AMAL - AN AUTONOMOUS DELIVERY ROBOT

Muhammad Ammar Khan

Laraib Aftab Zedie

Muzammil Tariq

Syed Muhammed Abdullah Arif

HABIB UNIVERSITY

KARACHI, PAKISTAN

2022

AMAL - AN AUTONOMOUS DELIVERY ROBOT

The Capstone Design Project
presented to the academic faculty

by

Muhammad Ammar Khan

Laraib Aftab Zedie

Muzammil Tariq

Syed Muhammed Abdullah Arif

in partial fulfillment of the requirements for
BS Electrical Engineering
in the
School of Science and Engineering

Habib University

May 2022

AMAL - AN AUTONOMOUS DELIVERY ROBOT

This capstone design project was advised by:

Dr. Muhammad Farhan	Dr. Muhammad Mobeen Movania
Faculty of Electrical Engineering	Faculty of Computer Science
Habib University	*Habib University*

Approved by the Faculty of Electrical Engineering on April 19, 2022.

”The greatest glory in living lies not in never falling, but in rising every time we fall.”

Nelson Mandela

This work is dedicated to our families, teachers, and most importantly, our batch of electrical engineers (2022) whom efforts, support, positivity, and guidance always kept us motivated and eventually enabled us to achieve much in life that we were able to complete this work.

ACKNOWLEDGEMENTS

We want to acknowledge Dr. Muhammad Farhan for his support, guidance, technical help, appreciation, and motivation throughout the course of this project.

We want to acknowledge Dr. Shafayat Abrar for his guidance and assistance in research related work of the project.

We want to acknowledge Dr. Abdullah Umair Bajwa for his assistance in our mechanical related challenges of the robot

We want to acknowledge Dr. Abdul Basit Memon for his advices, appreciation, and guidance in the technical challenges we faced in the capstone.

We want to acknowledge Mr. Irfan for his help in workshop related work.

We want to acknowledge Mr. Hassaan Shah for his advices & recommendations in engineering work.

We want to acknowledge Mr. Zeeshan Nafees for his major assistance in designing and fabricating PCB related work and other hardware related challenges.

We want to acknowledge Dr. Muhammad Mobeen Movania on his consistent support and appreciation throughout the course of this project.

We really want to acknowledge the efforts of Mr. Ghayas Ahmedani in always providing us motivational, technical, and financial support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

EXECUTIVE SUMMARY

This project is an inter-disciplinary project between CS and EE. **The primary objective is to build an automated delivery system based on the industry 4.0 paradigm, with the motivation of helping society run itself with minimal human interactions.** Industry 4.0 is the automation of digital systems that exist right now. It uses the idea of IoT to enable automation in everyday tasks hence saving time.[1] The main idea behind our project proposal is a robot that will receive food orders from an accompanying mobile and web application. Along with this, the order will also receive the location of the user and then retrieve and deliver the order from the cafe counter to the location of the user.[1] The work distribution between EE and CS is primarily in the hardware and software components with some overlap in the connection between them. The main EE component would be the robot itself which will house the necessary sensors such as the LIDAR, IMU, and camera which will be used for localization, mapping, and obstacle avoidance.

KEYWORDS: Industry 4.0, Autonomous robots, Navigation, Mapping, Obstacle Avoidance

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Introduction

Robotic package delivery, once thought to be a distant future, is now a very plausible possibility. Because of advancements in robotics, GPS monitoring, automation, and navigation, you may not see a delivery person carrying your product at your door.

The delivery robots market is expected to grow from USD 212 million in 2021 to USD 957 million by 2026; it is expected to grow at a CAGR of 35.1 percent during 2021–2026[2]. Reduction in delivery costs in last-mile deliveries, and Increase in venture funding are the key factors driving the growth of the market. Further, the Worldwide growth of the e-commerce market is a factor propelling the growth of the market. The significant growth is due to the impact of the current pandemic and the change in market dynamics[1]. Apart from the need in the current era, optimizing tasks and

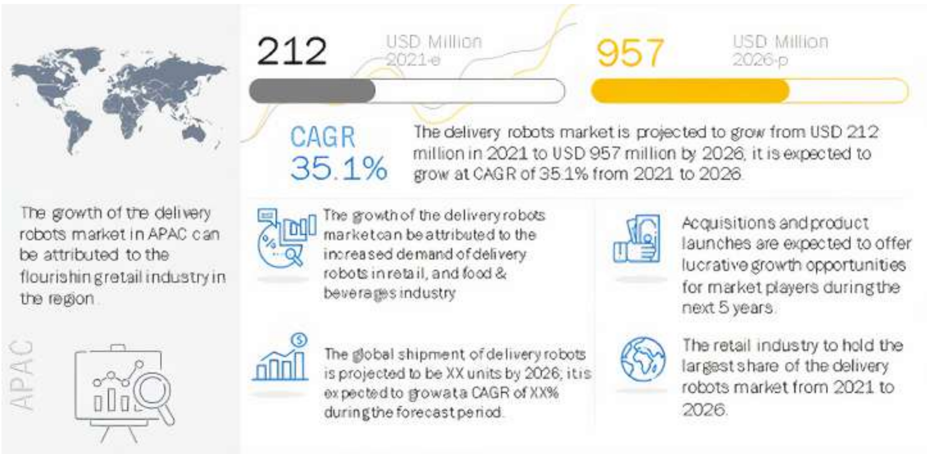


Figure 1.1: CAGR of Delivery Robots [2]

productivity, increasing convenience for people, and automating processes have always been a big priority for us in terms of development[3]. In this context, the industrial revolution has also been under consideration where we are aiming towards increasing the productivity of industry and automating internal processes. Thus, there is a **need** of such techniques that allow certain operations to be automated within a premise and we propose that it could be achieved through the intervention of our solution.

1.2 Literature Review

Internet of Things (IoT) based food delivery systems started to come into existence ever since the fourth industrial revolution happened. Industry 4.0, the name given to the fourth industrial revolution, is an incremental advancement over the previous works in the industrial field. It utilizes IoT to increase efficiency and gearing products more towards customer requirements [4]. Industry 4.0 uses technologies such as sensors and machine-to-machine communication to allow separate devices to connect hence creating an internet of things. A good blend of modern technologies creates the necessary frameworks required to engineer a digital solution. For an IoT-based food delivery system to work properly, it would need a Robot Operating Systems (ROS), a Light Detection and Ranging (LiDAR) sensor, and a rigid robot structure along with the required software technologies to create an accompanying application. The applications can be created using several of the available frontend and backend technologies. In recent times Flutter and Firebase [5] have proved to be a superb combination of application development. Several works have been done in the past utilizing some of these technologies.[6]

SwiftBot is a great example of how to use technologies like LiDAR and ROS, as well as other technologies. SwiftBot's goal was to create a self-driving delivery robot that could perform inside deliveries for couriers. The delivery of packages in a safe, secure, and timely way is the key issue that this program tries to address. It functions in such a way that the user may communicate with the robot using a smartphone app. When the robot arrives, the user inserts the package that needs to be delivered, and the robot then delivers it autonomously[7] I'm going to delve a little more into the robot's technicalities. It's the user's location that's connected to the app that lets users schedule and manage deliveries. The location is sent to the robot via a central server. SwiftBot performs 2D environment mapping using LiDAR, obstacle identification and avoidance using Lidar, autonomous navigation, and path planning sensors and odometric information as some of its most critical functions.

Food delivery robots have been in use in a number of towns and cities across Europe and the United States for a number of years, and in the current context, they offer the important benefit of limiting inter-personal interaction, hence reducing the risk of infection spreading. Since 2018, Starship Technologies[8] Figure (1.2) has been providing robotic delivery services in the United Kingdom and the United States through its battery-powered, six-wheeled robot. The robot can carry a weight of 10 kg and travel around 10 kilometers between charges. It was announced in March 2020 that it will

Figure 1.2: Starship Delivery Robot [8]

provide a free delivery service to National Health Service staff who live within its operational region and that it had performed 100,000 autonomous deliveries in the town. Start-up in Silicon Valley Nuro has started delivering groceries in the Houston region with its R2 autonomous robot[8] Figure (1.3), which can move at speeds of up to 40 km/h and carry up to 190 kg. The robots are also transporting meals, personal protective equipment, clean linens, and other supplies to personnel at two California facilities: the Event Centre in San Mateo and the Sleep Train Arena, both of which have been turned into field hospitals to accommodate the overflow of COVID19 patients. ZhenRobotics, located in Beijing, is the manufacturer of the six-wheeled RoboPony delivery robot. This has a 30-kilogram capacity and can run for up to 16 hours on a single charge from its lithium-ion battery. It has an inbuilt UV source to disinfect objects in its compart-

Figure 1.3: R2 Autonomous Robot [8]

ment and is being utilized by Sunin.com Group Ltd.[8], a prominent Chinese retailer, to carry food and other items to COVID-affected families in Nanjing. Orders have increased threefold since the epidemic began, according to the business, which plans to build 90 additional units to accommodate demand. Also in China, White Rhino Zhida Technology's conventionally powered, autonomous vehicles are bringing fresh food to 2,700 Beijing residents as well as medications and other essential supplies to Wuhan Fangcai Hospital.

Hence, Autonomous robots are used in a variety of applications. Similarly, the goal of this thesis is to design and build a self-contained wheeled vehicle that can make deliveries. Because an autonomous agent may be used in both indoor and outdoor situations, this thesis focuses on interior settings, with all criteria and design procedures met for both operating bounds. Companies such as Keenon Robotics Co. fig(1.4), Ltd. is a global leader in artificial intelligence with an emphasis on indoor intelligent service robots. They are experts in the field of interior autonomous driving and provide our customers with cutting-edge intelligent unmanned delivery solutions. Catering, medical care, hotels, entertainment, retail, venues, government affairs, offices, real estate, communities, banking, posts, finance, insurance, airports, and stations are just a few of the industries where our goods are used. Keenon Robotics Co. Ltd is based in Shanghai and focuses on commercial service robots such as 'Delivery Robot-T6'[9] and 'Delivery Robot-T5' that are reliable, efficient, and practical[9]. Their fundamental competitiveness is autonomous positioning navigation and core sensors, which allow our robots to interact with their surroundings on their own and function for long periods of time without human involvement. The robots featured are defined as lighter weight and slim body, ”running” more flexible and smarter. More cost-effective and affordable. Also, in Multi-point delivery mode, four tables of dishes can be delivered at one time

Figure 1.4: Delivery Robot- T6 [9]

Locus Robotics Figure (1.5) creates autonomous, mobile robots to help merchants and warehouse logistics providers with e-commerce. Its technology works in tandem with employees to boost order productivity by 2X-3X and enhance fulfillment speed and throughput by 2X-3X with near-perfect order accuracy, ensuring that customers receive their goods as soon and precisely as possible.[10] Keeping in view the existing products,

Figure 1.5: Locus Robotics

Our motivation is to develop and implement a Wheeled Mobile Robot (WMR) and use a System Engineering technique to verify skid-steering performance on specified trajectories. System needs in mechanical, electrical, and software are analyzed from this perspective, and the whole system is separated into three subblocks: motor processor, image processor, ROS (Robot Operating System) which acts as a central processor. The indoor and outdoor location is one of the most important considerations [10]. While the Global Positioning System (GPS) is frequently used to tackle outside activities, inside navigation presents certain difficulties. In this paper, we introduce a solution in which navigation can be done in indoor environments through different mapping techniques using ROS (Robot Operating Systems)

1.3 Problem Statement/Objective

The primary objective is to build an automated delivery system based on the industry 4.0 paradigm, with the motivation of helping small businesses and societies. Our long-term goal is to not only use this framework for food delivery in indoor spaces but to also serve society such as helping autistic patients, and in hospitals for indoor delivery of medicine. In a nutshell, The robot is meant to function as an autonomous mobile robot platform for a variety of applications, including transportation in industrial sectors, food and medication delivery in hospitals, unmanned missions, and security. Our goal for this year is to create a self-driving delivery mobile robot that can deliver food items as part of the indoor service from point A (cashier's counter) to point B (any of the serving tables) in the Habib Universities Cafeteria.

CHAPTER 2

DESIGN PROCESS

2.1 Clients/Stakeholders and their requirements

Figure 2.1 highlights our major stakeholders for the project and also tells the overall impact of our project on them. Several stakeholders, including hospitals, office complexes,

Stakeholder Name/Role	Description	Interest	Primary Benefits	Primary Detriments	Net Impact
Habib University	The application is directed towards making a delivery system within the premises of the university	High	Increased efficiency, marketing, source of attraction, and research opportunity	None	Positive
Industries and Hospitals	Since the primary focus is on industry 4.0, they are the utmost stakeholders	High	Increased efficiency, safety, less monetary overhead, precise and better operations	High Investment	Positive
Hotels and Supermarkets	Serving robots can be deployed to aid general operations within the premises of such areas	High	Robots can provide easier management, better production, and can be attract public towards them	Might need constant supervision as these are public areas	Positive
Work Force and Labor	Less labor will be required after implementing this framework	High	Labor would be trained to operate high end equipment and would be relaxed	Work force responsible for physical work and deliveries would lose their jobs	Neutral
R&D Departments	Technical novel aspects do open up various research opportunities	High	Newer platform to test out their research and would be able to explore newer ideas	None	Positive
Disabled personnel's	The idea is to serve those who cant help themselves, in terms of self service	High	it will be more feasible for them to get things delivered to them without them making an effort	None	Positive.

Figure 2.1: Stakeholder Assessment Summary

university campuses, and hotels. Based on the goal of implementing an automated delivery service to make the domestic transportation of materials more efficient, some of

The characteristics that have been identified are listed as follows:

1. An automated system one robot can navigate automatically in the facility operates with a payload.

2. The act of planning the path to the target for receipt or delivery.

3. Obstacle detection and avoidance to avoid impeding the user in the operating environment and ensure unhindered movement to the destination.

4. Mobile application-based service to coordinate, monitor, and manage deliveries.

5. An admin panel to monitor shipping agents and delivery status of items and the ability to operate the agent remotely.

2.2 Design Concepts

ZAAVIA solutions is a recognized provider of electronic design, development, and other engineering services for a variety of public/private organizations, the avionics/aerospace community, and industrial and academic institutions. based on it. and broader development team/electronic design. As per our affiliated Industries requirements, our project aims to introduce a concept that is not usually thought about in the context of the indoor food delivery industry or service robotics. A goal of the project is to introduce an idea that isn't commonly thought of in the context of Indoor food delivery and service robotics. It is one of the aims of the project to introduce an idea that is not commonly applied to the indoor food delivery industry and service robotics. In the context of food delivery indoors, we want to introduce an idea that is not typically thought of, along with service robotics. One of the goals of the project is to develop a new concept that is rarely thought about in terms of the Indoor food delivery industry and service robotics. It is the project's objective to introduce an idea that is not typically associated with indoor food delivery, or with service robotics. Our goal with this project is to introduce an innovative concept that is rarely considered in the context of indoor food delivery and service robotics. Among the aims of the project is to introduce an idea that is not generally thought about in the context of the Indoor food delivery industry and service robotics. We're working on introducing an idea rarely seen in the context of indoor food delivery and robots for service. As part of the project, we want to introduce an idea that is not usually associated with the industry of Indoor Food Delivery and Service Robotics.

For our proof of concept prototype, we had several options that could've been implemented in our hardware design. The ideation segment of the assignment provided a range of opportunity layout selections that needed to be weighed in opposition to every difference even as preserving in thoughts the meant functionality, scope, and person necessities of the assignment.

2.2.1 Hardware Design

Our prototype system is planned and constructed following design standards, and open field tests have begun. Environmental problems have been adequately addressed, and a solid structure has been built. Our proof-of-concept prototype weighs around 8kg and has a payload capacity of 3.063kg (theoretically). In terms of hardware design, there were a total of six significant restrictions. These are outlined in the following paragraphs:

1. Modularity: The robot platform must be modular to add units or parts to the robot quickly. Additional navigation sensors, payload space, extra onboard power, and other gadgets for an effective human-machine interaction are examples of these modules or pieces.

2. Low-cost Manufacturing: While mobile robots are available on the market, they are often costly, raising research and development expenditures. Furthermore, in the event of big volume production, using a ready-made robot will increase the cost of production even more.[10]

3. Truncated Structure: During the prototype phase, the construction is kept basic and truncated to save time and money while focusing on the intended functional-

ity.

4. Suitability of Environmental Conditions: The robot is intended for usage in outside situations, which means it will need to travel on roads and other surfaces.

5. Environmental Suitability: The robot is intended to be utilized in outside areas, which means it must be able to travel on roadways and pass over tiny barriers. Additionally, the robot's electronic equipment must be safeguarded.[10]

6. Uniqueness: To contribute to scientific study and progress, the robot must be unique.

The robot's hardware structure was created with the design criteria in mind. The anatomy of the robot is depicted in Figure. Every component of the robot is explained in this section in order of design progress.

2.2.2 Driving System

To begin, the robot's driving system was created. It's essentially a four-wheel-drive system. The robot's chassis, geared motor, Pololu geared ratio DC Motor, and motor driver make up the differential driving structure. To speed up the design and implementation process, pre-built chassis were employed. An acrylic sheet serves as the chassis. By adding gear to the chassis' gearbox, it was possible to spin the available drive shaft. To drive the shaft, a 150 rpm DC geared motor with a 70:1 gear ratio is mounted on a specially designed motor mount.

2.2.3 Power Module

A BMS was developed in order to fulfil the needs of the drive system. It is possible to divide the power system into three sections. A 4S charging circuit and charger, as well as the battery pack, are included in this group of components. It has forty-eight Lithium-Ion batteries with a voltage of 3.7 V. The battery management system keeps track of the voltages and temperatures of the batteries, as well as the amount of current drawn from the battery pack. An on/off switch, as well as switches for switching between charging and discharging modes, may be found on the robot's side panels.

2.2.4 Controller and Driver Board

1. Nvidia Jetson Nano (development board): The robot's brain is the mainboard. Higher-level control is provided by the Nvidia Jetson Nano. The communication between the various modules is handled via ROS. Its Nvidia Maxwell GPU (128 cores) enables deep neural network inference to be completed quickly. All other operations, such as the global planning algorithm, local planning algorithm, control algorithm, and so on, are executed by it. Control methods and sensor fusion algorithms will be implemented on the mainboard in the later stages of the project. The Jetson Nano wins in terms of GPU thanks to its 128-core Maxwell GPU running at 921 Mhz [10]. In comparison to the Jetson Nano, the Raspberry Pi 4 GPU is less powerful. This is extremely beneficial when running software that requires a greater level of computation.

2. Arduino Mega (Control Board): Arduino Mega is utilized for lower-level control. It manages all of the sensors and controls all four motors for driving and steering. It interfaces with the main processor (Nvidia Jetson Nano) via serial to obtain motor commands and publish sensor data.

3. Pololu Dual VNH5019 Motor Driver Shield: This shield makes it simple to use an Arduino or an Arduino-compatible board to operate our four high-power DC gear motors. Its twin VNH5019 motor drivers can deliver a constant 12 A (30 A peak) per motor or a continuous 24 A (60 A peak) to a single motor connected to both channels and operate from 5.5 to 24 V. These fantastic drivers also include current-sense feedback and can operate at ultrasonic PWM frequencies, making them even quieter. If the default Arduino pin mappings aren't handy, you can change them, and the motor driver control lines are separated down the left side of the shield for usage without an Arduino. This is why we chose not to use the L298N Motor drivers.[10]

4. 12V/20A Step-down Buck Converter: This adjustable constant current module has adjustable output voltage ranges from 1.2V to 36V and has a power rating of 300W and a current rating of 20A. The module has a wide range of current output, up to a maximum of 20 amps.

2.2.5 Sensors

1. RP-LIDAR: Sensors may be used for mapping and localization in a variety of ways. It was decided to go with a 2D perception scheme that is accurate, has widely accessible software support, includes error correction systems, and has a reasonable range without a significant error factor[11]. All of these factors led to the decision to map utilizing a LIDAR (Light Detection and Ranging) system. It's also less expensive than 3D lidar.

2. SparkFun IMU Sensor(MPU-9250): An inertial measuring unit (IMU) is a device that measures inertial (IMU). A 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer are included. This is useful for robot localization. Razor 9DoF (IMU Sensor) was disregarded because of its non- Compatibility with ROS (Robot Operating System).

The robot's hardware is designed at three different levels. The following are the levels:

2.2.6 System Level Design

(a) The robot chassis, motors, batteries, and control board are all found on this level. The body level has immovable parts that are specific to the robot platform.

(b) Control unit level: This is where you'll find the powerboard, mainboard, 5V DC bus, motor driver board, and distance sensors. This level is removable, allowing for customization. As long as the motor driver and battery are installed, the control unit level can be utilized on any other platform.

(c) Top Level: Sensors are installed on the rooftop level. The camera and LiDAR are set at this height to guarantee that the range is not obstructed. Additional navigation components and application-specific equipment can be added to this level at a later stage of the project.

2.3 Society, Economic, and Ethical considerations

Due to COVID-19, in our society, there is a growing importance of social distancing and isolating oneself it is important to have access to daily human requirements without any restrictions, hence an autonomous system would help people reach out to things they can not usually reach out to because of the required SOP's. Apart from this, people who would need special care in traversing around their environment would benefit from such an autonomous system as this would help them complete their tasks. We have an external industry connection with **Zaaviaa**, which is a company that provides digital solutions to everyday problems.

The team's core design and system specs are based on the target audience, which is predominantly individuals on campus. The robot's prototype is meant to be as little as possible, not too big or tall so that it does not become a hindrance or annoyance to pedestrians, and so that it can move around in congested areas more easily. Paths for such robots have been planned in several newly proposed smart cities. The robot is equipped with security measures such as real-time tracking and face recognition, ensuring that it cannot be stolen and that its contents are only accessible to the designated recipient. Because a database of users' facial traits is created for verification reasons, face recognition does raise some possible privacy problems.

2.4 Environment and Sustainability considerations

The robot's lightweight design, compactness, and adaptability were chosen with the power needs and the necessity to move effectively in mind, to avoid imposing any needless usage of electricity. While there is no defined measure for calculating the system's carbon footprint, our team is making comparisons to other robots of comparable size and scope in terms of materials and battery systems. The use of numerous layers of reinforced plastic sheets for the robot's body, for example, was examined to minimize weight and assure smooth movement without taxing the system's actuators or drawing more power than is required for the robot to operate. To encourage repeatability and scalability, the materials utilized are also inexpensive and widely available, needing no difficult-to-find parts. The majority of greenhouse gas emissions come from the vehicle, with robotics and automation accounting for less than 20-percent of a package's impact. The package's footprint is mostly determined by vehicle engine and fuel economy. According to the experts, switching to these robots and lowering the carbon intensity of the power they use might have the greatest impact on long-term delivery.

2.5 Technical Requirements

Figure 2.2 depicts a high-level system diagram of the system. The delivery hardware, perception module, flutter application, and QR code scanning are the primary components that make up the entire system.

The technical and functional requirements are explained as followed:

1. A 4 wheeled robot with controlled movement
   - Differential drive mechanism would be implemented but without the support of caster wheels. This would regularize the stability along with much more torque induced by the wheels. This would enable the robot to handle a much larger load at once.
   - Differential drive consists of 2 or 4 drive wheels mounted on a common axis, and each wheel can independently be driven either forward or backward.
2. The robot should be able to detect sudden obstacles and avoid it
   - Since implementing raspberry pi reduces the computation per unit time, jetson nano is used to ensure that a lot more frames from the camera are pro-

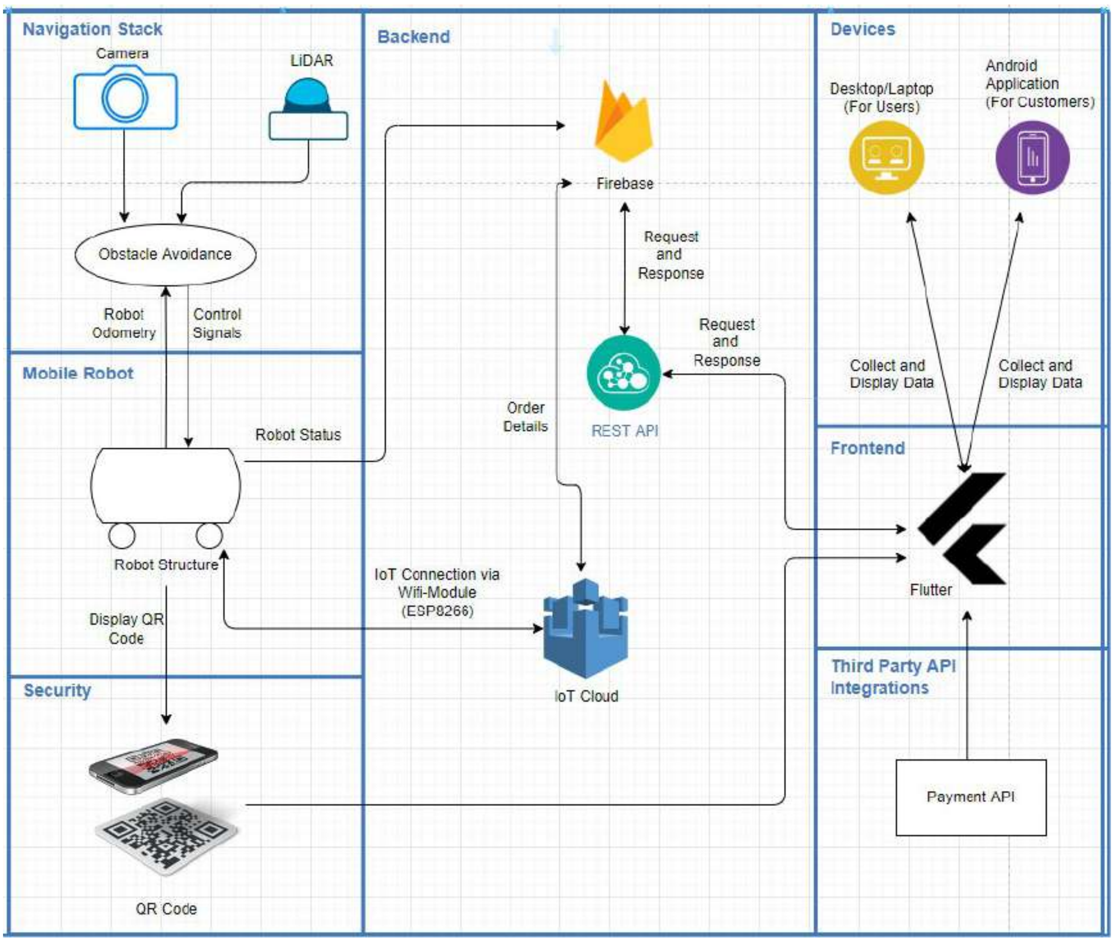


Figure 2.2: High Level System Diagram

cessed at a time and therefore, much faster object detection can be implemented with higher accuracy of decision making.

3. The application back-end should be connected to the robot through IoT and should have fast M2M communication

    - Since the robot will be controlled through an application, it is necessary that the back-end of the application is connected via wireless connection with the robot so that as soon as an order is placed, the robot is given its task automatically without any human intervention

4. The robot should be able to move at a considerable speed (at least 2 cm per second).

    - As discussed, the previous robotic projects were not able to move at a feasible speed and since our application is based on making deliveries, the robot must move comparatively faster than them so that our application is catered appropriately

- For that we are using 4 wheel drive instead of 2 or 3 wheels or castor wheels, to move the robot at a considerable speed with appropriate turning.

5. LIDAR-Camera fusion for obstacle avoidance & path planning algorithm

   - Major novelty in this work would comee from the LiDAR-camera fusion as it proves to be much more efficient and reliable than a simple LiDAR/camera system. Implementing this would allow us to have two decision-making entities at once and thus, a much more reliable decision would be made by the autonomous system
   - This again will increase the efficiency of avoiding obstacles as the decision-making time will decrease.
   - However, the main prototype won't have the camera integrated, it is a plan devised for any future work that happens on this robot.
   - We process the camera image separately, get an output, and verify against the processed LI-DAR output or vice-versa. This is a very useful fusion method since it can help in increasing the reliability of a system.

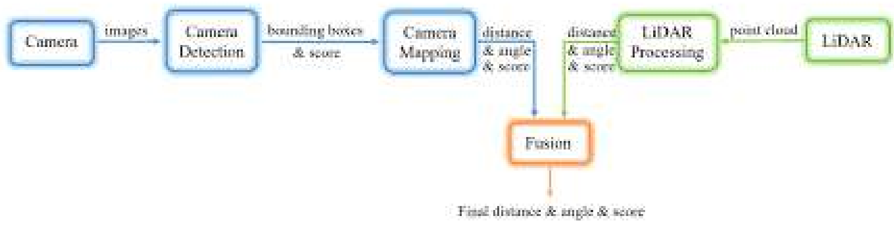


Figure 2.3: Block diagram of Camera and LI-DAR fusion[12]

6. A user friendly application dedicated for placing orders

- Since the application would be used by general public, it is necessary that it is easy for the users to understand and operate with (as it would need to know user's location) and should be pleasing for the customers to operate on

7. Integrating smart security system on the robot

    - Apart from all the technicalities, industry 4.0 demands a solid security system which is end-to-end since machines are at work for most of the time, it is required to keep the food items safe inside the robot so that during the delivery, no theft can be done.

Following highlight the technical requirements for each team that have to be full filled:

2.5.1 EE Aspects

1. Design of an optimum robot perception system for the work at hand- Due to its precision and dependability, a LIDAR is the primary component that is being used.

2. Locomotion control system design and implementation - odometry for speed tracking and PID control for smooth movement.

3. Communication between embedded systems and servers through IoT — This necessitates consideration of specialized protocols and wireless communication methods (e.g. Remote Connection).

4. Hardware design and implementation: CAD models, base design, sensor attachments, payload securing attachment, and other hardware attachment design and analysis are all part of the hardware building process.

5. Fusion of Lidar and Camera data points is necessary as well for the robot to be robust in obstacle avoidance and path planning

2.5.2 CS Aspects

1. Implementation of user-server communication for the robot and mobile app to transmit and receive data and analyses.

2. Development of an app for the delivery system as well as a web-based admin interface for operations monitoring and management.

3. Implementation of a database to keep track of deliveries and manage data for system users.

2.5.3 EE/CS Cross-listed Aspects

1. Robot Operating System based mapping

2. Setting up an IoT cloud server

3. Integrating the map on the robot

4. Interoperability between the application and the robot (M2M communication)

5. Integrating any sort of security system

2.6 Solution Statement

We have worked upon a project intending to provide efficient and contact-less deliveries within the premise of our university through orders placed by the customers on our designed application. This will allow them to have a much more convenient way of procuring items within the premise of our university (from the cafeteria in our case), also being much more efficient in their daily routine, mitigating the risk factors involved in human to human contact (especially avoiding the spread of diseases), and providing an alternative for people who can not perform self-service. Moreover, this idea was pitched by the Zaaviaa industry to us as they wanted to have a proof of concept for a robot that can work under a miniature industry 4.0 framework as it would allow them to deploy such robots on different locations for different applications e.g., in hospitals where they can be used to contact people with autism or deliver certain medicines. Therefore, most of the significance of this project is based on making such a framework with minimal cost, procurable equipment within Pakistan, and flexibility that it can sustain the idea pitched by Zaaviaa that can prove to have a very positive impact on society.

Considering the existing solutions, we have highlighted their weaknesses in the table below. Considering the weaknesses above, our robot is mitigating all of the above problems which will be discussed in detail later and providing a much better form of solution. Moreover, The electronic components and microcontroller systems are chosen and used with the availability and cost margin in consideration. ROS is adopted so there is a possibility of future development work with a larger technical resource pool and publicly available community help. An open source system also reduces the expense of developing, customising, and testing our system.

Robot Name	Weaknesses
Locus Robotics	Specific to warehouse management Available to limited countries Very expensive
Keenon Robot	Very expensive and hard to procure in Pakistan High maintenance cost - requires experienced labor
Robotnik	Very expensive for small businesses Hard to maintain and operate on
SwiftBot	Only LiDAR based/Unreliable obstacle detection Utilizes Raspberry Pi so unable to avoid sudden obstacles 2 wheeled differential drive thus slower speed Limited flexibility. (Not flexible for multiple applications)
Starship Delivery Robot	Specific to deliveries for themselves Highly overloaded with sensor devices thus costly Only available in San Francisco
R2 Autonomous Robot	Operating in limited countries Not flexible for different applications Huge design and very expensive Limited flexibility. (Not flexible for multiple applications)
Zhen Robotics	High cost Limited to China Low flexibility (only delivery based)

CHAPTER 3

DESIGN DETAILS

3.1 Solution Overview

The overall system block diagram of this project is shown in Figure 3.1. It highlights the main components that are involved in our project. We can see that here, we three main modular subsystems namely the locomotion module, ROSCORE, and obstacle avoidance module. Through brainstorming several representations, we believe that this is an accurate representation of how modularity is achieved in our project as these subsystems are changeable within themselves given that the other subsystems communicate with same inputs and outputs.

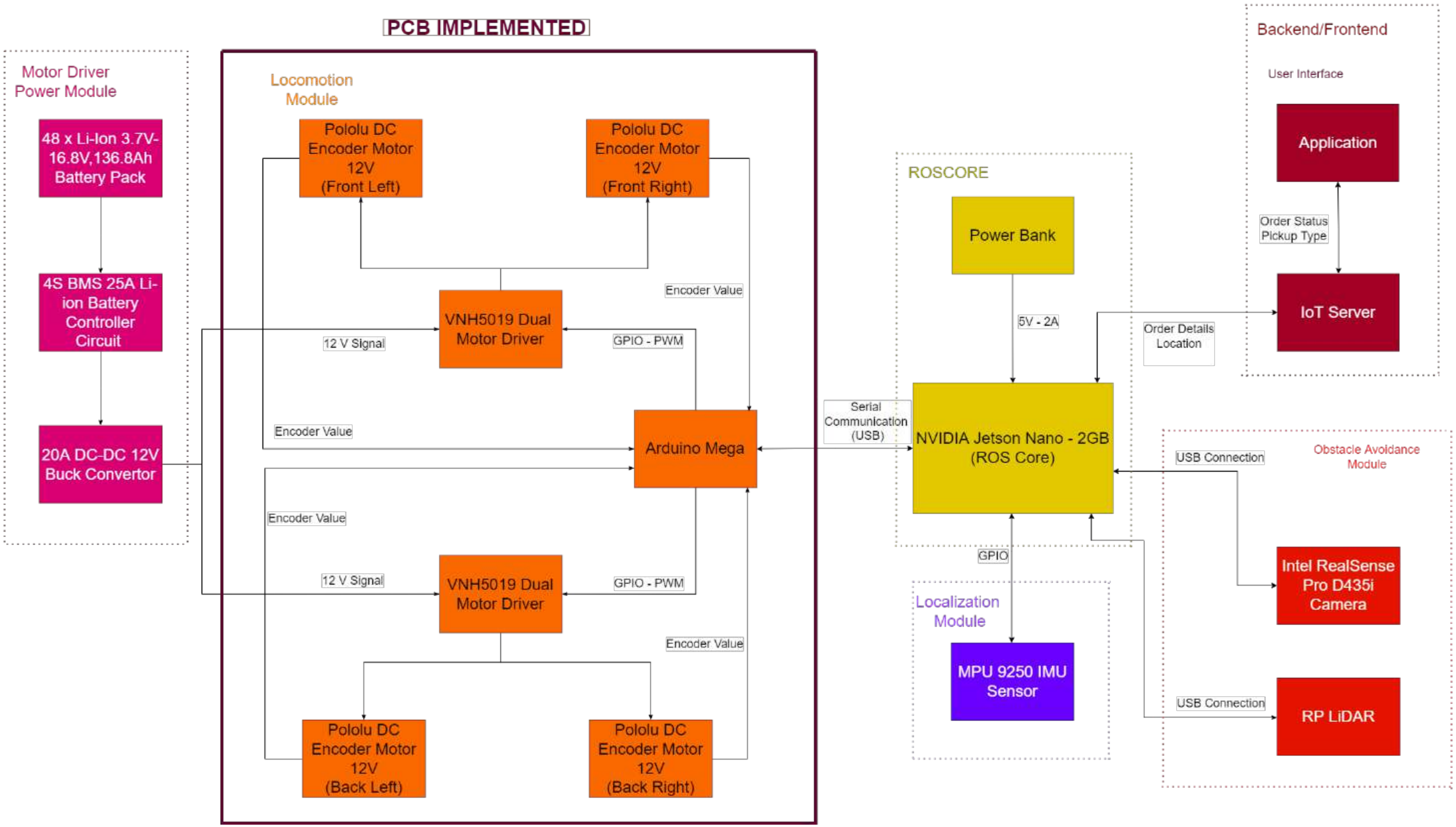


Figure 3.1: System Block Diagram

3.2 Mechanical Structure of Robot

3.2.1 4-Wheel Differential Drive

A differential drive is a two-wheeled drive system having separate actuators for each wheel. A caster wheel is frequently used as a non-driven wheel. Unfortunately, castor wheels can cause problems if the robot reverses direction, and they also do not contribute to providing force to the robot's structure, so we used a four-wheel differential drive, in which the differential wheels (left and right) are mounted on the front and back sides of the robot, simulating a four-wheel drive with differential drive turning capabilities. Figure 3.2 shows how locomotion is achieved given movements of each of the wheels in the four-wheel differential drive of a robot. We could have used other op-

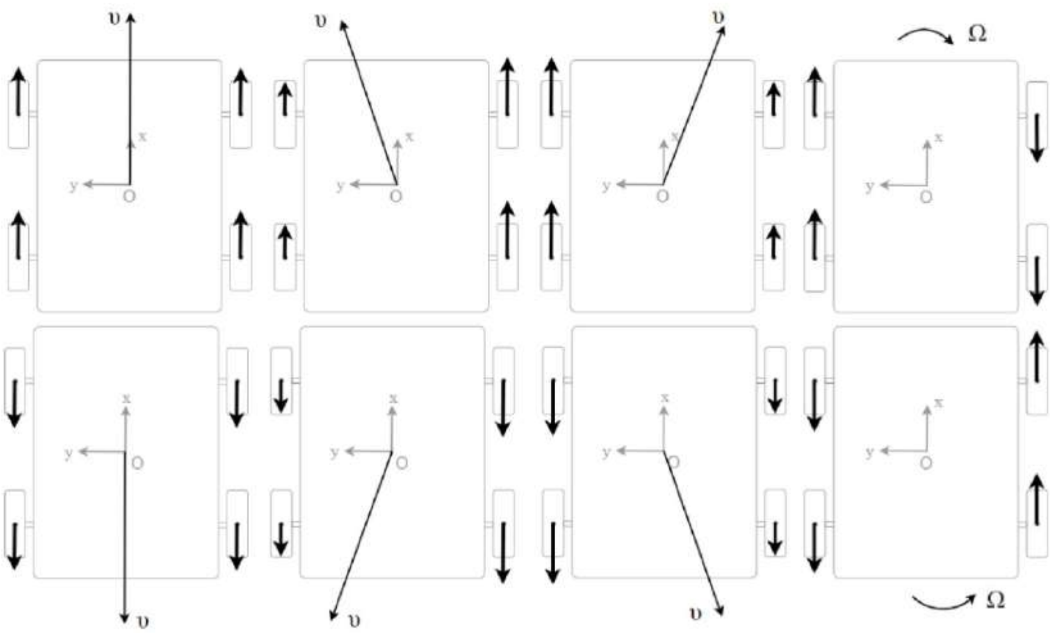


Figure 3.2: Locomotion of a four wheel differential drive [3]

tions in terms of employing a turning mechanism e.g., Ackermann steering however, we had already worked upon differential drive in the past (in microcontrollers project), and thus, was convenient for us to make it again moreover, in our course 'Mobile Robotics', we also studied in depth kinematics, dynamics, and control of a differential drive robot thus, we were capable of tuning the associated parameters in the ROS packages to employ it on four-wheel differential drive whereas, in Ackermann steering, much of our time would have lost in learning the mathematics behind all these concepts.

3.2.2 Structure of PoC-1

Our prototype consisted of four different levels. The first level is a thick 6mm acrylic sheet that provides the support to mount the L-Brackets of the motors and to sustain most of the weight of the robot's hardware. For now, we have poorly considered the weight of the payload attached to this since it is the structure of our first PoC and were advised by our external supervisor as well to not worry much about the structure for now. However, we will be transforming this further towards a much stabler structure for our final prototype as we have learned much from tinkering around with our current structure. Nevertheless, the other three levels are of 3mm thin acrylic sheet as they were not supposed to handle the high amount of weights. The second level consisted of motor drivers, buck convertor, and Arduino mega and thus, had most of the wiring components. On the third level, we have our jetson nano resting alongside the power bank which is being used to power it. It is on the mid-level to provide convenient wiring towards the sensors that are attached either below or above a level. Finally, we have our topmost fourth level where the LiDAR and IMU sensors are mounted and the reason for them to be on the top is to provide room for them to scan properly without any hindrance in between and to avoid any electrical interference between their signals. Figure 3.3 aids in visualizing all levels at once. Furthermore, the number of sensors on the robot alongside the four wheels with four different dc encoder motors can be visualized from Figure 3.4. Here, we can notice that there is no room for the payload however, we could identify another level between levels 3 and 4 that could serve as a payload space since it would allow no obstruction to the sensors at the top and would be away from the circuitry as well.

The acrylic sheet on the first floor was not available in the market or university and we also had to employ motor L-brackets onto it so precise cutting was required there. Hence, we planned on drawing its outlay on software called Corel Draw as the local

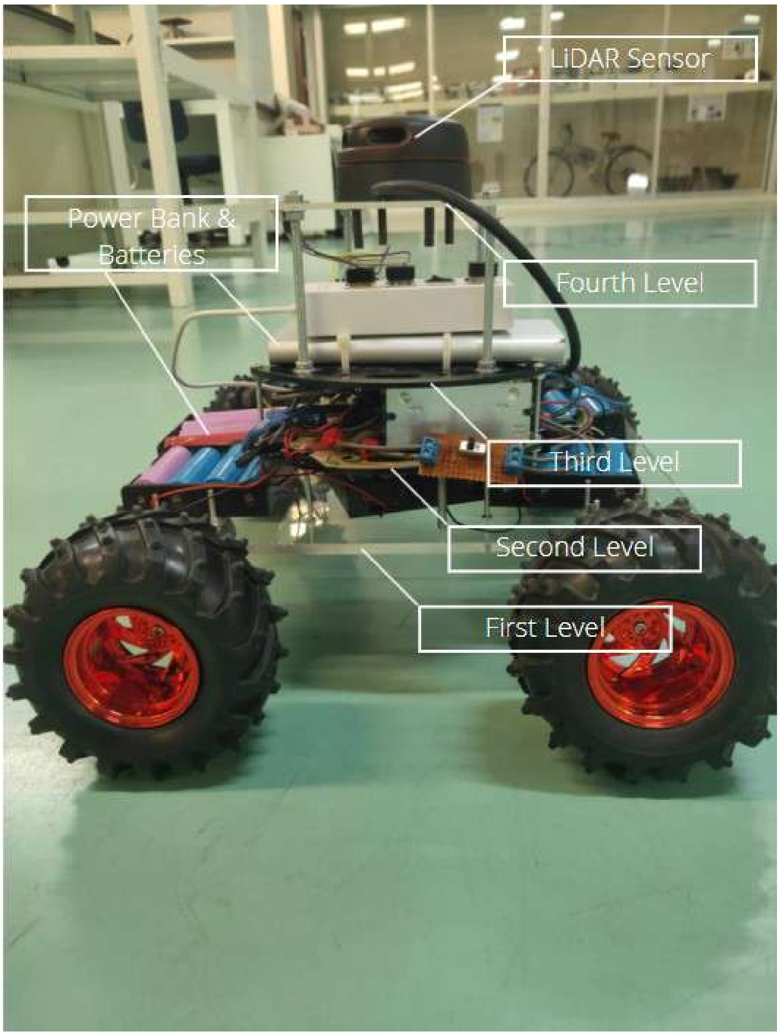


Figure 3.3: Side view of our first prototype

acrylic sheet cutting markets here support the designs drawn on this software and they would eventually help us in getting the desired cut on the thick acrylic. Hence, a design was made as shown in Figure 3.5 that represented the design of the bottom layer. For now, the design is not supportive of stacking multiple similar layers as in the future, we might use a similar design but with such capabilities to provide much more room in the robot and also accommodate space for our payload. Hence, this serves to be useful for our structure in PoC-2 as well. However, for now, we had embarked small holes onto it to attach a small level on top of it to deliver the requirements of the structure of PoC-1. We have also considered the parameters that are involved in the ROS packages (discussed later) related to the mechanical structure of the robot and hence, would be adjusted once we move onto our final product. The following points highlight the

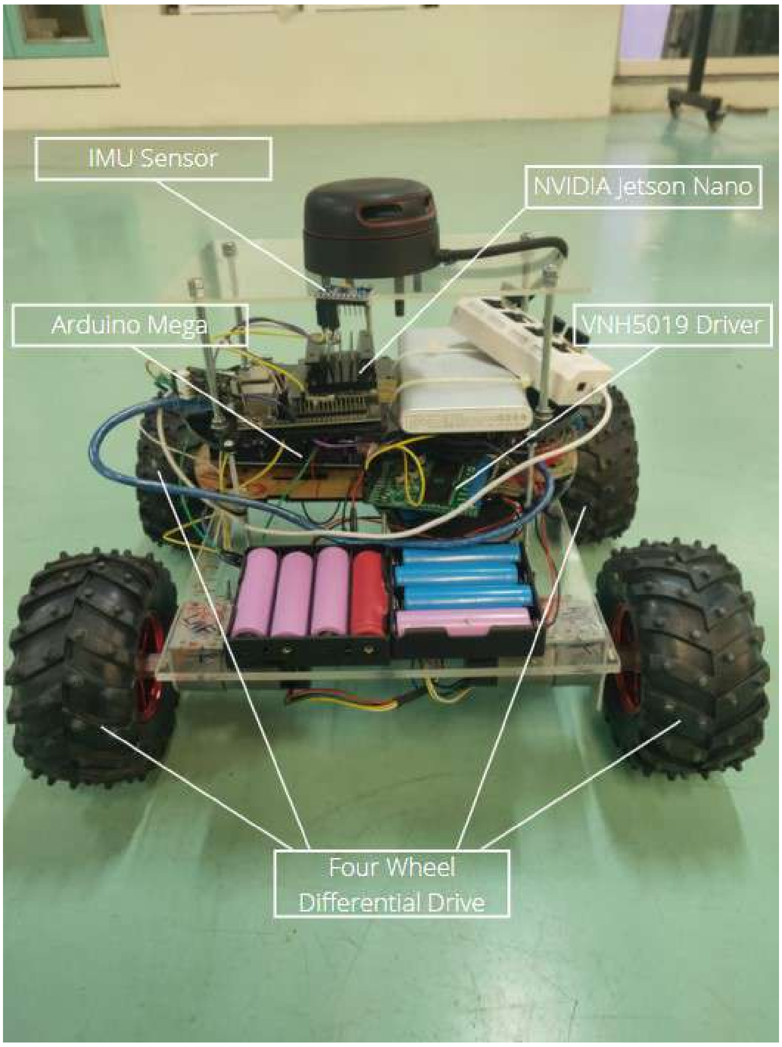


Figure 3.4: Front view of our first prototype

parameters that would be adjusted during the shift.

1. Base Width and Length
2. Wheel Radius
3. Height of LiDAR mount
4. Height of IMU sensor mount
5. Weight of the robot

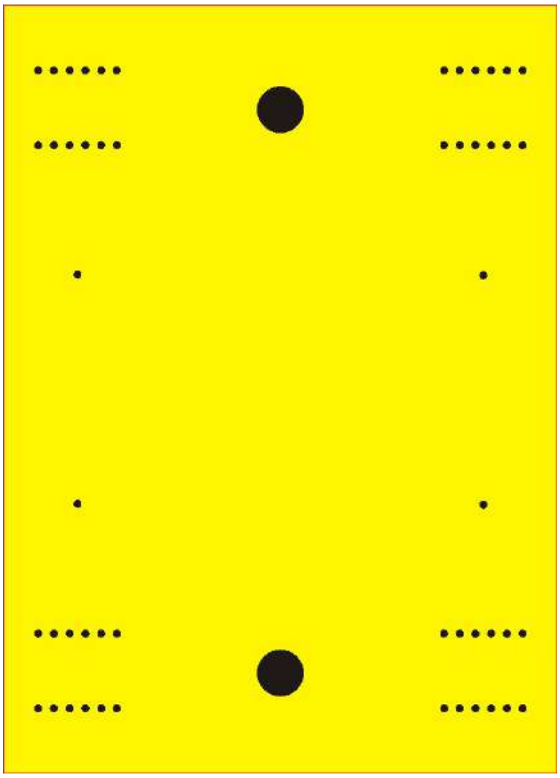


Figure 3.5: Corel Draw of level 1 of structure

3.2.3 Structure of PoC-2

Our final prototype is made up of three different levels. An 8mm thick piece of acrylic is the first level. It is used as a support for the L-Brackets of the motors and as a way to hold the majority of the weight of the robot. When we connect a payload this time, we have taken into account the weight of the payload. The third level is about where to put the payload. When we looked at how the structure was made, we found that it was more stable and well-defined when we looked at the parameters and measurements. The second level was made for motor drivers, a PCB, an Arduino mega, and a Jetson Nano. It had a lot of wire parts at first, but now that the PCB and bus cables have been used. On top of the third level, we have the LIDAR such that it have enough space to scan without any obstructions in the way, and also to make sure there isn't any signal interference between the structure and the LIDAR.

First level

The motors along with the L-brackets have been mounted on top of the acrylic sheet to provide stability and to avoid bending as well, also we have used a thicker acrylic sheet to support the weight of the BMS and the motors as well. It is illustrated in Figure 3.6 based on a CAD model we formed. The tyres are similar to those that were used in POC-1, as we couldn't find bigger tyres in the local markets.

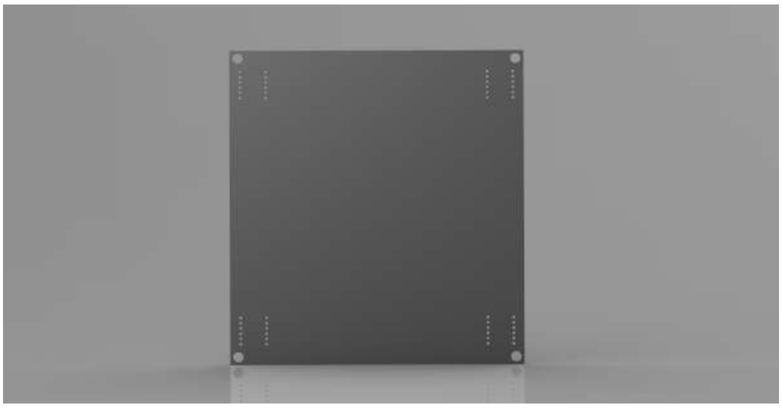


Figure 3.6: Base of Final Prototype

Second level

The measurements and the hole sizes were illustrated through CREO again, which was then provided for laser cutting. This level as illustrated in Figure 3.7 comprises of all kinds of development boards which are being used to drive and navigate the robot.

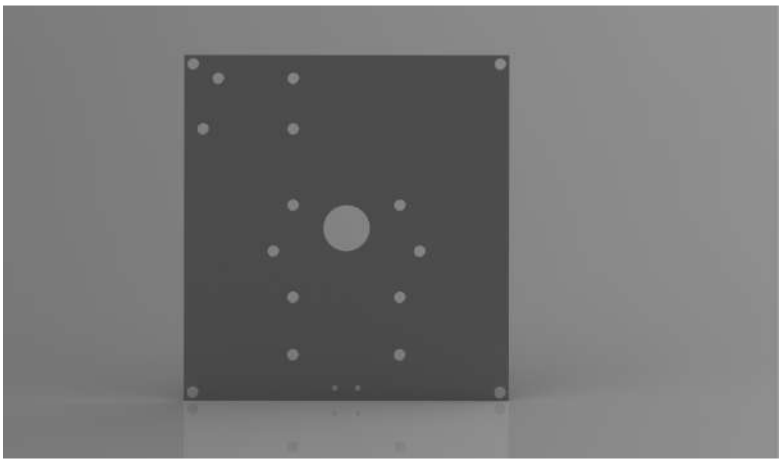


Figure 3.7: Middle Base of Final Prototype

Third level

The measurements and the hole sizes were illustrated through CAD model which was then provided for laser cutting as shown in Figure 3.8, the bottom base, top base and the sides were made up of 3mm acrylic sheet.On top of the box the LiDAR is mounted and inside the box we have the weight sensor attached to measure the payload that will be placed inside it.

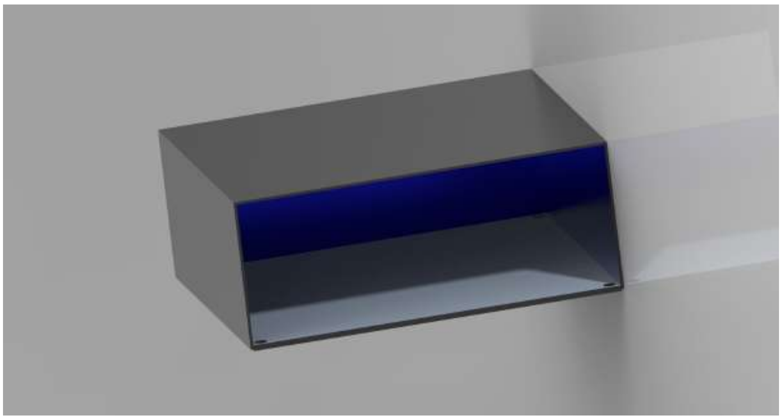


Figure 3.8: Top Base of Final Prototype

3.3 Hardware Components

3.3.1 Arduino Mega

The ATmega2560 is the basis for the Arduino Mega 2560 microcontroller board. It contains 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power connector, an ICSP header, and a reset button. For our application, it will be used to provide PWM signals to the motor drivers and retrieve wheel encoder values through digital pins. It can also be used to perform interrupt operations to carry out service routines within the program as well. Table 3.1 shows the specification of this microcontroller.

Table 3.1: Arduino Mega Specifications

Type	Specification
Operating Voltage	5V
Digital I/O Pins	54
Analog Input Pins	16
Flash Memory	256 KB
EEPROM	4KB
Weight	37g

3.3.2 NVIDIA Jetson Nano

The NVIDIA® Jetson NanoTM Developer Kit is a compact, powerful computer that allows you to run several neural networks in parallel for image classification, object identification, segmentation, and voice processing applications. It is supposed to be the brain of our robot since it has the capability of performing tasks at higher speeds and also, supports Linux software which is essential to integrate ROS. It initializes the ROSCORE and also handles multiple packages that are used within ROS. It has multiple connectivity slots to handle other sensors and microcontrollers as well. e.g., LiDAR and Arduino have connected through USB and IMU sensors through GPIO pins. Alongside, the device can be WiFi capable through the intervention of a dongle as well. Therefore, it is a perfect developing kit to power our project. Table 3.2 shows the specifications of this development kit.

Table 3.2: NVIDIA Jetson Nano Specifications

Type	**Specification**
GPU	128-core NVIDIA Maxwell architecture-based GPU
CPU	Quad-core ARM A57
Memory	2 GB 64-bit LPDDR4; 25.6 gigabytes per second
Storage	64 GB (External SD Card)
Pins	40-pin header (GPIO, I2C, I2S, SPI, UART)
Dimensions	100 mm x 80 mm x 29 mm
USB Connectivity	1x USB 3.0 ,2x USB 2.0 Type A, USB 2.0

3.3.3 RP LIDAR A2

The RPLIDAR A2 is a modern, 360-degree 2D LIDAR that can be used indoors. Because of its fast rotation speed, each RPLIDAR A2 can capture up to 8000 samples of laser ranging per second. Within a 12-meter range, the onboard device can execute 2D 360° scans (18m with a bit of firmware adjustment). The 2D point cloud data collected may also be utilized for mapping, localization, and object/environment modeling.

The RPLIDAR A2 is made up of a range scanner core and mechanical powering components that cause the core to spin at a rapid rate. When it's working properly, the scanner rotates and scans clockwise, enabling you to acquire range scan data via the RPLIDAR's communication interface and control the rotation motor's start, stop, and rotation speed via PWM. The RPLIDAR A2 360° Laser Scanner's standard scanning frequency is 10hz (600rpm), with an actual scanning frequency that may be easily altered within the 5-15hz range depending on the needs of users. The RPLIDAR A2 uses a SLAMTEC-developed laser triangulation measuring method, which allows the RPLIDAR A2 to work well in a variety of indoor and outdoor conditions without direct sunlight exposure.

This device will be used to map an indoor environment and in our case, that would be

Tapal Cafeteria and also, it will help us in determining any dynamic obstacles that needs to be avoided in the navigation stack. Table 3.3 shows the specifications of RP LIDA used in our project.

Table 3.3: RP LIDAR A2 Specifications

Type	**Specification**
System Voltage	5V
System Current	450-600 mA
Power Consumption	2.25-3 W
Output	UART Serial (3.3 Voltage Level)
Angular Range	360 degrees in 2D plane
Range	12-16 meters
Rotational Speed	5-15 Hz

3.3.4 VHH-5019 Dual Motor Driver

VNH5019 motor drivers can supply a constant 12 A (30 A peak) per motor or a continuous 24 A (60 A peak) to a single motor connected to both channels and operate between 5.5 and 24 V. These fantastic drivers also include current-sense feedback and can operate at ultrasonic PWM frequencies, making them even quieter. If the default Arduino pin mappings aren't handy, you may change them, and the motor driver control lines are separated off down the left side of the shield for usage without an Arduino. The VNH5019 Dual Motor Driver is employed because of its safety features as well as its ability to function as a regular motor driver. It has reverse-voltage protection up to -16 V, Undervoltage and overvoltage shutdown, high-side and low-side thermal shutdown, short-to-ground and short-to-VCC protection, and can withstand input voltages of up to 41 V. These drivers would be used to operate our motors and since we will be having 4 different motors, we will be using two of these motor drivers. Table 3.4 shows the specifications of this motor driver.

Table 3.4: VNH-5019 Motor Driver Specifications

Type	**Specification**
Operating Voltage	5.5 - 24 V
MOSFET on-resistance (per leg)	18 m-ohm typ
Max PWM frequency	20 kHz
Over-voltage shutoff	24 V min. / 27 V typ
Logic input high threshold	2.1 V min
Current for infinite run time	12 A

3.3.5 Polulu DC Encoder Motor

These encoder motors are linked to the Arduino through the motor driver and are mounted on the robot's wheels. Encoder motors assist in obtaining position by converting the shaft rotations within the motor into digital signals that may subsequently be sent to a microcontroller, such as an Arduino, to determine the robot's exact location. Each motor has six wires, four of which are used to power the motors and assign rotational direction, while the other two are used to provide position signals to the microcontroller. This motor was procured from the university and hence, was used as it saved a big expense for us however, we believe that if our budget was flexible, we could have taken brushless motors or motors with high gear ratios. Table 3.5 shows the specifications of this motor.

Table 3.5: Pololu DC Encoder Motor Specifications

Type	**Specification**
Rated Voltage	12 V
Stall Current	5.5 A
No-Load Current	0.2 A
Gear Ratio	70:1
No-Load Speed	150 RPM
Max Power	10 W

3.3.6 Power System & BMS

To prevent battery failures, many types of BMSs are employed. A battery monitoring system, for example, captures essential operating characteristics such as voltage, current, and the internal temperature of the battery, as well as the ambient temperature while charging and discharging. If any of the parameters exceed the safety zone's level, the system delivers inputs to the protective devices, causing the monitoring circuits to create warnings and even disconnect the battery from the load or charger.
As a result, battery monitoring and protection systems, a system that maintains the battery ready to give full power when needed, and a system that can increase the battery's life should all be included in the BMS in this sort of application. Systems that govern the charging regime and those that handle thermal concerns should be included in the BMS.

3.3.7 What is BMS?

According to the definition, the primary tasks of the BMS are the same as their aims. Although different types of BMS have varied goals, the average BMS has three:

- It protects the battery cells from damage and abuse;
- It maximizes battery life
- It ensures that the battery is always ready to use.

3.3.8 Functions Of BMS

Discharging control

A BMS's primary objective is to protect the battery from running outside its safe operating range. During discharging, the BMS must safeguard the cell from all potential dangers. Otherwise, the cell could be able to act outside of its bounds.

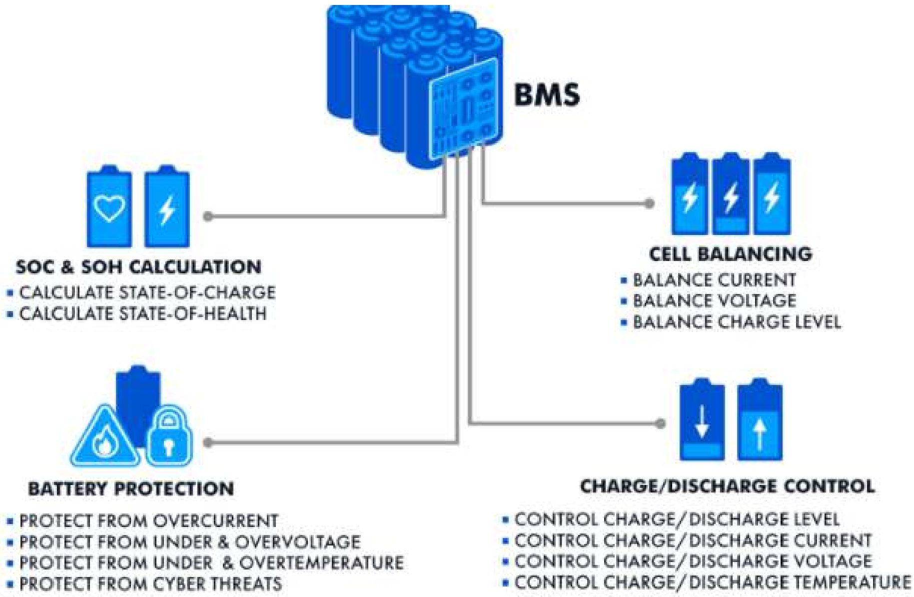


Figure 3.9: BMS Functionalities

Charging control

Incorrect charging damages batteries more commonly than any other factor. As a result, charging regulation is a critical component of the BMS. The constant current – constant voltage (CC-CV) charging method is used to charge lithium-ion batteries in two stages. The charger delivers a constant current that raises the battery voltage during the first charging stage (the constant current stage). The constant voltage (CV) stage begins when the battery voltage reaches a consistent value and the battery is nearly complete. The charger maintains a constant voltage until the battery is fully charged as the battery current decays exponentially as illustrated in Figure 3.9.

State-of-Charge

Determination The BMS keeps track of the battery's charge level, for example (SOC). The SOC might alert the user and manage the charging and draining of the battery. Direct measurement, coulomb counting, and a combination of the two procedures are the three ways of measuring SOC. Because the battery voltage declines more or less linearly over the discharging cycle of the battery, a voltmeter might be used to measure the SOC directly. The current moving into or out of a battery is incorporated in the coulomb-counting method to create the relative value of its charge. This is akin to counting the currency entering and exiting a bank account to establish the account's relative balance. Furthermore, the two approaches might be merged. The voltmeter might monitor the battery voltage and calibrate the SOC when the actual charge approaches either end. Meanwhile, the battery current might be used to calculate the relative charge moving into and out of the battery.

State-of-Health

When compared to a new battery, the state of health (SOH) is a measurement that represents the general condition of the battery and its ability to achieve the given performance. Any metric that varies considerably with age, such as cell impedance or conductance, might reflect the cell's SOH. In reality, the SOH may be calculated with only a single measurement of cell impedance or conductance.

Cell Balancing

Cell balancing compensates for weaker cells by equalizing the charge on all cells in the chain to improve the total battery life. With each charge-discharge cycle in a chain of multi-cell batteries, minor discrepancies between the cells owing to manufacturing tolerances or operating circumstances tend to be exacerbated. Weak cells may be overstressed when charging and become weaker until they fail, leading the battery to fail prematurely. The BMS may incorporate one of three cell balancing schemes to equalize the cells and prevent individual cells from becoming overstressed to provide a dynamic solution to this problem while taking into account the age and operating conditions of the cells: the active balancing scheme, the passive balancing scheme, and the charge shunting scheme. The charge from the more essential cells is withdrawn and supplied to the weaker cells in static cell balancing. Dissipative methods are employed in passive balancing to locate the cells with the highest charge in the pack, as evidenced by higher cell voltages. The extra energy is then evacuated through a bypass resistor until the voltage or charge on the weaker cells matches the voltage on the more potent cells. The voltage on all cells would be leveled upward to the rated voltage of a suitable cell in charge shunting. When the cell reaches its rated voltage, the current will skip the fully charged cells and charge the weaker cells until they achieve full voltage.

3.3.9 BMS Topology

A battery management system's components can be configured in various ways. Topologies, or organizational structures, can be centralized, dispersed, or modular.

Centralized BMS topology

A single BMS printed circuit board (PCB) with a control unit regulates all cells in a battery using various communication channels in a centralized architecture. A BMS with this configuration is a big, inflexible, yet cost-effective option.

Distributed BMS topology

Every battery cell has its own BMS PCB, and the entire battery is connected to a single channel via a control unit. The daisy chain is a distributed topology variant designed for systems with low fault tolerance requirements. Because of the availability of electronics, distributed BMSs are simple to set up but expensive to maintain.

Modular BMS topology

A modular BMS is a hybrid of the two topologies mentioned above. This configuration is often referred to as a decentralized, star, or master and slaves topology. There are numerous linked control units (slaves) in a battery, and each supervises a set of cells. The slaves are connected to the primary control unit, or master, in charge of the battery's integrity and safety. A modular BMS topology might strike a compromise between cost and design complexity.

Illustration of the topolgies is shown in Figure 3.10

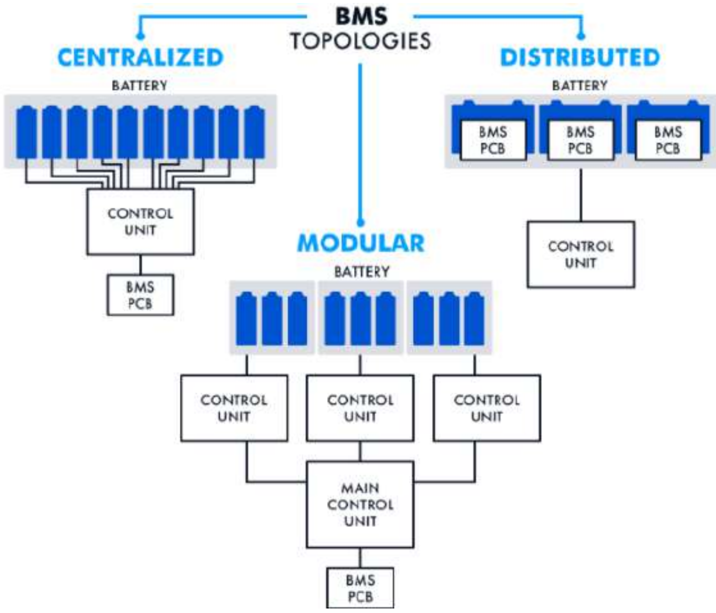


Figure 3.10: BMS Topologies

3.3.10 Calculations for BMS

The BMS we are using is a 16.8V and 25A. Assuming if one Li-ion battery has the voltage capacity of 4.2V, this will give us the cells we need in series. Mathematically this can be expressed as:

$$\textit{Number of series connection} = \frac{16.8}{4.2} = 4 \tag{3.1}$$

This mean 4 rows of cells will be in series (4S) which is equivalent to 48 cells in total and 144Ah of capacity. Since cumulative load taken by the motors will be 20A at max torque, the BMS can last upto 7 hours of operation when fully charged. Table 3.6 shows the specification for the Centralized BMS.

Table 3.6: Li-ion cell Specifications

Type	**Specification**
Nominal Voltage	16.8 V
Nominal capacity	144Ah
Max Current	25 A
Charging time	6 hours

3.3.11 Power Bank

We have also employed a Huawei 20000mah power bank to supply power to the Nvidia Jetson Nano. Since the nano is providing power to multiple sensors in the robot e.g., Arduino mega, LiDAR, IMU, etc. it needs a separate power back up where it is capable of providing power to the rest of the components along with itself for a longer time. Therefore, a 20000 mAh power bank was ideal for our use case as it does not increase the cost by a higher margin and also delivers enough. Table 3.7 shows the specifications of our power bank.

Table 3.7: Huawei Power Bank Specifications

Type	**Specification**
Battery Capacity	20000mAh
Input	5V-2A
Output	5V-2A to 5V-3.4A (Ideal for Nano)
Input port	USB-C
Output port	2x USB-A

3.3.12 20A Buck Converter

The buck converter was used to step down the voltage from the batteries as a result of regulating it at a fixed level for better control of the motors, and it was also useful in increasing the DC output of the cells. This method proved to be very useful in our case since it allows outputting much more current and therefore, the motors were able to sustain a high amount of torque in their operation.

From 1.2V to 36V, the 300W 20A DC-DC Step-Down Adjustable Constant Current Module can be used. There is a wide range of output currents available from the module up to 20 A. High-power applications can be run continuously with the heat sink.

In addition, the Module is built with an integrated benchmark chip, a high-precision current sensing resistor that ensures stable constant current (when the temperature ranges from 20°C to 100°C constant current 1A, the temperature drift is less than 1 percent), and thus it is ideal for LED driver applications. 20A is the maximum output current that can be achieved with this module (for continuous use it is recommended up to 15A with an additional cooling fan).

Six high-frequency capacitances are used to stabilise the output voltage, resulting in a smaller output ripple. The use of two heat sinks makes it simple and quick to remove heat from the system. An independent heat sink for the MOS Schottky diode, which is good at dissipating heat and won't interfere with each other. Improved productivity and reduced fever are also achieved by the utilisation of a Sendust Core of a larger size and dual pure copper wiring.

High-accuracy voltage and current regulation is provided by 3296 multiturn potentiometers included inside the module. The adjustable voltage and current make it easy to utilise for a variety of applications, including battery charging, LED driver power supply, and vehicle power supply, making this module a popular choice for our manufacturer. Table 3.8 shows the specifications of our buck converter.

Table 3.8: 300W 20A DC-DC Buck Converter Specifications

Type	Specification
Input Voltage	6-40 VDC
Output Voltage	1.25 to 36 V
Output Current	20 A
Output Power	200W (with natural cooling)
Conversion Efficiency	97%
Short Circuit/Reverse Connect Protection	Yes
Switching frequency	180KHZ

3.3.13 MPU-9250 IMU Sensor

An inertial measurement unit (IMU) is an electronic device that uses a combination of accelerometers, gyroscopes, and magnetometers to measure and report a body's specific force, angular rate, and orientation. Sensors having 9 degrees of freedom (DoF) are usually capable of providing robot orientation. The MPU 9250 is one of the most affordable 9DoF IMU sensors, and it's also compatible with ROS thanks to existing packages. As a result, we utilized it in our project to aid in the improved localization of the robot. Table 3.9 shows the specifications of the MPU-9250 IMU sensor.

Table 3.9: MPU-9250 IMU Sensor Specifications

Type	Specification
Max Voltage Supply	3.6 V
Max Operating Temperature	86 °C
Number of Pins	24
Dimensions	1x3.1x3.1 mm
Interface	I2C - SPI%
Mount	Surface Mount

3.3.14 TP Link TL-WN725 Wi-Fi Dongle

This WiFi dongle was necessary to enable Jetson Nano to have wifi connection on it as then, we were able to remotely connect the Jetson nano and make our robot purely remote. It also gives us the functionality to connect with IoT nodes and therefore, to the backend of the application server. Hence, proves to be a necessary device in terms of robot's connectivity. Table 3.10 shows the specifications of the Wi-Fi dongle.

Table 3.10: TP Link TL-WN725 Wi-Fi Dongle Specifications

Type	Specification
Connectivity	USB
Wireless Transmission	150Mbps
Security	WEP, WPA, PA2/WPA-PSK/WPA2-PSK
Support	Windows 10/8.1/8/7/XP, Mac OS X, Linux

3.4 Weight Instrument

A 5KG load cell is used that will measure force induced by the weight of the delivery item within the robot's structure and indicate the weight of it on a VDU. Since the robot is constraint towards delivering food items within 5KG or otherwise, the navigation of the robot is compromised, it is necessary to indicate the user of what weight it is carrying right now.

3.4.1 Sensor Details

The load cell sensor utilised in this project has a maximum weight limit of 5 kg. It's a 4-wire strain gauge sensor with a full bridge arrangement. The load cell relies on the idea that the bending moment in the bar is proportional to the applied force. This can convert pressure up to 5kg into an electrical signal. Each load cell can detect variations in electrical resistance in reaction to and proportional to the strain applied to the bar.

The physical and schematic model of the sensor is shown in figure 3.11 and figure C.3.

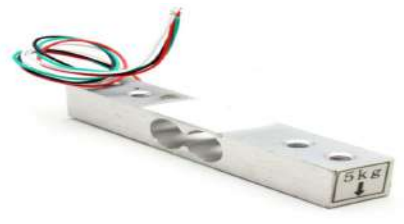


Figure 3.11: Load Cell - Physical

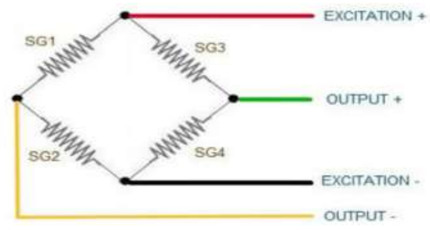


Figure 3.12: Load cell wiring convention

Here, we can see that the principle behind the sensor is of resistance measurement and since we know that with the applied force, the physical dimensions of the strain gauge will change and therefore, we will notice an unbalanced bridge and the magnitude of that is used to measure the applied force/weight. The changes caused to the load cell from the applied pressure can be seen in figure C.2 from the appendix. But things are not always that easy! The change in resistance is very small and which is, for most device, undetectable. Therefore, we will need to condition the signal such that is is feasible for the device in the next step to easily process the signal.

The measured force from the load cell is given by the formula:

$$Force_{measured} = A * (V_{measured}) + B \tag{3.2}$$

$$V_{measured} = V_{in} \left(\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2} \right) \tag{3.3}$$

The values of A and B in above equation are usually determined by the placement of load cell on the platform and different other ambient conditions i.e., screws, weight placement, and input voltage however, this equation indicates that the behavior with change of force is always going to be linear as the voltage change in resistance is linear with change in applied force of the strain gage and subsequently, linear to the change in output voltage. Other important specifications of the device are given in the figure C.1.

3.5 ROS - Robot Operating System

3.5.1 What is ROS?

The Robot Operating System (ROS) is a collection of software libraries and tools that assist in the development of robot applications. ROS contains everything a robot software system needs, from drivers to cutting-edge algorithms and strong development tools, and it's all open source.

ROS has features such as hardware abstraction, device drivers, process communication across many machines, testing and visualization tools. The way the program is executed and communicated is a fundamental component of ROS since it allows you to develop complicated software without having to understand how specific hardware works e.g., it helps the user in implementing kinematics of a mobile wheeled robot without the need for the user to thoroughly understand the mathematics and concepts involved behind it.

ROS allows a network of processes (nodes) to communicate with a central hub (core). Nodes can operate on many devices and communicate with the hub in a variety of ways. Providing requestable services or defining publisher/subscriber relationships with other nodes are two of the most common approaches to building a network. Both approaches use certain message types to communicate. Although the core packages give some kinds, other packages can specify their message types through declaring normal classes and structs in python or C++.

Some of the ROS concepts are explained briefly below:

1. Nodes - Processes that execute computing are referred to as nodes. A robot control system often consists of numerous nodes, and ROS is designed to be modular at a fine-grained scale. One node, for example, controls a laser range-finder, another node controls the wheel motors, another node conducts localization, another node performs path planning, another node displays a graphical representation of the system, and so on. A ROS node is created by using a ROS client library like roscpp or rospy.

2. Master - To the entirety of the Computation Graph, the ROS Master provides

name registration and lookup. Nodes would be unable to locate one another, exchange messages, or execute services without the Master.

3. Parameter Server - Data can be kept in a central location using the Parameter Server. Currently, it is a component of the Master.

4. Messages - are used to communicate between nodes. A message is nothing more than a data structure with typed fields. Arrays of primitive kinds, as well as standard primitive types (integer, floating-point, boolean, etc.), are supported. Messages can contain structures and arrays that are arbitrarily nested (much like C structs).

5. Topics - Messages are routed through a transport system that uses publish/subscribe semantics to route them. A message is sent out by a node by publishing it to a specific topic. The message's topic is a name that is used to identify the message's content. A node interested in a particular type of data will subscribe to the relevant subject. A single topic can have several publishers and subscribers at the same time, and a single node can publish and/or subscribe to multiple topics. Publishers and subscribers are generally unaware of each other's existence. The objective is to separate the creation and consumption of information. A subject can be thought of as a strongly typed message bus. Each bus has a unique name, and anyone can connect to it to transmit or receive messages as long as they have the appropriate type of device.

6. Services - The publish/subscribe model is a fairly flexible communication paradigm, but its one-way, many-to-many transport is ineffective for request/response interactions, which are common in distributed systems. Requests and responses are handled by services, which are described by two message structures: one for the request and one for the response. A client uses a service that a supplying node offers under a name by sending a request message and waiting for a response. This interaction is usually presented to the programmer as if it were a remote procedure call in ROS client libraries.

7. Bags - Bags are a storage and playback format for ROS message data. Bags are a crucial device for storing data that can be difficult to acquire but is required for creating and testing algorithms, such as sensor data. (Mostly used by ROS developers)

3.5.2 Why are we using ROS?

1. ROS is a well-known open-source framework that offers pre-built packages for the majority of actuators, sensors, and other hardware, as well as other packages to create drivers for components that do not have drivers. In our case, we have planned on using the ROS packages to implement different types of sensor fusion (LiDAR and IMU), kinematics, navigation, etc. without coding it from scratch.

2. Before a project can be executed, it must go through a series of simulations. The Rviz and Gazebo modules are pre-installed in ROS, allowing for the creation of a customizable virtual environment for testing the robot's functions. Furthermore, these simulations can save time during execution. Rviz can also be used to show a SLAM live map or localization on a preloaded map.

3. ROS has a wide range of language support. When working in groups, people with varying programming backgrounds get together to construct something, this attribute can be extremely valuable. A ROS-based system can incorporate packages and libraries from many programming languages that function together as a whole. It's especially important when creating a cross-platform system or incorporating a large number of packages, some of which may be written in several languages. Therefore, it will eventually be useful when establishing M2M communication through IoT nodes.

3.6 ROS Packages

This section explains the technicalities of the packages that were used in our project. It will also be explaining how these packages were implemented and how the data pipelining was managed.

3.6.1 Differential_Drive

The differential drive package contains the tools needed to connect a wheeled mobile robot with a differential drive to the ROS Navigation Stack. The goal of this package is to develop a differential drive controller implementation that is independent of the individual hardware used to implement the robot, such as a certain micro-controller unit. This package's goal is to offer a user interface for the navigation stack. It receives a twist message from the navigation stack and outputs lwheel and rwheel messages as motor driver strengths. The module accepts wheel encoder signals from the hardware and creates the ROS navigation stack's tf transform messages. The package includes the following nodes:

1. diff_tf - Provides the base_link transform.
2. pid_velocity - A basic PID controller with a velocity target.
3. twist_to_motors - Translates a twist to two motor velocity targets
4. virtual_joystick - A small GUI to control the robot.

Figure 3.13 shows how the nodes are connected to each other. Here, we can see that the control is only implemented for two nodes of motors whereas in our case, we have a four-wheel differential drive and thus, requires four such control systems. Therefore, the control and odometric functionality of the package was changed to fit our needs as it was replicated for the other two wheels as well. Thus, gave us an initial framework to work upon in the shape of the functional differential drive of a four-wheel robot.

3.6.2 rosserial

The Rosserial package is used to communicate between the Arduino and the NVIDIA Jetson Nano. The data that is sent to the Arduino includes the motor PWM signals, and the data from the wheel encoder is received by the Jetson nano. The Arduino uses

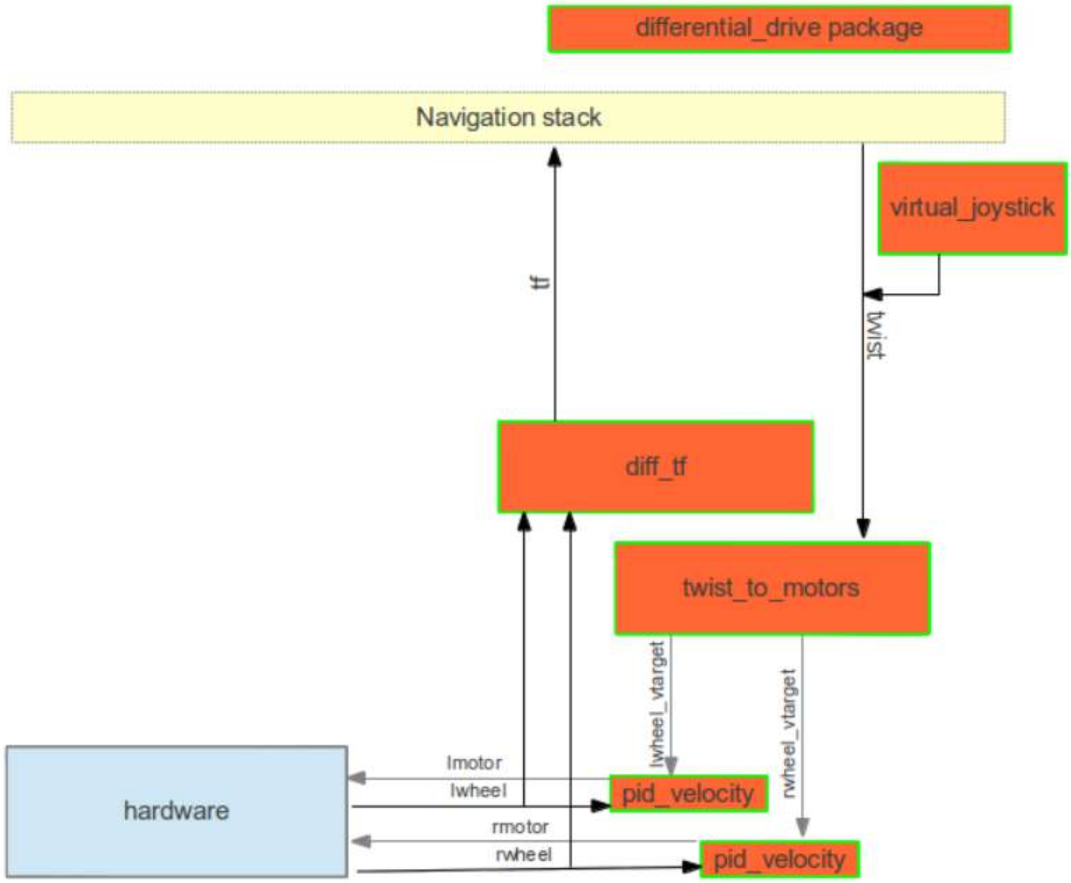


Figure 3.13: Diffferential_Drive package APIs [13]

the package to subscribe to the ROS topics and can post the encoder values back to the Nano. The package uses a typical 115200 baud rate Universal Asynchronous Receiver Transceiver (UART) connection [14]. This enables us to create modularity within the system since now, the Arduino mega is responsible to supply commands to the motor drivers and also providing the encoder values to the jetson nano thus, saving some of the memory of jetson nano and providing a separate platform to operate the wheels.

3.6.3 Transform Frame (tf)

tf is a tool that allows us to identify several coordinate frames throughout time. tf keeps a record of the relationships between coordinate frames in a time-buffering tree structure, allowing the user to transform points, vectors, and other objects between any two coordinate frames at any moment in time. A world frame, base frame, gripper frame, head frame, and other 3D coordinate frames in a robotic system usually change over time. tf records all of these frames over time and lets us answer the questions such as:

1. Where was the head frame relative to the world frame?
2. What is the pose of the sensors on the robot relative to my base?
3. What is the current pose of the base frame in the map frame?

Thus, it is very useful in terms of localization of the robot and is essential for us to implement as there are multiple sensors on the robot in our case e.g. IMU and LiDAR. Figure 3.14 helps in visualizing how these co-ordinate frames are formed concerning each other.

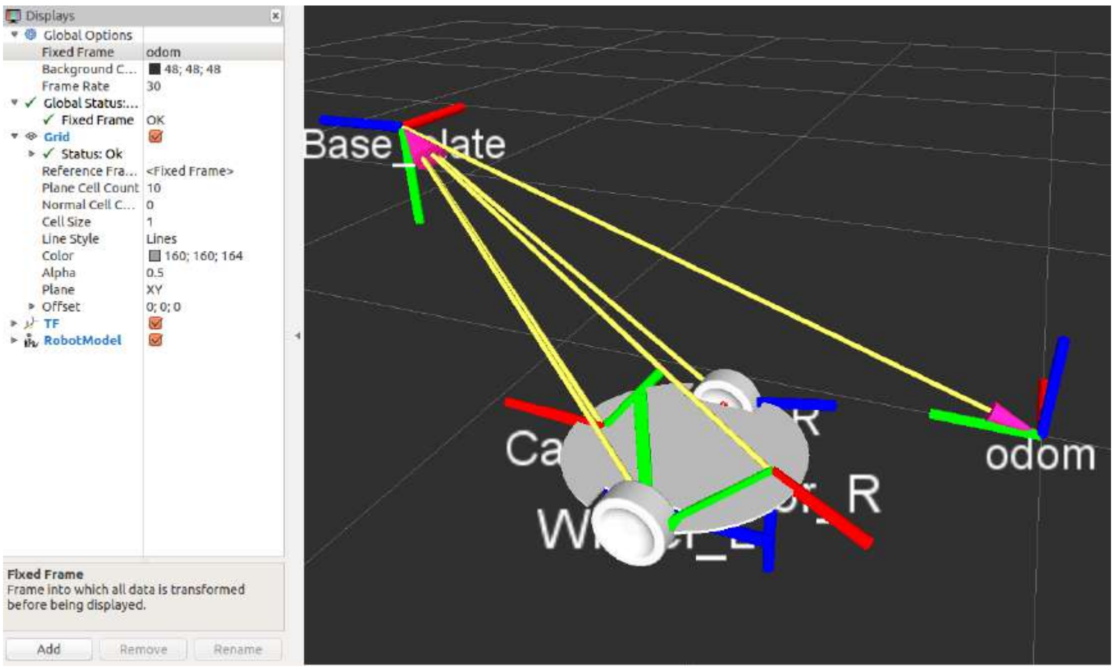


Figure 3.14: transform frame (tf) formation [15]

The package subscribes to tf type nodes only and publishes the data on a different tf node. This allows other packages to take the data of any co-ordinate frame and use it to its cause. The following points briefly describe the mainframes in our application:

1. map - The origin of the map frame is an arbitrarily chosen location in the world. In the globe, this coordinate frame is fixed.

2. odom - The origin of the frame is where the robot is initialized. In the world, this coordinate frame is fixed.

3. base_footprint - has its origin exactly beneath the robot's core. It's the robot's two-dimensional pose. As the robot moves, this coordinate frame moves as well.

4. base_link - has its origin directly at the robot's pivot point or center. As the robot moves, this coordinate frame moves as well.

5. laser_link - has its origin at the laser sensor's center (i.e. LIDAR). Concerning the base link, this coordinate frame remains fixed (i.e. "static").

6. imu_link - has a similar concept as the laser_link however, is placed at the center of the IMU sensor.

The graph for our current development in the tf is shown in Figure 3.15

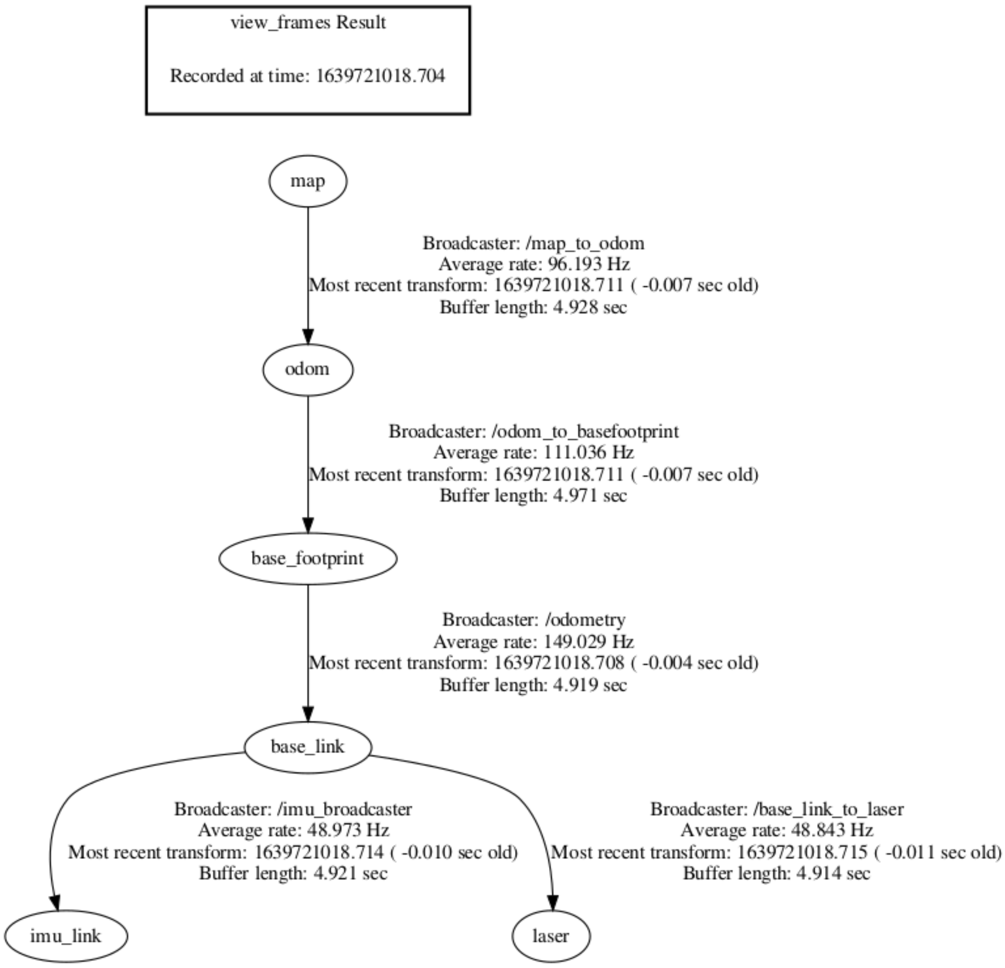


Figure 3.15: Links of tf nodes (Tree Diagram)

3.6.4 rplidar

This package is essential in reading the data from the LiDAR hardware and publishing the acquired data on the topic 'scan'. The output is in form of multiple numbers that are displayed at 5-15 Hz per array. This consists of the range where the scan is detected from and the range is till 12 meters. This package helps the data from LiDAR to be integratable with other packages as it converts it to the form which is understandable for the other packages. It does not need to subscribe to any of the nodes to publish its data however, it needs the address of the connection to the LiDAR device (which is mainly of a USB bus ID namely ttyUSB0x) [16].

3.6.5 Hector SLAM

HectorSLAM combines a robust scan matching technique with a 2D SLAM system. In this experiment, multiple parameters of scanning rate from a LiDAR sensor were investigated, as well as estimation of robot movement in real-time [17]. Therefore, when working with Hector SLAM, we do not need to worry about the localization data that is needed at the time of mapping and thus, is a very useful tool in mapping the environment only through LiDAR. A map of our project's lab section was made through this method and shown in Figure 3.16. This package can be simply be used through following the tutorial in *this* link

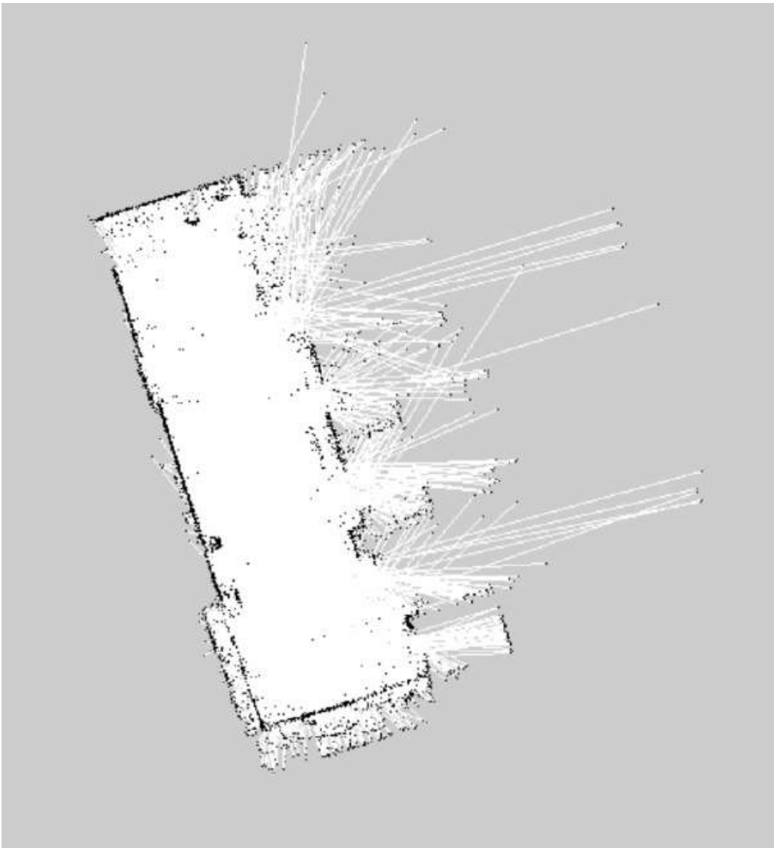


Figure 3.16: Hector SLAM generated map of Projects Lab

3.6.6 Gmapping

As a ROS node called slam gmapping, the package enables laser-based SLAM (Simultaneous Localization and Mapping). From laser and posture data gathered by a mobile robot, it can produce a 2-D occupancy grid map (like a building floorplan) using slam gmapping node of the package. This mapping is a bit tricky as it requires a transform tree to already exist which contains the odometry, baselink, and laser scan transforms. It can use these to more accurately map the environment when compared to Hector SLAM. Thus, we have used this in our prototype as well to map the cafeteria. The figure 3.17 shows the map we formed for the cafeteria using gmapping after editing it using the gimp 2.0 software.

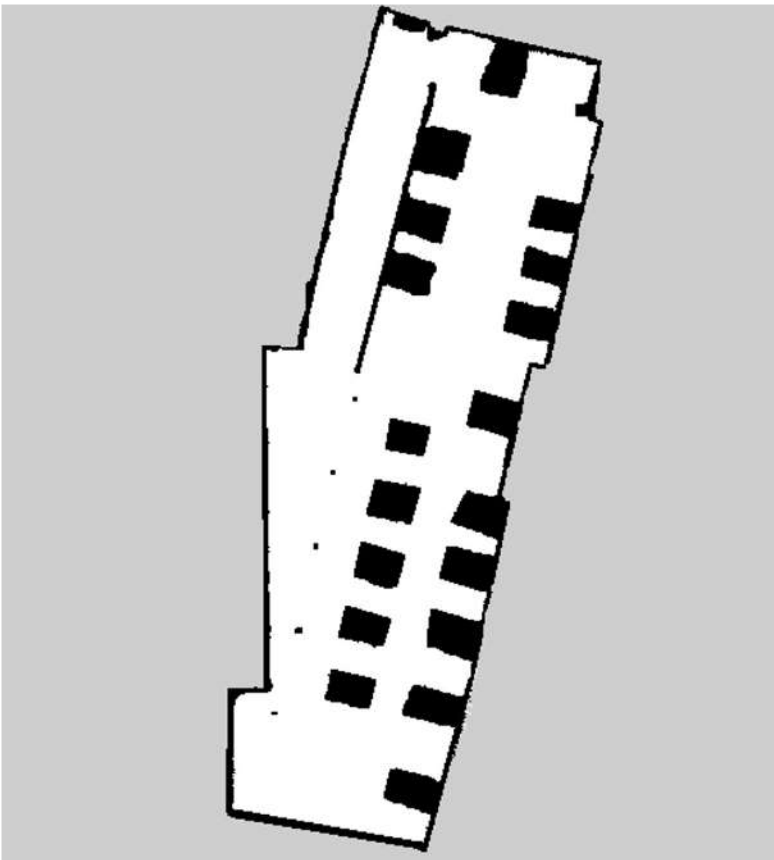


Figure 3.17: Gmapping generated map of Tapal Cafeteria

We can clearly witness how much clearer the map is formed and thus, it helps us in navigating in a much better way.

NOTE: the package already uses the amcl localization so doing it seperately might negatively impact the results.

3.6.7 mpu_6050_driver

This package is used to read the data of IMU sensors 6050 and 9250 specifically. In our case, we have used the 9250 IMU sensor and is useful in calibrating the IMU sensor in it's initial placement and also retrieving odometric data from it. The package does not subscribe to any topic as the address for it only works through GPIO pin connection with the IMU sensor whose address is already hardcoded inside the package. After making it's a connection with the Jetson nano, we can simply start the package and the data would be published on the topic 'imu_data'.

3.6.8 Robot_pose_EKF

The Robot Pose EKF package is used to estimate a robot's 3D pose using (partial) pose measurements from various sources. To combine measurements from wheel odometry, IMU sensor, and visual odometry, it employs an extended Kalman filter with a 6D model (3D position and 3D orientation). The core idea is to provide loosely coupled sensor integration, with sensor signals being received as ROS messages. For the time being, we'll combine odometry data (based on wheel encoder tick counts) with data from an IMU sensor (i.e. sensor fusion) to provide enhanced odometry data, allowing us to acquire regular estimations of the robot's location and orientation as it travels about its environment. For a robot to navigate properly and produce useful maps, it needs accurate information. The workhorse behind it all is an extended Kalman filter. It gives a more reliable estimation of the robot's pose than only using wheel encoders or an IMU.

When compared to using only actual sensor measurements, EKFs produce more accurate state estimates (state vectors). In robotics, EKFs can provide a smooth approximation of a robotic system's current state over time by using both real sensor data and predicted sensor measurements to reduce the impact of noise and mistakes in sensor measurements. Figure 3.18 shows the effect of applying this filter on a noisy system.

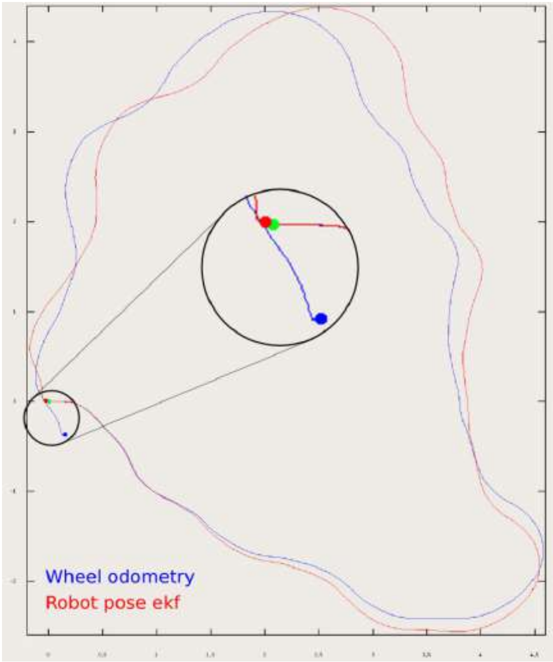


Figure 3.18: EKF comparison with normal sensor data [18]

Figure 3.19 shows the subscriber and publishing nodes of the package in our case, and how the data pipelining is done in this API. It uses the information of the wheel encoder which is being inputted through subscribing to the encoder topic from the Arduino node and it also takes in the value from mpu_6050_package node which is publishing imu_data simultaneously.

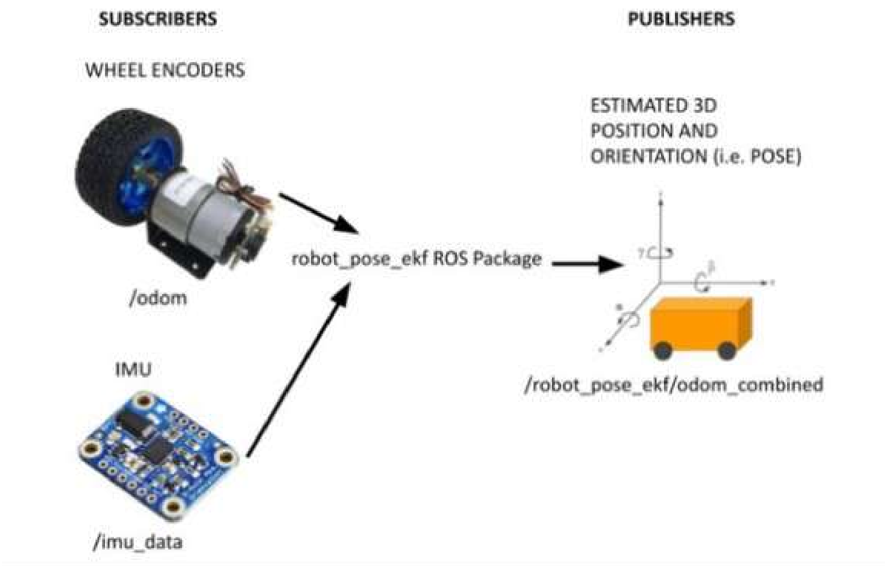


Figure 3.19: EKF package implementation [18]

3.6.9 AMCL

A probabilistic localization system for a robot moving in two dimensions is amcl. It uses a particle filter to track a robot's pose against a known map, as described by Dieter Fox's adaptive (or KLD-sampling) Monte Carlo localization approach. amcl receives a laser-based map, laser scans, and transform signals as input and returns pose estimations. Amcl initializes its particle filter according to the parameters specified when it starts up. Because of the defaults, the first filter state will be a modestly sized particle cloud centered about if no settings are set (0,0,0). Therefore, it would be hard for a system to recognize the odometry of a robot to the magnitude of the length of a map and that is where amcl contributes to clearing things out by using a probabilistic model. Figure 3.20 shows how the estimate is used in providing a much better transform between multiple frames.

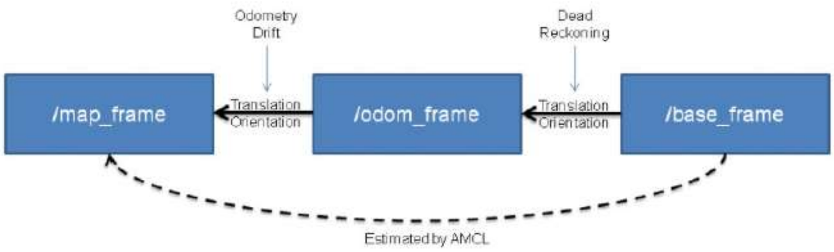


Figure 3.20: AMCL Package Working Mechanism [19]

3.6.10 Move_Base

The move base package implements an action that attempts to attain a target in the world using a mobile base when given a goal. To complete its global navigation duty, the move base node connects a global and local planner. It supports any global planner that adheres to the navigation core BaseGlobalPlanner interface in the nav core package, as well as any local planner that adheres to the navigation core BaseLocalPlanner interface. The move base node also keeps track of two costmaps, one for the global planner and one for the local planner (see the costmap 2d package), which are needed to complete navigation tasks. The costmaps functionalities can be explained briefly as:

1. Global Costmap - This costmap is used to build long-term plans for the entire environment, such as calculating the shortest path on a map from point A to point B.

2. Local Costmap - This costmap is used to create short-term environmental plans, such as avoiding obstacles.

The configurations of the costmaps are done using a .yaml file as they store the input arguments to certain functions. Furthermore, the Trajectory Rollout and Dynamic-Window techniques to local robot navigation on a plane are implemented in this package. The controller generates velocity orders to deliver to a mobile base based on a plan and a costmap. Figure 3.21 shows how the data is pipelined within the move_base package.

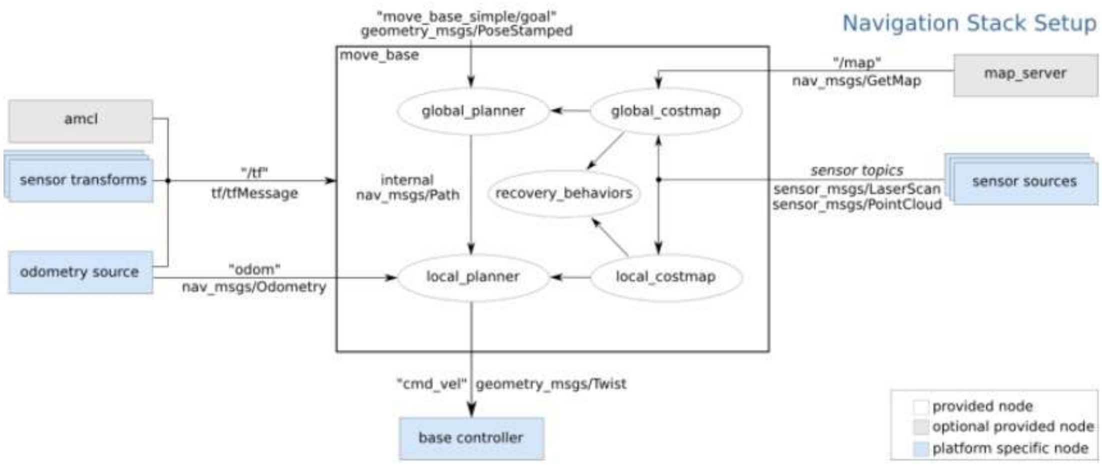


Figure 3.21: Move_base package internal processes [20]

CHAPTER 4

PROTOTYPING AND TESTING

4.1 Final Prototype Assembly

4.1.1 Prototype on CREO

Before fabricating the levels of the robot, we first tested out the assembly in simulation using CREO assembly and through that, we were able to verify how the parts will align with each other. The figure 4.1 and figure 4.2 shows how it looked.

Figure 4.1: Robot Assembly CREO (Front-side)

Figure 4.2: Robot Assembly CREO (Back-side)

The simulation allowed us to see how the structure will be formed and also helped in deciding the dimensions of the screws, plates, and level difference between the robot's platforms. Since laser cutting is expensive and a time consuming task, we needed to make sure that the design we fabricate is errors free.

4.1.2 Physical Prototype

After getting the design laser-cutted, we assembled the structure and implemented the system architecture on it from the first PoC and the additional hardware that was involved in the second PoC. Figure 4.3 shows the front side of the new structure of the robot. We can see the newly assembled BMS on the bottom level alongside the motor's PCB. The first level (i.e., the box) contains the weight instrument sensor and a free space for the payload to rest. On top of the third level, we have the LIDAR sensor such that no obstruction occurs in it's line of sight/sensing.

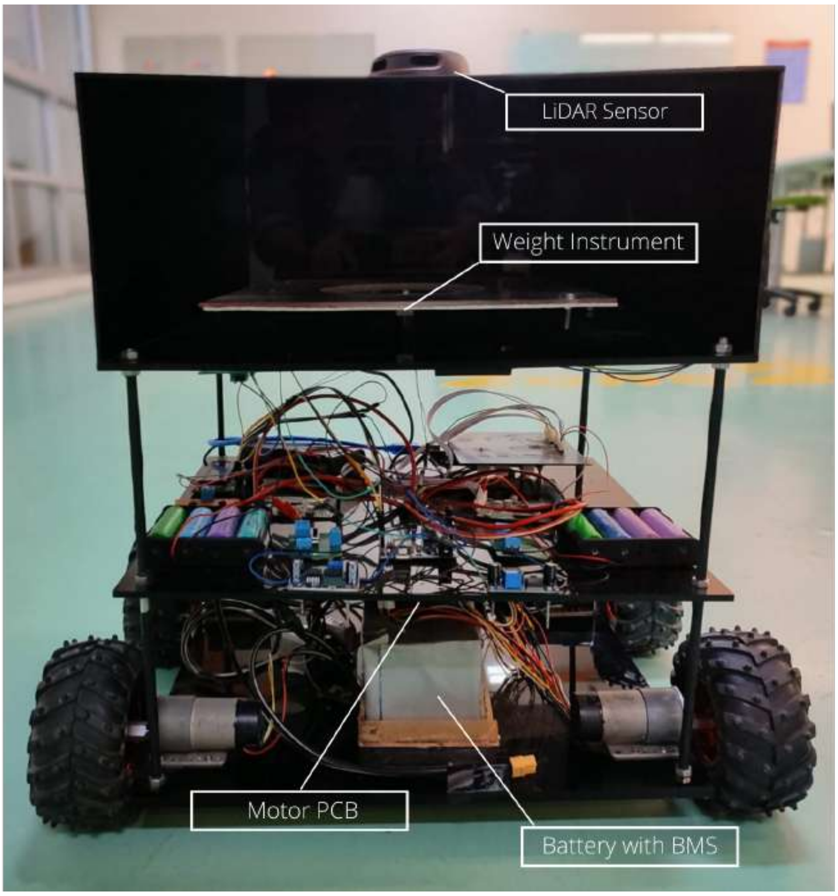


Figure 4.3: Front view of the robot's final structure

Moreover, we can see from figure 4.4 that there is a raspberry pi 4B instead of Jetson Nano here and that is due to a hardware fault and therefore, we had to replace it with easily available raspberry pi and also, we can see the PCB shield for arduino mega alongside busses coming from the motor drivers and various other components as well.

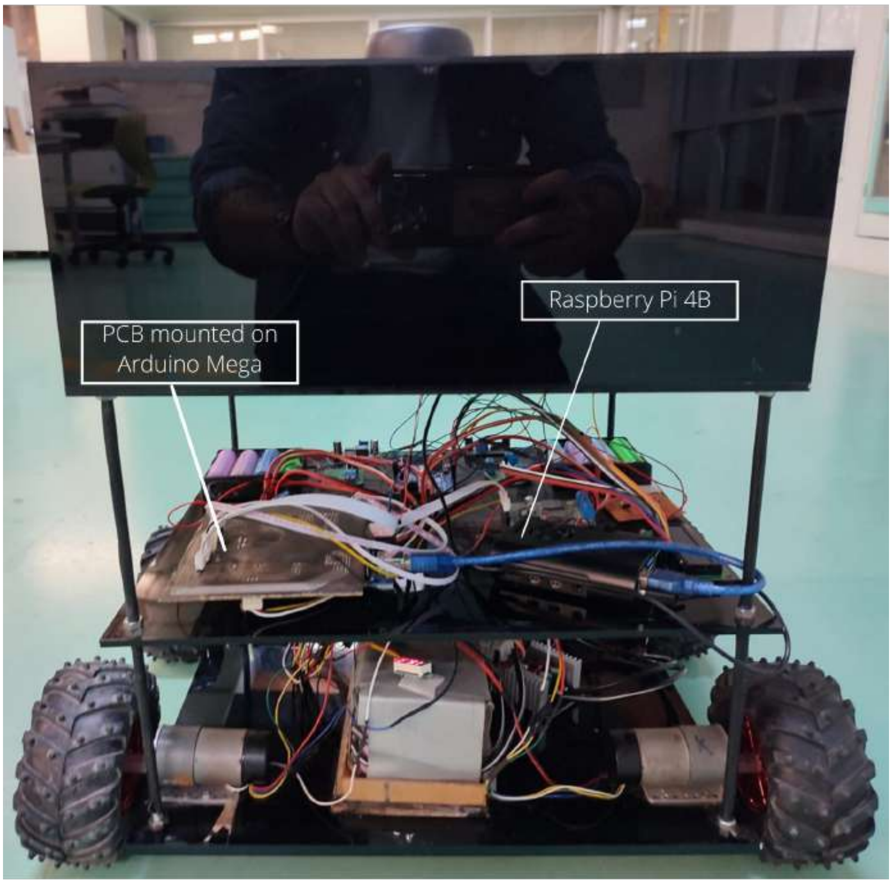
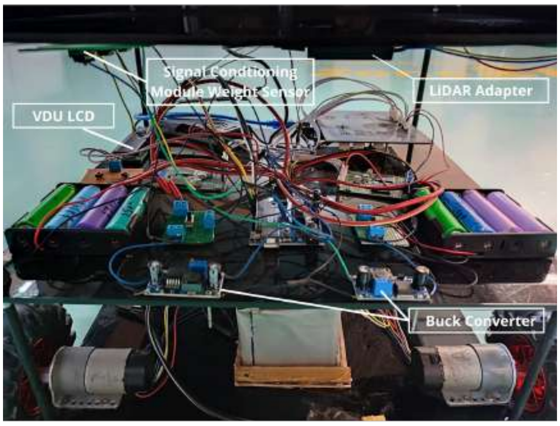


Figure 4.4: Back view of the robot's final structure

Alongside this, from figure 4.5 and figure 4.6, we can also see the buck converter, main switches of the robot, weight instrument signal conditioning and power circuit, and the VDU for displaying the weight of the payload.



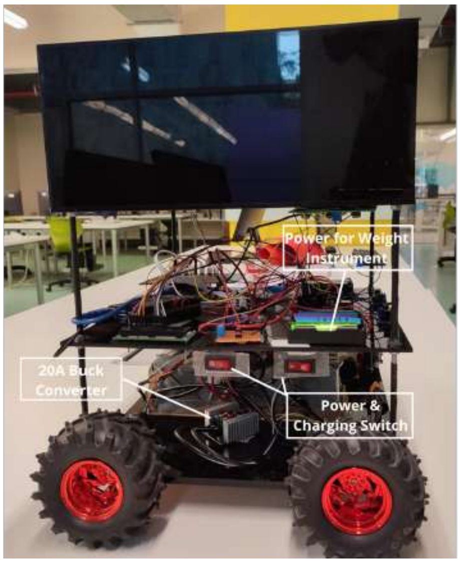


Figure 4.5: Second level of the robot

Figure 4.6: Side of the robot

4.2 Constructing Power System of Robot

We needed to construct a standalone power system for the robot and the design details for it are already discussed in the section 3.3.6. Here, we will mostly be discussing how the power system was practically constructed. Since we know that in order to make a bigger power, we can make combinations of smaller power systems i.e., the lithium ion batteries. As we needed 48 batteries such that 4 sets of 12 batteries in series will be in parallel, we figured that this will form a very heavy package, we used cage for the batteries to hold their place all the time. figure 4.7 shows how such a skeletal with batteries was formed. Moreover, from figure 4.8, we can see how the end result was formed alongside the BMS installed on side of it. We can also see the power cables coming out of the BMS and the charging port for recharging purposes also had to be installed here. Moreover, we had to perform spot welding for attaching the batteries and the conducting strips are placed on top of them in such a manner that the desired configuration was formed.

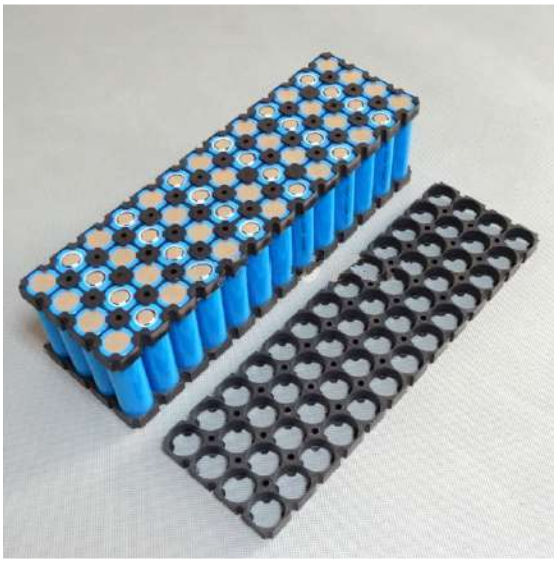


Figure 4.7: Cage of battery pack

Figure 4.8: battery pack with BMS

4.3 Forward Kinematic Modelling of the Robot

4.3.1 Derivation

First, we will need to design the 4WD scenario as such that it helps us estimate the velocity, acceleration, position, and orientation of the robot. We will be taking help from Karakurt paper that is cited in the literature review. We will be using the figure below to explain it's working.

For it to behave as a differential drive, we will need the wheels of a particular side to have equal velocities and therefore, will assume in these calculations as well. Thus,

$$v_1(t) = v_2(t) \text{ and } v_3(t) = v_4(t)$$

$$v_L = \frac{v_1 + v_2}{2} = \frac{\omega_1 + \omega_2}{2} \cdot r$$

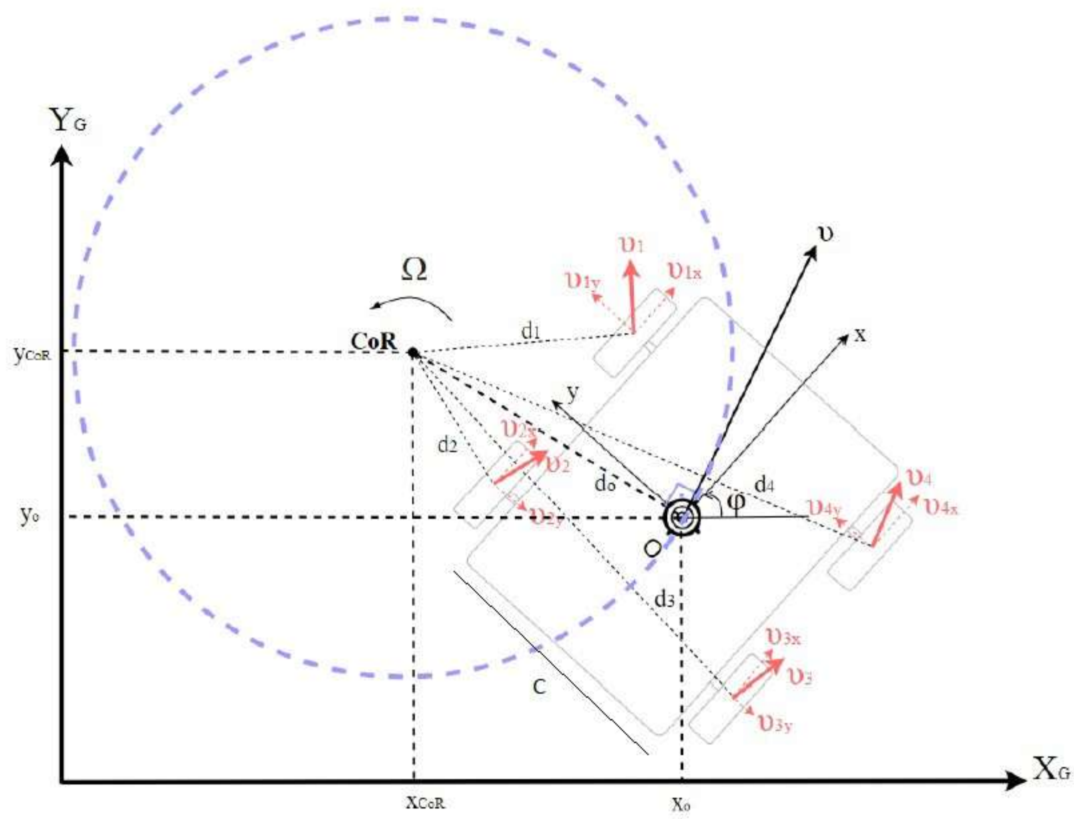


Figure 4.9: Kinematic Model DeLRO

$$v_R = \frac{v_3 + v_4}{2} = \frac{\omega_3 + \omega_4}{2} \cdot r$$

Since there is no velocity in y-direction, we can rewrite the equations for the overall velocity as:

$$v_x = \frac{v_L + v_R}{2} = \frac{\omega_1 + \omega_2 + \omega_3 + \omega_4}{4} \cdot r$$

The difference in the speed of left and right wheels will determine the change of orientation of the robot. Therefore,

$$v_R - v_L = \omega \cdot C$$

$$\frac{\omega_1 + \omega_2}{2} \cdot r - (\frac{\omega_3 + \omega_4}{2}) \cdot r = \omega \cdot C$$

Therefore, the forward kinematics can be written as:

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{C} & -\frac{1}{C} & \frac{1}{C} & \frac{1}{C} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}$$

For global FoR, we will use phi to denote the orientation of robot and use that to calculate velocities in global x-y co-ordinates.

$$\begin{bmatrix} \tilde{v}_x(t) \\ \tilde{v}_y(t) \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} v_x(t) \\ v_y(t) \end{bmatrix}$$

$$\phi = \frac{d\omega}{dt} = \frac{d}{dt}\left(\frac{\omega_1 + \omega_2}{2} \cdot \frac{r}{C} - (\frac{\omega_3 + \omega_4}{2}) \cdot \frac{r}{C}\right)$$

$$\tilde{v}_x(t) = v_x(t)\cos(\phi) = \frac{\omega_1 + \omega_2 + \omega_3 + \omega_4}{4} \cdot r\cos(\phi)$$

$$\tilde{v}_y(t) = v_x(t)\sin(\phi) = \frac{\omega_1 + \omega_2 + \omega_3 + \omega_4}{4} \cdot r\sin(\phi)$$

We can further discretize this model using different models e.g. rectangular, trapezoidal, exact, etc. and therefore, program it on our microcontroller as well.
The models when applied to a two wheel differential drive gives us the following equations.

Rectangular Model

$$x_{n+1} = x_n + v_n T_s \cos(\phi_n)$$

$$y_{n+1} = y_n + v_n T_s \sin(\phi_n)$$

$$\phi_{n+1} = \phi_n + \Delta\phi_n$$

$$\Delta\phi_n = T_s\omega_n$$

Trapezoidal Integration Model

$$x_{n+1} = x_n + \frac{v_n T_s}{2}\left(\cos(\phi_n) + \cos(\phi_n + \Delta\phi_n)\right)$$

$$y_{n+1} = y_n + \frac{v_n T_s}{2}\left(\sin(\phi_n) + \sin(\phi_n + \Delta\phi_n)\right)$$

$$\phi_{n+1} = \phi_n + \Delta\phi_n$$

$$\Delta\phi_n = T_s\omega_n$$

Exact Integration Model

$$x_{n+1} = x_n + \frac{v_n}{\omega_n}\left(\sin(\phi_n + \Delta\phi_n) - \sin(\phi_n)\right)$$

$$y_{n+1} = y_n + \frac{v_n}{\omega_n}\left(\cos(\phi_n + \Delta\phi_n) - \cos(\phi_n)\right)$$

$$\phi_{n+1} = \phi_n + \Delta\phi_n$$

$$\Delta\phi_n = T_s\omega_n$$

Geometry-Based Model

$$x_{n+1} = x_n + v_n T_s \cos\left(\phi_n + \frac{\Delta\phi_n}{2}\right)$$

$$y_{n+1} = y_n + v_n T_s \sin\left(\phi_n + \frac{\Delta\phi_n}{2}\right)$$

$$\phi_{n+1} = \phi_n + \Delta\phi_n$$

$$\Delta\phi_n = T_s\omega_n$$

Taylor-Series Model

$$x_k = x_{k-1} + \frac{v_k}{\omega_k}\left(\Delta\phi_k\left(cos(\phi_{k-1}) - \frac{\Delta\phi_k}{2}\sin(\phi_{k-1})\right)\right)$$

$$y_k = y_{k-1} - \frac{v_k}{\omega_k}\left(\Delta\phi_n\left(-\sin(\phi_n) - \frac{\Delta\phi_n}{2}\cos(\phi_n)\right)\right)$$

$$\phi_k = \phi_{k-1} + \Delta\phi_k, \quad \text{where } \Delta\phi_k = T_s\omega_k.$$

Range-Kutta Method - 4th Order

$$\Delta x_1 = T_s f(X_n, U_n) = T_s V_n \cos(\phi_n)$$
$$\Delta x_2 = T_s V_n \cos\left(\phi_n + \frac{1}{2}\left(T_s V_n \cos(\phi_n)\right)\right)$$
$$\Delta x_3 = T_s V_n \cos\left(\phi_n + \frac{1}{2}\left(T_s V_n \cos\left(\phi_n + \frac{1}{2}(T_s V_n \cos(\phi_n)\right)\right)\right)$$
$$\Delta x_4 = T_s V_n \cos\left(\phi_n + T_s V_n \cos\left(\phi_n + \frac{1}{2}\left(T_s V_n \cos\left(\phi_n + \frac{1}{2}(\ldots)\right)\right)\right)\right)$$

$$x_{n+1} = x_n + \frac{1}{6}\left(\Delta x_1 + 2\Delta x_2 + 2\Delta x_3 + \Delta x_4\right)$$

$$\Delta y_1 = T_s V_n \sin(\phi_n)$$
$$\Delta y_2 = T_s V_n \sin\left(\phi_n + \frac{1}{2}\left(T_s V_n \sin(\phi_n)\right)\right)$$
$$\Delta y_3 = T_s V_n \sin\left(\phi_n + \frac{1}{2}\left(T_s V_n \sin\left(\phi_n + \frac{1}{2}(T_s V_n \sin(\phi_n))\right)\right)\right)$$
$$\Delta y_4 = T_s V_n \sin\left(\phi_n + T_s V_n \sin\left(\phi_n + \frac{1}{2}(T_s V_n \sin\left(\phi_n + \frac{1}{2}(\ldots)\right)\right)\right)$$

$$y_{n+1} = y_n + \frac{1}{6}(\Delta y_1 + 2\Delta y_2 + 2\Delta y_3 + \Delta y_4)$$

$$\Delta\phi_1 = T_s f(\phi_n, U_n) = T_s\omega_n$$

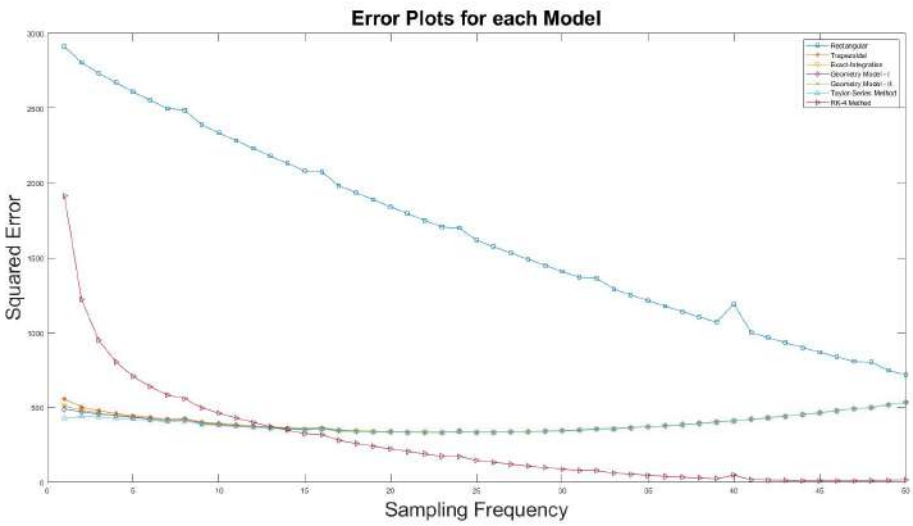
$$\Delta\phi_2 = T_s f(\phi_n + \frac{1}{2}\Delta\phi_1, U_n) = T_s\omega_n$$

$$\Delta\phi_3 = T_s f(\phi_n + \frac{1}{2}\Delta\phi_2, U_n) = T_s\omega_n$$

$$\Delta\phi_4 = T_s f(\phi_n + \frac{1}{2}\Delta\phi_3, U_n) = T_s\omega_n$$

$$\phi_{n+1} = \phi_n + \frac{1}{6}(\Delta\phi_1 + 2\Delta\phi_2 + 2\Delta\phi_3 + \Delta\phi_4) = T_s\omega_n$$

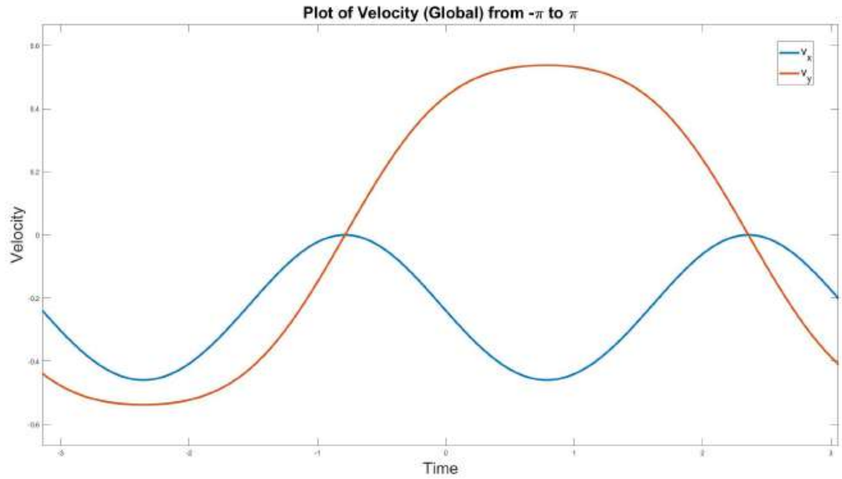
Here, we figured out the error of different discrete models and analyzed that many of the models appear to act at same accuracy except the RK4 method which approximates the value of prediction till 4th order.

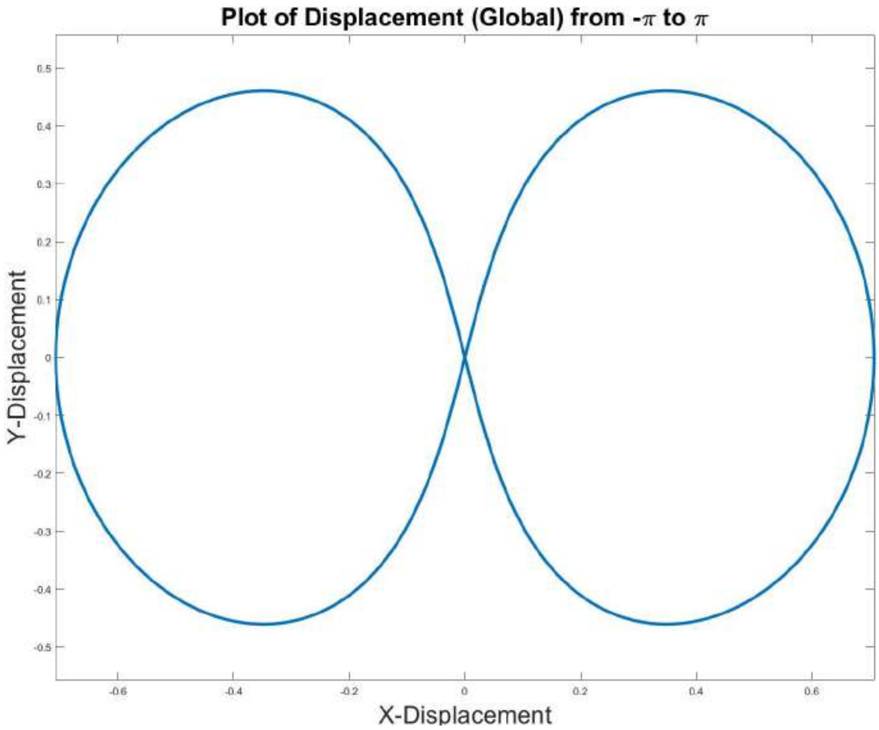


However, it is a bit computational expensive and given the constraints of our jetson nano being a 2GB model, therefore, we can test these models out through changing the ROS package configurations.

4.3.2 Simulation

The following results were obtained when simulating the above model on MATLAB for left wheel velocities equal sine function and right wheel velocities equal cosine function.





4.4 Inverse Kinematic Modeling of the Robot

4.4.1 Derivation

Similarly, the inverse kinematic equation can be written as:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 0 & -C \\ 1 & 0 & -C \\ 1 & 0 & C \\ 1 & 0 & C \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

In globabl FoR, we can rewrite the equations using the derivations above as:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{r}{4} \begin{bmatrix} \cos(\phi) & \sin(\phi) & -C \\ \cos(\phi) & \sin(\phi) & -C \\ \cos(\phi) & \sin(\phi) & C \\ \cos(\phi) & \sin(\phi) & C \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Therefore, using the control signals from the cmd/twist.h library of ROS, we can convert those signals into wheel PWM signals and hence, give directions to the robot.

4.4.2 Discretization

We can use the following formulas to get the useful parameters in determining the trajectory plan of the robot.

Assuming previous trajectory and simple substitution of nT gives us the model as:

$$\begin{bmatrix} x_n \\ y_n \\ \phi_n \\ x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} \sin(nT) \\ \cos(nT) \\ \tan^-1\left(\dfrac{y_{n+1} - y_n}{x_{n+1} - x_n}\right) \\ \sin(nT + T) \\ \cos(nT + T) \end{bmatrix}$$

$$\mu = \frac{1}{2}\left(\frac{\sin(\phi_n)(y_{n+1} - y_n) + \cos(\phi_n)(x_{n+1} - x_n)}{\cos(\phi_n)(y_{n+1} - y_n) - \sin(\phi_n)(x_{n+1} - x_n)}\right)$$

$$x_m = \frac{1}{2}(x_n + x_{n+1})$$

$$y_m = \frac{1}{2}(y_n + y_{n+1})$$

$$x_* = x_m + \mu(y_n - y_{n+1})$$

$$y_* = y_m + \mu(x_{n+1} - x_n)$$

$$\theta_1 = \tan^{-1}\left(\frac{y_* - y_n}{x_n - x_*}\right)$$

$$\theta_2 = \tan^{-1}\left(\frac{y_* - y_{n+1}}{x_{n+1} - x_*}\right)$$

We can use this model to plot the trajectory on MATLAB. The code to plot the trajectory and errors are in appendix section D. Moreover, the model derived are used in changing the functionality of differential drive package of ROS from two wheels to four wheels as the package for four wheels do not exist and this was a necessary step to be taken by us in order to ensure that precise odometry calculations are processed in our system.

4.5 PCB Design for Controller Circuit

At first, our whole circuitry was based on jumper wires and veroboards however, around week 5/6 of the capstone 2, we stumbled upon a major problem of incorrect values read by arduino and delays in catching pwm signals etc. We brainstormed on what is causing the problem and it was actually originating from the high resistance and breakage within the jumper wires. apart from that, sometimes the open solder would short itself as not every board was fixed due to space limitations. Therefore, we planned onto designing a PCB module for the major circuits in our robot and technically, most of the wires were emerging from the motors, drivers, and arduino mega. Hence, the designed PCBs were for these modules and were created in two iterations.

4.5.1 First Iteration of the Design

Here, we designed two PCBs as shown in Figure 4.10 and Figure 4.11. The notion behind making two separate designs was that we wanted to mitigate the possibility of any cross track capacitance within the circuit as that would delay the signals reaching to it's desired port. Here, we can see that the designed PCB is on the software easyEDA

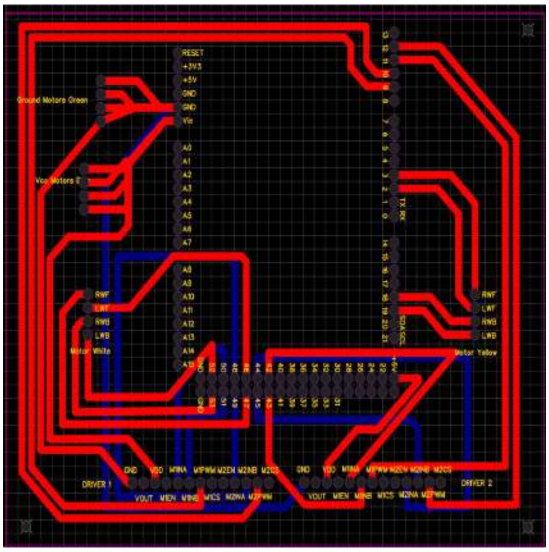


Figure 4.10: PCB Design of Mega shield

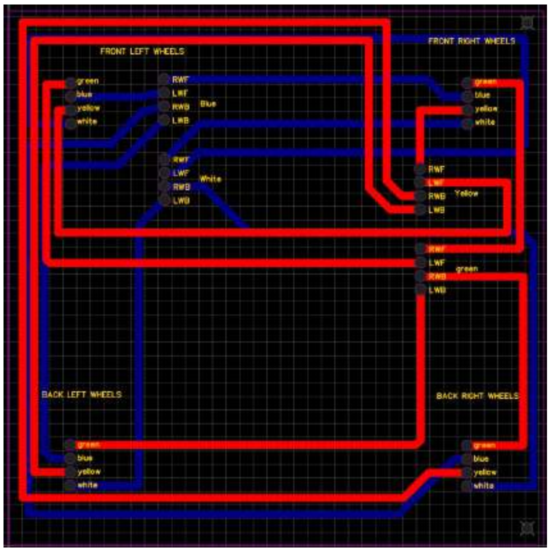


Figure 4.11: PCB Design for motors

and is based on 2 layer PCB. from figure 4.10, we have outlined each and every pin of the arduino mega to ensure that the pcb can act as a shield for it and the connections coming from various sides indicates that bus wires attach to them in a clean manner. The busses will be coming from the second PCB and the motor drivers mostly. on figure 4.11, we can see that we have given slots for all 4 motors to connect their wires onto the PCB and then, we have systematically arranged the colors of each of the wire and this allowed us to ensure that the capacitive tracks were omitted as of now, the ground (or VCC) wires are following along a single path rather them being all over the PCB and interfering with other signals. Through this, we were able to successfully design the PCB however, the problem arose when the fabricated plates had to be soldered and in this design, we overlooked one key aspect of double sided PCB that whenever we want to solder headers onto it, we need to make sure that the tracks joining onto the header pins located in parallel must be coming from the same side of the plate or otherwise, soldering becomes a headache and is prone to errors. Therefore, we had to design the PCB again with a different perspective.

4.5.2 Second Iteration of the Design

The second iteration of the PCB designs are shown in figure 4.12 and figure 4.13. In these designs, we can see that a wider clearance is set between the tracks, track width is slightly changed, header pins are reduced, and vias are used to mitigate the soldering problem. Before going into vias, we also minimized the 90 degree turns in the tracks as that would cause voltage drift and possible signal packet losses within the circuit as ideally, the tracks should follow a circular path from one point to another. Vias are smaller holes that we see are used to interconnect the plates on the opposite sides and these were used when we realized while designing that there are not much we could do to route a point to point connection such that the ends lie on the same side of the PCB plate. Thus, these points allowed us to route the path on other end of the PCB until the near point of the header i.e., the end point and after which, vias were used to redirect the path to the other side of the plate and eventually ending on it as well. This ensured us that the soldering vulnerability caused in the first iteration is mitigated and eventually, we were successfully able to create the PCB designs for our robot.

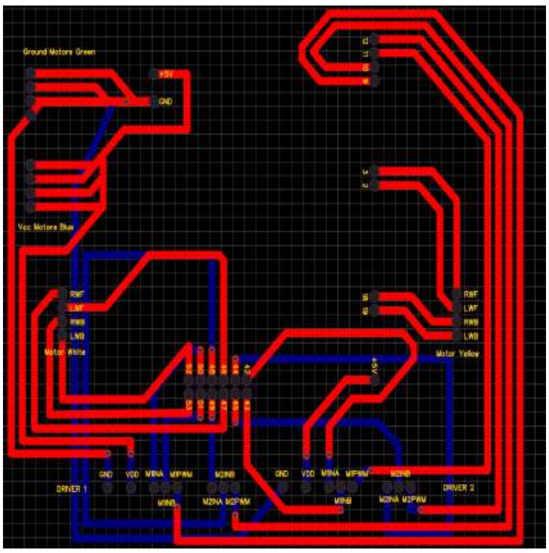


Figure 4.12: 2nd PCB design for Mega

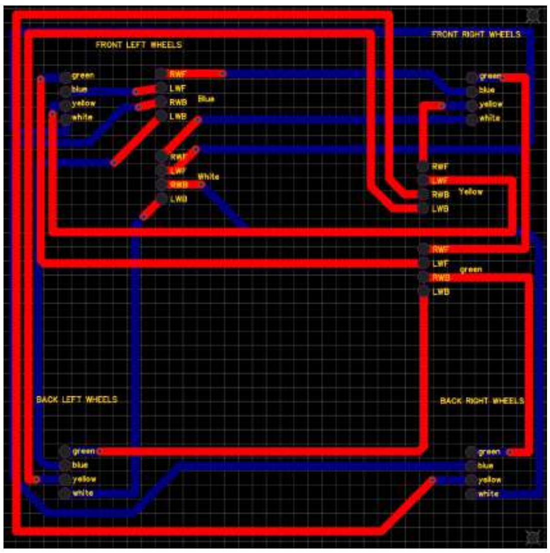


Figure 4.13: 2nd PCB Design for motors

Additionally, the parameters used in PCB designing are shown in table 4.1 that were used to check for any DRC errors in our design.

Table 4.1: Parameters in PCB Designing

Specification Rule	**Value**
Track Width	1.5mm
Clearance	0.4mm
Via Diameter	0.61mm
Via Drill Diameter	0.305mm
Routing Width	1.5mm

4.6 Mapping Environments

We employed two different methods in mapping the environment which are Hector SLAM and Gmapping. These two mapping methods were employed in order to create a comparative analysis between the two methods in order to realize the better method for our application. The two methods are discussed below:

4.6.1 Hector SLAM

Hector-SLAM is a map-making algorithm that leverages laser scan data. Hector-SLAM has an advantage over other SLAM approaches in that it simply needs laser scan data to work. It does not require odometry information. Simultaneous Localization and Mapping is what SLAM stands for. SLAM is a popular technique in which a robot creates a map of an unknown area while tracking its position within it. Thus, it helps in building maps without requiring the odometry data and for which, it estimates the position of the robot through the laserscans obtained from the environment. Therefore, in order to run the process, we only need to publish laser scan values on a ROS topic and subscribe the node of hector mapping to it. This simple tasks is responsible in generating the map and that can be viewed from the rviz software. The maps generated are usually very noisy in nature since the algorithm is dependent on rigorous state estimation and hence, always needs some editing via softwares like gimp. A hector slam generated map of one side of the projects lab is shown in the figure 3.16. We can see that the generated map have a lot of outlier points and that causes major errors in the costmaps that are used in the navigation of the robot.

4.6.2 Gmapping

Gmapping is a much more conventional way of creating maps as it requires a transform frame for the odometry data alongside the laser scans. The gmapping already uses the monte carlo filter onto the map so it mitigates the errors that are involved in it's mapping. Moreover, it was found that the mapping technique is much more accurate than the Hector-SLAM as it uses more data for it's estimation. A map generated for the cafeteria is shown in figure 3.17. From it, we can realize that the map formed is much more accurate than the Hector SLAM.

4.7 Control Model for Motors

4.7.1 Modelling Motor Control System

Using data from Datasheet

Here, during our calculations, our model will be based on the following parameters:

- E_a = Armature Voltage
- K_m = Motor back EMF constant
- K_t = Motor torque constant
- J_m = Armature Inertia
- R_a = Armature Resistance
- L_a = Armature Inductance

The values of this motor's R_a and L_a are obtained through impedance analyzer from the project "Development of a Cart-Mounted Inverted Pendulum Test Bench" from the last year's final year projects.

Table 4.2: Pololu DC Encoder Motor Specifications

Datasheet Parameters	**Data**
Rated Voltage	12V
Armature Resistance (R_a)	4.2234 ohms
Armature Inductance (L_a)	1.81mH

A common actuator in control systems is the DC motor. It directly provides rotary motion and, coupled with wheels or drums and cables, can provide translational motion. The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure Considering a simple DC Motor(Brushed, we first get the Armature voltage equation using the Figure 4.14 as:

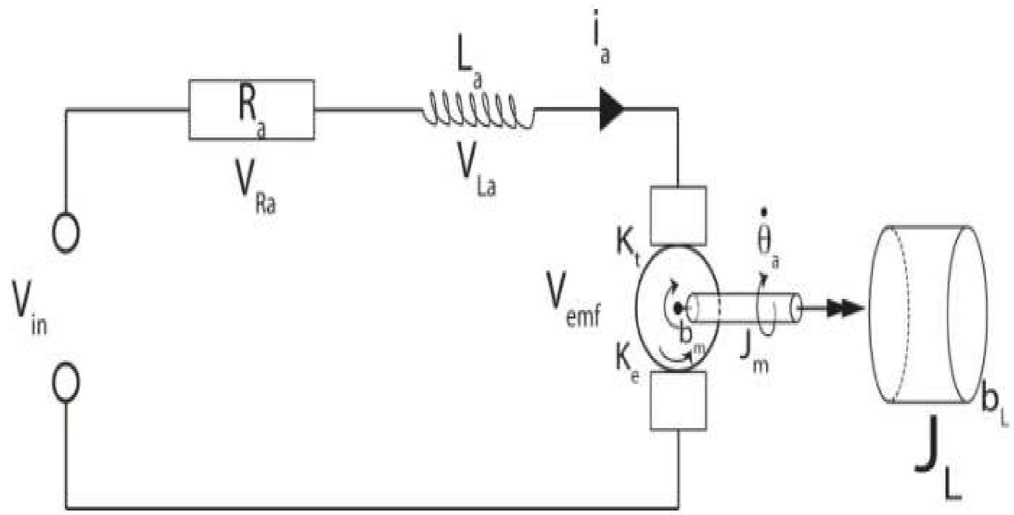


Figure 4.14: Internal Model of the Motor

$$V_a = R_a + L_a \frac{di}{dt} + k_m \omega$$

Where, k_m is a motor constant, ω is our angular velocity, V_a is our nominal Voltage, R_a is armature resistance and L_a is our armature inductance. Here, There are two other mechanical forces that govern how a DC motor system behaves. Since the system inertia (J) and friction constant (b) are typically a result of material properties and load characteristics of the system, how their values are determined will not be summarized in this application note. The motor itself will have a very small (if not negligible) inertia and friction constant.For this reason, their values are approximated using standard models of similarly-sized 12 Volt DC Motors. Since these values are incredibly small, the exact effect on the transfer function are relatively small compared to other system properties. This concludes the basic characterization of a DC motor. The following conditions were taken into consideration while evaluating the transfer function of the angular velocity:

$$K_m = K_t = J\alpha = J\frac{d\omega}{dt}$$

Substituting the above equation, we get:

$$V_a = R_a\frac{J}{K_t}\dot{\omega} + L_a\frac{J}{K_t}\ddot{\omega} + K_m\omega$$

Hence, for the angular velocity, we obtain the following transfer function:

$$\frac{\omega(s)}{V_a(s)} = \frac{K_t}{s^2L_aJ + sR_aJ + K_mK_t}$$

Grey-Box Modelling

Since the hardware sometimes shows different results from that given in datasheet due to wear and tear within itself, we felt a need for gray box modelling as well to get an overview of the motor model. Thus, we can derive a first order function directly as:

$$T(s) = \frac{K}{\tau s + 1}$$

We know that SI-units of Km and Kt are identical hence we can assume them as K, where K and τ are the the gain and mechanical time constant of DC motor model. Therefore, we can eliminate the second order term from the obtained tranfer function and get:

$$\frac{\omega(s)}{V_a(s)} = \frac{K_t}{sR_aJ + K_mK_t}$$

The relation between angular velocity and armature voltage are shown in figure 4.15

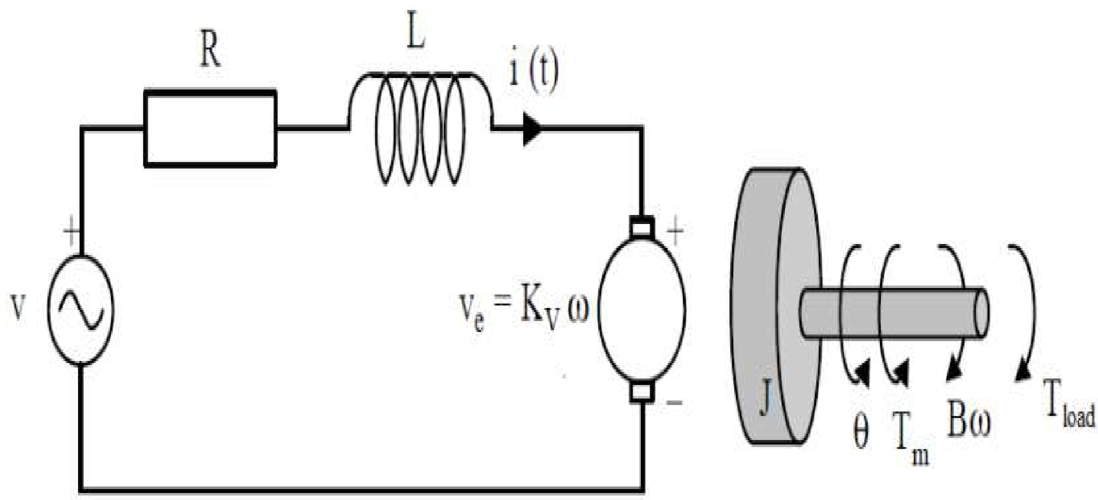


Figure 4.15: Relation between angular velocity and armature voltage of the motor

Here, $K_v = K_m$ and ω is the angular velocity. Thus, we can get it as:

$$K_m = \frac{E_a - I_aR_a}{\omega}$$

Table 4.3 shows the maximum value for each component as determined through measurements.

Table 4.3: Pololu DC Encoder Motor Parameters

Model Parameters	Value
E_a	12V
R_a	4.2234 ohms
ω_{noload}	16.05 rad/s
I_a	0.18 A

This helps us in determining the values for K_t and K_m as:

$$K_t = K_m = 0.7502\frac{Vs}{rad}$$

To obtain the values of voltage and current we used the digital multimeter and for angular velocity, we used a camera to obtain the revolution per second in slow motion and obtained the angular velocity in rad/sec by multiplying the revolutions with the conversion factor. Now, comparing the equations, we get the motor constant K as:

$$\tau = \frac{J_m R}{K_m K_t}$$

$$K = \frac{1}{K_1}$$

Here, τ is the 63.63% of the steady state speed for which the time for the motor is required to reach. This can be analyzed through the motor response model through the MATLAB. Nevertheless, we can obtain the value of it as:

$$\tau = 0.0986 \textit{ seconds}$$

Subsequently, we can obtain the value of J_m now as:

$$J_m = 0.013139Nm^2$$

Table 4.4 summarizes each and every parameter obtained in grey box modelling:

Table 4.4: Motor Gray Box Modelling Paramters

Parameters	**Value**
E_a	12V
R_a	4.2234 ohms
ω_{noload}	16.05 rad/s
I_a	0.18 A
τ	0.0986 s
K_m/K_t	0.7502 Vs/rad
J_m	0.013139 Nm^2

The transfer function obtained through the parameters by substituting the value and simplifying the equation is:

$$\frac{\omega}{E_a} = \frac{13.519}{s + 9.515}$$

The response of the transfer function is shown in figure 4.16 and it shows the rise time to it's maximum value and settling point as well.

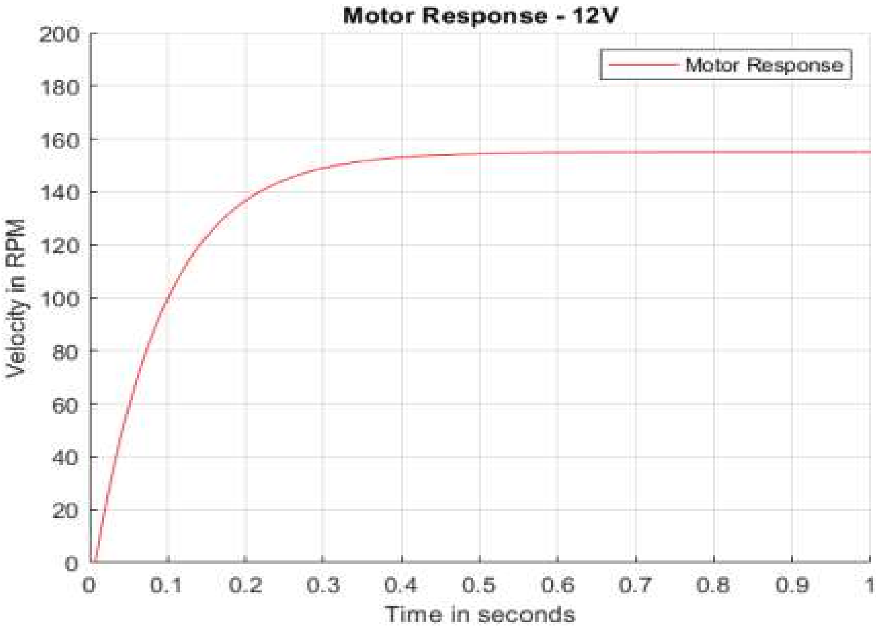


Figure 4.16: Motor Response to unit step voltage

4.7.2 Verification through Simulation

Without Disturbance Observations

Figure 4.17 model represents the simulation of the transfer function and obtain, also we validate our obtain transfer functions angular velocity response with the true response.As seen in the simulation below the disturbance was also added which is considered to be the load on on the platform, hence this was used to obtain the results for the validation and approximation of our grey-box model.

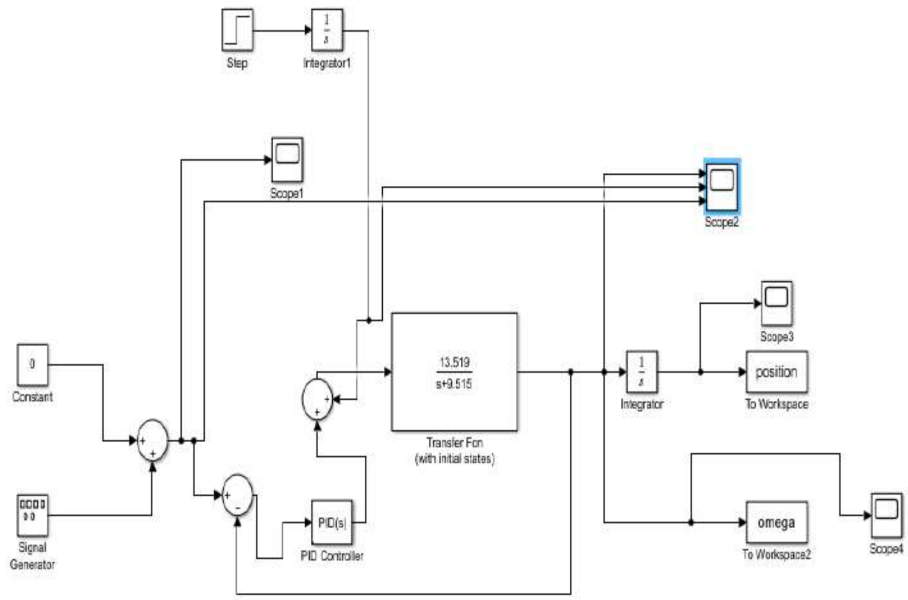


Figure 4.17: Motor Model Simulation Setup

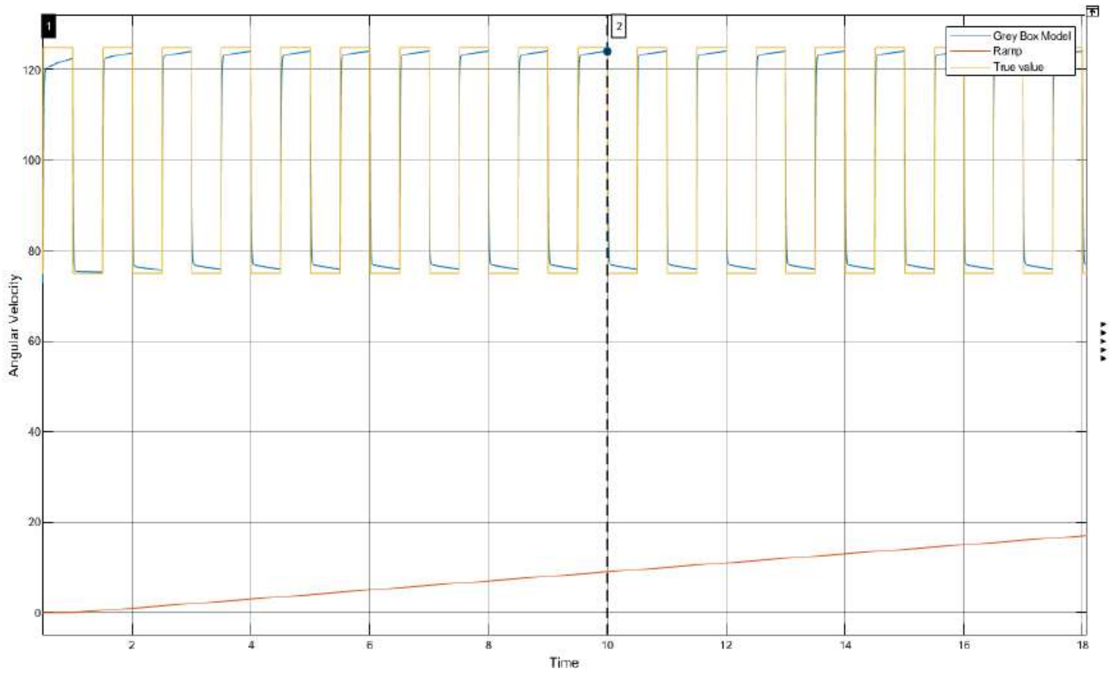


Figure 4.18: Motor Response to square input

An error of 3.3 percent was observed, when modelling both responses, the error margin is quite less as the values were obtained from different motor experiments, hence it was deduced that the grey-box modelling is presumed to be the closest approximation for the motor modelling against the true response. Therefore, we inclined towards varying the K_P, K_D, and K_I parameters of the model to see what changes does that bring into the system. The initial PID values are shown in figure 4.19

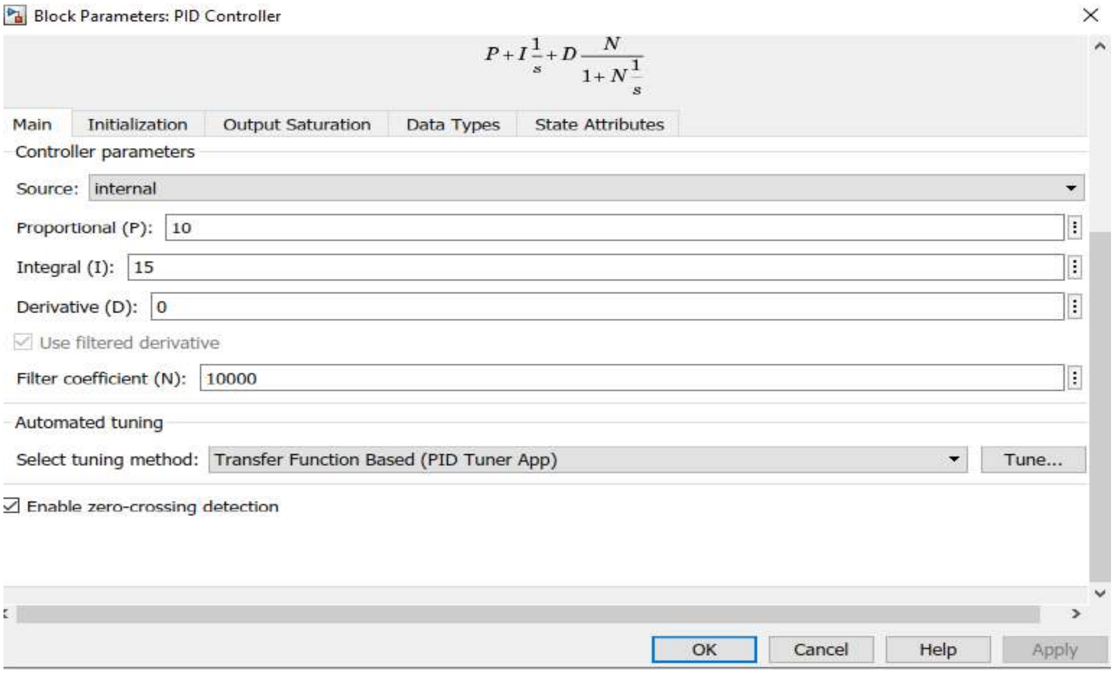


Figure 4.19: Initial values of PID controller

Now, we will increase the value for each parameter and also add some constant value to the K_D parameter. Using the values showed in figure 4.20, we obtained the result showed in 4.21

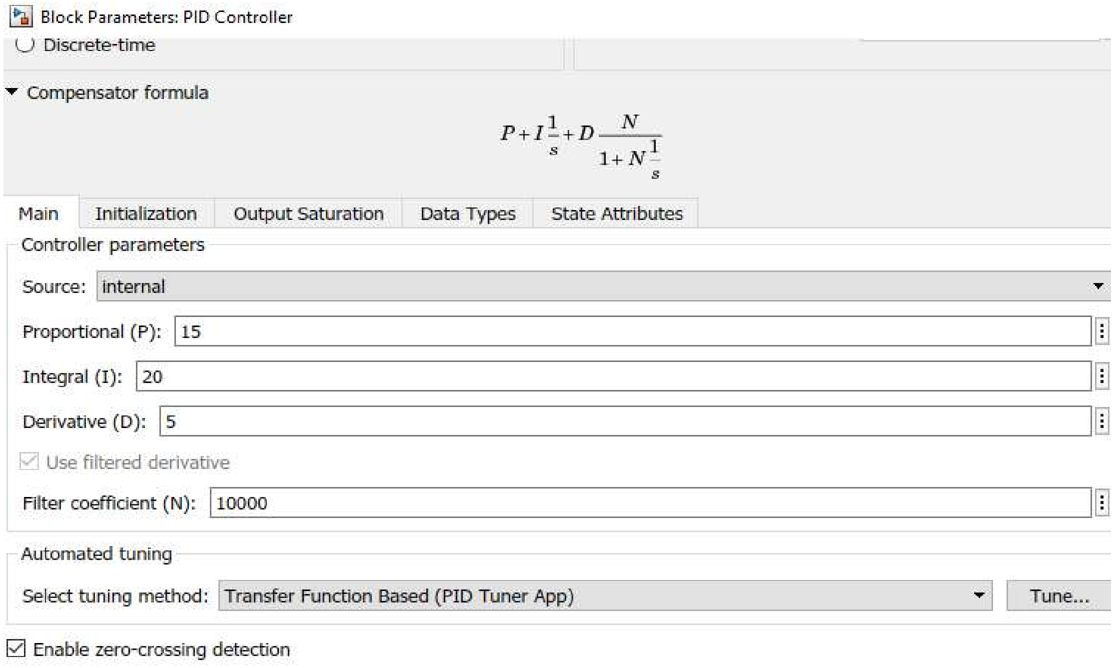


Figure 4.20: Second values of PID controller

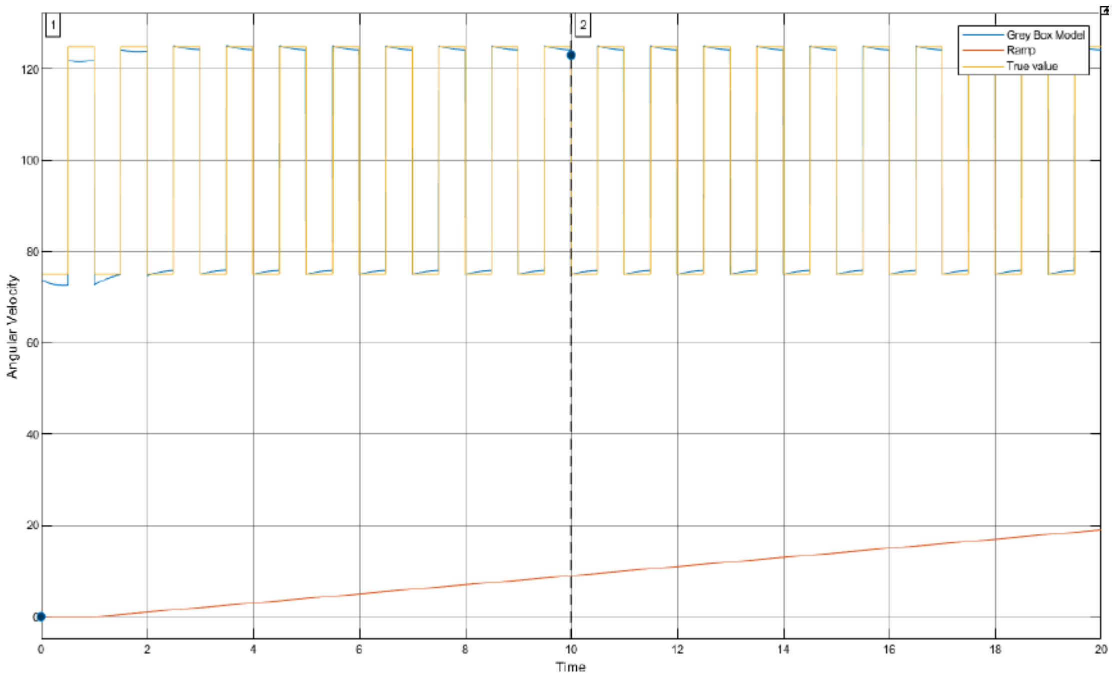


Figure 4.21: Response of model to second changes in PID controller

We can see that no major difference is observed and mostly, there is a difference observed in settling time and rise time of the signal but other things are pretty much the same. Hence, we now we tried lowering the values of PID controller compared to it's initial value as shown in figure 4.22 and we can observe that this time on figure 4.23, the signal is affected in it's settling point which is very important for us to be correct and therefore, we can conclude that using higher values of K_P is preferred whereas we can tune the values of K_D and K_I towards our desired results.

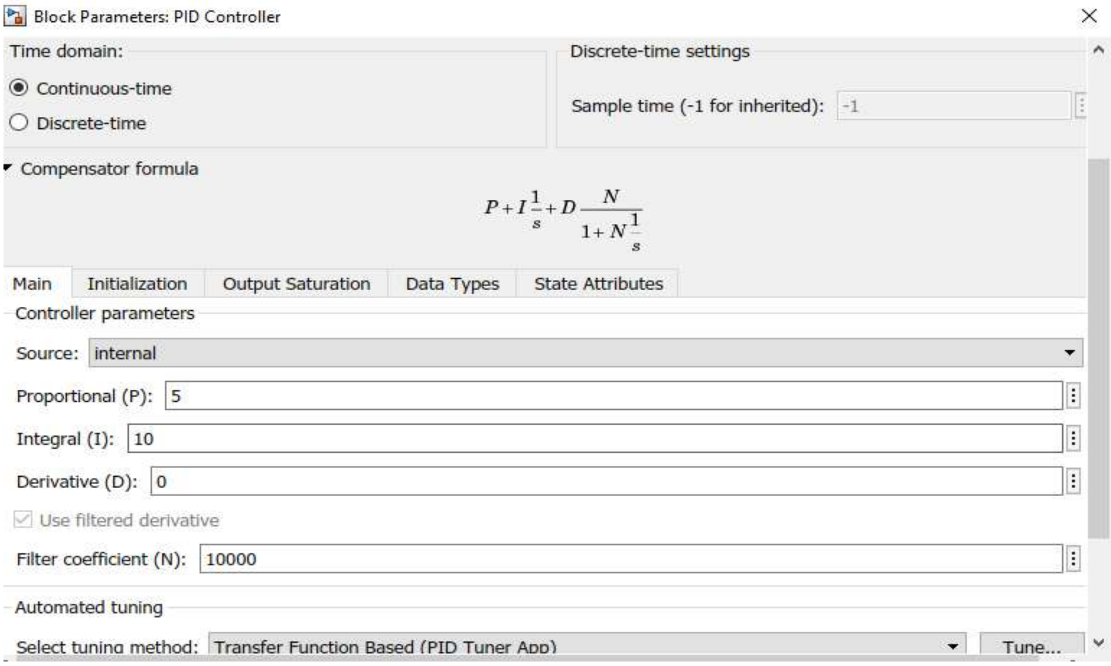


Figure 4.22: Third values of PID controller

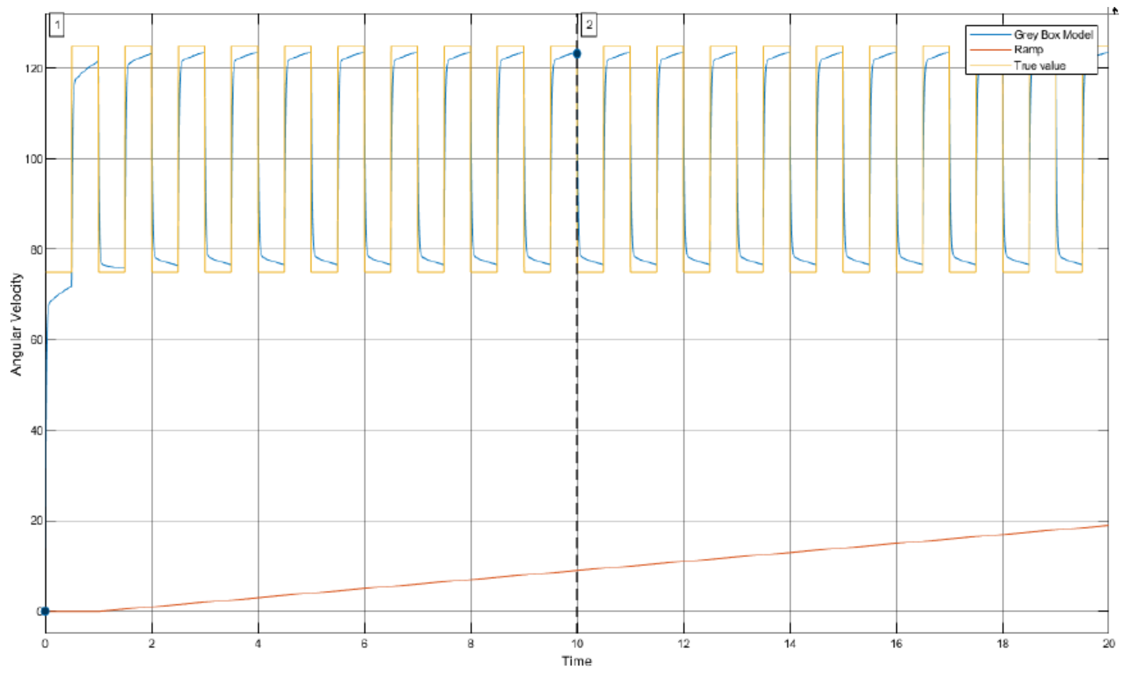


Figure 4.23: Response of model to third changes in PID controller

Adding Disturbance to the Model

Now, we will be adding simple disturbance to the model in order to see how it performs under those conditions. For now, we have added a unit ramp disturbance as shown in figure 4.24. Through it's addition, we can see the affects of it on figure 4.25.

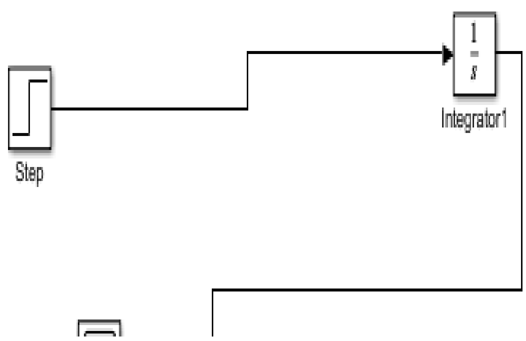


Figure 4.24: Blocks for adding ramp disturbance to the model

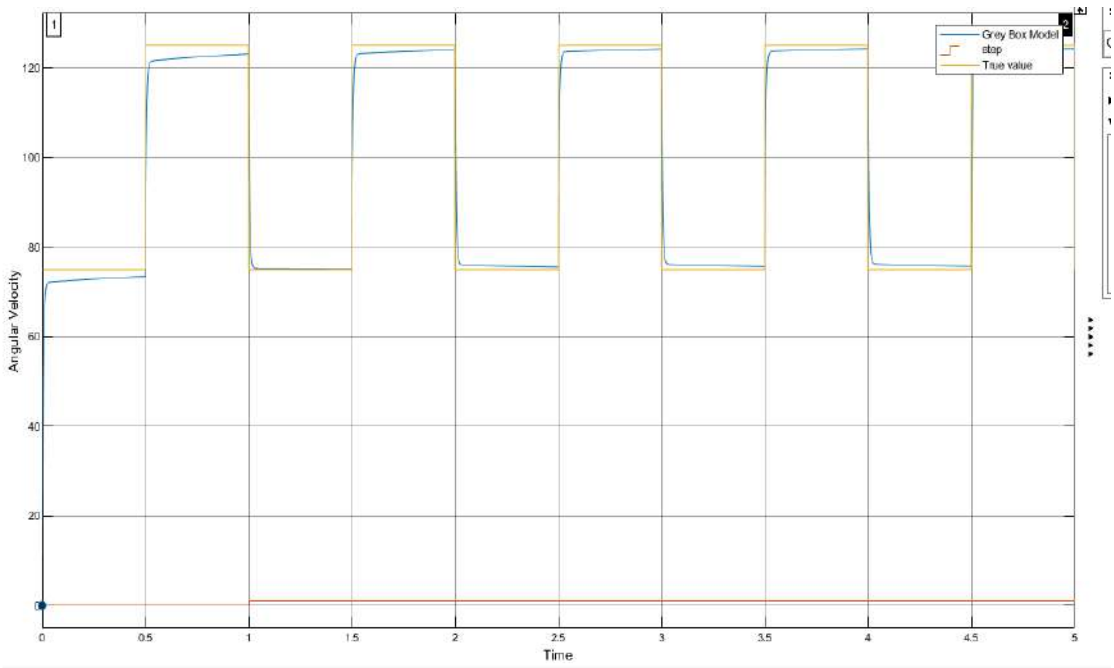


Figure 4.25: Affect of Adding Disturbance to the Model

We can see that the model is able to retain it's behavior in a much better way and this is due to the second order nature of the system. However, we will surely see big errors if the error applied to the system is increasing in it's rate as well but in nature, we won't see much of those errors affecting our practical system unless we model the make the robot go down a ramp that has increasing steepness which is very unlikely to happen in our application.

4.7.3 Practical Tuning of the Control Parameters

When it comes to setting the control parameters of the motors, we were able to use the model from simulation to create such behaviour that the motor was able to achieve desired results however, we figured that it required different adjustments for different motors to act same for a given signal. We observed that some motors were rotating faster than the others at a given PWM signal from the arduino. Therefore, we had to tune the motors on practical observation again as it was essential for us to have identical behavior from each and every motor in order to replicate a perfect differential drive behavior. Therefore, we used the data coming from the encoder to match the outputs of each motor and figure 4.26 shows how the data was published on the rostopics for each wheel.

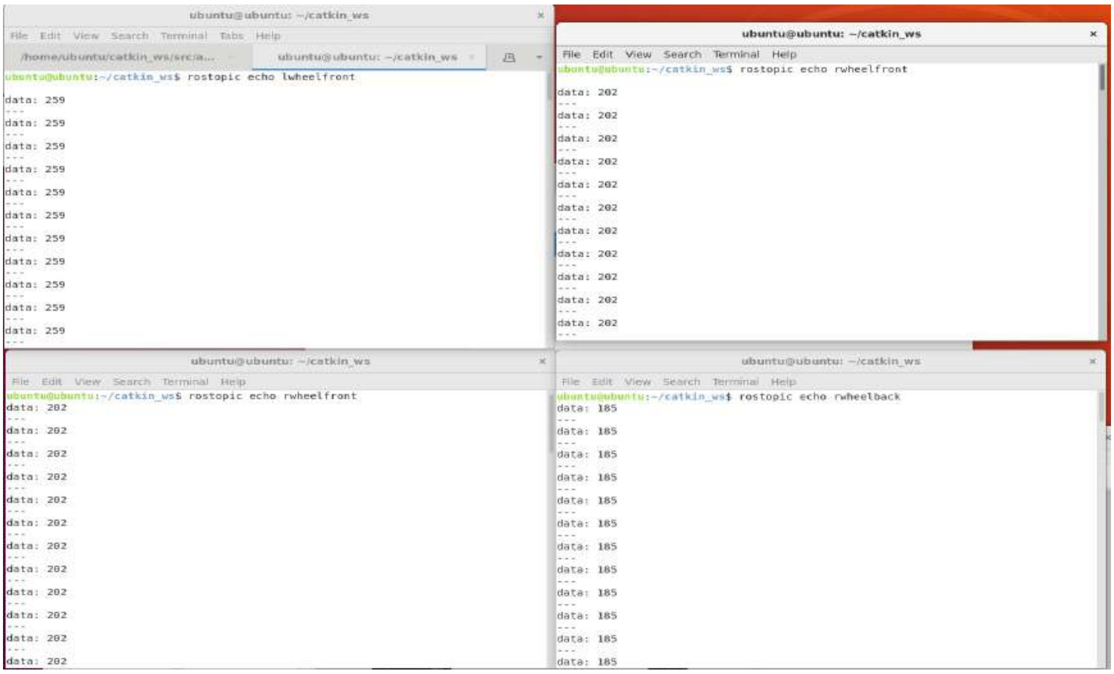


Figure 4.26: Odometry data published on ROS topics

Our goal was to match the values of each and every wheel such that whenever the robot moves in straight line i.e., each of it's wheel is moving at the same speed, the encoder values are yielding same value and the robot is visually moving straight as well.

To achieve this, we tuned the parameter for each wheel such that the robot first appeared to move straight when told i.e., there are no major biases. After that, we observed the outputs by plotting them using rqt plots of ROS and figure 4.27 shows the obtained result.

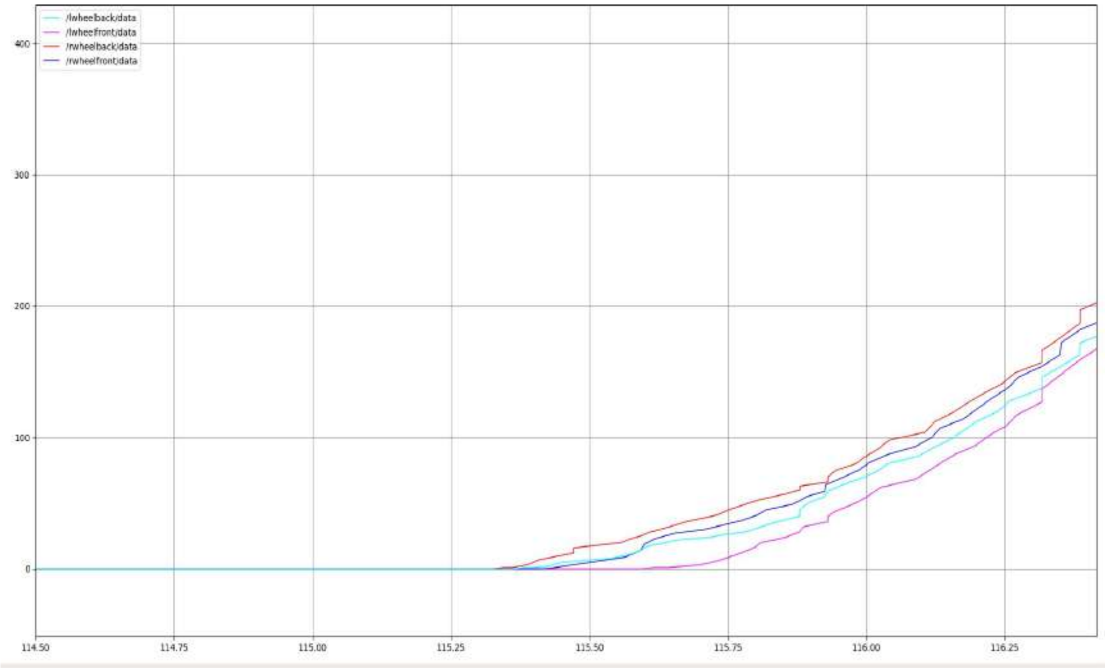


Figure 4.27: RQT plot after visual tuning

Here, we can still see that there still exist some biases in terms of rise time for each motor as that is not required since robot will frequently rise from zero speed and this phenomenon will cause jerks and eventually, big errors within the navigation. Hence, we further tuned the controller through increasing K_P gain to reduce rising time for affected motors and added K_D to it as well in order to reduce the overshoots in the system. This tuning yielded us the results shown in figure 4.28

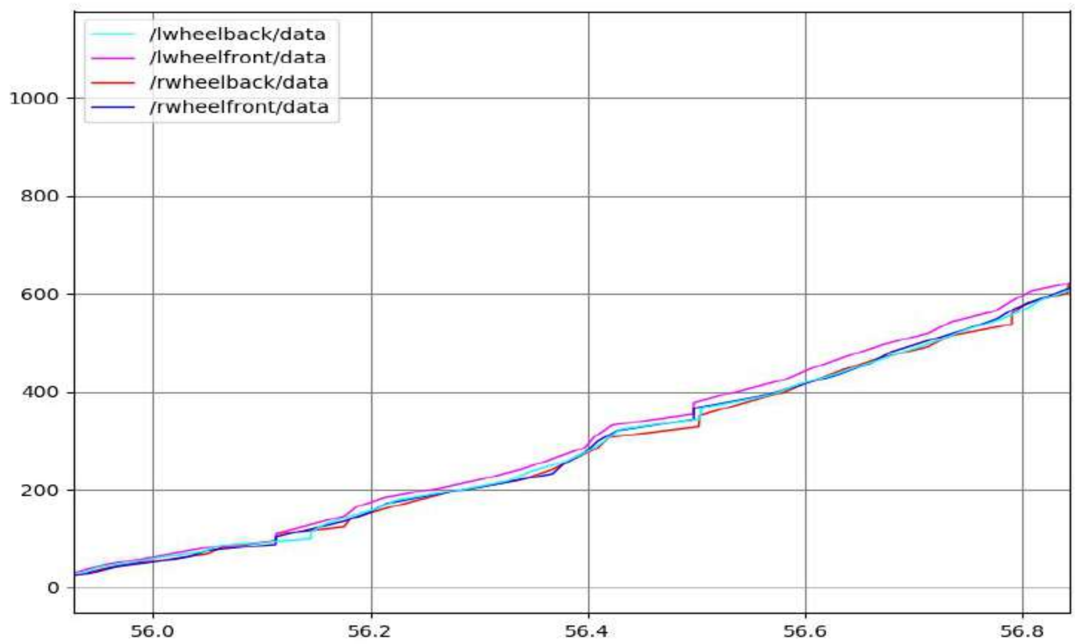


Figure 4.28: RQT plot after second tuning

Now, we can observe that the rise time is of each motor is very much identical to each other and there exist very few errors in it. Therefore, we proceeded onto giving it a constant speed for 1 second and observe it's behavior and as expected, the system is very stable now and the results for it are shown in figure 4.29. We can still see some settling point errors but those are very minimal and such results are obtained after rigorous controller tuning.

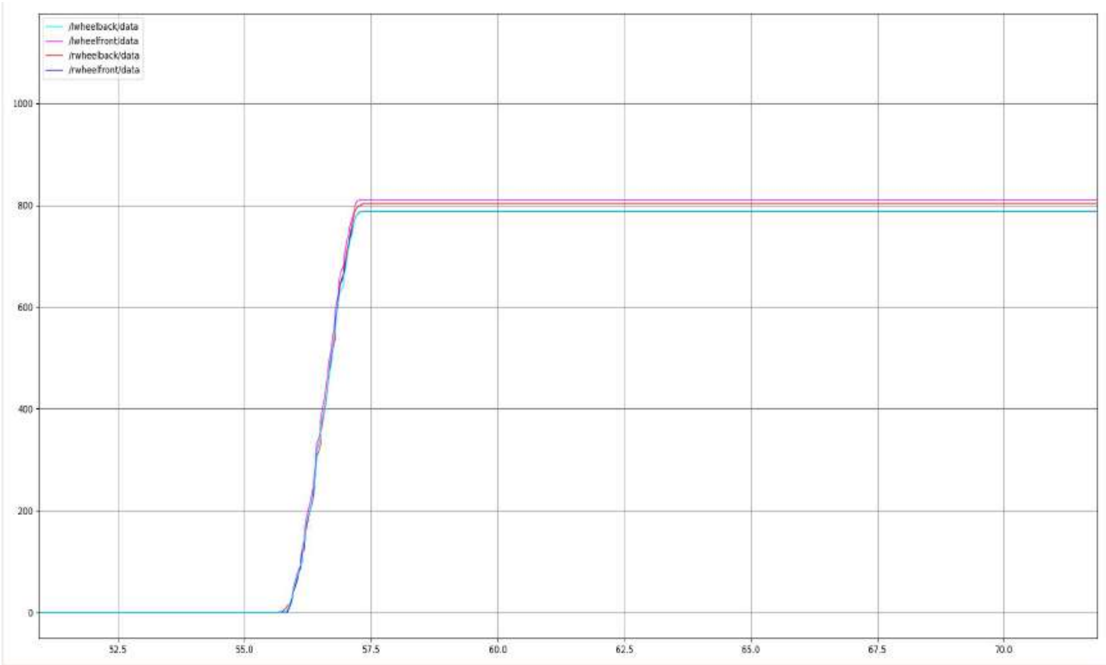


Figure 4.29: Response on Final Practical Tuning

4.8 Tuning Weight Instrument of Robot

4.8.1 Sensor Behavior

Since the lab inventory only had the load cell of 20KG range, we procured one load cell from the market with 5KG range and tested out different known weights on it to see how it's output is affected from it. Figure 4.30 shows how the output voltage was varied with the change in weight and it from it, we can notice that the characteristic obtained is very linear as well.

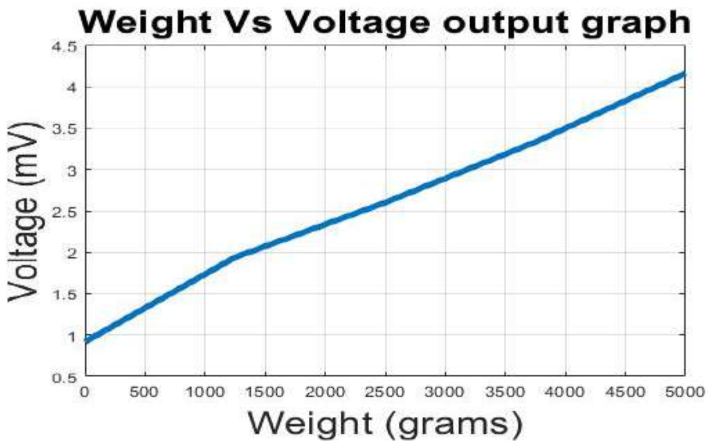


Figure 4.30: Behavioral change of load cell with change in weight

4.8.2 Sensor Mounting on Robot

Figure 4.31 and Figure 4.32 shows how the sensor is currently mounted onto the robot. It has a space at the center of the platform for the payload to rest and that enables uniform force applied on the sensor. For the FYP, we can see that when the project will be completed, it will allow the users to measure the weight before delivering the payload and thus, can avoid any overloading on the robot which may affect it's maneuverability. Here, we have noticed one problem that whenever we apply weight onto the plate, it typically bends the acrylic beneath it so to avoid that, it can be seen that the sensor is towards the middle side of the plate rather at the corner. Moreover, we were also advised that slight changes to the tightening of the sensor can make drastic changes to the result and since our robot is dynamic, it is very plausible that the sensor mount may face some disturbance and therefore, we plan onto applying some re-calibration method onto the signal processing part such that if the there occurs some errors on the sensitivity or the zero drift, the later part can be used to fix the issue.

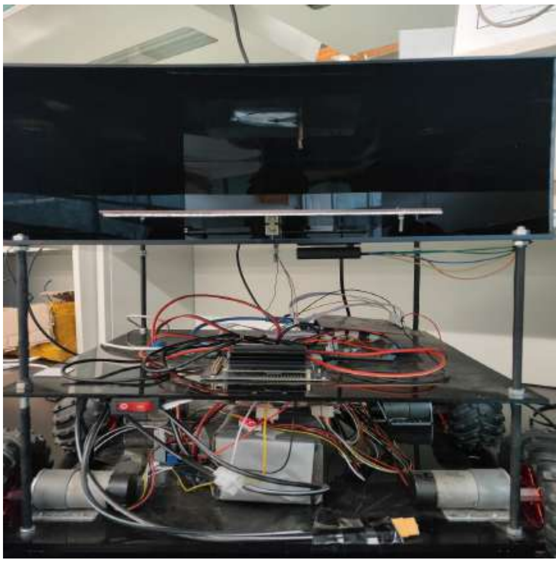


Figure 4.31: Sensor mount (Front View)

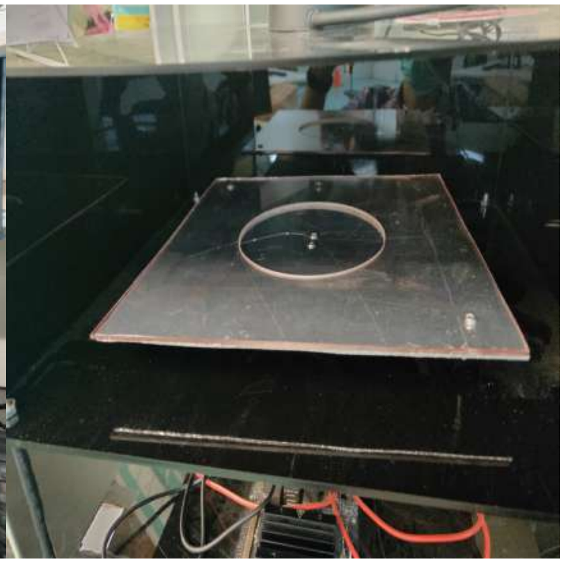


Figure 4.32: Sensor mount (Top View)

4.8.3 Applied Methodology

As previously discussed that there is a need of signal conditioning in reading the change in sensor values, we will need to amplify the measurement such that the noise is neglected as much as possible and an output signal is received such that it ranges from 0-5V such that it is readable from the arduino to process the signal further. From the specification table, we can confirm that the resistance of one of the strain gage is 1000 ohms from the output impedance. Through that, we can model the load cell on the simulation environment. The instrumentation amplifier would be used in this project for signal conditioning. The purpose of an instrumentation amplifier is to enhance extremely low magnitude signals while rejecting noise and interference. Furthermore, when it comes to differential amplifiers, the CMMR ratio of instrumentation amplifiers is quite high (section C.1.4), and the input impedance is theoretically infinite while the output impedance is zero. The gain equation of the instrumentation amplifier referred to figure C.4 is given as:

$$v_o = \frac{R_{F_2}}{R_2}\left(1 + 2\frac{R_{F_1}}{R_1}\right)(v_{I_1} - v_{I_2}) \tag{4.1}$$

From the specifications table of load cell, we can see that the rated voltage of it is 1mV/V meaning that if we will provide it 5V from arduino uno, then it will be giving 5mV at max load (i.e., 5Kg). Thus, we can use this to get the required gain (assuming 4V to be max voltage for arduino) as: (although the curve for sensor in figure 4.30 shows that it reaches till 4mV, we assume 5mV for simulation and nevertheless, we will be using a potentiometer to control gain in series with R_1 when shifting to hardware.)

$$G_{req} = \frac{4V}{5mV} = 800 = \frac{R_{F_2}}{R_2}\left(1 + 2\frac{R_{F_1}}{R_1}\right) \tag{4.2}$$

Using this, we can first assume the values of few resistors and calculate for the others. So, assuming $\frac{R_{F_1}}{R_1} = 0.5$, we get:

$$\frac{R_{F_2}}{R_2} = \frac{800}{2} = 400 \tag{4.3}$$

This allows us to take the following values for the resistors.

$$R_1 = 2K\Omega \;:::\; R_{F_1} = 1K\Omega \;:::\; R_2 = 250\Omega \;:::\; R_{F_2} = 100K\Omega$$

The final schematic is shown in Figure 4.33

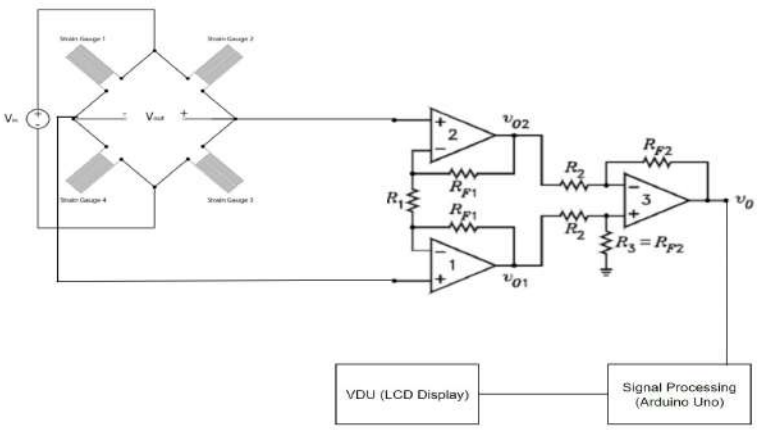


Figure 4.33: Final Schematic of the Instrument

However when simulating, it came out that the output voltage was a bit different compared to what was predicted and that is due to the non-ideal conditions of op-amp within the lm741 IC. However, we have planned towards giving the possibility of calibration in our instrument so we have not yet tuned the resistor values to achieve better results. Moreover, We also realized that the saturation voltage of each op-amp also contributes in this. If we assume that we are not using a boost convertor, we are restricted towards using a 5V as a saturation point for the op-amp and therefore, we have used that in our simulation to get the output and it also plays a role in the output error we are getting. Nevertheless, Figure 4.34 shows the simulation results at zero weight and Figure 4.35 shows the simulation results at max weight (i.e., 5KG). Here, we can see that

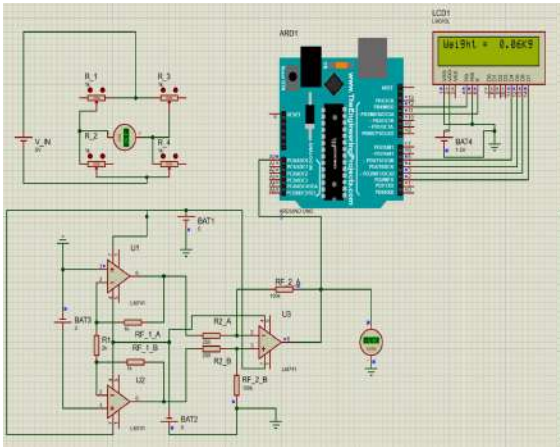


Figure 4.34: Output at zero weight

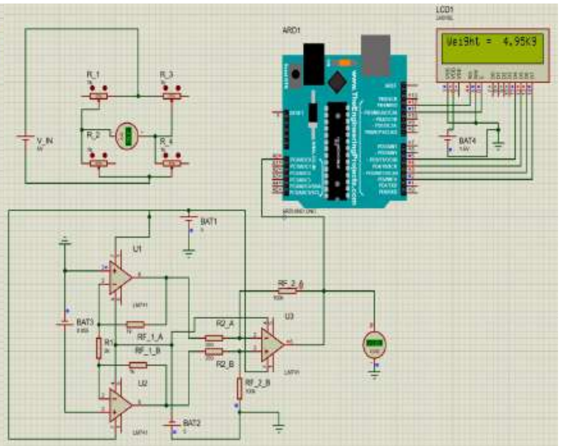


Figure 4.35: Output at maximum weight

there is a certain offset at zero weight and a certain offset at maximum weight. For this, we can set an ISR within signal processing unit to ensure that there is a button to avoid zero error and also a re calibration method (which is tentative) so that we can get two points for the straight line equation (1) and then re calibrate our device based on that. Also, it can be noticed from here that we have not used the output from the load cell bridge directly since it was hard to realize the change in resistance when certain load was applied onto it. Therefore, we applied the input directly onto the instrumentation amplifier.

4.9 Tuning Navigation & Obstacle Avoidance

Before going into the navigation, figure 4.36 shows the whole infrastructure of applications running inside ROS. We can see that there are particular nodes working for each of the tasks which are discussed earlier i.e., wheel odometers, PID tuners, amcl localization, etc. and this figure gives an overview of how the communication is being held between different nodes of the robot. This sort of infrastructure helps in modelling the data acquisition, data transferring, transformations, and processing within the roscore and hence, acts as the brain of the robot.

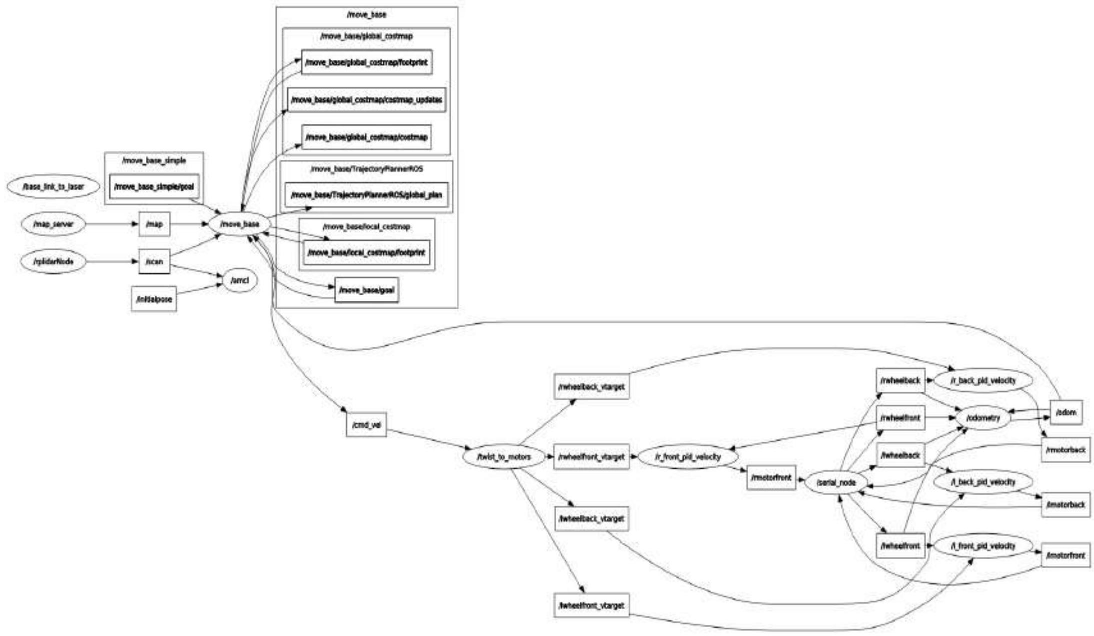


Figure 4.36: Final RQT Graph

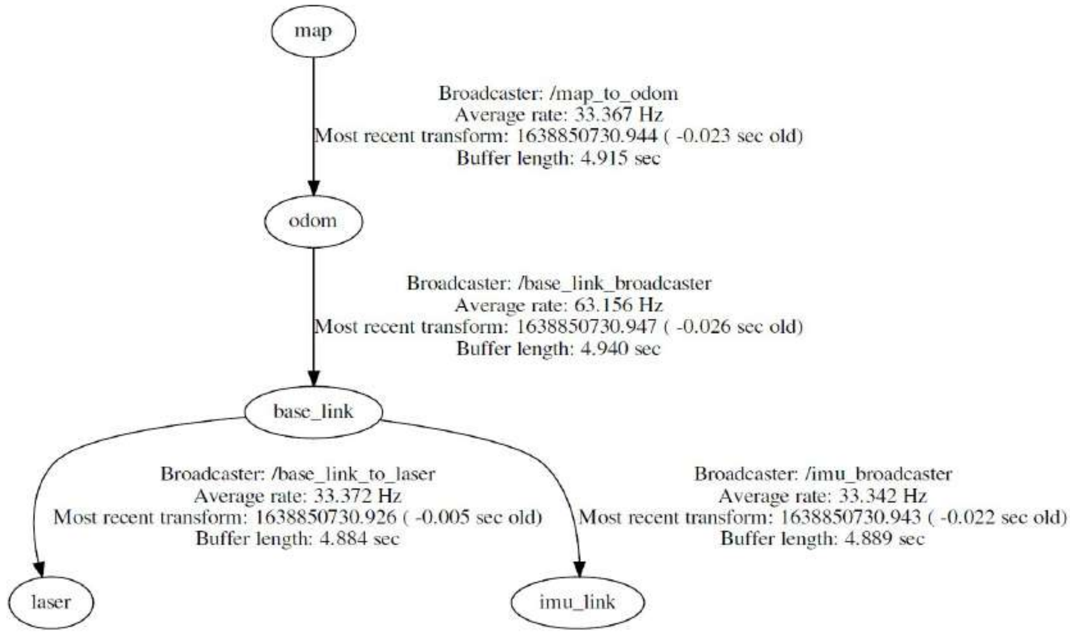


Figure 4.37: Final Tranform frame correlation diagram

The top left blocks in the figure 4.36 is for the navigation nodes and from the package movebase. it uses different parameters to help the robot navigate from the initial point to the goal point. This is how we have formed the navigation stack for our robot.

4.9.1 LIDAR Data Perception

In order to percieve the environment (even in known maps), the robot uses the laser scans all the time to detect realtime obstacles and to get a better estimate of it's position on the map through amcl. figure 4.38 shows how such environment is perceived by the robot and here, we can see that the closer the objects are, the more accurate data is being gathered by the laserscans and thus, we will be using a threshold for the laserscans to detect obstacles within a specific range as that will decrease computation and increase the accuracy of robot's navigation

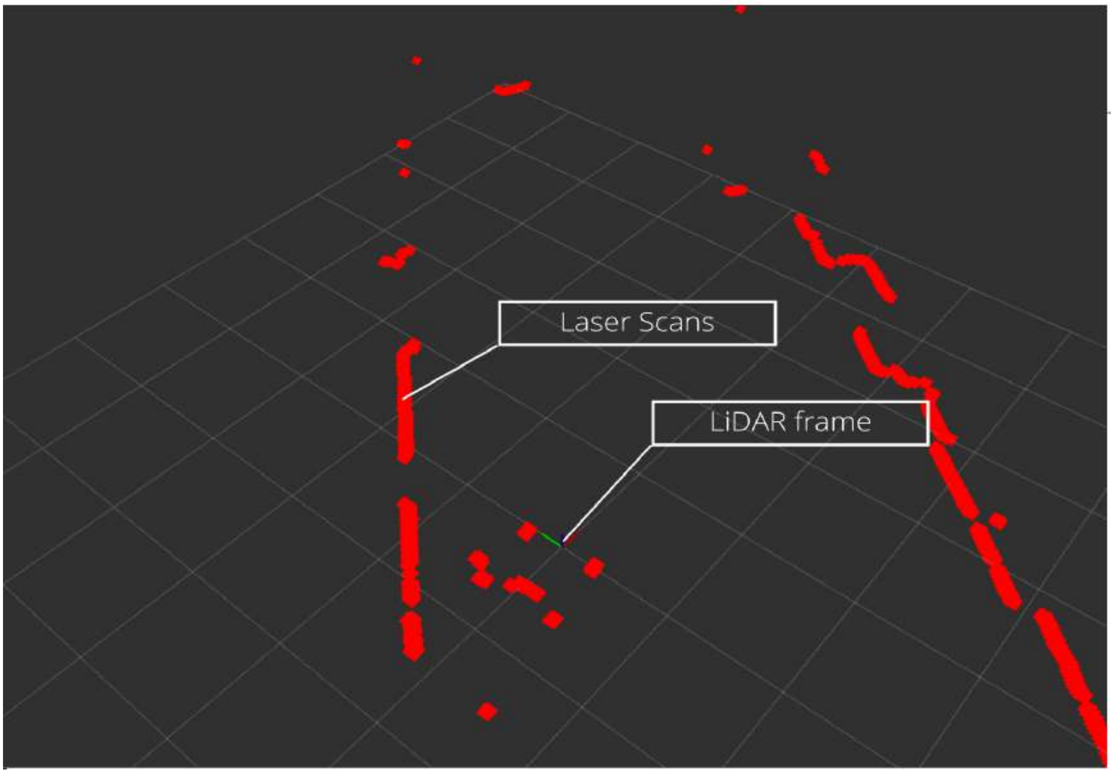


Figure 4.38: Data recieved from LIDAR

4.9.2 Parameters Functionality

Since we have the sensor data for obstacle detection, we can use this to detect obstacles in the environment. The movebase package allows us to set different parameters for the navigation and these are stored in the yaml files of the parameters. There are four different yaml files that we need to set for the robot which are:

- base_local_planner_params
- costmap_common_params
- global_costmap_params
- local_costmap_params

These files have different parameters to set the navigation for the robot. Here, we will mostly discuss the parameters involved in the base_local_planner_params as they are most essential ones when talking about navigation. The parameters are shown in the table 4.5.

Table 4.5: Robot Configuration Parameters

Parameters	**Functionality**
acc_lim_x	x acceleration limit
acc_lim_y	y acceleration limit
acc_lim_theta	Rotational acceleration limit
max_vel_x	Maximum forward velocity
min_vel_x	Minimum forward velocity
max_vel_theta	Maximum rotational velocity
min_vel_x	Minimum rotational velocity
min_in_place_vel_theta	Minimum rotational velocity during in-place rotations

Other parameters that are used in defining the goal point tolerances and controller frequency are shown in table 4.6

Table 4.6: Goal Configuration Parameters

Parameters	**Functionality**
yaw_goal_tolerance	Tolerance in radians when achieving goal
xy_goal_tolerance	Tolerance in meters (x,y) when achieving goal
sim_time	Amount of time to forward simulate trajectories
controller_frequency	Controller call in Hz

Moving ahead, we have costmap_common_params to set the parameters defining the dimensions of the robot with respect to the origin of its transform frame and how it perceives the obstacles. table **??** shows some important parameters that needed to be defined.

Table 4.7: Costmap Configuration Parameters

Parameters	Functionality
Inflation layer	adds layer to obstacles for extra safety
obstacle_range	distance till which obstacles are marked (in meters)
raytrace_range	Clears distance ahead of the sensor
footprint	defining dimensions of robot base

Now, figure 4.39 shows how a costmap is developed using the map and above parameters for navigation purposes. We can see that it uses the lidar scans and the predefined map to detect and mark the obstacles such that they are avoided during the navigation.

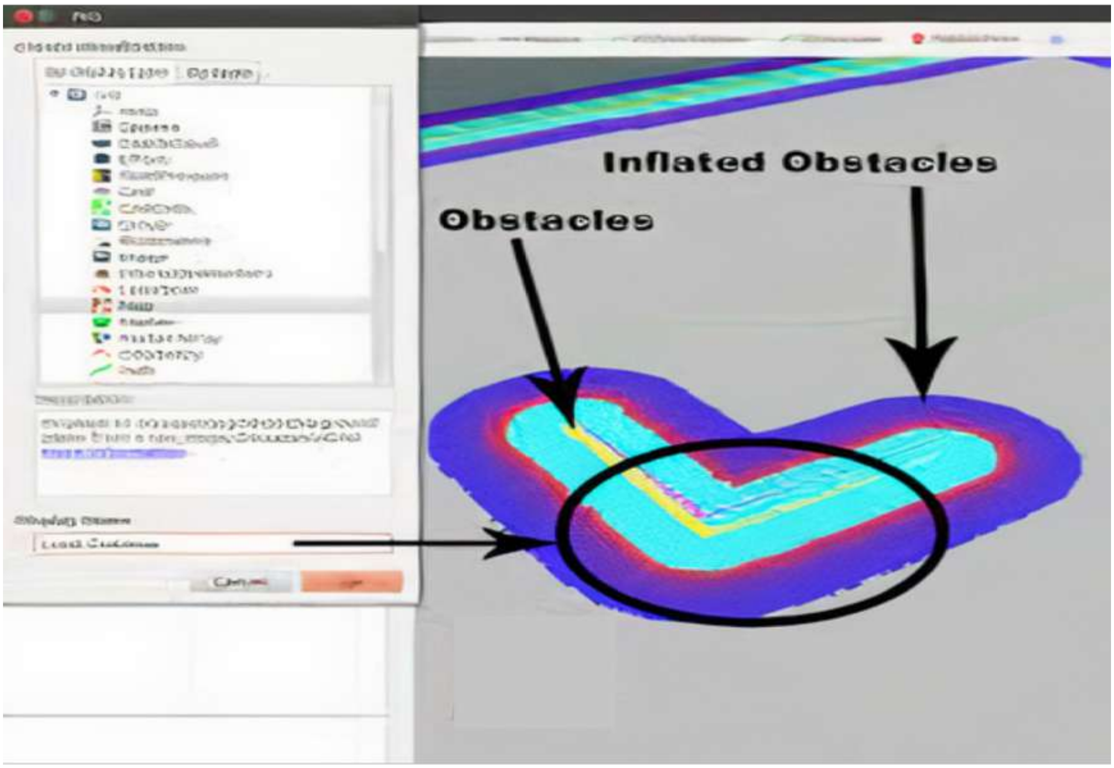


Figure 4.39: Costmap formation

4.9.3 Testing

Now, it time to test navigation system that we have created. We can simply use the rviz visualization tool to send our robot goal locations and visually analyze how it reaches the goal locations. Now, we have used this convention to tune the parameters as well and this helped us fine tuning the parameters as well. Moreover, when testing out the navigation processes, figure 4.40 shows how each of the defined paramter is collaborating. We can see that laserscans are detecting the walls whilst being dependent on the coordinate frames. The global map is used to navigate the robot towards the goal point (defined through rviz goal publish tool), and the robot is directed towards the goal location shown by the next pose estimate generated by the monte carlo particle filter (i.e., amcl). Hence, we can confirm that the desired operations are achieved.

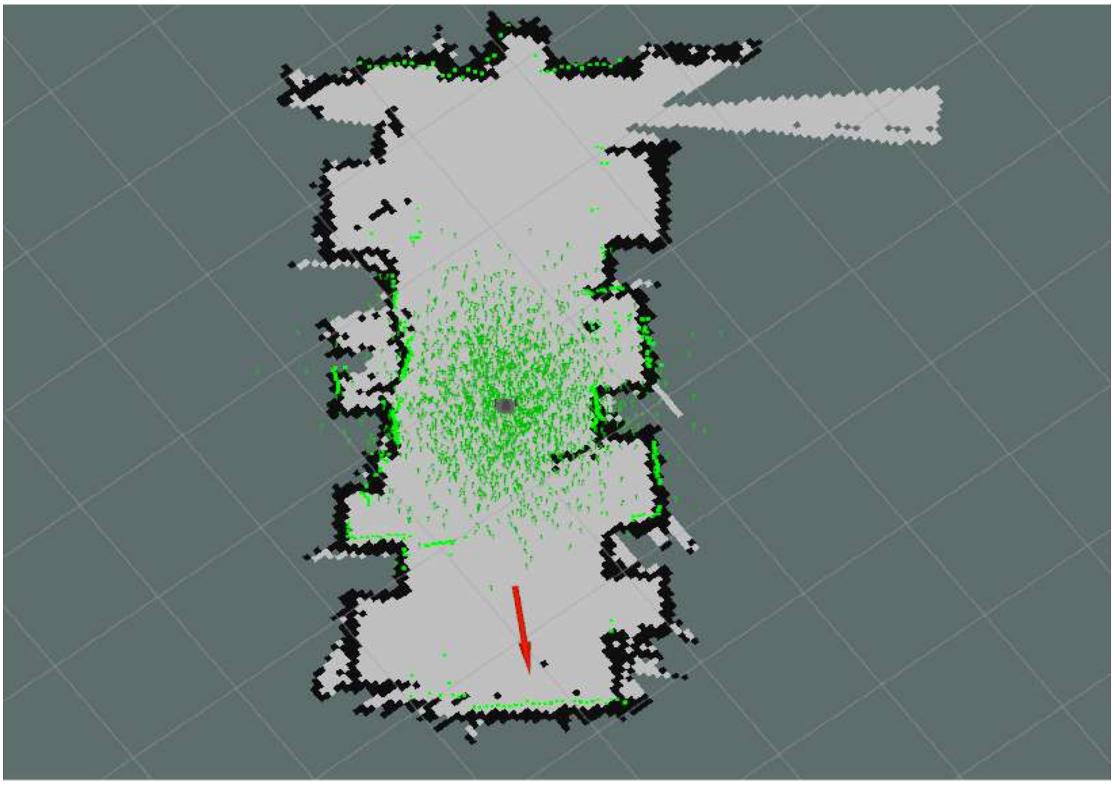


Figure 4.40: Operation during navigation

Now evaluating on results, we tested out this behavior for same point with different number of obstacles in it's sight at different configurations. Using these performance parameters, we were able to mostly navigate the robot in easier test cases however in complex environment, the robot seemed to not being able to find path towards the goal or either hit an obstacle (when suddenly bought in front of it). Firstly, figure 4.41 shows the time it took robot to initialize from it's point and reach the goal once the goal location is given to it. This sort of navigation was mostly free of errors since it is least challenging for the robot. Moreover, we can see that the robot is covers the distance which is in straighter line faster and is mostly concerned in aligning itself with the direction towards the goal first.

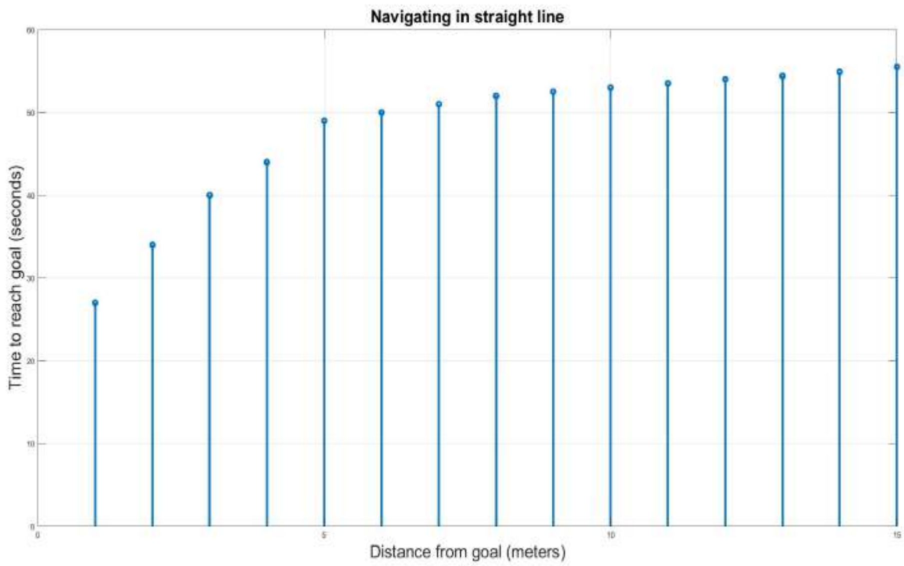


Figure 4.41: Navigating the robot in straight line

Now, we will be testing how it performs while taking turns and for this, we marked a point such that it is approximately 10 meters away from the robot rach time but includes different amount of turns. For extreme cases where a large number of turns were required, we insisted the robot to move between two seperate points such that the number of turns were achieved i.e., moving the robot from point O to A which will take 2 turns and then from point A to B which will take 1 turn only so in total we get 3 turns. Such experimentation was done and the results are shown in figure 4.42. Here, we can see that there is a pretty much linear relationship with the number of turns and the amount of time robot spends navigating. So this concludes that the non-linear behavior of the robot in figure 4.41 was mainly due to the turn it takes in initializing itself otherwise, the relation between moving from point to point in straight line is pretty much linear.

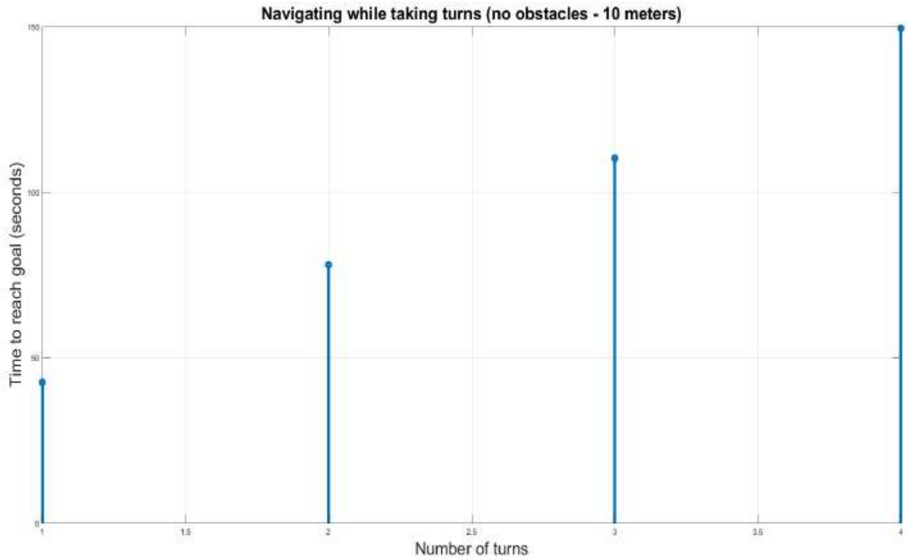


Figure 4.42: Navigating the robot while taking turns

Coming onto some complex testcases, we marked setups with similar sort of environment for the most part however, when we used obstacles, the behavior of robot was very unpredictable given that when the obstacles were spread sparsely with larger distances between them, the robot was able to navigate between them pretty easily whereas if the obstacles were very close, the navigation was taking unusually longer time. This correlates to the parameters we discussed in section 4.9.2 as if we increase the obstacle ranging parameter, the robot would cater it differently and that would lead to robot sometimes colliding with the obstacle or not being able to find a path. Therefore, it was not possible to make testcases in order to test the robot's maneuverability amongst different types of obstacles. However, we set up some obstacles as to block the pathway of the robot in parallel to each other when robot had to move in straight line and the results for that are shown in figure 4.44. The setup that we followed for testing obstacle avoidance is shown in figure 4.43.

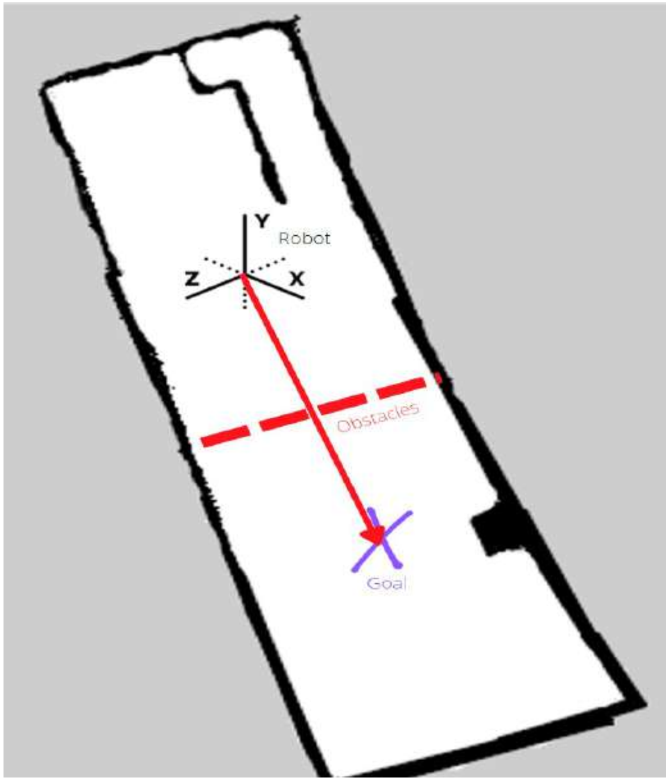


Figure 4.43: Testcase setup for the obstacle detection

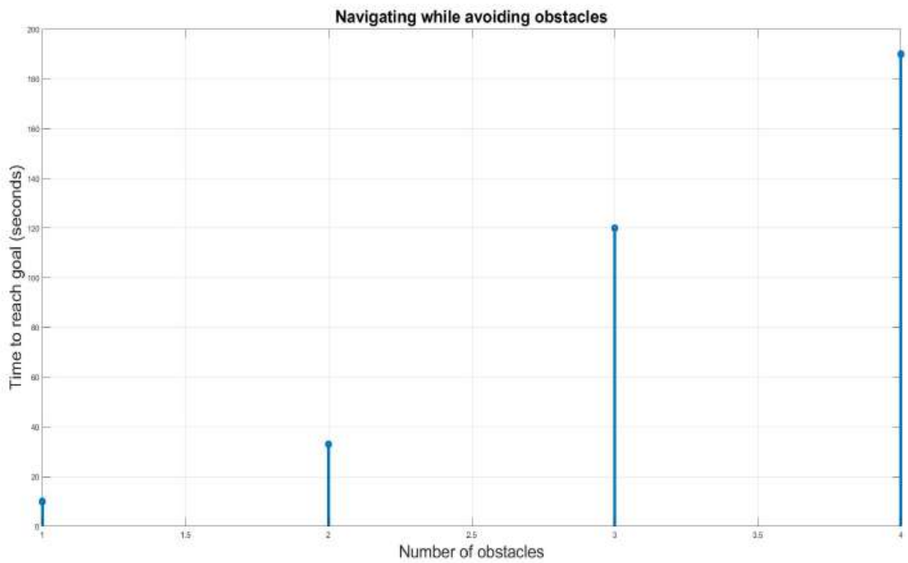


Figure 4.44: Navigating the robot while avoiding obstacles

From the results, we can see that the robot is taking more time to navigate from an obstacle overloaded path however, there seems to be an exponential increase in the time it takes for the robot to navigate but we think that this can not be concluded given that the testcase is very specific and do not cater much practicality. Now, we also analyzed some of the testcases where the robot was not able to navigate and was going into the routine that clears the costmaps and restarts the navigation repeatedly and these scenarios are highlighted in points below:

- The obstacles were placed in front of the robot such that less than 2 meters of space is given to the robot for navigation between them
- Something is thrown in front of the robot at fast pace
- Obstacles are present in front of robot such that they are below LIDAR level
- An obstacle comes in the way of robot's navigation suddenly (less than 2 seconds)

4.10 Mobile Application UI

The mobile application for this project was primarily created using the Flutter SDK created by Google. It utilises the programming language Dart for its frontend development and backend integration. The backend of the application is made using Firebase which is an online NoSQL database also created by Google. Both of these have excellent integration with each other as they were created by the same parent company. The UI of the application went through several iterations based on feedback by fellow peers. Following figures shows the screens of the mobile application:

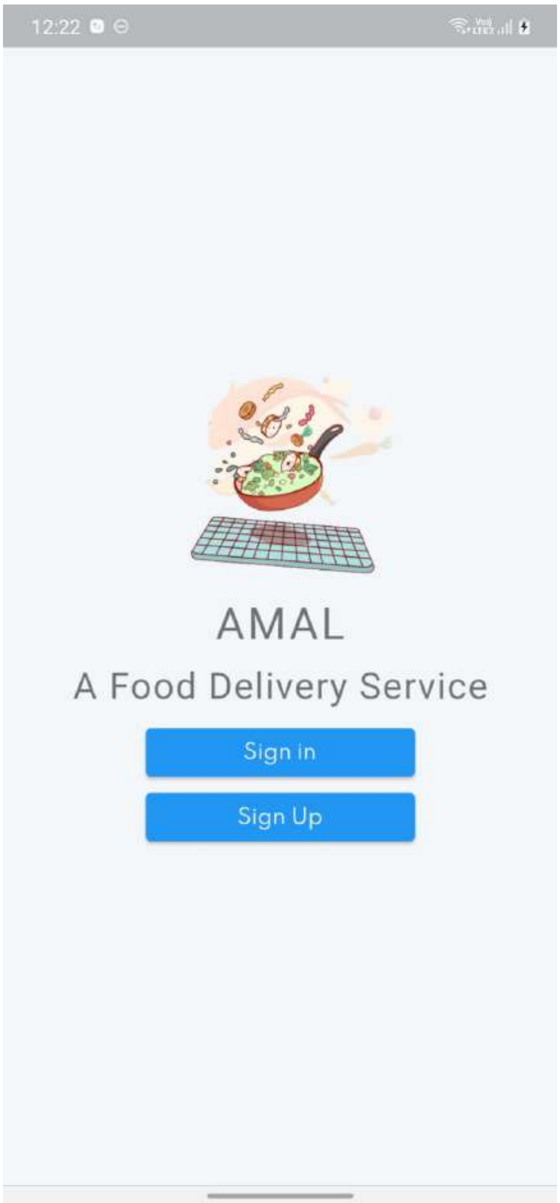


Figure 4.45: Home Screen

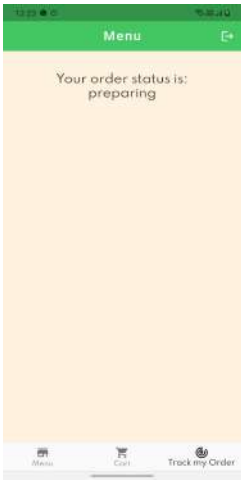


Figure 4.46: Status Screen

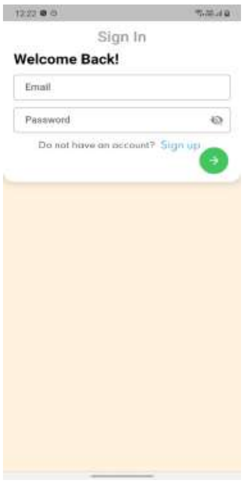


Figure 4.47: Signin Screen

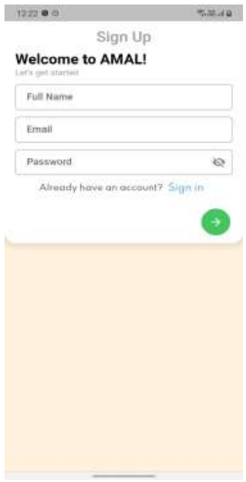


Figure 4.48: Signup Screen

Figure 4.49: Menu Screen

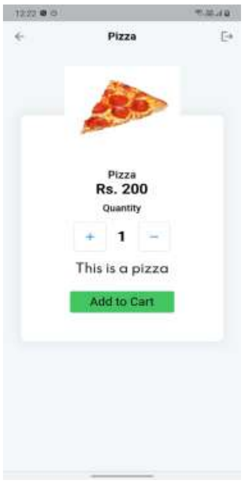


Figure 4.50: Item Screen

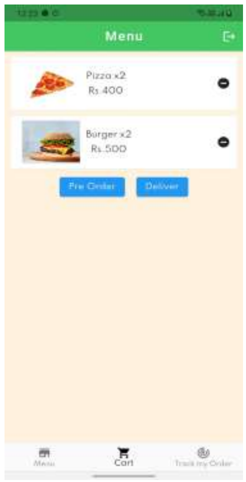
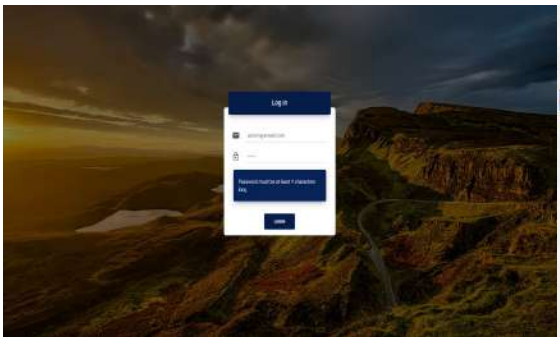


Figure 4.51: Cart Screen

4.11 Web Application Dashboard

The dashboard of the application was necessary to provide admin controls to the operator such that the order status and tracking is updated through it. This also enables a complete connection between the application and the software backend. It is also essential for managing orders and users in our application. Following figures displays the screens that are developed as a UI for the web application:



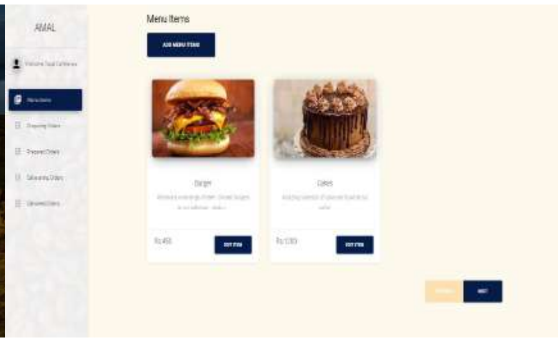


Figure 4.52: Menu Screen - Web

Figure 4.53: Item Screen - Web

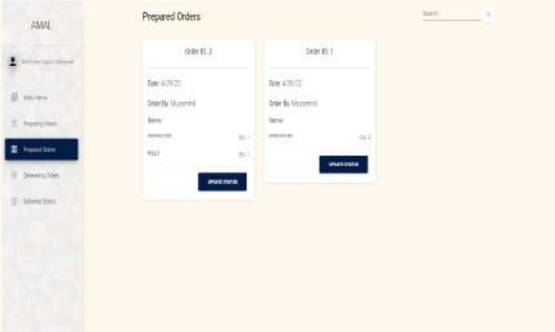


Figure 4.54: Preparing Screen - Web

Figure 4.55: Prepared Screen - Web

Figure 4.56: Delivered Screen - Web

Figure 4.57: Delivering Screen - Web

4.12 Setting up Connection between Application & ROS

While setting up the connection between the mobile application and ROS, we had to make sure that the connection is real-time and does not cause any major delays. Connection had to be setup with firebase and ROS and for that purpose, we researched on different ROS packages that enable a connection either directly or indirectly between them. However, each of the explored packages had some integration errors or unwanted results that were forcing us to avoid using them. table 4.8 shows those packages and the reasons they were excluded.

Table 4.8: ROS packages for application integration with robot

Packages	**Major Problem**
ekorobe_firebase	Communication delay of more than 7 seconds
iot_bridge	Not integratable with navigation stack due to language versions
ros_firebase_pusher	Not supported on newer version of firebase

In order to mitigate this issue, we created our own ROS package that created a linkage between the Firebase and ROSCORE. This required us to learn the concepts involved in these two platforms in depth and making the whole process much more efficient as well. Using them, we were able to produce remarkable results that only had a 220 ms of time delay between the whole communication given that the internet traffic is low. The files and packages that were formed by us are given in section B.2. So now, whenever a order is placed, the application push the order details on the firebase and the robot is able to retrieve the data from there. The data that both platforms uses to communicate can be visualized on firebase in realtime and is shown in figure 4.58

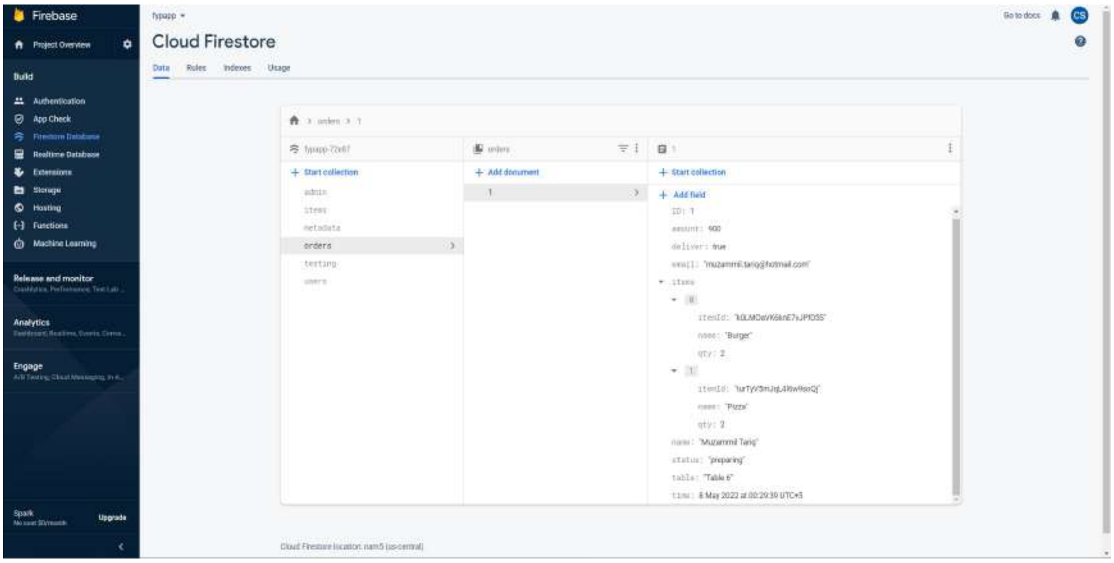


Figure 4.58: Firebase dashboard & data visualization

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

One of the project's primary objectives has been achieved: the construction of a fully automated delivery solution that includes an app, a back-end server, and an autonomous robot. The robot's most prominent features include:

- A four-wheel differential drive based robot that can develop a map of its surroundings and can travel autonomously using a feedback loop controller within an area, given a known map and reference coordinates.
- An embedded system platform that includes both high-level(Jetson Nano) and low-level (Arduino Mega) controllers to actuate a robot control system and receive feedback from the environment and wheels has been developed and implemented. A hierarchical structure is used to isolate the implementation of the upper level computationally costly algorithms from the interfacing of components and data transfer at the sensor level.
- It's possible to create an accurate map of the surroundings and avoid collisions by employing a cutting-edge sensor such as a LIDAR.
- Use of A-star path planning and navigation algorithms for the implementation of autonomous robot behaviour. Tuning the costmap parameters for optimizing the navigation and testing it through multiple set testcases.
- A point of contact and information management implemented through developing a mobile and web-based application for the admins and customers.
- Integrating the robot's front end with a server-side data management system that allows for real-time communication between the two and making a fast communication bridge between them and the robot.

- A user-friendly UI allows customers to place orders for their preferred food items and gives them the choice for delivery through robot or takeaways.

5.2 Future Works

5.2.1 Camera Integration

The final prototype can use a depth camera sensor that is able to retrieve an occupancy from the environment. Camera would allow the whole navigation system to mitigate the errors that are caused to the limitations of LIDAR i.e., detecting obstacles on different heights. This needs a integration between the data recieved from the two sensors and one way of doing that is to transform the data of depth sensing from the camera into laserscans through a ROS package called depthimage_to_laserscanse and finding a way such that the navigation stack is able to read data from both LIDAR and this packages output. Similarly, other operations will also take advantage of this like mapping will be enhanced and amcl will get much better estimation as well.

5.2.2 Obtaining Initialization Point

One limitation of our robot is that it always needs to start from the origin of the map or otherwise, the robot will never be able to localize itself in the map. This is caused from the fact that robot thinks it is at the origin of the map whereas in reality, the robot is somewhere else in the map. This causes the robot to not being able to reach the goal location. In order to mitigate this issue, we can use different techniques that allows the robot to estimate it's position on map. One way of doing this is that we can send the odometry data of wheels onto the backend of our web application through the connection we created and whenever the robot is initialized, it uses that data to estimate it's location inside the map. Other way could be to implement image recognition through a basic camera such that it is faced upwards and whenever the robot is initialized, the image from camera is used to estimate the location of robot within the map.

5.2.3 Increased Security

Increased security in the application and on the robot such that the user is verified before the order is handed over to it i.e., a QR code is generated for each order and whenever the robot reaches the defined table for order delivery, it allows the person to scan a QR code that is appearing on the robot's LED and if matched, a lock is opened for the user to conveniently take out the order. This sort of modification will ensure that no theft takes place between deliveries.

5.2.4 Better Usage of weight Sensor

For now, we have installed weight sensor just to inform the user about the current weight of the payload and to tell whether the robot is overloaded or not whereas the whole instrument could be used for other things as well since the weight we have obtained is in electrical quantity as well. We can use it to know whether an order is placed on the robot or taken out of it and that could be used to optimize the whole operations and enhance security. Moreover, in case of overload, we can use the electrical signal to stop the controller of the robot from working such that robot does not move from it's position until the weights are in safer limits.

5.2.5 Recovery from Breakage/Complete discharge

Currently, our robot have no solid solution for catering situations where the robot stops mid delivery due to power outage or breakage however, we can install various things to identify such potential scenarios. One way to identify them is to use the odometry data to constantly see whether the robot is moving when it is supposed to be moving and if not, inform the admin or another way is to implement an emergency logic such that an alarm is placed inside the robot that is turned on whenever the the backend of the application receives no response from the robot.

Appendices

APPENDIX A

PROJECT MANAGEMENT

A.1 Bill of Material and Estimated Cost

Items	Price/PKR	Price/PKR
Jetson Nano 2gb	17000	17000
Arduino Uno + Cable	750	0
Memory Card (Jetson Nano)	2850	2850
5V 3A Charger (Jetson Nano)	400	0
L293D Shield Driver	300	0
MPU 9250 & MPU 6500 IMU Soldered	1200	1200
Step Down 9A Buck-Boost Converter - 3 pieces	2250	2250
Miscellaneous (Jumper Wires, Veroboard, Soldering Wire, Double Sided Tapes)	1000	1000
Acrylic Laser Cutting	1500	0
TP link Wifi Dongle	1500	1500
Li-Ion Battery Holders and Chargers	700	700
22V Battery - 2A	1000	1000
Step Down 15-A Buck Convertor	1050	1050
2 x 18650 - 4 cell holder	300	300
4 x Wheels (High Traction)	0	1800
4 x VNH5019 Dual Motor Driver	0	20200
4 x Pololu 12V DC encoder Motors	0	8200
RPLIDAR A2	0	70000
USB Extender	0	800
Arduino Mega + Cable	0	2000
~~Intel Realsense d435I Camera~~	~~0~~	~~0~~
Final Structure Base + Middle + Top Box	6000	6000
Others (Fixing LIDAR Adapter PCB, 2m Screw rod with nuts)	920	600
BMS + Charger + Battery Pack	6500	6500
PCB fabrication | MPU 9250 Sensor (fault replacement) | Bus Cables	2500	2500
Total	47720	147450
Total Cost for the project (Price = 0 depicts that the unit was procured from the university and hence, free of cost)		
Total Cost for the product (Price = 0 depicts that the unit is not required for product however was necessary for prototyping)		

Figure A.1: Bill of Material for the Project

APPENDIX B

CODES RELATED TO MOBILE ROBOT

B.1 The Arduino Codes

B.1.1 Final_ROS_Integrated_Arduino_Code

```
#include <ros.h>
# include <std_msgs/Int16.h>
# include <geometry_msgs/Twist.h>
# include <std_msgs/Float32.h>
// Handles startup and shutdown of ROS
ros::NodeHandle nh;
//Encoder pins Front Wheels
#define ENC_A_FR 3
#define ENC_B_FR 47

#define ENC_A_FL 2
#define ENC_B_FL 46

//Encoder pins Back Wheels
#define ENC_A_BR 18
#define ENC_B_BR 53

#define ENC_A_BL 19
#define ENC_B_BL 52

float lmotorfront,rmotorfront,lmotorback,rmotorback;
// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;
// 16bit integer
const int encoder_minimum = -32768;
const int encoder_maximum = 32767;
```

```
//Front Tyres ROS nodes publisher
std_msgs::Int16 RWT_front;
ros::Publisher rwheelfront("rwheelfront", &RWT_front);

std_msgs::Int16 LWT_front;
ros::Publisher lwheelfront("lwheelfront", &LWT_front);

//Back Tyres ROS nodes publisher
std_msgs::Int16 RWT_back;
ros::Publisher rwheelback("rwheelback", &RWT_back);

std_msgs::Int16 LWT_back;
ros::Publisher lwheelback("lwheelback", &LWT_back);

// Time interval
const int interval = 30;
long previousMillis = 0;
long currentMillis = 0;

// Motor Front Left
const int enA_FL = 11;
const int in1_FL = 44;
const int in2_FL = 45;

// Motor Front Right
const int enB_FR = 12;
const int in3_FR = 43;
const int in4_FR = 42;

// Motor Back Left
const int enA_BL = 9;
const int in1_BL = 50;
const int in2_BL = 51;

```

```
// Motor FBack Right
const int enB_BR = 10;
const int in3_BR = 48;
const int in4_BR = 49;

// Number of ticks per wheel revolution.
const int TICKS_PER_REVOLUTION = 1120;

// Wheel radius in meters
const double WHEEL_RADIUS = 0.05969;

// Distance from center of the left tire to the center of the right
    tire in m
const double WHEEL_BASE = 0.3048;

const double TICKS_PER_METER = 2932;


double pwmLeft_front = 0;
double pwmRight_front = 0;

double pwmLeft_back = 0;
double pwmRight_back = 0;

double lastCmdVelReceived = 0;




void right_front_wheel_tick() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_B_FR);

  if (val == LOW) {
```

```
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }

  if (Direction_right) {

    if (RWT_front.data == encoder_maximum) {
      RWT_front.data = encoder_minimum;
    }
    else {
      RWT_front.data++;
    }
  }
  else {
    if (RWT_front.data == encoder_minimum) {
      RWT_front.data = encoder_maximum;
    }
    else {
      RWT_front.data--;
    }
  }
}


void left_front_wheel_tick() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_B_FL);

  if (val == LOW) {
    Direction_left = false; // Reverse
  }
  else {
```

```
    Direction_left = true; // Forward
  }

  if (Direction_left) {

    if (LWT_front.data == encoder_maximum) {
      LWT_front.data = encoder_minimum;
    }
    else {
      LWT_front.data++;
    }
  }
  else {
    if (LWT_front.data == encoder_minimum) {
      LWT_front.data = encoder_maximum;
    }
    else {
      LWT_front.data--;
    }
  }
}




void right_back_wheel_tick() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_B_BR);

  if (val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
```

```
  }

  if (Direction_right) {

    if (RWT_back.data == encoder_maximum) {
      RWT_back.data = encoder_minimum;
    }
    else {
      RWT_back.data++;
    }
  }
  else {
    if (RWT_back.data == encoder_minimum) {
      RWT_back.data = encoder_maximum;
    }
    else {
      RWT_back.data--;
    }
  }
}


void left_back_wheel_tick() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_B_BL);

  if (val == LOW) {
    Direction_left = false; // Reverse
  }
  else {
    Direction_left = true; // Forward
  }

  if (Direction_left) {
```

```

      if (LWT_back.data == encoder_maximum) {
        LWT_back.data = encoder_minimum;
      }
      else {
        LWT_back.data++;
      }
    }
    else {
      if (LWT_back.data == encoder_minimum) {
        LWT_back.data = encoder_maximum;
      }
      else {
        LWT_back.data--;
      }
    }
}



void pwmrmf(const std_msgs::Float32& msg)
{
    rmotorfront = msg.data;
    }
void pwmlmf(const std_msgs::Float32& msg)
{
    lmotorfront = msg.data;
    }
void pwmrmb(const std_msgs::Float32& msg)
{
    rmotorback = msg.data;
    }
void pwmlmb(const std_msgs::Float32& msg)
{
    lmotorback = msg.data;
```

```
  }


//Instantiating subscriber nodes which takes data from differential_drive package
ros::Subscriber<std_msgs::Float32> rmotor_F("rmotorfront",&pwmrmf);
ros::Subscriber<std_msgs::Float32> lmotor_F("lmotorfront",&pwmlmf);
ros::Subscriber<std_msgs::Float32> rmotor_B("rmotorback",&pwmrmb);
ros::Subscriber<std_msgs::Float32> lmotor_B("lmotorback",&pwmlmb);

void setup() {

  // Set pin states of the encoder Forward
  pinMode(ENC_A_FR, INPUT_PULLUP);
  pinMode(ENC_B_FR, INPUT);
  pinMode(ENC_A_FL, INPUT_PULLUP);
  pinMode(ENC_B_FL, INPUT);

  // Set pin states of the encoder Back
  pinMode(ENC_A_BR, INPUT_PULLUP);
  pinMode(ENC_B_BR, INPUT);
  pinMode(ENC_A_BL, INPUT_PULLUP);
  pinMode(ENC_B_BL, INPUT);


  // Every time the pin goes high, this is a tick
  attachInterrupt(digitalPinToInterrupt(ENC_A_FR), left_front_wheel_tick, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_A_FL), right_front_wheel_tick, RISING);

  attachInterrupt(digitalPinToInterrupt(ENC_A_BR), left_back_wheel_tick, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_A_BL), right_back_wheel_tick, RISING);
```

```
// Motor control pins Front
pinMode(enA_FL, OUTPUT);
pinMode(enB_FR, OUTPUT);
pinMode(in1_FL, OUTPUT);
pinMode(in2_FL, OUTPUT);
pinMode(in3_FR, OUTPUT);
pinMode(in4_FR, OUTPUT);

// Motor control pins Back
pinMode(enA_BL, OUTPUT);
pinMode(enB_BR, OUTPUT);
pinMode(in1_BL, OUTPUT);
pinMode(in2_BL, OUTPUT);
pinMode(in3_BR, OUTPUT);
pinMode(in4_BR, OUTPUT);


// Turn off motors - Initial state
digitalWrite(in1_FL, LOW);
digitalWrite(in2_FL, LOW);
digitalWrite(in3_FR, LOW);
digitalWrite(in4_FR, LOW);

digitalWrite(in1_BL, LOW);
digitalWrite(in2_BL, LOW);
digitalWrite(in3_BR, LOW);
digitalWrite(in4_BR, LOW);

// Set the motor speed
analogWrite(enA_FL, 0);
analogWrite(enB_FR, 0);
analogWrite(enA_BL, 0);
analogWrite(enB_BR, 0);
```

```
  // ROS Setup
  nh.getHardware()->setBaud(115200);
  nh.initNode();
  //Publishing encoder values
  nh.advertise(rwheelfront);
  nh.advertise(lwheelfront);
  nh.advertise(rwheelback);
  nh.advertise(lwheelback);
  //subscribing to these topics to get PWM values
  nh.subscribe(rmotor_F);
  nh.subscribe(lmotor_F);
  nh.subscribe(rmotor_B);
  nh.subscribe(lmotor_B);
}


//void timerIsr(){
//   Timer1.detachInterrupt();//stopthetimer
//   right_wheel_enc.data=knobRight.read();
//   left_wheel_enc.data=knobLeft.read();
//   right_wheel_enc_pub.publish(&right_wheel_enc);
//   left_wheel_enc_pub.publish(&left_wheel_enc);
//   Timer1.attachInterrupt(timerIsr);
//   //enablethetimer
//}
void loop() {

  nh.spinOnce();


  lwheelfront.publish( &LWT_front );
  rwheelfront.publish( &RWT_front );
  lwheelback.publish( &LWT_back );
  rwheelback.publish( &RWT_back );

```

```
if(lmotorfront > 0 && rmotorfront > 0)
{
  digitalWrite(in1_FL,HIGH);
  digitalWrite(in2_FL,LOW);
  digitalWrite(in3_FR,HIGH);
  digitalWrite(in4_FR,LOW);

  digitalWrite(in1_BL,HIGH);
  digitalWrite(in2_BL,LOW);
  digitalWrite(in3_BR,HIGH);
  digitalWrite(in4_BR,LOW);
}


else if(lmotorfront < 0 && rmotorfront < 0)
{
  digitalWrite(in1_FL,LOW);
  digitalWrite(in2_FL,HIGH);
  digitalWrite(in3_FR,LOW);
  digitalWrite(in4_FR,HIGH);

  digitalWrite(in1_BL,LOW);
  digitalWrite(in2_BL,HIGH);
  digitalWrite(in3_BR,LOW);
  digitalWrite(in4_BR,HIGH);
}



else if(lmotorfront > 0 && rmotorfront < 0)
{
  digitalWrite(in1_FL,LOW);
  digitalWrite(in2_FL,HIGH);
  digitalWrite(in3_FR,HIGH);
```

```
    digitalWrite(in4_FR,LOW);

    digitalWrite(in1_BL,LOW);
    digitalWrite(in2_BL,HIGH);
    digitalWrite(in3_BR,HIGH);
    digitalWrite(in4_BR,LOW);
  }



  else if(lmotorfront < 0 && rmotorfront > 0)
  {
    digitalWrite(in1_FL,HIGH);
    digitalWrite(in2_FL,LOW);
    digitalWrite(in3_FR,LOW);
    digitalWrite(in4_FR,HIGH);

    digitalWrite(in1_BL,HIGH);
    digitalWrite(in2_BL,LOW);
    digitalWrite(in3_BR,LOW);
    digitalWrite(in4_BR,HIGH);
  }


  else
  {
    digitalWrite(in1_FL,LOW);
    digitalWrite(in2_FL,LOW);
    digitalWrite(in3_FR,LOW);
    digitalWrite(in4_FR,LOW);

    digitalWrite(in1_BL,LOW);
    digitalWrite(in2_BL,LOW);
    digitalWrite(in3_BR,LOW);
    digitalWrite(in4_BR,LOW);
```

```
  }




  analogWrite(enA_FL, abs(lmotorfront));
  analogWrite(enB_FR, abs(rmotorfront));
  analogWrite(enA_BL, abs(lmotorback));
  analogWrite(enB_BR, abs(rmotorback));
  Serial.println(LWT_back.data);
  delay(25);
}
```

B.1.2 Interfacing_Rosserial_Code

```

#include <ros.h>
#include <std_msgs/Int16.h>
#include <geometry_msgs/Twist.h>

ros::NodeHandle nh;

#define ENC_IN_LEFT_A 19
#define ENC_IN_RIGHT_A 18
#define ENC_IN_LEFT_B 53
#define ENC_IN_RIGHT_B 52


boolean Direction_left = true;
boolean Direction_right = true;


const int encoder_minimum = -32768;
const int encoder_maximum = 32767;

// Keep track of the number of wheel ticks
```

```
std_msgs::Int16 right_wheel_tick_count;
ros::Publisher rightPub("right_ticks_1", &right_wheel_tick_count);

std_msgs::Int16 left_wheel_tick_count;
ros::Publisher leftPub("left_ticks_1", &left_wheel_tick_count);

// Time interval for measurements in milliseconds
const int interval = 30;
long previousMillis = 0;
long currentMillis = 0;


const int enA = 9;
const int in1 = 50;
const int in2 = 51;
const int enB = 10;
const int in3 = 48;
const int in4 = 49;


const int PWM_INCREMENT = 1;


const int TICKS_PER_REVOLUTION = 1120;

// Wheel radius in meters
const double WHEEL_RADIUS = 0.05969;


const double WHEEL_BASE = 0.3048;
const double TICKS_PER_METER = 2932;

const int K_P = 278;

const int b = 52;
```

```
const int DRIFT_MULTIPLIER = 120;

const int PWM_TURN = 80;

const int PWM_MIN = 80;
const int PWM_MAX = 250;

double velLeftWheel = 0;
double velRightWheel = 0;
double pwmLeftReq = 0;
double pwmRightReq = 0;
double lastCmdVelReceived = 0;


void right_wheel_tick() {

  int val = digitalRead(ENC_IN_RIGHT_B);

  if (val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }

  if (Direction_right) {

    if (right_wheel_tick_count.data == encoder_maximum) {
      right_wheel_tick_count.data = encoder_minimum;
    }
    else {
      right_wheel_tick_count.data++;
    }
  }
```

```
  else {
    if (right_wheel_tick_count.data == encoder_minimum) {
      right_wheel_tick_count.data = encoder_maximum;
    }
    else {
      right_wheel_tick_count.data--;
    }
  }
}

// Increment the number of ticks
void left_wheel_tick() {

  // Read the value for the encoder for the left wheel
  int val = digitalRead(ENC_IN_LEFT_B);

  if (val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count.data == encoder_maximum) {
      left_wheel_tick_count.data = encoder_minimum;
    }
    else {
      left_wheel_tick_count.data++;
    }
  }
  else {
    if (left_wheel_tick_count.data == encoder_minimum) {
      left_wheel_tick_count.data = encoder_maximum;
    }
```

```
    else {
      left_wheel_tick_count.data--;
    }
  }
}


void calc_vel_left_wheel(){

  // Previous timestamp
  static double prevTime = 0;

  // Variable gets created and initialized the first time a function
    is called.
  static int prevLeftCount = 0;

  // Manage rollover and rollunder when we get outside the 16-bit
    integer range
  int numOfTicks = (65535 + left_wheel_tick_count.data -
    prevLeftCount) % 65535;

  // If we have had a big jump, it means the tick count has rolled
    over.
  if (numOfTicks > 10000) {
        numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  velLeftWheel = numOfTicks/TICKS_PER_METER/((millis()/1000)-prevTime
    );

  // Keep track of the previous tick count
  prevLeftCount = left_wheel_tick_count.data;

  // Update the timestamp
```

```
  prevTime = (millis()/1000);

}

void calc_vel_right_wheel(){

  // Previous timestamp
  static double prevTime = 0;

  // Variable gets created and initialized the first time a function is called.
  static int prevRightCount = 0;

  // Manage rollover and rollunder when we get outside the 16-bit integer range
  int numOfTicks = (65535 + right_wheel_tick_count.data - prevRightCount) % 65535;

  if (numOfTicks > 10000) {
        numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  velRightWheel = numOfTicks/TICKS_PER_METER/((millis()/1000)-prevTime);

  prevRightCount = right_wheel_tick_count.data;

  prevTime = (millis()/1000);

}

// Take the velocity command as input and calculate the PWM values.
void calc_pwm_values(const geometry_msgs::Twist& cmdVel) {

```

```
// Record timestamp of last velocity command received
lastCmdVelReceived = (millis()/1000);

// Calculate the PWM value given the desired velocity
pwmLeftReq = K_P * cmdVel.linear.x + b;
pwmRightReq = K_P * cmdVel.linear.x + b;

// Check if we need to turn
if (cmdVel.angular.z != 0.0) {

  // Turn left
  if (cmdVel.angular.z > 0.0) {
    pwmLeftReq = -PWM_TURN;
    pwmRightReq = PWM_TURN;
  }
  // Turn right
  else {
    pwmLeftReq = PWM_TURN;
    pwmRightReq = -PWM_TURN;
  }
}
// Go straight
else {

  static double prevDiff = 0;
  static double prevPrevDiff = 0;
  double currDifference = velLeftWheel - velRightWheel;
  double avgDifference = (prevDiff+prevPrevDiff+currDifference)/3;
  prevPrevDiff = prevDiff;
  prevDiff = currDifference;

  // Correct PWM values of both wheels to make the vehicle go straight
  pwmLeftReq -= (int)(avgDifference * DRIFT_MULTIPLIER);
  pwmRightReq += (int)(avgDifference * DRIFT_MULTIPLIER);
```

```
  }

  // Handle low PWM values
  if (abs(pwmLeftReq) < PWM_MIN) {
    pwmLeftReq = 0;
  }
  if (abs(pwmRightReq) < PWM_MIN) {
    pwmRightReq = 0;
  }
}

void set_pwm_values() {

  // These variables will hold our desired PWM values
  static int pwmLeftOut = 0;
  static int pwmRightOut = 0;

  // If the required PWM is of opposite sign as the output PWM, we want to
  // stop the car before switching direction
  static bool stopped = false;
  if ((pwmLeftReq * velLeftWheel < 0 && pwmLeftOut != 0) ||
      (pwmRightReq * velRightWheel < 0 && pwmRightOut != 0)) {
    pwmLeftReq = 0;
    pwmRightReq = 0;
  }

  // Set the direction of the motors
  if (pwmLeftReq > 0) { // Left wheel forward
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
  }
  else if (pwmLeftReq < 0) { // Left wheel reverse
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
```

```
}
else if (pwmLeftReq == 0 && pwmLeftOut == 0 ) { // Left wheel stop
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
}
else { // Left wheel stop
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
}

if (pwmRightReq > 0) { // Right wheel forward
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}
else if(pwmRightReq < 0) { // Right wheel reverse
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}
else if (pwmRightReq == 0 && pwmRightOut == 0) { // Right wheel
  stop
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
}
else { // Right wheel stop
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
}

// Increase the required PWM if the robot is not moving
if (pwmLeftReq != 0 && velLeftWheel == 0) {
  pwmLeftReq *= 1.5;
}
if (pwmRightReq != 0 && velRightWheel == 0) {
  pwmRightReq *= 1.5;
}
```

```
  // Calculate the output PWM value by making slow changes to the current value
  if (abs(pwmLeftReq) > pwmLeftOut) {
    pwmLeftOut += PWM_INCREMENT;
  }
  else if (abs(pwmLeftReq) < pwmLeftOut) {
    pwmLeftOut -= PWM_INCREMENT;
  }
  else{}

  if (abs(pwmRightReq) > pwmRightOut) {
    pwmRightOut += PWM_INCREMENT;
  }
  else if(abs(pwmRightReq) < pwmRightOut) {
    pwmRightOut -= PWM_INCREMENT;
  }
  else{}

  // Conditional operator to limit PWM output at the maximum
  pwmLeftOut = (pwmLeftOut > PWM_MAX) ? PWM_MAX : pwmLeftOut;
  pwmRightOut = (pwmRightOut > PWM_MAX) ? PWM_MAX : pwmRightOut;

  // PWM output cannot be less than 0
  pwmLeftOut = (pwmLeftOut < 0) ? 0 : pwmLeftOut;
  pwmRightOut = (pwmRightOut < 0) ? 0 : pwmRightOut;

  // Set the PWM value on the pins
  analogWrite(enA, pwmLeftOut);
  analogWrite(enB, pwmRightOut);
}

// Set up ROS subscriber to the velocity command
ros::Subscriber<geometry_msgs::Twist> subCmdVel("cmd_vel", &calc_pwm_values );
```

```

void setup() {

  // Set pin states of the encoder
  pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);
  pinMode(ENC_IN_LEFT_B , INPUT);
  pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_B , INPUT);
  pinMode(five_volt , OUTPUT);
  // Every time the pin goes high, this is a tick
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A),
    left_wheel_tick , RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A),
    right_wheel_tick , RISING);

  // Motor control pins are outputs
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  // Turn off motors - Initial state
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  digitalWrite(five_volt , HIGH);
  // Set the motor speed
  analogWrite(enA, 0);
  analogWrite(enB, 0);

  // ROS Setup
  nh.getHardware()->setBaud(115200);
```

```
  nh.initNode();
  nh.advertise(rightPub);
  nh.advertise(leftPub);
  nh.subscribe(subCmdVel);
}

void loop() {

  nh.spinOnce();

  // Record the time
  currentMillis = millis();

  // If the time interval has passed, publish the number of ticks,
  // and calculate the velocities.
  if (currentMillis - previousMillis > interval) {

    previousMillis = currentMillis;

    // Publish tick counts to topics
    leftPub.publish( &left_wheel_tick_count );
    rightPub.publish( &right_wheel_tick_count );

    // Calculate the velocity of the right and left wheels
    calc_vel_right_wheel();
    calc_vel_left_wheel();

  }

  // Stop the car if there are no cmd_vel messages
  if((millis()/1000) - lastCmdVelReceived > 1) {
    pwmLeftReq = 0;
    pwmRightReq = 0;
  }

```

```
  set_pwm_values();
}
```

B.1.3 Publish_Tick_on_ROS

```

#include <ros.h>
#include <std_msgs/Int16.h>
#include <geometry_msgs/Twist.h>
// Handles startup and shutdown of ROS
ros::NodeHandle nh;

// Encoder output to Arduino Interrupt pin. Tracks the tick count.
#define ENC_IN_LEFT_A_BACK 19
#define ENC_IN_RIGHT_A_BACK 18


#define ENC_IN_LEFT_A_FRONT 2
#define ENC_IN_RIGHT_A_FRONT 3

// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_B_BACK 53
#define ENC_IN_RIGHT_B_BACK 52

#define ENC_IN_LEFT_B_FRONT 47
#define ENC_IN_RIGHT_B_FRONT 46

// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;

// Minumum and maximum values for 16-bit integers
// Range of 65,535
const int encoder_minimum = -32768;
const int encoder_maximum = 32767;
```

```
// Keep track of the number of wheel ticks
std_msgs::Int16 right_wheel_tick_count_BACK;
ros::Publisher rightPub_BACK("right_ticks_back", &
    right_wheel_tick_count_BACK);

std_msgs::Int16 left_wheel_tick_count_BACK;
ros::Publisher leftPub_BACK("left_ticks_back", &
    left_wheel_tick_count_BACK);


std_msgs::Int16 right_wheel_tick_count_FRONT;
ros::Publisher rightPub_FRONT("right_ticks_front", &
    right_wheel_tick_count_FRONT);

std_msgs::Int16 left_wheel_tick_count_FRONT;
ros::Publisher leftPub_FRONT("left_ticks_front", &
    left_wheel_tick_count_FRONT);

// Time interval for measurements in milliseconds
const int interval = 30;
long previousMillis = 0;
long currentMillis = 0;

///////////////// Motor Controller Variables and Constants
    ///////////////////

// Motor A connections - LEFT
const int enA_BACK = 9;
const int in1_BACK = 50;
const int in2_BACK = 51;

// Motor B connections - RIGHT
const int enB_BACK = 10;
const int in3_BACK = 48;
```

```
const int in4_BACK = 49;


// Motor A connections - LEFT
const int enA_FRONT = 11;
const int in1_FRONT = 44;
const int in2_FRONT = 45;

// Motor B connections - RIGHT
const int enB_FRONT = 12;
const int in3_FRONT = 43;
const int in4_FRONT = 42;

// How much the PWM value can change each cycle
const int PWM_INCREMENT = 1;

// Number of ticks per wheel revolution. We won't use this in this code.
const int TICKS_PER_REVOLUTION = 1120;

// Wheel radius in meters
const double WHEEL_RADIUS = 0.05969;

// Distance from center of the left tire to the center of the right tire in m
const double WHEEL_BASE = 0.3048;

// Number of ticks a wheel makes moving a linear distance of 1 meter
// This value was measured manually.
const double TICKS_PER_METER = 2932; // Originally 2880

// Proportional constant, which was measured by measuring the
// PWM-Linear Velocity relationship for the robot.
const int K_P = 10;
const int K_I = 15;
```

```
// Y-intercept for the PWM-Linear Velocity relationship for the robot
const int b = 52;

// Correction multiplier for drift. Chosen through experimentation.
const int DRIFT_MULTIPLIER = 120;

// Turning PWM output (0 = min, 255 = max for PWM values)
const int PWM_TURN = 150;

// Set maximum and minimum limits for the PWM values
const int PWM_MIN = 80; // about 0.1 m/s
const int PWM_MAX = 250; // about 0.172 m/s

// Set linear velocity and PWM variable values for each wheel
double velLeftWheel_BACK = 0;
double velRightWheel_BACK = 0;
double pwmLeftReq_BACK = 0;
double pwmRightReq_BACK = 0;

double velLeftWheel_FRONT = 0;
double velRightWheel_FRONT = 0;
double pwmLeftReq_FRONT = 0;
double pwmRightReq_FRONT = 0;

// Record the time that the last velocity command was received
double lastCmdVelReceived = 0;

/////////////////////// Tick Data Publishing Functions
    ////////////////////////

// Increment the number of ticks
void right_wheel_tick_back() {

  // Read the value for the encoder for the right wheel
```

```
  int val = digitalRead(ENC_IN_RIGHT_B_BACK);

  if (val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }

  if (Direction_right) {

    if (right_wheel_tick_count_BACK.data == encoder_maximum) {
      right_wheel_tick_count_BACK.data = encoder_minimum;
    }
    else {
      right_wheel_tick_count_BACK.data++;
    }
  }
  else {
    if (right_wheel_tick_count_BACK.data == encoder_minimum) {
      right_wheel_tick_count_BACK.data = encoder_maximum;
    }
    else {
      right_wheel_tick_count_BACK.data--;
    }
  }
}




// Increment the number of ticks
void left_wheel_tick_back() {

  // Read the value for the encoder for the left wheel
```

```
  int val = digitalRead(ENC_IN_LEFT_B_BACK);

  if (val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count_BACK.data == encoder_maximum) {
      left_wheel_tick_count_BACK.data = encoder_minimum;
    }
    else {
      left_wheel_tick_count_BACK.data++;
    }
  }
  else {
    if (left_wheel_tick_count_BACK.data == encoder_minimum) {
      left_wheel_tick_count_BACK.data = encoder_maximum;
    }
    else {
      left_wheel_tick_count_BACK.data--;
    }
  }
}




void right_wheel_tick_FRONT() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_RIGHT_B_FRONT);

```

```
  if (val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }

  if (Direction_right) {

    if (right_wheel_tick_count_FRONT.data == encoder_maximum) {
      right_wheel_tick_count_FRONT.data = encoder_minimum;
    }
    else {
      right_wheel_tick_count_FRONT.data++;
    }
  }
  else {
    if (right_wheel_tick_count_FRONT.data == encoder_minimum) {
      right_wheel_tick_count_FRONT.data = encoder_maximum;
    }
    else {
      right_wheel_tick_count_FRONT.data--;
    }
  }
}




// Increment the number of ticks
void left_wheel_tick_FRONT() {

  // Read the value for the encoder for the left wheel
  int val = digitalRead(ENC_IN_LEFT_B_FRONT);

```

```
  if (val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count_FRONT.data == encoder_maximum) {
      left_wheel_tick_count_FRONT.data = encoder_minimum;
    }
    else {
      left_wheel_tick_count_FRONT.data++;
    }
  }
  else {
    if (left_wheel_tick_count_FRONT.data == encoder_minimum) {
      left_wheel_tick_count_FRONT.data = encoder_maximum;
    }
    else {
      left_wheel_tick_count_FRONT.data--;
    }
  }
}










///////////////////////// Motor Controller Functions
    ////////////////////////////
```

```

// Calculate the left wheel linear velocity in m/s every time a
// tick count message is rpublished on the /left_ticks topic.
void calc_vel_left_wheel_BACK(){

  // Previous timestamp
  static double prevTime = 0;

  // Variable gets created and initialized the first time a function is called.
  static int prevLeftCount_BACK = 0;

  // Manage rollover and rollunder when we get outside the 16-bit integer range
  int numOfTicks = (65535 + left_wheel_tick_count_BACK.data - prevLeftCount_BACK) % 65535;

  // If we have had a big jump, it means the tick count has rolled over.
  if (numOfTicks > 10000) {
        numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  velLeftWheel_BACK = numOfTicks/TICKS_PER_METER/((millis()/1000)-prevTime);

  // Keep track of the previous tick count
  prevLeftCount_BACK = left_wheel_tick_count_BACK.data;

  // Update the timestamp
  prevTime = (millis()/1000);

}

```

```
// Calculate the right wheel linear velocity in m/s every time a
// tick count message is published on the /right_ticks topic.
void calc_vel_right_wheel_BACK(){

  // Previous timestamp
  static double prevTime = 0;

  // Variable gets created and initialized the first time a function
    is called.
  static int prevRightCount_BACK = 0;

  // Manage rollover and rollunder when we get outside the 16-bit
    integer range
  int numOfTicks = (65535 + right_wheel_tick_count_BACK.data -
    prevRightCount_BACK) % 65535;

  if (numOfTicks > 10000) {
        numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  velRightWheel_BACK = numOfTicks/TICKS_PER_METER/((millis()/1000)-
    prevTime);

  prevRightCount_BACK = right_wheel_tick_count_BACK.data;

  prevTime = (millis()/1000);

}


//FRONT WHEELS
void calc_vel_left_wheel_FRONT(){

  // Previous timestamp
```

```
  static double prevTime = 0;

  // Variable gets created and initialized the first time a function
    is called.
  static int prevLeftCount_FRONT = 0;

  // Manage rollover and rollunder when we get outside the 16-bit
    integer range
  int numOfTicks = (65535 + left_wheel_tick_count_FRONT.data -
    prevLeftCount_FRONT) % 65535;

  // If we have had a big jump, it means the tick count has rolled
    over.
  if (numOfTicks > 10000) {
        numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  velLeftWheel_FRONT = numOfTicks/TICKS_PER_METER/((millis()/1000)-
    prevTime);

  // Keep track of the previous tick count
  prevLeftCount_FRONT = left_wheel_tick_count_FRONT.data;

  // Update the timestamp
  prevTime = (millis()/1000);

}

// Calculate the right wheel linear velocity in m/s every time a
// tick count message is published on the /right_ticks topic.
void calc_vel_right_wheel_FRONT(){

  // Previous timestamp
  static double prevTime = 0;
```

```

  // Variable gets created and initialized the first time a function is called.
  static int prevRightCount_FRONT = 0;

  // Manage rollover and rollunder when we get outside the 16-bit integer range
  int numOfTicks = (65535 + right_wheel_tick_count_FRONT.data - prevRightCount_FRONT) % 65535;

  if (numOfTicks > 10000) {
        numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  velRightWheel_FRONT = numOfTicks/TICKS_PER_METER/((millis()/1000)-prevTime);

  prevRightCount_FRONT = right_wheel_tick_count_FRONT.data;

  prevTime = (millis()/1000);

}











// Take the velocity command as input and calculate the PWM values.
```

```
void calc_pwm_values(const geometry_msgs::Twist& cmdVel) {

  // Record timestamp of last velocity command received
  lastCmdVelReceived = (millis()/1000);
  long currT = micros();
  float deltaT = ((float) (currT - prevT))/( 1.0e6 );
  prevT = currT;
//  int error_left_back = cmdVel.linear.x - ACTUAL_VEL_LEFT_BACK;
//  int error_right_back = cmdVel.linear.x - ACTUAL_VEL_RIGHT_BACK;
//  int error_left_front = cmdVel.linear.x - ACTUAL_VEL_LEFT_FRONT;
//  int error_right_front = cmdVel.linear.x - ACTUAL_VEL_RIGHT_FRONT;
  int error_left_back = cmdVel.linear.x;
  int error_right_back = cmdVel.linear.x;
  int error_left_front = cmdVel.linear.x;
  int error_right_front = cmdVel.linear.x;
  // Calculate the PWM value given the desired velocity
  pwmLeftReq_BACK = K_P * cmdVel.linear.x + b;
  pwmLeftReq_FRONT = pwmLeftReq_BACK;
  pwmRightReq_FRONT = K_P * cmdVel.linear.x + b;
  pwmRightReq_BACK = pwmRightReq_FRONT;

  // Check if we need to turn
  if (cmdVel.angular.z != 0.0) {

    // Turn left
    if (cmdVel.angular.z > 0.0) {
      pwmLeftReq = -PWM_TURN;
      pwmRightReq = PWM_TURN;
    }
    // Turn right
    else {
      pwmLeftReq = PWM_TURN;
      pwmRightReq = -PWM_TURN;
    }
  }
```

```
  // Go straight
  else {

    // Remove any differences in wheel velocities
    // to make sure the robot goes straight
    static double prevDiff = 0;
    static double prevPrevDiff = 0;
    double currDifference = velLeftWheel - velRightWheel;
    double avgDifference = (prevDiff+prevPrevDiff+currDifference)/3;
    prevPrevDiff = prevDiff;
    prevDiff = currDifference;

    // Correct PWM values of both wheels to make the vehicle go straight
    pwmLeftReq -= (int)(avgDifference * DRIFT_MULTIPLIER);
    pwmRightReq += (int)(avgDifference * DRIFT_MULTIPLIER);
  }

  // Handle low PWM values
  if (abs(pwmLeftReq) < PWM_MIN) {
    pwmLeftReq = 0;
  }
  if (abs(pwmRightReq) < PWM_MIN) {
    pwmRightReq = 0;
  }
}

void set_pwm_values() {

  // These variables will hold our desired PWM values
  static int pwmLeftOut = 0;
  static int pwmRightOut = 0;

  // If the required PWM is of opposite sign as the output PWM, we want to
```

```
// stop the car before switching direction
static bool stopped = false;
if ((pwmLeftReq * velLeftWheel < 0 && pwmLeftOut != 0) ||
    (pwmRightReq * velRightWheel < 0 && pwmRightOut != 0)) {
  pwmLeftReq = 0;
  pwmRightReq = 0;
}

// Set the direction of the motors
if (pwmLeftReq > 0) { // Left wheel forward
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
}
else if (pwmLeftReq < 0) { // Left wheel reverse
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
}
else if (pwmLeftReq == 0 && pwmLeftOut == 0 ) { // Left wheel stop
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
}
else { // Left wheel stop
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
}

if (pwmRightReq > 0) { // Right wheel forward
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}
else if(pwmRightReq < 0) { // Right wheel reverse
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}
else if (pwmRightReq == 0 && pwmRightOut == 0) { // Right wheel
```

```
    stop
      digitalWrite(in3, LOW);
      digitalWrite(in4, LOW);
    }
    else { // Right wheel stop
      digitalWrite(in3, LOW);
      digitalWrite(in4, LOW);
    }

    // Increase the required PWM if the robot is not moving
    if (pwmLeftReq != 0 && velLeftWheel == 0) {
      pwmLeftReq *= 1.5;
    }
    if (pwmRightReq != 0 && velRightWheel == 0) {
      pwmRightReq *= 1.5;
    }

    // Calculate the output PWM value by making slow changes to the
      current value
    if (abs(pwmLeftReq) > pwmLeftOut) {
      pwmLeftOut += PWM_INCREMENT;
    }
    else if (abs(pwmLeftReq) < pwmLeftOut) {
      pwmLeftOut -= PWM_INCREMENT;
    }
    else{}

    if (abs(pwmRightReq) > pwmRightOut) {
      pwmRightOut += PWM_INCREMENT;
    }
    else if(abs(pwmRightReq) < pwmRightOut) {
      pwmRightOut -= PWM_INCREMENT;
    }
    else{}

```

```
  // Conditional operator to limit PWM output at the maximum
  pwmLeftOut = (pwmLeftOut > PWM_MAX) ? PWM_MAX : pwmLeftOut;
  pwmRightOut = (pwmRightOut > PWM_MAX) ? PWM_MAX : pwmRightOut;

  // PWM output cannot be less than 0
  pwmLeftOut = (pwmLeftOut < 0) ? 0 : pwmLeftOut;
  pwmRightOut = (pwmRightOut < 0) ? 0 : pwmRightOut;

  // Set the PWM value on the pins
  analogWrite(enA, pwmLeftOut);
  analogWrite(enB, pwmRightOut);
}

// Set up ROS subscriber to the velocity command
ros::Subscriber<geometry_msgs::Twist> subCmdVel("cmd_vel", &
    calc_pwm_values );

void setup() {

  // Set pin states of the encoder
  pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);
  pinMode(ENC_IN_LEFT_B , INPUT);
  pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_B , INPUT);

  // Every time the pin goes high, this is a tick
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A),
    left_wheel_tick, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A),
    right_wheel_tick, RISING);

  // Motor control pins are outputs
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
```

```
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  // Turn off motors - Initial state
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);

  // Set the motor speed
  analogWrite(enA, 0);
  analogWrite(enB, 0);

  // ROS Setup
  nh.getHardware()->setBaud(115200);
  nh.initNode();
  nh.advertise(rightPub);
  nh.advertise(leftPub);
  nh.subscribe(subCmdVel);
}

void loop() {

  nh.spinOnce();

  // Record the time
  currentMillis = millis();

  // If the time interval has passed, publish the number of ticks,
  // and calculate the velocities.
  if (currentMillis - previousMillis > interval) {

    previousMillis = currentMillis;

```

```
    // Publish tick counts to topics
    leftPub.publish( &left_wheel_tick_count );
    rightPub.publish( &right_wheel_tick_count );

    // Calculate the velocity of the right and left wheels
    calc_vel_right_wheel();
    calc_vel_left_wheel();

  }

  // Stop the car if there are no cmd_vel messages
  if((millis()/1000) - lastCmdVelReceived > 1) {
    pwmLeftReq = 0;
    pwmRightReq = 0;
  }

  set_pwm_values();
}
```

B.1.4 Velocity_Command_Intake_from_ROS

```
#include <ArduinoHardware.h> //THIS SHOULD BE THE FINAL CODE. Acha
    abb dekhna ke how to program a publisher which publishes
//a geometry on the topic and this node can subscribe to that node
    and apply the differential controller code from github.
#include <AFMotor.h>
#include <ros.h>
#include <geometry_msgs/Twist.h>

ros::NodeHandle nh;
geometry_msgs::Twist msg;

#define PWM_MIN 300

AF_DCMotor motorLB(1);
AF_DCMotor motorRB(2);
```

```
AF_DCMotor motorRF(3);
AF_DCMotor motorLF(4);
// 1 - LB || 2 - RB || 3 - RF || 4 - LF


float move1;
float move2;

void roverCallBack(const geometry_msgs::Twist& cmd_vel)
{
  move1 = cmd_vel.linear.x * 127 ;
  move2 = cmd_vel.angular.z * 127 ;
}

ros::Subscriber <geometry_msgs::Twist> sub("/cmd_vel", roverCallBack)
    ;

void setup()
{
  nh.initNode();
  nh.subscribe(sub);
  Serial.begin(115200);
  while(!Serial){;}
  motorLB.run(RELEASE);
  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
  motorLF.run(RELEASE);
}

void loop()
{
  nh.spinOnce();
  delay(1);
}
```

B.1.5 Wheel_RPM_Calculate_Code

```
/*
   Author: Automatic Addison
   Website: https://automaticaddison.com
   Description: Calculate the angular velocity in radians/second of a DC motor
   with a built-in encoder (forward = positive; reverse = negative)
*/

// Motor encoder output pulses per 360 degree revolution (measured manually)
#define ENC_COUNT_REV 1120

// Encoder output to Arduino Interrupt pin. Tracks the pulse count.
#define ENC_IN_RIGHT_A 18

// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_RIGHT_B 53



// Encoder output to Arduino Interrupt pin. Tracks the pulse count.
#define ENC_IN_LEFT_A 19

// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_B 52
// True = Forward; False = Reverse
boolean Direction_right = true;
boolean Direction_left = true;
// Keep track of the number of right wheel pulses
volatile long right_wheel_pulse_count = 0;
volatile long left_wheel_pulse_count = 0;
// One-second interval for measurements
```

```
int interval = 1000;
int pwmA = 9;
int pwmB = 10;
int enA1 = 50;
int enA2 = 51;
int enB1 = 49;
int enB2 = 48;
// Counters for milliseconds during interval
long previousMillis = 0;
long currentMillis = 0;

// Variable for RPM measuerment
float rpm_right = 0;
float rpm_left = 0;
// Variable for angular velocity measurement
float ang_velocity_right = 0;
float ang_velocity_right_deg = 0;

float ang_velocity_left = 0;
float ang_velocity_left_deg = 0;

const float rpm_to_radians = 0.2923989365;
const float rad_to_deg = 16.753225;

void setup() {

  // Open the serial port at 9600 bps
  Serial.begin(9600);
  pinMode(pwmA, OUTPUT);
  pinMode(pwmB, OUTPUT);
  pinMode(enA1, OUTPUT);
  pinMode(enA2, OUTPUT);
  pinMode(enB1, OUTPUT);
  pinMode(enB2, OUTPUT);
  // Set pin states of the encoder
```

```
  pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_B , INPUT);
  pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);
  pinMode(ENC_IN_LEFT_B , INPUT);
  // Every time the pin goes high, this is a pulse
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A),
    right_wheel_pulse, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A),
    left_wheel_pulse, RISING);
  digitalWrite(enA1, HIGH);
  digitalWrite(enA2, LOW);
  digitalWrite(enB1, LOW);
  digitalWrite(enB2, HIGH);
}

void loop() {
  analogWrite(pwmA, 250);
  analogWrite(pwmB, 250);
  // Record the time
  currentMillis = millis();

  // If one second has passed, print the number of pulses
  if (currentMillis - previousMillis > interval) {

    previousMillis = currentMillis;

    // Calculate revolutions per minute
    rpm_right = (float)(right_wheel_pulse_count * 60 / ENC_COUNT_REV)
    ;
    ang_velocity_right = rpm_right * rpm_to_radians;
    ang_velocity_right_deg = ang_velocity_right * rad_to_deg;

    //Serial.print(" Right Pulses: ");
    //Serial.println(right_wheel_pulse_count);

```

```
//     Serial.println(" RPM");
//     Serial.print(" Angular Velocity: ");
//     Serial.print(rpm_right);
//     Serial.print(" rad per second");
//     Serial.print("\t");
//     Serial.print(ang_velocity_right_deg);
//     Serial.println(" deg per second");
    Serial.println();

    rpm_left = (float)(left_wheel_pulse_count * 60 / ENC_COUNT_REV);
    ang_velocity_left = rpm_left * rpm_to_radians;
    ang_velocity_left_deg = ang_velocity_left * rad_to_deg;

    //Serial.print(" Left Pulses: ");
    //Serial.println(left_wheel_pulse_count);
    //Serial.print(" Right Speed: ");
    Serial.print(rpm_right);
    //Serial.println();
    //Serial.print(" Left Speed: ");
    Serial.print(rpm_left);
//     Serial.print(" Angular Velocity: ");
//     Serial.print(rpm_right);
//     Serial.print(" rad per second");
//     Serial.print("\t");
//     Serial.print(ang_velocity_right_deg);
//     Serial.println(" deg per second");
    //Serial.println();



    right_wheel_pulse_count = 0;
    left_wheel_pulse_count = 0;


  }
```

```
}

// Increment the number of pulses by 1
void right_wheel_pulse() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_RIGHT_B);

  if (val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }

  if (Direction_right) {
    right_wheel_pulse_count++;
  }
  else {
    right_wheel_pulse_count--;
  }
}


void left_wheel_pulse() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_LEFT_B);

  if (val == LOW) {
    Direction_left = false; // Reverse
  }
  else {
    Direction_left = true; // Forward
  }
```

```

  if (Direction_right) {
    left_wheel_pulse_count++;
  }
  else {
    left_wheel_pulse_count--;
  }
}
```

B.1.6 Wheel_Tick_Counter_Code

```
// Encoder output to Arduino Interrupt pin. Tracks the tick count.
#define ENC_IN_LEFT_1_A 19
#define ENC_IN_RIGHT_1_A 18
#define ENC_IN_LEFT_2_A 2
#define ENC_IN_RIGHT_2_A 3


// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_1_B 53
#define ENC_IN_RIGHT_1_B 52
#define ENC_IN_LEFT_2_B 47
#define ENC_IN_RIGHT_2_B 46

// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;

// Minumum and maximum values for 16-bit integers
const int encoder_minimum = -32768;
const int encoder_maximum = 32767;

// Keep track of the number of wheel ticks
volatile int left_wheel_tick_1 = 0;
volatile int right_wheel_tick_1 = 0;
```

```
volatile int left_wheel_tick_2 = 0;
volatile int right_wheel_tick_2 = 0;


// One-second interval for measurements
int interval = 1000;
long previousMillis = 0;
long currentMillis = 0;

// Variable for RPM measuerment
float rpm_right_1 = 0;
float rpm_left_1 = 0;
float rpm_right_2 = 0;
float rpm_left_2 = 0;



// Variable for angular velocity measurement
float ang_velocity_right_1 = 0;
float ang_velocity_right_deg_1 = 0;
float ang_velocity_left_1 = 0;
float ang_velocity_left_deg_1 = 0;

float ang_velocity_right_2 = 0;
float ang_velocity_right_deg_2 = 0;
float ang_velocity_left_2 = 0;
float ang_velocity_left_deg_2 = 0;


void setup() {

  // Open the serial port at 9600 bps
  Serial.begin(9600);

  // Set pin states of the encoder
```

```
  pinMode(ENC_IN_LEFT_1_A , INPUT_PULLUP);
  pinMode(ENC_IN_LEFT_1_B , INPUT);
  pinMode(ENC_IN_RIGHT_1_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_1_B , INPUT);

  pinMode(ENC_IN_LEFT_2_A , INPUT_PULLUP);
  pinMode(ENC_IN_LEFT_2_B , INPUT);
  pinMode(ENC_IN_RIGHT_2_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_2_B , INPUT);

  // Every time the pin goes high, this is a tick
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_1_A),
    left_wheel_tick_1, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_1_A),
    right_wheel_tick_1, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_2_A),
    left_wheel_tick_2, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_2_A),
    right_wheel_tick_2, RISING);
}

void loop() {

  // Record the time
  currentMillis = millis();

  // If one second has passed, print the number of ticks
  if (currentMillis - previousMillis > interval) {

    previousMillis = currentMillis;

    Serial.println("Number of Ticks: ");
    Serial.println(right_wheel_tick_count);
    Serial.println(left_wheel_tick_count);
    Serial.println();
```

```
  }
}

// Increment the number of ticks
void right_wheel_tick_1() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_RIGHT_1_B);

  if(val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }

  if (Direction_right) {

    if (right_wheel_tick_count_1 == encoder_maximum) {
      right_wheel_tick_count_1 = encoder_minimum;
    }
    else {
      right_wheel_tick_count_1++;
    }
  }
  else {
    if (right_wheel_tick_count_1 == encoder_minimum) {
      right_wheel_tick_count_1 = encoder_maximum;
    }
    else {
      right_wheel_tick_count_1--;
    }
  }
}

```

```
// Increment the number of ticks
void left_wheel_tick_1() {

  // Read the value for the encoder for the left wheel
  int val = digitalRead(ENC_IN_LEFT_1_B);

  if(val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count == encoder_maximum) {
      left_wheel_tick_count = encoder_minimum;
    }
    else {
      left_wheel_tick_count++;
    }
  }
  else {
    if (left_wheel_tick_count == encoder_minimum) {
      left_wheel_tick_count = encoder_maximum;
    }
    else {
      left_wheel_tick_count--;
    }
  }
}
```

B.2 ROS Files

B.2.1 AMAL_Robot

```
<launch>


 <!--<param name="/use_sim_time" value="true"/>-->
 <rosparam param="ticks_meter">2920</rosparam>

 <node pkg="differential_drive" type="diff_tf.py" name="odometry" output="screen">
     <!-- <remap from="rwheel" to="rwheelback" />
     <remap from="lwheel" to="lwheelback" />
     <remap from="rwheel2" to="rwheelfront" />
     <remap from="lwheel2" to="lwheelfront" /> -->
     <rosparam param="base_width">0.595</rosparam>
     <rosparam param="odom_frame_id" subst_value="True"> "/odom" </rosparam>
     <rosparam param="base_frame_id" subst_value="True"> "/base_link" </rosparam>
     <rosparam param="global_frame_id" subst_value="True"> "/map" </rosparam>
     <rosparam param="rate">30</rosparam>
 </node>
 <!-- <node pkg="localization_data_pub" type="odometry" name="odometry">
 </node> -->
 <!-- <node pkg="mpu_6050_driver" type="imu_node.py" name="imu_data" output="screen">
 </node> -->


 <node pkg="tf" type="static_transform_publisher" name="base_link_to_laser" args="0 0 0.4826 0 0 0 base_link laser 30" />
```

```
<!-- <node pkg="tf" type="static_transform_publisher" name="imu_broadcaster" args="0 0.1778 0.1524 0 0 0 base_link imu_link 30" /> -->
 <!-- <node pkg="tf" type="static_transform_publisher" name="base_link_broadcaster" args="0 0 0.03 0 0 0 map base_link 30" /> -->
<!-- <node pkg="tf" type="static_transform_publisher" name="odom_to_basefootprint" args="0 0 0 0 0 0 odom base_footprint 30" /> -->
<!-- <node pkg="tf" type="static_transform_publisher" name="map_to_odom" args="0 0 0 0 0 0 map odom 30" /> -->



<node pkg="differential_drive" type="pid_velocity.py" name="l_front_pid_velocity">
    <remap from="wheel" to="lwheelfront"/>
    <remap from="motor_cmd" to="lmotorfront"/>
    <remap from="wheel_vtarget" to="lwheelfront_vtarget"/>
    <remap from="wheel_vel" to="lwheelfront_vel"/>
    <rosparam param="Kp">54</rosparam> <!--217.5-->
    <rosparam param="Ki">0</rosparam> <!--50-->
    <rosparam param="Kd">0</rosparam> <!--0.001-->
    <rosparam param="out_min">-255</rosparam>
    <rosparam param="out_max">255</rosparam>
    <rosparam param="rate">30</rosparam>
    <rosparam param="timeout_ticks">4</rosparam>
    <rosparam param="rolling_pts">5</rosparam>
</node>

<node pkg="differential_drive" type="pid_velocity.py" name="r_front_pid_velocity">
    <remap from="wheel" to="rwheelfront"/>
    <remap from="motor_cmd" to="rmotorfront"/>
    <remap from="wheel_vtarget" to="rwheelfront_vtarget"/>
```

```
    <remap from="wheel_vel" to="rwheelfront_vel"/>
    <rosparam param="Kp">80</rosparam> <!--322.5-->
    <rosparam param="Ki">0</rosparam> <!--50-->
    <rosparam param="Kd">0</rosparam> <!--0.001-->
    <rosparam param="out_min">-255</rosparam>
    <rosparam param="out_max">255</rosparam>
    <rosparam param="rate">30</rosparam>
    <rosparam param="timeout_ticks">4</rosparam>
    <rosparam param="rolling_pts">5</rosparam>
</node>

<node pkg="differential_drive" type="pid_velocity.py" name="
  l_back_pid_velocity">
    <remap from="wheel" to="lwheelback"/>
    <remap from="motor_cmd" to="lmotorback"/>
    <remap from="wheel_vtarget" to="lwheelback_vtarget"/>
    <remap from="wheel_vel" to="lwheelback_vel"/>
    <rosparam param="Kp">65</rosparam> <!--262.5-->
    <rosparam param="Ki">0</rosparam> <!--50-->
    <rosparam param="Kd">0</rosparam> <!--0.001-->
    <rosparam param="out_min">-255</rosparam>
    <rosparam param="out_max">255</rosparam>
    <rosparam param="rate">30</rosparam>
    <rosparam param="timeout_ticks">4</rosparam>
    <rosparam param="rolling_pts">5</rosparam>
</node>

<node pkg="differential_drive" type="pid_velocity.py" name="
  r_back_pid_velocity">
    <remap from="wheel" to="rwheelback"/>
    <remap from="motor_cmd" to="rmotorback"/>
    <remap from="wheel_vtarget" to="rwheelback_vtarget"/>
    <remap from="wheel_vel" to="rwheelback_vel"/>
    <rosparam param="Kp">43</rosparam> <!--172.5-->
    <rosparam param="Ki">0</rosparam> <!--50-->
```

```
        <rosparam param="Kd">0</rosparam> <!--0.001-->
        <rosparam param="out_min">-255</rosparam>
        <rosparam param="out_max">255</rosparam>
        <rosparam param="rate">30</rosparam>
        <rosparam param="timeout_ticks">4</rosparam>
        <rosparam param="rolling_pts">5</rosparam>
    </node>

    <!-- Extended Kalman Filter from robot_pose_ekf Node-->
    <!-- Subscribe: /odom, /imu_data, /vo -->
    <!-- Publish: /robot_pose_ekf/odom_combined -->
    <!-- <remap from="odom" to="odom_data_quat" /> -->

    <!-- <remap from="imu_data" to="imu/data" />
    <node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf">
      <param name="output_frame" value="odom"/>
      <param name="base_footprint_frame" value="base_footprint"/>
      <param name="freq" value="100.0"/>
      <param name="sensor_timeout" value="1.0"/>
      <param name="odom_used" value="true"/>
      <param name="imu_used" value="false"/>
      <param name="vo_used" value="false"/>
      <param name="gps_used" value="false"/>
      <param name="debug" value="false"/>
      <param name="self_diagnose" value="false"/>
    </node> -->

    <node pkg="localization_data_pub" type="rviz_click_to_2d" name="rviz_click_to_2d">
    </node>


    <node name="rplidarNode"          pkg="rplidar_ros"  type="rplidarNode" output="screen">
```

```
    <param name="serial_port"         type="string"  value="/dev/ttyUSB0"/>
    <param name="serial_baudrate"     type="int"     value="115200"/><!--A1/A2 -->
    <!--param name="serial_baudrate"  type="int"     value="256000"--><!--A3 -->
    <param name="frame_id"            type="string"  value="laser"/>
    <param name="inverted"            type="bool"    value="false"/>
    <param name="angle_compensate"    type="bool"    value="true"/>
  </node>


  <node pkg="rosserial_python" type="serial_node.py" name="serial_node">
      <param name="port" value="/dev/ttyACM0"/>
      <param name="baud" value="115200"/>
  </node>


<!--    <node type="rviz" name="rviz" pkg="rviz" args="-d $~/catkin_ws/rviz_load_template.rviz" /> -->
<node type="rviz" name="rviz" pkg="rviz" args="-d $(find amal_robot)/muzzu.rviz" />


<!-- <node pkg="rviz" type="rviz" name="rviz">
</node>  -->

 <!-- <arg name="map_file" default="/home/amal/catkin_ws/maps/my_map.yaml"/>
<node pkg="map_server" name="map_server" type="map_server" args="$(arg map_file)" /> -->

<node pkg="rqt_robot_steering" type="rqt_robot_steering" name="rqt_robot_steering">
```

```
    </node>

  <node pkg="differential_drive" type="twist_to_motors.py" name="twist_to_motors" output="screen">
      <rosparam param="base_width">0.595</rosparam>
  </node>

<!-- <node pkg="differential_drive" type="goals.py" name="goals" output="screen">
  </node> -->


<include file="$(find amcl)/examples/amcl_diff.launch"/>
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find amal_robot)/param/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find amal_robot)/param/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find amal_robot)/param/local_costmap_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find amal_robot)/param/global_costmap_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find amal_robot)/param/base_local_planner_params.yaml" command="load" />
  </node>


</launch>
```

B.2.2 rplidar

```
<launch>
  <node name="rplidarNode"          pkg="rplidar_ros"  type="rplidarNode" output="screen">
  <param name="serial_port"         type="string" value="/dev/ttyUSB0
```

```
"/>
  <param name="serial_baudrate"     type="int"    value="115200"/><!--A1/A2 -->
  <!--param name="serial_baudrate"     type="int"    value="256000"--><!--A3 -->
  <param name="frame_id"            type="string" value="laser"/>
  <param name="inverted"            type="bool"   value="false"/>
  <param name="angle_compensate"    type="bool"   value="true"/>
  <param name="scan_mode"           type="string"  value="Standard"/>
  </node>
</launch>
```

B.2.3 Firebase_Integration_Files

```python
#!/usr/bin/env python


import rospy
import roslib
roslib.load_manifest('differential_drive')
from math import sin, cos, pi
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
from nav_msgs.msg import Odometry
from tf.broadcaster import TransformBroadcaster
from std_msgs.msg import Int16
from geometry_msgs.msg import PoseStamped

cred = credentials.Certificate("/home/amal/catkin_ws/src/differential_drive/serviceAccountKey.json")
firebase_admin.initialize_app(cred)

db = firestore.client()

class goals:
```

```


    def __init__(self):

        rospy.init_node("goals")
        self.nodename = rospy.get_name()
        rospy.loginfo("-I- %s started" % self.nodename)
        #### parameters #######
        self.rate = rospy.get_param('~rate',10.0)  # the rate at which to publish the transform

        self.directory = {'Charging Base':[0,0,0], 'Cafeteria Counter':[2.35834,-6.18966,0.0], 'Table 1':[-1.74200,-0.995179,0],'Table 1':[-3.00177,4.98015,0]}



        self.then = rospy.Time.now()


        self.GoalPub = rospy.Publisher("move_base_simple/goal", PoseStamped, queue_size=10)



    def update(self):

 now = rospy.Time.now() #Gets the current time
 #print(now)
 doc = db.collection('orders').document('test').get().to_dict()
 #print(doc)
 flag = doc['flag']

 if flag == True:
   current_goal = PoseStamped()
```

```
    location = 'Table 1'
    #print(flag, location)
    current_goal.header.frame_id = "map"
    current_goal.header.stamp.secs = now.secs
    current_goal.header.stamp.nsecs = now.nsecs
    current_goal.pose.orientation.x = 0
    current_goal.pose.orientation.y = 0
    current_goal.pose.orientation.z = 1
    current_goal.pose.orientation.w = 0
    current_goal.pose.position.x = self.directory[location][0]
    current_goal.pose.position.y = self.directory[location][1]
    current_goal.pose.position.z = self.directory[location][2]
    self.GoalPub.publish(current_goal)
  elif flag == False:
    current_goal = PoseStamped()
    location = 'Cafeteria Counter'
    #print(flag, location)
    current_goal.header.frame_id = "map"
    current_goal.header.stamp.secs = now.secs
    current_goal.header.stamp.nsecs = now.nsecs
    current_goal.pose.orientation.x = 0
    current_goal.pose.orientation.y = 0
    current_goal.pose.orientation.z = 1
    current_goal.pose.orientation.w = 0
    current_goal.pose.position.x = self.directory[location][0]
    current_goal.pose.position.y = self.directory[location][1]
    current_goal.pose.position.z = self.directory[location][2]
    self.GoalPub.publish(current_goal)


    def spin(self):

        r = rospy.Rate(self.rate)
        while not rospy.is_shutdown():
            self.update()
```

```
            r.sleep()

if __name__ == '__main__':
    """ main """
    try:
        Goal = goals()
        Goal.spin()
    except rospy.ROSInterruptException:
        pass
```

```
#!/usr/bin/env python
import rospy
import roslib
import time
roslib.load_manifest('differential_drive')
from math import sin, cos, pi
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
from geometry_msgs.msg import Quaternion
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from std_msgs.msg import Bool
from tf.broadcaster import TransformBroadcaster
from std_msgs.msg import Int16

cred = credentials.Certificate("/home/amal/catkin_ws/src/
    differential_drive/serviceAccountKey.json")
firebase_admin.initialize_app(cred)

db = firestore.client()



class DiffTf:

```

```
    def __init__(self):

        rospy.init_node("diff_tf")
        self.nodename = rospy.get_name()
        rospy.loginfo("-I- %s started" % self.nodename)
        self.flag = False
        self.rate = rospy.get_param('~rate',10.0)
        self.ticks_meter = float(rospy.get_param('ticks_meter', 50))
        self.base_width = float(rospy.get_param('~base_width', 0.245)
  )

        self.base_frame_id = rospy.get_param('~base_frame_id','
  base_link')
        self.odom_frame_id = rospy.get_param('~odom_frame_id', 'odom'
  )

        self.encoder_min = rospy.get_param('encoder_min', -32768)
 #cred = credentials.Certificate("serviceAccountKey.json")
 #firebase_admin.initialize_app(cred)

 #db = firestore.client()ram('encoder_min', -32768)
        self.encoder_max = rospy.get_param('encoder_max', 32768)
        self.encoder_low_wrap = rospy.get_param('wheel_low_wrap', (
  self.encoder_max - self.encoder_min) * 0.3 + self.encoder_min )
        self.encoder_high_wrap = rospy.get_param('wheel_high_wrap', (
  self.encoder_max - self.encoder_min) * 0.7 + self.encoder_min )

        self.t_delta = rospy.Duration(1.0/self.rate)
        self.t_next = rospy.Time.now() + self.t_delta

        # internal data
        self.enc_bleft = None        # wheel encoder readings
        self.enc_bright = None
 self.enc_fleft = None         # wheel encoder readings
```

```
        self.enc_fright = None

self.bleft = 0              # actual values coming back from robot
        self.bright = 0
self.fleft = 0
        self.fright = 0

        self.blmult = 0
        self.brmult = 0
self.flmult = 0
        self.frmult = 0

self.covar = Odometry()

        self.prev_blencoder = 0
        self.prev_brencoder = 0
self.prev_flencoder = 0
        self.prev_frencoder = 0

        self.x = 0                  # position in xy plane
        self.y = 0

        self.th = 0
        self.dx = 0                 # speeds in x/rotation
        self.dr = 0

        self.then = rospy.Time.now()

        # subscriptions
        rospy.Subscriber("lwheelback", Int16, self.BLwheelCallback)
        rospy.Subscriber("rwheelback", Int16, self.BRwheelCallback)
rospy.Subscriber("lwheelfront", Int16, self.FLwheelCallback)
        rospy.Subscriber("rwheelfront", Int16, self.FRwheelCallback)
rospy.Subscriber("odom", Odometry, self.recall_cov)
        self.odomPub = rospy.Publisher("odom", Odometry, queue_size
```

```
      =10)
    self.flagger = rospy.Publisher("flag_firebase",Bool , queue_size
      =10)
          self.odomBroadcaster = TransformBroadcaster()


    def spin(self):
        r = rospy.Rate(self.rate)
        while not rospy.is_shutdown():
            self.update()
      #self.create_con()
            r.sleep()



    def update(self):

        now = rospy.Time.now()
    #print(self.flag)
    if self.flag == True:
      #print('Changing State to False')
      self.flag = False
    if self.flag == False:
      self.create_con()
    #if flag:
     #   self.create_con()
      #  flag = False

        if now > self.t_next:
            elapsed = now - self.then
            self.then = now
            elapsed = elapsed.to_sec()

            # calculate odometry
            if self.enc_bleft == None:
```

```
                d_left = 0
                d_right = 0
            else:
    left_dis = (self.bleft + self.fleft)/2
    right_dis = (self.bright + self.fright)/2
    left_E = (self.enc_bleft + self.enc_fleft)/2
    right_E = (self.enc_bright + self.enc_fright)/2

                d_left = (left_dis - left_E) / self.ticks_meter
                d_right = (right_dis - right_E) / self.ticks_meter

            self.enc_bleft = self.bleft
            self.enc_bright = self.bright
      self.enc_fleft = self.fleft
            self.enc_fright = self.fright

            # distance traveled is the average of the two wheels
            d = ( d_left + d_right ) / 2
            # this approximation works (in radians) for small angles
            th = ( d_right - d_left ) / self.base_width
            # calculate velocities
            self.dx = d / elapsed
            self.dr = th / elapsed



            if (d != 0):
                # calculate distance traveled in x and y
                x = cos( th ) * d
                y = -sin( th ) * d
                # calculate the final position of the robot
                self.x = self.x + ( cos( self.th ) * x - sin( self.th
    ) * y )
                self.y = self.y + ( sin( self.th ) * x + cos( self.th
    ) * y )
            if( th != 0):
```

```
                self.th = self.th + th

            # publish the odom information
            quaternion = Quaternion()
            quaternion.x = 0.0
            quaternion.y = 0.0
            quaternion.z = sin( self.th / 2 )
            quaternion.w = cos( self.th / 2 )
            self.odomBroadcaster.sendTransform(
                (self.x, self.y, 0),
                (quaternion.x, quaternion.y, quaternion.z, quaternion.w),
                rospy.Time.now(),
                self.base_frame_id,
 self.odom_frame_id
                )

            odom = Odometry()
            odom.header.stamp = now
            odom.header.frame_id = self.odom_frame_id
            odom.pose.pose.position.x = self.x
            odom.pose.pose.position.y = self.y
            odom.pose.pose.position.z = 0
            odom.pose.pose.orientation = quaternion
            odom.child_frame_id = self.base_frame_id
            odom.twist.twist.linear.x = self.dx
            odom.twist.twist.linear.y = 0
            odom.twist.twist.angular.z = self.dr
      odom.pose.covariance = list(self.covar.pose.covariance)
      if self.covar.twist.twist.linear.x != odom.twist.twist.linear.x or self.covar.twist.twist.linear.y != odom.twist.twist.linear.y or self.covar.pose.covariance[0] == 0:
          for i in range(36):
              if i == 0 or i == 7 or i == 14:
          odom.pose.covariance[i] = 0.01
```

```
        elif i == 21 or i == 28 or i == 35:
          odom.pose.covariance[i] += 0.01
        else:
          odom.pose.covariance[i] = 0

             self.odomPub.publish(odom)


  def create_con(self):


self.flag = True

#cred = credentials.Certificate("serviceAccountKey.json")
#firebase_admin.initialize_app(cred)
#db = firestore.client()
#querying
#Adding
#data = {'name':'Chomu', 'age':69, 'flag' : False}
#db.collection('persons').add(data)
#time.sleep(10)
#Reading
#result = db.collection('persons').document("ccJ2gMzZ0SIpMgcPHOHP")
  .get()

# if result.exists:
#       print(result.to_dict())

#getting all res
#print(self.flag, "Inside Con")
docs = db.collection('users').get()
#print (docs[1].id)

for doc in docs:
  temp = doc.to_dict()
```

```
        self.flag = temp['flag']
        #print(temp)

    self.flagger.publish(self.flag)


    def BLwheelCallback(self, msg):

        enc = msg.data
        if (enc < self.encoder_low_wrap and self.prev_blencoder > self.encoder_high_wrap):
            self.blmult = self.blmult + 1

        if (enc > self.encoder_high_wrap and self.prev_blencoder < self.encoder_low_wrap):
            self.blmult = self.blmult - 1

        self.bleft = 1.0 * (enc + self.blmult * (self.encoder_max - self.encoder_min))
        self.prev_blencoder = enc

    def BRwheelCallback(self, msg):

        enc = msg.data
        if(enc < self.encoder_low_wrap and self.prev_brencoder > self.encoder_high_wrap):
            self.brmult = self.brmult + 1

        if(enc > self.encoder_high_wrap and self.prev_brencoder < self.encoder_low_wrap):
            self.brmult = self.brmult - 1

        self.bright = 1.0 * (enc + self.brmult * (self.encoder_max - self.encoder_min))
        self.prev_brencoder = enc
```

```
    def FLwheelCallback(self, msg):

        enc = msg.data
        if (enc < self.encoder_low_wrap and self.prev_flencoder > self.encoder_high_wrap):
            self.flmult = self.flmult + 1

        if (enc > self.encoder_high_wrap and self.prev_flencoder < self.encoder_low_wrap):
            self.flmult = self.flmult - 1

        self.fleft = 1.0 * (enc + self.flmult * (self.encoder_max - self.encoder_min))
        self.prev_flencoder = enc

    def FRwheelCallback(self, msg):

        enc = msg.data
        if(enc < self.encoder_low_wrap and self.prev_frencoder > self.encoder_high_wrap):
            self.frmult = self.frmult + 1

        if(enc > self.encoder_high_wrap and self.prev_frencoder < self.encoder_low_wrap):
            self.frmult = self.frmult - 1

        self.fright = 1.0 * (enc + self.frmult * (self.encoder_max - self.encoder_min))
        self.prev_frencoder = enc



    def recall_cov(self, msg):
```

```
  self.covar = msg


if __name__ == '__main__':
    """ main """
    try:
        diffTf = DiffTf()
  #IoTnode = IoTNode()
        diffTf.spin()
    except rospy.ROSInterruptException:
        pass
```

```
#!/usr/bin/env python

import rospy
import roslib

from std_msgs.msg import Int16
from std_msgs.msg import Float32
from numpy import array



class PidVelocity():



    def __init__(self):

        rospy.init_node("pid_velocity")
        self.nodename = rospy.get_name()
        rospy.loginfo("%s started" % self.nodename)

        ### initialize variables
        self.target = 0
        self.motor = 0
```

```
        self.vel = 0
        self.integral = 0
        self.error = 0
        self.derivative = 0
        self.previous_error = 0
        self.wheel_prev = 0
        self.wheel_latest = 0
        self.then = rospy.Time.now()
        self.wheel_mult = 0
        self.prev_encoder = 0

        ### get parameters ####
        self.Kp = rospy.get_param('~Kp',10)
        self.Ki = rospy.get_param('~Ki',10)
        self.Kd = rospy.get_param('~Kd',0.001)
        self.out_min = rospy.get_param('~out_min',-255)
        self.out_max = rospy.get_param('~out_max',255)
        self.rate = rospy.get_param('~rate',30)
        self.rolling_pts = rospy.get_param('~rolling_pts',2)
        self.timeout_ticks = rospy.get_param('~timeout_ticks',4)
        self.ticks_per_meter = rospy.get_param('ticks_meter', 20)
        self.vel_threshold = rospy.get_param('~vel_threshold', 0.001)
        self.encoder_min = rospy.get_param('encoder_min', -32768)
        self.encoder_max = rospy.get_param('encoder_max', 32768)
        self.encoder_low_wrap = rospy.get_param('wheel_low_wrap', (self.encoder_max - self.encoder_min) * 0.3 + self.encoder_min )
        self.encoder_high_wrap = rospy.get_param('wheel_high_wrap', (self.encoder_max - self.encoder_min) * 0.7 + self.encoder_min )
        self.prev_vel = [0.0] * self.rolling_pts
        self.wheel_latest = 0.0
        self.prev_pid_time = rospy.Time.now()
        rospy.logdebug("%s got Kp:%0.3f Ki:%0.3f Kd:%0.3f tpm:%0.3f" % (self.nodename, self.Kp, self.Ki, self.Kd, self.ticks_per_meter))

```

```python
        #### subscribers/publishers
        rospy.Subscriber("wheel", Int16, self.wheelCallback)
        rospy.Subscriber("wheel_vtarget", Float32, self.targetCallback)
        self.pub_motor = rospy.Publisher('motor_cmd',Float32, queue_size=10)
        self.pub_vel = rospy.Publisher('wheel_vel', Float32, queue_size=10)


    def spin(self):

        self.r = rospy.Rate(self.rate)
        self.then = rospy.Time.now()
        self.ticks_since_target = self.timeout_ticks
        self.wheel_prev = self.wheel_latest
        self.then = rospy.Time.now()
        while not rospy.is_shutdown():
            self.spinOnce()
            self.r.sleep()


    def spinOnce(self):

        self.previous_error = 0.0
        self.prev_vel = [0.0] * self.rolling_pts
        self.integral = 0.0
        self.error = 0.0
        self.derivative = 0.0
        self.vel = 0.0

        # only do the loop if we've recently recieved a target velocity message
        while not rospy.is_shutdown() and self.ticks_since_target < self.timeout_ticks:
```

```
            self.calcVelocity()
            self.doPid()
            self.pub_motor.publish(self.motor)
            self.r.sleep()
            self.ticks_since_target += 1
            if self.ticks_since_target == self.timeout_ticks:
                self.pub_motor.publish(0)


    def calcVelocity(self):

        self.dt_duration = rospy.Time.now() - self.then
        self.dt = self.dt_duration.to_sec()
        rospy.logdebug("-D- %s caclVelocity dt=%0.3f wheel_latest=%0.3f wheel_prev=%0.3f" %
                       (self.nodename, self.dt, self.wheel_latest, self.wheel_prev))

        if (self.wheel_latest == self.wheel_prev):
            cur_vel = (1 / self.ticks_per_meter) / self.dt
            if abs(cur_vel) < self.vel_threshold:
                # if the velocity is < threshold, consider our velocity 0
                rospy.logdebug("-D- %s below threshold cur_vel=%0.3f vel=0" % (self.nodename, cur_vel))
                self.appendVel(0)
                self.calcRollingVel()
            else:
                rospy.logdebug("-D- %s above threshold cur_vel=%0.3f" % (self.nodename, cur_vel))
                if abs(cur_vel) < self.vel:
                    rospy.logdebug("-D- %s cur_vel < self.vel" % self.nodename)

                    self.appendVel(cur_vel)
```

```
                        self.calcRollingVel()

        else:
            # we received a new wheel value
            cur_vel = (self.wheel_latest - self.wheel_prev) / self.dt
            self.appendVel(cur_vel)
            self.calcRollingVel()
            rospy.logdebug("-D- %s **** wheel updated vel=%0.3f ****
    " % (self.nodename, self.vel))
            self.wheel_prev = self.wheel_latest
            self.then = rospy.Time.now()

        self.pub_vel.publish(self.vel)

    def appendVel(self, val):

        self.prev_vel.append(val)
        del self.prev_vel[0]


    def calcRollingVel(self):

        p = array(self.prev_vel)
        self.vel = p.mean()


    def doPid(self):

        pid_dt_duration = rospy.Time.now() - self.prev_pid_time
        pid_dt = pid_dt_duration.to_sec()
        self.prev_pid_time = rospy.Time.now()

        self.error = self.target - self.vel
        self.integral = self.integral + (self.error * pid_dt)

```

```
        self.derivative = (self.error - self.previous_error) / pid_dt
        self.previous_error = self.error

        self.motor = (self.Kp * self.error) + (self.Ki * self.integral) + (self.Kd * self.derivative)

        if self.motor > self.out_max:
            self.motor = self.out_max
            self.integral = self.integral - (self.error * pid_dt)
        if self.motor < self.out_min:
            self.motor = self.out_min
            self.integral = self.integral - (self.error * pid_dt)

        if (self.target == 0):
            self.motor = 0

        rospy.logdebug("vel:%0.2f tar:%0.2f err:%0.2f int:%0.2f der:%0.2f ## motor:%d " %
                       (self.vel, self.target, self.error, self.integral, self.derivative, self.motor))





    def wheelCallback(self, msg):

        enc = msg.data
        if (enc < self.encoder_low_wrap and self.prev_encoder > self.encoder_high_wrap) :
            self.wheel_mult = self.wheel_mult + 1

        if (enc > self.encoder_high_wrap and self.prev_encoder < self.encoder_low_wrap) :
            self.wheel_mult = self.wheel_mult - 1
```

```


        self.wheel_latest = 1.0 * (enc + self.wheel_mult * (self.encoder_max - self.encoder_min)) / self.ticks_per_meter
        self.prev_encoder = enc




    def targetCallback(self, msg):

        self.target = msg.data
        self.ticks_since_target = 0



if __name__ == '__main__':
    """ main """
    try:
        pidVelocity = PidVelocity()
        pidVelocity.spin()
    except rospy.ROSInterruptException:
        pass
```

```
#!/usr/bin/env python


import rospy
import roslib
from std_msgs.msg import Float32
from geometry_msgs.msg import Twist

class TwistToMotors():


    def __init__(self):
```

```
        rospy.init_node("twist_to_motors")
        nodename = rospy.get_name()
        rospy.loginfo("%s started" % nodename)

        self.w = rospy.get_param("~base_width", 0.2)

        self.pub_lmotor_front = rospy.Publisher('lwheelfront_vtarget'
    , Float32, queue_size=10)
        self.pub_rmotor_front = rospy.Publisher('rwheelfront_vtarget'
    , Float32, queue_size=10)
        self.pub_lmotor_back = rospy.Publisher('lwheelback_vtarget',
    Float32, queue_size=10)
        self.pub_rmotor_back = rospy.Publisher('rwheelback_vtarget',
    Float32, queue_size=10)
        rospy.Subscriber('twist', Twist, self.twistCallback)


        self.rate = rospy.get_param("~rate", 50)
        self.timeout_ticks = rospy.get_param("~timeout_ticks", 2)
        self.left = 0
        self.right = 0

    def spin(self):

        r = rospy.Rate(self.rate)
        idle = rospy.Rate(10)
        then = rospy.Time.now()
        self.ticks_since_target = self.timeout_ticks

        ###### main loop ######
        while not rospy.is_shutdown():

            while not rospy.is_shutdown() and self.ticks_since_target
    < self.timeout_ticks:
```

```
                self.spinOnce()
                r.sleep()
            idle.sleep()


    def spinOnce(self):

        # dx = (l + r) / 2
        # dr = (r - l) / w

        self.right = 1.0 * self.dx + self.dr * self.w / 2
        self.left = 1.0 * self.dx - self.dr * self.w / 2
        # rospy.loginfo("publishing: (%d, %d)", left, right)
        # For now, we are publishing same signals to both
        self.pub_lmotor_front.publish(self.left)
  self.pub_lmotor_back.publish(self.left)
        self.pub_rmotor_front.publish(self.right)
        self.pub_rmotor_back.publish(self.right)
        self.ticks_since_target += 1

    def twistCallback(self,msg):
        # rospy.loginfo("-D- twistCallback: %s" % str(msg))
        self.ticks_since_target = 0
        self.dx = msg.linear.x
        self.dr = msg.angular.z
        self.dy = msg.linear.y

if __name__ == '__main__':
    """ main """
    try:
        twistToMotors = TwistToMotors()
        twistToMotors.spin()
    except rospy.ROSInterruptException:
        pass
```

B.2.4 Navigation Parameters

```
TrajectoryPlannerROS:
  max_vel_x: 0.24
  min_vel_x: -0.24
  max_vel_theta: 0.13
  min_vel_theta: -0.13
  min_in_place_vel_theta: 0.2
  backup_vel: -0.2
 #acc_lim_theta: 3.2
 #acc_lim_x: 2.5
 #acc_lim_Y: 2.5

  holonomic_robot: false

  meter_scoring: true

  xy_goal_tolerance: 0.2
  yaw_goal_tolerance: 0.3
  transform_tolerance: 5
```

```
obstacle_range: 3
raytrace_range: 8.5
footprint: [[-0.32, -0.32], [-0.32, 0.32], [0.32, 0.32], [0.32,
    -0.32]]
map_topic: /map
subscribe_to_updates: true
global_frame: map
robot_base_frame: base_link
update_frequency: 100.0
publish_frequency: 100.0
rolling_window: false
transform_tolerance: 5

plugins:
  - {name: static_layer, type: "costmap_2d::StaticLayer"}
```

```
  - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
  - {name: inflation_layer, type: "costmap_2d::InflationLayer"}

static_layer:
  map_topic: /map
  subscribe_to_updates: false

obstacle_layer:
    observation_sources: laser_scan_sensor
    laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan,
   topic: scan, marking: true, clearing: true}

inflation_layer:
  inflation_radius: 0.3
```

```
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 100.0
  publish_frequency: 100.0
  transform_tolerance: 5
  resolution: 0.1
```

```
local_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 100.0
  publish_frequency: 100.0
  transform_tolerance: 5
  static_map: false
  rolling_window: true
  resolution: 0.05
  inflation_radius: 0.1
  width: 0.5
  height: 0.5

```

```
  plugins:
    - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}

  obstacle_layer:
    observation_sources: laser_scan_sensor
    laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan,
   topic: scan, marking: true, clearing: true}
```

B.3 Application Code

B.3.1 main.dart

```
import 'package:app/screens/signup_screen.dart';
import 'package:flutter/material.dart';
import 'package:app/screens/auth_screen.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  final database = await openDatabase(
    join(await getDatabasesPath(), "app_database_final.db"),
    onCreate: (db, version) {
      return db.execute(
          'CREATE TABLE shoppingcart(itemId TEXT PRIMARY KEY, name
   TEXT, price INTEGER, imageUrl TEXT, description TEXT, category
   TEXT, quantity INT)');
    },
    version: 1,
  );
  runApp(MyApp(database));
}

```

```
class MyApp extends StatelessWidget {
  Database database;
  MyApp(this.database, {Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        scaffoldBackgroundColor: Colors.orange[50],
        textTheme: GoogleFonts.spartanTextTheme(),
      ),
      home: MyHomePage(database)
    );
  }
}

class MyHomePage extends StatelessWidget {
  Database database;
  // Future<void> updateDB() async {
  //   final db = await database;
  //   db.execute("ALTER TABLE shoppingcart ADD quantity INT");
  //   print("DB updated");
  // }
  MyHomePage(this.database,{ Key? key }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
          child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: <Widget>[
          Image.asset('assets/images/logo.jpg', width: 190, height:
    190),
          Container(
```

```
          margin: EdgeInsets.only(bottom: 10, top: 0),
          child: Text(’AMAL’, style: TextStyle(fontFamily: ’Pacifico’,fontSize: 30,color: Colors.black54,letterSpacing: 2)),
        ),
        Container(
          margin: EdgeInsets.only(bottom: 10, top: 0),
          child: Text(’A Food Delivery Service’, style: TextStyle(fontFamily: ’Pacifico’,fontSize: 25,color: Colors.black54,letterSpacing: 1)),
        ),
        Container(
          width: 200,
          margin: EdgeInsets.only(bottom: 0),
          child: ElevatedButton( child: Text("Sign in"), onPressed: (){
            Navigator.push(context,
                MaterialPageRoute(builder: (context) => Auth_Screen(database)));
         }),
        ),
        Container(
          width: 200,
          padding: EdgeInsets.all(0),
          child: ElevatedButton( child: Text(’Sign Up’), onPressed: (){
            Navigator.push(context,
                MaterialPageRoute(builder: (context) => SignUpScreen(database)));
           }),
        ),
        // Container(
        //   width: 200,
        //   padding: EdgeInsets.all(0),
        //   child: ElevatedButton( child: Text(’Update DB’), onPressed: (){
```

```
        //     updateDB();
        //   }),
        // ),
      ],
    )),
    backgroundColor: Color(0xffF4F7FA),
  );
 }
}
```

B.3.2 Data Models

ItemData

```
class ItemData {
 final String itemId;
 final String name;
 final int price;
 final String imageUrl;
 final String description;
 final String category;
 int quantity;

 ItemData(
   this.quantity,
     {required this.itemId,
     required this.name,
     required this.price,
     required this.imageUrl,
     required this.description,
     required this.category,});
 ItemData.fromJson(Map<dynamic, dynamic> json, this.quantity)
     : name = json['name'] as String,
       price = json['price'] as int,
       imageUrl = json['imageUrl'] as String,
       itemId = json['itemId'] as String,
```

```
        description = json['description'] as String,
        category = json['category'] as String;
  Map<dynamic, dynamic> toJson() => <dynamic, dynamic>{
        'name': name,
        'price': price,
        'imageUrl': imageUrl,
        'itemId': itemId,
        'description': description,
        'category': category,
      };
  Map<String, dynamic> toMap() {
    return {
      'itemId': itemId,
      'name': name,
      'price': price,
      'imageUrl': imageUrl,
      'description': description,
      'category': category,
      'quantity': quantity,
    };
  }
}
```

UserData

```
import 'package:firebase_database/firebase_database.dart';
class UserData {
  final String name;
  final String email;
  final String password;

  UserData(this.name, this.email, this.password);

  UserData.fromJson(Map<dynamic, dynamic> json)
  : name = json['name'] as String,
    email = json['email'] as String,
```

```
    password = json['password'] as String;
  Map<dynamic,dynamic> toJson() => <dynamic,dynamic> {
    'name': name,
    'email': email,
    'password': password,
  };
}
```

B.3.3 Screens

Authentication Screen

```
import 'package:app/screens/menu_screen.dart';
import 'package:app/screens/signup_screen.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_database/firebase_database.dart';
import "package:flutter/material.dart";
import 'package:app/screens/home_screen.dart';
import 'package:sqflite/sqflite.dart';

class Auth_Screen extends StatefulWidget {
  Database database;
  Auth_Screen(this.database);

  @override
  _Auth_ScreenState createState() => _Auth_ScreenState(this.database)
    ;
}

class _Auth_ScreenState extends State<Auth_Screen> {
  final TextEditingController _emailController =
    TextEditingController();
  final TextEditingController _passwordController =
    TextEditingController();
  bool pass = true;
  Database database;
```

```
_Auth_ScreenState(this.database);

Future SignIn() async {
  try {
    await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: _emailController.text, password: _passwordController.text);
    Navigator.push(context,
        MaterialPageRoute(builder: (context) => MenuScreen(database)));
  } on FirebaseAuthException catch (e) {
    if (e.code == 'user-not-found') {
      showAlertDialog1(context);
    } else if (e.code == 'wrong-password') {
      showAlertDialog2(context);
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      elevation: 0,
      backgroundColor: Colors.white,
      centerTitle: true,
      title: Text('Sign In',
          style: TextStyle(
              color: Colors.grey, fontFamily: 'Poppins', fontSize: 25)),
    ),
    body: ListView(
      shrinkWrap: true,
      children: <Widget>[
        Container(
```

```
      padding: EdgeInsets.only(left: 18, right: 18),
      child: Stack(
        children: <Widget>[
          Column(
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              Text('Welcome Back!', style: TextStyle(color: Colors.
black,fontSize: 24,fontWeight: FontWeight.w800,fontFamily: '
Poppins')),
              Container(
                margin: EdgeInsets.only(top: 13),
                child: TextField(
                  controller: _emailController,
                  cursorColor: Color(0xff44c662),
                  style: TextStyle(fontFamily: 'Poppins',
fontWeight: FontWeight.w500),
                  decoration: InputDecoration(
                      labelText: "Email",
                      hintStyle: TextStyle(fontFamily: 'Poppins',
color: Color(0xff444444)),
                      focusedBorder: OutlineInputBorder(
borderRadius: BorderRadius.all(Radius.circular(6)),borderSide:
BorderSide(color: Color(0xff44c662),)),
                      contentPadding: EdgeInsets.symmetric(
horizontal: 20, vertical: 10),
                      border: OutlineInputBorder(gapPadding: 0,
borderRadius: BorderRadius.all(Radius.circular(6)))),
                ),
              ),
              Container(
                margin: EdgeInsets.only(top: 13),
                child: TextField(
                  controller: _passwordController,
                  cursorColor: Color(0xff44c662),
```

```
          style: TextStyle(fontFamily: 'Poppins', fontWeight: FontWeight.w500),
         obscureText: pass,
          decoration: InputDecoration(
            suffixIcon: IconButton(
             onPressed: () {
              setState(() {
               pass = !pass;
              });
             },
             icon: Icon(
               pass ? Icons.visibility_off : Icons.visibility)),
            labelText: "Password",
            hintStyle: TextStyle(fontFamily: 'Poppins', color: Color(0xff444444)),
            focusedBorder: OutlineInputBorder(borderRadius: BorderRadius.all(Radius.circular(6)),borderSide: BorderSide(color: Color(0xff44c662),)),
            contentPadding: EdgeInsets.symmetric(horizontal: 20, vertical: 10),
            border: OutlineInputBorder(gapPadding: 0, borderRadius: BorderRadius.all(Radius.circular(6)))),
         ),
        ),
        Row(
          children: <Widget>[
           const Text('Do not have an account?'),
           TextButton(
            child: const Text(
             'Sign up',
             style: TextStyle(fontSize: 16),
            ),
            onPressed: () {
             Navigator.push(
```

```
                              context,
                              MaterialPageRoute(
                                  builder: (context) =>
                                      SignUpScreen(database)));
                        },
                      )
                    ],
                    mainAxisAlignment: MainAxisAlignment.center,
                  )
              ],
            ),
            Positioned(
              bottom: 15,
              right: -15,
              child: FlatButton(
                onPressed: ()async {
                  await SignIn();
                    //Navigator.pushReplacement(context,
PageTransition(type: PageTransitionType.rightToLeft, child:
Dashboard()));
                },
                color: Color(0xff44c662),
                padding: EdgeInsets.all(13),
               shape: CircleBorder(),
                child: Icon(Icons.arrow_forward, color: Colors.white)
,
              ),
            ),
          ],
        ),
        height: 245,

        width: double.infinity,
        decoration: BoxDecoration(
    color: Colors.white,
```

```
    boxShadow: [
      BoxShadow(
          color: Color.fromRGBO(0, 0, 0, .1),
          blurRadius: 10,
          spreadRadius: 5,
          offset: Offset(0, 1))
    ],
    borderRadius: BorderRadiusDirectional.only(
        bottomEnd: Radius.circular(20), bottomStart: Radius.circular
    (20))),
      ),
        ],
      )
    );
  }
}

showAlertDialog1(BuildContext context) {
  // set up the button
  Widget okButton = ElevatedButton(
    child: Text("OK"),
    onPressed: () {
      Navigator.pop(context);
    },
  );
  // set up the AlertDialog
  AlertDialog alert = AlertDialog(
    title: Text("Sign in Failed"),
    content: Text("This user does not exist"),
    actions: [
      okButton,
    ],
  );
  // show the dialog
  showDialog(
```

```
    context: context,
    builder: (BuildContext context) {
      return alert;
    },
  );
}

showAlertDialog2(BuildContext context) {
  // set up the button
  Widget okButton = ElevatedButton(
    child: Text("OK"),
    onPressed: () {
      Navigator.pop(context);
    },
  );
  // set up the AlertDialog
  AlertDialog alert = AlertDialog(
    title: Text("Sign in Failed"),
    content: Text("Wrong password"),
    actions: [
      okButton,
    ],
  );
  // show the dialog
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return alert;
    },
  );
}
```

Signup Screen

```
import 'package:app/screens/auth_screen.dart';
import "package:flutter/material.dart";
```

```
import ’package:firebase_auth/firebase_auth.dart’;
import ’package:firebase_database/firebase_database.dart’;
import ’package:cloud_firestore/cloud_firestore.dart’;
import ’package:app/data_models/users.dart’;
import ’package:sqflite/sqflite.dart’;

class SignUpScreen extends StatefulWidget {
  Database database;
  SignUpScreen(this.database);

  @override
  _SignUpScreenState createState() => _SignUpScreenState(this.database);
}

class _SignUpScreenState extends State<SignUpScreen> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final TextEditingController _nameController = TextEditingController();
  bool pass = true;
  Database database;
  _SignUpScreenState(this.database);
  Future SignUp() async {
    try {
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
          email: _emailController.text, password: _passwordController.text);
      UserData user = UserData(_nameController.text, _emailController.text,
          _passwordController.text);
      CollectionReference users =
          FirebaseFirestore.instance.collection(’users’);
```

```
    users
        .doc(user.email)
        .set({'name': user.name, 'email': user.email, 'flag': "True"});
    // final DatabaseReference _userdataref =
    //     FirebaseDatabase.instance.ref().child('users');
    // _userdataref.push().set(user.toJson());
    showAlertDialog1(context, database);
  } on FirebaseAuthException catch (e) {
    if (e.code == 'weak-password') {
      // print('The password provided is too weak.');
    } else if (e.code == 'email-already-in-use') {
      showAlertDialog2(context);
    }
  } catch (e) {
    print(e);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      elevation: 0,
      backgroundColor: Colors.white,
      centerTitle: true,
      title: Text('Sign Up',
          style: TextStyle(
              color: Colors.grey, fontFamily: 'Poppins', fontSize: 25)),
    ),
    body: ListView(
      shrinkWrap: true,
      children: <Widget>[
        Container(
```

```
      padding: EdgeInsets.only(left: 18, right: 18),
      child: Stack(
        children: <Widget>[
          Column(
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              Text(’Welcome to AMAL!’, style: TextStyle(color: Colors.black,fontSize: 24,fontWeight: FontWeight.w800,fontFamily: ’Poppins’)),
              Text(’Let\’s get started’, style: TextStyle(color: Colors.grey, fontFamily: ’Poppins’)),
              Container(
                margin: EdgeInsets.only(top: 13),
                child: TextField(
                  controller: _nameController,
                  cursorColor: Color(0xff44c662),
                  style: TextStyle(fontFamily: ’Poppins’, fontWeight: FontWeight.w500),
                  decoration: InputDecoration(
                      labelText: "Full Name",
                      hintStyle: TextStyle(fontFamily: ’Poppins’, color: Color(0xff444444)),
                      focusedBorder: OutlineInputBorder( borderRadius: BorderRadius.all(Radius.circular(6)),borderSide: BorderSide(color: Color(0xff44c662),)),
                      contentPadding: EdgeInsets.symmetric( horizontal: 20, vertical: 10),
                      border: OutlineInputBorder(gapPadding: 0, borderRadius: BorderRadius.all(Radius.circular(6)))),
                ),
              ),
              Container(
                margin: EdgeInsets.only(top: 13),
                child: TextField(
```

```
                  controller: _emailController,
                  cursorColor: Color(0xff44c662),
                  style: TextStyle(fontFamily: 'Poppins',
fontWeight: FontWeight.w500),
                  decoration: InputDecoration(
                      labelText: "Email",
                      hintStyle: TextStyle(fontFamily: 'Poppins',
color: Color(0xff444444)),
                      focusedBorder: OutlineInputBorder(
borderRadius: BorderRadius.all(Radius.circular(6)),borderSide:
BorderSide(color: Color(0xff44c662),)),
                      contentPadding: EdgeInsets.symmetric(
horizontal: 20, vertical: 10),
                      border: OutlineInputBorder(gapPadding: 0,
borderRadius: BorderRadius.all(Radius.circular(6)))),
                ),
              ),
              Container(
                margin: EdgeInsets.only(top: 13),
                child: TextField(
                  controller: _passwordController,
                  cursorColor: Color(0xff44c662),
                  style: TextStyle(fontFamily: 'Poppins',
fontWeight: FontWeight.w500),
                obscureText: pass,
                  decoration: InputDecoration(
                      suffixIcon: IconButton(
                        onPressed: () {
                          setState(() {
                            pass = !pass;
                          });
                        },
                        icon: Icon(
                            pass ? Icons.visibility_off : Icons.
visibility)),
```

```
                        labelText: "Password",
                        hintStyle: TextStyle(fontFamily: 'Poppins',
color: Color(0xff444444)),
                        focusedBorder: OutlineInputBorder(
borderRadius: BorderRadius.all(Radius.circular(6)),borderSide:
BorderSide(color: Color(0xff44c662),)),
                        contentPadding: EdgeInsets.symmetric(
horizontal: 20, vertical: 10),
                        border: OutlineInputBorder(gapPadding: 0,
borderRadius: BorderRadius.all(Radius.circular(6)))),
                  ),
                ),
                                Row(
                    children: <Widget>[
                      const Text('Already have an account?'),
                      TextButton(
                        child: const Text(
                          'Sign in',
                          style: TextStyle(fontSize: 16),
                        ),
                        onPressed: () {
                          Navigator.push(
                              context,
                              MaterialPageRoute(
                                  builder: (context) =>
                                      Auth_Screen(database)));
                        },
                      )
                    ],
                    mainAxisAlignment: MainAxisAlignment.center,
                  )
              ],
            ),
            Positioned(
              bottom: 15,
```

```
            right: -15,
            child: FlatButton(
              onPressed: () async {
                await SignUp();
                  //Navigator.pushReplacement(context, PageTransition(type: PageTransitionType.rightToLeft, child: Dashboard()));
              },
              color: Color(0xff44c662),
              padding: EdgeInsets.all(13),
             shape: CircleBorder(),
              child: Icon(Icons.arrow_forward, color: Colors.white),
            ),
          )
        ],
      ),
      height: 360,

      width: double.infinity,
      decoration: BoxDecoration(
      color: Colors.white,
      boxShadow: [
        BoxShadow(
            color: Color.fromRGBO(0, 0, 0, .1),
            blurRadius: 10,
            spreadRadius: 5,
            offset: Offset(0, 1))
      ],
      borderRadius: BorderRadiusDirectional.only(
          bottomEnd: Radius.circular(20), bottomStart: Radius.circular(20))),
    ),
      ],
    )
```

```
    );
  }
}

showAlertDialog1(BuildContext context, Database databasse) {
  // set up the button
  Widget okButton = ElevatedButton(
    child: Text("OK"),
    onPressed: () {
      Navigator.push(context,
          MaterialPageRoute(builder: (context) => Auth_Screen(
   databasse)));
    },
  );
  // set up the AlertDialog
  AlertDialog alert = AlertDialog(
    title: Text("Success"),
    content: Text("New User Created"),
    actions: [
      okButton,
    ],
  );
  // show the dialog
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return alert;
    },
  );
}

showAlertDialog2(BuildContext context) {
  // set up the button
  Widget okButton = ElevatedButton(
    child: Text("OK"),
```

```
    onPressed: () {
      Navigator.pop(context);
    },
  );
  // set up the AlertDialog
  AlertDialog alert = AlertDialog(
    title: Text("Sign up Failed"),
    content: Text("User already exists"),
    actions: [
      okButton,
    ],
  );
  // show the dialog
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return alert;
    },
  );
}
```

Item Screen

```
import 'package:app/main.dart';
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:app/data_models/items.dart';
import 'package:sqflite/sqflite.dart';
import 'package:fluttertoast/fluttertoast.dart';

class ItemScreen extends StatefulWidget {
  late ItemData item;
  Database datab;
  ItemScreen(this.item, this.datab, {Key? key}) : super(key: key);

  @override
```

```
  State<ItemScreen> createState() => _ItemScreenState(item, datab);
}

class _ItemScreenState extends State<ItemScreen> {
  ItemData item;
  Database datab;
  Future<void> _signOut() async {
    await FirebaseAuth.instance.signOut();
  }

  Future<void> insertdata(ItemData item, int quantity) async {
    item.quantity = quantity;
    final db = await datab;
    await db.insert('shoppingcart', item.toMap(),
        conflictAlgorithm: ConflictAlgorithm.replace);
  }

  _ItemScreenState(this.item, this.datab);
  int _quantity = 1;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        backgroundColor: Color(0xffF4F7FA),
        appBar: AppBar(
          elevation: 0,
          backgroundColor: Color(0xffF4F7FA),
          centerTitle: true,
          leading: BackButton(
            color: Colors.black54,
          ),
          actions: [
            IconButton(
              color: Colors.black54,
              onPressed: () {
                _signOut();
```

```
              Navigator.push(context,
                  MaterialPageRoute(builder: (context) => MyHomePage(datab)));
            },
            icon: Icon(Icons.logout),
          ),
        ],
        title: Text(item.name,
            style: TextStyle(
                color: Colors.black,
                fontSize: 18,
                fontWeight: FontWeight.w700,
                fontFamily: 'Poppins')),
      ),
      body: ListView(
        children: <Widget>[
          Container(
            margin: EdgeInsets.only(top: 20),
            child: Center(
              child: Stack(
                children: <Widget>[
                  Align(
                    alignment: Alignment.center,
                    child: Container(
                      margin: EdgeInsets.only(top: 100, bottom: 100),
                      padding: EdgeInsets.only(top: 100, bottom: 50),
                      width: MediaQuery.of(context).size.width * 0.85,
                      child: Column(
                        mainAxisAlignment: MainAxisAlignment.start,
                        crossAxisAlignment: CrossAxisAlignment.center,
                        children: <Widget>[
```

```
                        Text(item.name,
                            style: TextStyle(
                                color: Colors.black,
                                fontSize: 18,
                                fontWeight: FontWeight.w500,
                                fontFamily: 'Poppins')),
                        Text("Rs. " + item.price.toString(),
                            style: TextStyle(
                                color: Colors.black,
                                fontSize: 24,
                                fontWeight: FontWeight.w800,
                                fontFamily: 'Poppins')),
                        Container(
                          margin: EdgeInsets.only(top: 10, bottom
: 25),
                          child: Column(
                            children: <Widget>[
                              Container(
                                child: Text('Quantity',
                                    style: TextStyle(
                                        color: Colors.black,
                                        fontSize: 16,
                                        fontWeight: FontWeight.
w500,
                                        fontFamily: 'Poppins')),
                                margin: EdgeInsets.only(bottom:
15),
                              ),
                              Row(
                                mainAxisAlignment:
MainAxisAlignment.center,
                                crossAxisAlignment:
                                    CrossAxisAlignment.center,
                                children: <Widget>[
                                  Container(
```

```
                                    width: 55,
                                    height: 55,
                                    child: OutlinedButton(
                                      onPressed: () {
                                        setState(() {
                                          if (_quantity == 2)
return;
                                          _quantity += 1;
                                        });
                                      },
                                      child: Icon(Icons.add),
                                    ),
                                  ),
                                  Container(
                                    margin: EdgeInsets.only(
                                        left: 20, right: 20),
                                    child: Text(_quantity.
toString(),
                                        style: TextStyle(
                                            color: Colors.black,
                                            fontSize: 24,
                                            fontWeight:
FontWeight.w800,
                                            fontFamily: 'Poppins
')),
                                  ),
                                  Container(
                                    width: 55,
                                    height: 55,
                                    child: OutlinedButton(
                                      onPressed: () {
                                        setState(() {
                                          if (_quantity == 1)
return;
                                          _quantity -= 1;
```

```
                                        });
                                      },
                                      child: Icon(Icons.remove),
                                    ),
                                  ),
                                ],
                              ),
                              Padding(padding: EdgeInsets.all(10.0)),
                              Text(
                                item.description,
                                style: TextStyle(
                                    fontSize: 20.0,
                                    fontWeight: FontWeight.bold),
                              ),
                            ],
                          ),
                        ),
                        Container(
                          width: 180,
                          child: TextButton(
                            onPressed: () async {
                              await insertdata(item, _quantity);
                              var name = item.name;
                              Fluttertoast.showToast(
                                msg: "$name added to cart",
                                toastLength: Toast.LENGTH_SHORT,
                                gravity: ToastGravity.BOTTOM,
                                timeInSecForIosWeb: 1,
                                backgroundColor: Colors.white,
                                textColor: Colors.black,
                                fontSize: 16.0,
                              );
                            },
                            child: Text(
```

```
                               "Add to Cart",
                               style: TextStyle(
                                   color: Colors.black, fontSize: 20.0),
                             ),
                             style: ButtonStyle(
                                 backgroundColor:
                                     MaterialStateProperty.all<Color>(
                                         Color(0xff44c662)),
                                 textStyle: MaterialStateProperty.all(
                                     const TextStyle(color: Colors.black))),
                             // textColor: Colors.white,
                             // color: Color(0xff44c662),
                             // shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(4)),
                           ),
                         )
                       ],
                     ),
                     decoration: BoxDecoration(
                         color: Colors.white,
                         borderRadius: BorderRadius.circular(10),
                         boxShadow: [
                           BoxShadow(
                               blurRadius: 15,
                               spreadRadius: 5,
                               color: Color.fromRGBO(0, 0, 0, .05))
                         ]),
                   ),
                 ),
                 Align(
```

```
                      alignment: Alignment.center,
                      child: SizedBox(
                        width: 200,
                        height: 160,
                        child: Container(
                          height: 200,
                          width: 200,
                          child: Image(
                              image:
                                  AssetImage("assets/images/" + item.
   imageUrl)),
                        ),
                      ),
                    )
                  ],
                ),
              ),
            )
          ],
        ));
  }
}
```

Home Page Screen

```
import 'package:app/data_models/items.dart';
import 'package:app/screens/item_screen.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:sqflite/sqflite.dart';
import 'package:fluttertoast/fluttertoast.dart';

import '../main.dart';
import 'after_order_screen.dart';
```

```

class MenuScreen extends StatefulWidget {
  Database database;
  MenuScreen(this.database);

  @override
  _MenuScreenState createState() => _MenuScreenState(database);
}

class _MenuScreenState extends State<MenuScreen> {
  CollectionReference ref_items =
      FirebaseFirestore.instance.collection("items");
  FirebaseStorage storage = FirebaseStorage.instance;
  List<ItemData> cartlist = [];
  Database datab;
  int selectedindex = 0;
  bool flag = false;
  String dropdownValue = 'Table 1';
  _MenuScreenState(this.datab);
  Future<void> _signOut() async {
    await FirebaseAuth.instance.signOut();
  }

  Future<void> insertdata(ItemData item) async {
    final db = await datab;
    await db.insert('shoppingcart', item.toMap(),
        conflictAlgorithm: ConflictAlgorithm.replace);
  }

  getitems() async {
    QuerySnapshot query = await ref_items.get();
    List docs = query.docs;
    List<ItemData> lst =
        docs.map((doc) => ItemData.fromJson(doc.data(), 0)).toList();
    return lst;
```

```
}

Future itemslist() async {
  final db = await datab;

  final List<Map<String, dynamic>> maps = await db.query('shoppingcart');

  return List.generate(maps.length, (i) {
    return ItemData(maps[i]['quantity'],
        itemId: maps[i]['itemId'],
        name: maps[i]["name"],
        price: maps[i]["price"],
        imageUrl: maps[i]["imageUrl"],
        description: maps[i]['description'],
        category: maps[i]['category']);
  });
}

getstatus() async {
  try {
    var status = '';
    var docs = await FirebaseFirestore.instance
      .collection('orders')
      .where("email", isEqualTo: FirebaseAuth.instance.currentUser?.email)
      .get()
      .then((value) => value.docs.map((e) => status = e['status']).toList());
  print(docs);
  return Column(
    children: [
      Padding(padding: EdgeInsets.all(20.0)),
      Text(
        "Your order status is:",
```

```
        style: TextStyle(fontSize: 20.0),
      ),
      Text(
        docs[0],
        style: TextStyle(fontSize: 20.0),
      ),
    ],
  );
  }
  catch (error){
    return Column(children: [
      Padding(padding: EdgeInsets.all(20.0)),
      Text("You do not have any order",style: TextStyle(fontSize:
 20.0))
    ]);
  }
}

setdatadelivery(List lst, num amount, String tableno) async {
  var name = await FirebaseFirestore.instance
      .collection('users')
      .doc(FirebaseAuth.instance.currentUser?.email)
      .get()
      .then((doc) => doc.get('name'));
  var id = await FirebaseFirestore.instance
      .collection('metadata')
      .doc('data')
      .get()
      .then((doc) => doc.get('orderids'));
  id = id + 1;
  FirebaseFirestore.instance.collection("orders").doc(id.toString()
 ).set({
    'amount': amount,
    'name': name,
    'ID': id,
```

```
      "email": FirebaseAuth.instance.currentUser?.email,
      'items': lst,
      'deliver': true,
      'time': DateTime.now(),
      'status': 'preparing',
      'table': tableno
    });
    FirebaseFirestore.instance
        .collection('metadata')
        .doc('data')
        .update({'orderids': id});
}

setdatapreorder(List lst, num amount) async {
    var name = await FirebaseFirestore.instance
        .collection('users')
        .doc(FirebaseAuth.instance.currentUser?.email)
        .get()
        .then((doc) => doc.get('name'));
    var id = await FirebaseFirestore.instance
        .collection('metadata')
        .doc('data')
        .get()
        .then((doc) => doc.get('orderids'));
    id = id + 1;
    FirebaseFirestore.instance.collection("orders").doc(id.toString()
   ).set({
      'amount': amount,
      'name': name,
      'ID': id,
      "email": FirebaseAuth.instance.currentUser?.email,
      'items': lst,
      'deliver': false,
      'time': DateTime.now(),
      'status': 'preparing'
```

```
  });
  FirebaseFirestore.instance
      .collection('metadata')
      .doc('data')
      .update({'orderids': id});
}

Future<void> deleteItem(String id) async {
  final db = await datab;
  await db.delete(
    'shoppingcart',
    where: 'itemId = ?',
    whereArgs: [id],
  );
}

getwidgettree(int index) {
  List<Widget> widgets = <Widget>[
    Column(
      children: <Widget>[
        FutureBuilder(
            future: getitems(),
            builder: (BuildContext context, AsyncSnapshot snapshot) {
              if (snapshot.connectionState == ConnectionState.done) {
                if (snapshot.hasData) {
                  List<Widget> list = [];
                  for (var i = 0; i < snapshot.data.length; i++) {
                    var currentItem = snapshot.data[i];
                    var image = currentItem.imageUrl;
                    list.add(Column(
                      children: <Widget>[
                        Container(
                            width: 180,
```

```
                          height: 180,
                          child: ElevatedButton(
                              style: ButtonStyle(
                                elevation: MaterialStateProperty.all(12),
                                shape: MaterialStateProperty.all(
                                    RoundedRectangleBorder(
                                        borderRadius:
                                            BorderRadius.circular(5))),
                                backgroundColor:
                                    MaterialStateProperty.all<Color>(
                                        Color.fromARGB(255, 255, 255, 255)),
                              ),
                              onPressed: () {
                                Navigator.push(
                                    context,
                                    MaterialPageRoute(
                                        builder: (context) => ItemScreen(
                                            currentItem, datab)));
                              },
                              child: Hero(
                                  transitionOnUserGestures: true,
                                  tag: currentItem.name,
                                  child: Image.asset(
                                      "assets/images/" +
                                          currentItem.imageUrl,
                                      width: 200)))),
                      Padding(padding: EdgeInsets.fromLTRB(0, 10.0, 0, 0)),
                      Text(currentItem.name,
```

```
                        style: TextStyle(
                            color: Colors.black,
                            fontSize: 19,
                            fontWeight: FontWeight.w800,
                            fontFamily: 'Poppins')),
                    Text("Rs. " + currentItem.price.toString(),
                        style: TextStyle(
                            color: Colors.black,
                            fontSize: 19,
                            fontWeight: FontWeight.w800,
                            fontFamily: 'Poppins')),
                  ],
                ));
              }
              return Expanded(
                child: GridView.count(
                  physics: const ScrollPhysics(),
                  childAspectRatio: 1 / 3,
                  padding: const EdgeInsets.all(20),
                  crossAxisSpacing: 10,
                  mainAxisSpacing: 10,
                  crossAxisCount: 2,
                  children: list,
                ),
              );
            }
          }
          return Center(child: Text("Items Loading"));
        })
    ],
  ),
  Container(
      padding: EdgeInsets.fromLTRB(10.0, 10.0, 10.0, 10.0),
      child: ListView(
        children: [
```

```
FutureBuilder(
  future: itemslist(),
  builder: (BuildContext context, AsyncSnapshot snapshot) {
    if (snapshot.connectionState == ConnectionState.done) {
      if (snapshot.hasData) {
        List<Widget> cartlist = [];
        for (var i = 0; i < snapshot.data.length; i++) {
          var name = snapshot.data[i].name;
          var qty = snapshot.data[i].quantity;
          var price = (snapshot.data[i].price * qty).toString();
          var image = snapshot.data[i].imageUrl;
          cartlist.add(Column(
            children: [
              Container(
                  color: Colors.grey[500],
                  height: 100.0,
                  width: 400.0,
                  child: Container(
                    decoration: BoxDecoration(
                      color: Colors.white,
                      //borderRadius: BorderRadius.all(Radius.circular(10.0)),
                    ),
                    child: Row(
                      mainAxisAlignment: MainAxisAlignment.start,
                      children: [
                        Padding(padding: EdgeInsets.all(10.0)),
                        SizedBox(
                          height: 100,
```

```
  width: 100,
  child: Image(
      image: AssetImage(
          "assets/images/" + image)),
),
const Padding(
    padding: EdgeInsets.all(5.0)),
Column(
  mainAxisAlignment:
      MainAxisAlignment.center,
  children: [
    Text(name + " x" + qty.toString()),
    const Padding(
        padding: EdgeInsets.all(5.0)),
    Text("Rs." + price),
  ],
),
const Padding(
    padding: EdgeInsets.all(10.0)),
Expanded(
  child: Align(
    alignment: Alignment.centerRight,
    child: IconButton(
        onPressed: () {
          setState(() {
            deleteItem(
                snapshot.data[i].itemId);
          });
```

```
                                           },
                                           icon: const Icon(
                                               Icons.remove_circle)),
                                     ),
                                   ),
                                 ],
                               ),
                             )),
                         Padding(padding: EdgeInsets.all(5.0))
                       ],
                     ));
                   }
                   cartlist.add(Row(
                     crossAxisAlignment: CrossAxisAlignment.center,
                     mainAxisAlignment: MainAxisAlignment.center,
                     children: [
                       ElevatedButton(
                           onPressed: snapshot.data.length < 1
                               ? null
                               : () {
                                   List lst = [];
                                   num amount = 0;
                                   for (var i = 0;
                                       i < snapshot.data.length;
                                       i++) {
                                     lst.add({
                                       "itemId": snapshot.data[i].itemId
                                           .toString(),
                                       "name": snapshot.data[i].name,
                                       "qty": snapshot.data[i].quantity
```

```
                                    });
                                    amount = amount +
                                        (snapshot.data[i].price *
                                            snapshot.data[i].
quantity);
                                  }
                                  setdatapreorder(lst, amount);
                                  for (var i = 0;
                                      i < snapshot.data.length;
                                      i++) {
                                    deleteItem(snapshot.data[i].
itemId);
                                  }
                                  Navigator.push(
                                      context,
                                      MaterialPageRoute(
                                          builder: (context) =>
                                              Preorder_Screen()))
;
                                },
                          child: Text("Pre Order")),
                      Padding(padding: EdgeInsets.all(10.0)),
                      ElevatedButton(
                        child: Text("Deliver"),
                        onPressed: snapshot.data.length < 1
                            ? null
                            : () {
                                showDialog(
                                    context: context,
                                    builder: (BuildContext
context) {
                                      return AlertDialog(
                                        scrollable: true,
                                        title: Text('Table
Selector'),
```

```
                                        content: Padding(
                                          padding:
                                              const EdgeInsets.
   all(8.0),
                                          child: Form(
                                            child: Column(
                                              children: <Widget>[
                                                Text(
                                                    "Please
   select your table:"),
                                                DropdownButton(
                                                  value:
   dropdownValue,
                                                  items: <String
   >[
                                                    'Table 1',
                                                    'Table 2',
                                                    'Table 3',
                                                    'Table 4',
                                                    'Table 5',
                                                    'Table 6',
                                                    'Table 7',
                                                    'Table 8',
                                                    'Table 9',
                                                    'Table 10',
                                                    'Table 11',
                                                    'Table 12',
                                                    'Table 13',
                                                    'Table 14',
                                                    'Table 15',
                                                    'Table 16',
                                                    'Table 17'
                                                  ].map<

   DropdownMenuItem<
```

"),

```
String>>(
                                                         (String
value) {
                                                       return
DropdownMenuItem<
                                                           String>(
                                                         value:
value,
                                                         child: Text
(value),
                                                       );
                                                     }).toList(),
                                                     onChanged:
                                                         (String?
newValue) {
                                                       setState(() {
dropdownValue =
newValue!;
                                                       });
                                                     },
                                                   )
                                                 ],
                                               ),
                                             ),
                                           ),
                                           actions: [
                                             ElevatedButton(
                                                 child: Text("Submit
"),
                                                 onPressed: () {
                                                   List lst = [];
                                                   num amount = 0;
```

```
for (var i = 0;
    i <
        snapshot
            .data
.length;
    i++) {
  lst.add({
    "itemId":
snapshot
        .data[i].
itemId
        .toString
(),
    "name":
snapshot
        .data[i].
name,
    "qty":
snapshot
        .data[i].
quantity
  });
  amount = amount
 +
      (snapshot.
data[i]
          .
price *

snapshot.data[i]
          .
quantity);
}
setdatadelivery(
lst, amount,
```

```
                                                                          dropdownValue
);
                                                                      for (var i = 0;
                                                                          i <
                                                                              snapshot
                                                                                  .data
.length;
                                                                          i++) {
                                                                        deleteItem(
snapshot
                                                                                .data[i].
itemId);
                                                                      }
                                                                      Navigator.push(
                                                                          context,

MaterialPageRoute(
                                                                              builder:
(context) =>

Deliver_Screen()));
                                                                    })
                                                              ],
                                                            );
                                                          });
                                                    },
                                            ),
                                          ],
                                        ));
                                        return Column(
                                          children: cartlist,
                                        );
                                      }
                                      return Center(child: Text("No items in cart"));
                                    }
```

```
         return Center(child: Text("No items in cart"));
       },
      ),
     ],
    )),
  Container(
    child: Center(
      child: FutureBuilder(
        future: getstatus(),
        builder: (BuildContext context, AsyncSnapshot snapshot) {
         if (snapshot.connectionState == ConnectionState.done) {
          if (snapshot.hasData) {
           return Column(
            children: [
             snapshot.data
            ],
           );
          }
         }
         return Text("loading");
        }))),
 ];
 return widgets[index];
}

void _onItemTapped(int index) {
 setState(() {
  selectedindex = index;
 });
}

@override
Widget build(BuildContext context) {
```

```
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          "Menu",
          textAlign: TextAlign.center,
          style: TextStyle(
              fontFamily: 'Pacifico',
              fontSize: 21,
              letterSpacing: 2,
              color: Colors.white),
        ),
        automaticallyImplyLeading: false,
        backgroundColor: Color(0xff44c662),
        elevation: 0,
        centerTitle: true,
        actions: [
          // IconButton(
          //     onPressed: () {
          //       Navigator.push(
          //           context,
          //           MaterialPageRoute(
          //               builder: (context) => Cart_Screen(datab)))
;
          //     },
          //     icon: Icon(Icons.shopping_cart)),
          IconButton(
            onPressed: () {
              _signOut();
              Navigator.push(context,
                  MaterialPageRoute(builder: (context) => MyHomePage(
datab)));
            },
            icon: Icon(Icons.logout),
          ),
        ],
```

```
      ),
      body: getwidgettree(selectedindex),
      bottomNavigationBar: BottomNavigationBar(
        items: [
          BottomNavigationBarItem(
              icon: Icon(Icons.store_mall_directory), label: "Menu"),
          BottomNavigationBarItem(
              icon: Icon(Icons.shopping_cart), label: "Cart"),
          BottomNavigationBarItem(
              icon: Icon(Icons.track_changes), label: "Track my Order
    "),
        ],
        currentIndex: selectedindex,
        selectedItemColor: Colors.black,
        onTap: _onItemTapped,
      ),
    );
  }
}
```

APPENDIX C

HARDWARE CONFIGURATIONS

C.1 Weight Instrument

C.1.1 Load Cell Specifications

Apart from the specifications discussed in above section, this table shows other specifications obtained from the datasheet.

S. #	Specification	Value
1	Hysteresis & Non-Linearity	0.05% FS
2	Input & Output Impedance	1000 Ohms
3	Excitation Voltage	$\geq$ 5V
4	Ultimate Overload	150% Capacity
5	Zero Balance Error	$\pm$1.5% FS
6	Rated Voltage	1 mV/V

Figure C.1: Other specifications of load cell

The following figures shows the internal schematic and the mount of the load cell.

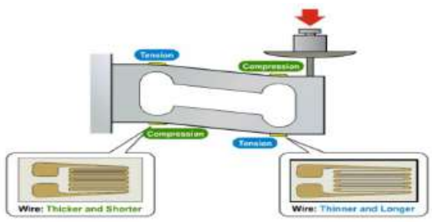


Figure C.2: Change to strain gage when force is applied

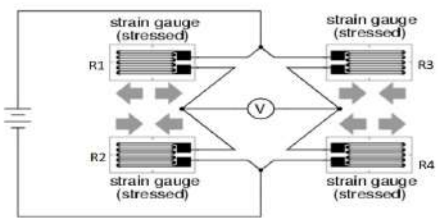


Figure C.3: Load Cell - Schematic

C.1.2 Instrumentation Amplifier Schematic

This schematic of instrumentation amplifier was taken from the amplifier document provided at canvas.

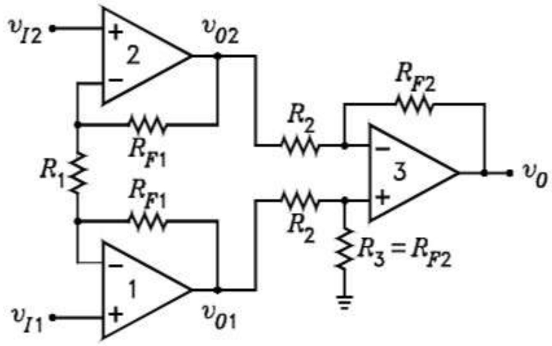


Figure C.4: Schematic For Instrumentation Amplifier

C.1.3 Load Cell Characteristic Table

The following table shows the data points gathered to plot the response of load cell on different known weights.

S.#	Weight (g)	Output Voltage (mV)
1	0	0.04
2	200	0.2
3	400	0.44
4	600	0.64
5	800	0.78
6	1000	1.01
7	1250	1.24
8	1500	1.45
9	2000	2.06
10	2500	2.56
11	3750	3.82
12	5000	4.96

C.1.4 CMRR of Signal Conditioning

Since we already know the transfer function of instrumentation amplifier, we can use that to provide it a common mode signal and a differential signal to get the value for the gain it provides to the signal.

$$A_{cm} ::: V_o = \frac{R_{F_2}}{R_2}\left(1 + 2 = \frac{R_{F_1}}{R_1}\right)(v_{cm} - v_{cm}) = 0 \tag{C.1}$$

Since we assume that the common mode signal that enters the circuit is same from both terminals, theoretically, the gain to the common mode signal is 0.

$$A_d ::: V_o = \frac{R_{F_2}}{R_2}\left(1 + 2 = \frac{R_{F_1}}{R_1}\right)(\frac{v_d}{2} - (\frac{v_d}{2})) \tag{C.2}$$

$$A_d = \frac{V_o}{v_d} = \frac{R_{F_2}}{R_2}\left(1 + 2 = \frac{R_{F_1}}{R_1}\right) = 800 \tag{C.3}$$

So theoretically, CMRR of instrumentation amplifier is infinite as A_{cm} is 0. Thus, we assume that the CMRR of instrumentation amplifier is very high.

C.1.5 Arduino Code

```
#include <LiquidCrystal.h>
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
float weight;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
}

void loop() {
  int sensorValue = analogRead(A0);
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
```

```
  Serial.println(voltage);
  weight = (voltage-0.36)/3.6 * 5;
  lcd.print("Weight =  " + String(weight) + "Kg");
  lcd.setCursor(0, 0);
}
```

APPENDIX D

MATLAB SIMULATION FOR KINEMATIC

D.1 Forward Kinematics Code

```
c = 1;
r = 1;
t = -pi:0.1:pi;
syms f(x)
f(x) = sin(x);

omega1 = f(x); % Wheel 1
omega2 = omega1; % Wheel 2

syms f(x)
f(x) = cos(x);

omega3 = f(x); % Wheel 3
omega4 = omega3; % Wheel 4
%Mobile FoR
%%
v_x = r/4 *(omega1+omega2+omega3+omega4);
v_y = 0;
W = r/4 * (-omega1/c - omega2/c + omega3/c + omega4/c);
phi = diff(W);

%Global FoR
%%
v_xg = v_x.*cos(phi);
v_yg = v_x.*sin(phi);

d_xg = diff(v_xg);
d_yg = diff(v_yg);
```

```
ezplot(d_xg,d_yg)
title('Plot of Displacement (Global) from -\pi to \pi')
xlabel('Time')
ylabel('Velocity')
```

D.2 Error for Discrete Models Code

```
% Sample code for Rectangular Integration based Kinematic model
ti = 0; % Initial time
tf = 100; % Final time

tf = td(end);

Error_Rect = zeros(1,7);
Error_Trap = zeros(1,7);
Error_Exact = zeros(1,7);
Error_Geo1 = zeros(1,7);
Error_Geo2 = zeros(1,7);
Error_Taylor = zeros(1,7);
Error_Runge = zeros(1,7);


%Continuous
dt = 0.01; % Time increment for continuous scenario
tc = ti:dt:tf; % Continuous time (CT) frame
wR = 0.3*cos(0.002*tc); % Right wheel angular
cont_len = length(tc);
wL = sin(0.002*tc); % Left wheel angular speed
sl = length(tc); % Number of steps in CT simulation
xc = zeros(1,sl); % Vector to store CT x(t)
yc = zeros(1,sl); % Vector to store CT y(t)
phc = zeros(1,sl); %% Vector to store CT \phi(t)
for ii=1:sl-1 % <= one less simulation for the reason mentioned above
vR=r*wR(ii);
vL=r*wL(ii);
v=(vR+vL)/2;
```

```
w=(vR-vL)/L;
dph=dt*w;
xc(ii+1)=xc(ii)+v*dt*cos(phc(ii));
yc(ii+1)=yc(ii)+v*dt*sin(phc(ii));
phc(ii+1)=phc(ii)+dph;
end
size = 50;
for looper=1:size
T = 1/looper;
td = ti:T:tf; % Discrete time defined
sl = length(td);
xc_new = zeros(1,sl);
yc_new = zeros(1,sl);
td = ti:T:tf; % Discrete time defined
% Due to discrete steps, t_d(end) is not necessarily equal to tf.
% That is why we need to recompute tf by assigning t_d(end) to it.
sl = length(td); % Number of samples in discrete time frame
wR = 0.3*cos(0.002*td); % Right wheel angular speed
wL = sin(0.002*td); % Left wheel angular speed
r = 5; % Radius of wheels [cm]
L = 10; % Width of robot platform [cm]
xd = zeros(1,sl); % Vector to store global x_n
yd = zeros(1,sl); % Vector to store global y_n
phd = zeros(1,sl); % Vector to store global \phi_n



%Rectangular
for ii=1:sl-1
% note that we need to run this loop for one less iteration because
% the pose quantities have time index (ii+1)
vR=r*wR(ii);
vL=r*wL(ii);
v=(vR+vL)/2;
w=(vR-vL)/L;
```

```
dph=T*w;
xd(ii+1)=xd(ii)+v*T*cos(phd(ii));
yd(ii+1)=yd(ii)+v*T*sin(phd(ii));
phd(ii+1)=phd(ii)+dph;
end







%Trapezoidal

tt = ti:T:tf; % Continuous time (CT) frame
wR = 0.3*cos(0.002*tt); % Right wheel angular speed
wL = sin(0.002*tt); % Left wheel angular speed
sl = length(tt); % Number of steps in CT simulation
x_T = zeros(1,sl); % Vector to store CT x(t)
y_T = zeros(1,sl); % Vector to store CT y(t)
phc = zeros(1,sl); %% Vector to store CT \phi(t

for ii=1:sl-1
% note that we need to run this loop for one less iteration because
% the pose quantities have time index (ii+1)
vR=r*wR(ii);
vL=r*wL(ii);
v=(vR+vL)/2;
w=(vR-vL)/L;
dph=T*w;
x_T(ii+1)=x_T(ii)+v*0.5*T*(cos(phc(ii)) + cos(phc(ii) + dph));
y_T(ii+1)=y_T(ii)+v*0.5*T*(sin(phc(ii)) + sin(phc(ii) + dph));
phc(ii+1)=phc(ii)+dph;
end

```

```
%Exact Integration
dt = 0.01; % Time increment for continuous scenario
te = ti:T:tf; % Continuous time (CT) frame
wR = 0.3*cos(0.002*te); % Right wheel angular speed
wL = sin(0.002*te); % Left wheel angular speed
sl = length(te); % Number of steps in CT simulation
x_EI = zeros(1,sl); % Vector to store CT x(t)
y_EI = zeros(1,sl); % Vector to store CT y(t)
phc = zeros(1,sl); % % Vector to store CT \phi(t

for ii=1:sl-1
% note that we need to run this loop for one less iteration because
% the pose quantities have time index (ii+1)
vR=r*wR(ii);
vL=r*wL(ii);
v=(vR+vL)/2;
w=(vR-vL)/L;
dph=T*w;
x_EI(ii+1)=x_EI(ii)+v/w*(sin(phc(ii) + dph) - sin(phc(ii)));
y_EI(ii+1)=y_EI(ii)-v/w*(cos(phc(ii) + dph) - cos(phc(ii)));
phc(ii+1)=phc(ii)+dph;
end



%Geometry Based 1
dt = 0.01; % Time increment for continuous scenario
tg1 = ti:T:tf; % Continuous time (CT) frame
wR = 0.3*cos(0.002*te); % Right wheel angular speed
wL = sin(0.002*te); % Left wheel angular speed
sl = length(te); % Number of steps in CT simulation
x_G1 = zeros(1,sl); % Vector to store CT x(t)
y_G1 = zeros(1,sl); % Vector to store CT y(t)
```

```
phc = zeros(1,sl); %% Vector to store CT \phi(t

for ii=1:sl-1
% note that we need to run this loop for one less iteration because
% the pose quantities have time index (ii+1)
vR=r*wR(ii);
vL=r*wL(ii);
v=(vR+vL)/2;
w=(vR-vL)/L;
dph=T*w;
x_G1(ii+1)=x_G1(ii)+v*T*cos(phc(ii) + dph / 2);
y_G1(ii+1)=y_G1(ii)+v*T*sin(phc(ii) + dph / 2);
phc(ii+1)=phc(ii)+dph;
end




%Geometry Based 2
dt = 0.01; % Time increment for continuous scenario
tg1 = ti:T:tf; % Continuous time (CT) frame
wR = 0.3*cos(0.002*te); % Right wheel angular speed
wL = sin(0.002*te); % Left wheel angular speed
sl = length(te); % Number of steps in CT simulation
x_G2 = zeros(1,sl); % Vector to store CT x(t)
y_G2 = zeros(1,sl); % Vector to store CT y(t)
phc = zeros(1,sl); %% Vector to store CT \phi(t

for ii=1:sl-1
% note that we need to run this loop for one less iteration because
% the pose quantities have time index (ii+1)
vR=r*wR(ii);
vL=r*wL(ii);
v=(vR+vL)/2;
w=(vR-vL)/L;
```

```
dph=T*w;
x_G2(ii+1)=x_G2(ii)+2*(v/w)*sin(dph/2)*cos(phc(ii) + dph / 2);
y_G2(ii+1)=y_G2(ii)+2*(v/w)*sin(dph/2)*sin(phc(ii) + dph / 2);
phc(ii+1)=phc(ii)+dph;
end


%Taylor Series Method

dt = 0.01; % Time increment for continuous scenario
tg1 = ti:T:tf; % Continuous time (CT) frame
wR = 0.3*cos(0.002*te); % Right wheel angular speed
wL = sin(0.002*te); % Left wheel angular speed
sl = length(te); % Number of steps in CT simulation
x_TS = zeros(1,sl); % Vector to store CT x(t)
y_TS = zeros(1,sl); % Vector to store CT y(t)
phc = zeros(1,sl); %% Vector to store CT \phi(t

for ii=1:sl-1
% note that we need to run this loop for one less iteration because
% the pose quantities have time index (ii+1)
vR=r*wR(ii);
vL=r*wL(ii);
v=(vR+vL)/2;
w=(vR-vL)/L;
dph=T*w;
x_TS(ii+1)=x_TS(ii)+(v/w)*(dph*(cos(phc(ii)) - (dph/2)*sin(phc(ii))))
    ;
y_TS(ii+1)=y_TS(ii)-(v/w)*(dph*(-sin(phc(ii)) - (dph/2)*cos(phc(ii)))
    );
phc(ii+1)=phc(ii)+dph;
end
```

```
%RK4 Method

 % Time increment for continuous scenario
ta = ti:T:tf; % Continuous time (CT) frame
wR = 0.3*cos(0.002*ta); % Right wheel angular speed
wL = sin(0.002*ta); % Left wheel angular speed
sl = length(ta); % Number of steps in CT simulation
x_rk= zeros(1,sl); % Vector to store CT x(t)
y_rk = zeros(1,sl); % Vector to store CT y(t)
phc = zeros(1,sl); % % Vector to store CT \phi(t)

for ii=1:sl-1
% note that we need to run this loop for one less iteration because
% the pose quantities have time index (ii+1)
vR=r*wR(ii);
vL=r*wL(ii);
v=(vR+vL)/2;
w=(vR-vL)/L;
dph=T*w;
x1 = T*v*cos(phc(ii));
x2 = T*v*cos(phc(ii)+dph/2);
x3 = T*v*cos(phc(ii)+x2/2);
x4 = T*v*cos(phc(ii)+x3);
x_rk(ii+1)=x_rk(ii)+(1/6)*(x1+2*x2+2*x3+x4);


y1 = T*v*sin(phc(ii));
y2 = T*v*sin(phc(ii)+dph/2);
y3 = T*v*sin(phc(ii)+y2/2);
y4 = T*v*sin(phc(ii)+y3);
y_rk(ii+1)=y_rk(ii)+(1/6)*(y1+2*y2+2*y3+y4);
```

```
phc(ii+1)=phc(ii)+dph;
end





% xc-yc Continuous ----
% xd-yd Rectangular ----
% x_T-y_T Trapezoidal ----
% x_EI-y_EI Exact Integration --
% x_G1-y_G1 Geometry Based I----
% x_G2-y_G2 Geometry Based II----
% x_TS-y_TS Taylor Series----
% x_rk-y_rk Runge Kutta Method----





for i=1:sl-1
    xc_new(i) = xc(round(i*T/dt));
    yc_new(i) = yc(round(i*T/dt));
end

xc_new(sl) = xc(cont_len);
yc_new(sl) = yc(cont_len);


%Rectangular Error
E_rect_x = xc_new - xd;
E_rect_y = yc_new - yd;
Error_Rect(looper) = sum(E_rect_x)^2 + sum(E_rect_y)^2;
```

```
%Trapezoidal Error
E_Trap_x = xc_new - x_T;
E_Trap_y = yc_new - y_T;
Error_Trap(looper) = sum(E_Trap_x)^2 + sum(E_Trap_y)^2;

%Exact Integration Error
E_Int_x = xc_new - x_EI;
E_Int_y = yc_new - y_EI;
Error_Exact(looper) = sum(E_Int_x)^2 + sum(E_Int_y)^2;


%Geometry Based 1 Error
E_G1_x = xc_new - x_G1;
E_G1_y = yc_new - y_G1;
Error_Geo1(looper) = sum(E_G1_x)^2 + sum(E_G1_y)^2;


%Geometry Based 2 Error
E_G2_x = xc_new - x_G2;
E_G2_y = yc_new - y_G2;
Error_Geo2(looper) = sum(E_G2_x)^2 + sum(E_G2_y)^2;


%Taylor Series Error
E_TS_x = xc_new - x_TS;
E_TS_y = yc_new - y_TS;
Error_Taylor(looper) = sum(E_TS_x)^2 + sum(E_TS_y)^2;



%RK4 Error
E_rk_x = xc_new - x_rk;
E_rk_y = yc_new - y_rk;
Error_Runge(looper) = sum(E_rk_x)^2 + sum(E_rk_y)^2;

```

```
end
Sampling_Freq = 1:size;
plot(Sampling_Freq, Error_Rect,'-s','linewidth',0.5)
hold on;
plot(Sampling_Freq, Error_Trap,'-*','linewidth',0.5)
plot(Sampling_Freq, Error_Exact,'-v','linewidth',0.5)
plot(Sampling_Freq, Error_Geo1,'-d','linewidth',0.5)
plot(Sampling_Freq, Error_Geo2,'-x','linewidth',0.5)
plot(Sampling_Freq, Error_Taylor,'-^','linewidth',0.5)
plot(Sampling_Freq, Error_Runge ,'->','linewidth',0.5)


xlabel('Sampling Frequency','fontsize',26)
ylabel('Squared Error','fontsize',26)
title('Error Plots for each Model','fontsize',26)

legend('Rectangular','Trapezoidal','Exact-Integration','Geometry
    Model - I','Geometry Model - II','Taylor-Series Method', 'RK-4
    Method')
```

D.3 Inverse Kinematic Code

```
L = 1;
r = 1;
T = 0.01;
lim = 500;
%n = -lim:T:lim;
n = -lim:1:lim;
x_n = cos(cos(n.*T)./2 + sin(n.*T)./2).*(cos(n.*T)./2 - sin(n.*T)./2)
     - sin(cos(n.*T)./2 + sin(n*T)./2).*(cos(n.*T)./2 - sin(n.*T)./2)
    .*(cos(n.*T)./2 + sin(n.*T)./2);
y_n = -sin(cos(n.*T)./2 + sin(n.*T)./2).*(cos(n.*T)./2 - sin(n.*T)
    ./2) - cos(cos(n.*T)./2 + sin(n*T)./2).*(cos(n.*T)./2 - sin(n.*T)
    ./2).*(cos(n.*T)./2 + sin(n.*T)./2);

% y_np1 = [y_n(2:end) 0];
```

```
% x_np1 = [x_n(2:end) 0];

y_nm1 = [0 y_n(1:end-1)];
x_nm1 = [0 x_n(1:end-1)];



phi_n = atan2(y_n - y_nm1, x_n - x_nm1);

x_m = 0.5*(x_nm1+x_n);
y_m = 0.5*(y_nm1+y_n);
% temp = (cos(phi_n).*(y_n-y_nm1)-sin(phi_n).*(x_n-x_nm1));
% temp_2 = temp./abs(temp);

%temp_2(isnan(temp_2))=1;
%mew = 0.5*(sin(phi_n).*(y_n-y_nm1)+cos(phi_n).*(x_n-x_nm1))./(cos(
    phi_n).*(y_n-y_nm1)-sin(phi_n).*(x_n-x_nm1));
%mew = (sin(phi_n).*(y_m-y_nm1)-cos(phi_n).*(x_nm1-x_m)+1)./(cos(
    phi_n).*(y_n-y_nm1)-sin(phi_n).*(x_n-x_nm1) + T);
mew = 0.5*(sin(phi_n).*(y_n-y_nm1)+cos(phi_n).*(x_n-x_nm1))./(cos(
    phi_n).*(y_n-y_nm1)+sin(phi_n).*(x_n-x_nm1));
%mew = 0.5*(sin(phi_n).*(y_n-y_nm1)+cos(phi_n).*(x_n-x_nm1));
tempo1 = cos(phi_n).*(y_n-y_nm1);
tempo2 = sin(phi_n).*(x_n-x_nm1);
% plot(tempo1-tempo2);
% figure;
% plot(tempo2);
%mew = 0.5*(sin(phi_n).*(y_n-y_nm1)+cos(phi_n).*(x_n-x_nm1))./(cos(
    phi_n).*(y_n-y_nm1)+sin(phi_n).*(x_n-x_nm1));
x_star = x_m + mew.*(y_nm1-y_n);
y_star = y_m + mew.*(x_n-x_nm1);


theta_1 = atan2(y_star - y_nm1,x_nm1 - x_star);
theta_2 = atan2(y_star - y_n,x_n - x_star);
```

```
deltaphi = theta_1 - theta_2;
deltaphi2 = wrapToPi(deltaphi);

omega = deltaphi2./T;
radii = sqrt((x_nm1-x_star).^2+(y_nm1-y_star).^2);
v_n = radii .* abs(omega);
plot(n(2:end).*T,v_n(2:end));
xlabel('Time (Seconds)','fontsize',26)
ylabel('Velocity (m/s)','fontsize',26)
title('v(t)','fontsize',26)
figure;
plot(n(2:end).*T,omega(2:end));

xlabel('Time (Seconds)','fontsize',26)
ylabel('Angular Velocity (rad/s)','fontsize',26)
title('\omega(t)','fontsize',26)


%%
C = (2.*radii + L) ./ (2.*radii - L);
v_right = (C.*omega*L) ./ (1+C);
v_left = L*omega - v_right;
w_right = radii .* v_right;
w_left = radii .* v_left;
figure;
plot(n(2:end).*T,w_right(2:end));
hold on;
plot(n(2:end).*T,w_left(2:end));
xlabel('Time (Seconds)','fontsize',26)
ylabel('Angular Velocity (rad/s)','fontsize',26)
title('\omega_R and \omega_L','fontsize',26)
legend('\omega_R','\omega_L')
hold off;
```

```
%plot(x_n,y_n)
%%
figure;
for x=2:2*lim
    plot(x_n(1:x),y_n(1:x));
    title('Trajectory of the Mobile Robot','fontsize',26);
    xlabel('X - Displacement','fontsize',26);
  ylabel('Y - Displacement','fontsize',26);
    pause_time = T; %This is time = dist/speed but is taking too long
     to process so used some constant value for time
    pause(pause_time*0.001);
end
```

REFERENCES

[1] marketsandmarkets, *Delivery Robots Market with COVID-19 Impact by Load Carrying Capacity , Components, Number of Wheels, End-User Industry (Food & Beverages, Retail, Postal)*. Real Books, Inc., 2021.

[2] U. D. of Commerce, *Quarterly E-Commerce Report 1st Quarter*. Washington, D.C., Publication, 2018.

[3] T. Karakurt, “Design of delro autonomous delivery robot and ai based 1itlocalization,” p. 73, 2020.

[4] F. P. K. H. G. F. T. H. M. Lasi H., “Industry 4.0. business & information systems engineering,,” pp. 239–242, 2014.

[5] A. S. Narayanan, “Qr codes and security solutions,” *International Journal of Computer Science and Telecommunications*, vol. 3, no. 7, pp. 891–921, 2012.

[6] S. Vaidiya, P. Ambad, and S. Bhosle, “Industry 4.0 - a glimple,” *2nd International Conference on Materials Manufacturing and Design Engineering*, vol. 12, no. 7, pp. 233–238, 2018.

[7] R. S. J. S. A. S. H. S. Azeem S, “An autonomous courier robot for indoor deliveries,” *Habib University*, p. 114, 2020.

[8] B. Robert, “Robots in a contagious world,” vol. 47, pp. 642–673, 5 2020.

[9] K. Robots, “Delivery robot t6,” 2019.

[10] P. U. P. Y. D. R. J. A. C. S. Gujarathi A. Kulkarni A., “Design and development of autonomous delivery robot,” 2020.

[11] . W. G. Y. Cheng Y., “Mobile robot navigation based on lidar,” *Chinese Control and Decision Conference*, pp. 1243–1246, 2018.

[12] N. Rahim, “Sensor fusion of lidar and camera — an overview,” 2018.

[13] J. Stephan. “Differential drive.” (2020), (visited on 2021).

[14] M. Ferguson. “Rosserial.” (2018), (visited on 2021).

[15] W. M. Tully Foote Eitan Marder-Eppstein. “Tf (transform frame).” (2020), (visited on 2021).

[16] ROSWiki. “Rplidar.” (2019), (visited on 2021).

[17] J. M. Stefan Kohlbrecher. “Hector slam.” (2021), (visited on 2021).

[18] W. Meeussen. “Robot_pose_ekf.” (2012), (visited on 2021).

[19] B. P. Gerkey. “Amcl.” (2020), (visited on 2021).

[20] E. Marder-Eppstein. “Move base.” (2021), (visited on 2021).

CHAPTER 6

VITA

6.1 Muhammad Ammar Khan

Ammar is pursuing Electrical Engineering from Habib University and is from the class of 2022. His ambition of becoming a perfect person lies in the belief of developing a subtle nature, one that is highly skilled but doesn't put on a show about it much like big thinkers who don't need to flaunt about how much information they have because wisdom can only be genuine if it is based on a profound experience that does not need to be displayed because the actions are enough. He thinks that it is in good smell and in eyes, where we can sense completeness achieved primarily through search of peace in hardship. Patience and understanding are both vital ingredients of this jigsaw and humility is a mysterious concept as it appears not to be, but is always beautiful in the end. His life is still developing but is surely reaching what he desires.

6.2 Laraib Aftab Zedie

Laraib is an undergraduate Electrical Engineering student in the Class of 2022. During his undergraduate, he has developed an interest in automation, Power, and Embedded systems through courses like Microcontrollers and Interfacing, Intro to Robotics, Mobile Robotics and, Digital Image Processing, Electric Network Analysis and Power Generation, Transmission and Distribution as well as several projects based around object-oriented programming, Internet of Things, and hardware-software interfacing. He hopes to use the experience of this Capstone project to explore his interests.