Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN2160 - Electronic Design Realization

Final Project Report- Design Documentation

Obstacle Avoidance System for Warehouse AMR and AGV

Group H

210212N - Herath B.H.M.K.S.B.
210325M - Kuruppu M.P.
210349N - Madhushan I.D.
210454G - Peiris D.L.C.J.

Contents

Chapter 1

Obstacle Avoidance System for Warehouse AMR and AGV

1.1 Abstract

In the rapidly advancing field of warehouse management, the integration of sophisticated technologies such as Autonomous Mobile Robots (AMRs) and Autonomous Guided Vehicles (AGVs) has become essential for enhancing efficiency, safety, and productivity. This report provides a detailed examination of the development and implementation of obstacle avoidance systems for AMRs and AGVs, highlighting significant achievements. This document also includes comprehensive design details, daily log entries documenting ongoing development efforts, and a roadmap for future improvements, emphasizing the importance of continuous innovation to achieve fully autonomous and reliable warehouse operations.

1.2 Introduction

In the evolving landscape of warehouse management, the adoption of advanced technologies has become imperative to meet the increasing demands for efficiency, safety, and productivity. Autonomous Mobile Robots (AMRs) and Automated Guided Vehicles (AGVs) have emerged as pivotal solutions in automating and optimizing material handling processes. A critical aspect of these technologies is their ability to detect and avoid obstacles in dynamic and often unpredictable warehouse environments. This capability ensures smooth and safe operations, significantly reducing the risk of accidents and enhancing overall efficiency.

This design documentation delves into the comprehensive development and implementation of obstacle avoidance systems for AMRs and AGVs. It provides a detailed analysis of the system architecture, including sensor integration, algorithmic strategies for obstacle detection and path planning, and communication protocols for coordinated operations. Furthermore, it documents daily log entries to track the progress and challenges encountered during the development process.

The purpose of this document is to offer a thorough understanding of the current state of obstacle avoidance systems, highlight the achievements and solutions implemented, and outline a strategic roadmap for future advancements. By addressing the existing challenges and proposing enhancements, this report aims to contribute to the ongoing efforts to improve the autonomy and reliability of AMRs and AGVs in warehouse settings.

Chapter 2

Circuit Design Details

Our project features two main circuits: one with a microcontroller and the other with a motor controller. The main circuit, which includes the microcontroller, is designed around the ATmega328P-AU, an 8-bit microcontroller.

2.0.1 Main Circuit:

Control and Navigation

- Motor Control: The ATmega328P-AU controls the motors that drive the wheels of the AGV/AMR. This involves handling speed, direction, and acceleration by generating PWM (Pulse Width Modulation) signals.
- Sensor Integration: The ATmega328P-AU interfaces with various sensors to gather environmental data crucial for navigation and obstacle avoidance.
- Navigation Algorithms: Simple navigation algorithms are implemented on the ATmega328P-AU to determine the vehicle's path based on sensor inputs.

Obstacle Avoidance

- Sensor Data Processing: The ATmega328P-AU processes data from sensors to detect obstacles and make real-time decisions to avoid collisions.
- Decision Making:It implements basic decision-making algorithms to change the path or stop the vehicle when an obstacle is detected.

Status Indication

- LED Indicators: LEDs controlled by the ATmega328P-AU provide visual feedback on the status of the vehicle, such as connectivity and error states.

2.0.2 Motor Driver Circuit

- Motor Driver ICs: We use the BTS7960B motor driver ICs, which are half H-bridge drivers capable of handling 40A. Since each IC is a half H-bridge, we use four BTS7960B ICs, with two dedicated to each side (left and right) of the vehicle.
- Signal Isolation: Buffers and line drivers, specifically the 74HC244BQ-Q100/SOT764/DHVQFN20, are used to isolate the PWM and enable inputs for the motor driver ICs.
- Current Measurement: The INA226 IC is utilized to measure the current through each motor, providing feedback for monitoring and control purposes.

- Distance and Velocity Feedback: Encoders are employed to provide feedback on the distance traveled and the velocity of the vehicle.
- Power Supply:
- Voltage Regulation: A 12V power supply is regulated down to 5V using the L78M05 voltage regulator.
- Polarity Protection: The Infineon SPD50P03LG MOSFET is used for reverse polarity protection to safeguard the power supply circuitry.

2.0.3 Summary

The control and navigation circuit, based on the ATmega328P-AU microcontroller, handles motor control, sensor integration, navigation algorithms, obstacle avoidance, and status indication. The motor driver circuit, using BTS7960B ICs, manages the power and control of the motors, with additional components for signal isolation, current measurement, distance and velocity feedback, voltage regulation, and polarity protection.

2.1 Bill of Materials

Table 2.1: Bill of Materials of Main PCB

Designator	Description	Data Sheet
Y1	ABM11-16.000MHz-12-N, 16.000MHz crystal oscillator	https://www.mouser.in/datasheet/2/3/ABM11-1775087.pdf
C7, C8	C1206C220JGGACTU , 22 pF capacitor	https://www.mouser.in/datasheet/2/447/KEM_C1009_C0G_HV_SMD-3316207.pdf
C4, C5	KAM31BR81H103KT, 10 nF Capacitor	https://www.mouser.in/datasheet/2/40/AutoMLCCKAM-3216307.pdf
U1	L78M05ACDT-TR, Linear Voltage Regulators 5.0V 0.5A Positive	https://www.mouser.in/datasheet/2/389/l78m-1849664.pdf
IC1	ATMEGA328P-AU, 8-bit Microcontrollers - MCU 32KB In-system Flash 20MHz	https://www.mouser.in/datasheet/2/268/Atmel_7810_Automotive_Microcontrollers_ATmega328P_-3445629.pdf
J2, J3, J9, J10, J11	B2B-XH-A(LF)(SN) Connector Header Through Hole 2 position 0.098" (2.50mm)	https://www.snapeda.com/parts/B2B-XH-A(LF)(SN)/JST%20Sales%20America%20Inc./datasheet/
J4, J5, J13	B3B-XH-A(LF)(SN), Connector Header Through Hole 3 position 0.098" (2.50mm)	https://www.jst-mfg.com/product/pdf/eng/eXH.pdf
J6, J7, J8	B4B-XH-A(LF)(SN), Connector Header Through Hole 4 position 0.098" (2.50mm)	https://www.jst-mfg.com/product/pdf/eng/eXH.pdf
S1	TS-1187A-B-A-B, Switch 50mA 5.1mm 100,000 Times 160gf 12V 5.1mm	https://www.lcsc.com/datasheet/lcsc_datasheet_2304140030_XKB-Connection-TS-1187A-B-A-B_C318884.pdf
R1, R2	RC1206-R-SBE24L, 10k resistor 500 mW (1/2 W), 250 mW (1/4 W)	https://www.mouser.in/datasheet/2/447/PYu_RC_Group_51_RoHS_L_11-1947909.pdf
C1, C3, C6	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50V 0.1uF X7R 1206 10%	https://www.mouser.com/datasheet/2/396/Taiyo_Yuden_1102023_MC_mlcc_all_e-3081584.pdf
C2	Multilayer Ceramic Capacitors MLCC - SMD/SMT 100V 0.33uF X7R 1206 10% AEC-Q200	https://www.mouser.com/datasheet/2/447/KEM_C1090_X7R_ESD-3316292.pdf
L1	Standard LEDs - SMD WL-SMCW SMDMono TpVw Waterclr 0805 Blue	https://www.we-online.com/components/products/datasheet/150080BS75000.pdf

Table 2.2: Bill of Materials for PWM Enable Control

Designator	Description	Data Sheet
IC5	Logic Gates 74HC00BQ/SOT762/DHVQFN14	https://www.mouser.com/datasheet/2/916/74HC_HCT00-2937322.pdf
R24,R25	Thin Film Resistors - SMD 1206 100Kohm 0.1% 25ppm	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
J4	B4B-XH-A(LF)(SN) Connector Header Through Hole 4 position 0.098" (2.50mm)	https://www.snapeda.com/parts/B4B-XH-A(LF)(SN)/JST%20Sales/datasheet/

Table 2.3: Bill of Materials for Power Supply

Designator	Description	Data Sheet
Q5	MOSFET P-Ch -30V 50A DPAK-4 OptiMOS P	https://www.mouser.com/datasheet/2/196/Infineon_SPD50P03LG_DS_v01_09_en-1732178.pdf
J1	3 Position Wire to Board Terminal Block Horizontal with Board 0.197" (5.00mm) Through Hole	https://www.we-online.com/components/products/datasheet/691137710003.pdf
R22	Thin Film Resistors - SMD 1206 10Kohm 0.1% 25ppm	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
R23	Thin Film Resistors - SMD 1206 100Kohm 0.1% 25ppm	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
L5	Standard LEDs - SMD WL-SMCW SMDMono TpVw Waterclr 0805 Red	https://www.we-online.com/components/products/datasheet/150080RS75000.pdf
D1	Zener Diodes ZENER SOD123W/CFP3	https://www.mouser.com/datasheet/2/916/HPZR_Q_SER-3045048.pdf
C7	Aluminum Electrolytic Capacitors - Radial Leaded 6.3VDC 470uf SU BI-POL	https://www.mouser.com/datasheet/2/315/ABA0000C1053-947519.pdf
C6	Multilayer Ceramic Capacitors MLCC - SMD/SMT 100V 0.47uF X7R 1206 20%	https://www.mouser.com/datasheet/2/447/KEM_C1090_X7R_ESD-3316292.pdf
IC3	Linear Voltage Regulators 5.0V 0.5A Positive	https://www.mouser.com/datasheet/2/389/l78m-1849664.pdf
C5	Multilayer Ceramic Capacitors MLCC - SMD/SMT 100V 0.33uF X7R 1206 10% AEC-Q200	https://www.mouser.com/datasheet/2/447/KEM_C1090_X7R_ESD-3316292.pdf
C4	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50V 0.1uF X7R 1206 10%	https://www.mouser.com/datasheet/2/396/Taiyo_Yuden_1102023_MC_mlcc_all_e-3081584.pdf

Table 2.4: Bill of Materials for input part and input indicators

Designator	Description	Data Sheet
R10, R11, R12, R13	Thin Film Resistors - SMD 1206 1.0Kohm 0.1% 25ppm	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
R14, R15, R16, R17	Thin Film Resistors - SMD 1206 10Kohm 0.1% 25ppm	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
L1, L2, L3, L4	Standard LEDs - SMD WL-SMCW SMDMono TpVw Waterclr 0805 Blue	https://www.we-online.com/components/products/datasheet/150080BS75000.pdf
Q1, Q2, Q3, Q4	Bipolar Transistors - BJT SUPER FAST RECOVERY DIODE	https://fscdn.rohm.com/en/products/databook/datasheet/discrete/transistor/bipolar/bc847chzgt116-e.pdf

Table 2.5: Bill of Materials for H bridge Motor Driver

Designator	Description	Data Sheet
IC2	IC HALF BRIDGE DRVR 40A TO263-7	https://drive.google.com/file/d/1TOawd-2K2jNzJu8-zffXoSbYFfjPE7yD/view?usp=sharing
R6, R7	Thin Film Resistors - SMD 1206 10Kohm 0.1% 25ppm	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
R9	Thin Film Resistors - SMD 1206 1.0Kohm 0.1% 25ppm	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
R8	Thin Film Resistors - SMD 5.1K OHM .1% 10PPM 1/4W	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
C2	Multilayer Ceramic Capacitors MLCC - SMD/SMT 1kV 100pF C0G 1206 %	https://www.mouser.com/datasheet/2/447/KEM_C1009_C0G_HV_SMD-3316207.pdf
J2, J3	2 Position Wire to Board Terminal Block Horizontal with Board 0.197" (5.00mm) Through Hole	https://www.we-online.com/components/products/datasheet/691137710002.pdf
J6	B2B-XH-A(LF)(SN) Connector Header Through Hole 2 position 0.098" (2.50mm)	https://www.snapeda.com/parts/B2B-XH-A(LF)(SN)/JST%20Sales%20America%20Inc./datasheet/

Table 2.6: Bill of Materials for Line Driver

Designator	Description	Data Sheet
IC3	Buffers and Line Drivers 74HC244BQ-Q100/SOT764/DHVQFN20	https://www.mouser.com/datasheet/2/916/74HC_HCT244_Q100-2937283.pdf
R18, R19, R20, R21	Thin Film Resistors - SMD 5.1K OHM .1% 10PPM 1/4W	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf
C3	Multilayer Ceramic Capacitors MLCC - SMD/SMT 100V 1uF X7R 1206 10% Flex AEC-Q200	https://www.mouser.com/datasheet/2/447/KEM_C1078_X7R_FT_CAP_AUTO_SMD-3316638.pdf

Table 2.7: Bill of Materials for Current Sensor

Designator	Description	Data Sheet
IC1, IC6	Current & Power Monitors & Regulators Hi-Side Msmt,Bi-Dir Current/Pwr Mon	https://drive.google.com/file/d/1nxFOuBpqzkl-vuOSdjNEv78ri7r-sjzS/view?usp=sharing
J5	B4B-XH-A(LF)(SN) Connector Header Through Hole 4 position 0.098" (2.50mm)	https://www.snapeda.com/parts/B4B-XH-A(LF)(SN)/JST%20Sales/datasheet/
C1, C8	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50V 0.1uF X7R 1206 10 %	https://www.mouser.com/datasheet/2/396/Taiyo_Yuden_1102023_MC_mlcc_all_e-3081584.pdf
R4, R28	Current Sense Resistors - SMD .015ohms 1% 1w	https://www.vishay.com/docs/30402/wfc.pdf
R1, R2, R3, R5, R26, R27, R29	Thin Film Resistors - SMD 1206 10Kohm 0.1% 25ppm	https://www.mouser.com/datasheet/2/315/AOA0000C307-1149632.pdf

2.1.1 Power Calculations

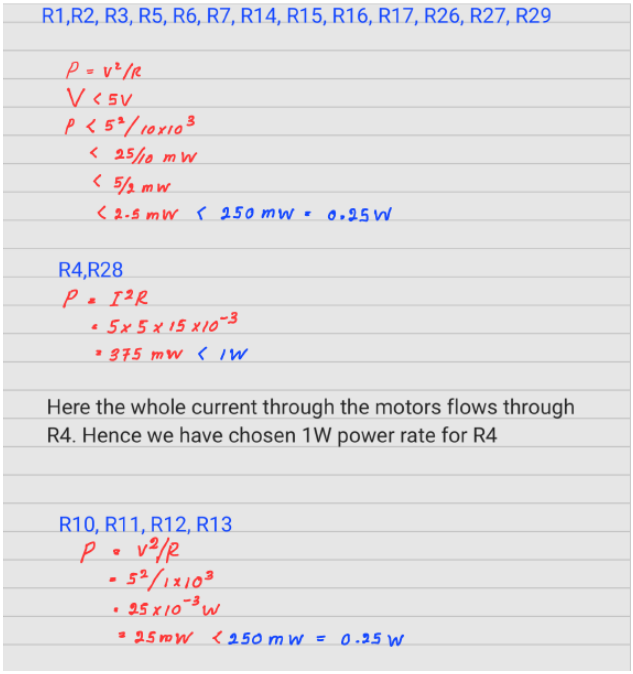


Figure 2.1: Power Calculations 1

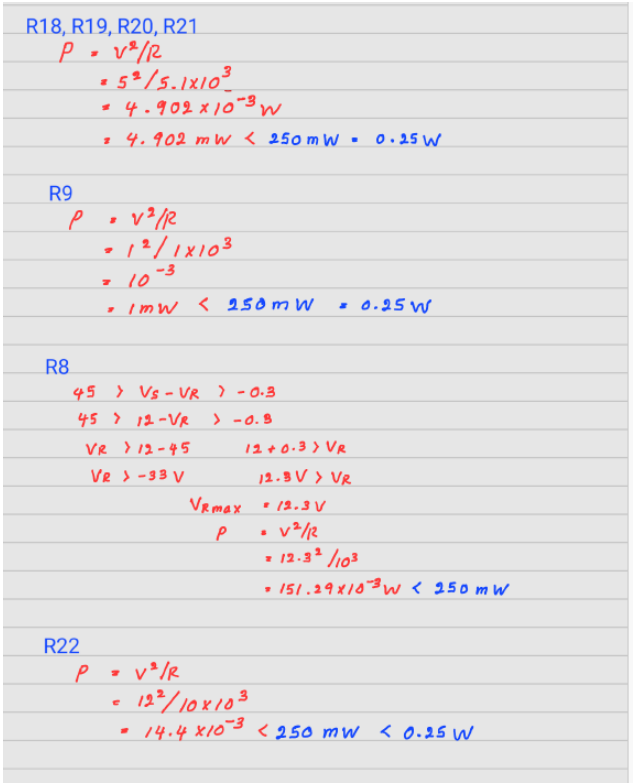


Figure 2.2: Power Calculations 2

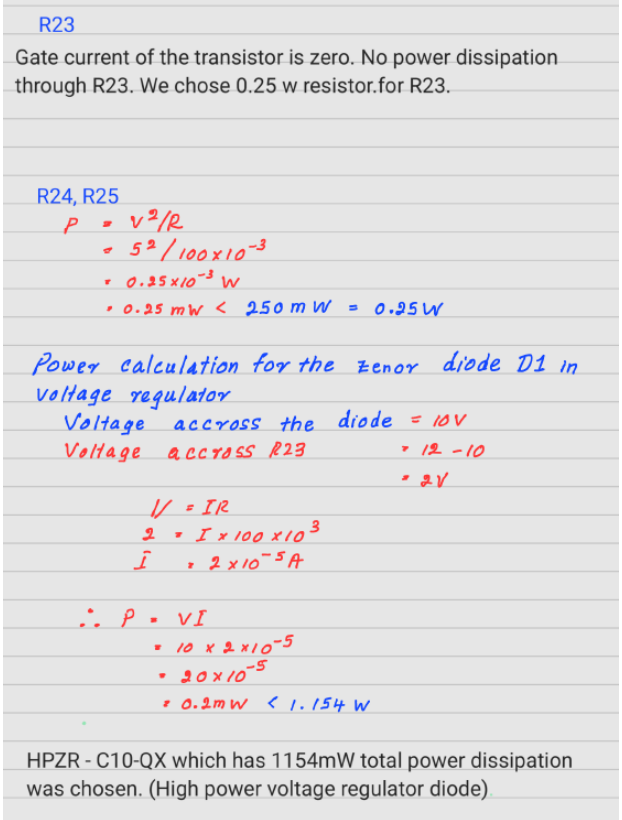


Figure 2.3: Power Calculations 3

2.2 PCB Designing

For our project, we designed two PCBs. The first is the main PCB, which houses the microcontroller and its associated components. The second PCB is dedicated to the motor controller circuit. Both PCBs were designed using Altium 23.7.1, a powerful and versatile PCB design software.

The Printed Circuit Board (PCB) is designed to ensure it works well and lasts long under different conditions. Here are the detailed specifications of the PCB:

Material and Layers

- **Base Material:** FR4
  - *Reasoning:* FR4 is a commonly used and reliable material for PCBs. It offers good mechanical strength and electrical insulation.
- **Number of Layers:** 2-layer PCB
  - *Reasoning:* A double-layer PCB provides a good balance between complexity and cost, allowing for adequate routing of signals and power distribution.

Trace Widths and Thickness

- **Input and GND Traces:**
  - **Width:** 0.381 mm (15 mils) for main PCB 1.00 mm (39.37 mil) for motor controller PCB.
  - **Thickness:** 1 oz (35 µm)

  - **Current Capacity:** Can handle up to 1.2 A
  - *Reasoning:* These traces are designed to handle slightly higher current requirements, ensuring a stable and low-resistance path for the main power inputs and ground connections. The width and thickness are chosen to minimize voltage drop and heat generation.

- **Signal Traces:**
  - **Minimum Width:** 0.254 mm (10 mils) for main PCB & 0.305 mm (12 mils) for Motor Controller PCB
  - **Thickness:** 1 oz (35 µm)
  - *Reasoning:* Signal traces carry lower currents, so a smaller width is enough. This maintains proper signal integrity and reduces electromagnetic interference (EMI).

Clearance

- **Clearance:** 0.254 mm (10 mils)
  - *Reasoning:* Adequate clearance is critical to prevent electrical shorts and ensure reliable operation. This spacing adheres to standard IPC-2221 guidelines for internal conductors.

Hole Sizes

- **Mounting Holes:**
  - **Size:** 3.1 mm
  - **Plating:** Plated through holes (PTH)
  - *Reasoning:* The holes are designed to accommodate 3 mm diameter screws, ensuring a secure attachment of the PCB to its enclosure. The slightly larger hole size allows for easy installation and some tolerance for misalignment.

Solder Mask and Silkscreen

- **Solder Mask:** Green, standard finish
  - *Reasoning:* The solder mask provides insulation between solder pads and prevents solder bridges during the assembly process, enhancing the overall durability of the PCB.
- **Silkscreen:** Clear component labels and orientation markers
  - *Reasoning:* The silkscreen layer provides clear component labels and orientation markers, which are essential for assembly, troubleshooting, and maintenance.

2.2.1 PCB Manufacturing

The PCB was designed using Altium 23.7.1, a versatile PCB design software. Both PCBs were then manufactured by JLC PCB, known for their high-quality and reliable production services. This ensured that the PCBs met our exact specifications and quality standards, providing a solid foundation for our project.
Go to Website : `https://jlcpcb.com/`

The manufacturing process of the PCB includes several critical steps to ensure the board meets the specified requirements. This involves designing the layout, fabricating the board, assembling components, and performing quality checks.

PCB Specification

- **Number of layers**: 2
- **Board thickness**: 1.6 mm
- **Copper weight**: 1 oz
- **Material Type**: FR4- Standard
- **Surface finish**: HASL(Hot Air Solder Leveling)
- **Solder mask color**: Green

2.3 Soldering

The soldering process for our Surface-Mount Device (SMD) printed circuit board (PCB) was executed to ensure optimal quality and reliability. Initially, we prepared the PCB by thoroughly cleaning its surface to remove any contaminants that could interfere with the soldering process. We then utilized a stencil to apply solder paste accurately to the designated pads on the PCB. The stencil, made of fine metal with precisely cut apertures corresponding to the pad locations, was aligned carefully over the PCB.

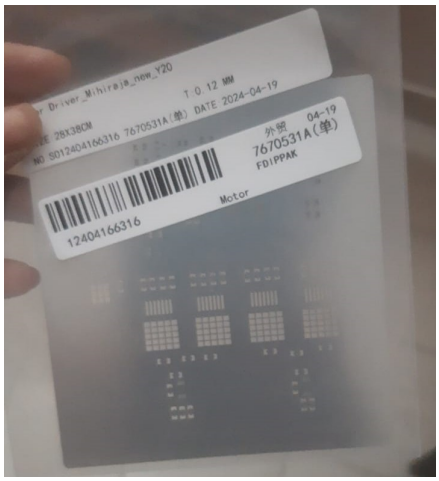


Figure 2.4: Stencil of the motor driver PCB

Using a squeegee, we spread the solder paste evenly across the stencil, ensuring consistent application to all pads. After removing the stencil, we positioned the SMD components onto the PCB, aligning them with the solder paste-covered pads. The PCB, with components in place, was then subjected to a reflow soldering process. We brought the Nozzle of Hot Air Gun close to the Circuit board and wait until the Solder paste melts and fuses with the IC Pins. This cycle melted the solder paste, creating strong electrical and mechanical connections between the components and the PCB. Finally, the soldered PCB was inspected to ensure all connections were properly formed and free of defects, guaranteeing the functionality and durability of the assembled board.

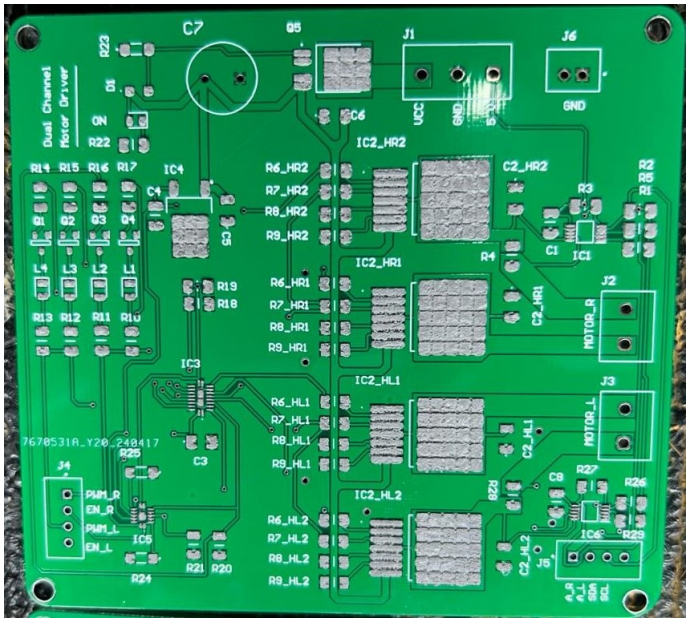


Figure 2.5: PCB with Soldering Paste

2.4 Testing

Although testing is not complete, we have incorporated features to help detect errors easily. We added a red LED to indicate when the motor driver is powered on. Additionally, we included a circuit to show the status of the Enable inputs. Specifically, an LED labeled L4 lights up when the Left Enable is active, and another LED labeled L3 lights up when the Left Enable is off. For the right side, L2 turns on when the Right Enable is active, and L1 lights up when the Right Enable is off. These indicators make it easy to identify the status of the motor driver and its inputs.

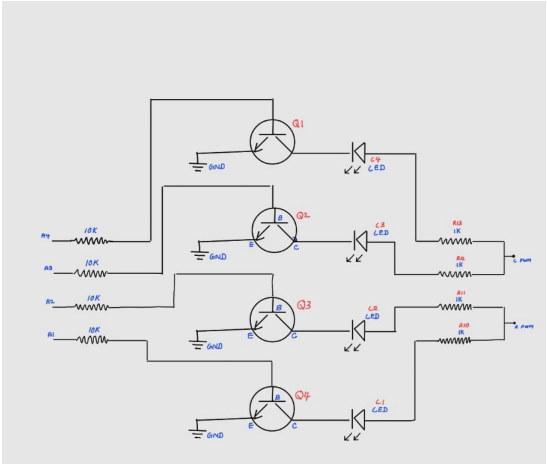


Figure 2.6: Input Indicator Circuit

2.5 Assembly

Not Yet Done

Chapter 3

Enclosure design Details

Our enclosure design comprises two main sections: the top and bottom parts. The top section is designed with three openings to accommodate ultrasonic sensors, to detect objects in the robot's environment. The bottom part consits of two PCBs which are main PCB, the motor driver PCB, and the battery. These are some key design aspects that we have considered when designing our enclosure.

PCB placement

There are two PCBs in our design. So placing those two PCBs appropriately within the enclosure is very important. Motor controller PCB is placed to minimize wire lengths, particularly critical for connections to the motors, where higher currents are drawn. This arrangement not only reduces potential electrical interference, but it also enhances the overall efficiency of the robot's motor control system. The orientation of the two PCBs is also crucial. Both the main PCB and motor controller PCB are oriented within the enclosure to minimize the length of connections and ensure integration with external inputs and outputs. Also, we have taken space considerations into account when orienting the PCBs. Also, we used mounting bosses to Mount the PCBs. It helps in dissipating heat generated during operation and helps maintain electrical isolation to prevent interference and ensure reliable performance in different operating conditions.

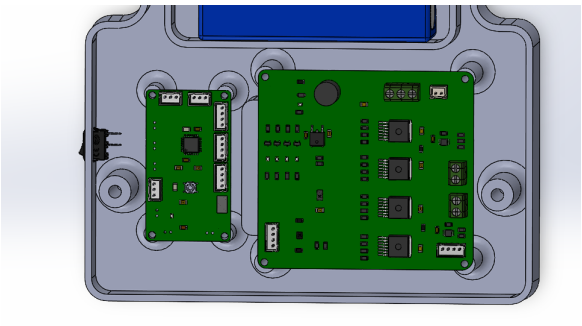


Figure 3.1: PCB palcement

Wheel placement

Wheel placement in the enclosure is also a main concern when designing the enclosure. We are using two main wheels and one caster wheel in our design. The stability of our design mainly relies on how we place those wheels in the enclosure. We have placed the main wheels at the rear end of the enclosure. And we have allocated space in the front of our design to place the caster wheel. We have placed caster wheels in such a way that the it levels perfectly with two main wheels. For that we made a special arrangement in our enclosure.

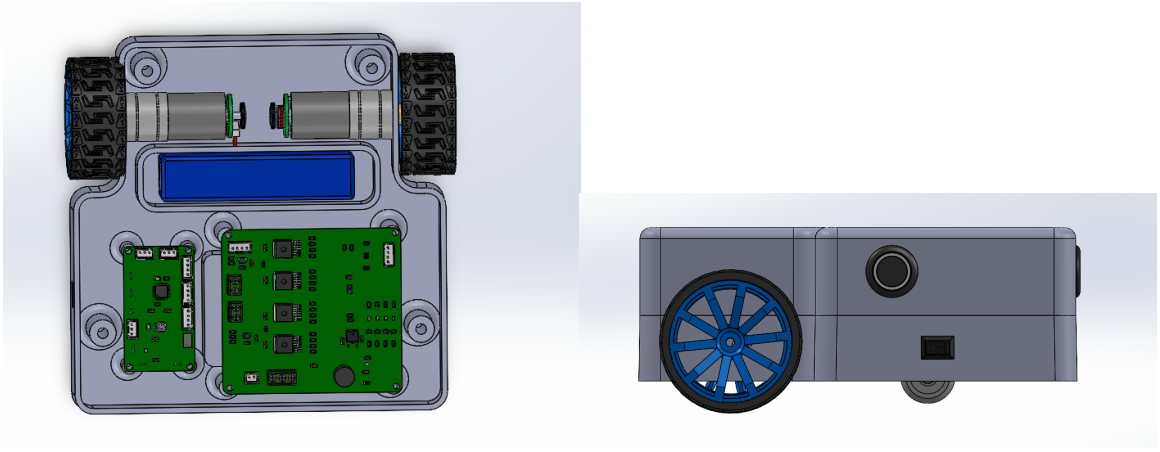


Figure 3.2: Wheel placement

Stability of the design

The stability of autonomous mobile robots (AMRs) and automated guided vehicles (AGVs) is a major concern in our design. To Achieve optimal stability we placed the robot's center of gravity as low as possible to minimize its gravitational potential energy during movement. Since the battery of the robot contributes a significant portion of the robot's total weight, we have given special attention to how to place the battery to balance torque effectively with other components, to enhance the robot's overall stability. To achieve that we have placed the battery horizontally and near to the center of our design.

Compactness of the design

Achieving maximum compactness was a main concern while designing the enclosure. The orientation of the battery within the bottom section of the enclosure is carefully planned to optimize spatial efficiency. This not only conserves valuable space but also contributes to the overall compactness and the appearance of the robot. To secure the battery firmly in place and ensure it remains stable during operation, a frame has been incorporated into the enclosure's design. We also have placed the PCBs in such a way that it uses minimum space within our design. So we have designed the enclosure to place the two PCBs near each other to improve the compactness of the design. When placing the two PCBs near to each other we also have taken the other concerns when placing the two PCBs nearby.

Mouldability and Draft angle consideration

To ensure our design is suitable for the moulding process, we have prioritized mouldability as a key aspect. In addition to weight-bearing requirements, we have also focused on manufacturability to ensure efficient production. These are the key things we have done to enhance Mouldability:

Incorporation of Draft Angles: Draft angles are incorporated into the design of the enclosure components. These angles allow the parts to be easily ejected from the mould without damaging the product or the mould. By adding a slight taper (typically between 1 to 3 degrees) to the vertical faces of the design, we ensure smoother release from the mould, reducing cycle times and manufacturing costs.

Uniform Wall Thickness: We have ensured that the wall thickness is uniform throughout the design. This minimizes the risk of warping, reduces internal stresses, and improves the quality of the final product. Uniform wall thickness also enhances material flow during the molding process, resulting in consistent and defect-free parts.

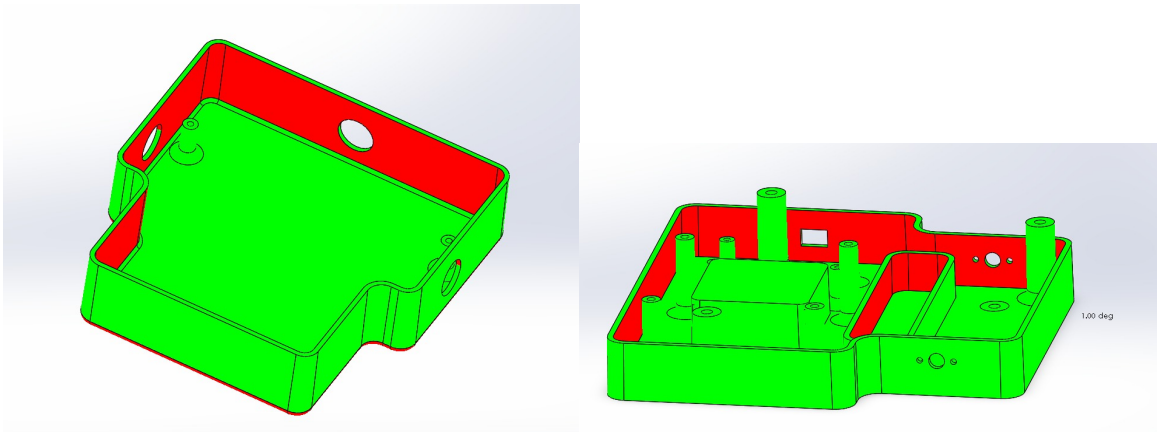


Figure 3.3: Draft analysis

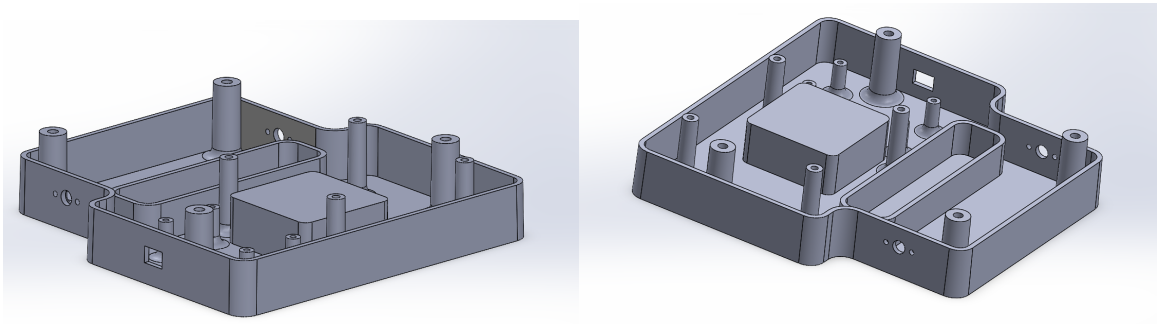


Figure 3.4: Fillets

Optimized Parting Lines: he design has been optimized to position parting lines in non-critical areas. This minimizes the visual and functional impact of any seams or flash that might occur during molding. Strategically placed parting lines facilitate easier mold fabrication and maintenance.

Rounded Corners and Edges: Sharp corners and edges have been replaced with rounded ones. This change improves material flow and reduces stress concentrations, enhancing the durability and appearance of the final product. Rounded corners also help in avoiding the accumulation of stress points during the ejection process.

Consideration of Undercuts: The design minimizes the use of undercuts to avoid complex mould designs and the need for additional tooling like side actions or lifters. Where undercuts are necessary, we have employed features like snap fits that can be moulded with slides or collapsible cores.

Material Selection: We have chosen materials that are compatible with the moulding process and have good flow properties. This choice ensures that the material fills the mould cavity completely and evenly, resulting in high-quality parts. By focusing on these aspects, we have significantly enhanced the moldability of our design, ensuring a smoother and more efficient manufacturing process.

Changes made to the Enclosure Design

Our new enclosure design is made to address the limitations of our previous design, particularly the previous design was not able to carry heavier weights because it is made from plastic. So by taking that into consideration we have designed a new enclosure from aluminum material to carry higher weights. Here are some of the enhancements and features of our new enclosure design:

1. Higher Structural Stability with Aluminum Extrusion Bars
The structure of the new enclosure is constructed using aluminum extrusion bars. Aluminum is chosen for its strength and lightweight. These extrusion bars form a rigid and stable framework, which significantly increases the enclosure's ability to support heavier loads compared to the previous design which was made from plastic.

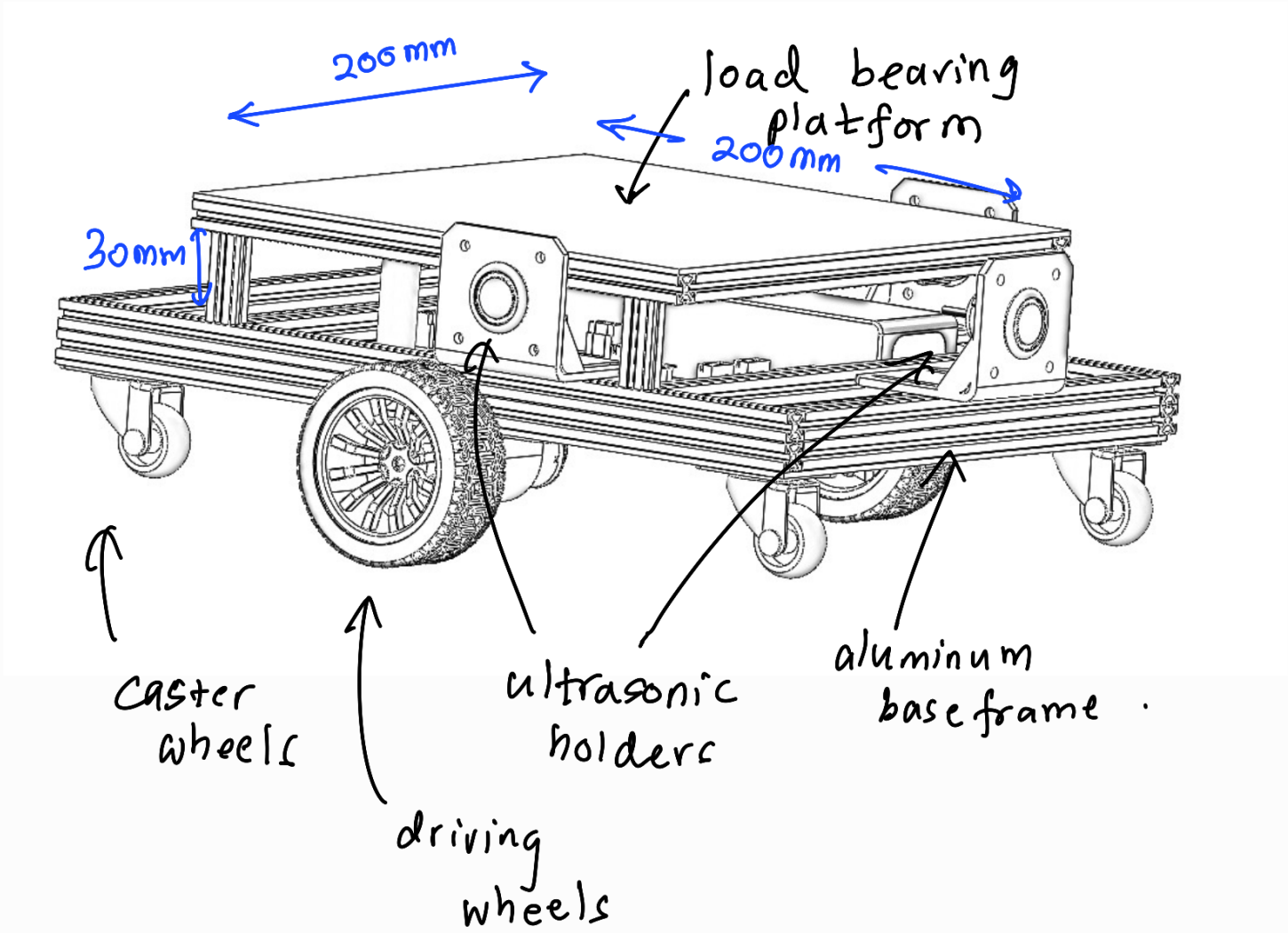
2. Aluminum Platform which is capable of carrying higher weight
The platform used to carry weights is made from a 3mm thick aluminum sheet. This thickness ensures that the platform is strong enough to support substantial loads without bending or warping. The platform's placement on top of the frame allows for an even distribution of weight, contributing to the overall stability of the enclosure.

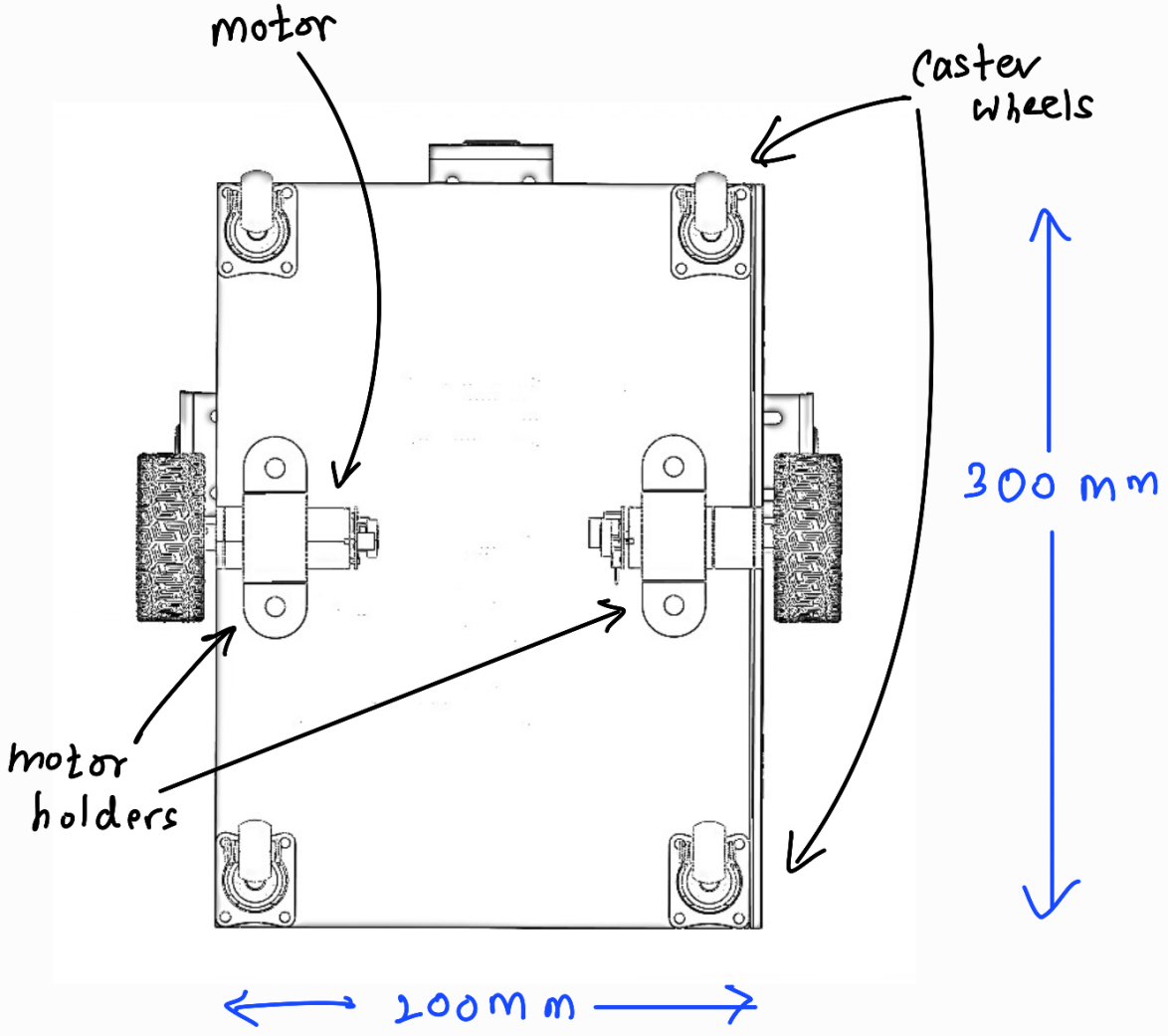
3. Aluminum Base platform to place Components
The base platform is constructed from 2mm thick aluminum sheets, it holds two PCBs (Main PCB and Motor Controller PCB) and the battery. Here we have chosen a 2mm thickness aluminum sheet to ensure the base platform avoid adding unnecessary weight. This base platform prevents the main components to make any movement or damage during operation.

4. Placement of Ultrasonic Sensors:
In our design, we have three ultrasonic sensors. We have placed the three ultrasonic sensors in such a way that it covers the maximum area as possible. So we have placed ultrasonic sensors in the front, left side, and right side of the enclosure. Here We have used Holed L bars to hold the ultrasonic sensors. These L bars are mounted on Aluminium extrusion bars. The placement and mounting of these sensors are done to ensure they function accurately and more reliably.

5. Improved Mobility and Stability with Wheel Configuration:
Our design has two driving wheels and four caster wheels to enhance both mobility and stability. In the prevous design we had the two driving wheels along with only one caster wheel. But in this design we have increased the number of caster wheels to 4 to increase the stability of the design. In this way our design is able to carry more weight than the previous design. Here the two main driving wheels are placed in the middle of the enclosure to proivde propulsion necessary to move the enclosure much more easier. These wheels are positioned centrally to provide better driving power and control. The four caster wheels, placed at the corners which improves maneuverability and stability of the design. These caster wheels enable the enclosure to turn smoothly and maintain balance, even on uneven surfaces.

6. Material Selection and Thickness Considerations:
The materials and their thicknesses are carefully selected to balance strength and weight. 3mm aluminum sheet for the load-bearing platform and 2mm sheets for the base frame which holds components are more than enough to carry the intended loads. By using aluminum extrusion bars and aluminum sheets, the new enclosure design provides a significant improvement in weight-bearing capacity and structural stability. The combination of a strong platform, and a strong base frame increases the overall stability of our design.


200 mm
load bearing platform
200 mm
30mm
caster wheels
driving wheels
ultrasonic holders
aluminum base frame


motor
caster wheels
300 mm
motor holders
200 mm

Screeshots of the New Assembled Solidworks Design

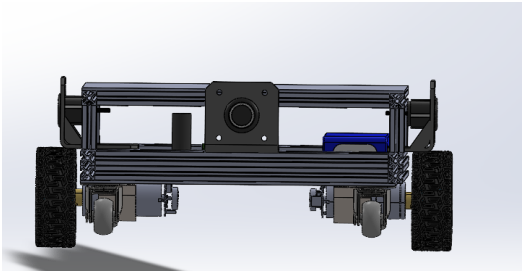


Figure 3.5: Enclosure Front View

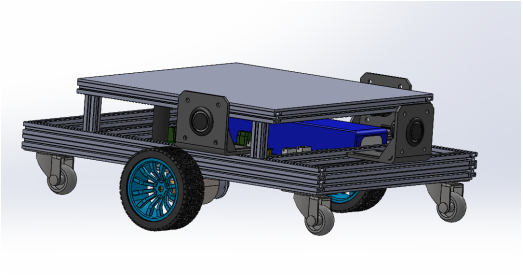


Figure 3.6: Enclosure Side View 1

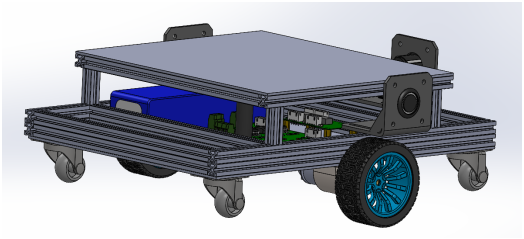


Figure 3.7: Enclosure Side View 2

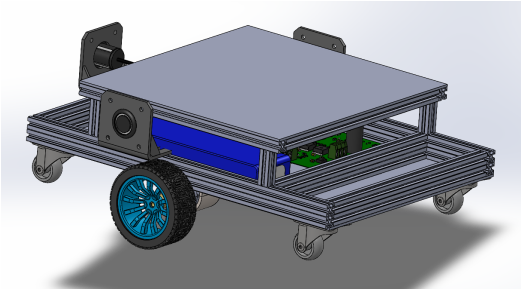


Figure 3.8: Enclosure Side View 3

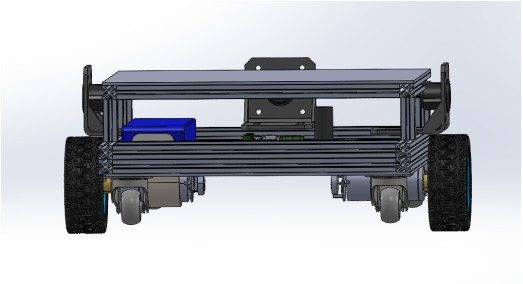


Figure 3.9: Rear View

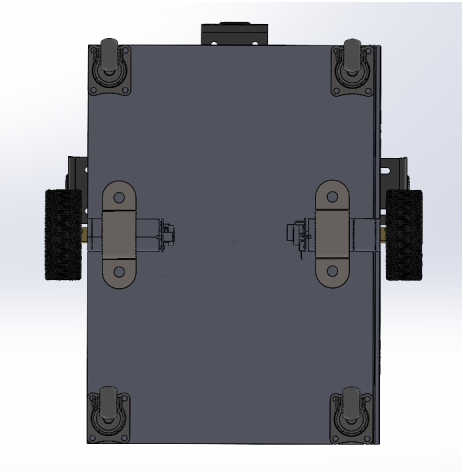


Figure 3.10: Bottom View

Chapter 4

AVR CODE

AVR (Advanced Virtual RISC) is a family of microcontrollers developed by Atmel (now Microchip Technology). AVR microcontrollers are widely used in embedded systems for their simplicity, efficiency, and versatility.

- **Microcontroller Basics:** AVR microcontrollers integrate a CPU, memory (RAM and Flash), various I/O peripherals (like timers, ADCs, UARTs), and often feature low power consumption, making them suitable for battery-operated devices.
- **Development Tools:** AVR development typically involves using tools like Atmel Studio (now Microchip Studio), AVR-GCC (GNU Compiler Collection for AVR), and AVRDUDE (AVR Downloader/UploaDEr) for programming and debugging.
- **Programming Language:** C/C++ is commonly used for AVR programming due to its efficiency and direct hardware access capabilities. Assembly language is also used for low-level optimizations.
- **Peripheral Interaction:** AVR microcontrollers interact with external devices through GPIO (General Purpose Input/Output) pins, USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), and more.
- **Interrupt Handling:** AVR microcontrollers support interrupts, allowing them to respond promptly to external events while performing other tasks.
- **PWM and Timers:** Pulse Width Modulation (PWM) is often used for controlling motors, LEDs, and other devices. AVR microcontrollers have built-in timers that facilitate PWM generation and precise timing operations.
- **Memory Management:** AVR microcontrollers have distinct memory areas for program storage (Flash memory), data storage (SRAM), and non-volatile storage (EEPROM), each with specific access methods and capacities.

In our robot project, we are using the ATMEL 328P-AU microcontroller, which is part of the AVR family. The decision to use AVR coding was driven by the microcontroller's capabilities and our familiarity with AVR development tools and programming languages like C/C++.

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>

// Define pins for ultrasonic sensors
#define TRIG_PIN_LEFT PB0
#define ECHO_PIN_LEFT PB1
#define TRIG_PIN_MIDDLE PB2
#define ECHO_PIN_MIDDLE PB3
#define TRIG_PIN_RIGHT PB4
#define ECHO_PIN_RIGHT PB5

// Define motor control pins
#define MOTOR_LEFT_PWM PD6
#define MOTOR_LEFT_EN PD7
#define MOTOR_RIGHT_PWM PD5
#define MOTOR_RIGHT_EN PD4


// Define encoder pins
#define ENCODER_LEFT_A PC0
#define ENCODER_LEFT_B PC1
#define ENCODER_RIGHT_A PC2
#define ENCODER_RIGHT_B PC3

// Define I2C addresses for current sensors
#define CURRENT_SENSOR_1_ADDR 0x40  // I2C address of current sensor 1
#define CURRENT_SENSOR_2_ADDR 0x41  // I2C address of current sensor 2

// Define constants
#define STOP_DISTANCE 15      // Distance in cm to stop the robot
#define DETOUR_DISTANCE 20    // Distance in cm to start detour
#define DESIRED_SPEED 0.5     // Desired speed (adjust as needed)

#define CPR 48  // Counts per revolution provided in the encoder's datasheet
#define WHEEL_DIAMETER 0.1  // Diameter in meters
#define TIME_INTERVAL 0.1  // Time interval in seconds
#define WHEELBASE 0.5 // Wheelbase

// Define acceleration and deceleration constants in m/s
#define ACCELERATION 0.5 // Example value
#define DECELERATION 0.5 // Example value

// PID controller constants
#define KP 1.0
#define KI 0.1
#define KD 0.01

// Global variables
volatile int32_t encoder_count_left = 0;
volatile int32_t encoder_count_right = 0;

// PID controller state
int16_t integral_left = 0;
int16_t integral_right = 0;
int16_t prev_error_left = 0;
int16_t prev_error_right = 0;

// Function prototypes
void init_ultrasonic();
void init_timer();
void init_motor_controller();
void init_encoders();
void init_pid();
void init_i2c();
void i2c_start();
```

```
69 void i2c_stop();
70 void i2c_write(uint8_t data);
71 uint8_t i2c_read_ack();
72 uint8_t i2c_read_nack();
73 uint16_t read_distance(uint8_t trig_pin, uint8_t echo_pin);
74 void stop_robot();
75 void move_forward();
76 void turn_left_90();
77 void turn_right_90();
78 void detour();
79 void set_motor_speed(uint8_t motor_pwm, uint8_t motor_en, uint8_t speed);
80 void accelerate(uint8_t motor_pwm, uint8_t motor_en, float target_speed, float
       initial_speed);
81 void decelerate(uint8_t motor_pwm, uint8_t motor_en, float target_speed);
82 int16_t read_encoder(uint8_t encoder_a, uint8_t encoder_b);
83 float update_heading();
84 int16_t compute_pid(int16_t setpoint, int16_t measured_value, int16_t *integral,
       int16_t *prev_error, uint16_t current_feedback);
85 uint16_t read_current_sensor(uint8_t sensor_addr);
86 float calculate_speed(int32_t encoder_counts, float time_interval);
87 float map_speed_to_pwm(float speed);
88
89
90
91 ISR(PCINT1_vect);
92
93 int main(void) {
94   // Initialize peripherals
95   init_ultrasonic();
96   init_timer();
97   init_motor_controller();
98   init_encoders();
99   init_i2c();
100  init_pid();
101
102  sei();  // Enable global interrupts
103
104  while (1) {
105    // Read distances from the ultrasonic sensors
106    uint16_t distance_left = read_distance(TRIG_PIN_LEFT, ECHO_PIN_LEFT);
107    uint16_t distance_middle = read_distance(TRIG_PIN_MIDDLE, ECHO_PIN_MIDDLE);
108    uint16_t distance_right = read_distance(TRIG_PIN_RIGHT, ECHO_PIN_RIGHT);
109
110    // Check if any distance is below the stop threshold
111    if (distance_middle < STOP_DISTANCE) {
112      stop_robot();
113      detour();
114      } else {
115      move_forward();
116    }
117
118    // Update heading
119    update_heading();
120
121    // PID control for motor speeds
122    int16_t left_encoder_counts = read_encoder(ENCODER_LEFT_A, ENCODER_LEFT_B);
123    int16_t right_encoder_counts = read_encoder(ENCODER_RIGHT_A, ENCODER_RIGHT_B);
124
125    // Convert encoder counts to speed in m/s
126    float speed_left = calculate_speed(left_encoder_counts, TIME_INTERVAL);
127    float speed_right = calculate_speed(right_encoder_counts, TIME_INTERVAL);
128
129    // Read current sensor feedback
130    uint16_t current_left = read_current_sensor(CURRENT_SENSOR_1_ADDR);
131    uint16_t current_right = read_current_sensor(CURRENT_SENSOR_2_ADDR);
132
133    // PID control for motor speeds
134        int16_t pid_output_left = compute_pid(DESIRED_SPEED, speed_left, &
```

```
    integral_left, &prev_error_left);
135         int16_t pid_output_right = compute_pid(DESIRED_SPEED, speed_right, &
    integral_right, &prev_error_right);
136 
137         // Map PID output to PWM duty cycle
138         uint8_t pwm_left = map_speed_to_pwm(pid_output_left);
139     uint8_t pwm_right = map_speed_to_pwm(pid_output_right);
140 
141     // Set motor speeds
142 
143     set_motor_speed(pwm_left, pwm_right);
144 
145     _delay_ms(100);  // Adjust delay as needed for PID control loop frequency
146   }
147 
148   return 0;
149 }
150 
151 
152 
153 // Initialize ultrasonic sensor pins
154 void init_ultrasonic() {
155   // Set trigger pins as output
156   DDRB |= (1 << TRIG_PIN_LEFT) | (1 << TRIG_PIN_MIDDLE) | (1 << TRIG_PIN_RIGHT);
157   // Set echo pins as input
158   DDRB &= ~((1 << ECHO_PIN_LEFT) | (1 << ECHO_PIN_MIDDLE) | (1 << ECHO_PIN_RIGHT));
159 }
160 
161 // Initialize timer for measuring pulse width
162 void init_timer() {
163   TCCR1B |= (1 << CS11);  // Set prescaler to 8
164 }
165 
166 // Initialize motor controller pins
167 void init_motor_controller() {
168 
169     // Set motor control pins as output
170     DDRD |= (1 << MOTOR_LEFT_EN) | (1 << MOTOR_RIGHT_EN);
171 
172     // Enable motors
173     PORTD |= (1 << MOTOR_LEFT_EN) | (1 << MOTOR_RIGHT_EN);
174 
175   // Set PWM mode for the motor control pins
176   TCCR0A |= (1 << WGM00) | (1 << WGM01);  // Fast PWM mode
177   TCCR0A |= (1 << COM0A1) | (1 << COM0B1);  // Clear OC0A/OC0B on Compare Match, set
      OC0A/OC0B at BOTTOM
178   TCCR0B |= (1 << CS00);  // No prescaling
179 }
180 
181 // Initialize encoder pins and interrupts
182 void init_encoders() {
183   // Set encoder pins as input
184   DDRC &= ~((1 << ENCODER_LEFT_A) | (1 << ENCODER_LEFT_B) | (1 << ENCODER_RIGHT_A) |
      (1 << ENCODER_RIGHT_B));
185 
186   // Enable pin change interrupts
187   PCICR |= (1 << PCIE1);
188   PCMSK1 |= (1 << PCINT8) | (1 << PCINT9) | (1 << PCINT10) | (1 << PCINT11);
189 }
190 
191 // Initialize PID controller variables
192 void init_pid() {
193   integral_left = 0;
194   integral_right = 0;
195   prev_error_left = 0;
196   prev_error_right = 0;
197 }
198 
```

```
// Initialize I2C communication
void init_i2c() {
  TWSR = 0x00;
  TWBR = ((F_CPU / 100000UL) - 16) / 2;  // Set SCL to 100kHz
  TWCR = (1 << TWEN);  // Enable TWI
}

// Start I2C communication
void i2c_start() {
  TWCR = (1 << TWSTA) | (1 << TWEN) | (1 << TWINT);
  while (!(TWCR & (1 << TWINT)));
}

// Stop I2C communication
void i2c_stop() {
  TWCR = (1 << TWSTO) | (1 << TWEN) | (1 << TWINT);
  while (TWCR & (1 << TWSTO));
}

// Write data to I2C
void i2c_write(uint8_t data) {
  TWDR = data;
  TWCR = (1 << TWEN) | (1 << TWINT);
  while (!(TWCR & (1 << TWINT)));
}

// Read data from I2C with ACK
uint8_t i2c_read_ack() {
  TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWEA);
  while (!(TWCR & (1 << TWINT)));
  return TWDR;
}

// Read data from I2C without ACK
uint8_t i2c_read_nack() {
  TWCR = (1 << TWEN) | (1 << TWINT);
  while (!(TWCR & (1 << TWINT)));
  return TWDR;
}

// Read distance from ultrasonic sensor
uint16_t read_distance(uint8_t trig_pin, uint8_t echo_pin) {
  // Send trigger pulse
  PORTB |= (1 << trig_pin);
  _delay_us(10);
  PORTB &= ~(1 << trig_pin);

  // Wait for echo pulse
  while (!(PINB & (1 << echo_pin)));
  TCNT1 = 0;  // Clear timer counter
  while (PINB & (1 << echo_pin));

  // Calculate distance in cm
  uint16_t pulse_width = TCNT1;
  uint16_t distance = pulse_width / 58;

  return distance;
}

// Stop the robot
void stop_robot() {
  // Decelerate both motors to 0 speed
  decelerate(0);
}

// Move the robot forward
void move_forward() {
  // Accelerate both motors to the desired speed
```

```
  accelerate(DESIRED_SPEED);
}


void turn_left_90_degrees() {
  encoder_count_left = 0;
  encoder_count_right = 0;
  int32_t target_encoder_count = (WHEELBASE * CPR) / (WHEEL_DIAMETER*8); // Calculate target encoder counts for 90 degrees

  set_motor_speeds(DESIRED_SPEED/2, DESIRED_SPEED/2, false, true); // Left motor backward, right motor forward

  while (abs(encoder_count_left) < target_encoder_count && abs(encoder_count_right) < target_encoder_count) {
    // Wait until the robot turns 90 degrees based on encoder counts
  }

  stop_motors(); // Stop motors after turning
}

void turn_right_90_degrees() {
  encoder_count_left = 0;
  encoder_count_right = 0;
  int32_t target_encoder_count = (WHEELBASE * CPR) /  (WHEEL_DIAMETER*8); // Calculate target encoder counts for 90 degrees

  set_motor_speeds(DESIRED_SPEED/2, DESIRED_SPEED/2, true, false); // Left motor forward, right motor backward

  while (abs(encoder_count_left) < target_encoder_count && abs(encoder_count_right) < target_encoder_count) {
    // Wait until the robot turns 90 degrees based on encoder counts
  }

  stop_motors(); // Stop motors after turning
}

// Perform a detour maneuver
void detour() {
    float total_distance = 0.0;
    float back_distance = 0.0;

    while (1) {
        uint16_t distance_left = read_distance(TRIG_PIN_LEFT, ECHO_PIN_LEFT);
        uint16_t distance_right = read_distance(TRIG_PIN_RIGHT, ECHO_PIN_RIGHT);

        if (distance_left <= distance_right) {
            if (distance_right > DETOUR_DISTANCE) {
                turn_right_90();
                while (distance_left < STOP_DISTANCE) {
                    distance_left = read_distance(TRIG_PIN_LEFT, ECHO_PIN_LEFT);
                    move_forward();
                    float distance_moved = update_heading();
                    total_distance += distance_moved;
                }
                stop_robot();
                turn_left_90();
                while (distance_left < STOP_DISTANCE) {
                    distance_left = read_distance(TRIG_PIN_LEFT, ECHO_PIN_LEFT);
                    move_forward();
                }
                stop_robot();
                turn_left_90();
                while (back_distance <= total_distance) {
                    move_forward();
                    float distance_moved = update_heading();
                    back_distance += distance_moved;
```

```
                    _delay_ms(100);  // Adjust delay as needed
                }
                break;
            } else {
                stop_robot();
            }
        } else {
            if (distance_left > DETOUR_DISTANCE) {
                turn_left_90();
                while (distance_right < STOP_DISTANCE) {
                    distance_right = read_distance(TRIG_PIN_RIGHT, ECHO_PIN_RIGHT);
                    move_forward();
                    float distance_moved = update_heading();
                    total_distance += distance_moved;
                }
                stop_robot();
                turn_right_90();
                while (distance_right < STOP_DISTANCE) {
                    distance_right = read_distance(TRIG_PIN_RIGHT, ECHO_PIN_RIGHT);
                    move_forward();
                }
                stop_robot();
                turn_right_90();
                while (back_distance <= total_distance) {
                    move_forward();
                    float distance_moved = update_heading();
                    back_distance += distance_moved;
                    _delay_ms(100);  // Adjust delay as needed
                }
                break;
            } else {
                stop_robot();
            }
        }
    }
}



//calculate_speed in meter/seconds
float calculate_speed(int32_t encoder_counts, float time_interval) {
  float wheel_circumference = 3.14159 * WHEEL_DIAMETER;
  float distance_per_count = wheel_circumference / CPR;
  float distance_traveled = encoder_counts * distance_per_count;
  float speed = distance_traveled / time_interval;
  return speed;
}

float map_speed_to_pwm(float speed) {
    if (speed > DESIRED_SPEED) speed = DESIRED_SPEED;
    if (speed < 0) speed = 0;
  uint8_t duty_cycle = (uint8_t)((speed / DESIRED_SPEED) * 255);
    return duty_cycle;
}


void set_motor_speeds(float forward_speed, float backward_speed, bool
    left_motor_forward, bool right_motor_forward) {
  uint8_t forward_pwm = map_speed_to_pwm(forward_speed);
  uint8_t backward_pwm = map_speed_to_pwm(backward_speed);

  if (left_motor_forward) {
    OCR0A = 0;            // Left motor forward
    OCR0B = forward_pwm;  // Set left motor forward speed
    } else {
    OCR0A = backward_pwm; // Left motor backward
    OCR0B = 0;            // Set left motor backward speed
  }
```

```
396
397   if (right_motor_forward) {
398     OCR2A = forward_pwm;  // Right motor forward
399     OCR2B = 0;            // Set right motor forward speed
400     } else {
401     OCR2A = 0;            // Right motor backward
402     OCR2B = backward_pwm; // Set right motor backward speed
403   }
404
405   PORTD |= (1 << MOTOR_LEFT_EN); // Enable left motor
406   PORTD |= (1 << MOTOR_RIGHT_EN); // Enable right motor
407 }
408
409
410
411 // Accelerate both motors to target speed
412 void accelerate(float target_speed) {
413   uint8_t target_pwm = map_speed_to_pwm(target_speed);
414   uint8_t current_pwm_left = OCR0B; // Get current PWM duty cycle for left motor
415   uint8_t current_pwm_right = OCR2A; // Get current PWM duty cycle for right motor
416   float time_interval = 1000.0 / (255.0 * ACCELERATION); // Time interval in
        milliseconds for each step
417
418   while (current_pwm_left < target_pwm || current_pwm_right < target_pwm) {
419     if (current_pwm_left < target_pwm) {
420       current_pwm_left++; // Increase left motor PWM by 1
421       float current_speed_left = (current_pwm_left*DESIRED_SPEED)/255;
422     }
423     if (current_pwm_right < target_pwm) {
424       current_pwm_right++; // Increase right motor PWM by 1
425       float current_speed_right = (current_pwm_right*DESIRED_SPEED)/255;
426     }
427     set_motor_speeds(current_pwm_left, 0, true, false); // Left motor forward
428     set_motor_speeds(0, current_pwm_right, false, true); // Right motor forward
429
430
431
432     _delay_ms(time_interval); // Delay to create a smooth transition
433   }
434
435   // Ensure both motors reach the target speed exactly
436   set_motor_speeds(target_speed, target_speed, true, true);
437 }
438
439
440 // Decelerate both motors to target speed
441 void decelerate(float target_speed) {
442   uint8_t target_pwm = map_speed_to_pwm(float target_speed);
443   uint8_t current_pwm_left = OCR0A; // Get current PWM duty cycle for left motor
444   uint8_t current_pwm_right = OCR0B; // Get current PWM duty cycle for right motor
445   float time_interval = 1000/(255*DECELERATION) ; // Time interval in milliseconds
        for each step
446   while (current_pwm_left > target_pwm || current_pwm_right > target_pwm) {
447     if (current_pwm_left > target_pwm) {
448       current_pwm_left--; // Decrease left motor PWM by 1
449       float current_speed_left = (current_pwm_left*DESIRED_SPEED)/255;
450     }
451     if (current_pwm_right > target_pwm) {
452       current_pwm_right--; // Decrease right motor PWM by 1
453       float current_speed_right = (current_pwm_right*DESIRED_SPEED)/255;
454     }
455     set_motor_speeds(current_pwm_left, 0, true, false); // Left motor forward
456     set_motor_speeds(0, current_pwm_right, false, true); // Right motor forward
457     _delay_ms(time_interval); // Delay to create a smooth transition
458   }
459   set_motor_speed(0, 0 , true , true); // Ensure both motors reach the 0
460 }
461
```

```
462
463 // Read encoder value
464 int16_t read_encoder(uint8_t encoder_a, uint8_t encoder_b) {
465   // These static variables retain their values between function calls
466   static uint8_t last_state_a = 0;
467   static uint8_t last_state_b = 0;
468   static int16_t count = 0;
469
470   // Read current states of encoder pins
471   uint8_t state_a = (PINC & (1 << encoder_a)) ? 1 : 0;
472   uint8_t state_b = (PINC & (1 << encoder_b)) ? 1 : 0;
473
474   // Check for changes in encoder states
475   if (state_a != last_state_a || state_b != last_state_b) {
476     // Determine the direction of rotation
477     if ((last_state_a == 0 && state_a == 1) || (last_state_b == 0 && state_b == 1)) {
478       if (state_a == state_b) {
479         count++;
480         } else {
481         count--;
482       }
483       } else if ((last_state_a == 1 && state_a == 0) || (last_state_b == 1 && state_b
      == 0)) {
484       if (state_a != state_b) {
485         count++;
486         } else {
487         count--;
488       }
489     }
490   }
491
492   // Update last states
493   last_state_a = state_a;
494   last_state_b = state_b;
495
496   return count;
497 }
498
499
500
501
502 // Update robot heading based on encoders
503 float update_heading() {
504   int16_t encoder_counts = (read_encoder(ENCODER_LEFT_A, ENCODER_LEFT_B) +
     read_encoder(ENCODER_RIGHT_A, ENCODER_RIGHT_B))/2; //to get avg count
505     float wheel_circumference = 3.14159 * WHEEL_DIAMETER; // Calculate wheel
     circumference
506     float distance_per_count = wheel_circumference / CPR; // Distance per encoder
     count
507     return encoder_counts * distance_per_count;
508 }
509
510
511
512
513 // Compute PID control output
514 int16_t compute_pid(int16_t setpoint, int16_t measured_value, int16_t *integral,
     int16_t *prev_error) {
515   int16_t error = setpoint - measured_value;
516   *integral += error;
517   int16_t derivative = error - *prev_error;
518   *prev_error = error;
519
520   int16_t output = (KP * error) + (KI * (*integral)) + (KD * derivative);
521   return output;
522 }
523
524
```

```
525
526
527
528
529 // Read current sensor value
530 uint16_t read_current_sensor(uint8_t sensor_addr) {
531   i2c_start();
532   i2c_write((sensor_addr << 1) | 0);
533   i2c_write(0x00);  // Assuming 0x00 is the register to read current
534   i2c_start();
535   i2c_write((sensor_addr << 1) | 1);
536   uint16_t current = (i2c_read_ack() << 8) | i2c_read_nack();
537   i2c_stop();
538   return current;
539 }
540
541 // ISR for pin change interrupt (encoder reading)
542 ISR(PCINT1_vect) {
543   // Read encoders
544   if (PINC & (1 << ENCODER_LEFT_A)) {
545     if (PINC & (1 << ENCODER_LEFT_B)) {
546       encoder_count_left++;
547       } else {
548       encoder_count_left--;
549     }
550     } else {
551     if (PINC & (1 << ENCODER_LEFT_B)) {
552       encoder_count_left--;
553       } else {
554       encoder_count_left++;
555     }
556   }
557
558   if (PINC & (1 << ENCODER_RIGHT_A)) {
559     if (PINC & (1 << ENCODER_RIGHT_B)) {
560       encoder_count_right++;
561       } else {
562       encoder_count_right--;
563     }
564     } else {
565     if (PINC & (1 << ENCODER_RIGHT_B)) {
566       encoder_count_right--;
567       } else {
568       encoder_count_right++;
569     }
570   }
571 }
```

Listing 4.1: full code

Here are some functions with explanations and calculations

1. read_encoder

```
// Read encoder value
int16_t read_encoder(uint8_t encoder_a, uint8_t encoder_b) {
    // These static variables retain their values between function calls
    static uint8_t last_state_a = 0;
    static uint8_t last_state_b = 0;
    static int16_t count = 0;

    // Read current states of encoder pins
    uint8_t state_a = (PINC & (1 << encoder_a)) ? 1 : 0;
    uint8_t state_b = (PINC & (1 << encoder_b)) ? 1 : 0;

    // Check for changes in encoder states
    if (state_a != last_state_a || state_b != last_state_b) {
        // Determine the direction of rotation
        if ((last_state_a == 0 && state_a == 1) || (last_state_b == 0 && state_b ==
    1)) {
            if (state_a == state_b) {
                count++;
            } else {
                count--;
            }
        } else if ((last_state_a == 1 && state_a == 0) || (last_state_b == 1 &&
    state_b == 0)) {
            if (state_a != state_b) {
                count++;
            } else {
                count--;
            }
        }
    }

    // Update last states
    last_state_a = state_a;
    last_state_b = state_b;

    return count;
}
```

Listing 4.2: Read encoder value

Explanation: Static Variables Initialization:

- `last_state_a` and `last_state_b` store the previous states of the encoder signals A and B.
- `count` keeps track of the total encoder ticks.

Reading Encoder States:

- The current states of encoder signals A and B are read from the pins using bitwise operations.
- Example: `state_a = (PINC & (1 « encoder_a)) ? 1 : 0;`

Change Detection:

- The function checks if there is any change in the state of either encoder signal.
- If there is a change, the function determines the direction of rotation and updates the count accordingly.

Updating Last States:

- The last known states of the encoder signals are updated to the current states for use in the next function call.

Return Count:

- The function returns the updated count value, which reflects the cumulative position of the encoder.

2. update_heading

```
// Update robot heading based on encoders
float update_heading() {
    // Read encoder counts and calculate average
    int16_t encoder_counts = (read_encoder(ENCODER_LEFT_A, ENCODER_LEFT_B) +
    read_encoder(ENCODER_RIGHT_A, ENCODER_RIGHT_B)) / 2;

    // Calculate wheel circumference
    float wheel_circumference = 3.14159 * WHEEL_DIAMETER;

    // Calculate distance per encoder count
    float distance_per_count = wheel_circumference / CPR;

    // Return total distance traveled
    return encoder_counts * distance_per_count;
}
```

Listing 4.3: Update robot heading based on encoders

Explanation: Read Encoder Counts and Calculate Average:

- The function reads the encoder counts from both the left and right wheels.
- It calculates the average encoder count by summing the counts from both encoders and dividing by 2.

Calculate Wheel Circumference:

- The wheel circumference is calculated using the formula for the circumference of a circle: $C = \pi \times$ diameter.

Calculate Distance per Encoder Count:

- The distance per encoder count is calculated by dividing the wheel circumference by the number of counts per revolution (CPR).

Return Total Distance Traveled:

- The function multiplies the average encoder counts by the distance per encoder count to get the total distance the robot has traveled.
- This distance is returned as the output of the function.

3. calculate_speed

```
// Calculate speed in meters/second
float calculate_speed(int32_t encoder_counts, float time_interval) {
    // Calculate wheel circumference
    float wheel_circumference = 3.14159 * WHEEL_DIAMETER;

```

```
    // Calculate distance per encoder count
    float distance_per_count = wheel_circumference / CPR;

    // Calculate total distance traveled
    float distance_traveled = encoder_counts * distance_per_count;

    // Calculate speed
    float speed = distance_traveled / time_interval;

    return speed;
}
```

Listing 4.4: Calculate speed in meters/second

Explanation: Steps to Convert Encoder Counts to Speed in meters per second (m/s):

1. Calculate the Distance per Count:
   * Determine the circumference of the wheel ($C = \pi \times D$).
   * Divide the circumference by the number of counts per revolution to get the distance per count (DPC).
2. Calculate Speed:
   * Count the number of pulses (encoder counts) within a specific time interval.
   * Multiply the number of counts by the distance per count to get the distance traveled in that interval.
   * Divide the distance traveled by the time interval to get the speed in meters per second.

4. map_speed_to_pwm

```
// Map speed to PWM duty cycle
float map_speed_to_pwm(float speed) {
    if (speed > DESIRED_SPEED) speed = DESIRED_SPEED;
    if (speed < 0) speed = 0;
    uint8_t duty_cycle = (uint8_t)((speed / DESIRED_SPEED) * 255);
    return duty_cycle;
}
```

Listing 4.5: Map speed to PWM

Explanation: Speed Clamping:

* The first two lines clamp the speed to ensure it falls within the range [0, DESIRED_SPEED].

Mapping Speed to Duty Cycle:

* The next line maps the clamped speed to a duty cycle value:
* `speed / DESIRED_SPEED` scales the speed to a range between 0 and 1.
* Multiplying by 255 scales this value to a range between 0 and 255.
* The result is cast to `uint8_t`, representing the duty cycle.

5. accelerate

```

// Accelerate both motors to target speed
void accelerate(float target_speed) {
  uint8_t target_pwm = map_speed_to_pwm(target_speed);
```

```
  uint8_t current_pwm_left = OCR0B; // Get current PWM duty cycle for
    left motor
  uint8_t current_pwm_right = OCR2A; // Get current PWM duty cycle for
    right motor
  float time_interval = 1000.0 / (255.0 * ACCELERATION); // Time
    interval in milliseconds for each step

  while (current_pwm_left < target_pwm || current_pwm_right < target_pwm
    ) {
    if (current_pwm_left < target_pwm) {
      current_pwm_left++; // Increase left motor PWM by 1
      float current_speed_left = (current_pwm_left*DESIRED_SPEED)/255;
    }
    if (current_pwm_right < target_pwm) {
      current_pwm_right++; // Increase right motor PWM by 1
      float current_speed_right = (current_pwm_right*DESIRED_SPEED)/255;
    }
    set_motor_speeds(current_pwm_left, 0, true, false); // Left motor
    forward
    set_motor_speeds(0, current_pwm_right, false, true); // Right motor
    forward



    _delay_ms(time_interval); // Delay to create a smooth transition
  }

  // Ensure both motors reach the target speed exactly
  set_motor_speeds(target_speed, target_speed, true, true);
}
```

Listing 4.6: Accelerate to target speed

Explanation

This function provides a mechanism to smoothly accelerate both motors from their current speeds to a specified target speed (`target_speed`). It utilizes PWM (Pulse Width Modulation) to control motor speeds, gradually increasing the PWM duty cycle in small steps (`current_pwm_left++` and `current_pwm_right++`) until the motors reach or exceed the desired target speed.
The use of `set_motor_speeds` ensures that each motor's direction and speed are correctly controlled based on the current PWM values. The delay (`_delay_ms(time_interval)`) between each step helps in achieving a gradual acceleration, which is important for smooth motor operation and to prevent sudden changes in speed.

6. turn_left_90

```
void turn_left_90_degrees() {
    encoder_count_left = 0;
    encoder_count_right = 0;
    int32_t target_encoder_count = (WHEELBASE * CPR) / (WHEEL_DIAMETER
    *8); // Calculate target encoder counts for 90 degrees

    set_motor_speeds(DESIRED_SPEED/2, DESIRED_SPEED/2, false, true); //
    Left motor backward, right motor forward

    while (abs(encoder_count_left) < target_encoder_count && abs(
    encoder_count_right) < target_encoder_count) {
        // Wait until the robot turns 90 degrees based on encoder counts
    }

```

```
    stop_motors(); // Stop motors after turning
}
```

Listing 4.7: Turn left approximately 90 degrees

Explanation: Calculate Encoder Counts for Approximately 90 Degrees Turn:

(a) **Circumference of Turn Path:** When turning in place, one wheel moves forward while the other moves backward, resulting in a circular path with a radius equal to half the wheelbase.
   * Radius of the path $= \frac{\text{WHEELBASE}}{2}$
   * For a 90-degree turn, the arc length each wheel travels is a quarter of the circumference of this circle.
   * Arc length $= \frac{\pi \cdot \text{WHEELBASE}}{2 \cdot 4} = \frac{\pi \cdot \text{WHEELBASE}}{8}$

(b) **Convert Arc Length to Encoder Counts:**
   * **Wheel Circumference:** $\pi \cdot \text{WHEEL\_DIAMETER}$
   * **Distance Per Encoder Count:** $\frac{\text{Wheel Circumference}}{\text{CPR}}$
   * **Target Encoder Count:** $\frac{\text{Arc Length}}{\text{Distance Per Encoder Count}}$

   Combining these steps:

$$\text{target\_encoder\_count} = \frac{\pi \cdot \frac{\text{WHEELBASE}}{8}}{\pi \cdot \frac{\text{WHEEL\_DIAMETER}}{\text{CPR}}} = \frac{\frac{\text{WHEELBASE}}{8}}{\frac{\text{WHEEL\_DIAMETER}}{\text{CPR}}} = \frac{\text{WHEELBASE} \cdot \text{CPR}}{8 \cdot \text{WHEEL\_DIAMETE}$$

7. read_distance

```
// Read distance from ultrasonic sensor
uint16_t read_distance(uint8_t trig_pin, uint8_t echo_pin) {
    // Send trigger pulse
    PORTB |= (1 << trig_pin);
    _delay_us(10);
    PORTB &= ~(1 << trig_pin);

    // Wait for echo pulse
    while (!(PINB & (1 << echo_pin)));
    TCNT1 = 0;  // Clear timer counter
    while (PINB & (1 << echo_pin));

    // Calculate distance in cm
    uint16_t pulse_width = TCNT1;
    uint16_t distance = pulse_width / 58;

    return distance;
}
```

Listing 4.8: Read distance from ultrasonic sensor

Explanation: Distance Calculation from Ultrasonic Sensor:

* Speed of Sound:
  · The speed of sound is approximately 343 meters per second in air at room temperature.
  · This translates to 0.0343 centimeters per microsecond (μs).
* Round-Trip Travel:
  · The ultrasonic sensor sends a sound wave that travels to the object and reflects back to the sensor.

- Therefore, the total distance traveled by the sound wave is twice the distance to the object.

* Time Calculation:
  - The time measured by the sensor (in microseconds) is the round-trip time for the sound wave.
  - To calculate the one-way distance to the object, we divide the round-trip time by 2.
* Distance Calculation:
  - The formula to calculate the distance in centimeters is:

$$\text{Distance (cm)} = \frac{\text{Time } (\mu s}{58}$$

Summary

This report explores the integration of advanced technologies, specifically obstacle avoidance systems for Autonomous Mobile Robots (AMRs) and Automated Guided Vehicles (AGVs), within warehouse management. The implementation of these systems is crucial for enhancing efficiency, safety, and productivity in warehouse operations. Key achievements include the successful design and integration of obstacle avoidance technologies, leveraging ultrasonic sensors for precise detection. Challenges encountered, such as sensor calibration and integration complexities, were mitigated through iterative testing and refinement processes. Moving forward, continuous advancements in sensor technology and algorithm optimization present opportunities to further enhance system accuracy and reliability. Overall, the deployment of obstacle avoidance systems represents a significant stride towards optimizing warehouse logistics, ensuring safer environments for personnel and equipment while maximizing operational efficiency.

Acknowledgement

We would like to extend our heartfelt gratitude to Prof. J.A.K.S. Jayasinghe and Dr. S. Thayaparan for their invaluable guidance, expertise, and unwavering support throughout this project. Their insightful feedback and encouragement have been instrumental in shaping the direction and outcomes of our project.
Special thanks are also due to all the members of our group who contributed tirelessly to various aspects of this project. Their dedication, collaboration, and diverse skills have significantly enriched our work.

Bibliography

[1] I. Susnea, V. Minzu, and G. Vasiliu, "Simple, Real-Time Obstacle Avoidance Algorithm for Mobile Robots," in *Proceedings of the IEEE International Conference on Control Engineering*, Galati, Romania, 2008, pp. 1-6.

[2] A. N. A. Rafai, N. Adzhar, and N. I. Jaini, "A Review on Path Planning and Obstacle Avoidance Algorithms for Autonomous Mobile Robots," *Journal Name*.

[3] M. C. De Simone, Z. B. Rivera, and D. Guida, "Obstacle Avoidance System for Unmanned Ground Vehicles by Using Ultrasonic Sensors," *Machines*, vol. 6, no. 2, p. 18, Apr. 2018, doi: 10.3390/machines6020018.

[4] R. Vairavan, S. Ajith Kumar, L. Shabin Ashiff, and C. Godwin Jose, "Obstacle Avoidance Robotic Vehicle Using Ultrasonic Sensor, Arduino Controller," *International Research Journal of Engineering and Technology (IRJET)*, vol. 5, no. 2, Feb. 2018.

[5] M. Pires, P. Couto, A. Santos, and V. Filipe, "Obstacle Detection for Autonomous Guided Vehicles through Point Cloud Clustering Using Depth Data," *Machines*, vol. 10, no. 5, p. 332, May 2022, doi: 10.3390/machines10050332.

[6] inVia Robotics, "AMR vs AGV Robotic Solutions in Warehouse Automation," YouTube, May 2, 2023. [Online]. Available: `https://youtu.be/qahujJ-8vdk?si=9dRc1LR8B7RQlFGD`.

[7] "Understanding AMR Robots: A Comprehensive Guide," *Wevolver*. [Online]. Available: `https://www.wevolver.com/article/understanding-amr-robots-a-comprehensive-guide`.

[8] P. Kanade, P. Alva, S. Kanade, and S. Ghatwal, "Automated Robot ARM using Ultrasonic Sensor in Assembly Line," *International Research Journal of Engineering and Technology (IRJET)*, vol. 7, no. 12, pp. 615, Dec. 2020. [Online]. Available: `https://www.researchgate.net/publication/347930903_Automated_Robot_ARM_using_Ultrasonic_Sensor_in_Assembly_Line`.

[9] "Motor Driver Open Loop vs Closed Loop," *Robot Platform*. [Online]. Available: `https://www.robotplatform.com/knowledge/motion_control/feedback_control.html`.

[10] "An Introduction to Types of Closed-loop Motor Control," *Control.com*. [Online]. Available: `https://control.com/technical-articles/in-introduction-to-types-of-closed-loop-motor-control/`.

[11] "Mobile Robot," *Wikipedia*.[Online]. Available: `https://en.wikipedia.org/wiki/Mobile_robot.`.

[12] P. Millett, "Motor Driver PCB Layout Guidelines – Part 1," *Monolithic Power Systems*. [Online]. Available: `https://www.monolithicpower.com/learning/resources/motor-driver-pcb-layout-guidelines-part-1`.

Appendix A

Daily Work Log

A.1 Background Research and Conceptual Design (04.03.2024 – 11.03.2024)

This week, we embarked on an in-depth exploration to identify the optimal design for our AMRS and AVGs. Leveraging a wide array of internet resources, we meticulously developed four distinct conceptual designs. Each design was crafted with careful consideration of various factors critical to our project's success.
After creating these initial concepts, we systematically evaluated each one against a set of predefined criteria tailored to our specific needs. These criteria included functionality, efficiency, safety, scalability, and cost-effectiveness. Through rigorous analysis and comparison, we were able to identify the conceptual design that best aligns with our requirements and project objectives.
This comprehensive approach ensured that our chosen design is not only innovative and effective but also practical and feasible, setting a solid foundation for the subsequent phases of our project.
Here is the finally selected Conceptual Design.

A.1.1 Selected Conceptual design

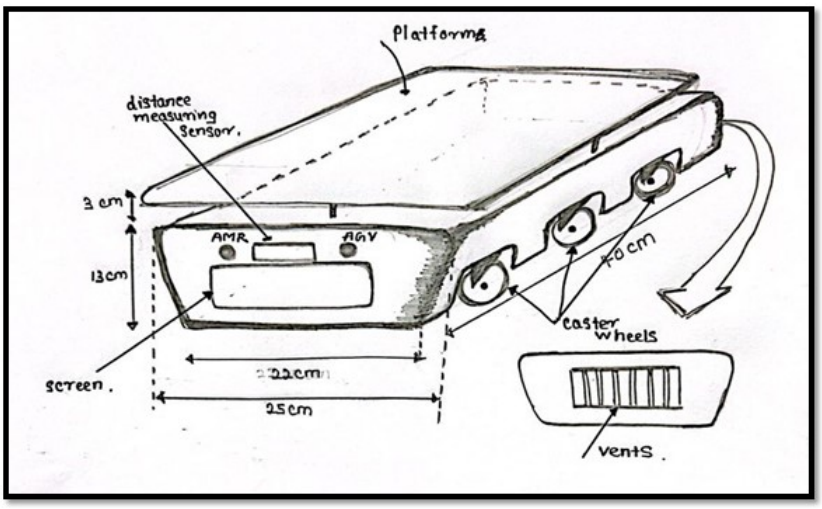


Figure A.1: Selected Conceptual Design

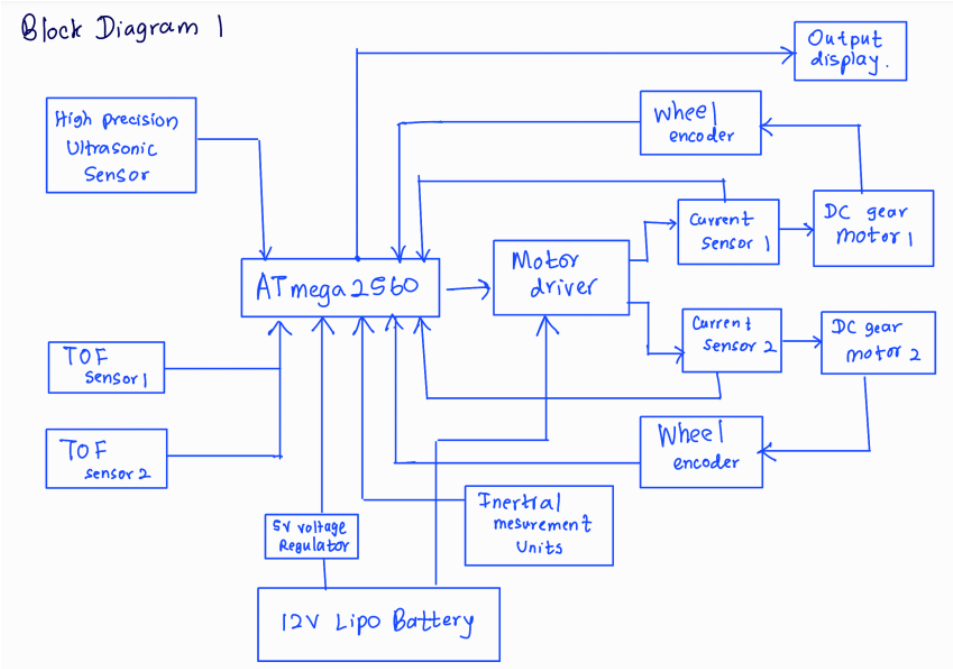


Figure A.2: Selected Functional Block Diagram

A.2 Design Motor Controller (12.03.2024 – 19.03.204)

This week, we delved into the realm of motor controllers. In our quest for an optimal design, we recognized the critical importance of incorporating a feedback system to enhance performance. Consequently, we designed our motor controller with a feedback mechanism to ensure improved functionality and precise control.
We chose a feedback system because:

(a) Improved Accuracy: A feedback system continuously monitors and adjusts motor output, ensuring precise control over speed, position, and torque, which leads to greater accuracy in executing desired tasks.

(b) Enhanced Stability: Feedback helps stabilize motor operation by detecting and correcting deviations from desired parameters, resulting in smoother and more consistent performance.

(c) Increased Responsiveness: With real-time feedback, the motor controller can quickly respond to changes in operating conditions or external disturbances, maintaining optimal performance even in dynamic environments.

(d) Fault Detection and Diagnosis: Feedback systems can identify abnormalities or malfunctions early, allowing for timely intervention and minimizing downtime or potential damage.

Here is our final design ,

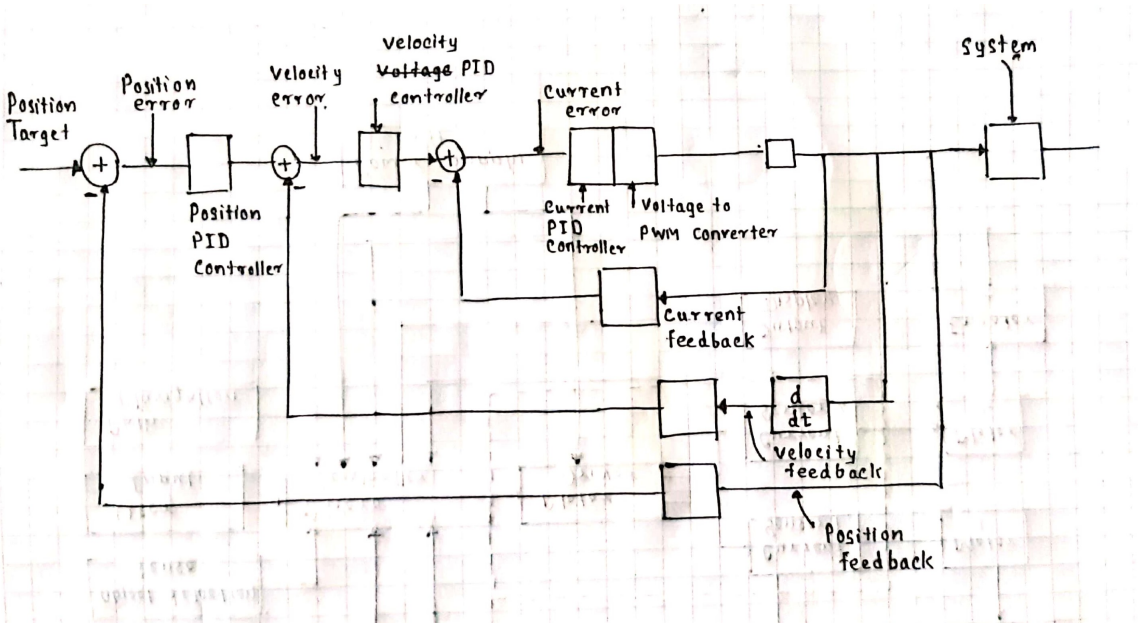


Figure A.3: Motor Control Design

A.3 Making Schematics and PCB for Main Controller (20.03.2024 – 27.03.2024)

This week, we mainly focus on making Schematics and PCB. Firstly, we embarked on a comprehensive process of component selection. This involved carefully evaluating various electronic components available in the market to choose those best suited to our project requirements. Factors such as functionality, performance, compatibility, availability, and cost were meticulously considered during this phase. Next, armed with our selected components, we proceeded to design the schematics. Adhering to industrial guidelines and best practices, we meticulously crafted the schematics to ensure clarity, efficiency, and reliability. Every connection, component placement, and signal flow was carefully planned and documented to facilitate seamless integration and troubleshooting in subsequent stages of the project. Utilizing industry-standard software such as Altium, we then translated our schematic designs into physical PCB layouts. This process involved placing components strategically on the board, optimizing routing paths to minimize signal interference, and ensuring adequate spacing for efficient heat dissipation and manufacturability. Attention to detail was paramount at this stage to guarantee the integrity and functionality of the final PCB design. Through these meticulous efforts, we successfully created the main controller PCB for our project. This PCB serves as a critical component, housing the core processing unit and facilitating communication between various subsystems. By leveraging advanced software tools and adhering to established design principles, we have laid a solid foundation for the realization of our project goals.

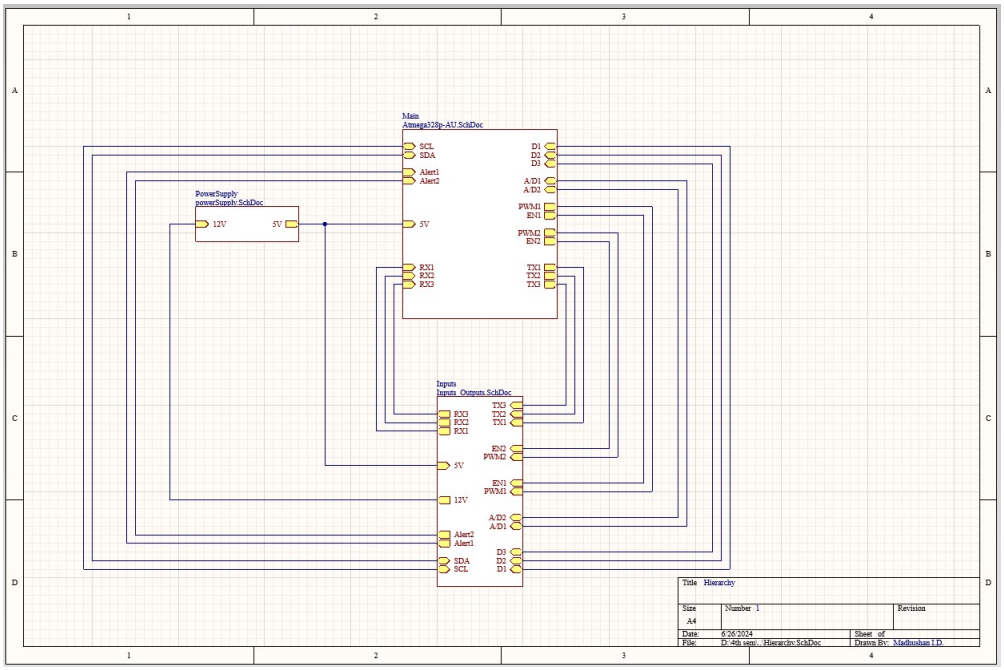


Figure A.4: Main PCB Schematic

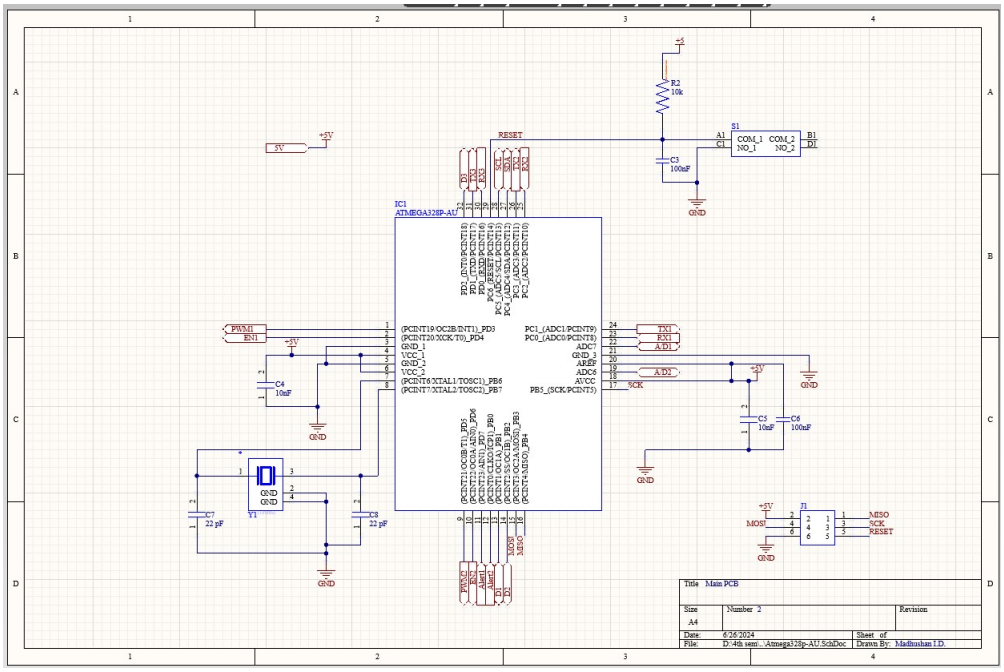


Figure A.5: Main PCB Schematic

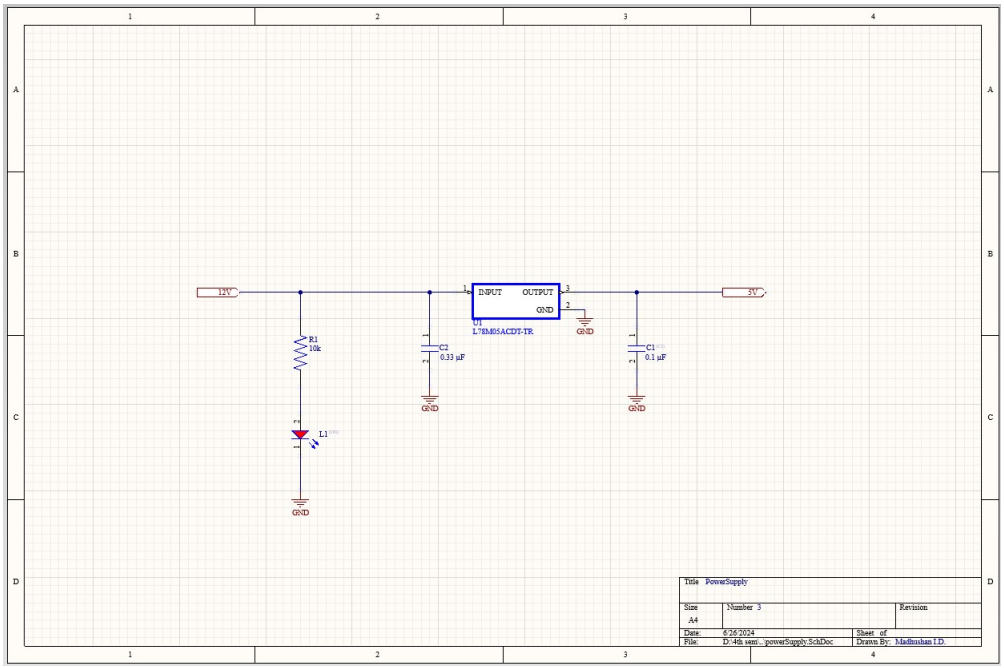


Figure A.6: Main PCB Schematic

A.4 Making Schematic and PCB for Motor Controller(28.03.2024 – 04.04.2024)

This week was dedicated to the crucial task of designing our motor controller schematic and PCB. Building upon our previous efforts of component selection, we meticulously curated a set of components tailored to the specific requirements of our motor controller. With our components in hand, we proceeded to translate our design into a comprehensive schematic. Then we design the PCB by using Altium as previous Week.

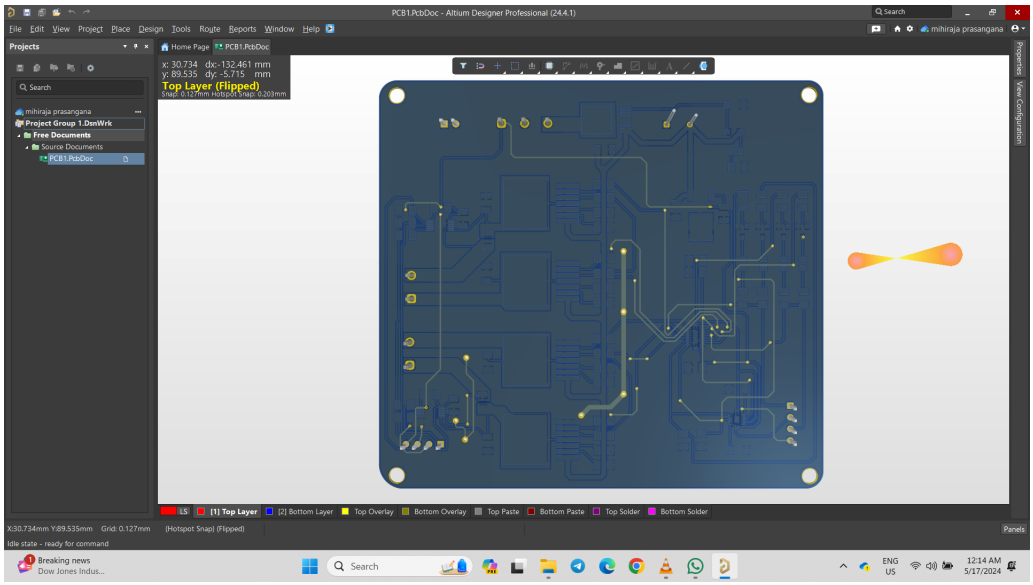


Figure A.7: Motor control PCB Top layer

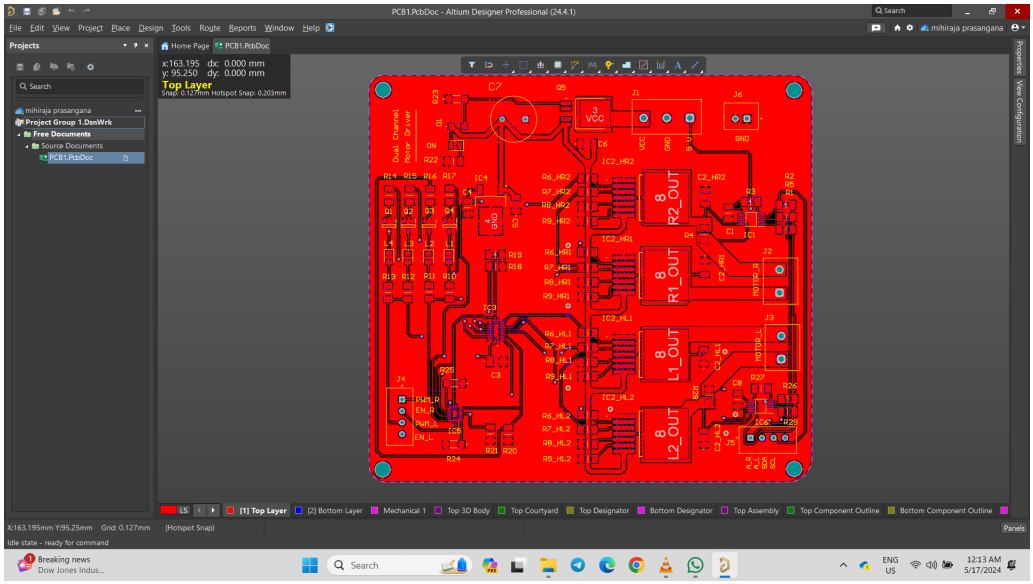


Figure A.8: Motor control PCB Bottom layer

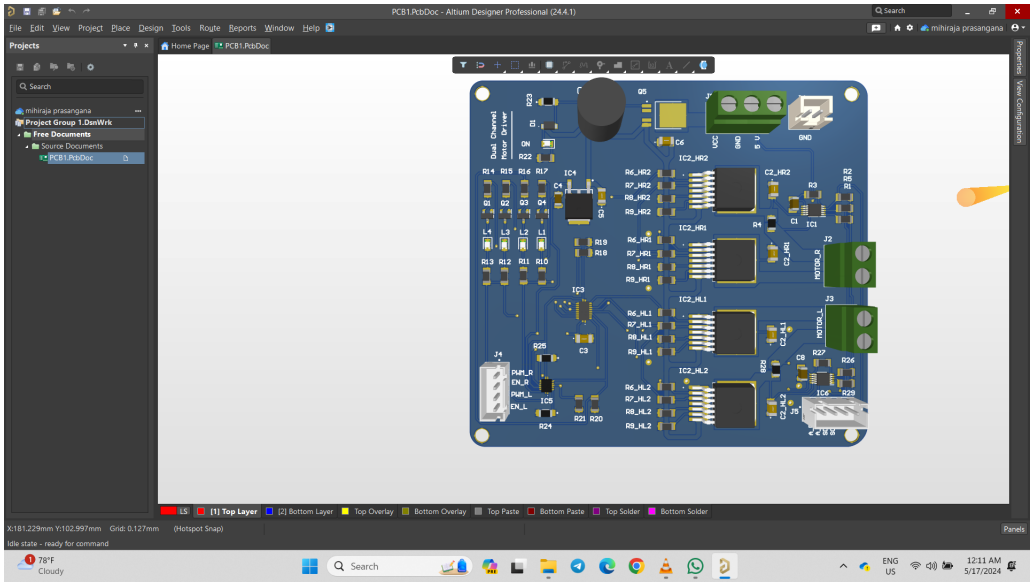


Figure A.9: Motor control PCB 3D view

A.5 Design Enclosure Using Solidworks (05.04.2024 – 11.04.2024)

This week, our focus shifted to the crucial task of designing the enclosure for our robot. Understanding the importance of this component in protecting the internal mechanisms while ensuring optimal functionality, we embarked on a meticulous planning process. To begin, we identified key evaluation criteria that would guide our design process. Functionality emerged as a paramount consideration, encompassing factors such as accessibility for maintenance, ease of assembly, and compatibility with the robot's components. Additionally, durability played a critical role, as the enclosure needed to withstand various environmental conditions and potential impacts encountered during operation. With these criteria in mind, we leveraged SolidWorks, a powerful 3D modeling software, to bring our design concept to life. Carefully considering the dimensions, shape, and materials, we crafted an enclosure that seamlessly integrated with the robot's form factor while providing ample protection and accessibility for its internal components.

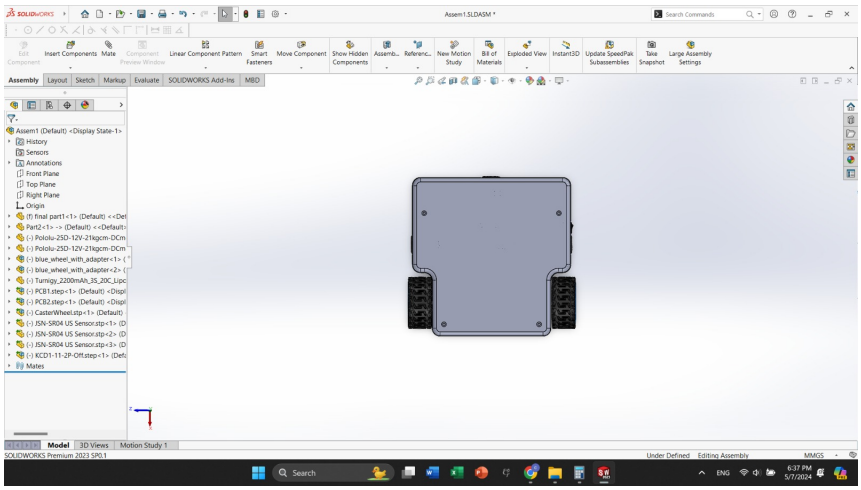


Figure A.10: Solid works design

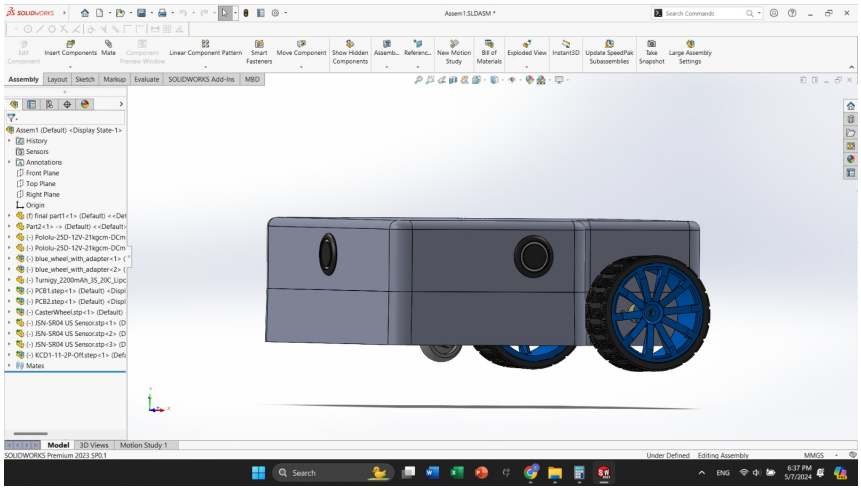


Figure A.11: Solid works design

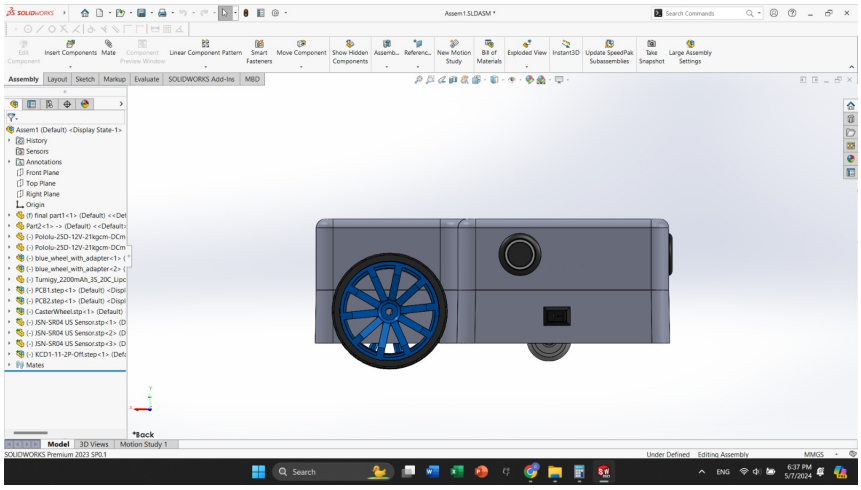


Figure A.12: Solid works design

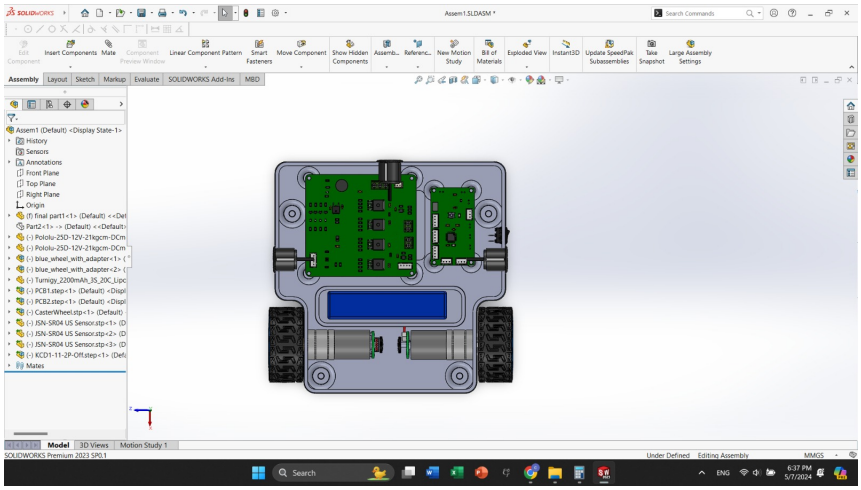


Figure A.13: Solid works design

A.6 Finalizing PCB ordering Components (15.04.2024-20.04.2024)

This week was marked by a crucial step in our project timeline: preparing our PCBs for manufacturing and ordering the necessary components to bring our designs to life. First and foremost, we understood the critical importance of ensuring that our PCB designs were error-free before sending them to JCL PCB for production. With meticulous attention to detail, we conducted thorough reviews of our PCB layouts, identifying any potential mistakes or discrepancies. Once identified, we promptly corrected these errors to guarantee that our designs were 100After finalizing our PCB designs, we proceeded to place our order with JCL PCB, confident in the precision and quality of our designs. This step marked a significant milestone in our project, as it signaled the transition from the design phase to the manufacturing phase. Simultaneously, we turned our attention to procuring the components needed to populate our PCBs. Leveraging the vast inventory and reliable service of Mouser.com, we meticulously compiled a list of the required components, taking into account factors such as availability, compatibility, and pricing. With our component list finalized, we placed our order with Mouser.com, ensuring that all necessary components would be readily available to complete our PCB assemblies upon their return from manufacturing.

A.7 Develop the Code for motor controller(21.04.2024 – 28.04.204)

This week, our focus shifted towards the implementation of code for our feedback motor control system. Recognizing the significance of this task in ensuring the proper operation and performance of our system, we delved deep into the intricacies of feedback control algorithms. Drawing upon resources available on the internet, we gained valuable insights and ideas regarding the structure and logic of the code. However, we encountered challenges in fully realizing and completing the implementation. Undeterred, we approached this task with a methodical mindset, breaking down the code into manageable components and systematically addressing each aspect. We carefully analyzed the requirements of our feedback system, including sensor input processing, control algorithm execution, and motor output modulation.

A.8 Soldering the PCBs and Testing (29.04.2024 – 05.05.2024)

After receiving our two PCBs from manufacturing and the necessary components from Mouser.com, we commenced the soldering process this week. With precision and care, we meticulously soldered each component onto the PCBs, following the layout and guidelines established during the design phase. Once the soldering process was complete, we conducted thorough inspections to ensure the integrity of the solder joints and the proper placement of all components. With confidence in the quality of our assembly, we proceeded to the testing phase. During testing, we systematically checked each functionality of the PCBs to verify their correct operation. This included testing input and output connections, sensor readings, and motor control responses. Any discrepancies or anomalies were carefully noted and addressed through troubleshooting and adjustments as needed. Through this rigorous testing process, we confirmed that both PCBs were functioning correctly and meeting the desired specifications. This validation step was crucial in ensuring the reliability and performance of our motor control system before integration into the larger project framework.

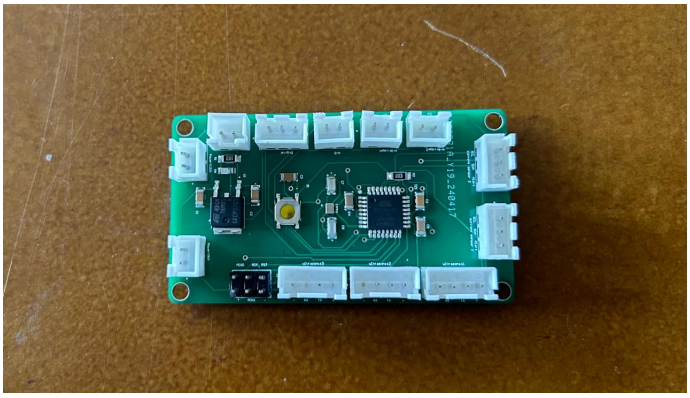


Figure A.14: Main PCB

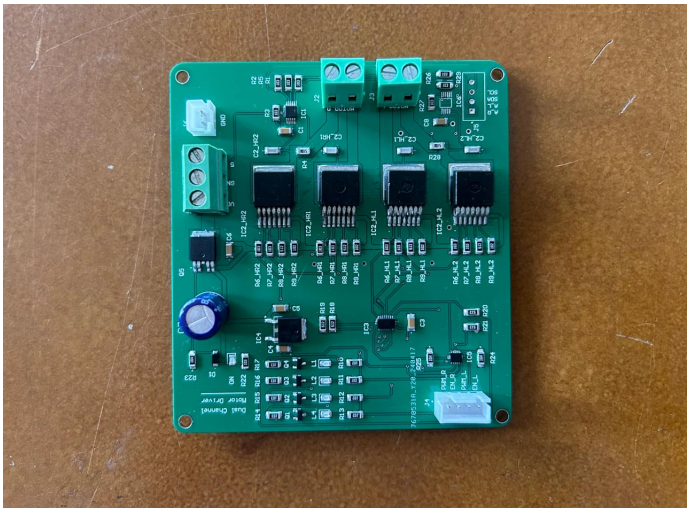


Figure A.15: Motor controller PCB

This documentation has been reviewed by Group H - Multi Turn Absolute Magnetic Encoder

Name	Signature
1. Epa Y. L. A.	
2. Epa Y. R. A.