Egyptian Academy for Engineering and Advanced Technology

Mechanical Department

Graduation Project (2): MEC 491

A Side-Walk Autonomous Delivery Robot for Efficient Urban Logistics

A graduation project in partial fulfilment of the requirements for the degree Bachelor of Engineering in Mechatronics Program

Submitted by

Abdelhaliem Shaamel Ragab	**2020088**
Mohamed Mamdouh Abdelmontalib	**2020044**
Omar Mohamed Fouad	**2021143**
Mohamed Enshan Abdelhamied	**2020119**
Abdelrahman Ahmed Ali	**2020049**
Nora Mohamed Eldefrawy	**2020001**
Mohamed Abdelhaq Mohamed	**2020084**
Yousef Mohamed Roshdy	**2020163**

Supervisors

Dr. Amin Danial

Dr. Ola Mohammed

Faculty of Engineering, EAE&AT
Cairo, Egypt
June 2025

Egyptian Academy for Engineering and Advanced Technology

Mechanical Department

Graduation Project (2): MEC 491

A Side-Walk Autonomous Delivery Robot for Efficient Urban Logistics

Supervisors

Dr. Amin Danial

Dr. Ola Mohammed

Approved by

Examination Committee:

Dr. Wael Mamdouh

- **EAE&AT**

Dr. Essam Morsi

- **Delta University**

Faculty of Engineering, EAE&AT
Cairo, Egypt
June 2025

Acknowledgment

At this stage of our Graduation Project, we would like to express our gratitude to everyone who supported and assisted us with suggestions, criticisms, and observations, allowing us to work in a friendly yet formal atmosphere. We are deeply grateful to those who were always available to provide useful advice and explanations to help us overcome the challenges we encountered during the development of this work. This invaluable support has enabled us to gain significant knowledge and fully benefit from this remarkable experience.

First and foremost, we extend our heartfelt thanks to our supervisors, Dr. Amin Daniel and Dr. Ola Mohammed, who gave us the opportunity to work on an engaging and contemporary topic. Their guidance, availability, and expertise were crucial throughout our dissertation.

We sincerely thank Dr. Mahmoud Abuhattab for his invaluable assistance with motor sizing, providing critical insights and support during this aspect of our work. We also express our gratitude to Dr. Wael Mamdouh for his expertise in energy management, which greatly contributed to the efficiency of our project.

Special thanks go to Eng. Abdelrahman Essam, whose contributions to the mechanical design were pivotal to the realization of this project. We are equally grateful to Eng. Abdelrahman Ashraf for his expertise in software tools, which greatly facilitated our work.

Special thanks go to Eng Mohamed Abozaid and his Company STEMEC for his help for Providing us the place that we could held meetings and train.

Finally, we would like to thank our families for their unwavering support, encouragement, and understanding throughout this journey. Their belief in us has been an indispensable source of strength and motivation.

Abstract

This project looks at the design and development of a sidewalk autonomous delivery robot to address inefficiencies in urban logistics and the ongoing issues of last-mile delivery. As urbanization and e-commerce grow, cities are under growing pressure to increase delivery efficiency while minimizing traffic congestion and environmental effects. Traditional distribution techniques frequently contribute to urban congestion while incurring significant costs. This study provides a novel solution in the shape of a compact, self-contained robot built to navigate sidewalks and pedestrian walkways, providing a sustainable, cost-effective alternative to traditional courier services.

The sidewalk delivery robot achieves real-time navigation, obstacle recognition, and collision avoidance by utilizing an integrated system of modern sensors such as LiDAR, and ultrasonic sensors. To ensure precise and effective delivery, the robot employs Global Path Planning using the Bing Maps API to generate geographic waypoints that guide it through complex zones like urban areas or logistics hubs. Between these waypoints, the robot uses Local Navigation with the VFH+ (Vector Field Histogram Plus) algorithm to dynamically avoid obstacles in real-time, ensuring smooth and safe operation. The robot's software architecture is intended to process sensory data efficiently and make autonomous judgments while following pedestrian traffic rules. Furthermore, a reliable communication system enables real-time tracking and user involvement, which improves the whole delivery experience.

The robot's performance was thoroughly tested in both simulated and real-world urban settings. Key performance parameters included delivery time, navigation accuracy, battery efficiency, and obstacle-handling ability. The results show that the robot dramatically decreases delivery delays caused by traffic congestion, with an average improvement of up to 30% over typical vehicle-based deliveries. The robot successfully negotiated a variety of urban environments, including congested sidewalks and rough terrain, exhibiting adaptability and reliability.

The project's findings demonstrate the transformational potential of sidewalk autonomous delivery robots in updating urban logistics. These robots can save delivery costs by about 60%, reduce carbon emissions from vehicle deliveries, and improve delivery dependability in highly populated locations. However, extensive implementation confronts several hurdles, including regulatory

compliance, public safety concerns, and infrastructure modification. Future study will concentrate on strengthening the robot's multi-robot coordination abilities, increasing energy economy, and exploring the ethical implications of autonomous systems in shared public places.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADR - Autonomous Delivery Robot

BLDC - Brushless Direct Current (Motor)

CAD - Computer-Aided Design

GPS - Global Positioning System

IMU - Inertial Measurement Unit

LiDAR - Light Detection and Ranging

MDF - Medium-Density Fiberboard

SLAM - Simultaneous Localization and Mapping

FoS - Factor of Safety

PWM - Pulse Width Modulation

RTK - Real-Time Kinematics

ROS - Robot Operating System

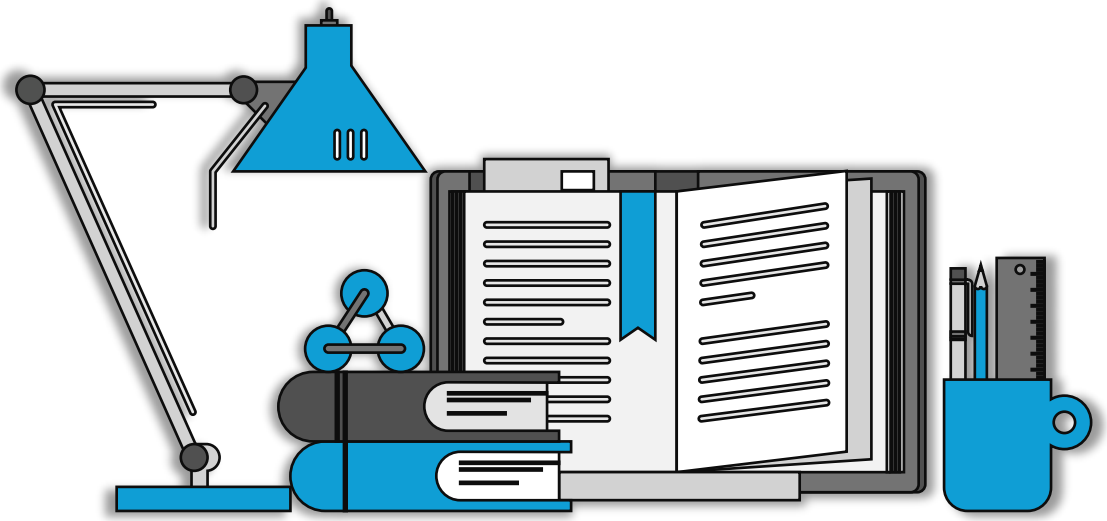
SDG - Sustainable Development Goals

NMEA - National Marine Electronics Association

PVT - Position, Velocity, Time

LORM – Lively Oxygen – Respecting Messenger

Chapter (1): Project Description



1.1 Introduction

In an era where urbanization and e-commerce growth are reshaping cities, Egypt stands at the forefront of these changes. Driven by increasing internet penetration, a young tech-savvy population, and evolving consumer habits, Egypt's e-commerce market reached a value of $6.2 billion in 2020, growing at an annual rate of 17% [1] However, this surge in online shopping has underscored the need for efficient and timely last-mile delivery solutions. Traditional delivery methods relying on cars and motorcycles are hampered by traffic congestion, particularly in urban centers like Cairo, where delays cost the economy approximately 4% of Egypt's GDP annually (about EGP 50 billion) [2]. Compounding these issues are inefficient infrastructure, unreliable addressing systems, and the dominance of cash-on-delivery (COD) transactions.

To address these challenges, sidewalk autonomous delivery robots offer a transformative and sustainable solution. These compact, intelligent machines can navigate pedestrian pathways, avoid road congestion, and deliver packages efficiently and cost-effectively. With advancements in robotics, AI, and sensor technology, these robots are capable of safely traversing urban environments, detecting obstacles, and interacting with their surroundings in real time. By leveraging sidewalks for deliveries, they can significantly reduce delays, lower delivery costs, and decrease carbon emissions, thus contributing to a greener urban environment. [3]

This project focuses on the design, development, and deployment of sidewalk autonomous delivery robots to revolutionize urban logistics in Egypt. By mitigating traffic congestion, improving delivery efficiency, and addressing environmental concerns, these robots can support Egypt's rapidly expanding e-commerce ecosystem and enhance logistics for startups, businesses, and consumers. Implementing such innovative solutions is timely and crucial for positioning Egypt as a leader in smart delivery systems within the MENA region and beyond [4], [5].

1.2 Literature Review

1.2.1 Introduction to Urban Logistics and Delivery Challenges

Urban logistics, especially last-mile delivery, faces significant challenges due to increasing urbanization and the rise of e-commerce. Congestion, environmental impact, and delivery

inefficiencies are major concerns for cities worldwide. The need for innovative, efficient, and sustainable delivery methods has driven the exploration of autonomous delivery systems, such as sidewalk delivery robots. These robots aim to reduce delivery times, ease traffic congestion, and offer cost-effective solutions.

1.2.2 Existing Autonomous Delivery Solutions

1. **Sidewalk Robots:** Sidewalk delivery robots are compact, autonomous machines designed to navigate pedestrian pathways. Prominent examples include Starship Technologies' robots, Amazon Scout, and Kiwibot.

Figure 1.2: Prime Scout

Figure 1.1: Starship Technologies

2. **Drones:** Drones are airborne delivery devices designed to transfer lightweight products over short to medium distances. They thrive in reaching isolated or difficult-to-access areas. Drones are equipped with GPS, cameras, and automatic navigation systems to assure fast delivery. Examples: Wing (by Alphabet), Amazon Prime Air, Zipline.

Figure 1.4: Amazon Prime Air Drone

Figure 1.3: Wing Drone

3. **Autonomous Vehicles:** Autonomous delivery vehicles are self-driving automobiles or vans that can handle heavier cargoes and travel longer distances. These cars travel on roadways and use modern technology such as LiDAR, cameras, and GPS to navigate and assure safe delivery. Examples: Nuro, Udely, Ford and Postmates Collaboration.

Figure 1.5: Udely Autonomous Vehicle

1.2.3 Technologies Behind Autonomous Navigation

- **Sensing Systems:** Sidewalk delivery robots rely on a variety of sensors to perceive their environment. Key sensors include:

  LiDAR: Provides detailed 3D mapping for navigation and obstacle detection.

  Cameras: Used for visual recognition of objects and pedestrians.

  Ultrasonic Sensors: Assist in detecting nearby obstacles and measuring distances.

- **Path Planning Algorithms:** Path planning is critical for autonomous navigation. Common algorithms include:

  A* Algorithm: Finds the shortest path by considering cost and heuristic estimates.

  RRT Algorithm: sampling-based path planning algorithm that builds a tree to explore spaces and find paths while avoiding obstacles.

  Reinforcement Learning: Allows robots to learn optimal paths based on experience.

- **Obstacle Detection and Avoidance:** Obstacle detection and avoidance are essential for safe navigation. Techniques such as Simultaneous Localization and Mapping (SLAM) are widely used. SLAM allows robots to build a map of their environment while determining their location within it.

1.2.4 Regulatory and Social Considerations

Deploying sidewalk delivery robots raises regulatory and social issues

Pedestrian Safety: Ensuring robots do not obstruct or endanger pedestrians.

Accessibility: Maintaining sidewalk accessibility for people with disabilities.

Public Perception: Studies show mixed reactions to sidewalk robots, with concerns about privacy and safety.

1.2.5 Identified Gaps and Opportunities

Despite advancements, there are gaps in current sidewalk delivery robot technologies

Technological Gaps: Improved navigation in highly crowded areas is needed.

Operational Gaps: Enhancing battery life and delivery speed.

Regulatory Gaps: Developing standardized policies for widespread deployment.

1.2.6 Traffic Congestion Impact on Delivery Delays

Traffic congestion significantly affects delivery times in urban areas. In cities like New York and San Francisco, delivery delays can average between 25% and 35%. The chart below illustrates the impact of congestion on delivery delays across several major cities.

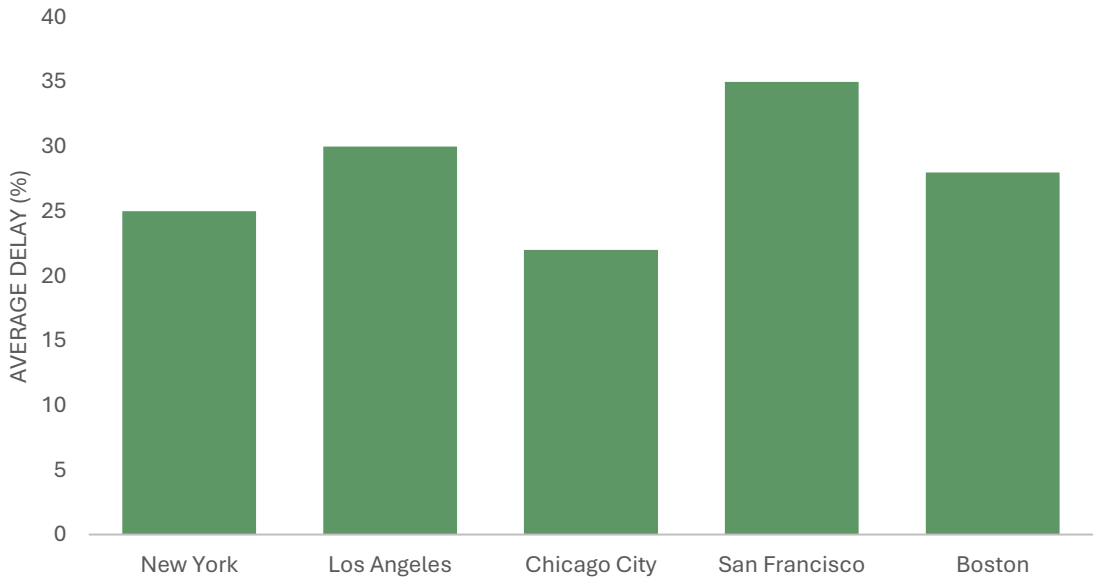


Figure 1.6: Traffic Congestion impact on Delivery Delays in Major Cities

1.2.7 Average Cost Per Last-Mile Delivery

The cost of last-mile delivery varies depending on the method used. Traditional courier services are the most expensive, with costs averaging around $10 per delivery. In contrast, sidewalk delivery robots offer a more economical solution, with costs averaging $4 per delivery. The chart below compares the average costs for different last-mile delivery methods.

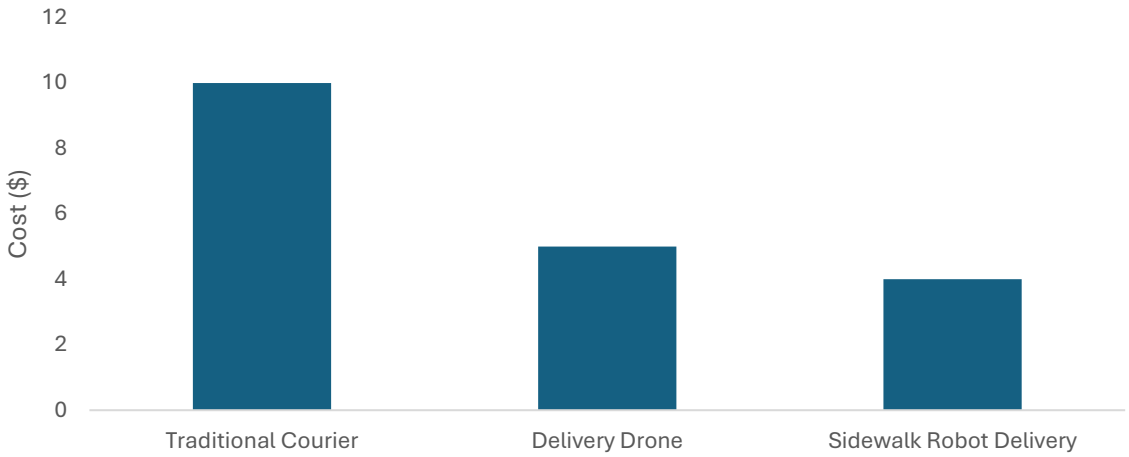


Figure 1.7: Average Cost per last-mile Delivery

1.2.8 Adoption Rates of Autonomous Delivery Methods

Among autonomous delivery methods, sidewalk robots have the highest adoption rate at 45%, followed by drones at 30% and autonomous vehicles at 25%. The pie chart below shows the distribution of adoption rates.

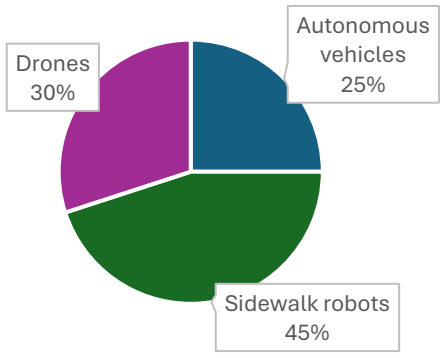


Figure 1.8: Adoption Rates of Autonomous Delivery Methods

1.2.9 Conclusion

The literature review highlights the potential of sidewalk autonomous delivery robots in addressing urban logistics challenges. By analyzing existing technologies, navigation methods,

and regulatory issues, this project aims to advance the field and provide an efficient solution for last-mile delivery.

1.3 Problem Statement

1. **Traffic Congestion:** Cairo ranks among the most congested cities in the world, with traffic delays costing the Egyptian economy approximately **EGP 50 billion annually** (about **4% of GDP**) Traditional delivery vehicles are frequently delayed by dense traffic, leading to inefficiencies and late deliveries [2].
2. **Inadequate Addressing System:** Egypt's lack of standardized postal codes, especially in informal and densely populated urban areas, makes accurate deliveries difficult This results in failed deliveries, with an estimated **40% of online orders** falling through due to logistical issues
3. **High Costs and Delays:** Last-mile delivery accounts for up to **50% of total delivery costs** Inefficiencies caused by traffic, manual processes, and logistical errors lead to increased fuel consumption, higher operational costs, and significant delivery delays.
4. **Environmental Impact:** The reliance on fuel-based delivery vehicles contributes to worsening air pollution in Egypt's already congested cities. Urban freight emissions account for a significant portion of the pollution burden

Conclusion:

An **autonomous sidewalk delivery robot** offers a viable and innovative solution to these challenges. By bypassing road traffic using sidewalks, employing efficient navigation algorithms, and automating delivery processes, this project aims to, Enhance delivery speed and reliability, Reduce operational costs and fuel consumption, Mitigate environmental impact by using electric-powered robots.

This solution addresses the growing demand for efficient, sustainable urban logistics and positions Egypt as a potential leader in smart delivery technologies within the **MENA region**.

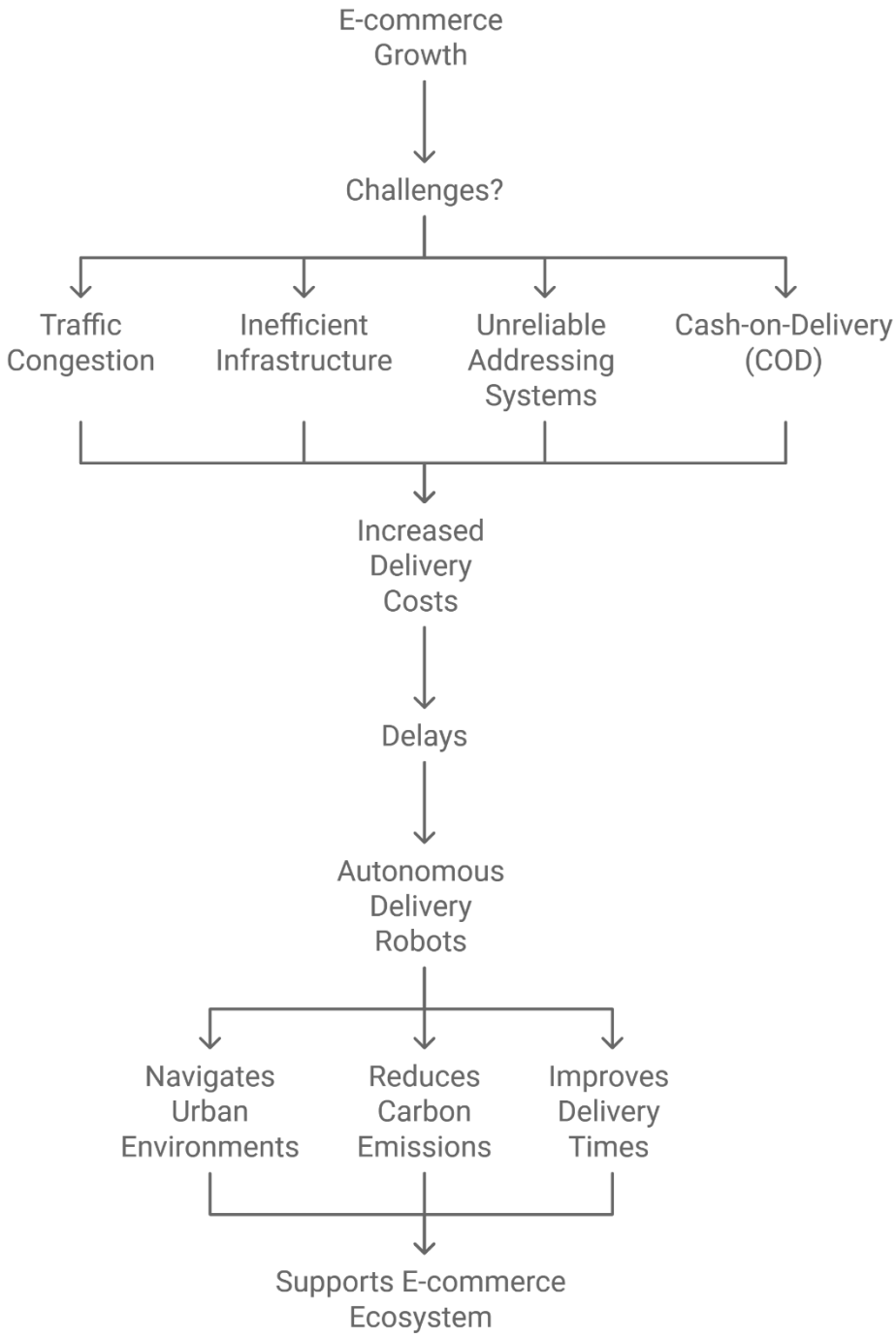


Figure 1.9: Problem Statement map

1.4 Motivation

1.4.1 Motivation for Egypt

1. **E-commerce Boom:** Egypt's e-commerce market is valued at $6.2 billion and growing at a rate of 17% annually Meeting the increasing demand for timely deliveries is critical for sustaining this growth.

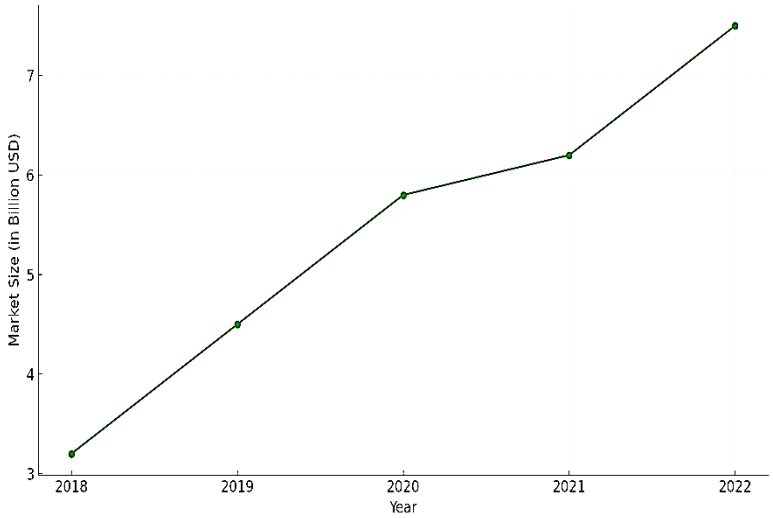


Figure 1.10: E-commerce Market Growth in Egypt (2018-2022)

2. **Cost-Effectiveness:** Autonomous robots can operate wih lower labor and operational costs compared to traditional delivery methods.

3. **Environmental Concerns:** Reducing emissions and traffic congestion is necessary for improving air quality and quality of life in urban areas

4. **Innovation & Technological Advancement:** Positioning Egypt as a leader in smart urban logistics and aligning with Egypt Vision 2030 goals for sustainable development and infrastructure improvement

1.4.2 Motivation for the MENA Region

1. **Market Potential:** The MENA region's e-commerce market is expected to reach $50 billion by 2025.

2. **Urbanization Challenges:** Rapid urbanization and population growth exacerbate traffic and logistics challenges in cities like Dubai, Riyadh, and Casablanca.

3. **Technological Adoption:** The region is increasingly investing in smart city initiatives and advanced technologies, making it ripe for autonomous logistics solutions.

1.4.3 Global Motivation

1. **Scalability:** Urban centers worldwide face similar challenges with traffic congestion, inefficient deliveries, and environmental concerns.

2. **Sustainability Goals:** Aligning with the United Nations Sustainable Development Goals (SDGs), such as SDG 11 (Sustainable Cities) and SDG 13 (Climate Action).

3. **Innovation Leadership:** Contributing to the global transition toward autonomous and smart delivery systems, enhancing efficiency, and reducing carbon footprints.

1.5 Market Need

The Market for sidewalk autonomous delivery robots is driven by e-commerce growth, traffic congestion, environmental goals, and the need for cost-effective and speedy deliveries. While these trends are evident worldwide, countries like Egypt, facing specific urban challenges, stand to benefit significantly from this technology. Delivery robots offer a transformative solution that aligns with the future of smart, sustainable cities.

1.5.1 Rapid Growth of E-commerce Worldwide

Global e-commerce sales are expected to reach $6.3 trillion in 2024 and continue to grow rapidly. This surge is driven by increased internet penetration, changing consumer behavior, and a demand for convenience.

Countries like the United States, China, India, and the European Union are experiencing double-digit annual growth in online shopping, increasing pressure on traditional delivery systems.

1.5.2 Challenges of Traffic Congestion

Urban centers worldwide, such as New York, London, Tokyo, and Cairo, suffer from severe traffic congestion.

For instance, traffic delays cost the U.S. economy $87 billion annually in wasted time and fuel.

Delivery vehicles contribute to this congestion, slowing down last-mile deliveries. Sidewalk delivery robots offer a solution by operating on pedestrian pathways, bypassing road traffic, and speeding up deliveries.

1.5.3 Rising Environmental Concerns

With over 25% of global CO_2 emissions stemming from transportation, sustainable solutions are critical., Many governments, including the European Union, Japan, and Egypt, are setting ambitious goals to reduce carbon footprints.

Delivery robots are typically electric-powered and produce zero emissions, making them a greener alternative to fuel-based vehicles.

1.5.4 Cost-Efficient and Scalable Solutions

Traditional delivery methods involve high operational costs due to fuel, labor, and vehicle maintenance. For example, in some regions, labor accounts for 50-60% of delivery costs.

Autonomous delivery robots reduce these costs by:

1. Eliminating the need for human drivers.
2. Operating efficiently on battery power.
3. Being scalable for various types of deliveries, from food to small parcels.

1.5.5 Consumer Demand for Faster and Flexible Delivery

Modern consumers expect faster, more reliable deliveries, often within hours or on the same day. Robots can operate 24/7 and offer quick deliveries by avoiding road traffic and using direct pedestrian routes. This meets the increasing demand for convenience and speed.

1.5.6 Urban Logistics Challenges

Cities worldwide face challenges such as:

1. Narrow streets and limited parking in dense urban areas.
2. Inefficient addressing systems in some regions, including parts of Africa and the Middle East.

Autonomous robots address these issues by delivering them directly to doorsteps without needing parking spaces or precise addresses.

1.5.7 Global Adoption and Investment Trends

Companies like Starship Technologies, Amazon Scout, Nuro, and FedEx are actively deploying delivery robots in cities across the U.S., UK, and China.

The sidewalk delivery robot market is projected to reach $1 billion by 2030, reflecting increasing confidence in this technology.

Countries like Egypt and India are exploring these innovations to improve urban delivery efficiency.

1.5.8 Versatility Across Industries

These robots are not limited to e-commerce and can serve multiple sectors:

1. Food Delivery: Restaurants and food chains use robots for contactless delivery.
2. Pharmaceuticals: Robots can deliver medicines and supplies efficiently.
3. Retail: Stores can fulfill local deliveries quickly and cost-effectivel.

1.6 Project Scope

1.6.1 Geographical Focus

- **Initial Deployment:** The project will be implemented in major Egyptian cities, such as Cairo, Alexandria, and Giza, where traffic congestion, inefficient last-mile delivery, and environmental concerns are most acute.
- **Expansion Plan:** Following successful deployment in Egypt, the project will scale to other cities in the MENA region and eventually target urban centers worldwide.

1.6.2 Core Features

- **Navigation & Routing:** Development of advanced AI-driven navigation systems capable of safely maneuvering on sidewalks.
- **Delivery Mechanism:** Secure and efficient handling of small-to-medium packages for last-mile delivery.
- **Connectivity:** Integration with mobile applications for real-time tracking, scheduling, and customer notifications.
- **Safety & Compliance:** Ensuring the robot complies with pedestrian safety standards and local regulations.

1.6.3 Technological Aspects

- Use of computer vision, machine learning, and GPS for accurate navigation and obstacle detection.
- Electric-powered and eco-friendly design to reduce carbon footprints.
- Integration with logistics platforms and e-commerce businesses to streamline delivery operations.

1.6.4 Operational Goals

- **Efficiency:** Reducing average delivery times by 30-50% compared to traditional delivery vehicles.
- **Cost Reduction:** Lowering operational costs for last-mile logistics by minimizing fuel usage and labor dependency.
- **Sustainability:** Decreasing carbon emissions and traffic congestion in urban areas.

Chapter (2): Introduction to LORM

Every city tells a story. In Egypt, it's the story of life moving fast — crowded streets, endless horns, couriers weaving through traffic just to deliver a single package. But as the streets fill up and delivery expectations grow higher, one question kept echoing in our minds: *Can we do it better?* That question is where **LORM** began.

LORM stands for **Lively Oxygen-Respecting Messenger**—but to us, it means a lot more than just four words. It's the name we gave to the sidewalk delivery robot we imagined, designed, built, and tested as a team of passionate mechatronics students. It's our answer to real problems we see every day: delayed deliveries, congested roads, high costs, and a city gasping for cleaner air.

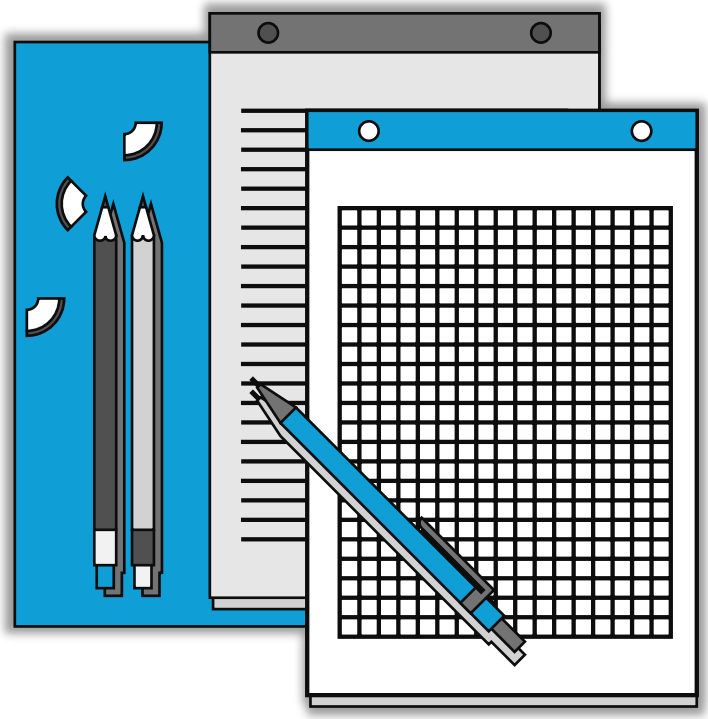
We didn't want to build a robot just because it's "cool" or trending. We wanted to build something that matters. Something that could **weave through a crowded sidewalk**, find its way using smart sensors, deliver a parcel safely, and do it all without a single drop of fuel. We wanted to prove that **engineering isn't just about machines—it's about improving lives**, While other delivery robots in places like the US and Europe are already rolling, they aren't designed for Cairo's sidewalks or our local logistics struggles. Our infrastructure, traffic behavior, and even addressing system are different—and that's exactly why we believed **Egypt needs its own solution**. Not imported. Not imagined elsewhere. Built here, by people who know the streets.

So we gave LORM eyes through **LiDAR and ultrasonic sensors**, a sense of direction through **GPS and IMU**, and a brain powered by smart algorithms like **A*** and **SLAM**. We gave it wheels, a heart (our code), and a mission: **to move with purpose**—quietly, cleanly, and efficiently.

But honestly? LORM is also a reflection of us. Of who we are becoming as engineers. This project taught us more than any textbook ever could. It taught us how to think across disciplines, how to deal with things that didn't go as planned, and how to stay inspired even when the wheels weren't turning yet.

More than anything, it taught us that technology doesn't have to be futuristic or foreign to be amazing. It can be local. It can be ours. this is LORM—**our vision walking on wheels**. Built for Egypt, shaped by the challenges of our cities, and guided by the belief that when problems feel too big, maybe the solution just needs to be smarter… and smaller.

Chapter (3): System Overview



3.1 System Block Diagram

This diagram represents the key components and workflow of an **Autonomous Robot System** within its operational environment. The system integrates sensors, processing units, motor drivers, and communication management to enable autonomous navigation and decision-making.

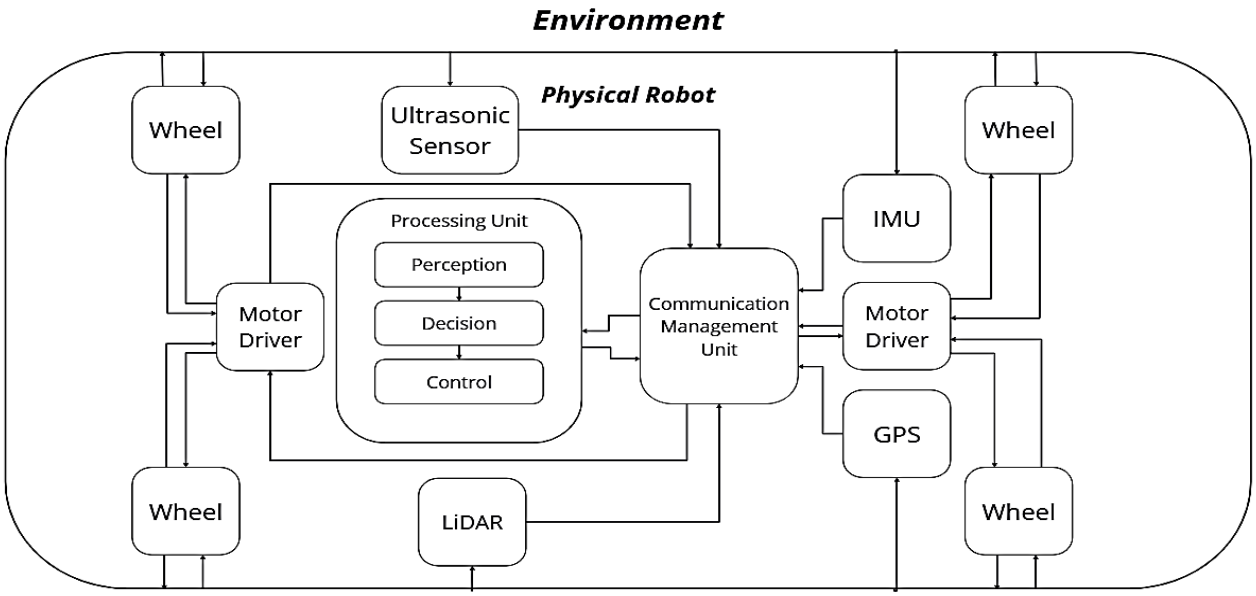


Figure 3.1: System block Diagram

Processing Unit

The core unit is responsible for making decisions.

1. Perception: Collects data from sensors (e.g., ultrasonic sensor, LiDAR, GPS, and IMU) to understand the robot's surroundings.
2. Decision: Processes the data to make decisions on how to navigate or avoid obstacles.
3. Control: Sends commands to the motor drivers to execute movements.

Sensors

- Ultrasonic Sensor: Detects nearby obstacles by emitting ultrasonic waves and measuring reflections.
- LiDAR (Light Detecting and Ranging): Function: Provides detailed mapping of the environment by measuring distances using laser beams.
- IMU (Inertial Measurement Unit): Measures the robot's orientation, acceleration, and angular velocity.
- GPS (Global Positioning System): Provides location data to help the robot navigate accurately.

Motor Drivers

Receive control signals from the processing unit and provide the necessary power to the wheels to move the robot.

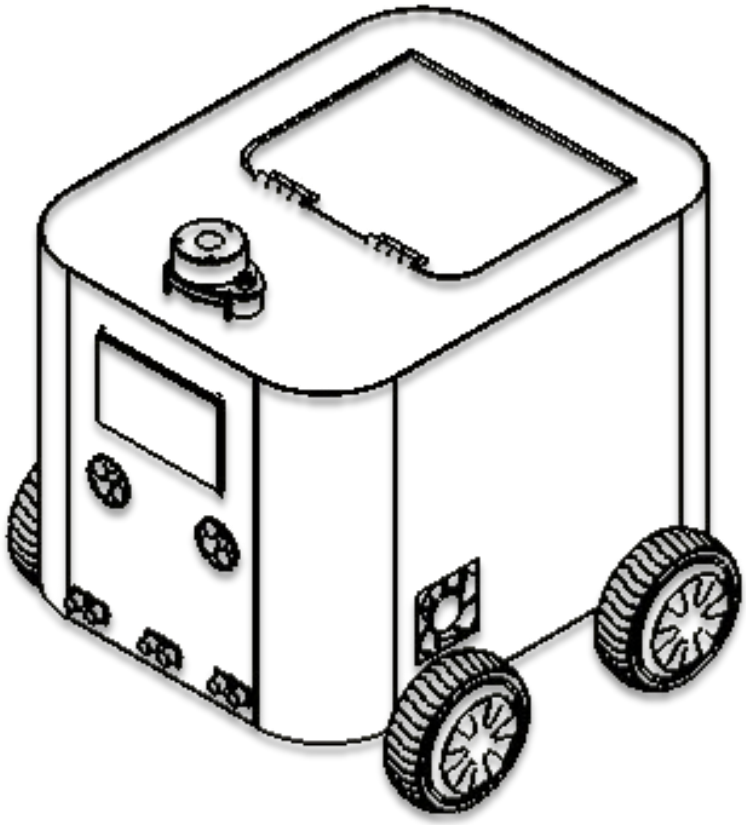
There are two motor drivers, each controlling a set of wheels to enable smooth movement and steering.

Communication Management Unit

Facilitates data transfer between the sensors, processing unit, and motor drivers.

Ensures commands and data flow smoothly between all components, allowing the system to operate cohesively.

Chapter (4): Mechanical Methodology



Structural Analysis, Material Optimization, and Comprehensive Development From conception to manufacture and testing, the autonomous delivery robot's mechanical design covers every facet of its construction. The foundation of the robot's operation, this project phase makes sure it functions dependably in a variety of scenarios. The robot is brought to life by mechanical design, which connects abstract ideas with real-world uses [6].

This section describes the methodical process used to design, develop, and improve the robot in order to make sure it is appropriate for the autonomous delivery task for which it was built. Every stage of the mechanical design process was carefully managed, from choosing suitable materials and assessing structural integrity to specifying dimensions based on market research. By using sophisticated programs like SolidWorks, carrying out thorough testing, and analysis, we made sure the robot would execute to the expected level and beyond [7].

Each step of this procedure is described in depth in the sections that follow.

4.1 Mechanical Body Design

4.1.1 Material

A key component of mechanical design is material selection, which has a direct bearing on the robot's weight, strength, and general functionality.

4.1.1.1 Selection Criteria

Table 4-1: Material Selection

criterion	MDF	Aluminum	steel
Stiffness	1	4	5
Availability	5	5	5
weight	5	3	2
Heat resistance	2	1	4
Manufacturing capability	5	3	4
cost	5	1	3
durability	2	3	5
load bearing capacity	2	3	5
Total	**27**	**23**	**33**

4.1.1.2 Prototype

Medium-Density Fiberboard (MDF) was the material used:

Because It is Cost-effective and lightweight; simple to CNC laser cutting machine. facilitated effective iteration and quick prototyping in the early phases of the design process [8].

A physical model of the robot was made available for testing its performance and construction.

4.1.1.3 Final Design

Steel Was the material used:

Because It is High tensile strength and durability. Resistance to wear, deformation, and environmental conditions. Steel is perfect for a robot meant for real-world applications because of its structural stability. it can tolerate dynamic loads and strains.

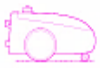
The requirement for long-term performance and durability led to the switch from MDF to steel, which signified the robot's development from **Concept** to **Reality**.

4.1.2 Dimensions

To identify market trends and set benchmarks, a thorough investigation was conducted before the design process started. Critical information on dimensions, weight distributions, and important design elements that affect usability and performance was discovered through an analysis of current delivery robots.

The KiwiBot model was chosen as the design inspiration because of its demonstrated effectiveness, small size, and commercial success [9]. A number of autonomous delivery robots were examined, with particular attention paid to their operating capabilities, structural characteristics, and dimensions.

Table 4-2: Different ADRs Specifications

Name	Design	Length (Cm)	Width (Cm)	Height (Cm)	Weight (Kg)	Max Speed (Km/h)	Max Payload (Kg)	Designer	Countries	Year
Starship Robot		67.8	56.9	55.4	23	6	10	Starship Technologies	United States United Kingdom Germany Estonia Finland	2014
KiwiBot		55.9	43.2	55.9	21	2.4	13	Kiwibot	United States	2017
Serve		79	64	105	73	10	23	Serve Robotics	United States	2018
TeleRetail Delivery Robot		147.3	71.1	86.4	27	56	35	Aitonomi TeleRetail	United States Switzerland	2018
DeliRo Delivery Robot		96.2	66.4	108.9	30	6	50	ZMP	Japan	2019
Amazon Scout		76.2	61	73.7	45	24	23	Amazon	United States	2019
FedEx Roxo		91.4	71.1	147.3	91	16	45	FedEx	Japan	2019
Refraction REV-1		137	76.2	96.5	45	24	127	Refraction	United States	2019
Robomart		347	140	176.5	Custom	40	Custom	Robomart	United States	2019
Nuro R2		274	110	186	1150	40	190	Nuro	United States	2020

best worst

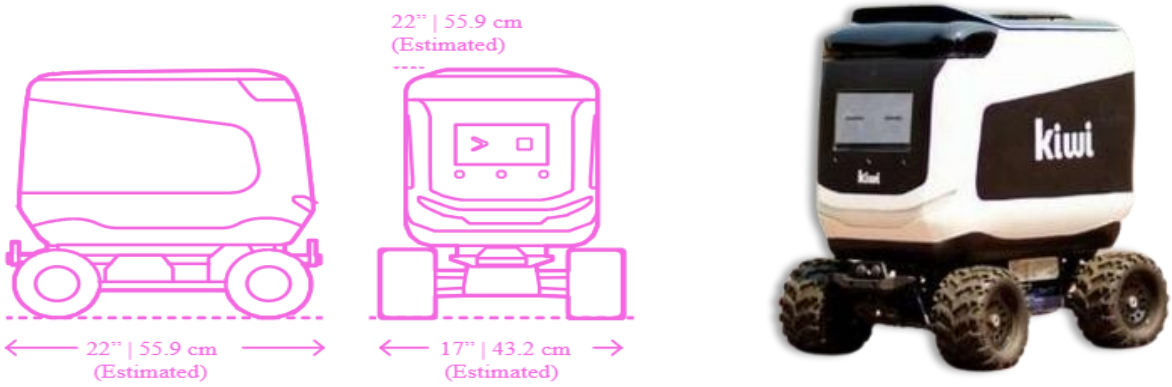


Figure 4.1: Kiwibot Dimensions

Prototype Dimensions: 560 × 430 × 564.3 mm

To guarantee compatibility with urban environments, these measurements were computed as averages using market data. The design was intended to combine enough interior room for parts and freight with compactness for agility.

The Final Steel Design Dimensions: 566.5 × 436.5 × 564.3 mm

Small modifications were made to maintain a useful form factor while improving durability and accommodating better structural characteristics.

Examines the connection between the size of a delivery robot and its rotational efficiency, stability, and mobility. It offers insights on the ideal design for a robot that can spin in situ while preserving efficiency and balance by examining particular ratios of length, breadth, height, wheel distance, and wheel diameter. The conclusions are supported by well-established theories of robotics dynamics and mechanical design.

Autonomous delivery robots work in conditions that need a high degree of stability and mobility. Finding a balance between these two characteristics while guaranteeing seamless functioning is a crucial component of their design. We study important dimensional ratios for a self-sufficient delivery robot that meets the following requirements.

Length:	56	cm	Wheelbase:	28	cm
Width:	43	cm	Wheel Diameter:	16.5	cm
Height:	56	cm			

4.1.2.1 Key Dimensional Ratios

- **Length to Width Ratio:** The length (56 cm) to width (43 cm) ratio is around 1.3:1

  Studies indicate that for robots working on flat surfaces:

  - length-to-width ratios of 1:1 to 2:1 offer the best stability.
  - While ratios below 1:1 may cause instability during linear motion.
  - ratios above 2:1 may reduce turning efficiency [10].

- **Wheelbase to Length Ratio:** There is a 1:2 ratio between the wheelbase (28 cm) and length (56 cm).
  - In-place rotation is made possible by shorter wheelbases in relation to the robot's length, which decrease the turning radius.
  - According to research on wheeled robots, zero-radius turning is best achieved with a wheelbase-to-length ratio of 1:2 or less.

- **Width to Wheelbase Ratio:** Ratio of Width to Wheelbase The ratio of width (43 cm) to wheelbase (28 cm) is 1.54:1.
  - Lateral stability is ensured by a ratio larger than 1. particularly while making abrupt turns. During fast rotating motions, **ratios less than one might provide tipping hazards.**

- **Diameter to Dimension of the Wheel Ratios:** About 30% of the robot's length and 59% of its wheelbase are made up of the wheel diameter, which is 16.5 cm.
  - While excessively big wheels might make accurate turns difficult,larger wheels perform better on rough terrain and enable smoother mobility [11].
  - For general-purpose robots, ratios between 25–40% of length and 50–70% of wheelbase are thought to be ideal.

4.1.2.2 Analysis of Turning Radius in Rotational Maneuverability

- **Zero-Radius Turning:** is accomplished by programming the wheels to turn in opposing directions at identical speeds. This is possible when the torque given to the wheels is enough to exceed the robot's breadth and the wheelbase is symmetrical.

- **Motion Stability:** While Rotating The center of gravity is at a suitable level for steady in-place rotation with a height (56 cm) to width (43 cm) ratio of 1.3:1. The chance of tipping is reduced by positioning heavier parts (like batteries) close to the base.

4.1.3 Wheels

For the robot to move smoothly and remain stable, its wheels are essential, especially in metropolitan settings with different topography.

Wheel Diameter Assumption: 165 mm

Designed to provide the robot with the most possible ground clearance so it can maneuver over minor obstacles and rough terrain.

Stability: By keeping the center of gravity low, the selected wheel size lowers the chance of tipping while in use.

Performance: The wheels' capacity to manage various terrains was assessed to guarantee dependable traction and smooth travel.
The entire functionality and operating efficiency of the robot are greatly enhanced by the careful selection of wheel size and materials.

4.2 Motors

4.2.1 Selection Criteria

Table 4-3:Motor Selection

Criterion	Brushless DC	Stepper	Easy servo	Servo
Availability	4	4	3	2

Criterion	Brushless DC	Stepper	Easy servo	Servo
Cost	4	4	2	1
Noise & Vibration	5	3	4	4
Torque vs. Speed	5	3	4	5
Durability & Lifespan	4	3	4	4
Precision	4	5	4	4
Efficiency	5	3	4	5
Total	**35**	**27**	**28**	**29**

4.2.2 Motor Sizing

Motor sizing doesn't depend only on equations and laws, but also depends on a lot of experiences and knowledge for good selection.

Table 4-4:BLDC Motor Selection

Criterion	HOVERBOARD BLDC	MAXON BLDC	BOSCH BLDC	INDUSTRIAL BLDC
Availability	4	2	2	2
Cost	4	1	2	1
Speed Capability	4	5	5	5
Torque Performance	4	4	5	5
Size & Weight	5	5	5	1
Precision	3	5	4	5

Criterion	HOVERBOARD BLDC	MAXON BLDC	BOSCH BLDC	INDUSTRIAL BLDC
Efficiency	4	4	4	5
Total	**28**	**26**	**27**	**24**

we select **Hoverboard Brushless DC motors** for many reasons:

The most important are budget, lack of capabilities, lack of motors and their drivers, and wheels that we need and overpriced most of the components.

Figure 4.2: Hoverboard BLDC Motor & wheel

We need to know the specifications of the motor that we need in our project by using some laws and calculating some Forces.

Our Give Data

Height of the body = Total height – Wheel radius – bracket – LiDAR height = 40 Cm

Body Characteristics							Wheel Characteristics
length	Width	Height	Mass of the body	Max. payload	Total mass	Frontal Area	Radius
0.56	0.43	0.4	28	12	40	0.172	0.08255
m	m	m	kg	kg	kg	m^2	m

Variables We Can Control		Constants				
Theta of Climbing	Desired Speed	Coefficient of rolling resistance	Gravity	Density of Air	Drag Coefficient	Efficiency of the motor
2	3	0.015	9.81	1.23	1.2	0.9
deg	m/s		m/s^2	kg/m^3		

Runtime Calculations

Force Calculations

Rolling Force

$$F_{Rolling} = \mu \times m \times g$$

Aerodynamics Force

$$F_{Aerodynamics} = 0.5 \times \rho \times v^2 \times C_A \times A_F$$

Gradient Force

$$F_{Gradient} = m \times g \times sin(\theta)$$

Total Forces

$$F_{Total} = F_{Rolling} + F_{Aerodynamics} + F_{Gradient}$$

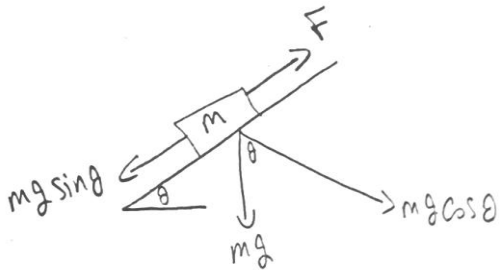


Figure 4.3: Gradient Force Analysis

Torque Calculations

Tractive Torque

$$T_{Tractive} = F_{Total} \times r_w$$

Power Calculations

Required Tractive Power

$$P_{Tractive} = T_{Tractive} \times \omega \times fos$$

Electrical Power for 1 Wheel

$$P_{elec} = {}^{P_{Tractive}}/_{\eta}$$

Maximum Current for 1 Wheel

$$I_{max} = {}^{P_{elec}}/_{Rated\ Voltage}$$

Where:

μ: Rolling Resistance Coefficient

m: Total mass (Body & Max. Payload)

g: Standard Gravity

θ: Theta of Climbing

ρ: Density of air at 15°C at sea level

C_A: Coefficient of Air Resistance, Where C_A is a function of several parameters like body shape, Reynolds number, and Froud number.

A_F: Frontal Area of the ADR

r_w: Wheel Radius

v: The Desired Speed

ω : Angular Velocity

η : Efficiency of the motor

FoS: Factor of Safety = 1.5

Angular Velocity

$$\omega = v \div r_w$$

$$\omega = 3 \div 0.08255 = 36.3416\ rad/sec$$

Required Speed

$$Requierd\ Speed = \frac{60 \times \omega}{2\pi}$$

$$Required\ Speed = \frac{60 \times 36.3416}{2 \times 3.14} = 347.036\ rpm$$

Now We have 4 Cases to Calculate Runtime:

- **1st Case: Inclined with Maximum load (worst case)**

Total Mass = Mass of body + Payload = 28 + 12 = 40 Kg

Theta of Climbing = 2°

$$F_{Rolling} = 0.015 \times 40 \times 9.81 = 5.886\ N \quad (3.1)$$

$$F_{Aerodynamics} = 0.5 \times 1.23 \times 3^2 \times 1.2 \times 0.172 = 1.1424\ N \quad (3.2)$$

$$F_{Gradient} = 40 \times 9.81 \times sin(2) = 13.6945\ N \quad (3.3)$$

$$F_{Total} = 5.886 + 1.1424 + 13.6945 = 20.7229\ N \quad (3.4)$$

$$T_{Tractive} = 20.7229 \times 0.08255 = 1.7106\ N.m \quad (3.5)$$

$$P_{Tractive} = 1.7106 \times 36.3416 \times 1.5 = 93.2489\ W \quad (3.6)$$

$$P_{elec} = {}^{93.2489}/_{0.9} = 103.6099\ W \quad (3.7)$$

Electrical Power for 4 wheels = 103.6099*4 =414.4397 W **(3.8)**

$$I_{max} = {103.6099}/{36} = 2.878\ A \quad (3.9)$$

Maximum Current for 4 wheels = 2.878*4 =11.5122 A (3.10)

Runtime = Battery Capacity/Maximum Current = 8.8/11.5122 = 0.7644 Hrs = 45.8643 min

- **2nd Case: Inclined with No load**

Total Mass = Mass of body = 28 kg

Theta of Climbing = 2°

$$F_{Rolling} = 0.015 \times 28 \times 9.81 = 4.1202\ N \quad (3.11)$$

$$F_{Aerodynamics} = 0.5 \times 1.23 \times 3^2 \times 1.2 \times 0.172 = 1.1424\ N \quad (3.12)$$

$$F_{Gradient} = 28 \times 9.81 \times sin(2) = 9.5861\ N \quad (3.13)$$

$$F_{Total} = 4.1202 + 1.1424 + 9.5861 = 14.8487\ N \quad (3.14)$$

$$T_{Tractive} = 14.8487 \times 0.08255 = 1.2257\ N.m \quad (3.15)$$

$$P_{Tractive} = 1.2257 \times 36.3416 \times 1.5 = 66.8158\ W \quad (3.16)$$

$$P_{elec} = {66.8158}/{0.9} = 74.2398\ W \quad (3.17)$$

Electrical Power for 4 wheels = 103.6099*4 =296.9592 W (3.18)

$$I_{max} = {74.2398}/{36} = 2.0622\ A \quad (3.19)$$

Maximum Current for 4 wheels = 2.878*4 =8.2488 A (3.20)

Runtime = Battery Capacity/Maximum Current = 8.8/8.2488 = 1.0668 Hrs = 64 min

- **3rd Case: Flat with Max load**

Total Mass = Mass of body + Payload = 28 + 12=40 Kg

Theta of Climbing = 0°

$$F_{Rolling} = 0.015 \times 40 \times 9.81 = 5.886\ N \quad (3.21)$$

$$F_{Aerodynamics} = 0.5 \times 1.23 \times 3^2 \times 1.2 \times 0.172 = 1.1424\ N \quad (3.22)$$

$$F_{Gradient} = 40 \times 9.81 \times sin(0) = 0 \quad (3.23)$$

$$F_{Total} = 5.886 + 1.1424 + 0 = 7.0284\ N \quad (3.24)$$

$$T_{Tractive} = 7.0284 \times 0.08255 = 0.58019\ N.m \quad (3.25)$$

$$P_{Tractive} = 0.58019 \times 36.3416 \times 1.5 = 31.6278\ W \quad (3.26)$$

$$P_{elec} = {}^{31.6278}/_{0.9} = 35.142\ W \quad (3.27)$$

Electrical Power for 4 wheels = 31.6278*4 =296.9592 W (3.28)

$$I_{max} = {}^{35.142}/_{36} = 0.9761\ A \quad (3.29)$$

Maximum Current for 4 wheels = 0.9761*4 =3.9046 A (3.30)

Runtime = Battery Capacity/Maximum Current = 8.8/3.9046 = 2.2537 Hrs = 135.222 min

- **4th Case: Flat with No load**

Total Mass = Mass of body = 28 Kg

Theat of Climb = 0°

$$F_{Rolling} = 0.015 \times 28 \times 9.81 = 4.1202\ N \quad (3.31)$$

$$F_{Aerodynamics} = 0.5 \times 1.23 \times 3^2 \times 1.2 \times 0.172 = 1.1424\ N \quad (3.32)$$

$$F_{Gradient} = 28 \times 9.81 \times sin(0) = 0 \quad (3.33)$$

$$F_{Total} = 4.1202 + 1.1424 + 0 = 5.2626\ N \quad (3.34)$$

$$T_{Tractive} = 5.2626 \times 0.08255 = 0.4344\ N.m \quad (3.35)$$

$$P_{Tractive} = 0.4344 \times 36.3416 \times 1.5 = 23.6817\ W \quad (3.36)$$

$$P_{elec} = {}^{23.6817}/_{0.9} = 26.313\ W \quad (3.37)$$

Electrical Power for 4 wheels = 26.313*4 =105.252 W (3.38)

$$I_{max} = {}^{26.313}/_{36} = 0.7309\ A \quad (3.39)$$

Maximum Current for 4 wheels = 0.7309*4 =2.9236 A (3.40)

Runtime = Battery Capacity/Maximum Current = 8.8/2.9236 = 3 Hrs = 180 min

From Hoverboard Data Sheet We have some Constraints:

Table 4-5:Hoverboard Datasheet Constraints

Rated Speed	Rated Power	Rated Torque
800	350	4.1
RPM	Watt	N.m

So, At Worst Case

Required Tractive Torque 1.8 N.m < 4.1 N.m

Required Motor Speed 348 rpm < 800 rpm

Then, We are in the Safe Zone.

4.3 Production

The prototype and the final design were the two separate stages of the production process. Every stage required meticulous preparation and perfect execution.

4.3.1 Manufacturing & Assembly Prototype

Product Type: MDF

Design Program: SolidWorks

Production Method:

Because SolidWorks was used to design the robot's components, precise measurements and smooth assembly were guaranteed. To fabricate components precisely and consistently, CNC laser cutting was used.

Figure 4.5: Prototype Assembly

Figure 4.6: SolidWorks MDF Body Design

Figure 4.4:MDF Body

4.3.2 Complete Assembly & Manufacturing

Body and chassis were made of laser-cut steel plates, which guaranteed accuracy and uniformity.

A strong, long-lasting structure that could support operational loads was created by applying welding techniques.

Brackets: specially made brackets for hoverboard motors were put in place, guaranteeing a stable motor attachment and chassis alignment. The final product satisfied all structural and functional criteria thanks to the incremental improvements made possible by this methodical production process.

Figure 4.8:Final Steel Body Design

Figure 4.7:Final Steel Body

4.4 Stress Evaluation

Comprehensive stress analysis, displacement analysis, and factor of safety evaluations were used to confirm the robot's integrity.

4.4.1 Stress Analysis

Goal: Evaluate the robot's resilience to operational stressors

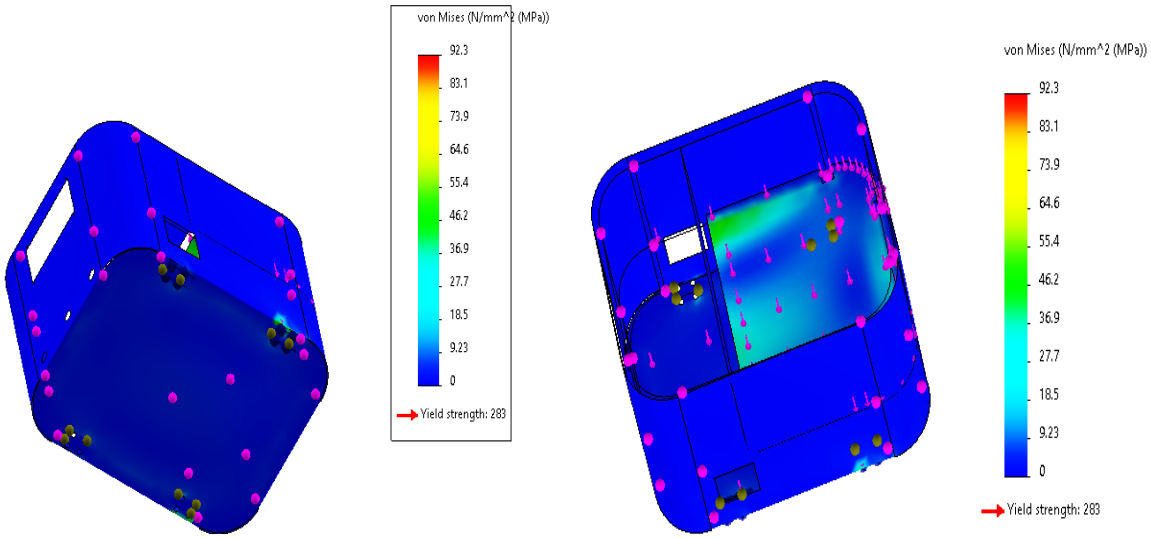


Figure 4.9: Body Stress Analysis

Findings: The steel structure's stress levels stayed far below its yield strength, demonstrating the durability of the design.

4.4.2 Displacement Analysis

Goal: To guarantee stability and structural integrity, measure deformation under load

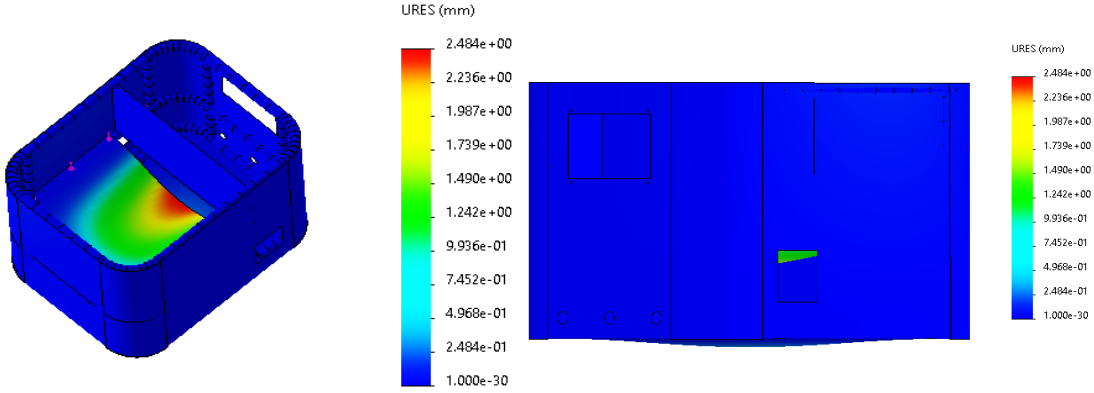


Figure 4.10: Body Displacement Analysis

Results: The robot's ability to retain its shape and operation under a variety of circumstances was confirmed by the minimal displacement that was seen.

4.4.3 Factor of Safety

Goal: Offer a safety buffer to take unforeseen loads or impacts into consideration

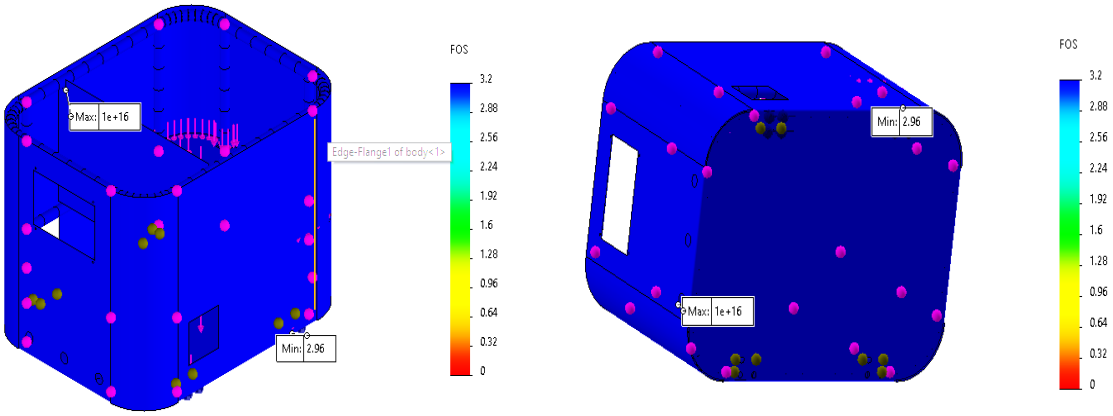


Figure 4.11: Factor of Safety

Result: A FoS of more than 1.5 was attained, demonstrating the structure's dependability and suitability for practical uses.

4.5 Conclusion

The autonomous delivery robot's mechanical design was a thorough process that included cutting-edge equipment, creative engineering, and careful planning. Every stage, from preliminary study and prototyping to final manufacture and stress analysis, was planned to guarantee a product that is sturdy, dependable, and practical. A delivery robot that not only satisfies practical needs but also establishes a standard for design excellence for next projects is the product of this meticulous approach.

Chapter (5): Hardware Methodology

5.1 Motor

5.1.1 Overview

In order to provide smooth and effective mobility, hoverboard motors are essential parts of the device. Since hoverboards first appeared, these motors have undergone substantial development, progressing from simple designs to complex, highly efficient ones. The brushless DC motor (BLDC), which is the principal motor type found in hoverboards, is superior than brushed motors in terms of performance, lifespan, and maintenance.

Figure 5.1:Hoverboard BLDC Motor

5.1.2 Principle of Operation

A stator and a rotor are common components of hoverboard motors. The rotor, which spins, has magnets, whereas the stator, which is fixed, has windings. With this arrangement, electrical energy may be efficiently transformed into mechanical energy.

In order to provide a greater range and prolong battery life, hoverboard motor efficiency is essential. Brushless DC motors minimize energy loss through heat and are renowned for their high efficiency, which can surpass 85%.

Electronically switching the motor's phases is how BLDC motors work. The rotor spins as a result of the rotor windings' sequential energization, which produces a revolving magnetic

field that interacts with the rotor magnets. Because BLDC motors don't have actual brushes to wear out as brushed motors do, they are more robust and effective.

The motor controller controls the commutation process in BLDC motors, alternating the current between various windings according to the position of the rotor. Smooth operation is made possible by this exact control.

BLDC Can Operate in two main modes:

- **Sensor-Based Commutation:** This mode detects the rotor's location using Hall sensors and modifies the current accordingly. This offers precise and dependable control, particularly at reduced speeds.
- **Sensorless Commutation:** This mode uses back EMF (Electromotive Force) feedback from the motor windings to allow the motor controller to estimate the rotor position. By eliminating the requirement for sensors, this technique lowers expenses and streamlines the motor design. Because the back EMF impulses are weaker at very low speeds, it may be less effective.

Hall-Effect Sensor

Because they can sense the rotor's location, Hall sensors are essential to BLDC motors. To ensure smooth and effective motor running, the motor controller uses this information to precisely time the stator windings' energizing. The controller receives data from the sensors and modifies the power output to ensure steady motor performance.

5.1.3 Motor Pins

- **Phase Pins:** These are the motor phase wires (A, B, C), connected to the driver. They carry the modulated power signals that drive the motor.
- **Hall-Effect Sensor Pins:**
  - **VCC (Power)**: Provides power to the Hall sensors.
  - **Ground**: Common ground for the sensors.
  - **H1, H2, H3**: Outputs from the Hall sensors that provide the rotor position feedback to the driver.

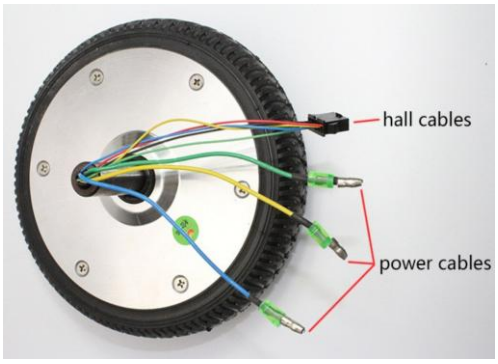


Figure 5.2: Motor Pins

5.2 Motor Driver

5.2.1 Overview

The motor driver, which controls the movement of the ADR, is one of the most significant and intricate challenges in our project.because it uses commands sent by a micro-controller to drive the motors on the wheels.In order to do this sending and receiving, it is composed of mosfets, regulators, dc connections, and input and output pins.by understanding the motor driver's structure and design in order to comprehend how it functions in our project.

To regulate motor , a complicated circuit board with several components is called a motor driver. The main elements consist of:

Capacitors: Used to stabilize and filter voltage supplies.

The microcontroller serves as the motor driver's brain, analyzing inputs and communicating the proper signals to the motors.

MOSFETs and diodes, which manage the high currents needed by the motors, are examples of power components.

In order to support the dense arrangement of components and guarantee effective signal routing, the board is constructed with numerous layers.

5.2.2 Pinout & Connections

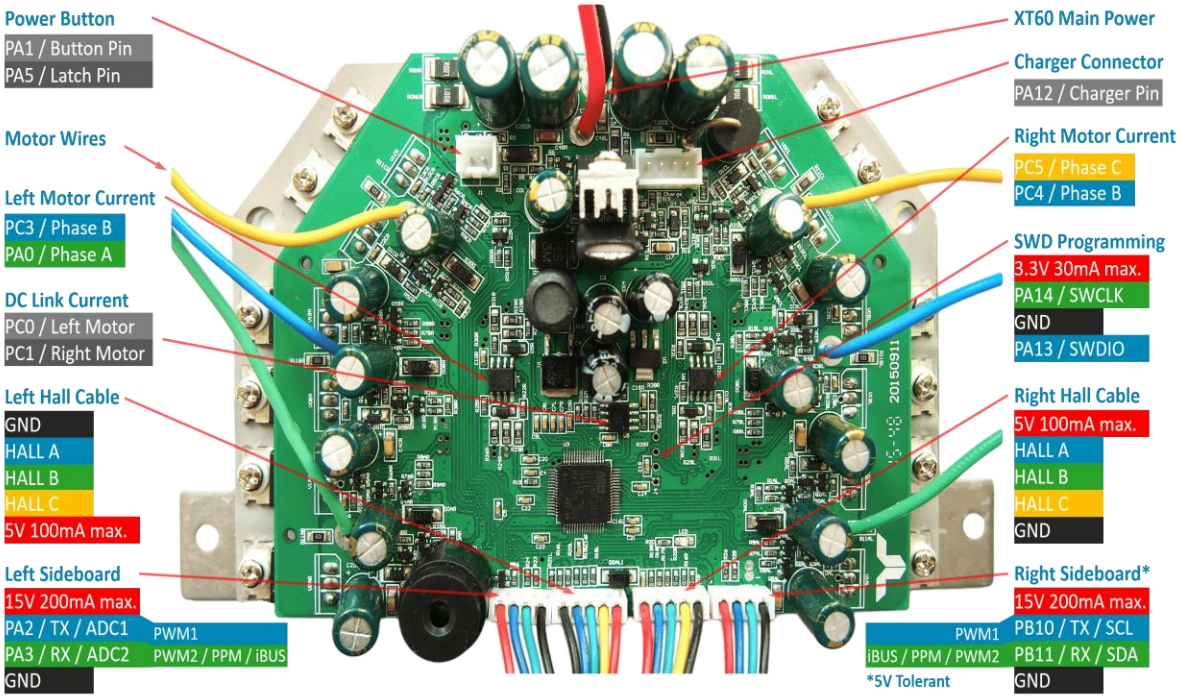


Figure 5.3: Motor Driver Pinout & Connections

The motor driver board's pinout is meticulously designed to support various functionalities. Here is a detailed explanation of each pin and its role:

- **Power Button** PA1 / Button Pin PA5 / Latch Pin
- **Motor Wires**
  - **Left Motor Current** PC3 / Phase B PA0 / Phase A
  - **Right Motor Current** PC5 / Phase C PC4 / Phase B
  - **DC Link Current** PC0 / Left Motor PC1 / Right Motor

- **Hall Sensors for Motor Positioning Feedback**
    - **Left & Right Hall Cables Each has**

      GND Hall A Hall B Hall C 5V 100mA max.

- **Sideboard Connectors**
    - **Left** PA2 / TX / ADC1 PA3 / RX / ADC2 PWM1 PWM2 / PPM / iBUS GND 15V 200mA max.
    - **Right** PB10 / TX / SCL PB11 / RX / SDA GND 15V 200mA max. (5V Tolerant)

- **Other**
    - **SWD Programming Pins** 3.3V 20mA max. PA14 / SWCLK PA13 / SWDIO GND
    - **XT60 Main Power Connector:** Supplies power to the board
    - **Charger Connectors:** PA12 / Charger Pin

5.2.3 System Integration

With the rest of the autonomous delivery robot system, the motor driver works in unison. Through certain pins, it is connected to the motors, Hall sensors, and the micro-controller, which serves as the central control unit. More reliable and accurate control is made possible by the driver's use of RX/TX communication for motor control rather than conventional PWM signals. Power efficiency, accurate motor control, and seamless operation are guaranteed by this integration.

5.3 Sensors

5.3.1 IMU

5.3.1.1 Role in Project

1. **Navigation:**

The IMU estimates the robot's current position and orientation by integrating acceleration and angular velocity data. This ensures stable movement and accurate trajectory execution, even in environments lacking external references like GPS signals.

2. **Stability:**

Provides real-time feedback to correct deviations in the robot's movement. For example, if the robot tilts or veers off course, motor speeds are adjusted to stabilize and return to the correct path [12].

3. **Sensor Fusion:**

IMU data is fused with inputs from GPS and motor feedback using complementary or Kalman filters [13]. This process enhances accuracy and reliability by mitigating individual sensor limitations.

5.3.1.2 Selection Criteria

Table 5-1: IMU Selection

Criterion	MPU6050	BNO055	LSM9DS1
Range of Applications	4	5	4
Accuracy	3	5	4
Ease of Integration	4	5	4
Cost	5	2	3
Availability	5	3	2
Power Consumption	4	3	4
Onboard Sensor Fusion	2	5	3
Total	**27**	**28**	**24**

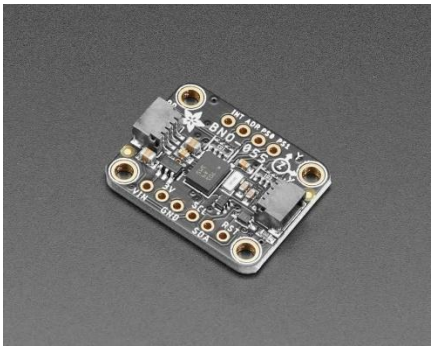


Figure 5.4: IMU BNO055

Selected IMU is BNO055

The **BNO055** was chosen initially for its onboard Sensor Fusion and accuracy. Combines a **3-axis accelerometer**, a **3-axis gyroscope and a magnetometer**. Communication through I2C ensures seamless integration with microcontrollers.

5.3.1.3 Principle of Operation

An IMU is an electronic device that measures acceleration, orientation, and angular velocity. These measurements are critical for determining the robot's position, orientation, and movement in real-time [13]. The IMU combines multiple sensors to provide a comprehensive motion profile:

Accelerometer: Measures linear acceleration along the x, y, and z axes.

Gyroscope: Measures angular velocity (rate of rotation) around the x, y, and z axes.

Magnetometer (optional): Measures magnetic fields to assist in determining orientation relative to the Earth's magnetic field.

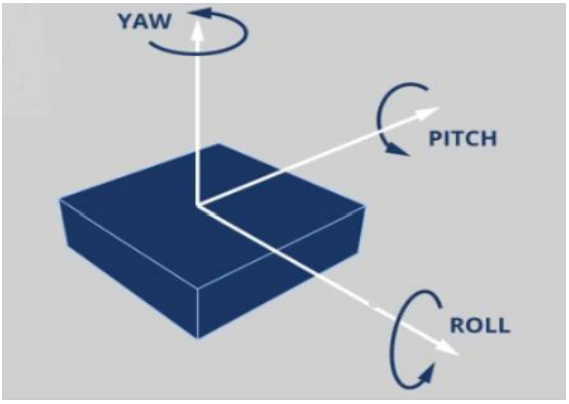


Figure 5.5: Principle of Operation of IMU

5.3.1.4 Integration Process

Communication Protocol

I2C Communication: Gained foundational knowledge of the I2C communication protocol to facilitate sensor data transmission. Prepared to implement the protocol for the BNO055 IMU sensor, which contains an accelerometer, gyroscope, and temperature sensor.

Driver Development

Implementation in Atmel Studio: Developed the IMU driver code in Atmel Studio, ensuring proper communication and data acquisition from the BNO055. Included functionalities to read accelerometer, gyroscope, and temperature data, validating the driver through systematic testing.

Testing & Calibration

Data Validation: Conducted tests to ensure the driver provided accurate and reliable readings. Fine-tuned calibration to enhance the precision of sensor readings.

5.3.1.5 Challenges

Vibration Sensitivity: Vibrations from the brushless DC motors negatively impacted IMU readings [14].

Solution: Fuse with other sensors to eliminate errors resulting from vibration.

The magnetometer can be disturbed by nearby motors, wires, or ferromagnetic materials, causing compass errors.

Solution: Mount BNO055 away from strong electromagnetic fields & Use **magnetometer calibration** routines frequently.

5.3.1.6 Expected Outcomes

Precise orientation and movement data for effective navigation & Improved robustness in dynamic and complex environments, ensuring reliable operation for the autonomous delivery robot.

5.3.2 GPS

5.3.2.1 Overview

GPS, in other word, is one of the cornerstones in the technologies of navigation and positioning of autonomous delivery robots. The location data provides highly accurate and relies on signals from network of satellites orbiting earth. This enables the robot to move through both urban and rural areas, optimize their delivery routes, and make sure deliveries are made in time.

GPS provides abase necessary for the efficient and faultless operation of autonomous systems in highly critical applications dependent on navigation, obstacle avoidance, and route optimization. the recent improvements in GPS technology, contributing to better resolution, faster signal acquisition and compatibility with multi constellation system, strongly consolidate the base upon which GPS stands within autonomous robotics. [15]

5.3.2.2 Technology

GPS was first developed in the 1970s by the US Department of Défense for military purposes. Over decades, its release for civilian application revolutionized industries including transportation to agriculture. The introduction of augmented systems, such as Differential GPS and Real-Time Kinematics, improved precision and reliability at a level of accuracy that facilitated applications requiring centimetre-level exactness. The modern GPS systems now incorporate other GNSS systems like the Russian GLONASS, the European Union`s Galileo, and China's Bei Dou to offer worldwide coverage with a robust reliability of signal and accuracy in demanding conditions like urban canyons and dense foliage [16].

5.3.2.3 Types

Various GPS modules are tailored for the unique demands of autonomous delivery robots, each offering distinct features and performance:

- **Ublox NEO-6M:** A cost-effective and widely used GPS module providing basic positioning with an accuracy of 2.5 meters, suitable for entry-level applications.
- **Ublox NEO-6MV2:** An upgraded version of the NEO-6M, offering improved sensitivity and faster signal acquisition for more reliable performance.
- **Ublox NEO-7M:** A versatile GPS module with better accuracy and faster processing than the NEO-6 series, ideal for applications requiring quicker location fixes.
- **Ublox NEO-M8N:** A high-performance multi-GNSS module that supports GPS, GLONASS, Galileo, and QZSS, offering enhanced accuracy and reliability in complex environments.
- **Ublox ZED-F9P:** A cutting-edge RTK GPS module capable of centimetre-level accuracy, designed for precision-critical applications like autonomous navigation and delivery. [16]

5.3.2.4 Key Considerations

The effective use of GPS in autonomous delivery robots depends on several factors' accuracy**, Signal** Reliability**, Power** Efficiency**,** Integration with Other Sensors and Cost

Open-access scientific research plays a critical role in driving the evolution of GPS technology. Advancements in GPS modules and systems continue to enhance performance, accuracy, and reliability, unlocking new possibilities for their use in various domains. As technology continues to evolve, we can expect even more precise, robust, and versatile GPS systems to shape the future of autonomous navigation and positioning applications.

5.3.2.5 Principle of Operation

How does GPS work with distance measurements from at least three satellites, the receiver determines its position by finding the intersection of spheres centred at each satellite with a radius equal to the calculated distance. Because one satellite only leads to infinite cloud points that a man could be in, and two intersected spheres from two satellites leads to two points of intersection that a man could be in, but at least three spheres intersecting together will end up to one point of intersection that a man could be in. This method is called trilateration and that is how GPS defines your location then to choose suitable GPS we make comparison between types.

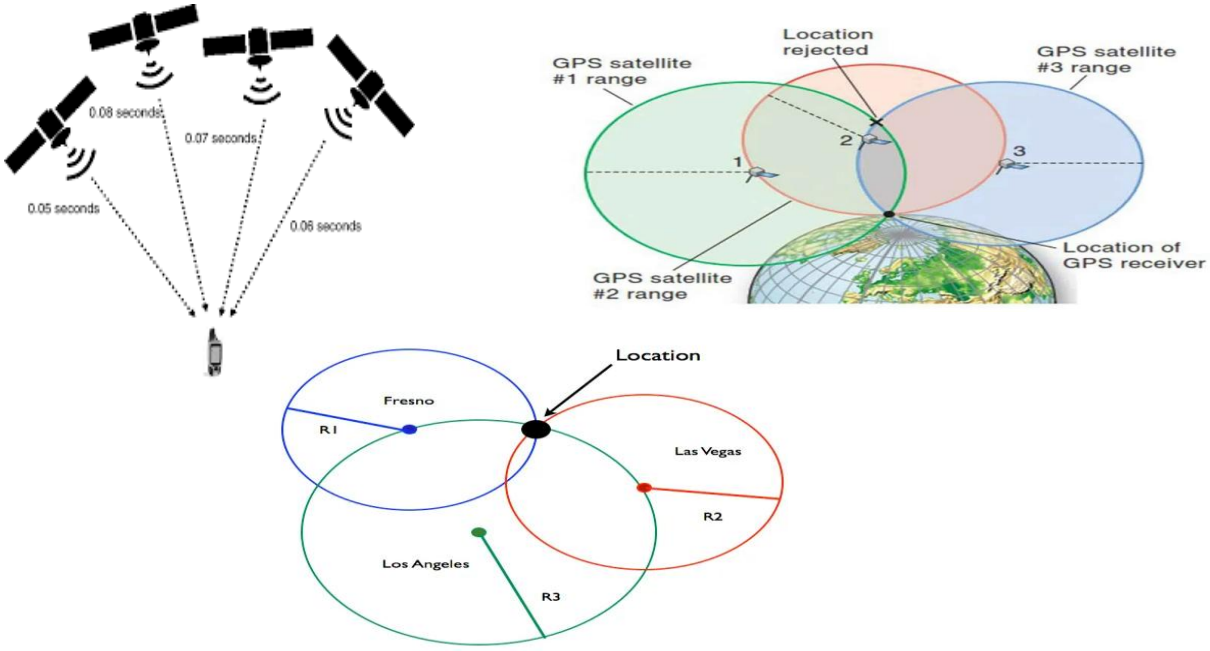


Figure 5.6: GPS Principle

5.3.2.6 Selection Criteria

Table 5-2: GPS Selection

Criterion	Ublox NEO-6M	Ublox NEO-6MV2	Ublox NEO-7M	Ublox NEO-M8N	Ublox ZED-F9P
Accuracy	3	4	4	5	5
Satellite Systems	3	3	3	4	5
Update Rate	3	3	4	4	5
Power Efficiency	3	3	3	5	4
Cold/Hot Start Time	3	3	4	5	5
Multipath Resistance	2	3	4	5	5
Durability	3	3	4	4	5
Cost-Effectiveness	4	4	3	3	2
Availability	5	1	1	1	1
Total	**29**	**27**	**30**	**36**	**37**

We must from the table of comparison choose Ublox ZED-F9P or any GPS near in ranking but we can't find them in Egypt store but we found Ublox NEO-6M so we work with it

5.3.2.7 NMEA Sentences

The National Marine Electronics Association (NMEA) has established a specification that standardizes communication between various marine electronic devices. This specification enables marine electronics to exchange information seamlessly with computers and other

marine equipment. GPS receiver communication adheres to this standard, which ensures compatibility with most computer programs that process real-time position data in the NMEA format. This format delivers a complete PVT (position, velocity, time) solution computed by the GPS receiver.

NMEA communication is based on transmitting a single line of data, referred to as a "sentence," which is self-contained and independent of other sentences. These sentences, beginning with a '$' and ending with a carriage return/line feed, are limited to 80 characters of visible text (excluding terminators). Data within the sentence is structured as ASCII text, separated by commas. While most data is fully contained within one variable-length sentence, specialized scenarios may allow the information to span multiple sentences. Additionally, standard sentences exist for device categories, and proprietary sentences can be defined for specific company use. Precision levels of the data vary depending on the specific message content.

The most important NMEA sentences include the GGA which provides the current Fix data, the RMC which provides the minimum GPS sentences information, and the GSA which provides the Satellite status data.

GGA - essential fix data which provide 3D location and accuracy data.

```
 $GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Where:
     GGA          Global Positioning System Fix Data
     123519       Fix taken at 12:35:19 UTC
     4807.038,N   Latitude 48 deg 07.038' N
     01131.000,E  Longitude 11 deg 31.000' E
     1            Fix quality: 0 = invalid
                               1 = GPS fix (SPS)
                               2 = DGPS fix
                               3 = PPS fix
                               4 = Real Time Kinematic
                               5 = Float RTK
                               6 = estimated (dead reckoning) (2.3 feature)
                               7 = Manual input mode
                               8 = Simulation mode
     08           Number of satellites being tracked
     0.9          Horizontal dilution of position
     545.4,M      Altitude, Meters, above mean sea level
     46.9,M       Height of geoid (mean sea level) above WGS84
                      ellipsoid
     (empty field) time in seconds since last DGPS update
     (empty field) DGPS station ID number
     *47          the checksum data, always begins with *
```

5.3.3 LiDAR

5.3.3.1 Overview

LiDAR (Light Detection and Ranging) can undoubtedly be considered one of the most crucial components in our project, if not the most critical. Think of it as the eyes or ears of our system—the key element responsible for perceiving the environment with unparalleled accuracy. It surpasses any other sensor in our system in terms of environmental perception. To truly appreciate the significance of LiDAR, we need to take a step back and explore the concept of perception in autonomous robots. Understanding this foundational element will illuminate why LiDAR plays such an indispensable role in our project.

The word "perception" comes from the Latin word "perceptio" (genitive "perceptionis"), which means "perception, apprehension, a taking or receiving [17]. The term made its way into Old French as "percepcion," which retained the meaning related to the act of perceiving or the process of perceiving something. In modern day robotics, the word refers to the ability of robots to interpret and understand their environment through the use of various sensors

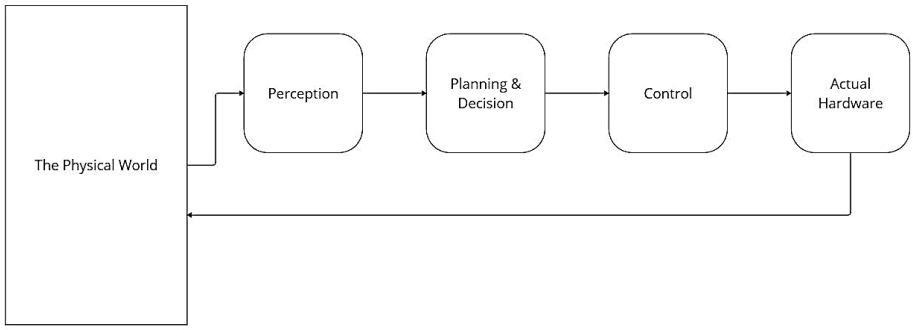


Figure 5.7: Process Diagram

and technologies. It's a crucial aspect of robotics, enabling robots to interact intelligently with their surroundings [18]

As illustrated in the diagram (Figure 4.3), perception serves as the foundational stage upon which all subsequent software processes rely. It acts as the critical initial step, gathering and interpreting sensory data to inform and guide the operations that follow. Without accurate and efficient perception, the entire system's performance would be compromised, highlighting its indispensable role in the overall architecture.

Stepping forward to the role of our LiDAR, it's evident that accurate and efficient perception cannot be achieved without an equally precise and reliable LiDAR system.

5.3.3.2 Technology

This sophisticated sensor operates by emitting laser beams in a single plane—it is important to note that we are exclusively discussing 2D LiDARs here—to measure distances to objects in its path. The laser beams reflect off objects and return to the sensor, enabling it to calculate distances based on the time taken for the light to travel to and from the object. This data is subsequently processed to create a comprehensive 2D map of the environment, which is crucial for obstacle detection, navigation, and overall environmental awareness.

The simplified diagram above (Figure 4.4) illustrates how the 2D LiDAR functions. The position of an object's point pp in Cartesian coordinates x_p and y_p, relative to the 2D LiDAR's position and orientation, can be determined using the following formula [19]:

$$x_p = r_p \cdot \sin(\theta_p),$$

$$y_p = r_p \cdot \cos(\theta_p)$$

where r_p and θ_p are the distance and the angle from the 2D LiDAR to the point p, respectively.

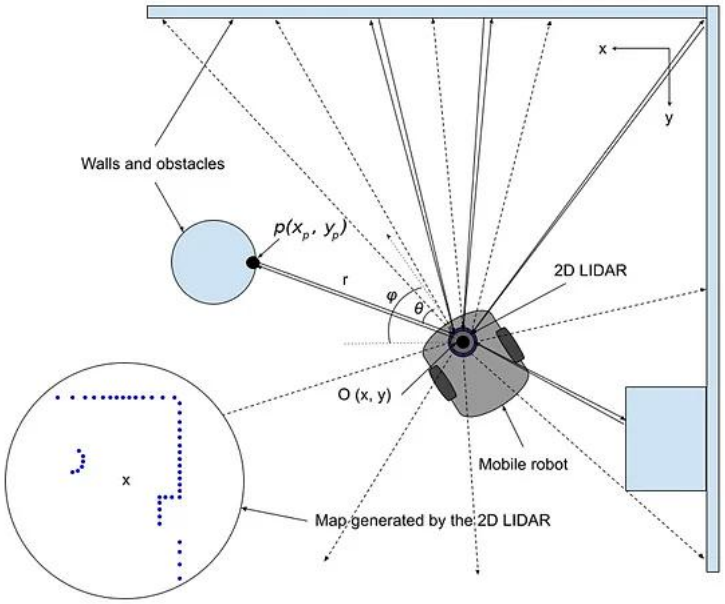


Figure 5.8: LiDAR 2D Generate A map

5.3.3.3 Selection Criteria

Table 5-3: LiDAR Selection

Criterion	YDLiDAR X2	Okdo LiDAR	RPLiDAR A1
Accuracy and Precision	4	2	4
Range and Field of View (FOV)	3	3	3

Criterion	YDLiDAR X2	Okdo LiDAR	RPLiDAR A1
Point Density and Scan Speed	3	2	2
Outdoor Usage	3	2	4
Portability and Ease of Use	4	2	2
Development and support	3	2	3
Availability	4	3	2
Cost	4	4	1
Total	**28**	**20**	**21**

5.3.3.4 Hardware

The LiDAR that is used in this project is the YDLIDAR X2, which will be referred to as X2. Based on the principle of Triangulation, this advanced sensor is capable of rotating 360° degrees, allowing it to continuously output angle information and point cloud data of the surrounding environment [20].

To facilitate secondary development, the X2 internally defines a polar coordinate system. In this system, the center of the rotating core of the X2 serves as the pole, with angles measured in a clockwise direction. The zero-degree angle is positioned directly in front of the X2 motor. However, due to individual manufacturing differences, there is a potential deviation of plus or minus 3 degrees, as depicted in the accompanying (Figure 4.5).

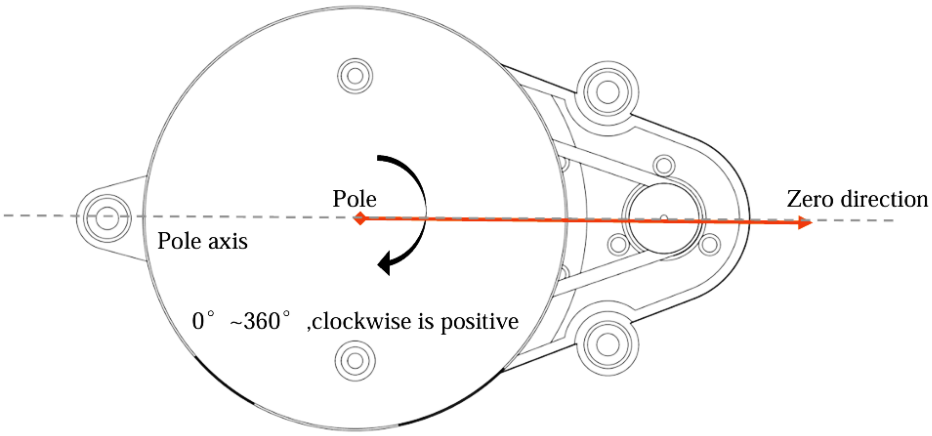


Figure 5.9:LiDAR Hardware

5.3.4 Ultrasonic

5.3.4.1 Role in the Project

- Obstacle Detection: Ultrasonic sensors provide real-time distance measurements to detect obstacles and avoid collisions beneath the 2D LiDAR.
- Navigation Assistance: Ultrasonic readings complement data from other sensors like IMUs and motor encoders for fine-tuning the robot's position and path adjustments.

5.3.4.2 Principle of Operation

Ultrasonic sensors work on the principle of SONAR (Sound Navigation and Ranging). They emit high-frequency sound waves (ultrasound) that bounce off objects and return as echoes. By measuring the time it takes for the echo to return, the sensor calculates the distance.

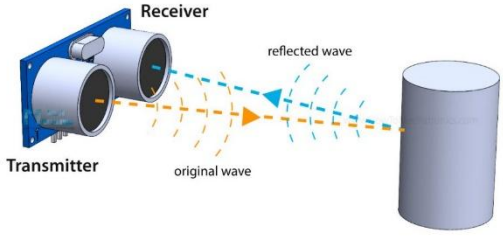


Figure 5.10: Ultrasonic Receiving & Transmitting

5.3.4.3 Selection Criteria

Table 5-4: Ultrasonic Selection

Criterion	HC-SR04	URM37	SRF05
Range	4	5	4

Criterion	HC-SR04	URM37	SRF05
Accuracy	3	5	4
Ease of Integration	5	4	4
Cost	5	2	3
Availabiltiy	5	3	2
Power Consumption	4	4	4
Reliability in Use	4	5	5
Total	**30**	**28**	**26**

HC-SR04 Module Features is:

Distance Range: Measures distances between 2 cm and 400 cm with an accuracy of ±2 cm.

Components: Includes a transmitter, receiver, and control circuit.

Figure 5.11: HC-SR04 Ultrasonic Module

5.3.4.4 Challenges

Range and Noise Sensitivity: Ultrasonic sensors can struggle with detecting objects at their maximum range or in noisy environments with sound reflections [21].

Solution: Optimized placement of sensors and implemented filtering techniques to improve reliability and minimize false readings.

Accuracy: Variations in sensor readings due to environmental factors, such as surface texture or irregular shapes, can affect accuracy.

Solution: Calibration was performed regularly to ensure consistency, and data from multiple sensors were cross-validated to improve accuracy.

5.3.4.5 Expected Outcome

Improved responsiveness to environmental changes, enabling the robot to navigate dynamic urban scenarios effectively. Enhanced safety and precision, allowing the robot to operate autonomously with reduced risks of collisions.

5.4 Power Supply

5.4.1 Battery

5.4.1.1 Overview

Batteries are the lifeblood of modern autonomous systems, providing the energy required for seamless operation across various applications. In autonomous delivery robots, batteries play a pivotal role in enabling mobility, control, and communication systems. The choice and management of the battery system significantly influence the robot's range, efficiency, and reliability.

As robotic technology has advanced, battery systems have evolved in parallel, enabling longer operational times, increased payload capacities, and improved reliability [22].

5.4.1.2 Evolution of Batteries

Early robotic systems relied on lead-acid batteries, which offered limited energy density and short operational times. However, the development of lithium-ion (Li-ion) batteries in the late 20th century marked a turning point. Li-ion batteries provide significantly higher energy density, lighter weight, and longer lifespans compared to their predecessors. These advancements enabled critical improvements, such as extended operational durations, support for heavier payloads, and expanded ranges [23].

5.4.1.3 Types of Batteries

Several battery chemistries have emerged to meet the varying demands of autonomous delivery robots:

Lead-Acid Batteries: Among the oldest and most established types, lead-acid batteries are cost-effective and reliable for applications with lower energy density requirements. However,

their heavy weight and shorter lifespan make them less suitable for advanced autonomous robots.

Lithium Polymer (LiPo): Known for high energy density and lightweight construction, LiPo batteries are widely used in delivery robots. However, they require careful handling and are sensitive to temperature variations.

Selection Criteria for Battery Systems

The performance of an autonomous delivery robot is heavily dependent on its battery system. Key considerations when selecting batteries include: [24]

Energy Density: Higher energy density supports longer operational times and heavier payloads.

Weight: Lightweight batteries enhance mobility and energy efficiency.

Discharge Rate: Batteries must provide adequate current for motors and electronic systems.

Lifespan: Extended lifespan reduces replacement costs and improves sustainability.

Safety: Battery chemistry and design should minimize risks of accidents.

Cost: Budget considerations play a significant role in battery selection.

Advancing battery technology in these robots opened the way to really revolutionary abilities for autonomous last mile delivery: travelling , longer , heavier and working more efficiently. Further.

5.4.1.4 Selection Criteria

We need in our side walk autonomous delivery robot a source of power that we decided it will be electric battery then we start to search for the suitable type that can afford our specifies that we need then we compare between the Lithium Polymer (LiPo)and Lead-Acid Batteries and Li-ion (Lithium-ion).

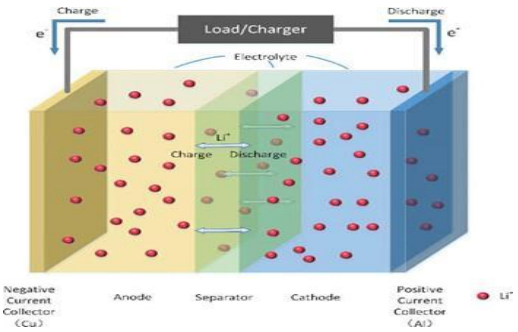


Figure 5.13: Lithium-ion Battery

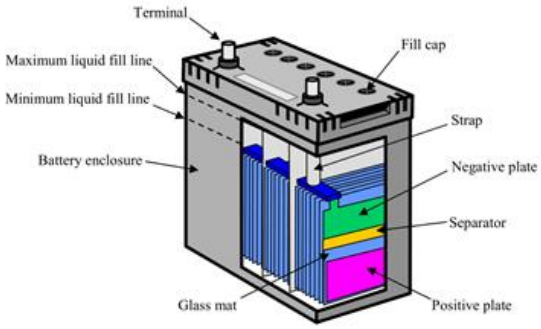


Figure 5.12: Lead Acid Battery

Table 5-5: Battery Selection

Criterion	Lead Acid	Li-ion	LiPo
Energy Density	1	3	5
Weight	1	4	4
Lifespan	2	3	5
Charge Time	1	5	4
Cost	5	3	3
Discharge Rate	2	4	5
Total	**12**	**22**	**26**

Table 5-6: List of the component power

component	No. component	current	voltage	power
Raspberry pi 4	1	3A	5V	15W
IMU	1	370mA	3.4V	1.25W
ultrasonic	3	8mA	5V	0.12W
Lidar	1	500mA	5V	2.5W
monitor	1	160mA	9.6V	1.536W
GPS	1	45mA	5V	2.25W
FAN	2	200mA	12V	4.8W
Arduino mega	1	500mA	5V	2.5W
motor	4	3A	36V	432W
Arduino Uno	1	500mA	5V	2.5W
ESP32	1	800mA	5V	4W
Flash	2	200mA	12V	4.8W
Total Power = 473W				

Then total drain current = 18.683 AH

We need our ADV operates for only an 30 min and we found that the maximum cost effective and low mass battery available is of 4.4Ah. Where the batteries are 36V and 4.4h

So, after take the total drain watt and watt that our batteries supply =316wh/473wh =40min

Charging time: as our charger give 42V and 2A that total power equal 84w in hour then our time of charge would equal total power /power of charger=384/84=4.6 h

5.4.1.5 Conclusion

The LiPo (Lithium Polymer) configuration shines as a versatile choice for various autonomous delivery robot applications, particularly those prioritizing weight, maneuverability, and cost-effectiveness. However, the power spectrum extends beyond this single star. Understanding the trade-offs between runtime, weight, cost, and additional factors empowers you to choose the power source that fuels your robot's unique aspirations. Let research continue to illuminate new technologies, paving the way for longer operation times, lighter payloads, and even more ground-breaking possibilities for these autonomous delivery systems.

5.4.2 Choose Battery

When powering a device or system we face the decision:
Should we combine two smaller batteries in parallel, or just buy a single larger battery with the same voltage and capacity?

5.4.2.1 Option 1: Two Batteries in Parallel

Parallel connection means connecting positive to positive and negative to negative:

1.Voltage stays the same

2.Capacity (Ah) adds up

3.Extended Current Capacity**:** Each battery shares the current load. If one battery can supply 5A, two in parallel can supply up to 10A, reducing the strain on each.

4. Lower Heat & Stress: Since each battery provides only part of the total current, heat generation and wear are reduced. This increases battery life.

5. Voltage Stability: Parallel batteries help maintain a more stable voltage under heavy load conditions, especially when one battery starts to drop voltage.

6. Redundancy and Safety: If one battery fails or disconnects, the other can still provide power (though reduced). This increases reliability.

7. Modular Expansion: You can upgrade or replace just one battery instead of buying a larger new one. Useful in modular designs.

8. Easier to Source and Fit: Two smaller batteries are often easier to buy and place than one large battery, especially in tight spaces.

5.4.2.2 Option 2: One Larger Battery

You buy a single battery that already has the desired voltage and total capacity.

1. Higher initial cost
2. Simpler wiring: Fewer wires, no need to balance between batteries easier and cleaner setup
3. Easier Battery Management: no need to match batteries or manage uneven charging/discharging. No risk of imbalance.
4. No Imbalance Risk: In parallel setups, if batteries aren't matched, current may flow unevenly single battery avoids this problem.

Table 5-7: Comparison between Options

Feature	Two Parallel Batteries	One Large Battery
Wiring	More complex	Simpler
Redundancy	Safer(can keep working if one fails)	One fails = total shutdown
Voltage Drop	Lower under high current	Good, but depends on size
Charging	More flexible	Easier to manage
Heat	Lower per cell	Depends on load
Cost	Often cheaper	Often more expensive
System Complexity	Higher (balance circuits)	Lower

After all of that we use the parallel connection of the batteries of 36v 4.4AH to support 8.8AH

5.4.3 Battery Monitoring

When using parallel connection or large battery for powering device or system we face the decision:

we must use Battery Management System (BMS) is an electronic circuit that monitors, protects, and controls rechargeable batteries (especially lithium-based types like Li-ion, $LiFePO_4$, etc.). It ensures the battery operates safely, efficiently, and reliably. or use different way

Main Functions of a BMS : Overcharge Protection, Over discharge Protection , Overcurrent Protection, Temperature Monitoring, Cell Balancing, Short Circuit Protection, State of Charge (SOC) and Communication Interface ,if we don't use BMS or different way to do same function you risk: Overheating and fire, Battery failure, Shortened battery life ,Unbalanced cells (one cell dies early, system fails)

Types of BMS:

1. Passive Balancing
2. Active Balancing
3. Smart BMS

Table 5-8: Comparison Between Types

Feature / Type	Passive BMS	Active BMS	Smart BMS
Function	Discharges excess charge as heat	Transfers charge between cells	Adds monitoring, data logging, control
Cell Balancing	Yes, via resistors (wastes energy)	Yes, via energy transfer circuits	Yes (active/passive)
Efficiency	Low (energy wasted as heat)	High (energy redistributed)	High (depends on type)
Cost	moderate	Higher	Medium to High

Communication	None	Rarely	Yes (UART, I2C, CAN, Bluetooth)
Data Logging	No	No	Yes (voltage, temp, SOC, etc.)
Protection Features	Basic (over/under voltage, short)	Basic	Full protection + smart features
Used For	DIY e-bikes, tools, power banks	EVs, high-end robotics, energy storage	EVs, drones, robots, smart battery packs
Availability in Egypt	no	yes	no

After the table we choose smart but not available in Egypt then we search for active BMS but was so expensive then we search for another way to make our robot more commerical

5.4.4 Alternative for BMS

We start search for options we find some interesting alternative like Voltage Divider + Arduino, Balancing Boards, CCCV Charger + Fuse, Poly fuse + Diode Setup, Smart Charger with Cell Monitoring

We start make comparison between them in the safety and protection

Table 5-9: Alternatives Study

Alternative	Protection cons
Voltage Divider + Arduino	No protection — just monitoring
Balancing Boards	No protection (you still need fuse)
, CCCV Charger + Fuse	No balancing or real monitoring

Poly fuse + Diode Setup	No voltage control or balancing
Smart Charger with Cell Monitoring	Only protects during charging

After the comparison we see no alternative is suitable for our project then we think of hypered alternative to can use as BMS and cheap price that alternative is using Voltage Divider + Arduino to know the voltage, the state of charge, using active balancing cells way to over come the over charge and less charge and using the diodes to block the reverse current and the 2batteries from charge each other if one discharge faster than the other , using the MOSFET as switch to can control the charging and the active balancing cell.

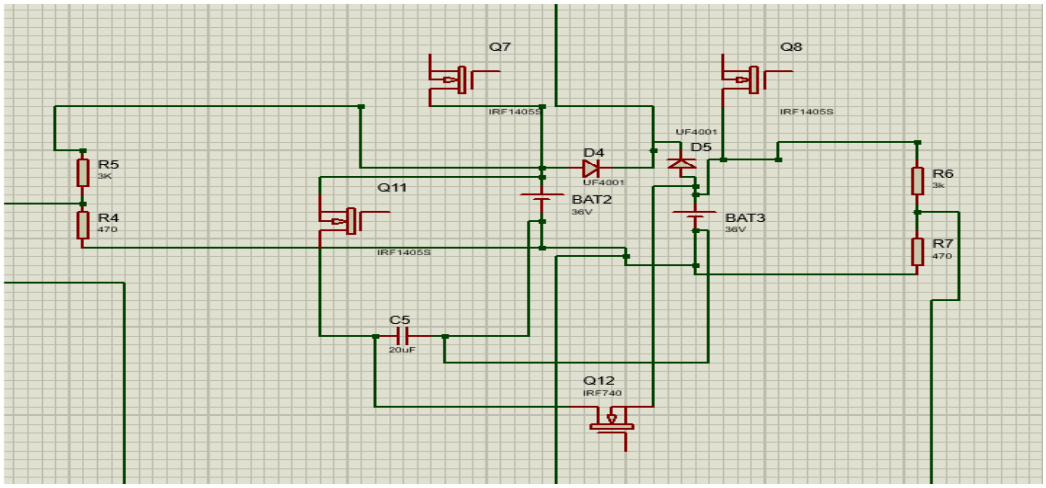


Figure 5.14: BMS Alternative

5.4.5 Design Circuit

In the design of the circuit, we start by taking some step or stage to can make the design.

Stage 1: We list the component from the software design, then list the component of the high voltage, low voltage, high current and low current.

Stage 2: After the listing step, then the step of the data sheet to read to know the voltage of breakdown, max current that can drain in the different mode and the feature, usage of it to can use the component in the best performance of it.

Stage 3: start to divide the circuit into small circuit according to there function and the voltage and then make all the circuit by calculate the needed for it to make it real and test it by the simulation.

Stage 4: after make small circuit and test it then collect the small circuit make the bigger on to the full circuit and that the stage we use in our project.

5.4.5.1 Overview of the Circuit

After the battery circuit design we by we need to isolate the motor drive, the micro controller and and low voltage device that we start by power the motor drive by parallel connection and then the third wire of the 3wire of the parallel connection to dc-dc buck 12 volt to power the fan and the flash by 3 wire parallel connection and the third wire of dc-dc converter to power the 5 v volt dc-dc converter to power the Arduino uno and the mega and the raspberry pi4 then we the micro controller to power the monitor and the rest sensor by open the data sheet of the micro controller to confirm the max to can drain from micro controller to power the sensor then calculate the power of the sensor to divide across the micro controller according to the datasheet.

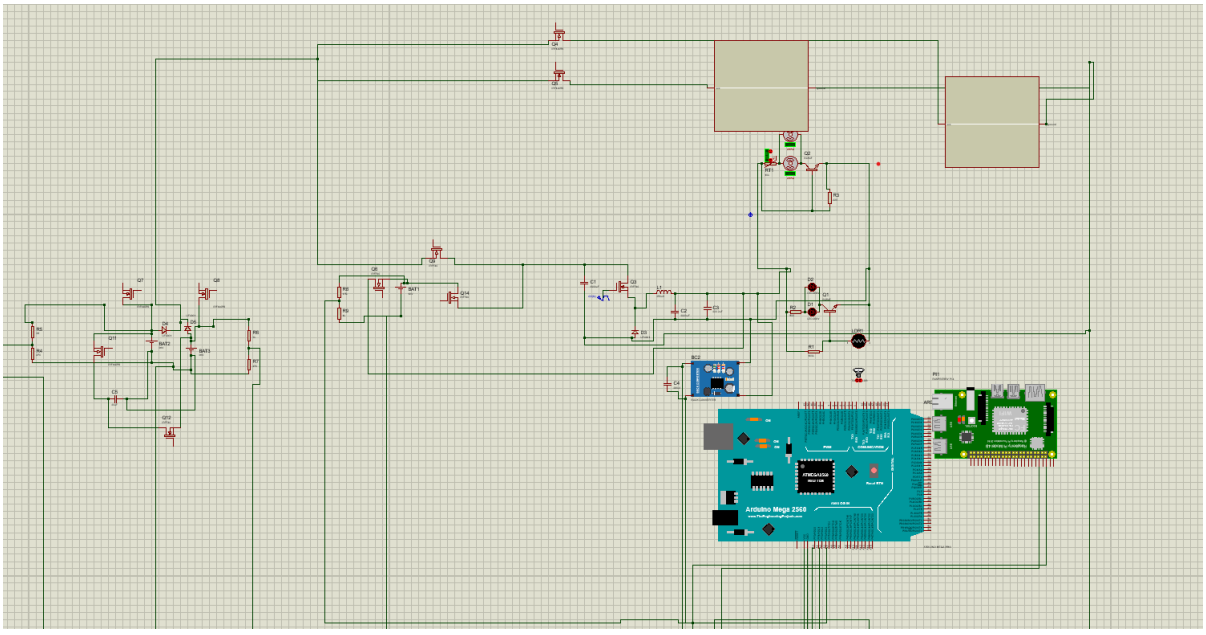


Figure 5.15: power distribution and control wiring

Then problem face us that stage of charging the circuit will cut of the Arduino that control the MOSFET then we use another battery as backup for the line of the 12 volt and by connect with another MOSFET as switch to open it after stage of charging like it shown in the figure(5.15).

5.5 Microcontroller

5.5.1 Overview

The microcontrollers are the intelligences that lie behind the operations of autonomous delivery robots which controlthecoordination and operation of different components and subsystems for efficient performance and reliability. These integrated circuits are small, containing a processor, memory, and I/O interfaces that can execute instructions, process data, and interface with sensors, actuators, and communication modules by their construction.

The choice of microcontroller significantly influences the functionality of the robot, which includes its processing capability, energy efficiency, and adaptability to real-world challenges. In autonomous robots, microcontrollers have been entrusted with vital tasks such as navigation, obstacle detection, motor control, and wireless communication. [24]

5.5.2 Evolution of Microcontrollers

The history of microcontrollers began with the development of the first 4-bit microcontroller in the early 1970s. These early devices had limited processing power and memory. However, advancements in semiconductor technology led to the creation of 8-bit and 16-bit microcontrollers, providing enhanced capabilities for embedded systems.

Today, 32-bit microcontrollers dominate robotics applications due to their superior processing power, high-speed peripherals, and ability to handle complex tasks. Innovations in power management and miniaturization have further optimized microcontrollers for energy-efficient operation, making them ideal for battery-powered autonomous systems.

5.5.3 Types of Microcontrollers

Several microcontrollers and microcomputers are widely used in autonomous delivery robots, each suited for different levels of complexity and functionality:

- **Arduino Mega 2560:** Ideal for projects requiring multiple input/output connections and extensive peripheral interfacing, thanks to its 54 digital I/O pins and robust community support.
- **Arduino Uno:** A versatile and beginner-friendly option, perfect for simpler robotic systems requiring straightforward programming and smaller memory.
- **Raspberry Pi 4 (8GB):** A powerful microcomputer capable of running full operating systems, suitable for advanced applications like real-time video processing, machine learning inference, and high-speed data handling.
- **STM32F4 Series:** Renowned for their high-performance ARM Cortex-M cores, these microcontrollers are ideal for real-time control and tasks demanding significant computational power.
- **ESP32:** A powerful dual-core microcontroller with built-in Wi-Fi and Bluetooth, ideal for applications requiring real-time processing, wireless communication, and multitasking. It offers rich I/O options and supports multiple protocols making it well-suited for sensor integration.
- **Beagle Bone Black:** This microcomputer offers robust GPIO access and support for Linux, making it suitable for robotics projects requiring high-level computation and versatile interfacing options.

5.5.4 Selection Criteria

Table 5-10: Microcontroller Selection

Criterion	Arduino Mega 2560	Arduino Uno	Raspberry Pi 4 (8GB)	STM32F4	ESP32	Beagle Bone Black
Processing Power	2	1	5	3	4	4

Criterion	Arduino Mega 2560	Arduino Uno	Raspberry Pi 4 (8GB)	STM32F4	ESP32	Beagle Bone Black
Memory (RAM/Flash)	2	1	5	3	4	4
GPIO and Peripherals	5	2	4	3	4	4
Analog Inputs	5	3	3	2	3	3
Real-Time Capability	2	2	4	4	4	4
Development Tools	4	4	5	3	4	3
Expandability	4	3	5	3	4	4
Cost	3	5	2	3	4	2
Community Support	5	5	5	3	4	2
Total	**37**	**27**	**43**	**28**	**35**	**30**

5.5.4.1 Reasons to Choose Arduino Mega 2560

High Number of GPIO Pins:

Offers 54 digital I/O pins and 16 analogue input pins, making it ideal for projects requiring many sensors and peripherals.

Great for robots with multiple actuators, sensors, and hardware add-ons.

Ease of Use:

Supported by the Arduino IDE, which is beginner-friendly and requires minimal setup.

Simplifies programming and deployment for those new to embedded systems.

Extensive Community Support:

Vast online resources, tutorials, and open-source libraries.

Easily integrates with off-the-shelf modules like motor drivers and communication shields.

Real-Time Processing:

Better suited for tasks requiring deterministic responses, such as controlling motors or reading sensor data.

Avoids the latency associated with multitasking operating systems.

Cost-Effective:

Offers excellent value for hardware capabilities, making it suitable for budget-sensitive projects.

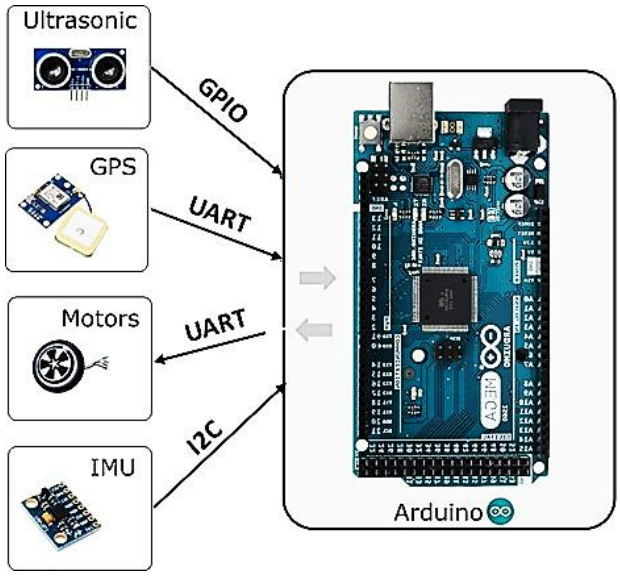


Figure 5.16: Interfacing

5.5.4.2 Reasons to Choose Raspberry Pi 4 (8GB)

High Processing Power:

Quad-core processor and up to 8GB of RAM enable advanced tasks like image processing, machine learning, and navigation.

Suitable for complex autonomous robots needing real-time data analysis and decision-making.

Figure 5.17:Raspberry Pi 4 (8GB)

Connectivity Options:

Comes with built-in Wi-Fi, Bluetooth, Ethernet, and USB ports for connecting to peripherals and the internet.

Simplifies communication with cloud systems, mobile devices, or other robots.

Supports Operating Systems:

Runs full Linux distributions, allowing the use of advanced software like Python, OpenCV, ROS (Robot Operating System), etc.

Ideal for AI-based applications and simultaneous multitasking.

Expandability:

Can connect to external monitors, cameras, and advanced sensors like LiDAR.

Includes a 40-pin GPIO header for interfacing with low-level hardware components.

Rich Software Ecosystem:

Access to libraries, drivers, and software development tools for complex robotics and AI projects.

Provides flexibility for developers accustomed to desktop-like environments.

Using both can leverage their strengths:

Arduino Mega 2560 handles real-time hardware control (e.g., motor control and sensor interfacing).

Raspberry Pi 4 performs computationally intensive tasks (e.g., object recognition, path planning).

Combining them creates a powerful and versatile platform for autonomous delivery robots, covering a wide range of functionalities.

5.5.4.3 Reasons to Choose ESP32

High Processing Power:

- Features a dual-core Tensilica Xtensa LX6 microprocessor, running at up to **240 MHz**, offering significantly higher performance than typical 8-bit microcontrollers.
- Capable of handling complex computations such as sensor fusion, real-time control, and lightweight machine learning models.

Figure 5.18: ESP32

Wireless Connectivity:

- Integrated Wi-Fi (802.11 b/g/n) and Bluetooth (Classic + BLE) eliminate the need for external modules.

Hardware Timers

- Includes up to **4** general-purpose 64-bit timers and additional peripheral timers
- Useful for precise time-based control in motor driving, encoder reading, and sensor sampling.
- Timer-based interrupts enable deterministic event handling, critical for real-time robotics.

5.6 Powertrain

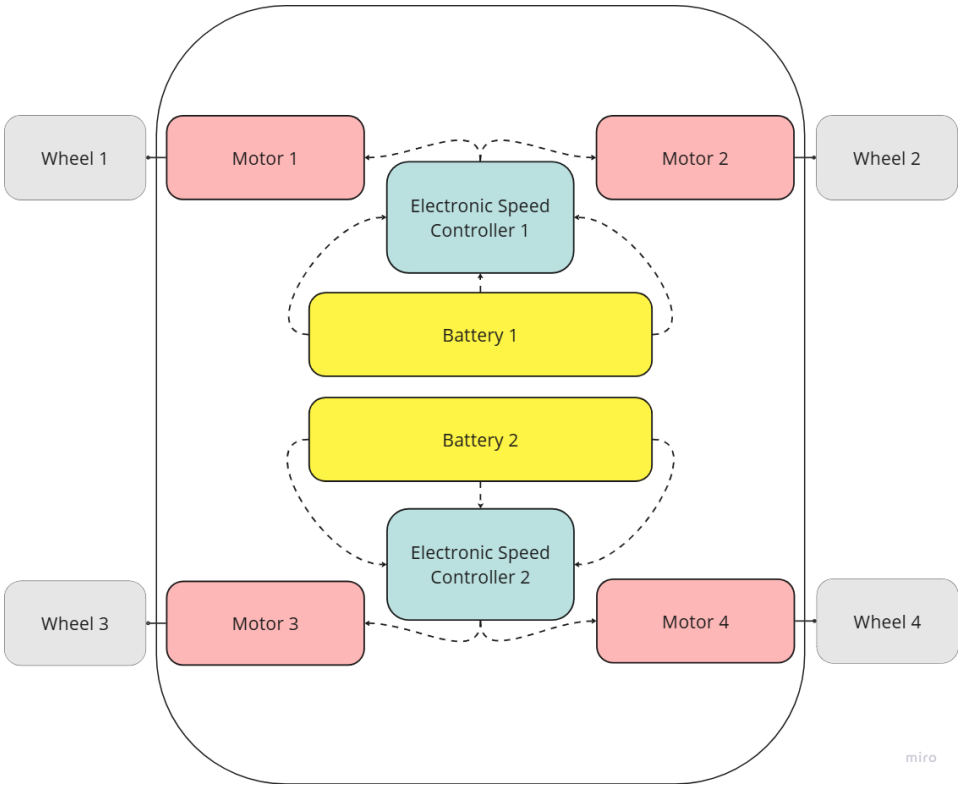


Figure 5.19: Powertrain Architecture

This (Figure4.8) represents the powertrain architecture of an autonomous delivery robot. The robot features a four-wheel-drive system, with each wheel (Wheel 1 to Wheel 4) driven by an individual motor (Motor 1 to Motor 4). This configuration allows for independent control of each wheel, providing excellent maneuverability and traction—essential for navigating various terrains in delivery scenarios. The motors are regulated by two electronic speed controllers (ESC 1 and ESC 2), which act as intermediaries between the power source and the motors. ESC 1 controls Motor 1 and Motor 2, while ESC 2 manages Motor 3 and Motor 4, ensuring efficient speed and torque control. The power for the entire system is supplied by two batteries (Battery 1 and Battery 2), likely lithium-ion batteries due to their high energy density and reliability. These batteries supply electrical energy to the ESCs, which convert it into a form suitable for powering the motors. This architecture ensures precise motion control, modular design for scalability, and redundant power supply for extended operational time. It is specifically designed to meet the demands of autonomous delivery tasks, enabling reliable and efficient performance in urban and suburban environments.

5.7 Schematic Wiring

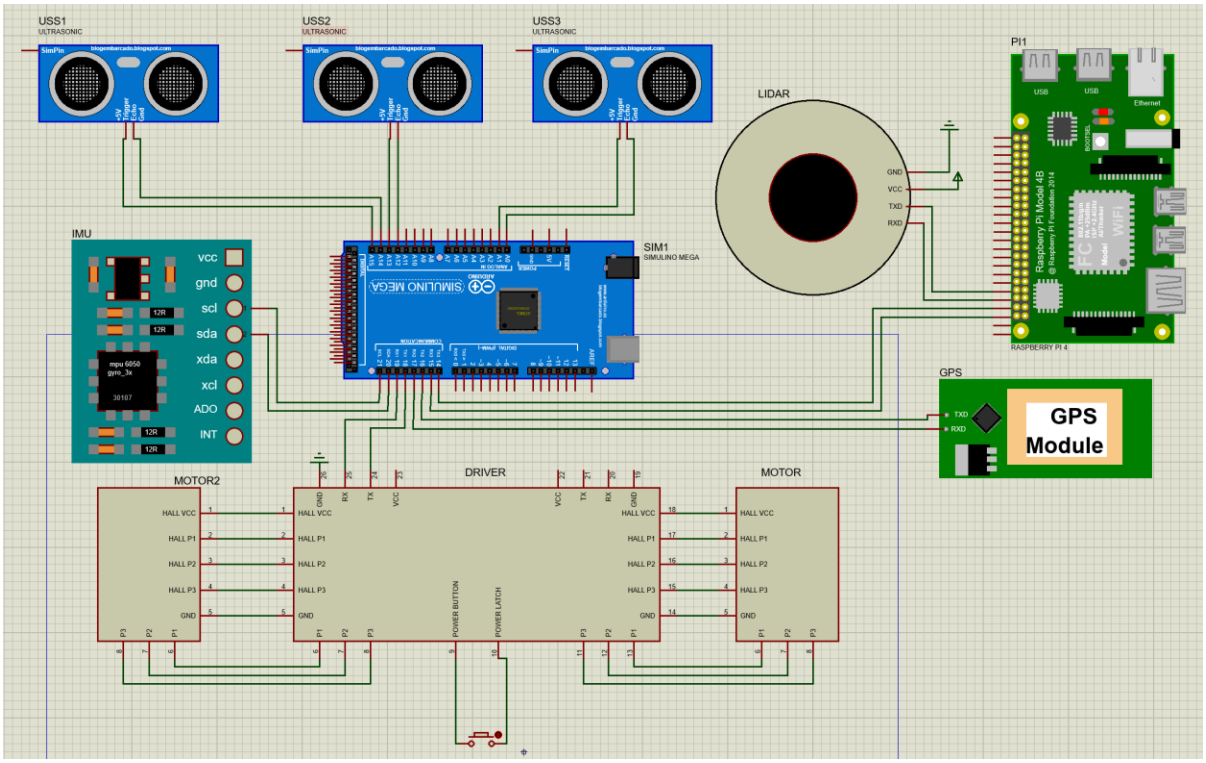


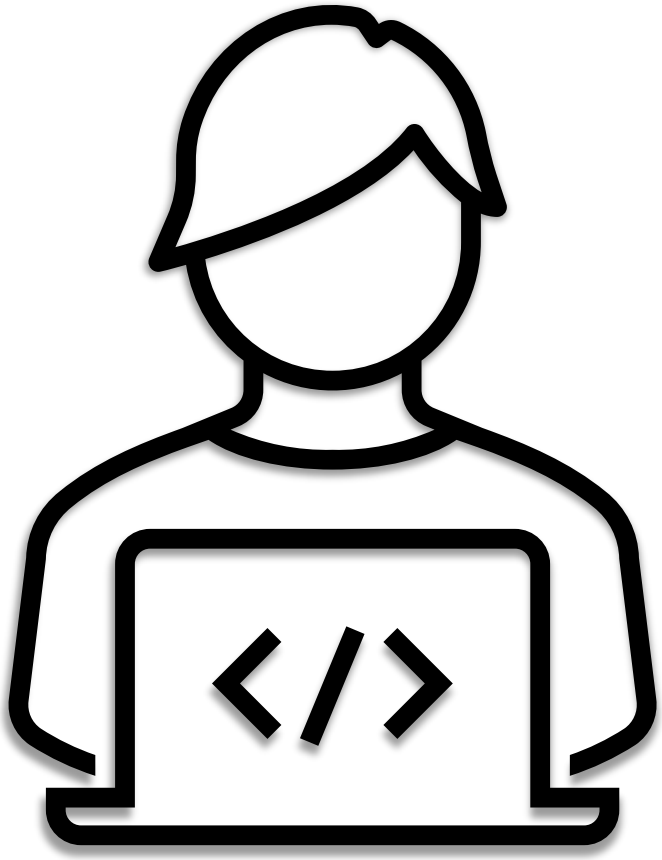
Figure 5.20: System Sensors Schematic Wiring

The schematic wiring diagram illustrates how essential parts of an autonomous robotic system are integrated, displaying a smooth fusion of actuation, processing, and sensor modules. The Arduino Mega is strategically connected to the ultrasonic sensors (USS1, USS2, and USS3) via digital pins. A 5V supply powers the VCC and GND pins of the sensors. Through the transmission and reception of ultrasonic pulses, these sensors allow for accurate obstacle recognition. The Arduino receives vital motion and orientation data from an Inertial Measurement Unit (IMU), namely the MPU-6050, via its accelerometer and gyroscope over the I2C protocol (SCL and SDA pins). In addition to the sensor array, a LIDAR module that is powered by a 5V supply and interfaced via TX and RX pins allows for high-resolution distance measurement for mapping and obstacle avoidance.

Accurate geographic localization and real-time tracking are provided via an inbuilt GPS module that connects to the Arduino via serial connection. At the center of the system is the Arduino Mega, which distributes power via its regulated 5V and GND outputs and coordinates the data flow among all attached peripherals. The Raspberry Pi 4 is tasked with handling more complex processing tasks including real-time mapping and path planning. It connects to the Arduino by a serial link, guaranteeing effective hardware and software coordination.

Two motors are controlled by a motor driver module, which translates control signals from the Arduino's digital outputs (IN1, IN2, IN3, and IN4) to adjust motor speed and direction for mobility. The driver module powers the motors, guaranteeing a sturdy and dependable driving system. Most components are powered by a shared power source, however the Raspberry Pi is powered separately to avoid voltage swings. A highly flexible and effective robotic system design is reflected in this seamless wire architecture, which guarantees optimal performance for real-time obstacle recognition, navigation, and mobility.

Chapter (6): Software Methodology



6.1 Perception

6.1.1 Sensor Fusion

6.1.1.1 Role in the Project

Improve Navigation: Combines data from GPS, IMU and motor feedback to calculate the robot's pose (position and orientation) [5]. Improves the accuracy of navigation by compensating for individual sensor limitations.

Reliable Path Tracking: Fused data enables the robot to maintain its trajectory, even in challenging environments, by dynamically adjusting its path.

6.1.1.2 Principle of Operation

Sensor fusion integrates data from multiple sensors to create a comprehensive understanding of the robot's environment and state. With combining raw data from multiple sources, the system increases the strengths of each sensor to improve accuracy, reliability and decision-making capabilities.

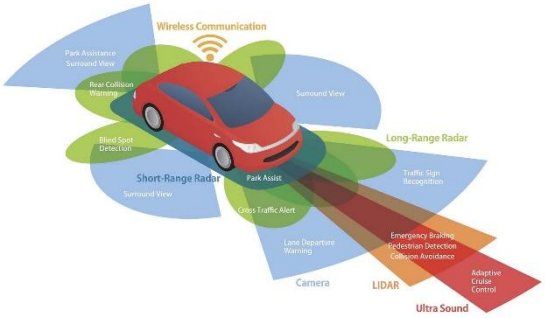


Figure 6.1:Principle of Sensor Fusion

6.1.1.3 Filter Selection

Table 6-1:Filter Selection

Criterion	Complementary Filter	Kalman Filter	Extended Kalman Filter (EKF)	Unscented Kalman Filter (UKF)
Computational Cost	5	4	3	2
Accuracy	3	4	5	5
Ease of Implementation	5	3	2	2
Real-Time Performance	5	4	4	3
Ease of Debugging	5	4	3	3

Criterion	Complementary Filter	Kalman Filter	Extended Kalman Filter (EKF)	Unscented Kalman Filter (UKF)
Noise Handling	3	4	4	5
Adaptability with Nonlinear Systems	2	3	4	5
Total	**28**	**26**	**25**	**25**

6.1.1.4 Fusion Algorithm

The project employs low-level fusion, where raw sensor data is directly combined before being processed. This approach provides a unified and detailed state representation.

Use Complementary filtering to merge IMU and GPS data for real-time position and orientation estimation [18]. Plans to exchange with Kalman filter variations for dynamic environments and scaling.

6.1.1.5 Sensor Contributions

GPS: Provides absolute position data but at a lower frequency.

Used for global positioning and correcting accumulated drift from other sensors.

IMU (BNO055): Offers high-frequency orientation and movement data, including angular velocity and acceleration.

Addresses short-term inaccuracies in GPS positioning.

Motor Feedback: Provides odometry data, such as RPM from each wheel, to estimate distance traveled and direction changes.

6.1.1.6 Challenges

Orientation Drift: IMU gyroscopes accumulate errors over time, causing drift in orientation estimation.

Solution: Motor feedback corrections, usage of BNO055 magnetometer and complementary filtering stabilize the pose estimation

Noisy Data: Sensors may produce noisy measurements due to environmental factors.

Solution: Preprocessing techniques like filtering reduce noise before fusion.

6.1.1.7 Expected Outcomes

Improved Pose Estimation: Accurate calculation of position and orientation in real-time.

Increased Navigation Reliability: Consistent performance across varying terrains and environments.

Enhanced Obstacle Avoidance: Faster and more reliable detection and response to potential hazards.

6.1.2 LiDAR Driver Deployment

6.1.2.1 Introduction

One of the earliest challenges we faced was developing a robust and reliable driver for the X2 LiDAR from scratch. Although the company provides its own driver, it wasn't suitable for our specific application. We required low-level access to the hardware to manipulate the data according to our needs. This necessitated designing a custom driver that could interface directly with the LiDAR's hardware, allowing us to precisely control and extract data in the format we needed.

6.1.2.2 Workflow

The following figure (Figure 5.2) shows the overall workflow.

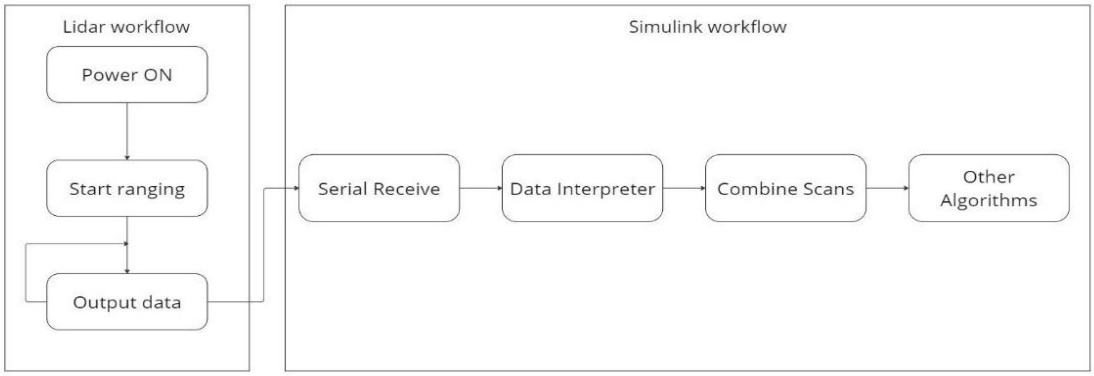


Figure 6.2: LiDAR Workflow

After power on, the system will automatically start ranging, and at the same time it will output to the serial port a message data to start scanning.

Upon power-on, the system will output initial power-on information once. This will include feedback on the device model, firmware version, hardware version, and the device's factory serial number.

6.1.2.3 Data Protocol

Once the system initiates scanning, it will transmit the scan data in the subsequent message. The data protocol is formatted in hexadecimal and sent to the external device through the serial port, following a specified data structure [20]. Each packet typically contains 40 Si nodes (80 bytes), resulting in a total packet length of 90 bytes.

Table 6-2: X2 Data Structure of The Scan Packet

PH		CT	LSN	FSA		LSA		CS		S1		S2		...
LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB	...

Table 6-3: Data Structure Description

Content	Name	Description
PH(2B)	Packet header	2B in length, fixed at 0x55AA, low in front, high in back
CT(1B)	Package type	Indicates the type of the current packet, CT[bit(0)]=1 means the beginning of a lap of data, CT[bit(0)]=0 means point cloud data packet, CT[bit(7:1)] is reserve bit
LSN(1B)	Sample quantity	Indicates the number of sampling points contained in the current packet. There is only one initial point of data in the zero packet. The value is 1
FSA(2B)	Start angle	The angle data corresponding to the first sample point in the sampled data

Content	Name	Description
LSA(2B)	End angle	The angle data corresponding to the last sample point in the sampled data
CS(2B)	Check code	The check code of the current data packet uses a two-byte exclusive OR to check the current data packet
Si(2B)	Sample data	The sampling data of the system test is the distance data of the sampling point, and the interference flag is also integrated in the LSB of the Si node

6.1.2.4 Serial Receive

The raw data is received from the serial port in little-endian format (i.e., the MSB is received before the LSB). The serial receive block is configured so that the data packets arrive in chunks of 2000 bytes at a rate of 10Hz. Typically, each chunk contains 22 scan packets structured as shown in Table 1. You can adjust the chunk size depending on your application. However, keep in mind that larger chunks will slow down the reception rate due to the LiDAR's baud rate limitation of 115200 bits per second. On the positive side, increasing the chunk size allows for more comprehensive scans of the surroundings and reduces the likelihood of losing data packets.

6.1.2.5 Data Interpreter

Once the packets have been extracted from the raw data chunks, the next step is to interpret these packets to obtain the necessary angle and range information. By offsetting 4 bytes from the start of the packet, we acquire the FSA (First Scan Angle) and LSA (Last Scan Angle), which represent the start and end angles, respectively.

To obtain the correct 16-bit representation of each angle, we combine the 2 bytes (MSB and LSB) that constitute each angle. Since the data is in little endian (MSB followed by LSB), we need to adjust this by shifting the MSB 8 bits to the left and then performing a bitwise "or" operation with the LSB. This process ensures that the angles are correctly interpreted, as illustrated in (Figure 5.3).

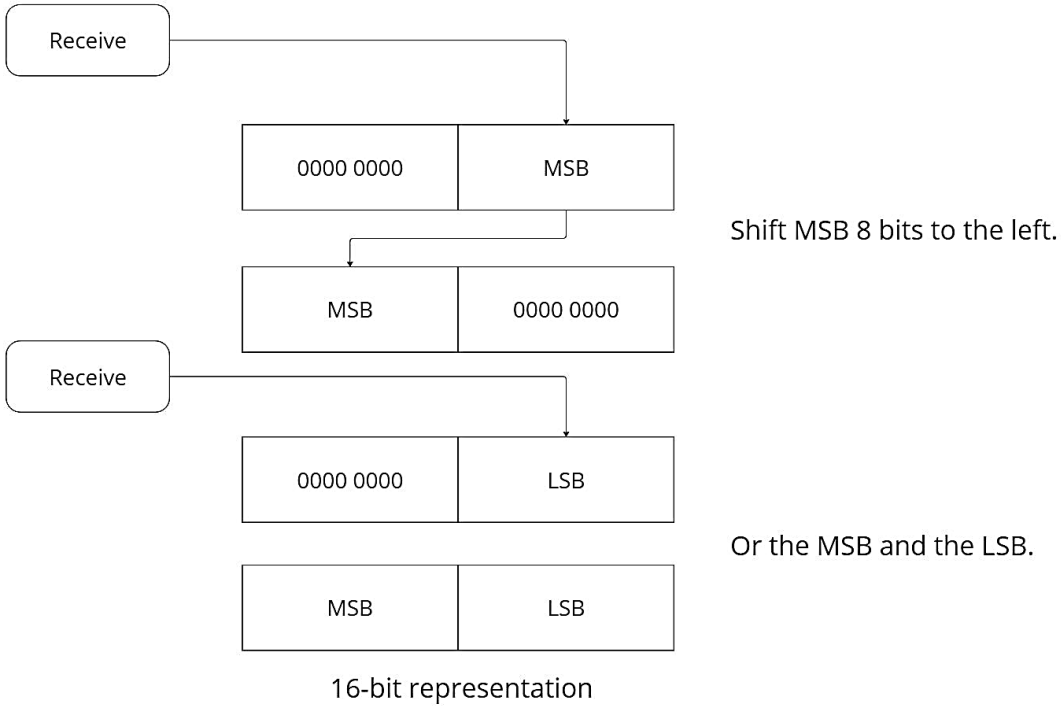


Figure 6.3: Data Interpreter

After retrieving the 16-bit angles, each angle data follows a specific structure illustrated in (Figure 5.4), where 'C' represents the check bit, which is always set to 1. So, we need to further shift the angle data by 1 bit to the right.

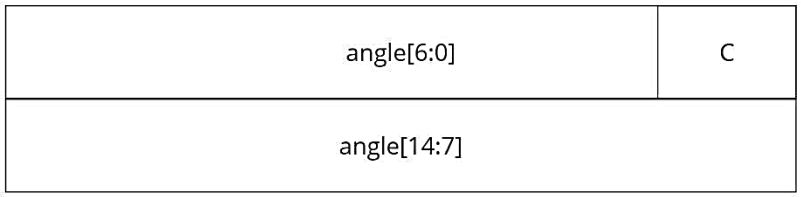


Figure 6.4: Angle Analysis

Angle analysis is carried out in two stages: the first-level analysis and the second-level analysis. The first-level analysis establishes the initial value of the angle, while the second-

level analysis further refines and corrects this initial value. The detailed procedure is described below [20]:

First-Level analysis

Starting Angle Solution Formula : $Angle_{FSA} = \frac{Rshiftbit(FSA,1)}{64}$

End Angle Solution Formula : $Angle_{LSA} = \frac{Rshiftbit(LSA,1)}{64}$

Intermediate Angle Solution Formula :

$$Angle_i = \frac{diff(Angle)}{LSN-1} * (i-1) + AngleFSA \qquad (i = 2,3,\dots,LSN-1)$$

Where:

- Rshiftbit(data, 1) means shifting the data to the right by one bit.
- diff(Angle) refers to the clockwise angle difference from the starting angle (uncorrected value) to the ending angle (uncorrected value).
- LSN denotes the number of packet samples in this frame.

Second-Level analysis

Angle Correction Formula : $Angle_i = Angle_i + AngCorrect_i \qquad (i = 1,2,3,\dots,LSN)$

If Distance $i == 0$

$$AngCorrecti = 0$$

Else

$$AngCorrecti = tand^{-1}(\frac{21.8*155.3-Distancei}{155.3*Distancei})$$

Distance analysis

Distance Solution Formula : $Distance_i = \frac{Si}{4}$

Where *Si* is sample data. Set the sampling data as E5 6F. Since the system is in little endian mode, the sampling point S = 0x6FE5 is substituted into the Distance solution formula to obtain the Distance=7161.25mm.

6.1.2.6 Check Code Parsing

The check code employs a two-byte exclusive OR (XOR) operation to verify the current data packet. Notably, the check code itself is excluded from the XOR operations, and the XOR sequence does not strictly follow byte order.

6.2 ESP32 as an Intelligent Communication Bridge in an Outdoor Autonomous Delivery Robot

6.2.1 Introduction

Autonomous mobile robots are progressively integrated into diverse operational environments, ranging from controlled industrial settings to complex outdoor delivery scenarios. The inherent complexity of these contemporary robotic systems necessitates the implementation of sophisticated control architectures capable of managing a multitude of sensors, actuators, and computational tasks with real-time precision. The development of robust, efficient, and reliable robotic platforms fundamentally relies on the judicious selection and seamless integration of embedded computing units.

This outdoor autonomous delivery robot, designed for navigating dynamic external environments, employs a distributed control system. This architectural choice is a common and advantageous approach in modern robotics, recognized for its capacity to enhance performance, foster modularity, and improve fault tolerance. The system's architecture is structured around three primary computing units, each meticulously assigned specific roles to optimize overall functionality. A Raspberry Pi serves as the high-level mission computer, responsible for complex decision-making and strategic planning. An Arduino Mega is dedicated to low-level motor control and the acquisition of raw sensor feedback. Crucially, an ESP32 development module functions as an intelligent communication bridge and an intermediate data processing unit, linking the higher and lower echelons of the control hierarchy.

This chapter aims to elucidate the strategic integration and specific contributions of the ESP32 within this distributed system. It will detail the "how" and "why" behind the ESP32's selection, its communication protocols with both the Raspberry Pi and Arduino Mega, and its critical role in odometry calculations. Through this exploration, the

chapter will demonstrate the ESP32's indispensable nature in achieving the robot's autonomous capabilities and the overall efficacy of the distributed control paradigm.

6.2.2 Distributed Control Architecture for Mobile Robotics

Distributed control systems (DCS) are characterized by the decentralized execution of tasks, where individual system components operate autonomously yet cooperatively to achieve a common objective. This architectural paradigm offers significant advantages for complex robotic systems, including enhanced modularity, improved fault-tolerance, greater integrability, and superior extendibility. By distributing computational load across multiple processing units, DCS can improve system reaction times and adaptability, particularly vital in dynamic environments that demand continuous data processing and real-time responses This approach also supports concepts of distributed knowledge and decentralized world representation, which facilitates parallel task execution and contributes to a more resilient system. This inherent design choice leads to enhanced adaptability and safety in dynamic environments, primarily by offloading computational load from individual processors and improving overall system reaction times [1].

The outdoor autonomous delivery robot employs a three-tiered distributed architecture, carefully designed to leverage the unique strengths of each component:

- **Raspberry Pi:** This unit serves as the high-level computational core. Its robust processing power and ability to run a full operating system with the Robot Operating System 2 (ROS 2) framework make it ideal for executing complex algorithms such as global path planning, sophisticated obstacle avoidance, and overall mission management. The Raspberry Pi's role is to make strategic decisions and interact with higher-level robotic frameworks, providing the intelligence necessary for autonomous navigation.
- **Arduino Mega:** Functioning as the low-level control unit, the Arduino Mega directly interfaces with and controls the robot's actuators, specifically the hoverboard drivers for the wheels. It is also responsible for acquiring raw sensor feedback, such as wheel speeds in revolutions per minute (RPM). Its real-time capabilities are crucial for executing precise motor control loops, ensuring immediate and accurate responses to commands.
- **ESP32:** The ESP32 acts as an intelligent intermediary, establishing a vital communication bridge between the high-level Raspberry Pi and the low-level Arduino Mega. Beyond merely relaying data, the ESP32 undertakes specific computational tasks, most notably odometry calculations. This strategic task assignment offloads computational burden from the Arduino and provides pre-processed, critical data to the

Raspberry Pi. This integration creates a cohesive and efficient system where each component contributes optimally to the robot's autonomy.
This specific three-tiered architecture leverages the strengths of each microcontroller, optimizing for computational power, real-time control, and communication bridging, rather than attempting to centralize all functions on a single, potentially overwhelmed, processor. This distribution is a deliberate design choice that capitalizes on the distinct capabilities of each microcontroller while mitigating their individual limitations. The Raspberry Pi excels at complex computations but lacks the precise, low-latency real-time input/output (I/O) capabilities typically found in dedicated microcontrollers. Conversely, the Arduino Mega offers robust real-time control for direct hardware interaction but possesses limited processing power and memory, making it unsuitable for complex algorithms or high-bandwidth network communication. The ESP32 fills this critical gap, providing a powerful yet compact platform capable of both efficient computation and versatile communication. This division of labor ensures that each processor can perform its specialized task without being constrained by the demands of other system functions, leading to a more reliable and scalable overall robot.

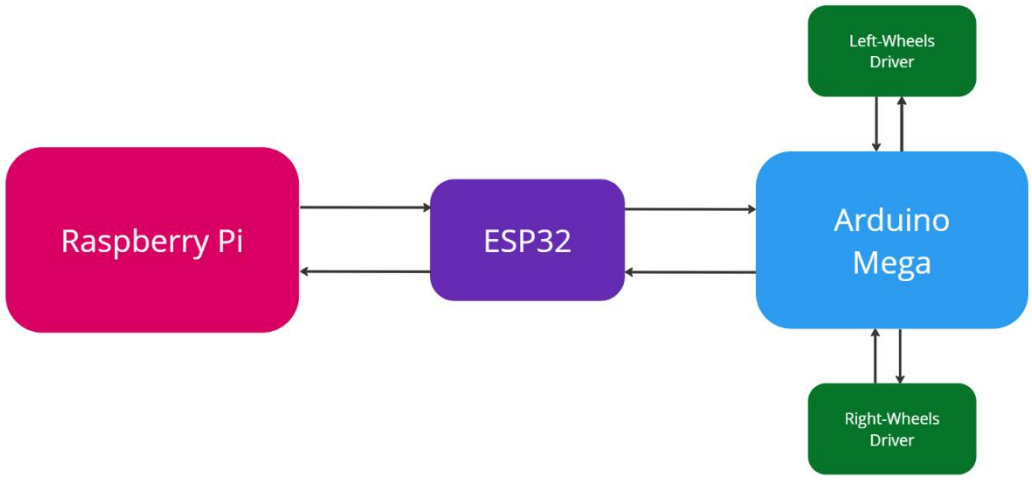


Figure 6.5: A General Overview Of Different Connections Between Controller.

6.2.3 ESP32: Selection and Advantages for Robotic Integration

The selection of the ESP32 development module for the intermediate processing and communication bridge role, in preference to the Arduino Mega, was a strategic decision predicated on the ESP32's superior technical specifications and capabilities . A comparative analysis reveals several key advantages:

Processing Power: The ESP32 features a dual-core Tensilica Xtensa LX6 processor, capable of operating at clock speeds up to 240 MHz. This significantly outperforms the Arduino Mega's 8-bit ATmega2560 microcontroller, which typically operates at 16 MHz[2]. This substantial difference in clock speed and core architecture allows the ESP32 to handle more complex calculations and multitasking operations with greater efficiency [2].

Memory: The ESP32 boasts 520 KB of Static Random-Access Memory (SRAM), a considerable increase compared to the Arduino Mega's 8 KB SRAM [3]. This ample memory space is crucial for accommodating larger programs, managing extensive data buffers, and handling complex data structures required for odometry calculations and communication protocols.

Advanced Timers: The ESP32 is equipped with multiple hardware timers that offer higher resolution and accuracy compared to those found on the Arduino Mega. This precision is critical for time-sensitive tasks such as velocity integration for odometry, where accurate time measurement directly impacts the fidelity of position estimation [2].

Connectivity: While not the primary mode of communication for the bridge in this specific setup, the ESP32's integrated Wi-Fi (802.11 b/g/n) and Bluetooth (BLE and classic Bluetooth) capabilities offer significant future expansion potential. These wireless features could be leveraged for wireless debugging, remote monitoring, or seamless integration into broader Internet of Things (IoT) ecosystems [2].

Real-time Operating System (RTOS) Support: The ESP32's architecture supports the implementation of FreeRTOS, a real-time operating system [2]. This enables the development of multi-threaded applications, allowing for concurrent execution of tasks such as serial communication, odometry calculation, and ROS message handling. The ability to manage multiple tasks concurrently ensures that critical operations are not delayed by less time-sensitive processes, contributing to overall system responsiveness.

The primary motivation for assigning velocity integration and pose estimation to the ESP32 was to alleviate the computational burden on the Arduino Mega. The Arduino Mega, despite its robustness for direct hardware control, is computationally less powerful. Attempting to perform floating-point arithmetic and complex trigonometric calculations for odometry on

the Arduino could introduce unacceptable delays in the motor control loop, potentially compromising the robot's real-time responsiveness and stability. The ESP32's superior processing power and more accurate timers ensure that odometry calculations are performed efficiently and precisely. This offloading allows the Arduino Mega to concentrate solely on its core function: receiving commands and translating them into precise motor control signals, thereby ensuring a robust and reliable low-level control layer. This strategic allocation of tasks prevents the computationally less powerful Arduino Mega from becoming a bottleneck in the control loop, thereby enhancing the performance and stability of the entire robot's control system.

Furthermore, by centralizing odometry on the ESP32, the system achieves a more robust and responsive navigation capability. The Raspberry Pi, which is responsible for high-level path planning and obstacle avoidance, receives timely and accurate pose data from the ESP32. This accurate ego-motion estimation is a critical component for any autonomous system, as cumulative errors in odometry can lead to significant localization drift over time . If the odometry data provided to the Raspberry Pi were inaccurate or delayed due to computational strain on the Arduino, the Raspberry Pi's navigation algorithms would operate on flawed information. This could result in navigation errors, collisions, or inefficient movement. By ensuring precise and timely odometry calculations on the ESP32, the quality of input to the Raspberry Pi's high-level algorithms is maximized, directly contributing to the robot's overall autonomous navigation performance and reliability in an outdoor environment.

Table 6-4: Comparative Features of Arduino Mega and ESP32

Parameter	Arduino Mega (ATmega2560)	ESP32 (Tensilica Xtensa LX6)
Processor	8-bit ATmega2560	Dual-core 32-bit Xtensa LX6
Architecture	AVR	RISC
Clock Speed	16 MHz	Up to 240 MHz
RAM (SRAM)	8 KB	520 KB
Flash Memory	256 KB	4-16 MB
Digital I/O Pins	54	~36

Parameter	Arduino Mega (ATmega2560)	ESP32 (Tensilica Xtensa LX6)
Analog Input Pins	16	~18 (ADC)
Wireless Connectivity	None	Wi-Fi, Bluetooth (BLE)
Operating Voltage	5V	3.3V
RTOS Support	No (typically bare metal)	Yes (FreeRTOS)
Typical Cost	Low	Very Low

6.2.4 Inter-Microcontroller Communication: Arduino Mega to ESP32

Serial communication, specifically Universal Asynchronous Receiver/Transmitter (UART), is a fundamental and widely adopted method for data exchange between microcontrollers and other peripheral devices in embedded systems. This protocol operates using distinct transmit (TX) and receive (RX) lines, eliminating the need for a shared clock signal, which simplifies wiring and streamlines board layout. UART is well-suited for tasks that do not demand extremely high data rates, such as the continuous reading of sensor data or the exchange of basic commands, primarily due to its efficiency and reduced wiring requirements.

To ensure reliable and structured data transfer of critical wheel speed feedback from the Arduino Mega to the ESP32, a custom 8-byte serial protocol was implemented. This protocol is designed for compactness and efficiency, which are paramount considerations in real-time embedded applications. The structure of each packet is precisely defined as follows:

- **Header (1 byte):** A fixed byte, `0xAA`, which serves as a clear indicator of the start of a valid data packet, enabling synchronization between the communicating devices.
- **Left Velocity (2 bytes):** A signed 16-bit integer representing the instantaneous speed of the left wheel in RPM.
- **Right Velocity (2 bytes):** A signed 16-bit integer representing the instantaneous speed of the right wheel in RPM.

- **XOR Checksum (1 byte):** A single byte containing the Exclusive OR (XOR) checksum of the header and the four bytes representing the left and right velocities. This byte is used for basic error detection to ensure data integrity during transmission.
- **Dummy Byte (1 byte):** A fixed byte, `0xEE`, which acts as a placeholder or an additional validation byte within the protocol.
- **Terminator (1 byte):** A fixed byte, `0xEF`, signaling the definitive end of the data packet, aiding in proper packet demarcation.

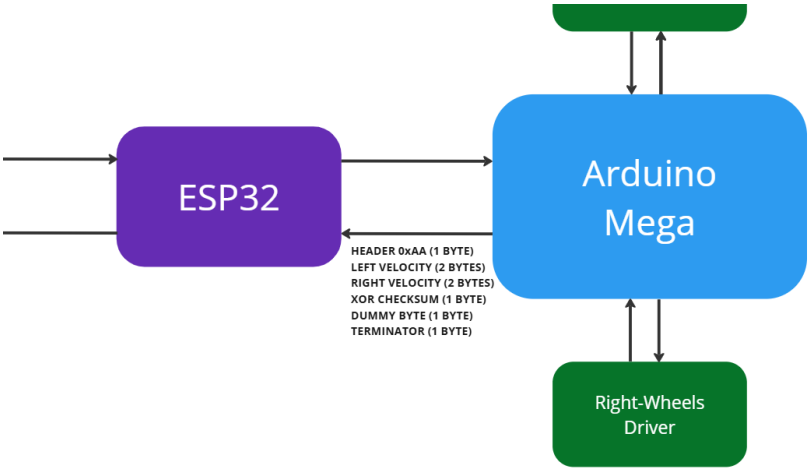


Figure 6.6: Mega to ESP32 data protocol.

The Arduino Mega continuously reads feedback from the two hoverboard drivers, which provide the left and right wheel speeds in RPM. These raw values are then formatted into the described 8-byte packet and transmitted serially to the ESP32.

The inclusion of an XOR checksum within this custom protocol provides a lightweight mechanism for detecting data transmission errors. The XOR checksum is computationally inexpensive, making it a suitable choice for resource-constrained microcontrollers like the Arduino Mega, where more complex error detection codes, such as Cyclic Redundancy Checks (CRCs), might impose an unacceptably high computational load. An XOR checksum possesses a Hamming Distance of two, meaning it can reliably detect all single-bit errors and any error pattern resulting in an odd number of bit errors. However, it is important to note that it is less effective at detecting certain multi-bit errors, particularly even numbers of bit errors, which can go undetected if they cancel each other out within the checksum

calculation. For an 8-bit checksum, approximately 12.5% of two-bit errors may go undetected [4]. The choice of an XOR checksum, while computationally efficient for resource-constrained microcontrollers like the Arduino Mega, represents a deliberate trade-off in error detection robustness. For critical real-time data like wheel speeds, this implies an acceptance of a low, but non-zero, probability of undetected errors, which might be mitigated by higher-level control loops or the inherent redundancy in odometry updates. Given that odometry is continuously updated and higher-level navigation algorithms on the Raspberry Pi likely incorporate filtering or fusion with other sensors (e.g., Inertial Measurement Unit, Global Positioning System), the impact of such a low-probability error is often deemed acceptable in practical embedded systems. This decision reflects a practical engineering balance between computational efficiency and absolute data integrity for a specific data stream.

Table 6-5: Arduino Mega to ESP32 Communication Protocol

Byte Index	Field Name	Value/Type	Description
0	Header	0xAA (uint8)	Start of packet
1-2	Left Velocity	int16	Left wheel speed (RPM)
3-4	Right Velocity	int16	Right wheel speed (RPM)
5	Checksum	XOR of bytes 0-4 (uint8)	Error detection
6	Dummy	0xEE (uint8)	Reserved
7	Terminator	0xEF (uint8)	End of packet

6.2.5 Odometry and Pose Estimation on the ESP32

Odometry is a fundamental technique in mobile robotics for estimating a robot's position and orientation (pose) by integrating wheel encoder data over time. For a differential drive robot, which employs two independently driven wheels, its motion can be accurately decomposed into linear and angular components. The kinematic equations relate the linear velocities of the left (VL) and right (VR) wheels to the robot's linear velocity (Vx) and angular velocity

(Vθ). Given the wheel separation (track width, B), these relationships are typically expressed as:

$$Vx=(VR+VL)/2$$

$$V\theta=(VR-VL)/B$$

The robot's pose, defined by its coordinates (x,y) and orientation (θ), is then updated by integrating these velocities over small, discrete time intervals (Δt). The incremental updates are calculated as follows:

$$\Delta distance=Vx\cdot\Delta t$$

$$\Delta\theta=V\theta\cdot\Delta t$$

$$xk+1=xk+\Delta distance\cdot\cos(\theta k+\Delta\theta/2)$$

$$yk+1=yk+\Delta distance\cdot\sin(\theta k+\Delta\theta/2)$$

$$\theta k+1=\theta k+\Delta\theta$$

Accurate motion estimation is a critical component for any autonomous system, as cumulative errors inherent in odometry can lead to significant localization drift over extended operational period [5].

The implementation of velocity integration on the ESP32 leverages its robust processing capabilities and highly accurate hardware timers. The ESP32 receives raw wheel speeds in RPM from the Arduino Mega, which are then converted into linear velocities (VL, VR) using the known wheel radius. The ESP32's integrated hardware timers are crucial for precise time measurement, specifically for calculating the time difference (Δt) between successive velocity readings. The `micros()` function, which relies on these accurate timers, provides microsecond-level resolution, enabling highly accurate integration of speed over time to compute accumulated distance [Appendix A]. The pseudocode provided in Appendix A illustrates the calculation of `dt = (now - lastUpdateMicros)/1e6` (converting microseconds to seconds), followed by `dist = ((vL+vR)/2)*dt` and `omega = (vR-vL)/0.441` (where 0.441m represents the track width). These calculated linear and angular velocities are then used to

update the robot's posX, posY, and accumTheta values. The odometry calculation on the ESP32, while based on standard kinematic principles for differential drive robots, benefits significantly from the ESP32's precise timers for accurate time integration (Δt). This precision is crucial for minimizing cumulative position errors over extended autonomous operation. Odometry, being an incremental estimation method, is inherently susceptible to error accumulation. The fidelity of the Δt measurement directly impacts the accuracy of the integrated distance and angle, thereby reducing the propagation of timing errors into the pose estimate and contributing to the long-term accuracy and reliability of the robot's navigation.

Once the posX, posY, and accumTheta values, representing the robot's current pose, are calculated, they are packaged into a structured 16-byte data packet. This packet includes a header (0xAA), the three float values (each stored as 4 bytes), an XOR checksum, a dummy byte (0xEE), and a terminator (0xEF) [Appendix A]. This packet is then published on a custom micro-ROS topic named /current_pose using an UInt8MultiArray message type. This ensures that the pose data is readily available to other ROS 2 nodes running on the Raspberry Pi. The ESP32's role in publishing /current_pose forms a critical feedback loop, providing the Raspberry Pi with the necessary real-time state information to close the navigation control loop and adapt its path planning. In an autonomous robot, the ability to accurately know its current position and orientation is fundamental for all higher-level navigation tasks, including path planning, obstacle avoidance, and goal tracking. The /current_pose topic serves as the primary input for the Raspberry Pi's localization and mapping algorithms. If this data is inaccurate or delayed, the Raspberry Pi's decisions will be flawed, leading to suboptimal or failed navigation. Thus, the ESP32's efficient and accurate calculation and publication of this pose information directly enable the Raspberry Pi to perform its high-level functions effectively, creating a closed-loop control system for autonomous navigation.

6.2.6 ROS System Integration: ESP32 and Raspberry Pi via micro-ROS

The Robot Operating System (ROS) is a widely adopted and robust framework for robotics development, with ROS 2 representing its latest iteration, specifically designed to support distributed systems and real-time capabilities. However, full ROS 2 implementations are resource-intensive and generally unsuitable for direct execution on microcontrollers due to their limited computational and memory resources. To address this disparity, micro-ROS was

developed. This lightweight framework extends ROS 2 functionalities to resource-constrained embedded devices like the ESP32. It provides an efficient solution, enabling microcontrollers to seamlessly integrate into a larger ROS 2 ecosystem without requiring the full overhead of a standard ROS 2 installation. Micro-ROS is built upon Micro XRCE-DDS (Extremely Resource-Constrained Environments Data Distribution Service), which effectively brings the Data Distribution Service (DDS) standard—the underlying communication middleware of ROS 2—to embedded systems.

Communication between the ESP32, functioning as a micro-ROS client, and the Raspberry Pi, running a full ROS 2 environment, is facilitated by a micro_ros_agent. This agent acts as a crucial bridge, wrapping the Micro XRCE-DDS Agent and serving as a server between the DDS network and the micro-ROS nodes on the microcontroller . The connection in this distributed system is established via a serial transport layer. The micro_ros_agent is typically launched on the Raspberry Pi with a specified serial device. Concurrently, the ESP32 runs the micro-ROS client firmware, which includes built-in support for serial transports, enabling it to communicate directly with the agent over the UART interface. Upon successful initialization, the agent and client establish a session, allowing for the reliable exchange of ROS 2 messages, including publish/subscribe patterns and service calls, over the serial connection . The successful implementation of micro-ROS serial communication, as evidenced by common issues like missing serial drivers or Docker configuration, underscores the importance of a meticulous setup process. This is often a critical, yet overlooked, aspect of integrating embedded systems with higher-level robotics frameworks. Challenges such as manually installing serial drivers or correctly configuring Docker device mapping highlight that while micro-ROS simplifies the conceptual integration of microcontrollers into ROS 2, the practical challenges often lie in the underlying operating system and hardware driver configurations. Overlooking these low-level details can significantly impede development, emphasizing that robust system integration requires attention to both software frameworks and foundational hardware/OS layers.

The Raspberry Pi, after executing complex path planning and obstacle avoidance algorithms, determines the robot's desired heading (theta) and the distance remaining to its current sub-

goal. These critical navigation commands are then published on a custom micro-ROS topic named /desired_theta (figure 6.7).

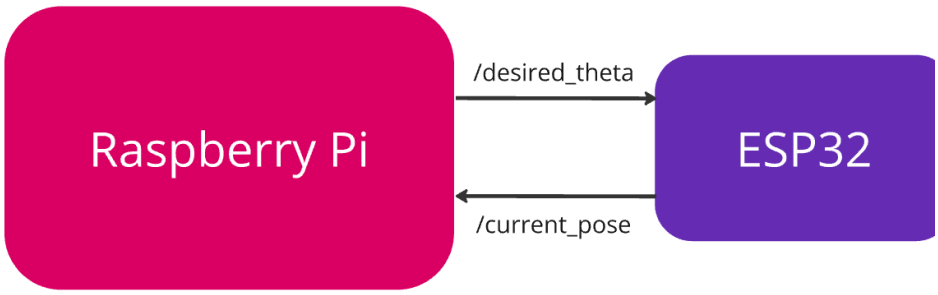


Figure 6.7: Communication Between ESP32 And Rapberry Pi using Micro-ROS.

The ESP32 is configured as a subscriber to this /desired_theta topic. Upon receiving a message, a dedicated subscription_callback function on the ESP32 processes the incoming data, extracting the desired theta and distance values [Appendix A]. The ESP32's subscription to /desired_theta completes the perception-action loop, allowing the robot to dynamically adjust its low-level motor commands based on high-level path planning decisions from the Raspberry Pi. This enables adaptive and goal-oriented navigation. This establishes a direct control flow: high-level planning on the Raspberry Pi informs the intermediate ESP32, which then translates these high-level goals into low-level commands for the Arduino. This closed-loop feedback mechanism is essential for any autonomous system to react to its environment and achieve its objectives. The ESP32 acts as the crucial link that translates abstract navigation goals into direct motor actuation, ensuring the robot can follow its calculated path dynamically.

The desired theta (in radians) and distance (in meters) are represented as floating-point numbers. For efficient and reliable serial transmission over a byte-oriented interface, these floating-point values must be accurately converted into a sequence of bytes. Common methods for this conversion in embedded C/C++ programming environments include using a union or memcpy operations. A union allows the same memory location to be accessed as different data types (e.g., a float and a byte array), while memcpy directly copies the byte representation of the float into a byte array. A critical consideration during this conversion is

endianness. The byte order of a floating-point number can vary significantly between different processor architectures (e.g., little-endian versus big-endian). To ensure correct interpretation by the receiving Arduino Mega, the byte order must be consistent or explicitly handled (e.g., by converting to a standardized network byte order if necessary). For direct microcontroller-to-microcontroller communication within a known system, ensuring consistency in implementation is often sufficient. The careful management of data types and endianness during float-to-byte array conversion for serial communication is a subtle but critical detail that ensures data integrity across heterogeneous processors. This prevents misinterpretation of navigation commands and maintains system robustness. If the Raspberry Pi and the ESP32 encode the float bytes differently from how the Arduino Mega expects to decode them, the `desired_theta` and `distance` values will be corrupted. This would lead to incorrect motor commands and erratic robot behavior. This seemingly minor technical detail is paramount for the reliability of the entire navigation system, as it directly impacts the accuracy of the commands sent to the motor controller.

6.2.7 Command Transmission: ESP32 to Arduino Mega

After processing the desired heading and distance values received from the Raspberry Pi, the ESP32 constructs another custom serial data packet specifically for transmission to the Arduino Mega. This protocol is meticulously designed to convey the high-level navigation commands to the low-level motor controller efficiently and reliably. The packet adheres to a compact 12-byte structure to ensure all necessary information is transmitted with minimal overhead:

- **HEADER (1 byte):** A fixed start byte, `0xAA`, for packet synchronization and identification.
- **THETA (4 bytes):** Four bytes representing the desired heading in radians, encoded as a 32-bit floating-point number.
- **DISTANCE (4 bytes):** Four bytes representing the desired distance to travel in meters, also encoded as a 32-bit floating-point number.

- **Checksum (1 byte):** A single byte XOR checksum calculated over the HEADER, THETA, and DISTANCE bytes, providing a basic error detection mechanism for the command packet.
- **Dummy Byte (1 byte):** A fixed byte, `0xEE`, included for protocol robustness and consistency.
- **TERMINATOR (1 byte):** A fixed end byte, `0xEF`, for clear packet demarcation.

The pseudocode in Appendix A demonstrates the construction of this `packet` array, including the use of a `write_float_bytes` helper function to convert the floating-point `posX`, `posY`, and `accumTheta` into their 4-byte representations for transmission.

Upon receiving this 12-byte packet, the Arduino Mega parses the incoming serial data. It performs validation checks on the header, checksum, and terminator bytes to ensure the integrity of the received data. Subsequently, it extracts the desired `theta` and `distance` values. These values serve as crucial inputs to the Arduino's internal control algorithm. Based on these desired values and its internal understanding of the robot's kinematics and current state, the Arduino calculates the precise command signals for the right and left hoverboard drivers. This typically involves a Proportional-Integral-Derivative (PID) control loop or a similar control algorithm to achieve the desired heading and distance by accurately adjusting individual wheel speeds. This process effectively closes the low-level control loop, translating abstract navigation goals into direct motor actuation. The use of a custom, compact serial protocol for command transmission from the ESP32 to the Arduino Mega minimizes communication latency and overhead. This ensures that the low-level motor control can react swiftly to updated navigation commands from the higher-level Raspberry Pi. In embedded systems, especially for real-time control, minimizing the size and complexity of communication protocols is critical for reducing latency and ensuring responsiveness . By keeping the packet small and the parsing straightforward, the time taken for data to travel from the ESP32 to the Arduino and be processed is minimized. This low latency is essential for the Arduino to rapidly adjust motor commands in response to new navigation goals, which is vital for precise robot movement and path following.

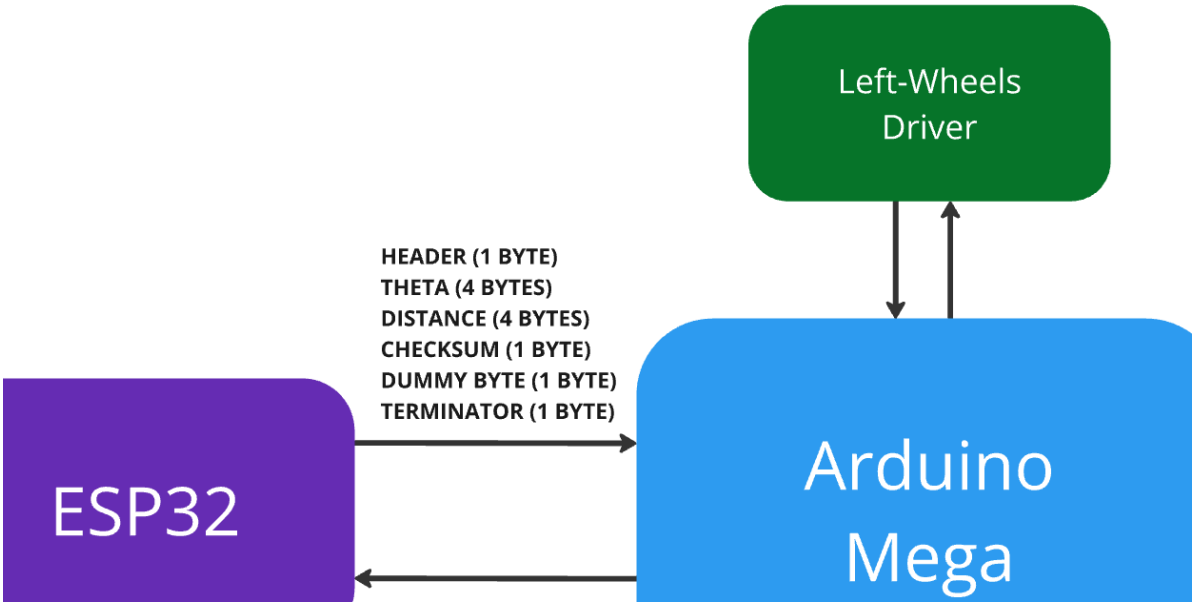


Figure 6.8: ESP32 to Arduino Mega Data Protocol.

Table 6-6: ESP32 to Arduino Mega Communication Protocol

Byte Index	Field Name	Value/Type	Description
0	Header	0xAA (uint8)	Start of packet
1-4	Desired Theta	float (4 bytes)	Desired heading in radians
5-8	Desired Distance	float (4 bytes)	Desired distance in meters
9	Checksum	XOR of bytes 0-8 (uint8)	Error detection
10	Dummy	0xEE (uint8)	Reserved
11	Terminator	0xEF (uint8)	End of packet

6.2.8 Conclusion

The ESP32 development module proved to be an indispensable component in the outdoor autonomous delivery robot's architecture. It effectively served as a robust communication bridge, seamlessly linking the high-level decision-making capabilities of the Raspberry Pi with the low-level motor control functions of the Arduino Mega. Beyond its role in data relay, the ESP32's superior processing power and accurate hardware timers enabled it to

efficiently perform critical odometry calculations, thereby offloading this computational burden from the Arduino and providing precise pose information to the Raspberry Pi.

This three-tiered distributed system successfully leveraged the unique strengths of each microcontroller, optimizing computational load, enhancing real-time responsiveness, and improving overall system reliability and precision. The modularity inherent in this design also facilitates easier debugging, streamlines maintenance, and allows for more straightforward future upgrades. The strategic division of labor among the Raspberry Pi, ESP32, and Arduino Mega has created a highly functional and adaptable autonomous platform.

Looking forward, potential future enhancements could include integrating additional perception sensors, such as LiDAR or cameras, directly with the ESP32 to further enrich its capabilities, potentially offloading some sensor fusion tasks from the Raspberry Pi. The ESP32's built-in wireless capabilities could also be leveraged for over-the-air firmware updates or more advanced remote diagnostics and monitoring. The robust and modular framework established by this distributed architecture provides a strong foundation for developing more complex and sophisticated autonomous systems capable of operating effectively in diverse and challenging outdoor environments.

6.3 Navigation

6.3.1 Implementation Approaches

6.3.1.1 Standalone

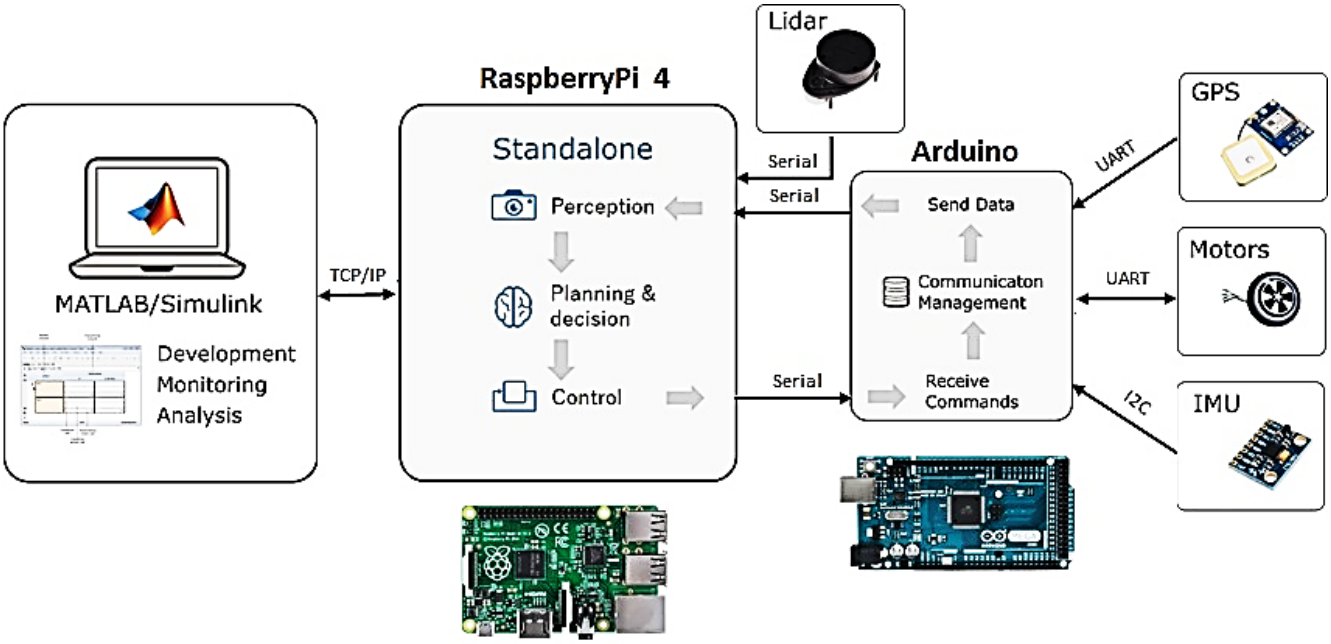


Figure 6.9:Standalone Implementation Approach

The initial implementation strategy involved executing the entire project autonomously on a Raspberry Pi. The Raspberry Pi was tasked with real-time perception and processing, subsequently transmitting motor commands to the motors via an Arduino Mega. The Arduino Mega functioned as a communication management unit, serving as an intermediary between various sensors and the primary processing unit. Simulink was utilized for code development and deployment onto the Raspberry Pi. Additionally, it facilitated the analysis and plotting of logged data. Despite testing this approach, it was ultimately abandoned due to significant delays within the system.

It became evident that a single Raspberry Pi, even under ideal conditions, could not manage the vast amount of data in real-time, nor could it execute complex algorithms like Simultaneous Localization and Mapping (SLAM) or Rapidly-exploring Random Trees (RRT*).

6.3.1.2 Remote Server

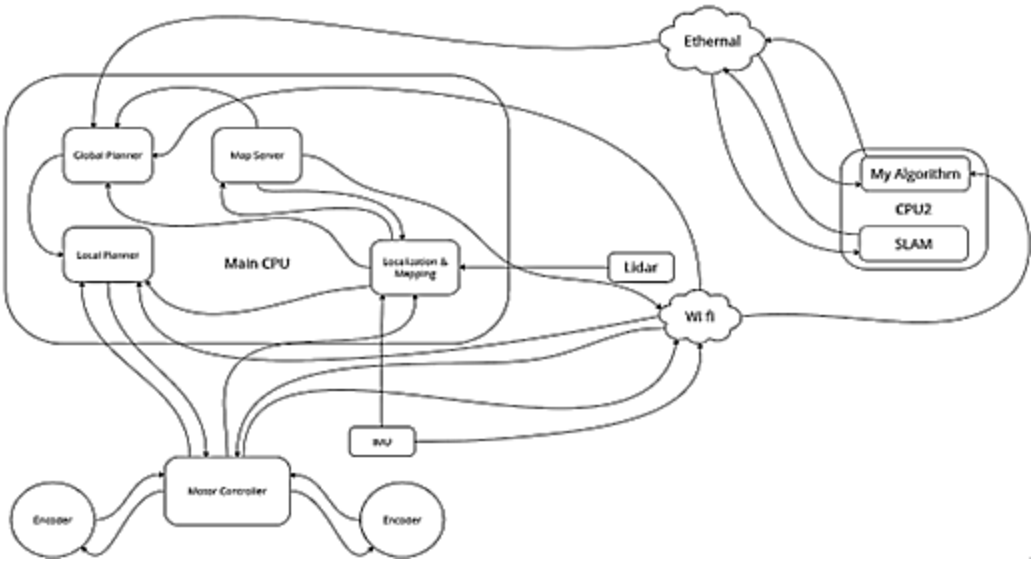


Figure 6.11: Remote Server Diagram

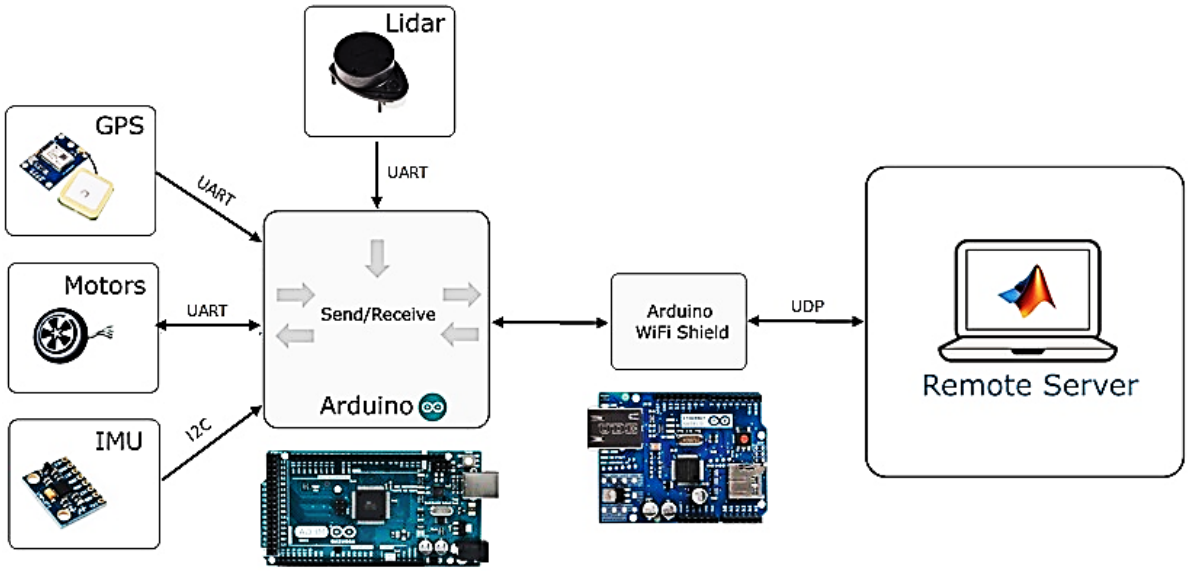


Figure 6.10:Remote Sever Implementation Approach

As a proposed solution to the issues encountered in the initial approach, the concept of offloading all intensive processing and computation to a remote server was explored. This method aimed to utilize the processing power and resources of a remote server, thereby alleviating the burden on the local hardware. The strategy involved equipping an Arduino with a Wi-Fi shield, enabling it to communicate with the remote server via the UDP

communication protocol. The Arduino was responsible for collecting data from various sensors and transmitting it to the server. The server, equipped with more robust processing capabilities, would then perform all the necessary computations and processing tasks.

Upon completing the processing, the server would generate the appropriate motor commands and send them back to the Arduino. The Arduino would then relay these commands to the motors, ensuring the robot followed the desired path. This approach aimed to capitalize on the superior computational power of the remote server, theoretically enhancing the system's overall performance.

However, while this approach was tested, it presented its own set of challenges. The primary issue was that the time saved through fast processing was negated by delays in communication. Several factors can affect UDP communication, leading to performance issues including Network latency, Packet loss, Interferences, Jitter and Bandwidth limitations.

6.3.1.3 ROS-MATLAB

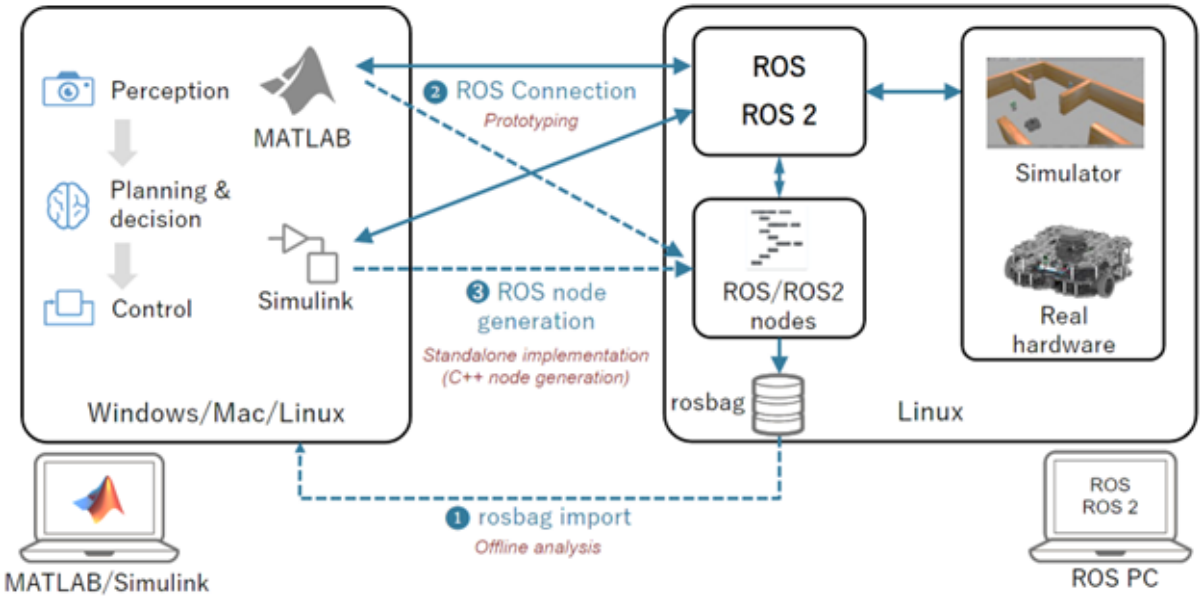


Figure 6.12:ROS-MATLAB Implementation Approach

The third implementation strategy involves integrating both ROS (Robot Operating System) and MATLAB/Simulink, leveraging the strengths of each platform to enhance the overall system's performance and efficiency. ROS, a powerful middleware, is well-suited for real-

time applications due to its robust communication framework and Quality of Service (QoS) features [26]By utilizing ROS's fast communication capabilities, the latency in data transmission can be significantly reduced, thus enhancing the system's responsiveness.

Simultaneously, MATLAB/Simulink continues to play a crucial role in the development, analysis, and visualization of the algorithms. The combination of ROS for real-time communication and MATLAB/Simulink for development and analytical tasks creates a powerful synergy. This hybrid approach aims to leverage the best of both worlds: ROS's efficient communication and MATLAB/Simulink's comprehensive development environment. This strategy is still under development and has yet to be tested. The potential benefits include improved real-time performance, enhanced data processing capabilities, and streamlined integration of various system components. The collaboration between ROS and MATLAB/Simulink is anticipated to offer a flexible and powerful framework for tackling complex robotic challenges, making it a promising avenue for future development and testing.

6.3.1.4 Fixed-Map

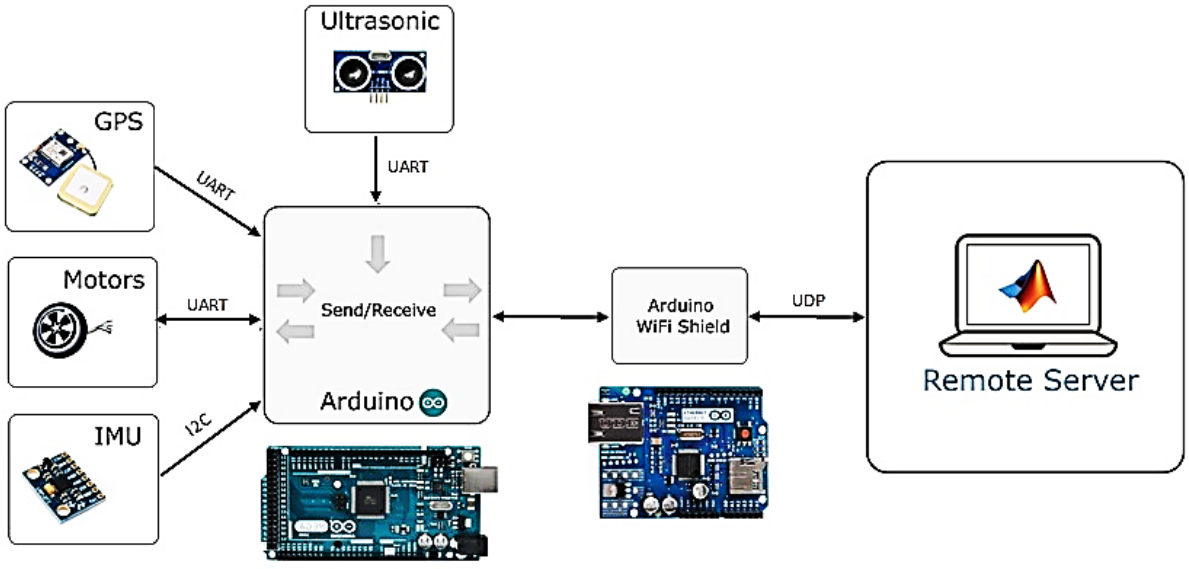


Figure 6.13: Fixed Map Implementation Approach

The fourth approach under consideration involves abandoning the idea of dynamically updating both the map and the path using complex algorithms like SLAM (Simultaneous Localization and Mapping) and RRT* (Rapidly-exploring Random Trees).

Instead, this strategy assumes that the map of the environment is fixed, and the robot navigates along predefined paths between known nodes. By employing a fixed map, the robot is relieved from the computational burden of continuously updating its environment.

This approach simplifies the navigation problem by providing a stable and consistent framework within which the robot operates. The robot's movement is constrained to predetermined paths, which connect specific, well-defined nodes within the environment. This strategy offers several advantages, including reduced computational load, as the system can operate more efficiently without the need for real-time mapping and path planning. It enhances reliability, as fixed maps and predetermined paths lead to more predictable and reliable robot behavior, minimizing the chances of unexpected deviations or navigation failures. Additionally, it simplifies the problem scope by setting clear parameters for the autonomous navigation problem, making it more manageable and solvable. This approach is particularly suitable for outdoor environments where the layout is relatively stable and known in advance. It offers a practical solution for autonomous navigation by leveraging fixed infrastructure, facilitating smoother and more reliable robot operation. This strategy is currently being considered and has the potential to effectively address some of the challenges faced in outdoor autonomous navigation.

6.3.2 Navigating the Outdoors: A Novel Approach to Autonomous Delivery Robot Design

6.3.2.1 Introduction: The Imperative for Outdoor Autonomous Delivery

Autonomous delivery robots represent a transformative technology poised to revolutionize last-mile logistics. Their potential benefits are extensive, encompassing enhanced efficiency, a reduction in carbon emissions, and considerable scalability across diverse operational environments, including urban centers, university campuses, and gated residential communities [26]. Despite these compelling advantages, the successful deployment of such robotic systems in real-world outdoor settings faces formidable challenges. The primary obstacle revolves around achieving consistently reliable and efficient autonomous navigation within dynamic and often unstructured environments. This chapter delineates the developmental trajectory of an outdoor autonomous delivery robot, specifically addressing the critical navigation problems

encountered during its design and implementation. It further elaborates on the innovative solutions devised to overcome the inherent limitations of conventional robotic navigation paradigms. The central difficulty lies in establishing robust environmental mapping and executing real-time path planning in unpredictable outdoor terrains, tasks that traditional methodologies frequently struggle to accomplish effectively.

6.3.2.2 Challenges in Outdoor Autonomous Navigation: Limitations of Conventional Approaches

The application of established robotic navigation paradigms to the intricate demands of outdoor autonomous delivery reveals several significant shortcomings. These limitations primarily arise from the substantial computational requirements of conventional algorithms, the inherent unpredictability of outdoor environments, and the constraints imposed by onboard hardware. A critical examination of these issues is essential to understanding the rationale behind the novel solutions presented in this work.

6.3.2.3 The Computational Burden of Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) stands as a foundational technique in mobile robotics, enabling a robot to construct a map of an unknown environment while concurrently determining its own position within that evolving map [27]. This capability is indispensable for autonomous navigation, effective obstacle avoidance, and exploration, particularly in areas where Global Positioning System (GPS) signals are unreliable or unavailable [27].

However, the practical implementation of SLAM, particularly variants based on scan matching, encounters substantial computational hurdles that severely impede real-time application performance. The computational cost of scan–match SLAM is known to escalate significantly as the number of scan points or loop closures increases, with the correspondence and optimization steps demanding far more processing time than the pose prediction step. This escalating requirement stems from the algorithm's underlying mathematical structure: the size of the pose graph or map representation grows with the number of scans ($O(N)$), but solving the resulting nonlinear optimization (up to $O(N^2)$ or worse) in practice [19]. This direct relationship between environmental

scale and computational load explains why even a relatively small 10 m × 10 m area could take approximately five minutes to converge in prior experimental attempts, rendering it unsuitable for dynamic outdoor operations. Consequently, extending scan–match SLAM to vast outdoor environments—such as a 10 km × 10 km area—would produce optimization times that are astronomically prohibitive, effectively making real-time mapping impossible due to the rapid growth in computational demands.

Beyond the sheer processing power, the challenges of SLAM in outdoor environments extend to its fundamental assumptions about environmental stability. Traditional SLAM approaches often presuppose a static world, a condition rarely met in dynamic outdoor settings where pedestrians, vehicles, and transient objects are ubiquitous. Such environmental dynamism can profoundly interfere with accurate positioning and consistent map construction, introducing inconsistencies and errors that undermine the system's reliability. Furthermore, environments that are feature-sparse, such as open fields, or those with repetitive structures can degrade SLAM performance, leading to issues like data sparsity and difficulties in map creation. The substantial volume of point cloud data generated by LiDAR sensors, while offering high precision, exacerbates these computational challenges, especially when real-time processing is required in constantly changing conditions [27]. This highlights that the problem of SLAM in outdoor settings is not merely one of computational speed, but also of maintaining map integrity and accuracy in inherently unpredictable surroundings. The pursuit of viable SLAM solutions often encounters a critical trade-off between performance and economic feasibility. While advanced SLAM systems can offer high precision, their associated costs present a significant barrier to widespread adoption and research. Conversely, many low-cost SLAM systems, constrained by limited resources, frequently fail to achieve the necessary precision and real-time performance for practical applications, thereby reinforcing the difficulty of balancing high-fidelity mapping with economic viability. This dilemma underscores the need for alternative navigation paradigms that can deliver robust performance without incurring prohibitive expenses.

6.3.2.4 Path Planning Inefficiencies: The Case of RRT* and A*

Path planning is an essential component of mobile robot navigation, enabling a robot to determine a collision-free trajectory from a starting point to a destination. Two prominent algorithms in this domain, Rapidly-exploring Random Tree Star (RRT*) and

A*, despite their theoretical strengths, exhibit significant inefficiencies when applied to large-scale, dynamic outdoor environments.

RRT* and its numerous variants are recognized for their probabilistic completeness and asymptotic optimality, implying that they are guaranteed to find an optimal path given sufficient time. However, this theoretical advantage is often undermined by their slow convergence rate, which severely limits their practical efficiency, particularly in expansive environments. While the base RRT algorithm is probabilistically complete and can be faster than other methods like Probabilistic Roadmaps (PRM) for single-query problems, the paths it generates are often suboptimal [5]. RRT* attempts to address this by continuously optimizing the path through processes like neighbor-searching and tree rewiring, which, while leading to asymptotically optimal solutions, demand a significant amount of memory and time to converge [27]. The computational complexity of RRT is typically described as O(K log K) or O(n log n), where K or n represents the number of nodes in the tree [19]. These computational demands, coupled with issues such as unnecessary sampling and searches, especially in complex environments with narrow passages, contribute to the algorithm's unsuitability for real-time applications.

A*, another widely used graph traversal and pathfinding algorithm, is celebrated for its completeness and optimality, guaranteeing the shortest path under specific conditions. Nevertheless, A* faces a substantial challenge in terms of space complexity, as it stores all generated nodes in memory. As shown in [Russell & Norvig (2020)], the time complexity for breadth-first search is $O(b^d)$, where b is the branching factor and d is the depth of the shallowest solution. This memory intensity renders A* less suitable for large-scale environments. The planning speed of A* is highly dependent on map resolution. For a 100x100 grid map, simulation results indicate planning times ranging from approximately 269 to 302 seconds (4.5 to 5 minutes) [16]. This aligns with observed performance, where A* could take between 3 to 7 minutes to plan a safe path in a 100m by 100m map, with higher resolutions extending planning time significantly. Higher resolution, while offering greater accuracy, dramatically increases the number of nodes in the search space, thereby extending planning time. For a 10km x 10km outdoor map, the computational time would be astronomically high, making real-time planning impossible. The asymptotic optimality of RRT* and the completeness of A* are

achieved at the expense of computational scalability. The slow convergence of RRT* and the exponential memory and time complexity of A* (with respect to map size and resolution) directly translate to the observed multi-minute planning times, rendering them impractical for real-time, dynamic path planning in large outdoor spaces.

Traditional A* algorithms, similar to SLAM, often rely on pre-constructed high-precision maps, which inherently limits their large-scale deployment in dynamic, unpredictable outdoor environments. While improved variants, such as EBS-A*, can enhance efficiency by a notable margin (e.g., by 278% over traditional A*) [27], the fundamental scalability issue for vast, dynamic outdoor maps persists. This highlights that the trade-off between path optimality and real-time performance is a critical design consideration for outdoor autonomous robots. Algorithms that prioritize theoretical optimality often sacrifice the responsiveness required for safe navigation in unpredictable environments. This compels the adoption of pragmatic, computationally lighter solutions for immediate, local planning. The impact of map resolution on A* performance is often underestimated; it is not merely the physical size of the map, but the granularity of its representation that exponentially impacts performance. This makes high-fidelity mapping for A* in large outdoor areas computationally prohibitive, as the desire for accuracy (higher resolution) directly conflicts with the need for real-time performance.

Table 6-7: Comparative Analysis of Mapping and Path Planning Algorithms (SLAM, RRT, A*)*

Algorithm	Primary Function	Computational Complexity (Theoretical)	Typical Performance (Observed/Cited)	Scalability for Large Outdoor Maps	Suitability for Real-time Outdoor Navigation	Key Limitations
SLAM	Mapping & Localization	O(N^2) (landmarks)	~5 min for 10m x 10m map update; "forever" for 10km x 10km; High cost with increased landmarks/range observations	Poor (computational burden, dynamic environments)	Low	High computational cost; Dynamic environment sensitivity; Feature scarcity

RRT*	Path Planning	O(K log K) or O(n log n) (nodes)	Slow convergence rate; Requires significant memory and time for optimal path	Poor (slow convergence, memory)	Low	Slow convergence; Suboptimal initial paths; High memory for optimality
A*	Path Planning	O(b^d) (branching factor/depth)	3-7 min for 100m x 100m map [User Query]; ~269-302 sec for 100x100 grid map; Space complexity O(b^d)	Poor (memory, resolution impact)	Low	High memory usage; Resolution-dependent performance; Dynamic environment sensitivity

6.3.2.5 The Fragility of Fixed Maps and Dedicated Lanes

To circumvent the computational intensity of real-time SLAM, a common strategy has been to assume the availability of a fixed, pre-existing map and to confine robot movement to designated lanes. While seemingly simplifying the navigation problem, this approach is fundamentally flawed and proves highly fragile in the context of dynamic outdoor environments.

Outdoor environments are inherently unpredictable and subject to constant change. Even supposedly "dedicated lanes" are susceptible to unforeseen obstructions, such as improperly parked vehicles, fallen refuse containers, or temporary construction barriers. This dynamic nature renders static, pre-defined maps rapidly outdated and unreliable for safe and efficient navigation. The inherent unpredictability and dynamism of real-world outdoor environments directly contradict the static assumptions underlying fixed maps and dedicated lanes. This fundamental mismatch inevitably leads to frequent blockages, operational failures, and safety hazards, making such an approach impractical and unsafe for autonomous delivery.

Reliance on fixed maps in a dynamic environment can lead to frequent operational failures, necessitating manual intervention or, more critically, resulting in collisions. Studies on sidewalk delivery robots have documented numerous near-misses and incidents, raising significant concerns regarding pedestrian flow, public safety, and accessibility, particularly for vulnerable populations such as older adults and

individuals with disabilities. These issues are exacerbated when robots operate outside highly controlled or isolated settings.22 Furthermore, this approach inherently fails to address the critical problem of real-time local path planning and immediate obstacle avoidance within these lanes, as the map itself does not account for transient obstacles. This limitation underscores the necessity for autonomous systems to possess robust real-time perception and adaptive planning capabilities, rather than relying solely on pre-defined environmental models. The problem shifts from merely knowing the environment to continuously understanding and reacting to its changes.

The concept of "dedicated lanes," while appearing to simplify navigation, inadvertently introduces new challenges related to human-robot interaction and safety liabilities. This demonstrates that infrastructure-based solutions, when applied in complex social environments, can sometimes generate as many problems as they purport to solve.

6.3.3 A Novel Approach to Outdoor Autonomous Delivery

To effectively address the aforementioned limitations, the developed autonomous delivery robot adopts a hierarchical navigation strategy. This approach intelligently combines global path planning with real-time local obstacle avoidance, further complemented by an innovative enhancement in environmental perception.

6.3.3.1 Global Path Planning with Microsoft Bing APIs

The challenge of generating a computationally feasible global path across expansive outdoor distances, while circumventing the scalability issues inherent in traditional path planning algorithms like A* and RRT* for vast maps, was a primary concern. The solution involved leveraging external, cloud-based services for high-level route generation.

Specifically, Microsoft Bing Maps APIs were utilized to generate a high-level, global path between the robot's current location and its designated destination. The Bing Maps Routes API is capable of creating detailed driving or walking routes, providing a series of intermediate waypoints that define the overall trajectory (figure 1) [27]. This approach directly circumvents the scalability limitations of onboard traditional

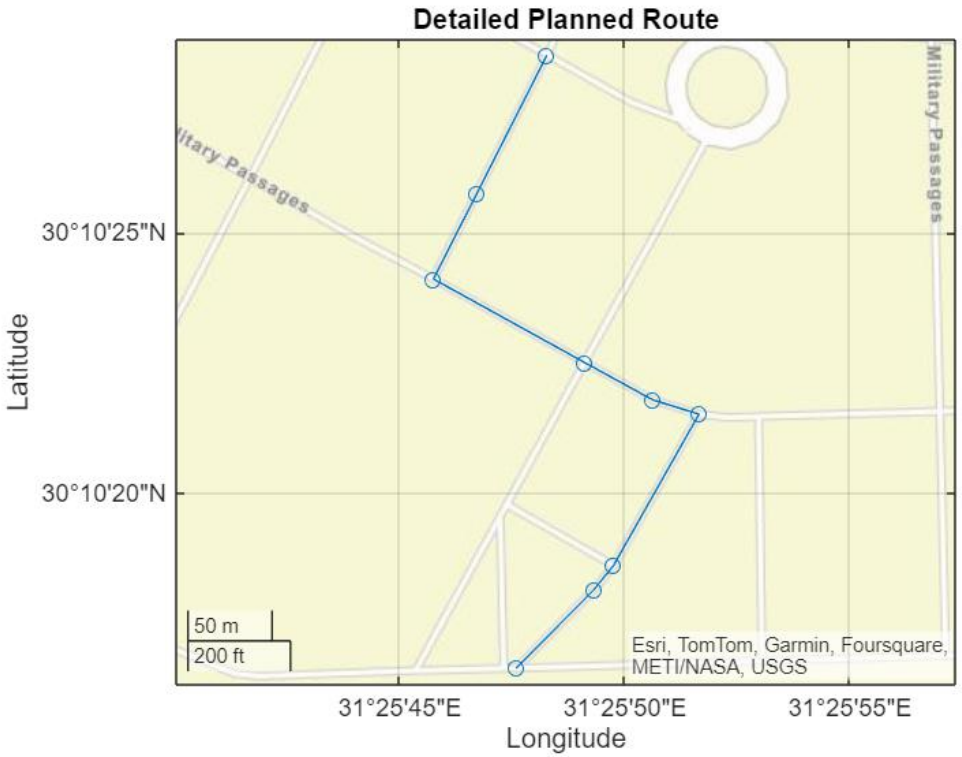


Figure 6.14: Driving Route Divided by a Series of Waypoints

algorithms (A*, RRT*) for large-scale maps, allowing the robot's limited onboard processing power to be dedicated to real-time local navigation.

The API offers robust features for route optimization based on various parameters, including minimizing distance or travel time, with or without consideration for real-time traffic information [27]. This optimization capability is particularly beneficial for autonomous delivery services, where efficiency and timely arrival are paramount. The API can optimize routes involving multiple waypoints, supporting up to 25 intermediate stops while maintaining fixed start and end points, which allows for flexible and efficient multi-stop deliveries [16]. This hybrid approach, combining cloud-based global planning with local onboard execution, represents a scalable and robust paradigm for autonomous navigation in large and dynamic environments. It leverages the strengths of both centralized, data-rich cloud services and decentralized, real-time edge computing.

It is pertinent to acknowledge that the Bing Maps Calculate a Route API is undergoing deprecation, with a transition underway to the Azure Maps Route Directions API [16]. This transition reflects a broader industry shift towards cloud-native, scalable mapping services, aligning with the ongoing need for robust and continuously updated global mapping infrastructure for autonomous systems. The evolution of these geospatial services underscores the importance of adaptability to new platforms and the trend towards more integrated, enterprise-grade cloud solutions in the field of autonomous robotics.

Figure 6.15: Satellite View Showing The Planned Path Between 2 Points

6.3.3.2 Local Obstacle Avoidance using the Vector Field Histogram (VFH) Algorithm

Once a global path, defined by a series of waypoints, is established by the Bing Maps API, the critical task shifts to navigating safely between these waypoints while avoiding immediate obstacles in the dynamic local environment. This problem of real-time local obstacle avoidance is addressed by the Vector Field Histogram (VFH) algorithm [26].

The VFH algorithm is a real-time motion planning algorithm highly regarded for its computational efficiency and robustness, particularly in environments densely

populated with obstacles. It operates by computing obstacle-free steering directions for a robot based on instantaneous range sensor readings, typically acquired from a LiDAR sensor.

The operational mechanism of VFH involves several key steps:

1. **Histogram Grid Construction**: VFH begins by constructing a statistical representation of the robot's immediate environment through a two-dimensional Cartesian histogram grid. This grid is continuously updated in real-time using data from range sensors, such as a LiDAR [19].
2. **Polar Histogram Generation**: The Cartesian histogram is then reduced to a one-dimensional polar histogram centered around the robot's current location. This polar histogram quantifies obstacle density across various angular sectors surrounding the robot, providing a simplified yet effective representation of the local obstacle landscape [19].
3. **Candidate Valley Identification**: The algorithm identifies "candidate valleys," which are consecutive angular sectors in the polar histogram where the obstacle density falls below a predefined threshold. These valleys signify potential clear paths or obstacle-free steering directions [26].
4. **Steering Direction Selection**: VFH evaluates multiple potential steering directions based on the available open space, considering the robot's current heading, its previous steering commands for smoothness, and the overarching target direction (i.e., the next global waypoint provided by Bing Maps). A cost function is employed to weigh these factors, utilizing tunable parameters such as CurrentDirectionWeight, PreviousDirectionWeight, and TargetDirectionWeight. This allows for the selection of an optimal steering direction that effectively avoids obstacles while progressing efficiently towards the global target [9].
5. **Robot Dynamics Integration**: A crucial aspect of VFH is its ability to incorporate robot-specific parameters directly into its calculations. These parameters include RobotRadius (the smallest circle enclosing the robot), SafetyDistance (an additional buffer for safety), MinTurningRadius (the robot's minimum turning capability at its current speed), and DistanceLimits (the effective range of sensor readings to consider). By factoring in these physical dimensions and kinematic constraints, VFH ensures that the generated steering

commands are not only collision-free but also physically feasible for the specific robotic platform, enhancing both safety and maneuverability [8].

The implementation of VFH in this project leveraged the controllerVFH System object available within the MATLAB Navigation Toolbox. This object specifically implements the VFH+ algorithm, providing a robust framework for creating the VFH instance, setting its properties, and computing obstacle-free steering directions from lidarScan data and a target direction [27]. The function within MATLAB's environment proved invaluable for visualizing the polar obstacles plot and masked polar histogram, significantly aiding in the iterative process of parameter tuning and algorithm prototyping (figure 3) [27].

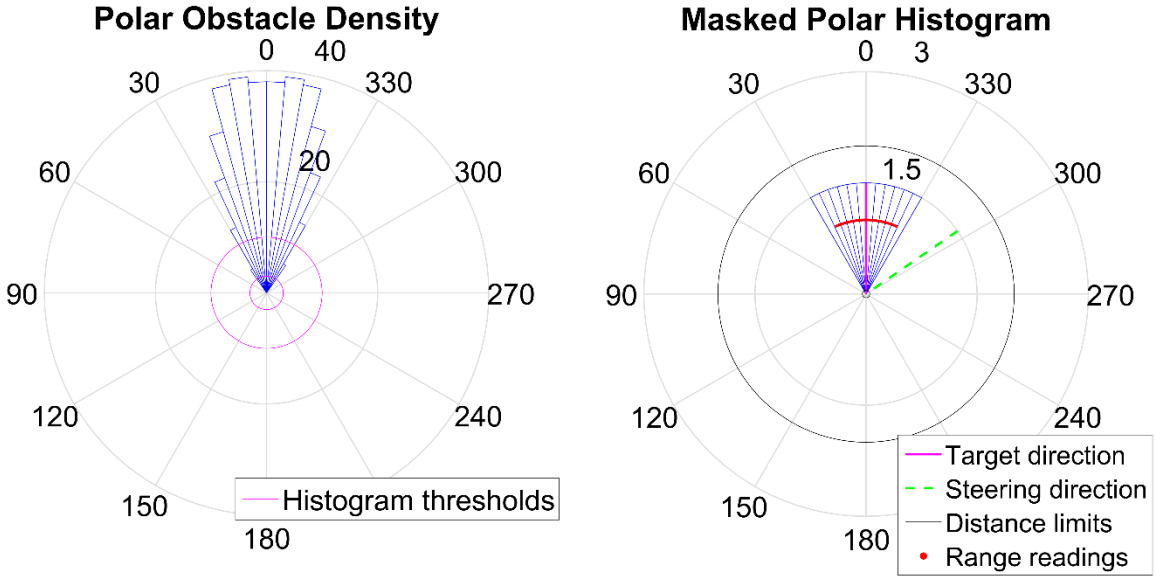


Figure 6.16: Polar Obstacles Plot and Masked Polar Histogram (VFH) method

VFH's design, which emphasizes computational efficiency through histogram-based representations and direct steering command generation, directly enables real-time local obstacle avoidance. This contrasts sharply with the slow, memory-intensive nature of global planners for immediate, dynamic environments, making it an ideal complement to the Bing Maps API for a two-tiered navigation strategy. The combination of a global planner (Bing Maps) and a local reactive planner (VFH) forms a powerful hierarchical navigation architecture. This architecture effectively decouples the complex, large-scale planning problem from the immediate, real-time obstacle avoidance problem, optimizing resource allocation and enhancing overall system

responsiveness and reliability. The tunability of VFH parameters (robot dimensions, safety distance, turning radius, cost function weights) allows for fine-grained control over the robot's reactive behavior, enabling customization for specific robot kinematics and environmental conditions, which is crucial for practical deployment and safety.

Table 6-8: Key Parameters and Impact of the VFH Algorithm (MATLAB Implementation)

Parameter (MATLAB Property Name)	Description	Impact on Robot Behavior	Default Value (MATLAB)
RobotRadius	Radius of the smallest circle enclosing the robot. (figure 3.a)	Ensures adequate clearance from obstacles based on robot size.	0.1 m [14]
SafetyDistance	Additional safety buffer around the robot. (figure 3.b)	Adds a factor of safety for obstacle avoidance.	0.1 m [14]
MinTurningRadius	Minimum turning capability of the robot at current speed. (figure 3.c)	Influences maneuverability around obstacles, ensuring feasible turns.	0.1 m [14]
DistanceLimits	Minimum and maximum range for sensor readings to be considered. (figure 3.d)	Filters out sensor noise/inaccuracies at close range and ignores distant, irrelevant obstacles.	[0.05 2] m [14]
NumAngularSectors	Number of angular bins in the polar histogram.	Affects resolution of obstacle detection; smaller values might miss small obstacles.	180 [14]
CurrentDirectionWeight	Weight for maintaining current heading.	Influences the robot's tendency to continue in its current direction.	(Varies, tunable) [14]

PreviousDirectionWeight	Weight for smoothness based on previous steering.	Promotes smoother trajectories by considering past movements.	(Varies, tunable) [14]
TargetDirectionWeight	Weight for steering towards the target direction.	Controls the robot's eagerness to move towards the global waypoint.	(Varies, tunable) [14]

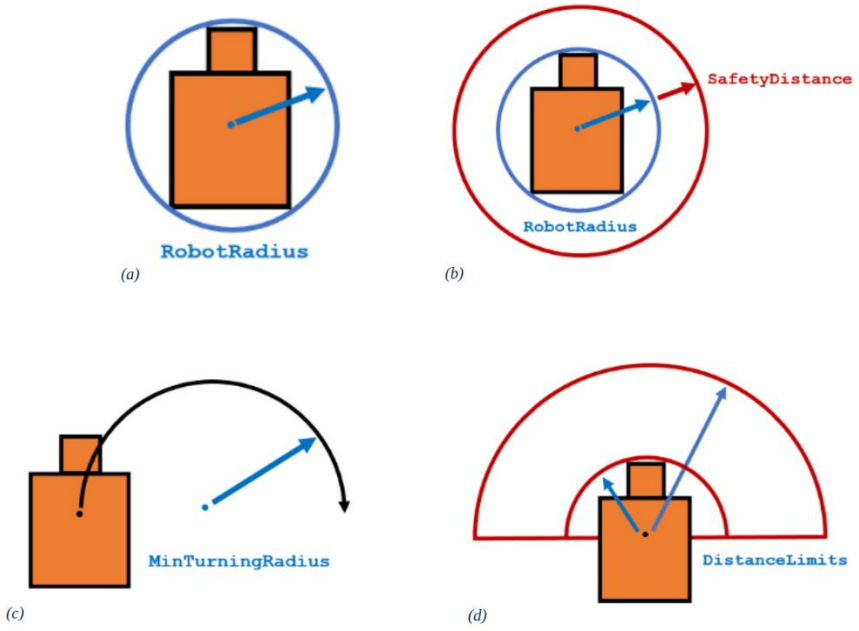


Figure 6.17: Obstacle avoidance using Vector Field Histogram (VFH)

6.3.3.3 Enhanced Perception: Tilting the 2D LiDAR for Ground Obstacle Detection

A critical safety limitation in autonomous ground navigation, particularly for outdoor delivery robots, is the inability of conventional 2D LiDAR sensors to detect ground-level obstacles. Due to their fixed horizontal scan plane, these sensors create a significant blind spot directly in front of the robot's wheels, making hazards such as curbs, potholes, speed bumps, or raised manhole covers undetectable. Such undetected obstacles pose a serious and immediate collision risk.

To address this perceptual deficiency, an innovative and cost-effective solution was implemented: physically tilting the 2D LiDAR sensor by a specific angle. This simple

mechanical modification redirects the sensor's single laser plane downwards, allowing it to scan the terrain immediately ahead of the robot. This effectively expands its vertical field of view, enabling the detection of these critical low-lying obstacles. The inherent limitation of a 2D LiDAR's fixed horizontal scan plane directly causes its inability to detect ground-level obstacles. Tilting the sensor physically alters its scan plane, enabling it to perceive these critical hazards, thereby directly enhancing safety and preventing collisions without requiring expensive hardware upgrades.

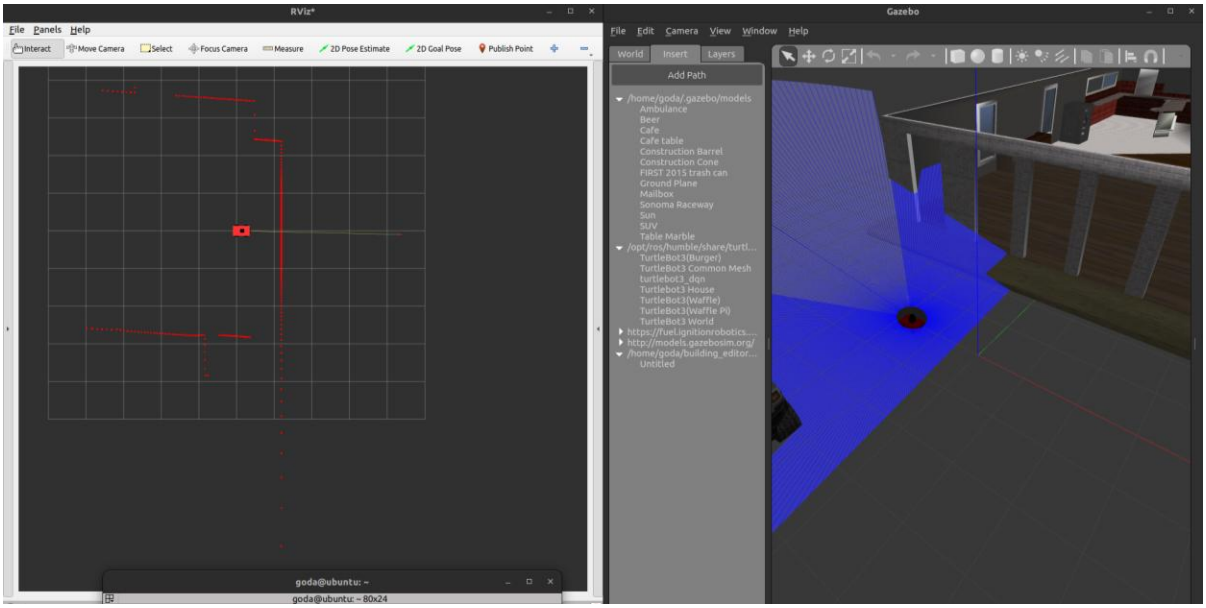


Figure 6.18: Tilting 2D LiDAR to Detect Ground-Level Obstacles - Using Gazebo & Rviz2 for Simulating Different Scenarios.

This approach offers distinct advantages over the direct adoption of more sophisticated 3D LiDAR sensors. While 3D LiDAR sensors provide a comprehensive three-dimensional view of the environment by employing multiple lasers at various angles, capturing millions of data points per second, they are significantly more expensive than their 2D counterparts.Tilting a standard 2D LiDAR provides a much more economical alternative to achieve essential ground-level obstacle detection without incurring the prohibitive cost associated with a full 3D sensor. Furthermore, while 3D LiDAR generates massive volumes of data, which demand substantial computational power for processing (a challenge previously identified with SLAM), a tilted 2D LiDAR maintains a manageable data stream. This reduces the processing burden on the robot's onboard computer, contributing to the overall real-time performance of the system. This solution exemplifies a clever engineering trade-off: instead of simply opting for more expensive,

higher-fidelity sensors, a targeted mechanical modification to a cheaper sensor can effectively address a critical perceptual blind spot. This highlights the value of innovative, low-cost solutions in robotics design. The approach provides targeted detection of the most critical low-lying obstacles, which often represent immediate collision threats for ground robots, without the computational overhead of processing an entire 3D point cloud when it is not strictly necessary for the primary navigation task. The challenge of "negative obstacles," such as depressions or holes, and general terrain variations in unstructured outdoor environments is a recurring problem in autonomous navigation. Tilting the LiDAR directly contributes to mitigating this specific aspect of terrain understanding, an area where traditional 2D and 2.5D methods typically struggle.

6.3.4 Conclusion

This chapter has meticulously detailed the development of an outdoor autonomous delivery robot, outlining the critical limitations encountered with conventional navigation approaches and presenting the innovative solutions implemented to overcome them. The inherent computational burdens associated with traditional Simultaneous Localization and Mapping (SLAM) and classical path planning algorithms such as RRT* and A* proved prohibitive for large, dynamic outdoor environments. These challenges were compounded by the fragility of relying on fixed maps in unpredictable settings and the performance bottlenecks introduced by standard hardware and middleware configurations.

The proposed system effectively addresses these multifaceted challenges through a robust hierarchical navigation architecture. By leveraging Microsoft Bing APIs for efficient global path planning, the computationally intensive task of generating long-distance routes is offloaded to a powerful cloud-based service. This strategic decoupling allows the robot to dedicate its limited onboard computational resources more effectively. For immediate, real-time obstacle avoidance, the Vector Field Histogram (VFH) algorithm provides a computationally efficient and robust solution, ensuring safe navigation between the global waypoints. This two-tiered approach optimizes resource allocation and enhances overall system responsiveness. Furthermore, a simple yet highly effective mechanical modification—tilting a 2D LiDAR sensor—dramatically enhances the robot's perception of critical ground-level obstacles, offering a cost-

effective alternative to the prohibitive expense and data volume associated with 3D sensors.

This integrated approach demonstrates a viable pathway for the development of practical, reliable, and economically feasible autonomous delivery robots capable of operating safely and efficiently in complex and dynamic outdoor environments. Future work could explore further optimization of VFH parameters for diverse and challenging terrains, integrate advanced semantic mapping techniques to enhance the robot's scene understanding capabilities, and adapt the system to evolving cloud mapping platforms, such as the Azure Maps API, to ensure long-term viability and scalability.

6.3.5 Control

6.3.5.1 Objective

The goal was to precisely control the hoverboard's motors by implementing a **Field-Oriented Control (FOC)** approach, using a firmware hack. This enabled full control over motor functionality, by passing the original firmware to integrate the motor control system with the autonomous delivery robot's navigation and obstacle avoidance systems.

6.3.5.2 Hoverboard Motor Control with FOC

Introduction to Field-Oriented Control (FOC)

- **Field-Oriented Control (FOC)** is an advanced technique used to improve the performance of brushless DC motors. Controlling the magnetic field directly enables more efficient, smoother, and precise control over motor torque and speed, which is essential for the robot's navigation tasks.

Firmware Modification Using Hoverboard Firmware Hack

- The **Hoverboard Firmware Hack** source on GitHub was used to replace the original motor control firmware. This hack allowed us to implement the FOC algorithm, which is crucial for achieving the precise control needed in the robot's movement.

- Replacing the factory firmware gave us full flexibility to adjust motor control behavior and optimize the system for the robotic application.

6.3.5.3 Motor Control Process

Key Steps in the Motor Control Process:

1. **Initial Firmware Flashing & Testing**

- Using **Arduino IDE** and embedded programming, the hoverboard's motor control was tested by flashing the modified firmware to the hoverboard's microcontroller. This allowed us to implement **FOC** and precisely control the motor's speed and torque, ensuring smooth motion and responsiveness.
- Initial tests were conducted to confirm motor response and ensure accurate control commands for movement.

2. **Motor Control with FOC**

- The **FOC algorithm** was implemented to control the motors with precision. Key features of this implementation include:
  - **Current Control:** Optimizes the motor's torque output.
  - **Speed and Position Control:** Ensures smooth acceleration and deceleration.
  - **Feedback from Sensors:** Provides real-time data to fine-tune motor responses and accuracy.

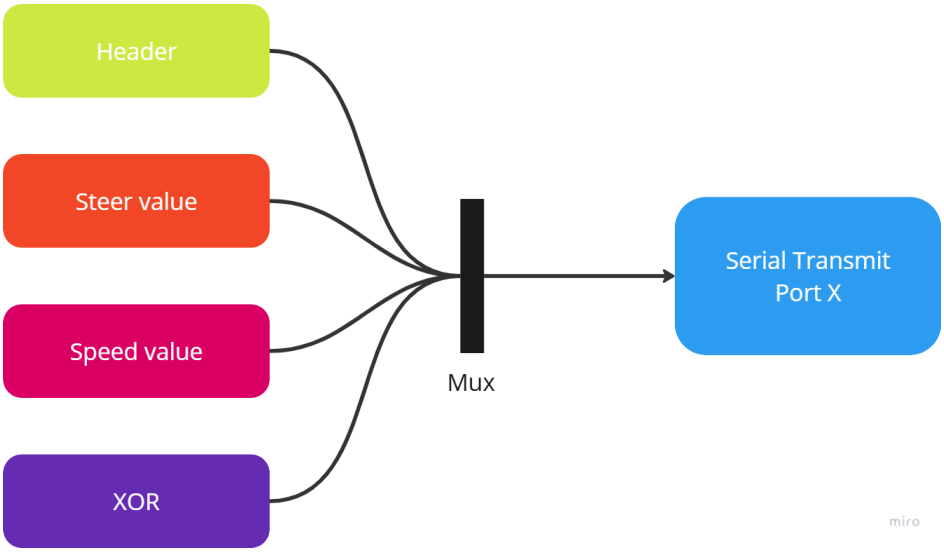


Figure 6.19: Speed & Steer Motor Control

3. **Integration with the Autonomous System**

- After successful firmware flashing, the hoverboard's motor control was integrated with the autonomous system:
    - **Arduino Mega:** Manages motor control and receives commands from the Raspberry Pi via UART.
    - **Raspberry Pi:** Runs the autonomous navigation algorithms and sends movement commands to the Arduino Uno.
    - This integration ensures the precise and responsive control for the robot's navigation, obstacle avoidance, and maneuverability.

6.3.5.4 Benefits of FOC-based Hoverboard Motor Control

1. **Enhanced Motor Efficiency & Smoothness**

FOC provides optimized motor control, ensuring smooth and continuous movement even during complex maneuvers. This is essential for maintaining the robot's stability and performance.

2. **Precision in Navigation**

The FOC implementation allows for finer control over the motor's speed and torque, enabling the robot to execute sharp turns and fine adjustments while navigating obstacles or following a path.

3. **Increased Flexibility**

The modified firmware allows easy customization and further tuning of the motor control algorithms, ensuring flexibility to meet evolving performance requirements.

6.3.5.5 Optimization

1. Transition to Simulink for Enhanced Control visualization

- The next phase involves transitioning the motor control system to **Simulink**. This will provide:

- **Better Control Visualization:** Simulink's graphical interface will allow for easy adjustments and tuning of the motor control parameters, making the system more intuitive and accessible.
- **Easier Integration:** Moving to Simulink will simplify the integration of motor control with the robot's **navigation** and **obstacle avoidance systems**. The control algorithms will be modeled, simulated, and fine-tuned, ensuring smooth interaction between the motor control and other robotic subsystems.

2. Sensor Integration & Feedback

Future improvements will focus on integrating additional sensors, such as encoders and IMUs, for real-time feedback. This feedback will further refine motor control and improve the robot's ability to navigate autonomously with high precision.

Chapter (7): User Interface

7.1 GUI-Based Raspberry Pi Password Lock System Using Servo Motor

7.1.1 Overview

This section presents a touch-based password system using a Raspberry Pi and servo motor. A graphical interface allows users to enter a password. Upon success, a connected servo motor unlocks the mechanism.

7.1.2 Objectives

- Implement a secure, user-friendly password lock system
- Integrate GUI using Python Tkinter
- Control a servo motor using GPIO pins
- Practice embedded programming with feedback

7.1.3 Hardware Components

- Raspberry Pi (with GPIO support)
- Servo Motor
- Jumper wires
- Display Touchscreen

Figure 7.1: Display Touch Screen

7.1.4 Servo Motor Unlock Logic

A PWM signal is used to control the servo. When the password is correct, the duty cycle changes to unlock and then returns to the locked state after a delay.

7.1.5 Screen Display

7.1.5.1 Start Home Screen

Figure 7.2: Start Home Screen

7.1.5.2 Keypad

Figure 7.3: press the password

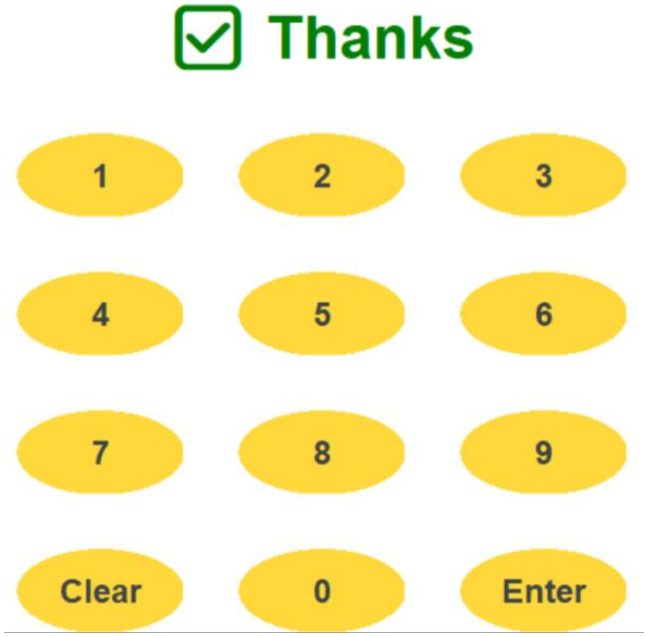


Figure 7.5: Correct Password

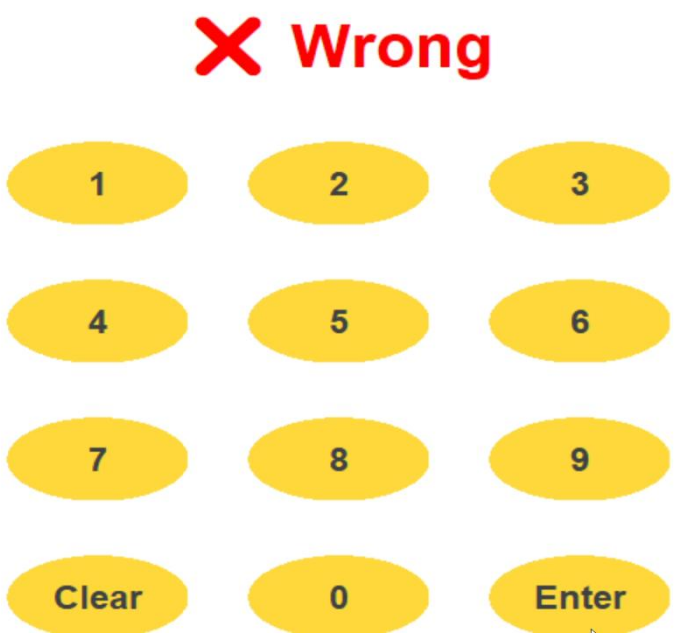


Figure 7.4: Wrong Password

7.1.6 Conclusion

This system demonstrates an effective method for implementing an embedded locking system with GUI feedback. It can be extended with more complex security features such as facial recognition, RFID, or OTP integration.

7.2 Future UI/UX

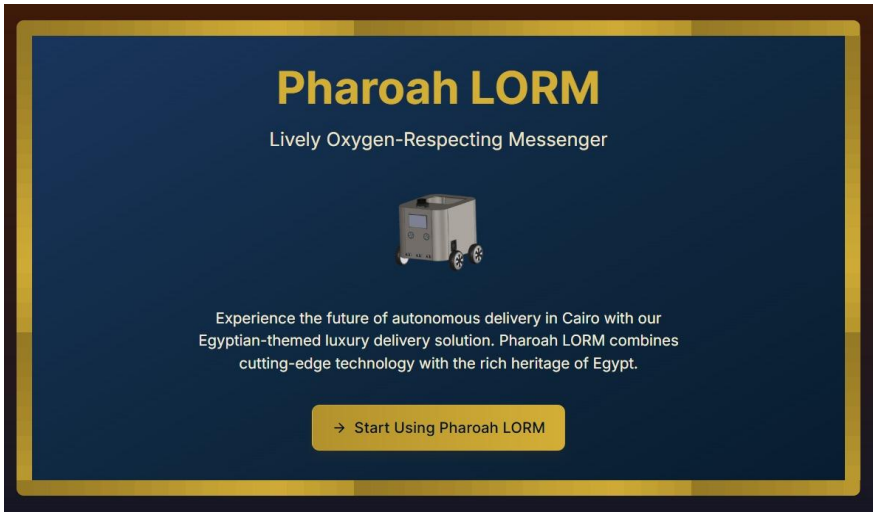
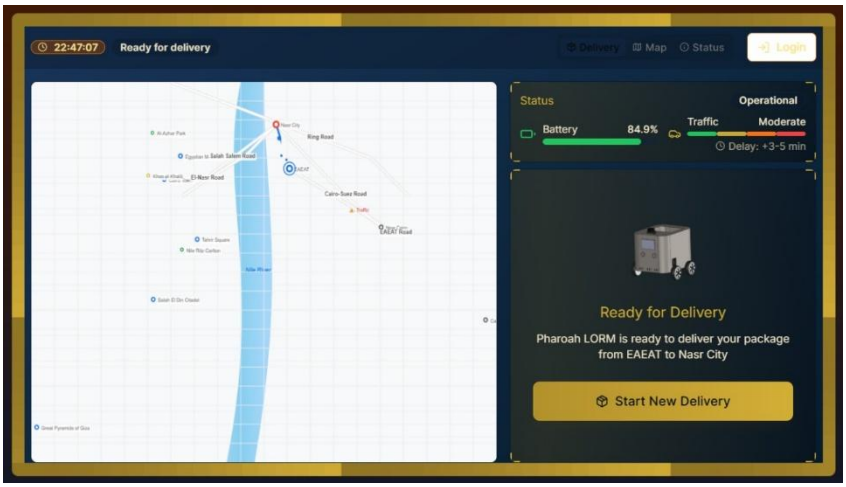
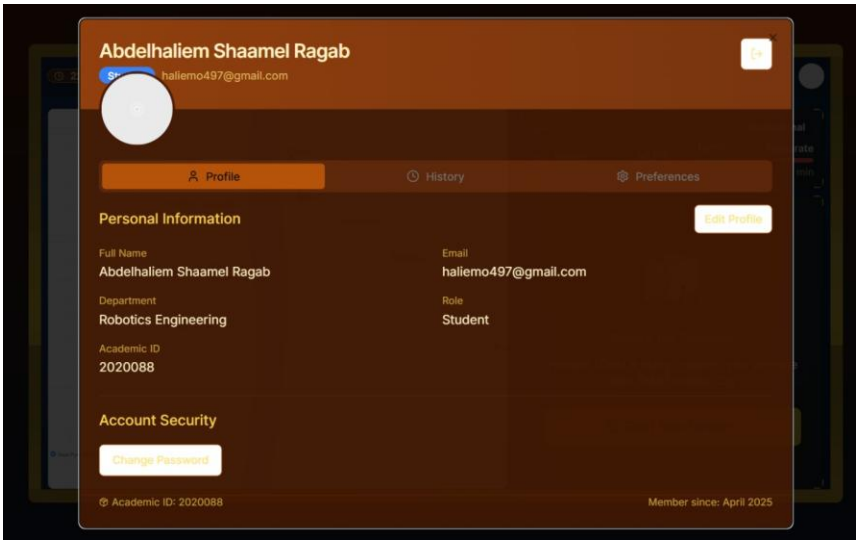


Figure 7.6: Home Screen





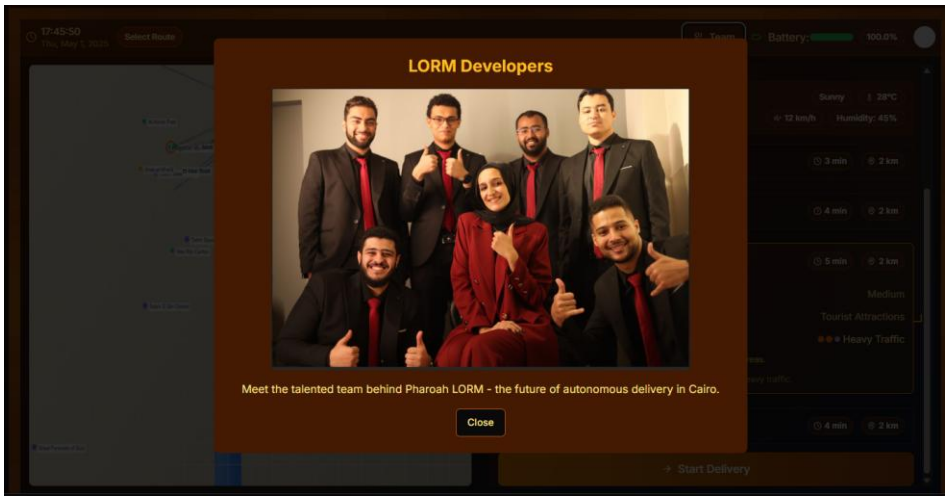


Figure 7.7: Future UI/UX

Chapter (8): Result

After months of design, coding, machining, testing, and refining, the real question was: *Would LORM actually deliver?*

We put LORM through a variety of real-world and simulated tests in urban-like settings. The results spoke clearly — not only did the robot perform its core tasks reliably, but it also exceeded many of our expectations.

LORM achieved **up to 30% faster delivery times** compared to vehicle-based systems in short-range logistics. It navigated both smooth sidewalks and irregular terrain with **minimal localization errors**, showing consistent path accuracy thanks to our integration of **LiDAR, IMU, GPS, and SLAM**.

Battery performance gave us up to **3 hours runtime** in light-load flat scenarios and around **45 minutes under worst-case incline with full load**, which is considered efficient for the payload and motor design.

Most importantly, the robot demonstrated its **adaptability and safety**, successfully avoiding static and dynamic obstacles in crowded paths. Its quiet, electric operation proved ideal for eco-conscious, pedestrian-heavy zones.

These results confirmed that LORM isn't just an academic prototype — it's a feasible, scalable step toward smarter, cleaner, and more efficient last-mile delivery in Egypt's urban environment.

Figure 8.1: Real world implementation

Chapter (9): Conclusion

When we first sketched the idea of LORM, it felt like a distant dream — a robot rolling through Egyptian sidewalks, delivering packages in silence, saving time, cutting emissions, and avoiding traffic.

Now, that dream has wheels.

This project was not just a technical challenge — it was a human journey. It pushed us beyond equations and CAD drawings, into moments of failure, teamwork, rethinking, rebuilding, and learning by doing. LORM became more than a robot. It became a symbol of **what's possible when technology meets local need, and when students think like engineers and act like innovators**.

Through this project, we've seen what it means to take an idea from paper to pavement. We built a system that can be replicated, improved, and scaled. And though there's more work to be done — in monitoring, regulation, and optimization — LORM lays a strong foundation for a new era of autonomous logistics in Egypt.

We close this chapter not with an end, but with the ignition of many futures. This is where smart delivery in Egypt begins.

Chapter (10): Future Work

10.1 Collaborative Robotics Framework

The development of a collaborative robotics framework could enable efficient coordination among multiple robots. Key areas include:

- **Cooperative Path Planning:** Create algorithms that allow several robots to efficiently coordinate their routes for the best possible delivery missions.
- **Shared Resources:** Implement systems for shared infrastructure like charging stations and data communication for better resource utilization.
- **Traffic Management:** Create advanced protocols to control communication and prevent gridlock in busy places.

10.2 Monitoring Platform

- **Real-Time Monitoring:** Develop a cloud-based platform to track LORM's location, battery, and sensor data live.
- **Remote Diagnostics:** Enable remote error detection, system health checks, and alerts for incidents or hardware issues.
- **Fleet Management:** Allow businesses to manage multiple LORM units through a centralized dashboard.
- **Scalability & User Integration:** Integrate with customer apps for delivery updates and support predictive maintenance to ensure reliability and future expansion.

10.3 Charging Station

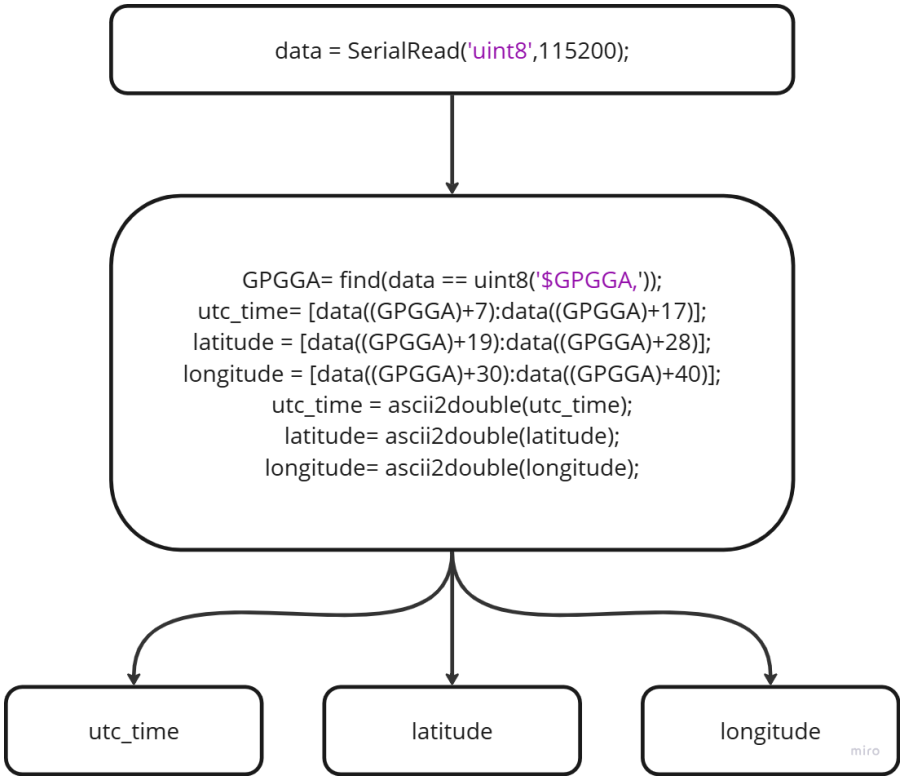
The robot's operational sustainability depends on the creation of an effective and user-friendly charging system. Areas that require improvement include:

- **Autonomous Docking:** Design a system enabling the robot to autonomously locate and connect to a charging station.
- **Fast Charging Technologies:** Investigate cutting-edge charging techniques to boost output and decrease downtime.
- **Integration of Renewable Energy:** Look into ways to include solar panels or other renewable energy sources into the infrastructure that charges.

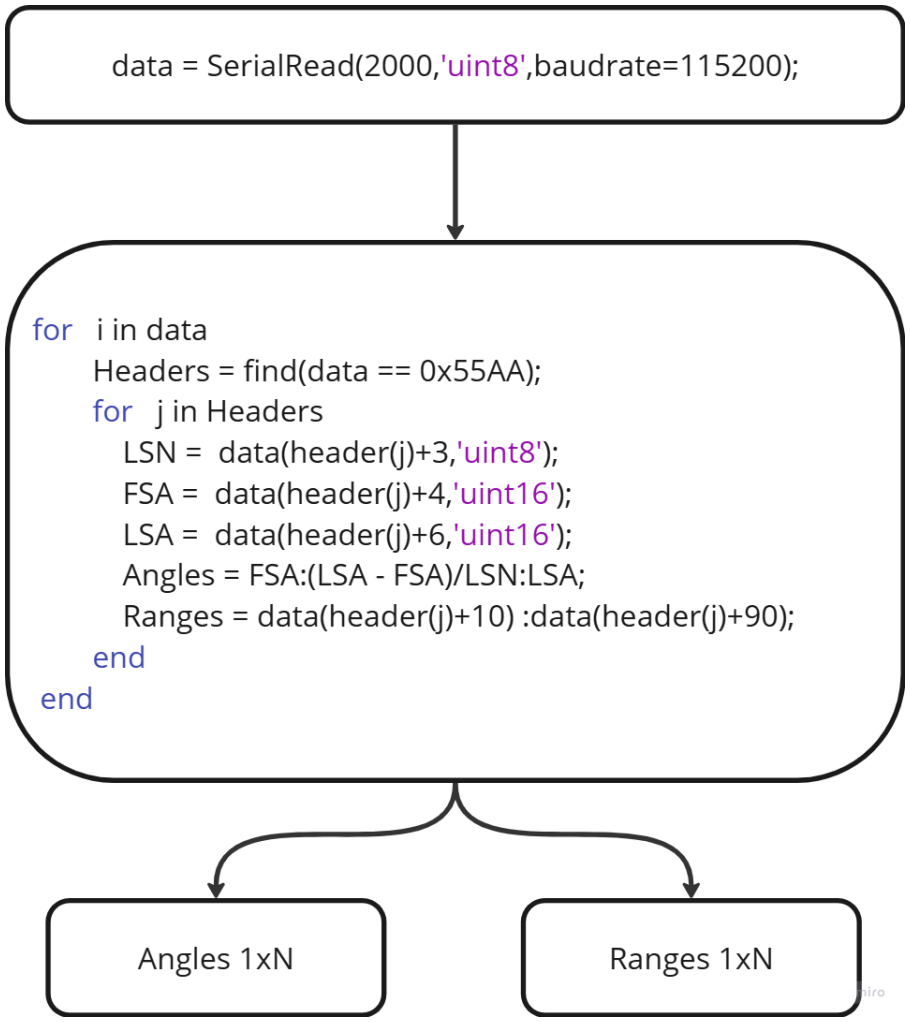
Chapter (11): Appendices

11.1 Appendix A

11.1.1 GPS Code



11.1.2 LiDAR Code



11.1.3 Pseudocode for ESP32 + micro-ROS Sketch

// Constants & Globals

LED_PIN ← 13

RX_BUFFER_SIZE ← 24

FIRST_TIME ← true

rxBuffer ← {0}

rxBufferIndex ← 0

```
packet ← {0}
posX ← 0.0
posY ← 0.0
accumTheta ← 0.0
lastUpdateMicros ← 0
desired_data ← {0}
number ← 0.0
bits ← 0
// ROS Entities
allocator, support, node, executor
publisher, pub_msg
subscription, sub_msg
// Helper: infinite error loop
function error_loop():
    loop forever:
        delay(100 ms)
// Helper: wrap angle into [−π, π]
function wrapTheta(theta):
    theta ← mod(theta + π, 2·π)
    if theta < 0 then
        theta ← theta + 2·π
    return theta − π
```

```
// Subscription callback for incoming desired_theta
function subscription_callback(msg):
    if msg.data.size ≠ 8:
        return
    copy msg.data → desired_data // Extract desired theta and distance
    // Build 12-byte packet: [AA][8 payload][xor][EE][EF]
    packet ← 0xAA
    for i in 0..7:
        packet[i+1] ← desired_data[i]
    xorVal ← packet
    for i in 1..8:
        xorVal ← xorVal XOR packet[i]
    packet ← xorVal
    packet ← 0xEE
    packet ← 0xEF
    Serial2.write(packet, length=12) // Send to Arduino Mega


// Arduino setup() function
function setup():
    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, HIGH)
    Serial.begin(115200) // For debugging/monitor
```

```
    Serial2.begin(115200, pins=16,17) // For communication with Arduino Mega
    delay(2000) // Allow time for serial to initialize
    set_microros_transports() // Configure micro-ROS serial transport
    allocator ← rcl_get_default_allocator()
    support ← rclc_support_init(allocator)
    node ← rclc_node_init("esp32_node", support)
    // Publisher for /current_pose
    publisher ← rclc_publisher_init(node, UInt8MultiArray, "/current_pose")
    init pub_msg with rxBuffer (capacity RX_BUFFER_SIZE)
    // Subscription for /desired_theta
    subscription ← rclc_subscription_init(node, UInt8MultiArray, "/desired_theta")
    allocate sub_msg for 8 bytes
    executor ← rclc_executor_init(support, 1)
    rclc_executor_add_subscription(executor, subscription, sub_msg, subscription_callback)
    lastUpdateMicros ← micros() // Initialize timestamp for odometry


// Arduino loop() function
function loop():
    // 1. Read from Serial2 into rxBuffer (from Arduino Mega)
    while Serial2.available() > 0 and rxBufferIndex < RX_BUFFER_SIZE:
        rxBuffer ← Serial2.read()
    // 2. Parse complete 8-byte frames
```

```
i ← 0
while i ≤ rxBufferIndex − 8:
    if rxBuffer[i]==0xAA and rxBuffer[i+6]==0xEE and rxBuffer[i+7]==0xEF:
        calcXor ← XOR of rxBuffer[i..i+4]
        if calcXor == rxBuffer[i+5]: // Check checksum
            // Extract and convert wheel speeds
            rawL ← combine_bytes(rxBuffer[i+1], rxBuffer[i+2]) // int16 left velocity
            rawR ← combine_bytes(rxBuffer[i+3], rxBuffer[i+4]) // int16 right velocity
            conv ← π*0.165/120 // Conversion factor from RPM to m/s (wheel radius 0.165m, 120 for RPM to rps and then to m/s)
            vL ← rawL*conv;
            vR ← rawR*conv
            // Time & odometry calculation
            now ← micros()
            dt ← (now − lastUpdateMicros)/1e6 // Time difference in seconds
            lastUpdateMicros ← now
            omega ← (vR−vL)/0.441 // Angular velocity (0.441m is track width)
            accumTheta ← accumTheta + omega*dt // Integrate angular velocity
            accumTheta ← wrapTheta(accumTheta) // Keep theta within [−π, π]
            dist ← ((vL+vR)/2)*dt // Linear distance traveled
            posX ← posX + dist*cos(accumTheta) // Update X position
            posY ← posY + dist*sin(accumTheta) // Update Y position
            // Build & publish pose packet (16 bytes) on /current_pose
```

```
            out ← 0xAA
            write_float_bytes(posX) → out[1..4]
            write_float_bytes(posY) → out[5..8]
            write_float_bytes(accumTheta) → out[9..12]
            xorOut ← XOR of out[0..12]
            out ← xorOut;
            out ← 0xEE;
            out ← 0xEF
            copy out → pub_msg.data // Copy constructed packet to ROS message
            pub_msg.data.size ← 16
            rcl_publish(publisher, pub_msg) // Publish pose
            i ← i + 8 // Move to next potential packet
            continue
    i ← i + 1 // Move to next byte if current byte is not a header
// 3. Shift leftovers in rxBuffer
if i < rxBufferIndex:
    move rxBuffer[i..] → rxBuffer
    rxBufferIndex ← rxBufferIndex − i
else:
    rxBufferIndex ← 0
// 4. Handle ROS callbacks
rclc_executor_spin_some(executor, 10 ms) // Process pending ROS messages
```

```
// 5. Heartbeat / Debugging (every 5s or on first cycle)
if millis() − (lastUpdateMicros/1000) > 5000 or FIRST_TIME:
    FIRST_TIME ← false
    lastUpdateMicros ← micros() // Update timestamp for heartbeat
    bits ← bit_copy(number as float) // Example: publish a changing float
    rxBuffer ← 0xAA
    write_uint32_be(bits) → rxBuffer[1..4]
    zero_fill(rxBuffer[5..12])
    rxBuffer ← XOR of rxBuffer[0..4]
    rxBuffer ← 0xEE;
    rxBuffer ← 0xEF
    pub_msg.data.size ← 16
    rcl_publish(publisher, pub_msg)
    rxBufferIndex ← 0
    number ← number + 1
```

11.2 Appendix B

11.2.1 GUI Code

```
import tkinter as tk
import RPi.GPIO as GPIO
import time
# === Servo Setup ===
SERVO_PIN = 18
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(SERVO_PIN, GPIO.OUT)
servo = GPIO.PWM(SERVO_PIN, 50)  # 50Hz
servo.start(0)
def set_servo_angle(angle):
    duty = 2 + (angle / 18)
    GPIO.output(SERVO_PIN, True)
    servo.ChangeDutyCycle(duty)
    time.sleep(0.5)
    GPIO.output(SERVO_PIN, False)
    servo.ChangeDutyCycle(0)
# === GUI Variables ===
input_code = ""
def show_keypad():
    smile_frame.pack_forget()
    keypad_frame.pack(fill='both', expand=True)
def return_to_smile():
    keypad_frame.pack_forget()
    smile_frame.pack(fill='both', expand=True)
def press(num):
    global input_code
```

```
    input_code += str(num)
    code_var.set(input_code)
def clear():
    global input_code
    input_code = ""
    code_var.set("")
def submit():
    global input_code
    if input_code == "1234":
        code_var.set("✅ Thanks")
        code_display.config(fg="green")
        set_servo_angle(90)
        root.after(1500, lambda: [return_to_smile(), clear()])
    else:
        code_var.set("❌ Wrong")
        code_display.config(fg="red")
        root.after(1500, lambda: [return_to_smile(), clear()])
# === GUI Setup ===
root = tk.Tk()
root.title("Delivery Robot Lock")
root.geometry("1024x600")
```

```
root.configure(bg="white")

smile_frame = tk.Frame(root, bg="white")

smile_frame.pack(fill='both', expand=True)

canvas = tk.Canvas(smile_frame, bg="white", highlightthickness=0)

canvas.pack(fill='both', expand=True)

canvas_width = 1024

canvas_height = 600

canvas.create_rectangle(0, 0, canvas_width, canvas_height, fill="#FFD93B", outline="#F4C534", width=4)

cx = canvas_width // 2

cy = canvas_height // 2

eye_r_x = 18

eye_r_y = 25

eye_dx = 40

eye_dy = 40

canvas.create_oval(cx - eye_dx - eye_r_x, cy - eye_dy - eye_r_y, cx - eye_dx + eye_r_x, cy - eye_dy + eye_r_y, fill="#3E4347", outline="")

canvas.create_oval((cx - eye_dx - eye_r_x) + 8, (cy - eye_dy - eye_r_y) + 8, (cx - eye_dx - eye_r_x) + 16, (cy - eye_dy - eye_r_y) + 16, fill="white", outline="")

canvas.create_oval(cx + eye_dx - eye_r_x, cy - eye_dy - eye_r_y, cx + eye_dx + eye_r_x, cy - eye_dy + eye_r_y, fill="#3E4347", outline="")

canvas.create_oval((cx + eye_dx - eye_r_x) + 8, (cy - eye_dy - eye_r_y) + 8, (cx + eye_dx - eye_r_x) + 16, (cy - eye_dy - eye_r_y) + 16, fill="white", outline="")
```

```
canvas.create_line(cx - 80, cy + 40, cx, cy + 100, cx + 80, cy + 40, smooth=True, width=6, fill="#3E4347")

arrow_canvas = tk.Canvas(smile_frame, width=100, height=100, bg="#FFD93B", bd=0, highlightthickness=0)

arrow_canvas.place(relx=0.95, rely=0.85, anchor="e")

arrow_canvas.create_text(50, 50, text="→", font=("Arial", 80, "bold"), fill="#3E4347")

arrow_canvas.bind("<Button-1>", lambda e: show_keypad())

keypad_frame = tk.Frame(root, bg="white")

code_var = tk.StringVar()

code_display = tk.Label(keypad_frame, textvariable=code_var, font=("Arial", 32, "bold"), bg="white", fg="#343a40")

code_display.pack(pady=20)

btn_frame = tk.Frame(keypad_frame, bg="white")

btn_frame.pack()

def draw_ellipse_button(frame, text, row, col, action):

    width = 120

    height = 60

    button = tk.Canvas(frame, width=width, height=height, bd=0, highlightthickness=0, bg="white")

    button.create_oval(0, 0, width, height, fill="#FFD93B", outline="")

    button.create_text(width // 2, height // 2, text=text, font=("Arial", 18, "bold"), fill="#3E4347")

    button.grid(row=row, column=col, padx=20, pady=20)

    button.bind("<Button-1>", lambda e: action())
```

```
    return button

buttons = [
    ('1', 0, 0), ('2', 0, 1), ('3', 0, 2),
    ('4', 1, 0), ('5', 1, 1), ('6', 1, 2),
    ('7', 2, 0), ('8', 2, 1), ('9', 2, 2),
    ('Clear', 3, 0), ('0', 3, 1), ('Enter', 3, 2),
]
for (text, row, col) in buttons:
    def make_action(x=text):
        if x.isdigit():
            press(x)
        elif x == 'Clear':
            clear()
        else:
            submit()
    draw_ellipse_button(btn_frame, text, row, col, make_action)
# Graceful cleanup on exit
def on_closing():
    servo.stop()
    GPIO.cleanup()
    root.destroy()
```

Chapter (12): References

[1] "Statista Research Department," *E-commerce in Egypt,* 2021.

[2] "Cairo Traffic Congestion Study," *World Bank,* 2014.

[3] M. &. Company, "The Sustainability promise of last-mile delivery robots.," 2021.

[4] S. B. w. &. F. D. thurn, "Probabilistic Robotics," *cambridge, MA:MIT press,* 2005.

[5] H. H. S. L. K. M. e. a. Choset, "Principles of Robots Motion: Theory Algorithns, and Implementation," *Cambridgy, MA: MIT press,* 2005.

[6] M. R. a. M. Ceccarelli, "A survey on mechanical solutions for hybrid mobile robots," *Robotics,* vol. 9, no. 2, p. 32, 2020.

[7] M. Koci, "Stress analysis of composite materials used for yacht production through solid work simulation," 2021.

[8] A. C. a. T. P. Bligh, "An approach to functional synthesis of mechanical design concepts: theory, applications, and emerging research issues," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AI EDAM),* vol. 10, no. 4, pp. 313-331, 1996.

[9] A. Antonini, "Mechanical Design of Autonomous Robot for Last Mile Delivery," 2022. [Online]. Available: https://www.polito.it/.

[10] A. G. M. H. S. a. A. K. A. Soltani, "Developing an active variable-wheelbase system for enhancing the vehicle dynamics," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering,* vol. 231, no. 12, pp. 1640-1659, 2017.

[11] M. T. a. P. Dąbek, "Mechanical properties of modern wheeled mobile robots," *Journal of Automation Mobile Robotics and Intelligent Systems,* vol. 13, no. 3, pp. 3-13, 2019.

[12] J. K. a. S. H. L. S. o. I. T. f. I. M. U. M.K. choi, "Survey of Isolation Techniques for Interial Measurment Units," 2017.

[13] I. S. I. S. M. f. G. T. P. Estimation, 12 May 2023. [Online]. Available: https://www.researchgate.net/publication/379877945_Intelligent_Single_IMU_Sensor_Module_for_Gait_Temporal_Parameter_Estimation.

[14] J.-H. P. a. S.-H. L. S.-W.Lee, "A Review of Vibration Isolation Techniques for Inertial Measurement Units," 2020.

[15] B. W. a. j. j. Spilker, "Global Positiong System," Theory and Applications, 1996.

[16] a. E. W. B. Hofmann-Wllenhof.H. Lichtenegger, "GNSS-Global Navigation Satellite Systems," 2007.

[17] R. a. B. R. Sekuler, "Perception:Theory," in *Development and Organization*, 2002.

[18] S. B. W. a. F. D. 2. Thrun, "Probabilistic Robotics," in *MIT press*.

[19] M. M. A. L. a. O. T. n. Bouazizi, "A 2D-Lidar-Equipped Unmanned Robot-Based Approach for Indoor Human Activity Detection".

[20] S. E. Technology, "Co.Ltd," YDLIDAR X2 Data Sheet, 2024. [Online]. Available: https://www.ydlidar.com/Public/upload/files/2024-02-01/YDLIDAR%20X2%20Data%20Sheet%20V1.2%28240124%29.pdf..

[21] D., "ultrasonic Sensor HC-SR04 and Arduino- Complete Guide," How to Mechatronics, 2 Feb 2022. [Online]. Available: https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/. [Accessed 12 May 2023].

[22] D. Andrea, "Battery Management Systems for Large Lithium Ion Battery Packs".

[23] S. F. B. a. D. J. Pack, "Embedded Systems : Design and Applications with The ARM Cortex-M Microcontrollers," 2020.

[24] L.-I. B. B. a. A. R. Korthauer.

[25] J. a. B. J. Fernandes-Madrigle.

[26] M. K. B. G. B. C. K. F. R. F. T. L. J. L. C. N. A. P. L. S. S. a. W. R. Quigley, "an Open source robot operating system," in *ICRA 2009*.

[27] L. j. C., "Robot motion planning," kluwer Academic Publishers, 1991.

[28] W. &. C. R. L. Zeng, "Finding shortest paths on real road networks," in *The Case for A**, International Journal of Geographical InformationScience, 2009, pp. 531-543.

[29] P. C. R. j. K. a. T. M.Suston, "A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV," 2008, pp. 340-345.

[30] S. P. R. W. G. G. a. K. K. P. Narkhede, "Cascaded Complementary Filter Architecture for Sensor Fusion in Attitude Estimation," 2021.

[31] N. M. L. A. a. Y. H. H. Fourati, "Complementary Observer for Body Segments Motion Capturing by Inertial and Magnetic Sensors," IEEE/ASME Transactions on Mechatronics, 2014, pp. 149-157.

[32] D. Dlqrk, "The Protocols. Addison wesly, reading MA," TCP/IP Illustrated, 1990.

[33] P. E. N. N. J. &. R. B. Hart, "A formal basis for the heuistic determination of minimum cost paths," IEEE Transactions on Systems Science and Cybernetics, 1968, pp. 100-107.

Chapter (13): Plagiarism Report

نبذة مختصرة

يتناول هذا المشروع تصميم وتطوير روبوت توصيل ذاتي على الأرصفة لمعالجة أوجه القصور في الخدمات اللوجستية الحضرية والقضايا المستمرة المتعلقة بتسليم الميل الأخير. مع نمو التحضر والتجارة الإلكترونية، تتعرض المدن لضغوط متزايدة لزيادة كفاءة التوصيل مع تقليل الازدحام المروري والآثار البيئية. غالبًا ما تساهم تقنيات التوزيع التقليدية في الازدحام الحضري مع تكبد تكاليف باهظة. تقدم هذه الدراسة حلاً جديدًا على شكل روبوت مدمج ومستقل مصمم للتنقل على الأرصفة وممرات المشاة، مما يوفر بديلاً مستدامًا وفعالًا من حيث التكلفة لخدمات البريد السريع التقليدية.

يحقق روبوت التوصيل على الأرصفة الملاحة في الوقت الفعلي والتعرف على العوائق وتجنب الاصطدام من خلال استخدام نظام متكامل من وأجهزة الاستشعار بالموجات فوق الصوتية. لضمان توصيل دقيق وفعال، يستخدم الروبوت LiDAR أجهزة الاستشعار الحديثة مثل لإنشاء نقاط مسار جغرافية توجهه عبر مناطق معقدة مثل المناطق Bing Maps تخطيط المسار العالمي باستخدام واجهة برمجة تطبيقات الحضرية أو مراكز الخدمات اللوجستية. بين هذه النقاط، يستخدم الروبوت نظام الملاحة المحلية مع خوارزمية VFH+ (Vector Field لتجنب العوائق ديناميكيًا وفي الوقت الفعلي، مما يضمن تشغيلًا سلسًا وآمنًا. صُممت بنية الروبوت البرمجية (Histogram Plus لمعالجة البيانات الحسية بكفاءة وإصدار أحكام مستقلة مع اتباع قواعد مرور المشاة. علاوة على ذلك، يتيح نظام اتصال موثوق به التتبع الفوري ومشاركة المستخدم، مما يُحسّن تجربة التوصيل بأكملها.

تم اختبار أداء الروبوت بدقة في كل من البيئات الحضرية المحاكية والواقعية. وشملت معايير الأداء الرئيسية وقت التوصيل، ودقة التنقل، وكفاءة البطارية، والقدرة على التعامل مع العوائق. تُظهر النتائج أن الروبوت يقلل بشكل كبير من تأخيرات التوصيل الناجمة عن ازدحام المرور، بمتوسط تحسن يصل إلى 30% مقارنةً بعمليات التوصيل التقليدية التي تعتمد على المركبات. نجح الروبوت في التنقل عبر مجموعة متنوعة من البيئات الحضرية، بما في ذلك الأرصفة المزدحمة والتضاريس الوعرة، مُظهرًا قدرة على التكيف والموثوقية.

تُظهر نتائج المشروع الإمكانات التحويلية لروبوتات التوصيل ذاتية التشغيل على الأرصفة في تحديث الخدمات اللوجستية الحضرية. يمكن لهذه الروبوتات توفير تكاليف التوصيل بنحو 60%، وخفض انبعاثات الكربون من عمليات التوصيل بالمركبات، وتحسين موثوقية التوصيل في المناطق ذات الكثافة السكانية العالية. ومع ذلك، يواجه التطبيق المكثف العديد من العقبات، بما في ذلك الامتثال للوائح، ومخاوف السلامة العامة، وتعديل البنية التحتية. ستركز الدراسات المستقبلية على تعزيز قدرات الروبوت على التنسيق بين الروبوتات المتعددة، وزيادة كفاءة استهلاك الطاقة، واستكشاف الآثار الأخلاقية للأنظمة ذاتية القيادة في الأماكن العامة المشتركة.

الأكاديمية المصرية للهندسة والتكنولوجيا المتقدمة

قسم الهندسة الميكانيكية

مشروع تخرج (2): MEC492

روبوت توصيل مستقل على الرصيف لتوفير خدمات لوجستية حضرية فعالة

مشروع تخرج استكمالا لمتطلبات الحصول على درجة البكالوريوس في الهندسة

تخصص الميكاترونيات

إعداد

عبدالحليم شامل عبدالحميد	**2020088**
محمد ممدوح عبدالمنطلب	**2020044**
عمر محمد فؤاد	**2021143**
محمد إنشان عبدالحميد	**2020119**
عبدالرحمن أحمد على	**2020049**
نورا محمد الدفراوي	**2020001**
محمد عبدالحق محمد	**2020084**
يوسف محمد رشدي	**2020163**

إشراف

د. أمين دانيال

د. علا محمد

يونيو 2025