

The first command in code snippet 9.8 is entered into bash to enter the MYSQL shell. Once in there, one should create a user and a password for said user, this is done to avoid using the root user for day-to-day operations. Then one must give the new user the appropriate privileges.

```
sudo mysql
CREATE USER 'my_username'@'localhost' IDENTIFIED BY 'my_password';
GRANT ALL PRIVILEGES ON *.* TO 'my_username'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

Code Snippet 9.8: Setup MySQL user

Now that all web application dependencies have been installed, one can start downloading the source code for the web application's back-end as shown in code snippet 9.9. This is done by cloning the project's Gihub repository. Subsequently, one must install all application libraries using composer.

Then, we should create a new .env file which contains data that is user-specific and should not be uploaded to any public repository. To help the user crete this .env file, we have included a .env.example file which contains a list of the needed variables, and some of these variables have default values set for ease of use, of course the user is free to substitute these default values with values that fit their specific setup. In the .env.example there are some variables that have no default value, this is because they are entirely user and machine dependent.

The last three lines in code snippet 9.9 are used to finalize the installation and create the needed key and database.

```
#!/bin/bash -eu
cd ~
git clone https://github.com/meh-land/GP_laravel.git
cd GP_laravel

composer install
cp .env.example .env


php artisan key:generate
php artisan migrate
php artisan passport:install
```

Code Snippet 9.9: Install back-end

Similar to the back-end, the user will need to clone the Github repository of the front-end, then install all the libraries needed for correct operation using NPM as shown in code snippet 9.10

```
cd ~
git clone https://github.com/meh-land/web_App.git
cd Dashboard

npm install
```

Code Snippet 9.10: Install front-end

9.2 Step 2: Install dependencies on the SBC

The SBC has two components running on it:

1. Web application endpoint (dashboard back-end).

2. ROS scripts needed for communication, localization and control.

Setup for Dashboard Back-End

This part is almost identical to setting up the web application's back-end, because they are both built using the same technology stack. This means that code snippets 9.4 - 9.6 will be repeated on the SBC without any modification. Code snippet 9.11 shows the process of cloning the Github repository of the dashboard back-end and setting up the code to be run. This step is similar to code snippet 9.9.

```
#!/bin/bash -eu
cd ~
git clone https://github.com/meh-land/Dashboard-Backend.git
cd Dashboard-Backend

composer install
cp .env.example .env


php artisan key:generate
php artisan migrate
php artisan passport:install
```

Code Snippet 9.11: Install dashboard back-end

Note that there is no need to install MySQL on the SBC because it is not used.

Setup for ROS scripts

To install ROS Noetic there are many steps that need to be followed, but our environment is fairly standard, so there is a simple script that can be used to setup ROS Noetic on the SBC. The commands shown in code snippet 9.12 are the commands needed to download the script, make it executable and run it. By the end of its execution the SBC should have ROS Noetic ready.

```
wget -c
   https://raw.githubusercontent.com/qboticslabs/ros_install_noetic/
master/ros_install_noetic.sh
chmod +x ./ros_install_noetic.sh
./ros_install_noetic.sh
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
```

Code Snippet 9.12: Install ROS Noetic

Now that ROS Noetic is installed, one needs to download the ROS scripts from their Github Repositories. The commands shown in code snippet 9.13 shows the commands needed to get and setup the ROS scripts that specialize in high level control and web interactions.

```
cd ~
mkdir -p web-ws/src
cd ~/web-ws/
catkin_make
cd src
catkin_create_pkg web-control std_msgs geometry_msgs nav_msgs
   rospy roscpp
cd web-control
rm ./*
git clone https://github.com/meh-land/torta_web_control.git .
cd ~/web-ws
catkin_make
source ~/web-ws/devel/setup.bash
echo "source ~/web-ws/devel/setup.bash" >> ~/.bashrc
```

Code Snippet 9.13: Get ROS scripts

The commands shown in code snippet 9.14 are used to get ROS scripts that are concerned with interacting with the low level sensors and micro-controllers.

```
cd ~
git clone https://github.com/meh-land/Tortabot.git
cd ~/Tortabot/
catkin_make
source ~/Tortabot/devel/setup.bash
echo "source ~/Tortabot/devel/setup.bash" >> ~/.bashrc
```

Code Snippet 9.14: Get ROS scripts

9.3 Step 3: Start Web Server Components

As previously discussed, the web server consists of two main components, namely, front-end and back-end. Code snippet 9.15 shows the commands needed to start the application's front-end. It is evident from code snippet 9.15 that the application reads its user-specific variables from a .env file. In the case of the front-end, it is just the IP address of the server on which the application is running, this variable is used by the front-end to know the address of the back-end to which it must send its requests. Note that the user must substitute $srv-ip with the actual IP address of the server.

```
cd ~/web_App
echo "REACT_APP_IP=$srv-ip" > .env
npm start
```

Code Snippet 9.15: Start front-end

Unlike the front-end, the back-end component needs multiple variables to be edited in the .env file. All those variables are needed for the proper operation of the database. Code snippet 9.16 shows the process of editing the .env file. Note that all the sed commands can be replaced by simply opening the .env file in any text editor and manually adding the appropriate values for these variables, the user is free to choose whichever method suits their use-case.

```
cd ~/GP_laravel
cp .env.example .env
sed -i "s/DB_DATABASE=/DB_DATABASE=$databaseName/g"
sed -i "s/DB_USERNAME=/DB_USERNAME=$username/g"
sed -i "s/DB_PASSWORD=/DB_PASSWORD=$password/g"
php artisan serve --host=0.0.0.0
```

Code Snippet 9.16: Start back-end

Note that in code snippets 9.15 and 9.16, all words preceded with a $ are variables that the user must replace with their actual values for their specific circumstances.

9.4 Step 4: Start SBC Components

The SBC is the communication hub and the bridge between the high level components and their low level counterparts, henceforth, the SBC needs to have two components running on it:

1. Dashboard back-end.
2. ROS scripts.

9.4.1 Starting Dashboard Back-End

Code snippet 9.17 shows the commands needed to start the dashboard's back-end. Note that there is no need to edit the .env file, this is because the dashboard has no database, thus, no database environment variables. Despite that, it is still necessary to have a .env file because there are other environment variables that are needed for the proper operation of the dashboard, mainly the ROS commands are defined as environment variables. There are other environment variables but they are not of any concern.

Also note that the dashboard runs on port 8001, that is unlike its counterpart on the server (shown in code snippet 9.16) which runs on port 8000. In code snippet 9.16 there was no need to specify the port because port 8000 is the default port fro Laravel applications, but we chose to make the dashboard's back-end run on a different port in order to offer more flexibility for the user. Theoretically, there is nothing preventing the user from running the server components and SBC components on the same machine, so we chose to make the web application's back-end to be on port 8000 and the dashboard's back-end to be on port 8001 to allow the user such flexibility, which might be useful in troubleshooting situations, or in situations where the SBC acts as the both the server and the central processing node.

```
1 cd ~/Dashboard-Backend
2 cp .env.example .env
3 php artisan serve --host=0.0.0.0 --port=8001
```

Code Snippet 9.17: Start dashboard back-end

9.4.2 Starting ROS Environment

It has been mentioned previously that ROS scripts need some setup to operate correctly, namely they need a ROS master. In addition, there are some special requirements that are needed specifically for our application. To facilitate the operation, there is a ROS launch

script that can be used to initialize all needed components for communication between ROS nodes and for communication between ROS nodes running on the SBC and those running on the various micro-controllers. The command needed to start the entire ROS environment is shown in code snippet 9.18, and the ROS launch script itself is shown

```
roslaunch torta-bringup robot_low_level_w_odom.launch
```

Code Snippet 9.18: Start ROS environment

```
<launch>

    <node pkg="rosserial_python" type="serial_node.py"
       name="serial_node_0">
      <param name="port" value="/dev/ttyACM0" />
      <param name="name" value="stm1" />
    </node>

    <node pkg="rosserial_python" type="serial_node.py"
       name="serial_node_1">
      <param name="port" value="/dev/ttyACM1" />
      <param name="name" value="stm2" />
    </node>

    <node pkg="rosserial_python" type="serial_node.py"
       name="serial_node_2">
      <param name="port" value="/dev/ttyACM2" />
      <param name="name" value="stm3" />
    </node>




    <node pkg="manual" type="website_control.py"
       name="website_control">
    </node>

    <!-- Launch Dead_reckoning script-->
    <node name="dead_reck" pkg="odom"
       type="dead_reckoning_script.py" output="screen">
    </node>

  </launch>
```

Code Snippet 9.19: ROS launch

9.5 Step 5: Use The Application

If the setup was done correctly, the web application should be accessible from any device on the same network as the server by simply opening a web browser and accessing the server on port 3000. Then the user must sign-up if they are accessing the web application for the first time, sign-up is done using the screen shown in figure 9.1. In subsequent uses the user can sign-in using the sign-in screen shown in figure 9.2.

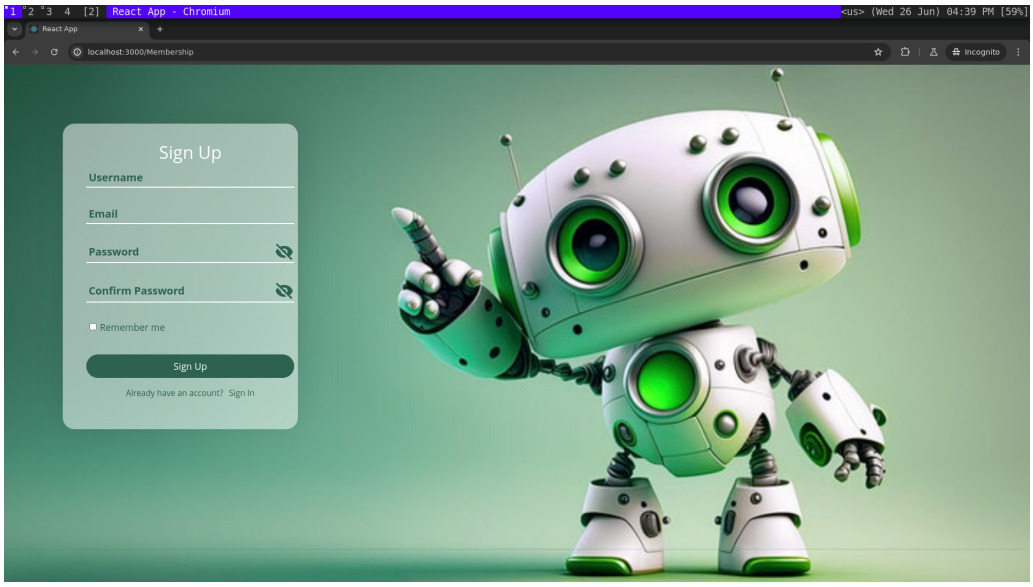

Figure 9.1.: Sign-up screen

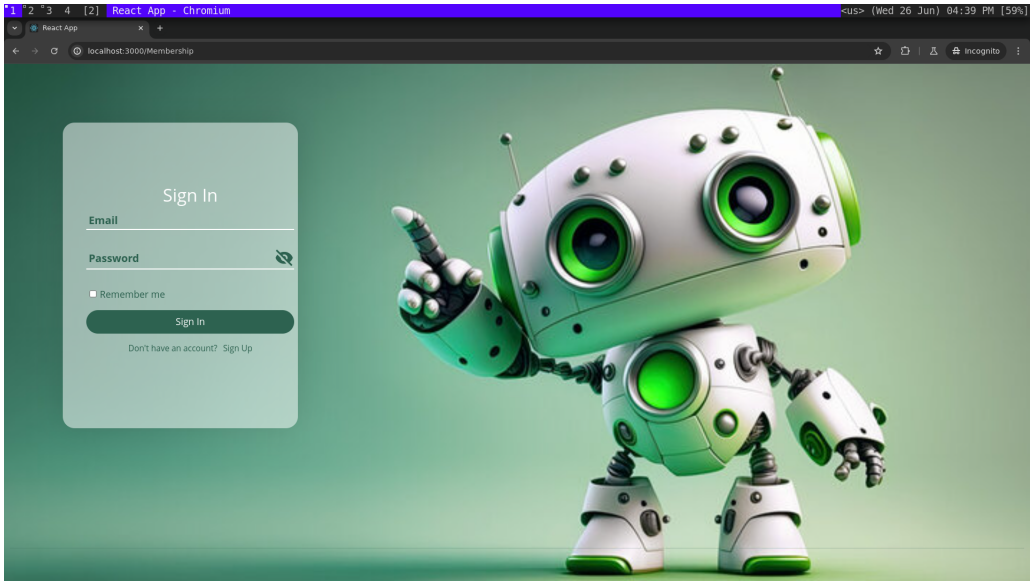

Figure 9.2.: Sign-in screen

Once logged in, the user can use the home screen shown in figure 7.1 to navigate the web

application. The typical use-case is that the user wants to create a task and assign it to a robot.

First the user must define the map in which the task will take place.Using the screen shown in figure 7.5, the user may create a new map or view predefined maps. If the user wishes to create a new map or edit a pre-existing one, they can use the map editor shown in figure 7.6.

Once the map is to the user's satisfaction, they can move on to task creation using the screen shown in 7.9. Similar to the map editor, this page allows the user to create, view, edit or delete tasks. If the user chooses to create a task, they can use the screen shown in figure 7.10 to specify the details of the task. Note that this window adapts according to the map that the user chooses, as the pick-up and drop-off nodes depend on the chosen map.

Now that the the task is properly defined, the last step is to assign this task to a robot. The robots management window is shown in figure 7.2, using this page the user can create a new robot as shown in figure 7.3, or edit a pre-existing robot. Editing a robot is the way in which the user can assign a task to a robot.

10 Future Work & Recommendations

10.1 Introduction

Although the project has fulfilled most if not all of its goals, there exists many areas that can be improved and/or built upon to achieve an even greater outcome. In this chapter, possible improvement ideas will be discussed in addition to recommendations for anyone willing to pick the project up where it was left off.

10.2 Areas For Improvements

There are many areas to improve in each layer of the robot. It can be summarized as follows:

1. Using a real-time operating system on all microcontrollers in order for the system to meet real-time demands.
2. Programming all microcontrollers in embedded C/C++ and abandoning Arduino platform for a better performing system.
3. Using MicroROS—which is built on the newer architecture of ROS 2—instead of ROS 1 for a more coherent and future-proof system.
4. Developing an embedded Linux image for the single board computer for minimum system overhead.
5. Containerizing the web application using Docker/Docker-Compose for more portability.
6. To enhance the web application, we should refactor our current React, Laravel, and Python components to a Next.js framework, leveraging its server-side rendering capabilities for improved performance and SEO.

Bibliography

[1] T. Instruments, *Analog Engineer's Circuit Amplifiers:Temperature Sensing with NTC Circuit*, 2022. [Online]. Available: https://bit.ly/3RAHu20

[2] D. Andrea, *Battery Management Systems for Large Lithium-Ion Battery Packs*. ARTECH HOUSE, 2010.

[3] E. Technologies, "Battery cell comparison," 2008, accessed: 2024-06-19. [Online]. Available: https://bit.ly/4cshLRm

[4] STMicroelectronics. (2024) Stm32f103c8. Accessed: 2024-06-23. [Online]. Available: https://shorturl.at/QDMz0

[5] AxiosContributors. (2024) Getting started with axios. Accessed: 2024-06-14. [Online]. Available: https://axios-http.com/docs/intro

[6] RsyncProject. (2024) Rsync: An open source utility for fast incremental file transfer. Accessed: 2024-06-14. [Online]. Available: https://github.com/RsyncProject/rsync

[7] O. Robotics, "Robot operating system (ros)," 2024, accessed: 2024-06-22. [Online]. Available: https://ros.org/

[8] ReactTeam. (2024) React – a javascript library for building user interfaces. Accessed: 2024-06-16. [Online]. Available: https://legacy.reactjs.org/

[9] ——. (2024) Using typescript with react. Accessed: 2024-06-16. [Online]. Available: https://react.dev/learn/typescript

[10] ReactRouterTeam. (2024) React router: Overview. Accessed: 2024-06-16. [Online]. Available: https://reactrouter.com/en/main/start/overview

[11] ——. (2024) react-router-dom. Accessed: 2024-06-16. [Online]. Available: https://www.npmjs.com/package/react-router-dom

[12] AxiosTeam. (2024) axios. Accessed: 2024-06-16. [Online]. Available: https://www.npmjs.com/package/axios

[13] Socket.IOTeam. (2024) socketio-client. Accessed: 2024-06-16. [Online]. Available: https://www.npmjs.com/package/socket.io-client

[14] ——. (2024) Using socket.io with react. Accessed: 2024-06-16. [Online]. Available: https://socket.io/how-to/use-with-react

[15] ReactFlowTeam. (2024) reactflow. Accessed: 2024-06-16. [Online]. Available: https://www.npmjs.com/package/reactflow

[16] ——. (2024) React flow: Examples. Accessed: 2024-06-16. [Online]. Available: https://reactflow.dev/examples

[17] ReactTeam. (2024) Introducing hooks. Accessed: 2024-06-16. [Online]. Available: https://legacy.reactjs.org/docs/hooks-intro.html

[18] ——. (2024) React: Hooks reference. Accessed: 2024-06-16. [Online]. Available: https://react.dev/reference/react/hooks

[19] ——. (2024) Using the state hook. Accessed: 2024-06-16. [Online]. Available: https://legacy.reactjs.org/docs/hooks-state.html

[20] ——. (2024) React: usecontext hook. Accessed: 2024-06-16. [Online]. Available: https://react.dev/reference/react/useContext

[21] ——. (2024) Using the effect hook. Accessed: 2024-06-16. [Online]. Available: https://legacy.reactjs.org/docs/hooks-effect.html

[22] LaravelTeam. (2024) Laravel documentation (version 11.x). Accessed: 2024-06-16. [Online]. Available: https://laravel.com/docs/11.x

[23] ——. (2024) Laravel middleware documentation. Accessed: 2024-06-16. [Online]. Available: https://laravel.com/docs/11.x/middleware#main-content

[24] ——. (2024) Laravel passport documentation. Accessed: 2024-06-16. [Online]. Available: https://laravel.com/docs/11.x/passport#main-content

[25] ——. (2024) Laravel filesystem documentation. Accessed: 2024-06-16. [Online]. Available: https://laravel.com/docs/11.x/filesystem#main-content

[26] FlaskTeam. (2024) Flask quickstart documentation. Accessed: 2024-06-16. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/quickstart/

[27] U. Community, "Ufw - uncomplicated firewall," 2024, accessed: 2024-06-22. [Online]. Available: https://help.ubuntu.com/community/UFW

Bill of Materials

A

A.1 Boards

A.1.1 Sensors' Board

B Theoretical Background – Sensors & PID

B.1 Encoders

A rotary encoder, also called a shaft encoder, is an electro-mechanical device that converts the angular position or motion of a shaft or axle to analog or digital output signals. There are two main types of rotary encoders: absolute and incremental. The output of an absolute encoder indicates the current shaft position, making it an angle transducer. The output of an incremental encoder provides information about the motion of the shaft, which typically is processed elsewhere into information such as position, speed, and distance.

B.1.1 Technologies used in Encoders

- **Optical**

  This uses a light shining onto a photodiode through slits in a metal or glass disc. Reflective versions also exist. This is one of the most common technologies. Optical encoders are very sensitive to dust. The resolutions of optical encoders are higher than magnetic encoders.

- **On-Axis Magnetic**

  This technology typically uses a specially magnetized 2-pole neodymium magnet attached to the motor shaft. Because it can be fixed to the end of the shaft, it can work with motors that only have 1 shaft extending out of the motor body. The accuracy can vary from a few degrees to under 1 degree. Resolutions can be as low as 1 degree or as high as 0.09 degrees (4000 CPR).

- **Off-Axis Magnetic**

  This technology typically employs the use of rubber-bonded ferrite magnets attached to a metal hub. This offers flexibility in design and low cost for custom applications. Due to the flexibility in many off-axis encoder chips, they can be programmed to accept any number of pole widths so the chip can be placed in any position required for the application. Magnetic encoders operate in harsh environments where optical encoders would fail to work.

B.1.2 Basic types

- **Absolute**

  This type maintains position information when power is removed from the encoder. The position of the encoder is available immediately on applying power. The relationship between the encoder value and the physical position of the controlled machinery is set at assembly; the system does not need to return to a calibration point to maintain position accuracy.

  An absolute encoder has multiple code rings with various binary weightings which provide a data word representing the absolute position of the encoder within one revolution. This type of encoder is often referred to as a parallel absolute encoder. A multi-turn absolute rotary encoder includes additional code wheels and toothed wheels. A high-resolution wheel measures the fractional rotation, and lower-resolution geared code wheels record the number of whole revolutions of the shaft.

- **Incremental**

  This type will immediately report changes in position, which is an essential capability in some applications. However, it does not report or keep track of the absolute position. As a result, the mechanical system monitored by an incremental encoder may have to be homed (moved to a fixed reference point) to initialize absolute position measurements. We will discuss the concept of working on incremental encoders and why we used them in our project.

B.1.3 Encoder Resolution Parameters

- **PPR(Pulses per Rotation):** describes the number of high pulses an encoder will have on either of its square waves outputs A or B over a single revolution. It is more common to represent the encoder resolution using PPR than CPR.

- **CPR(Counts per Rotation):** most commonly stands for Counts per Revolution, and refers to the number of quadrature decoded states that exist between the two outputs A and B. With both outputs A and B switching between high and low, there exist 2 bits of information represented as 4 distinct states. The term quadrature decoding describes the method of using both outputs A and B together to count each state change. This results in 4 times the number of counts that exist for each pulse or period. Therefore, the CPR of an encoder is the encoder's PPR multiplied by 4.

B.1.4 Encoder Working Principle

The encoder has a disk with evenly spaced contact zones that are connected to the common pin C and two other separate contact pins A and B, as illustrated below. When the disk

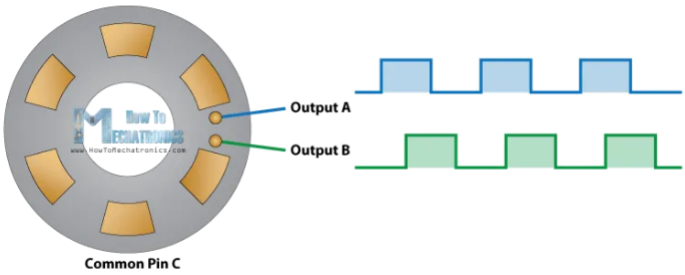

Figure B.1.: Pulses per Revolution Diagram

starts rotating step by step, the pins A and B will start making contact with the common pin and the two square wave output signals will be generated accordingly.

Any of the two outputs can be used for determining the rotated position if we just count the pulses of the signal. However, if we want to determine the rotation direction as well, we need to consider both signals at the same time.

We can notice that the two output signals are displaced at 90 degrees out of phase from each other. If the encoder is rotating clockwise output A will be ahead of output B. So if

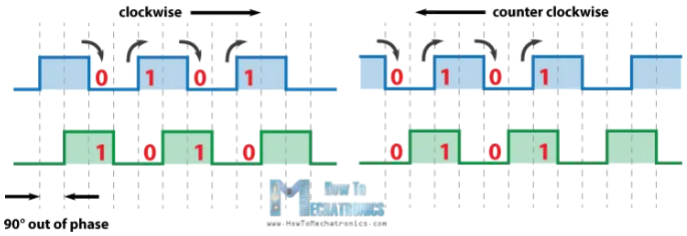

Figure B.2.: Encoder Output Signals

we count the steps each time the signal changes, from High to Low or from Low to High, we can notice that time the two output signals have opposite values. Vice versa, if the encoder is rotating counterclockwise, the output signals have equal values. So considering this, we can easily program our controller to read the encoder position and the rotation direction.

B.2 IMU (Inertial Measurement Units)

IMU sensors are based on multi-axis combinations of precision gyroscopes, accelerometers, magnetometers, and pressure sensors. its technology reliably senses and processes multiple degrees of freedom, even in highly complex applications and under dynamic conditions. These plugs-and-play solutions include full factory calibration, embedded compensation, and sensor processing, and a simple programmable interface. In our Project, we used MPU6050 IMU

B.2.1 MPU6050 Module

The MPU6050 sensor module is a complete 6-axis Motion Tracking Device. It combines a 3-axis Gyroscope, 3-axis Accelerometer, and Digital Motion Processor all in a small package. Also, it has the additional feature of the on-chip Temperature sensor. It has an I2C bus interface to communicate with the microcontrollers.

It has an Auxiliary I2C bus to communicate with other sensor devices like 3-axis Magnetometer, Pressure sensor, etc.

If a 3-axis Magnetometer is connected to an auxiliary I2C bus, then MPU6050 can provide a complete 9-axis Motion Fusion output. Let's see MPU6050 inside sensors.

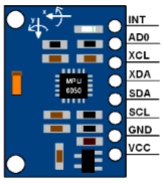

Figure B.3.: MPU6050 Module

- **3-Axis Gyroscope**

  The MPU6050 consists of a 3-axis Gyroscope with Micro Electro Mechanical System (MEMS) technology. It is used to detect rotational velocity along the X, Y, and Z axes as B.4.

  - When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a MEM inside MPU6050.
  - The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate.
  - This voltage is digitized using 16-bit ADC to sample each axis.

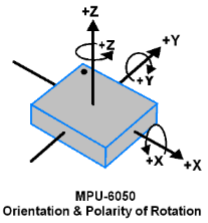

Figure B.4.: MPU-6050 3-Axis Gyroscope

- The full-scale range of output are +/- 250, +/- 500, +/- 1000, +/- 2000.
- It measures the angular velocity along each axis in degrees per second unit.

- **3-Axis Accelerometer**

  The MPU6050 consists 3-axis Accelerometer with Micro Electro Mechanical (MEMs) technology. It is used to detect the angle of tilt or inclination along the X, Y, and Z axes as B.5.

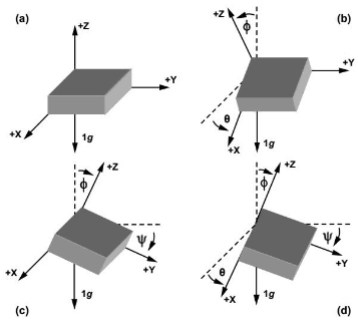

Figure B.5.: MPU-6050 3-Axis Accelerometer

- Acceleration along the axes deflects the movable mass.
- This displacement of the moving plate (mass) unbalances the differential capacitor which results in sensor output. Output amplitude is proportional to acceleration.
- The full-scale range of acceleration are +/- 2g, +/- 4g, +/- 8g, +/- 16g.
- It measures in g (gravity force) unit.
- When the device is placed on a flat surface it will measure 0g on the X and Y axis and +1g on the Z axis.

- **DMP (Digital Motion Processor)**

  The embedded Digital Motion Processor (DMP) is used to compute motion processing algorithms. It takes data from a gyroscope, accelerometer, and additional 3rd party

sensors such as a magnetometer and processes the data. It provides motion data like roll, pitch, yaw angles, landscape portrait sense, etc. It minimizes the processes of the host in computing motion data. The resulting data can be read from DMP registers.

B.3 Ultrasonic

An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves and converts the reflected sound into an electrical signal. Ultrasonic waves travel faster than the speed of audible sound (i.e. the sound that humans can hear).

Ultrasonic sensors have two main components: the transmitter (which emits the sound using piezoelectric crystals) and the receiver (which encounters the sound after it has traveled to and from the target).

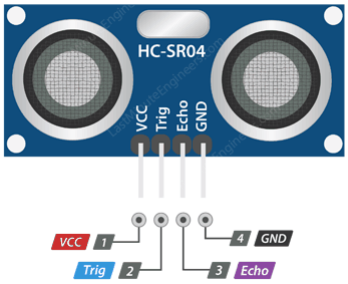

Figure B.6.: Ultrasonic Module

To calculate the distance between the sensor and the object, the sensor measures the time it takes between the emission of the sound by the transmitter to its contact with the receiver.

The formula for this calculation is

$$D = 0.5 * T * C$$

(where D is the distance, T is the time, and C is the speed of sound $\approx 343 m/s$).

For example, if a scientist set up an ultrasonic sensor aimed at a box and it took 0.025 seconds for the sound to bounce back, the distance between the ultrasonic sensor and the box would be:

$D = 0.5 * 0.025 * 343$ or about $4.2875 m$.

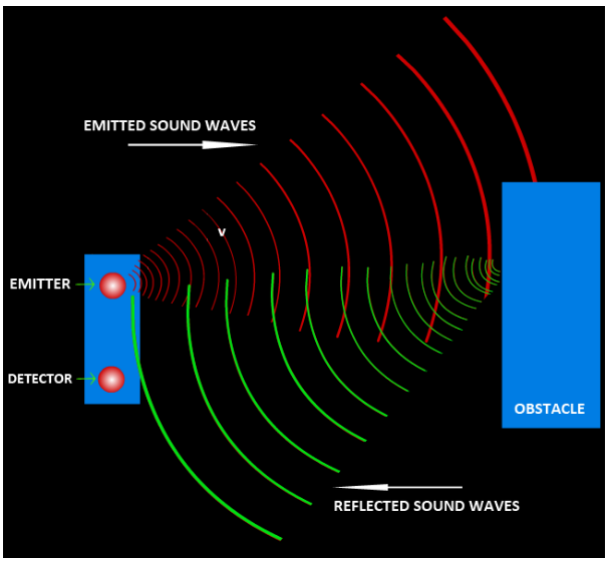

Figure B.7.: Ultrasonic sensor diagram

Ultrasonic sensors are used primarily as proximity sensors. They can be found in automobile self-parking technology and anti-collision safety systems. Ultrasonic sensors are also used in robotic obstacle detection systems, as well as manufacturing technology.

B.4 PID Control

B.4.1 Introduction

The term PID stands for proportional integral derivative, and it is one kind of device used to control different process variables like pressure, flow, temperature, and speed in industrial applications. In this controller, a control loop feedback device is used to regulate all the process variables and closed-loop feedback is used to maintain the real output from a method close to the objective, otherwise output at a fixed point if possible.

Proportional Integral Derivative (PID) control automatically adjusts a control output based on the difference between a set point (SP) and a measured process variable (PV). The value of the controller output $u(t)$ is transferred as the system input.

$$e(t) = SP - PV$$

$$u(t) = u_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \int_0^t e(t)dt - K_c \tau_D \frac{d(PV)}{dt}$$

The u_{bias} term is a constant that is typically set to the value of $u(t)$ when the controller is first switched from manual to automatic mode. This gives a "bumpless" transfer if the error is zero when the controller is turned on. The three tuning values for a PID controller are the controller gain, K_c, the integral time constant τ_I, and the derivative time constant τ_D The value of K_c is a multiplier on the proportional error and integral term and a higher value makes the controller more aggressive at responding to errors away from the set point. The integral time constant τ_I (also known as integral reset time) must be positive and have units of time. As τ_I gets smaller, the integral term is larger because τ_I is in the denominator. Derivative time constant τ_D also has units of time and must be positive. The set point (SP) is the target value and process variable (PV) is the measured value that may deviate from the desired value. The error from the set point is the difference between the SP and PV and is defined as $e(t) = SP - PV$.

B.4.2 Overview of PID Control

PI or PID controller is best suited for non-integrating processes, meaning any process that eventually returns to the same output given the same set of inputs and disturbances. A P-only controller is best suited to integrating processes. Integral action is used to remove offset and can be thought of as an adjustable u_{bias}

B.4.3 Discrete PID Controller

Digital controllers are implemented with discrete sampling periods and a discrete form of the PID equation is needed to approximate the integral of the error and the derivative. This modification replaces the continuous form of the integral with a summation of the error and uses Δt as the time between sampling instances and n_t as the number of sampling instances. It also replaces the derivative with either a filtered version of the derivative or another method to approximate the instantaneous slope of the (PV).

$$u(t) = u_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \sum_{i=1}^{n_t} e_i(t)\Delta t - K_c \tau_D \frac{PV_{n_t} - PV_{n_t-1}}{\Delta t}$$

The same tuning correlations are used for both the continuous and discrete forms of the PID controller.

B.4.4 IMC Tuning Correlations

The most common tuning correlation for PID control is the IMC (Internal Model Control) rules. IMC is an extension of lambda tuning by accounting for time delay. The parameters Kc, τ_P, and θ_P are obtained by fitting dynamic input and output data to a first-order plus dead-time (FOPDT) model.

- Kp = Process gain
- τ_P = Process time constant
- θ_P = Process dead-time

$$\text{Aggressive Tuning: } \tau_c = max(0.1\tau_p, 0.8\theta_p)$$

$$\text{Moderate Tuning: } \tau_c = max(1.0\tau_p, 8.0\theta_p)$$

$$\text{Conservative Tuning: } \tau_c = max(10.0\tau_p, 80.0\theta_p)$$

$$K_c = \frac{1}{K_p}\frac{\tau_p + 0.5\theta_p}{(\tau_c + 0.5\theta_p)} \qquad \tau_I = \tau_p + 0.5\theta_p \qquad \tau_D = \frac{\tau_p\theta_p}{2\tau_p + \theta_p}$$

B.4.5 Optional Derivative Filter

The optional parameter α is a derivative filter constant. The filter reduces the effect of measurement noise on the derivative term which can lead to controller output amplification of the noise.

$$\alpha = \frac{\tau_c(\tau_p + 0.5\theta_p)}{\tau_p(\tau_c + \theta_p)}$$

The PID with the filter is augmented as:

$$u(t) = u_{bias} + K_c e(t) + \frac{K_c}{\tau_I}\int_0^t e(t)dt - K_c\tau_D\frac{d(PV)}{dt} - \alpha\tau_D\frac{du(t)}{dt}$$

B.4.6 Simple Tuning Rules

Note that with moderate tuning and negligible dead-time ($\theta_p \rightarrow 0$ and $\tau_c = 1.0\tau_p$), IMC reduces to simple tuning correlations that are easy to recall without a reference book.

$$K_c = \frac{1}{K_p} \qquad \tau_I = \tau_p \qquad \tau_D = 0 \qquad \text{Simple tuning correlations}$$

B.4.7 Anti-Reset Windup

An important feature of a controller with an integral term is to consider the case where the controller output $u(t)$ saturates at an upper or lower bound for an extended period. This causes the integral term to accumulate to a large summation that causes the controller to stay at the saturation limit until the integral summation is reduced. Anti-reset windup is when the integral term does not accumulate if the controller output is saturated at an upper or lower limit.

B.4.8 Derivative kick

Although the "derivative" term implies $\frac{de(t)}{dt}$, the derivative of the process variable $\frac{d(PV)}{dt}$is used in practice to avoid phenomena termed "derivative kick". Derivative kick occurs because the value of the error changes suddenly whenever the set point is adjusted. The derivative of a sudden jump in the error causes the derivative of the error to be instantaneously large and causes the controller output to saturate for one cycle at either an upper or lower bound. While this momentary jump isn't typically a problem for most systems, a sudden saturation of the controller output can put undue stress on the final control element or potentially disturb the process.

To overcome derivative kick, it is assumed that the set point is constant with $\frac{dSP}{dt} = 0$.

$$\frac{de(t)}{dt} = \frac{d(SP - PV}{dt} = \frac{d(SP)}{dt} - \frac{d(PV)}{dt} = -\frac{(PV)}{dt}$$

This modification avoids derivative kick but keeps a derivative term in the PID equation.

Me

C

Localization Scripts

D

D.1 Forward Kinematics – Dead Reckoning

```python
#!/usr/bin/env python3
from dead_reckoning import ForwardKinematics, Mecanum
import rospy
import dead_reckoning
import numpy as np
from nav_msgs.msg import Odometry
from std_msgs.msg import Float32MultiArray, Float32
from math import pi
import math
from geometry_msgs.msg import Point, Pose, Quaternion, Twist,
    Vector3
import tf

rate_value = 200
lx = 0.15 #0.22
ly = 0.36 #0.158
wheel_radius = 0.044444444444444444 #0.075
speed_set = False


def RPM_to_rad_per_sec(speeds):
    return 2 *pi * speeds /60

def speeds_of_wheels_callback(data: Float32MultiArray):
    global speeds_of_wheels,speed_set
    speeds_of_wheels = np.array(data.data)
    speeds_of_wheels = RPM_to_rad_per_sec(speeds_of_wheels)
    speed_set = True

def theta_callback (data:Float32):
    global theta
    theta =  math.radians(data.data)



if __name__ == "__main__" :
```

```
    speeds_of_wheels = []
    theta = 0

    mecanum = Mecanum(lx , ly , wheel_radius)
    odom_broadcaster = tf.TransformBroadcaster()

    rospy.init_node("dead_reckoning")
    rospy.Subscriber("Espeeds", Float32MultiArray,
       speeds_of_wheels_callback)
    rospy.Subscriber("Mpu", Float32, theta_callback)
    odom_pub = rospy.Publisher('odom', Odometry, queue_size=10)
    current_time = rospy.Time.now()
    last_time = rospy.Time.now()
    rate = rospy.Rate(rate_value)

    while not rospy.is_shutdown():
        if not speed_set:
            continue
        odom_quat = tf.transformations.quaternion_from_euler(0, 0,
           theta)
        base_link_speeds = mecanum.calc_speed(speeds_of_wheels)
        current_time = rospy.Time.now()
        dt = (current_time - last_time).to_sec()
        integrated_distances,odom_speeds  =
           mecanum.calc_distance(dt, base_link_speeds,theta)

        x, y = integrated_distances[0], integrated_distances[1]
        vx, vy, vth = odom_speeds[0],odom_speeds[1],odom_speeds[2]


        odom_broadcaster.sendTransform(
        (integrated_distances[0], integrated_distances[1], 0),
        odom_quat,
        current_time,
        "base_link",
        "odom"
    )


        # next, we'll publish the odometry message over ROS
        odom = Odometry()
        odom.header.stamp = current_time
        odom.header.frame_id = "odom"

        # set the position
```

```
78          odom.pose.pose = Pose(Point(x, y, 0.),
                Quaternion(*odom_quat))
79
80          # set the velocity
81          odom.child_frame_id = "base_link"
82          odom.twist.twist = Twist(Vector3(vx, vy, 0), Vector3(0, 0,
                vth))
83
84          # publish the message
85          odom_pub.publish(odom)
86
87          last_time = current_time
88
89
90          rate.sleep()
```

Code Snippet D.1: Dead reckoning script

D.2 Inverse Kinematics

```
1  def inverse_kinematics(vx, vy, w, wheel_radius, wheel_base):
2  """
3
4  Compute the wheel speeds for a differential drive robot.
5
6
7  Parameters:
8  vx (float): Linear velocity in the x direction.
9  vy (float): Linear velocity in the y direction (should be zero for
       differential drive).
10 w (float): Angular velocity.
11 wheel_radius (float): Radius of the wheels.
12 wheel_base (float): Distance between the wheels.
13
14 Returns:
15 (float, float): Speed of the left and right wheels.
16 """
17 v_left = (vx - (w * wheel_base / 2)) / wheel_radius
18 v_right = (vx + (w * wheel_base / 2)) / wheel_radius
19 return v_left, v_right
```

Code Snippet D.2: Inverse Kinematics Function

D.3 Extended Kalman Filter

```
<launch>
<arg name="odom_frame" default="odom" />
<arg name="base_link_frame" default="base_link" />
<arg name="world_frame" default="odom" />

  <node pkg="robot_localization" type="ekf_localization_node"
     name="ekf_localization_node" output="screen">
    <param name="frequency" value="30" />
    <param name="sensor_timeout" value="0.1" />
    <param name="two_d_mode" value="true" />
    <param name="transform_time_offset" value="0.1" />
    <param name="transform_timeout" value="0.0" />
    <param name="print_diagnostics" value="true" />
    <param name="debug" value="false" />
    <rosparam command="load" file="$(find
       your_package_name)/config/ekf.yaml" />
    <param name="odom_frame" value="$(arg odom_frame)" />
    <param name="base_link_frame" value="$(arg base_link_frame)" />
    <param name="world_frame" value="$(arg world_frame)" />
    <remap from="odometry/filtered" to="odometry/ekf" />
  </node>
</launch>
```

Code Snippet D.3: EKF launch file

This template's footer/header shenanigans was based on *Clean Thesis* developed by Ricardo Langner. Additionally, the title page is inspired by Daan Zwaneveld's tudelft-report-thesis-template. I also added my own spice to this wicked mix.

Download this template at `http://dudetrustme.org`.