Delivero: Autonomous Delivery Robot

A Project Graduation Document Submitted in Partial Fulfillment of the Requirements for the BSc. Degree in Engineering

Done by:

Amro Gamar Aldwlah Alsammani

Mutaz Hamad Elhaj

Mahmoud mohy aldin Shamad

ALmuatasimbella Khalid Mustafa

Mohammed Anwar Mohammed Fadel

Alargam Mohamed Yagoub

Ayman Yousif kamil Mohamed

Abdalla Ibrahim Gomah Saeed

Ahmed Abdelgalil Abdelrahman

Abdullah Adel Almuallimi

Supervisors :

Assoc. Prof. Dr. Abeer Twakol

Eng. Ali Elhenidy

Faculty of Engineering, Mansoura University

Mansoura, Egypt

2025/2026

Acknowledgment

In the Name of Allah, the Most Gracious, the Most Merciful.

With profound gratitude, we acknowledge the completion of this graduation project. This achievement represents the culmination of dedicated effort and persistent pursuit of knowledge, made possible by the invaluable support of numerous individuals.

Primarily, we offer our sincere praise and thanks to Allah, the Almighty, for His divine guidance and blessings throughout this undertaking.

We express our deepest appreciation to our esteemed supervisor **Assoc. Prof. Dr. Abeer Twakol** , for her exceptional mentorship, unwavering support, and insightful guidance. Her expertise and dedication have been instrumental in shaping the direction and success of this project. We are deeply indebted for her invaluable contributions. We also extend our sincere gratitude to **Eng. Ali Elhenidy** for his valuable contributions to this project. His assistance was greatly appreciated and played a key role in our progress.

We extend our gratitude to the faculty and staff of Mechatronics department for fostering an enriching academic environment conducive to intellectual growth and development.

To our families, we express our deepest gratitude for their constant support and unwavering belief in our potential. Their sacrifices, understanding, and endless encouragement have been a constant source of motivation and strength, especially during challenging times. We are fortunate to have such incredible families who have supported us every step of the way.

This project has provided us with not only technical expertise but also valuable experience in teamwork, problem-solving, and project management. The lessons learned and the skills acquired will undoubtedly serve us well in our future careers.

We dedicate this achievement to all those who have contributed to our success. It is our sincere hope that this project will serve as a foundation for future contributions to the field and a testament to the value of collaborative effort and dedicated pursuit of knowledge. We offer our sincere thanks to all.

Abstract

This project presents the design and development of an Autonomous Mobile Robot (AMR) delivery platform aimed at transforming outdoor logistics and last-meter delivery in controlled environments such as campuses, hospitals, office buildings, and industrial facilities. The system combines robust mechanical engineering, intelligent control, and sensor integration to provide a reliable, flexible, and efficient alternative to manual delivery methods.

The robot features a modular, layered mechanical structure optimized for stability, payload capacity, and serviceability. Designed using advanced CAD tools and validated through Finite Element Analysis (FEA), the chassis ensures high structural integrity under various operational loads. A differential drive system with precision-matched DC motors and traction-optimized wheels enables smooth navigation in outdoor spaces. Integrated suspension and strategic component placement lower the center of mass, ensuring safe motion during turns and stops.

Sensor fusion is central to the platform's autonomy. Equipped with LiDAR, TOF sensors, and an Inertial Measurement Unit (IMU), the robot achieves accurate localization, obstacle avoidance, and environment perception. Real-time data acquisition and decision-making are handled by a dedicated compute unit, allowing autonomous route planning and adaptive behavior in dynamic settings. Careful electronics packaging, cable management, and protection zones enhance reliability and ease of maintenance.

This project addresses the growing need for autonomous outoor logistics in settings where efficiency, hygiene, and contactless delivery are increasingly vital. By reducing human workload, improving delivery speed, and supporting 24/7 operation, the system offers a sustainable and scalable solution for modern service environments. Moreover, its modularity and software-hardware integration allow future upgrades for broader applications, including medical delivery, inventory transport, and facility inspections.

By bridging mechanical design, embedded systems, and intelligent navigation, this project contributes a robust, application-ready platform that reimagines indoor delivery. It reflects our commitment to innovation, automation, and problem-solving in real-world scenarios, paving the way for smarter, safer, and more efficient mobility solution

Table Of Content

List Of Figures

List Of Tables

1 Introduction

In recent years, the food delivery industry has experienced significant growth due to changes in lifestyle, increased reliance on digital platforms, and the widespread use of smartphones. Customers now expect food delivery services to be fast, reliable, and easily accessible, particularly in densely populated and controlled environments such as university campuses, residential compounds, hospitals, and commercial complexes. These environments often require efficient last-meter delivery solutions that can operate smoothly within predefined boundaries. Traditional food delivery systems rely heavily on human couriers, which introduces several operational challenges. These challenges include inconsistent service availability, rising labor costs, delivery delays during peak hours, and increased risks related to safety and human error. Additionally, external factors such as weather conditions, traffic congestion, and access restrictions can negatively affect service quality and reliability.

At the same time, rapid advancements in robotics, autonomous navigation, embedded systems, and control technologies have enabled the development of intelligent mobile robots capable of performing repetitive tasks with high efficiency and accuracy. Autonomous robots are increasingly being adopted in various sectors, including manufacturing, healthcare, logistics, and service industries, due to their ability to operate continuously with minimal human intervention. These systems are designed to perceive their environment, make decisions based on sensor data, and execute tasks with consistent performance. As technology continues to evolve, autonomous robots are becoming more affordable and accessible, which makes them suitable for practical applications in real-world environments.

The motivation for this project stems from the need to leverage these technological advancements to improve food delivery services in controlled environments. By introducing an autonomous food delivery robot, this project aims to reduce dependency on human delivery personnel, enhance operational efficiency, and provide a reliable and contactless delivery solution that meets modern customer expectations. The proposed system aligns with current trends toward automation, smart services, and sustainable operational models. Moreover, the need for autonomous delivery solutions is reinforced by the increasing emphasis on sustainability and operational efficiency. Human-based delivery models not only consume significant labor resources but also contribute to traffic congestion and environmental pollution, particularly when multiple delivery trips are made within a small geographic area. In contrast, electric-powered delivery robots can operate with lower energy consumption and reduced emissions, especially when used in controlled environments where routes can be optimized. The development of such systems also reflects a broader shift toward smart infrastructure, where digital services are integrated with physical environments to provide seamless user experiences. For instance, university campuses and residential compounds are increasingly adopting smart access control, surveillance, and connectivity systems, which create a supportive environment for autonomous robots. This project therefore aligns with the global trend toward intelligent and sustainable services, and seeks to demonstrate a practical implementation of these concepts within realistic operational boundaries.

1.1 Problem Statement

Despite the widespread use of food delivery applications, current delivery systems continue to face several limitations, particularly in controlled environments. Human-based delivery services are often affected by inconsistent performance, limited availability, and high operational costs. During peak demand periods, delivery delays become more frequent, leading to reduced customer satisfaction and increased service complaints. In addition, human couriers may face difficulties navigating complex or restricted environments such as university campuses and residential compounds, where access control, safety regulations, and pedestrian traffic must be considered. These challenges shown in Figure. 1.1 highlight the inefficiency of traditional delivery approaches in such.

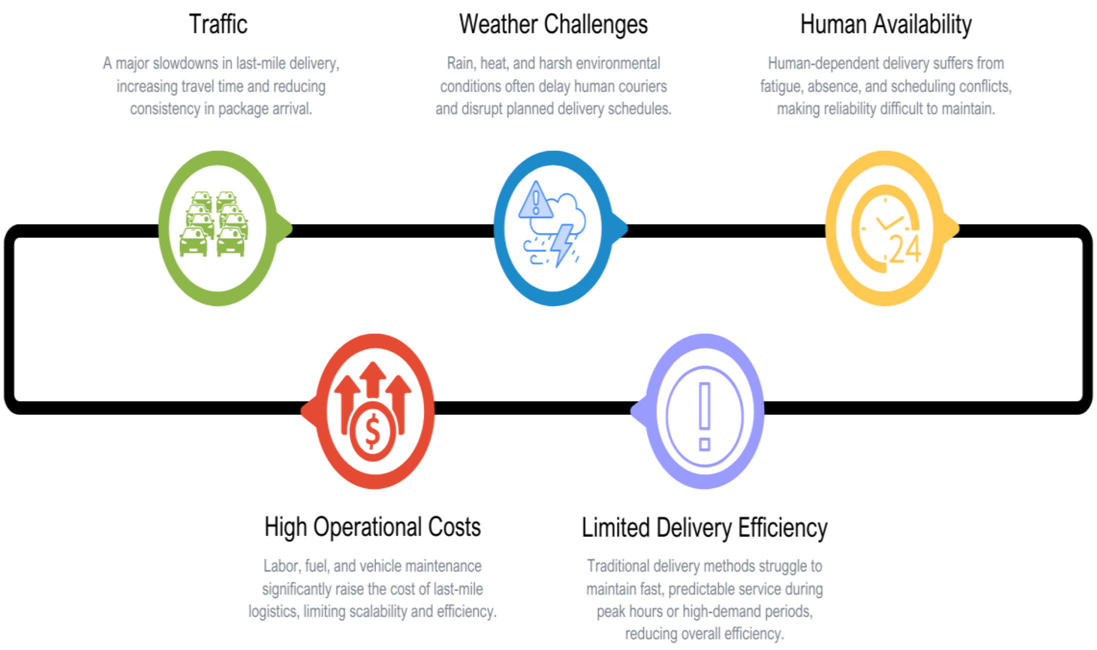

Traffic
A major slowdowns in last-mile delivery, increasing travel time and reducing consistency in package arrival.
Weather Challenges
Rain, heat, and harsh environmental conditions often delay human couriers and disrupt planned delivery schedules.
Human Availability
Human-dependent delivery suffers from fatigue, absence, and scheduling conflicts, making reliability difficult to maintain.
24
High Operational Costs
Labor, fuel, and vehicle maintenance significantly raise the cost of last-mile logistics, limiting scalability and efficiency.
Limited Delivery Efficiency
Traditional delivery methods struggle to maintain fast, predictable service during peak hours or high-demand periods, reducing overall efficiency.

Figure. 1.1 delivery challenges

Furthermore, existing delivery systems lack a fully integrated autonomous solution that combines mechanical design, electrical and power systems, embedded control, and navigation specifically optimized for food delivery tasks. This gap creates a need for a reliable and autonomous delivery platform that can operate efficiently within predefined environments while ensuring safety, consistency, and cost-effectiveness. In addition to operational challenges, the reliability and safety of delivery systems are critical concerns in controlled environments. Human couriers may face risks such as accidents, theft, or personal injury, especially when navigating unfamiliar areas or during night-time operations. Moreover, controlled environments often require adherence to specific safety regulations, access restrictions, and predefined pathways. This creates a need for a system that can operate autonomously while respecting boundaries and ensuring secure interaction with users. Autonomous robots can address these concerns by providing predictable behavior, controlled speed, and standardized delivery protocols. However, the challenge lies in designing a system that can maintain high reliability while adapting to dynamic conditions, such as sudden obstacles, changing pathways, and varying environmental conditions.

1.2 Project Objectives

The primary objective of this project is to design and develop an autonomous food delivery robot capable of operating efficiently and safely in both indoor and outdoor environments. The project aims to present a practical engineering solution that integrates mechanical design, electrical and power system development, embedded control, perception, and autonomous navigation into a single unified system. By achieving this integration, the proposed robot seeks to reduce reliance on human delivery personnel while enhancing delivery efficiency, reliability, and operational consistency across different environments.

Furthermore, the project focuses on developing a robust electrical and power management system that supports continuous operation under varying environmental conditions. Another objective is to implement a control and sensing framework that enables the robot to autonomously navigate structured and semi-structured environments while detecting and avoiding obstacles in real time. In addition, the project aims to validate the system through simulation and real-world experimental testing, thereby demonstrating the feasibility, adaptability, and effectiveness of the proposed autonomous food delivery solution. The project also aims to develop a modular and scalable architecture that allows future enhancements and integration with additional features. This includes the possibility of adding advanced perception modules, such as computer vision for identifying obstacles and detecting humans, as well as communication modules that enable real-time coordination with cloud services or other robots. By adopting a modular design approach, the system can be upgraded with minimal modifications, which supports future research and development efforts. Moreover, the project emphasizes the importance of user experience, ensuring that the delivery process is simple, intuitive, and secure for both the customer and the operator.

1.3 Scope of the Project

This project focuses on the design and implementation of an autonomous mobile robot for food delivery in both indoor and outdoor environments. The system is intended to operate in controlled settings such as university campuses, residential compounds, and commercial districts where the environment is semi-structured and manageable. The scope includes mechanical design, embedded system development, sensor integration, navigation, and user interface development. The project aims to demonstrate the functional feasibility of the proposed delivery system rather than pursuing large-scale commercialization or industrial deployment. The system is evaluated through laboratory testing and controlled environment experiments to validate performance and identify potential improvements. The project does not address long-distance outdoor navigation or integration with city-wide logistics infrastructures. Instead, it focuses on creating a reliable and practical prototype that can navigate both indoor corridors and outdoor pathways, serving as a foundation for future enhancements and scalability.

The project scope also includes the development of a simple but effective communication interface between the robot and the user. This interface is intended to support order tracking, status updates, and delivery confirmation. The interface will also provide safety notifications and allow users to interact with the robot in a controlled manner. Additionally, the scope includes the selection of appropriate sensors and actuators to ensure robust operation in both indoor and outdoor conditions. While the project does not focus on large-scale deployment, it aims to deliver a functional prototype that can serve as a baseline for future scaling and integration into larger logistics systems.

1.4 Project Constraints

The development of an autonomous food delivery robot is subject to several constraints that influence the design and implementation process. One of the primary constraints is the budget, as the project must rely on affordable components and materials while maintaining functional performance. The choice of sensors and actuators is therefore limited by cost considerations, which may affect the accuracy of localization and obstacle detection. Time constraints also play a significant role, as the project is completed within a limited academic timeframe, requiring efficient planning and prioritization of tasks. Another constraint is the availability of testing environments, since the robot must be evaluated in both indoor and outdoor settings, which requires access to suitable facilities and controlled areas. Moreover, safety and regulatory considerations impose restrictions on the robot's speed, size, and operational behavior, particularly in public spaces. The system must ensure safe interaction with humans and avoid causing disruptions or hazards. Finally, technical constraints such as limited processing power, communication range, and battery capacity must be considered during the design phase to ensure reliable operation in both indoor and outdoor environments.

Another important constraint is the integration of multiple subsystems, which requires careful coordination between mechanical, electrical, and software teams. The system must maintain compatibility between sensors, microcontrollers, motor drivers, and communication modules. Any mismatch or delay in integration can result in reduced performance or system failure. Additionally, the project must consider the reliability of power supply and battery management, especially during extended operation in outdoor conditions. Thermal management and protection against dust and moisture are also important considerations for outdoor use. Finally, the project must comply with safety guidelines to avoid any potential hazards to users and pedestrians.

1.5 Structure of the Report

This report is structured to present the development of the autonomous food delivery robot in a systematic and coherent manner. Chapter 1 introduces the project background, motivation, problem statement, objectives, scope, and constraints, providing a clear foundation for the subsequent chapters. Chapter 2 reviews the relevant literature and existing solutions in the field of autonomous delivery systems, including the key technologies and methodologies used for navigation, localization, obstacle detection, and human-robot interaction. This chapter aims to highlight the research gap and justify the technical choices made in the project. Chapter 3 describes the design and implementation of the proposed system, covering the mechanical design, embedded hardware architecture, sensor selection, and software framework. It details the integration process of the robot's components and explains how the system achieves autonomous navigation and safe operation in both indoor and outdoor environments. Chapter 4 presents the experimental setup and testing procedures used to evaluate the system's performance. It includes descriptions of the test environments, the metrics used for evaluation, and the data collection process. The chapter also discusses the challenges encountered during testing and the modifications made to improve performance. Chapter 5 analyzes the results obtained from the experiments, comparing the performance metrics with the project objectives. It provides a critical evaluation of the system's strengths and limitations, and proposes improvements for future development. Finally, Chapter 6 concludes the report by summarizing the main findings, reflecting on the achievements of the project, and outlining recommendations for future work, including potential enhancements, scalability considerations, and real-world deployment scenarios.

Each chapter of this report is designed to build upon the previous one, creating a clear and logical narrative that guides the reader through the project development. The report begins with a detailed analysis of the problem and motivation, then reviews the current state of technology and research in autonomous delivery systems. The design chapter explains the engineering decisions and implementation steps, while the testing and results chapters provide evidence of the system's functionality and performance. Finally, the conclusion chapter reflects on the project outcomes and suggests future improvements, ensuring that the report not only documents the work performed but also provides a roadmap for continued development and potential commercialization

2 Market and Literature Survey

2.1 Overview of Autonomous Delivery Robots

Autonomous Delivery Robots (ADRs) as shown Figure. 2.1 are mobile robots designed to transport goods to customers with minimal human intervention, typically in the *last-mile* of the delivery chain. These robots are envisioned to transform last-mile logistics by improving efficiency, reducing delivery times and costs, and enhancing sustainability. Recent advances in sensing, artificial intelligence, and robotics have accelerated the development and adoption of ADRs. Notably, the COVID-19 pandemic further spurred interest in contactless delivery solutions, positioning delivery robots as a timely innovation for safe and convenient service.

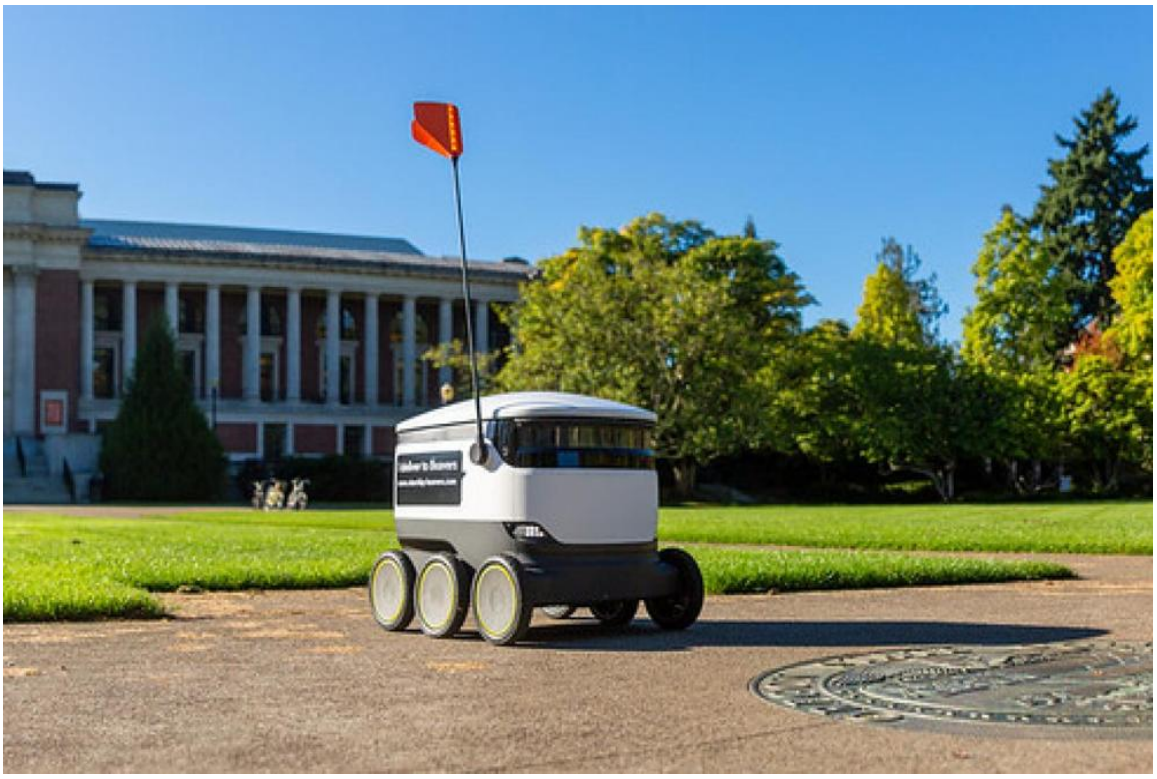


Figure. 2.1 A small autonomous delivery robot (Starship Technologies)

ADRs generally operate on sidewalks or pedestrian areas at low speeds (around walking pace) and are designed to navigate around obstacles and people. They leverage technologies like GPS for coarse positioning and a suite of onboard sensors (cameras, ultrasonic, radar, etc.) for local environment perception and obstacle detection. By traveling on sidewalks and using electric power, these robots can reduce road traffic and emissions, offering a more sustainable delivery method compared to traditional vans. In operation, ADRs benefit various stakeholders: customers enjoy quick, on-demand deliveries; businesses can reach more clients with lower labor costs; and communities see fewer delivery vehicles on streets and reduced pollution.

It's important to distinguish ground-based delivery robots from aerial delivery drones. Both are autonomous delivery vehicles, but each has pros and cons. Drones can travel faster and in straightline paths, but ADRs can carry heavier payloads, make multiple deliveries per trip, and use far less energy in most conditions. Ground robots also face fewer regulatory hurdles in many regions, since low-speed sidewalk robots are perceived as safer for public spaces than flying drones. For these reasons, our survey (and project) focuses on ground-based autonomous delivery robots operating in outdoor pedestrian environments.

2.2 Evolution of Autonomous Delivery Robots

Autonomous delivery robots have evolved from visionary concepts to real-world deployments over the span of several decades. Early ideas of robotic couriers appeared in science fiction long before the technology existed. In practice, the foundation was laid in the late 20th century through research in autonomous mobile robots and vehicles.

Early prototypes (1980s–1990s): Initial efforts focused on indoor or controlled environments. In the 1980s, robotics pioneers developed autonomous mobile platforms like *Shakey* (the first mobile robot to reason about its actions) shown Figure. 2.2 and experimental warehouse robots. The emphasis was on basic navigation and task execution, and outdoor delivery applications remained out of reach due to limited computing power and sensor technology.

Figure. 2.2 shakey the robot

By the 1990s, interest in automating logistics grew – major couriers experimented with automated sorting and handling systems, although true last-mile delivery robots had not yet emerged. Notably, in 1995 the Carnegie Mellon NavLab project demonstrated a van driving autonomously across America (with human supervision), and in 1997 NASA's **Sojourner** rover as shown Figure. 2.3 autonomously traversed Mars – achievements that proved mobile robots could negotiate complex terrains. Such milestones informed later designs of ground delivery robots. Still, through the 1990s, most “delivery” robots were confined to hospitals (e.g. mail and medicine carts) or indoor settings, and outdoor units were teleoperated or experimental.

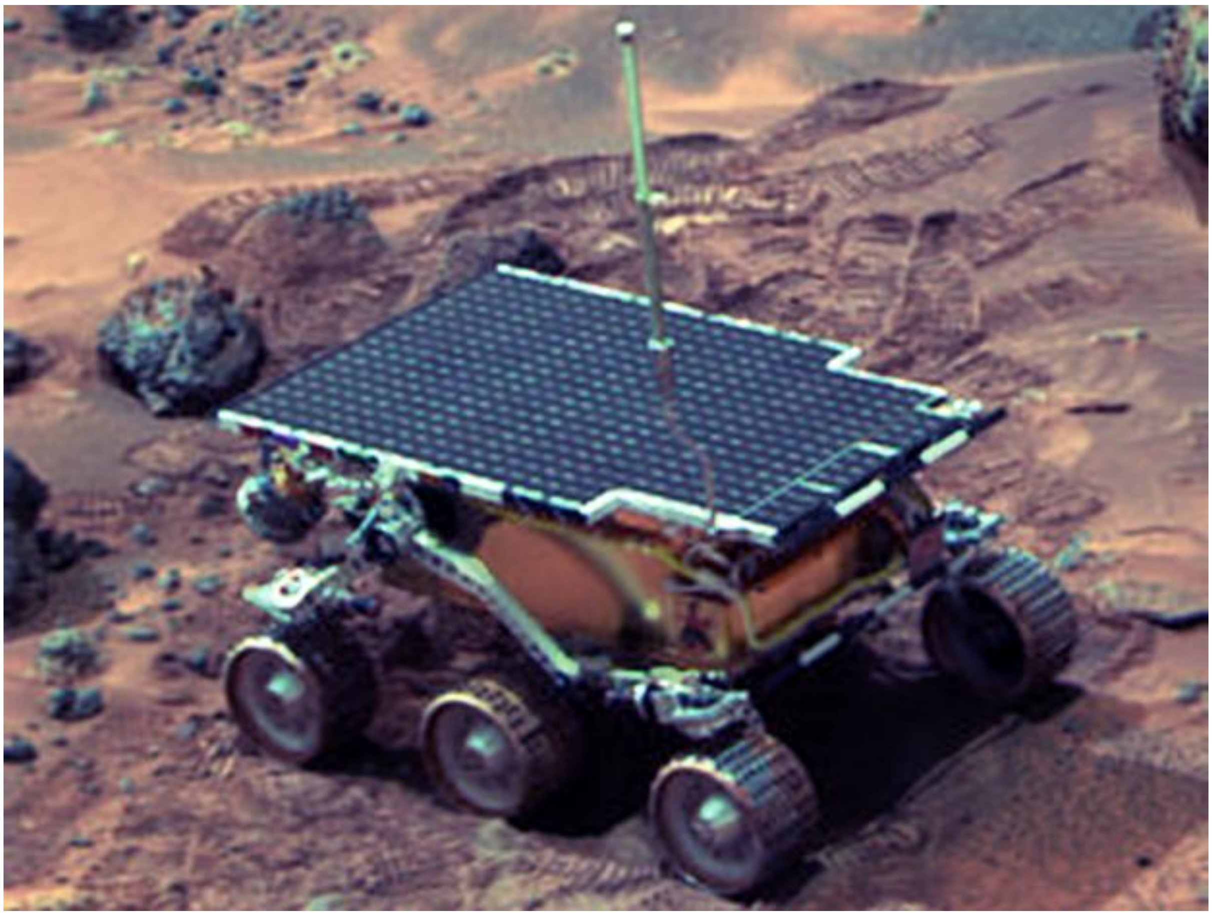


Figure. 2.3 NASA's Sojourner rover

Technological turning points (2000s–early 2010s): The 2000s saw rapid advances in sensors, artificial intelligence (AI), and machine learning that set the stage for autonomous deliveries. The 2004–2007 DARPA Grand Challenges spurred breakthroughs in self-driving vehicle technology, demonstrating that AI-powered machines could navigate real roads. By the early 2010s, these innovations trickled down to smaller delivery robots. A landmark moment came in 2013 when Amazon announced *Prime Air*, a project to deliver packages by autonomous drones. Although drone delivery faced regulatory hurdles and was not ground-based, this bold plan from a major retailer thrust the concept of autonomous last-mile delivery into the public eye. In 2014, **Starship Technologies** – founded by two Skype co-founders – introduced one of the first modern ground delivery robots. Starship’s six-wheeled robots were designed to travel on sidewalks, using GPS, computer vision, and AI to navigate around pedestrians and obstacles while carrying small parcels or food. Early trials in 2015–2016 in Estonia and London proved the concept’s viability. By **2016**, Starship had launched pilot deliveries in several cities, marking the first real autonomous **outdoor** delivery robots operating in public spaces. This period also saw growing startup activity worldwide: for example, **JD.com** in China built its first delivery robots in 2016 for campus deliveries, and **Yandex** in Russia began testing delivery rovers in 2017. These pioneers benefited from the maturation of LIDAR, affordable cameras, and embedded computing, as well as increasing internet connectivity for remote monitoring.

Mainstream pilots and commercialization (late 2010s): By 2018–2019, autonomous delivery robots moved from isolated experiments to mainstream pilot programs. Many companies secured significant funding around **2018** to develop delivery robots, reflecting investor confidence in the technology. The robots also diversified in form: *sidewalk drones* like Starship's remained popular, while others pursued larger street-driving robots. Notably, **Nuro**, founded by former Google engineers in 2016, developed a small autonomous delivery vehicle for roads. By 2020 Nuro's R2 vehicle became the first fully driverless delivery car approved on public roads in the U.S., used for grocery and pizza deliveries. Another milestone was the unveiling of **FedEx Roxo** in 2019 – a robot van for same-day deliveries that could climb curbs and even steps, leveraging technology from DEKA Research. **Amazon** also joined the fray: after exploring delivery drones, Amazon launched its ground robot **Scout** in 2019, deploying six-wheeled autonomous coolers to deliver packages in suburban U.S. neighborhoods. Scout expanded trials to multiple cities and even prompted Amazon to open an autonomy R&D center in 2021 to improve robot navigation. However, it also exemplified the challenges in this field – Amazon had to *halt* the Scout program in 2022 after technical and market difficulties, refocusing its efforts. Meanwhile, smaller startups like **Kiwibot**, **Serve Robotics** (Postmates' spin-off), **Starship**, and **Yandex** steadily grew their operations. A significant use-case emerged in food delivery: by early 2019, Starship robots were delivering food on a university campus (George Mason University) as part of the dining services – the first campus to integrate robots into meal plans. This success was replicated at dozens of colleges. Food apps such as Grubhub and Uber Eats began partnering with robot firms (e.g. Yandex, Serve) to offer robotic delivery options in certain locations. Around the same time, government bodies started adapting regulations for ground robots: for instance, several U.S. states (Virginia, Idaho, etc.) passed laws around 2017–2019 legalizing sidewalk delivery robots under specific weight/speed limits. These regulatory frameworks, along with improving technology, accelerated deployment.

Acceleration during the pandemic and current state-of-the-art (2020s): The COVID-19 pandemic in 2020 became an unexpected catalyst for autonomous delivery. Demand for contactless delivery services spiked, and robots offered a hygienic alternative for last-mile logistics. As a result, many pilot programs fast-tracked deployment. By late 2020, Starship had rolled out robots in cities across the US and Europe, and was operating what was claimed to be the world's largest autonomous delivery fleet in Milton Keynes, UK. College campuses and city communities that restricted human interactions welcomed delivery robots to ferry food and supplies. Hundreds of new robots were deployed in 2020–2021, and public acceptance of the technology grew. Studies indicate that human-robot interactions have been largely positive, with novelty giving way to routine usage in some towns. From **2021** onward, the industry saw *consolidation and scaling*: larger corporations partnered with or acquired robot startups, and logistics companies integrated robots into their services. For example, Uber Eats deployed Serve and Cartken robots in certain U.S. cities, and major retail chains (Walmart, Kroger) ran pilot deliveries with Nuro vehicles. By **2023**, Starship Technologies alone reported over **5 million** autonomous deliveries completed worldwide using its fleet of 2,000+ robots a testament to how far the technology has come in a short time. The capabilities of ADRs have also improved markedly: modern units use advanced AI for path planning, multi-modal sensor fusion (HD cameras, radar, ultrasonics, and LiDAR) for 360° obstacle avoidance, and are connected via 4G/5G for real-time monitoring. Many can operate for 8–12 hours on battery and carry up to 10–40 kg payloads in secure, temperature-controlled compartments. In parallel, autonomous **aerial drones** have continued to evolve, finding niches in medical supply drops and rural deliveries (as seen with Zipline and Wing), although regulatory constraints mean ground robots currently have a faster path to urban adoption.

A key technological shift in the evolution of ADRs has been the move from human teleoperation to **AI-based autonomy**. Early delivery robots were often remote-controlled or required a human supervisor on standby for difficult situations. Even today, most delivery robots are monitored by teleoperators who can intervene if a robot gets stuck or faces an unusual obstacle. However, the degree of autonomy has greatly increased: thanks to machine learning, computer vision, and better mapping, the latest robots handle the vast majority of trips without human input, relying on high-definition maps and on-board intelligence to make decisions in real time. Remote operators now typically oversee multiple robots and only occasionally take direct control (for example, if a robot encounters a construction zone it cannot navigate). This trend towards greater autonomy is expected to continue, reducing the dependence on teleoperation over time as robots learn from each encounter (many companies use data from remote interventions to improve their AI models).

In summary, the evolution of autonomous outdoor delivery robots has progressed from rudimentary research vehicles to sophisticated, AI-driven machines in active service. Starting as remote-controlled carts and theoretical concepts decades ago, ADRs have passed key milestones – early DARPA challenges, the first commercial pilots in mid-2010s, rapid expansion during COVID-19, and now integration into everyday logistics networks. Current state-of-the-art systems, whether sidewalk-sized drones or larger road-going pods, represent the convergence of advances in robotics, AI, and automotive technology. They are increasingly **commercially viable**, evidenced by partnerships between robot developers and retailers or delivery companies (for instance, the 2025 Noon–Yango partnership to deploy robots at scale in Dubai). Looking forward, experts predict continued growth and innovation in this field, with ADR fleets likely becoming a common sight in smart cities worldwide by the end of the decade. Major research and development efforts are now focused on improving reliability (all-weather operation), safety, and efficiency of these robots, as well as on integrating them with **smart city infrastructure** and traffic systems. Ultimately, the historical trajectory of autonomous delivery robots – from early trials to state-of-the-art – underscores a broader shift in last-mile logistics, where intelligent robots play an increasingly prominent role in delivering goods safely and efficiently to customers.

2.3 Existing Commercial Solutions

Multiple companies across the world have developed and deployed autonomous delivery robots in real-world settings. In the United States and Europe, one of the leading examples is **Starship Technologies**, which operates fleets of six-wheeled delivery robots. Starship's robots have been used on numerous university campuses and city sidewalks, accumulating millions of autonomous deliveries. By late 2024, Starship had a fleet of over 2,000 robots worldwide and surpassed **7 million** deliveries, demonstrating the viability of the concept at scale. The Starship robot can carry about 10 kg of goods (e.g. groceries or meals) and navigate autonomously 99% of the time, with remote human supervisors available if needed. Each unit travels at ~6 km/h and is equipped with a sensor suite (around 12 cameras along with GPS, ultrasonic sensors, radar, etc.) to perceive its surroundings and safely avoid obstacles. These robots have become a common sight in some communities, completing over **9 million** deliveries globally as of 2025.

Another major player is **Amazon** with its *Amazon Scout* robot displayed Figure. 2.4 cooler-sized, wheeled delivery device. Scout was introduced in 2019 to autonomously deliver Amazon packages on sidewalks. It featured cameras and ultrasonic sensors to navigate around pedestrians and pets. Amazon piloted Scout in several U.S. neighborhoods, touting it as a way to meet rising e-commerce demand with less road congestion. However, after a period of field testing, Amazon scaled back the Scout program in 2022, indicating that certain aspects did not meet customer needs. Despite this setback, Amazon's experiment with Scout demonstrated the potential for integrating ADRs into retail logistics and provided valuable lessons for the industry.

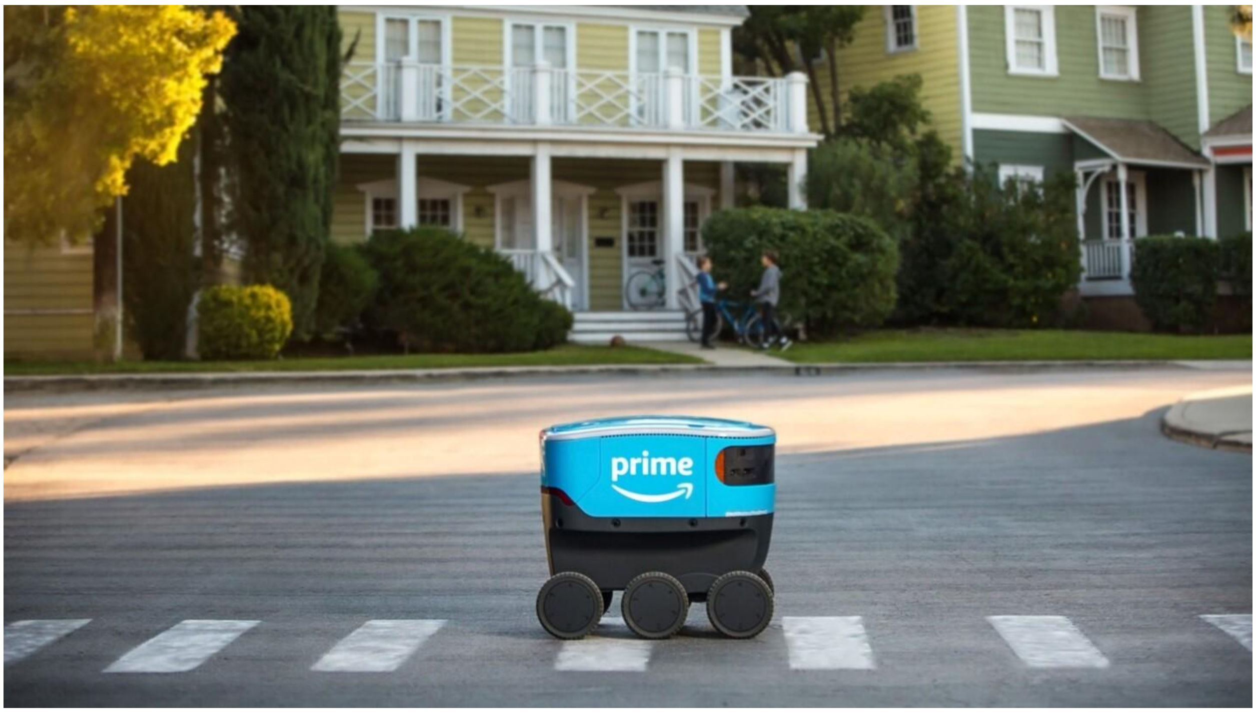


Figure. 2.4 Amazon Scout delivery robot

In the realm of self-driving delivery *vehicles* (larger than sidewalk robots), **Nuro** has developed a small autonomous van for last-mile delivery. Nuro's R2 vehicle is about the size of a golf cart and is **entirely driverless**, built specifically to carry goods (with no space for passengers). It is equipped with an array of sensors including 360° cameras, short- and long-range radar, ultrasonic sensors, and even thermal imaging, allowing it to detect pedestrians and traffic at a distance. The R2 can drive at speeds up to ~40 km/h (25 mph) on residential streets[18]. Nuro has partnered with companies like Domino's (for pizza delivery), FedEx, and Kroger grocery stores to pilot this service in U.S. cities. This shows another approach to autonomous delivery – using road-going vehicles for larger payloads – complementing the smaller sidewalk robots.

Chinese tech companies have also invested heavily in autonomous delivery. Notably, **Alibaba** developed the **Xiaomanlv** ("Small Donkey") delivery robot to handle packages from its e-commerce platform. These robots can carry about 50 packages at once and have been deployed on university campuses and urban neighborhoods in China. Alibaba reported that its fleet of Xiaomanlv robots had completed over **10 million** parcel deliveries as of early 2022 a remarkable scale achieved in a short time. Similarly, **JD.com** (another Chinese e-commerce giant) and other firms have introduced autonomous delivery carts that navigate sidewalks or gated communities to drop off parcels. The success in China is aided by controlled campus environments and strong demand for e-commerce deliveries, allowing rapid accumulation of operational miles and data.

The Middle East is beginning to see its own trials of delivery robots. For example, in the United Arab Emirates, the e-commerce platform **noon** has partnered with **Yandex** (via its Yango unit) as shown Figure. 2.5 to deploy autonomous delivery robots in Dubai. In a 2025 pilot, Yandex's wheeled robots (branded for noon) as displayed in figure 3 began operating in residential communities, delivering groceries and essentials to customers of the "noon Minutes" service. These robots are fully electric and navigate public walkways, autonomously planning routes and avoiding pedestrians. During the Dubai trials they covered over **1,500 km** in autonomous mode, and gained regulatory approval from the Roads and Transport Authority (RTA) to operate on city sidewalks. The initiative in Dubai – along with similar pilots in Abu Dhabi and Saudi Arabia – reflects a growing interest in ADRs in the Middle East, driven by smart city ambitions and the need for efficient delivery in congested urban areas. It's worth noting that local startups in the region (e.g. Egypt's MARSES Robotics) have mostly focused on indoor delivery robots (for hospitals, hotels, etc.), while outdoor delivery robots are so far being introduced via collaborations with global technology providers.

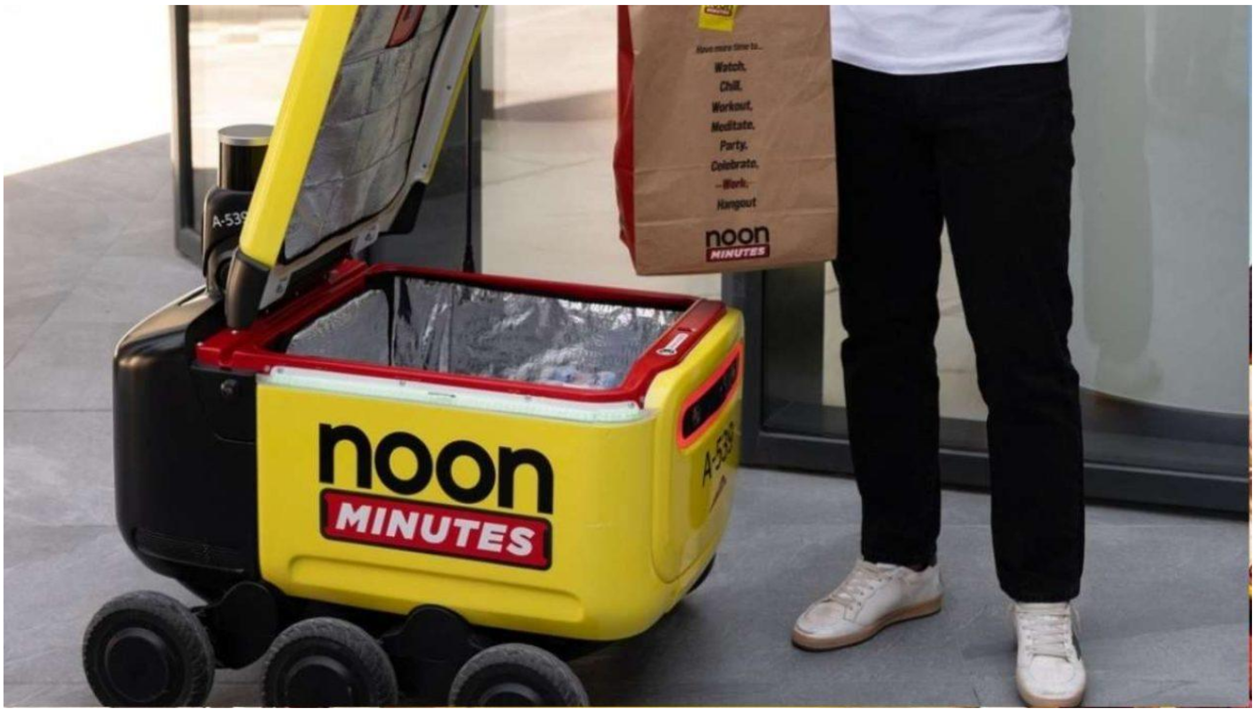


Figure. 2.5 Yandex's wheeled robot branded for noon

Overall, the market survey shows a diverse range of commercial ADR solutions worldwide. From small sidewalk drones like Starship and Scout to larger autonomous vans like Nuro, and across regions from the U.S. and Europe to China and the GCC, autonomous delivery robots are transitioning from experimental pilots to operational services. The most successful deployments so far have been in controlled environments (campuses, planned communities) and in partnership with delivery services or retailers. These real-world implementations provide valuable case studies on the capabilities and limitations of current ADR technology, informing both industry and academic progress.

The Middle East region has begun to embrace autonomous delivery robots (ADRs) through pilot programs and research initiatives, aligning with national innovation strategies. **Saudi Arabia** has taken bold steps in this domain. For example, King Abdullah University of Science and Technology (KAUST) launched a *"living lab"* pilot in late 2021 using a purpose-built delivery robot called **UNO Commuter as shown** Figure. 2.6. This vehicle, equipped with multiple LiDAR sensors, cameras, DGPS, and an onboard control unit, provides daily last-mile package delivery from the campus mailroom to residential areas. KAUST residents receive e-commerce orders (from partners like Amazon and Noon) delivered autonomously, interacting with the vehicle via SMS or a touchscreen to retrieve their packages. The project, run in partnership with the National Digital Transformation Unit (NDU), logistics carrier Alshrouq Express, and Hong Kong-based Teksbotics, demonstrates a local collaboration between academia and industry to realize cost-effective ADR solutions. This initiative supports Saudi Arabia's Vision 2030 goals by using the KAUST campus as a testbed before expanding such technologies nationally.

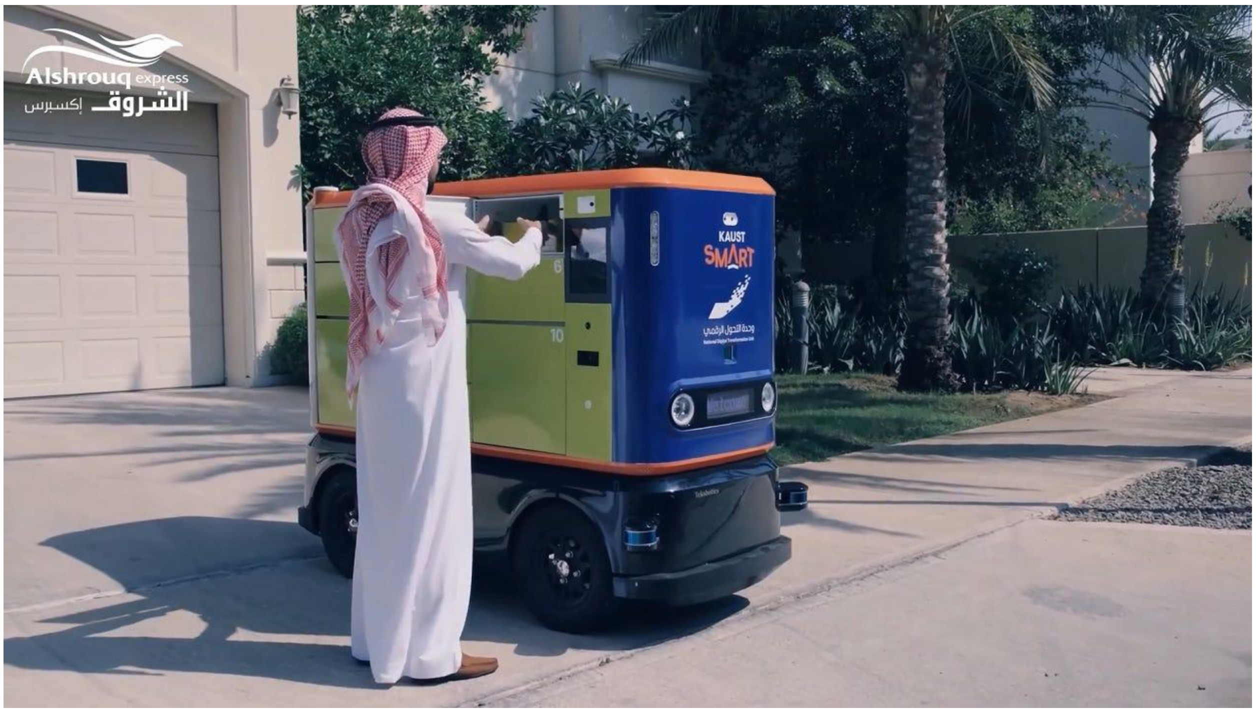


Figure. 2.6 The KAUST Smart "UNO Commuter

Beyond academia, **commercial pilots in Saudi Arabia** are underway. In 2025, the Transport General Authority (TGA) launched the Kingdom's first *commercial* autonomous food delivery service at the ROSHN Business Front in Riyadh. This pilot, a collaboration with ROSHN (a major real estate developer) and Jahez (a food-delivery platform), deployed five self-driving delivery robots to shuttle meals from nearby restaurants to offices within a closed campus. Each robot is outfitted with over 20 sensors, 6 cameras, GPS navigation, and even climate-controlled compartments to keep food fresh in high temperatures. The robots operate during working hours, navigating pedestrian areas and interacting safely with people (displaying messages like "I need help" when necessary). This pilot has dual aims: improving delivery efficiency (reducing times and labor needs) and supporting sustainability by cutting vehicle trips and emissions. It is estimated that each robot could eliminate up to 1.2 tons of CO_2 emissions annually if scaled up. Following its initial success, plans were announced to expand autonomous deliveries to shopping malls and residential communities, further integrating ADR services into daily life in line with Vision 2030's smart city objectives.

Despite these advances, **the Middle East's ADR adoption is still nascent** compared to North America, Europe, and Asia. Industry analyses note that the Middle East and Africa are in the early stages of ADR deployment, with relatively low but rising numbers of robots in service. Supportive government vision documents and initiatives are helping drive this growth. Saudi Arabia, the UAE, and Egypt have explicitly prioritized logistics automation and autonomous transport in their national strategies. For instance, Saudi Vision 2030 and the UAE's Smart Mobility Strategy 2030 encourage trials of drones and ground robots for deliveries as a way to improve efficiency and sustainability. In practice, this has led to projects like Saudi Arabia's planned **drone delivery corridors** (connecting logistics hubs to hospitals and homes) and the development of NEOM's *Oxagon* logistics hub where fleets of delivery robots and drones are envisioned to handle last-mile distribution in a fully autonomous, zero-emission system. In the UAE, postal and logistics companies (Emirates Post, Aramex) are complementing robot trials with electric vans and routing AI, illustrating a broader push for smart, low-carbon last-mile solutions.

Egypt, while not yet hosting high-profile public ADR pilots, is likewise laying groundwork for autonomous delivery. The Egyptian government's digital transformation and smart city initiatives (e.g. the New Administrative Capital's smart infrastructure) recognize the role of automation in logistics. Local tech talent and companies are beginning to contribute to the field. For example, the French automotive supplier Valeo, which operates a large R&D center in Cairo, is co-developing an advanced last-mile delivery robot in partnership with UAE-based W Motors as of 2025 The goal is to design and manufacture autonomous delivery droids tailored to regional needs, leveraging Valeo's expertise in electrification and autonomy together with local manufacturing capacity in the UAE. Additionally, regional robotics startups are emerging – *MARSES Robotics* in Egypt, for instance, builds autonomous service robots for hospitals and hospitality, indicating a growing ecosystem of robotics innovation in the country. While Egypt has yet to see commercial street delivery robots, the presence of Noon (which expanded its e-commerce platform to Egypt in 2019) and similar players suggests that pilot programs may eventually extend to Egyptian cities as the technology matures. Overall, the Middle East's recent projects – from KAUST's campus deliveries to Dubai's city pilots – demonstrate a clear momentum toward integrating ADRs into the local delivery landscape. These efforts are early but significant steps, signaling that the region is positioning itself to adopt autonomous outdoor delivery solutions in the near future in line with global trends.

2.4 Review of Academic Research

The rapid development of commercial delivery robots has been paralleled by a growing body of academic research. In recent years, scholars have investigated a variety of technical and logistical challenges associated with autonomous delivery robots (ADRs). A 2022 comprehensive review by Srinivas *et al.* noted that an increasing number of studies have examined different aspects of ADR-based delivery operations, though a holistic literature survey was lacking until then. Key research themes can be broadly categorized into: operational logistics, navigation and control, human-robot interaction, and socio-technical considerations.

Operational logistics and planning: Researchers have looked at how to efficiently integrate ADRs into the supply chain. This includes problems like routing and scheduling for fleets of delivery robots, optimizing the allocation of delivery tasks, and fleet size/infrastructure planning. For example, algorithms have been proposed to determine optimal paths and schedules for multiple robots delivering packages in a neighborhood, often aiming to minimize total distance or delivery time. These problems extend classic vehicle routing problems with new constraints specific to robots (limited battery life, lower speeds, need to return to base, etc.). Some studies also explore hybrid delivery networks (robots plus human couriers or drones) and how to position charging stations or dispatch hubs for the robots in a city. Overall, academic research in this area seeks to maximize the efficiency benefits of ADRs in last-mile logistics.

Navigation, perception, and control: A significant portion of the literature addresses the core robotics challenges in autonomous navigation. ADRs must perform *Simultaneous Localization and Mapping (SLAM)* to understand their environment, plan paths, and avoid obstacles in real time. Researchers have developed and tested various navigation algorithms suitable for delivery robots, often focusing on urban pedestrian environments. For instance, vision-based SLAM approaches have been explored, where the robot uses camera input (sometimes combined with LIDAR or other sensors) to build a map of sidewalks and localize itself. Techniques like 3D mapping, NavMesh path planning, and dynamic obstacle avoidance have been demonstrated to enable robots to handle complex environments with ramps, curbs, and moving people. Another line of work examines motion control strategies that ensure smooth and safe navigation; for example, adapting traditional wheeled robot controllers to the outdoor context of uneven terrain and pedestrian traffic. In summary, academic contributions in this category provide the algorithms and systems that make autonomous navigation reliable for delivery applications.

Human-robot interaction and acceptance: Since delivery robots operate in public spaces and often come into contact with end-users, researchers are studying how people perceive and interact with them. This includes user experience research on whether customers find robots convenient and trustworthy for deliveries, as well as how pedestrians react when encountering robots on the sidewalk. Studies have measured technology acceptance factors – such as perceived safety, usefulness, and social comfort – which influence whether people will embrace ADR services. Additionally, there is research on the *social intelligence* of delivery robots: for instance, making robots follow social norms (like queueing behind pedestrians, or communicating intentions with lights or sounds). Some experiments involve robots politely asking people to make way or using expressive cues to appear friendly, aiming to improve coexistence in community settings. These human-centered studies are important for guiding the design of ADR behavior and appearance so that they integrate smoothly into everyday life.

Legal and ethical considerations: Academic discourse has also identified regulatory and ethical issues around autonomous delivery. Topics include the need for new traffic rules or sidewalk regulations to accommodate robots, liability in case of accidents, and data privacy (since robots may record their surroundings). Different countries and cities are formulating policies for delivery robots, and researchers are analyzing these frameworks and recommending best practices. There are also discussions about labor impacts (how robots might affect delivery jobs) and accessibility (ensuring robots do not impede people with disabilities on sidewalks). While this is more of a policy and social science angle, it interfaces with technical research to ensure ADR deployment is done responsibly and inclusively.

In summary, academic research on autonomous delivery robots is multidisciplinary, spanning robotics, operations research, and human factors. The literature so far has proposed solutions to many technical challenges and shed light on user perspectives, but also points out open issues. For example, Srinivas *et al.* (2022) highlight that despite the progress, there remain research gaps in fully optimizing ADR operations and handling future challenges. As ADRs move from pilot projects to broader usage, ongoing research will be crucial in refining the technology (making navigation more robust, extending battery life, etc.) and ensuring these robots effectively serve both businesses and communities.

2.5 Summary and Theoretical Project Description

The market and literature survey above highlights that autonomous delivery robots are an emerging technology with significant momentum. Commercially, numerous examples (Starship, Amazon Scout, Nuro, Alibaba’s Xiaomanlv, Yandex/Noon, etc.) have demonstrated the feasibility of using robots for last-mile deliveries in various contexts. These robots can reduce delivery costs and times, alleviate workforce shortages, and contribute to lower traffic congestion and emissions. Technologically, they rely on a combination of sensors and intelligent algorithms to navigate complex outdoor environments. Vision-based navigation, often complemented by other sensors, allows ADRs to operate with a high degree of autonomy on sidewalks and pedestrian areas. Academic research corroborates and expands upon industry practices by exploring optimal routing, advanced SLAM and planning techniques, and human factors to improve robot deployment. The use of powerful embedded systems (like the NVIDIA Jetson platform coupled with microcontrollers) is enabling real-time processing on the robot, which is essential for independence from constant remote control. We also observed that while ADR technology is advancing rapidly in the US, Europe, and China, regions like the Middle East are beginning to adopt it through pilot projects, indicating a global interest in this innovation.

Theoretical Project Description: Based on these insights, our project will focus on the conceptual design (and eventual development) of an autonomous delivery robot tailored for **outdoor last-mile delivery** in an environment such as a university campus or a residential neighborhood. The proposed robot will be a small, wheeled unit capable of carrying a modest payload (on the order of 10–15 kg, e.g., a couple of grocery bags or a food order) in a secure compartment. It will utilize a **vision-centric navigation system**, leveraging cameras as the primary sensors for environmental perception. In practice, this means implementing a visual SLAM algorithm to build a map of the area and locate the robot within that map, as well as computer vision techniques to detect obstacles (pedestrians, bicycles, pets, vehicles) and interpret traffic signs or signals in its path. To enhance reliability, the vision system will be supported by additional sensors such as ultrasonic rangefinders (for near obstacle detection) and possibly a lightweight 2D LIDAR or RADAR for depth perception in low-visibility conditions. The navigation software will incorporate path planning and obstacle avoidance strategies identified from the literature, enabling the robot to find optimal routes and safely handle dynamic situations (like moving people or sudden obstacles).

For the control and computation, the robot will be built around a robust **embedded system architecture**. We plan to use a high-performance embedded computing board (for example, an NVIDIA Jetson Nano or Xavier) as the main controller to run the AI algorithms, perception processing, and high-level decision making. This will run a Linux-based OS with ROS middleware, which will manage sensor data flows and planning tasks. Complementing the main board, a microcontroller (or a set of microcontrollers) will handle low-level tasks in real time – such as reading wheel encoder ticks, maintaining motor speed via a feedback control loop, and monitoring critical safety sensors (like a bump sensor or emergency stop). This hierarchy follows best practices observed in existing solutions: the Jetson (brain) plans the robot's motion and analyzes the environment, while the microcontroller (body reflex) directly drives the motors and can promptly react to immediate hazards. Communication between these components will be designed to be reliable and low-latency, ensuring the robot's motions are smooth and responsive.

We will also account for practical considerations in our theoretical design. For operation in Egypt and similar regions, factors like high temperatures, uneven sidewalks, and dust must be considered. Therefore, the robot's chassis will be rugged and weather-proof, with an appropriate suspension or wheel design to handle bumps and curbs. The battery system will be sized to allow several hours of operation (e.g., an 18-hour battery life as seen in Starship robots would be a target, though our prototype may have a smaller scale). We will include a charging solution (whether manual swap or an automated docking system) to recharge the robot. Additionally, the design will include a user interface for customers – likely a smartphone app to track deliveries and unlock the robot's cargo compartment upon arrival, similar to the mechanisms used by existing delivery services.

In terms of navigation scope, our robot will be intended for *pedestrian environments* (sidewalks, campus pathways) rather than main roads. This choice aligns with current regulatory ease (as sidewalk robots are often classified separately from road vehicles) and our focus on a controlled deployment area (like a campus or a tech park). We will assume a detailed map of the area can be pre-loaded or learned, and the robot will obey pedestrian traffic rules. For example, it will be programmed to cross streets at designated crosswalks when possible and to yield to humans and pets to ensure safety and friendliness.

Overall, the theoretical project will result in a blueprint and possibly a prototype for an autonomous delivery robot that encapsulates the state-of-the-art features identified in this chapter. By synthesizing the market trends and literature findings, our design aims to be **feasible** with current technology and **adapted to local context**. In the following chapters, we will delve into the specific system design, component selection, and implementation plan, detailing how we will bring this autonomous delivery robot from concept to reality. The expectation is that this project will contribute a stepping stone toward localized ADR solutions, demonstrating the potential for such robots to operate in Egyptian campuses or neighborhoods in the near future, in harmony with global developments in autonomous last-mile delivery.

3 Methodology and System Design

3.1 Overall System Architecture

The AMR's system architecture is organized as a hierarchical, multi-layer design that integrates mechanical, electrical, embedded, software, and cloud components into a coherent whole as shown in Figure. 3.1 . At the lowest layer is the **mechanical chassis and drive system**, which provides mobility and payload support. On top of that sits the **electrical/power subsystem**, which supplies and manages energy to all components. The **embedded control layer** (microcontrollers and motor drivers) interfaces directly with sensors and actuators, executing time-critical tasks (e.g. motor PWM and encoder counting). Above this is the **onboard compute layer**, running ROS 2-based software for perception and navigation. Finally, a **cloud/IoT layer** provides remote monitoring and higher-level coordination (e.g. a Firebase dashboard or web interface). These layers communicate in a 3-tier fashion: tier-1 (embedded/actuators), tier-2 (onboard compute), and tier-3 (cloud services) in which sensor data flows upward and commands flow downward.

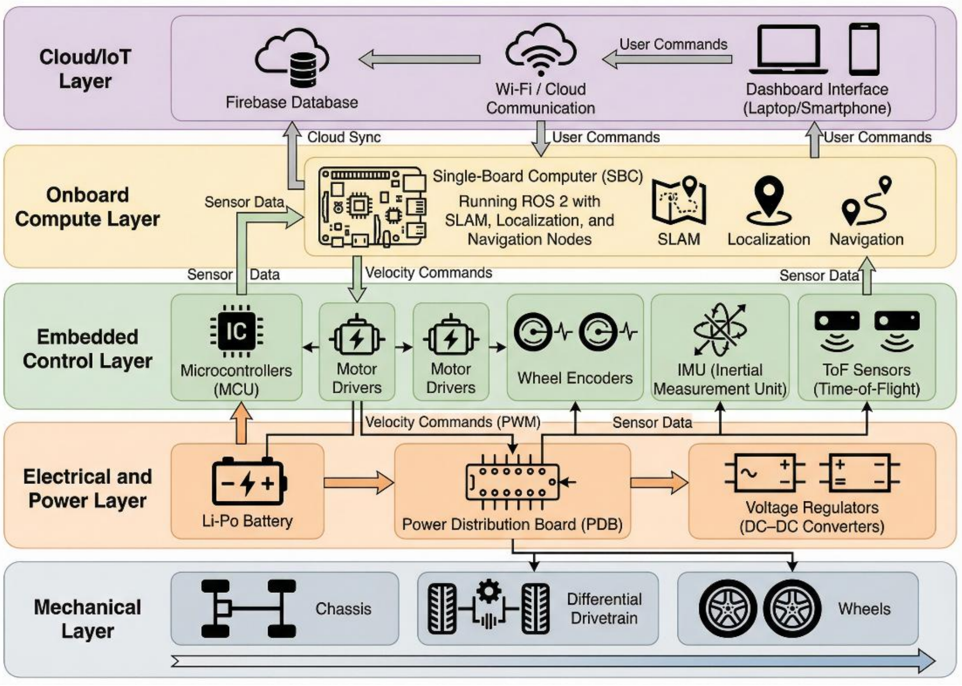


Figure. 3.1 The system block diagram

depicting the major subsystems and data flows. At start-up, sensor inputs (LIDAR, IMU, encoders, etc.) are read by the embedded controllers, converted into digital data, and passed to the onboard computer via a serial or USB interface. The onboard computer (e.g. an Intel NUC or SBC) runs ROS 2 nodes that fuse sensor data (for localization and mapping) and compute motion commands. These velocity commands are sent back down to the motor controllers, closing the control loop. In parallel, the onboard computer exchanges data with the cloud: it publishes telemetry (battery level, pose, status) to a cloud database and pulls high-level commands (e.g. new waypoints) from a dashboard. This layered architecture ensures modularity (each layer has clear responsibilities) and scalability.

3.2 Mechanical Design

3.2.1 Mechanical Design Overview

This chapter describes the mechanical design of the Autonomous Mobile Robot (AMR) delivery platform. The mechanical system was developed to provide a rigid and lightweight structure capable of carrying the target payload, protecting internal electronics, and supporting reliable mounting for sensors and actuators. The design also considers outdoor navigation requirements such as stability during turning, suitable ground clearance, and easy access for maintenance.

The mechanical assembly is organized into three main layers: (1) a chassis/base structure that carries the drivetrain and primary loads, (2) an internal packaging layout for power and control electronics, and (3) an external enclosure that provides protection and integration points for sensors. The complete design was modeled and assembled using CAD (SolidWorks) to verify clearances, mounting interfaces, and serviceability.

3.2.2 Design Requirements and Constraints

The mechanical design was guided by the following requirements and constraints:

Functional requirements

- Payload capacity: **[20] kg** (including safety margin).
- Platform dimensions: **[550L] × [540W] × [320H] mm** to suit corridor/indoor operation.
- Ground clearance: **[GC] mm** to tolerate minor floor irregularities.
- Stable motion during turning and acceleration without tipping.
- Modular assembly enabling fast access to battery, drivers, and wiring.

Integration requirements

- Mounting interfaces for: motors, wheels, battery, motor drivers, controller/compute unit, and sensors (LiDAR / TOF / IMU).
- Clear sensor field-of-view and protected placement against impacts.
- Cable routing paths to reduce snagging and improve reliability.

Manufacturing constraints

- Available manufacturing processes: **[CNC / laser cutting / manual cutting / 3D printing]**.
- Material availability and cost limitations.
- Assembly constraints based on fasteners, tools, and workshop capabilities.

3.2.3 Mechanical Architecture and Configuration

The robot adopts a **differential-drive configuration**, where the left and right sides are driven independently to produce linear and angular motion. This architecture is widely used for indoor mobile robots due to its simplicity, compact turning capability, and ease of control.

The platform utilizes **[4] wheels** arranged symmetrically to improve load distribution and stability. Depending on the final build, the design supports either:

- **4-wheel drive differential** (left pair driven together, right pair driven together), or
- **2-wheel drive + caster support** (if a caster is used for balance).

This configuration was selected to achieve a stable base, predictable kinematics, and a robust mechanical integration for sensors and enclosure.

3.2.4 Chassis and Structural Design

The chassis forms the primary load-bearing structure of the robot. It was designed to achieve sufficient stiffness while minimizing weight and ensuring manufacturability. The chassis layout includes dedicated mounting points for the drivetrain, battery, electronics, and enclosure supports.

Key design features

- Chassis type: **[single plate / stacked plates / frame].**
- Material**: [Aluminum / steel / acrylic], thickness [t] 0.5mm.**
- Reinforcement elements: **[brackets / ribs / corner supports]** to reduce bending and vibration.
- Standardized hole patterns for modular mounting and future expansions.
- Serviceability
- Clearance and access to critical components (battery replacement, wiring inspection).
- Cable routing holes/slots and tie points to support safe wire management.
- Mechanical protection zones for electronics and sensor wiring.

3.2.5 Drivetrain Integration (Motors, Wheels, Mounts)

The drivetrain integration focuses on secure motor mounting, alignment of wheel hubs, and reliable torque transmission. Motor mounts were designed to maintain alignment under load and to withstand vibration during operation.

Motor mounting

- Motor type: **[DC geared motor]**.
- Motor mounts: **[custom bracket / plate mount]** with bolt pattern matched to motor housing.
- Vibration considerations: use of **[washers / locknuts / rubber spacers]** if required.

Wheel coupling and alignment

- Wheel attachment: **[hub/coupler]** designed to prevent slip under torque.
- Alignment strategy: motor shaft to wheel hub coaxial alignment to reduce wobble and bearing stress.

Protection and safety

- Mechanical shielding for rotating elements where applicable.
- Strain relief for motor wires and routing away from moving parts.

3.2.6 Wheel Selection and Ground Interaction

Wheel selection was carried out based on traction requirements, load-carrying capability, and compatibility with the drivetrain mounting interface. As shown in Figure. 3.2, the selected wheel features a rugged, high-friction tread pattern and a relatively wide contact surface. This geometry improves grip on indoor surfaces (e.g., tiles and smooth floors) and reduces wheel slip during acceleration and differential turning.

In addition to traction, the wheel choice was validated against drivetrain integration constraints. Figure. 3.2 shows the wheel set mounted on the drive axle module, confirming mechanical compatibility with the axle/hub interface and ensuring stable torque transmission without wobble or misalignment. This integration check is critical for maintaining consistent odometry performance, minimizing vibration, and preventing mechanical loosening during prolonged operation.

Wheel selection criteria (applied in this project):

- Diameter **[18cm]**: balance between obstacle tolerance and required motor torque.
- Width **[80cm]**: increased ground contact for better grip and stability.
- Tread pattern: improved traction and reduced slipping during turning.
- Hub/axle interface: secure mounting and reliable torque transfer.
- Material: durability and wear resistance under expected operating conditions.

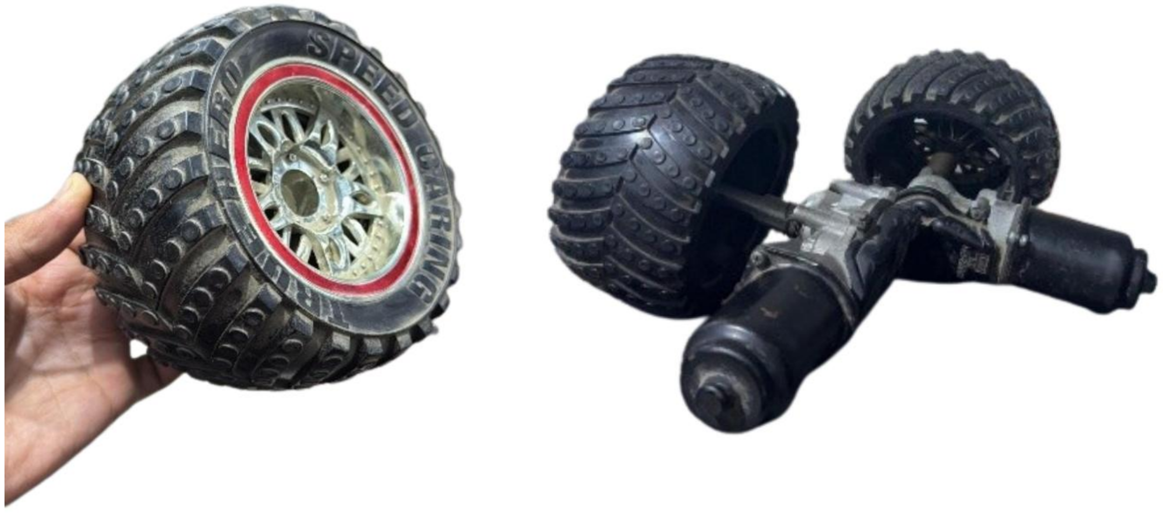


Figure. 3.2 Selected wheel used in the AMR platform showing the tread pattern and hub/rim geometry.

3.2.7 Enclosure and Packaging

The robot enclosure was designed to provide mechanical protection for the internal electronics while maintaining easy access for maintenance and component replacement. The enclosure follows a box-type structure with curved/rounded edges and pre-defined mounting holes to support modular assembly and future expansions.

As shown in Figure. 3.3, the enclosure panels were fabricated and assembled as a rigid shell with multiple fastener points along the edges, enabling secure attachment and improved structural stiffness. The front side includes a slotted opening that can be used as a carrying/handling feature and also supports practical access during assembly. Edge reinforcement elements (brackets/rails) are used to increase rigidity and maintain alignment between panels.

In addition to the physical build, the full enclosure integration was validated in CAD. Figure. 3.4 shows the SolidWorks assembly model of the robot with the enclosure installed on the 4-wheel platform, including the top cover and the sensor mount on the upper surface. This CAD verification ensured proper clearances with the wheels, confirmed the overall packaging volume for internal components (battery, drivers, controller), and supported serviceability through a top-access cover design.

Packaging considerations applied:

- Low placement of heavy components (battery/power stage) to improve stability.
- Defined internal mounting zones for electronics and cable routing.
- Top-access maintenance concept to reduce disassembly time.
- Protective body geometry to reduce exposure to impacts and dust.

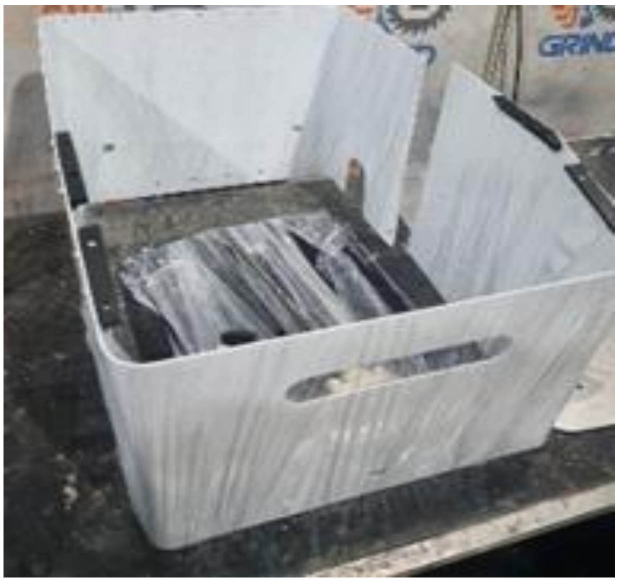


Figure. 3.3 Fabricated enclosure panels during assembly, showing mounting holes, edge reinforcements, and the front handling slot

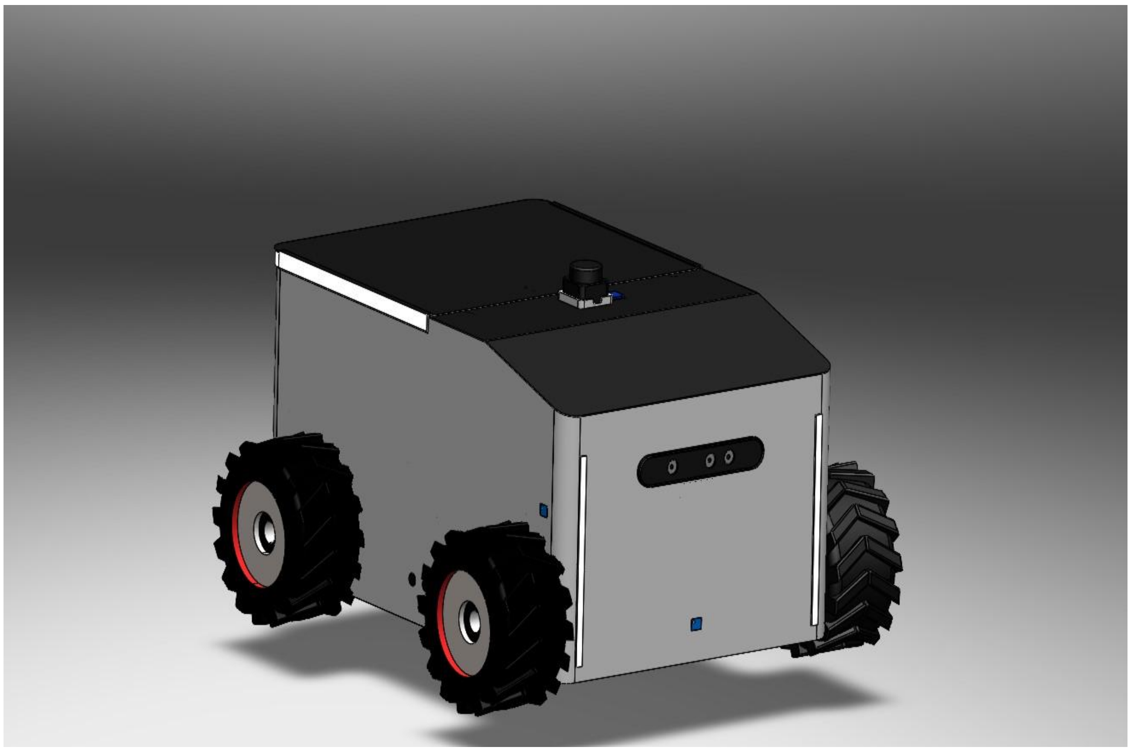


Figure. 3.4 SolidWorks assembly of the AMR with the enclosure installed

3.2.8 Sensor and Electronics Mounting

The sensor and electronics mounting strategy was developed to guarantee accurate measurements, stable alignment, and reliable operation during motion. The mounting design focused on: (1) maintaining clear sensor fields of view, (2) reducing vibration effects, (3) ensuring repeatable frame alignment relative to the robot base, and (4) providing safe cable routing and easy maintenance.

LiDAR Mounting

The LiDAR sensor was positioned on the top surface of the enclosure to maximize the scanning coverage and minimize occlusion from the robot body. The bracket and mounting interface were designed based on the LiDAR mechanical envelope and mounting footprint. As shown in Figure. 3.5, the sensor dimensions and hole spacing were used to ensure correct fit, sufficient clearance, and rigid attachment.

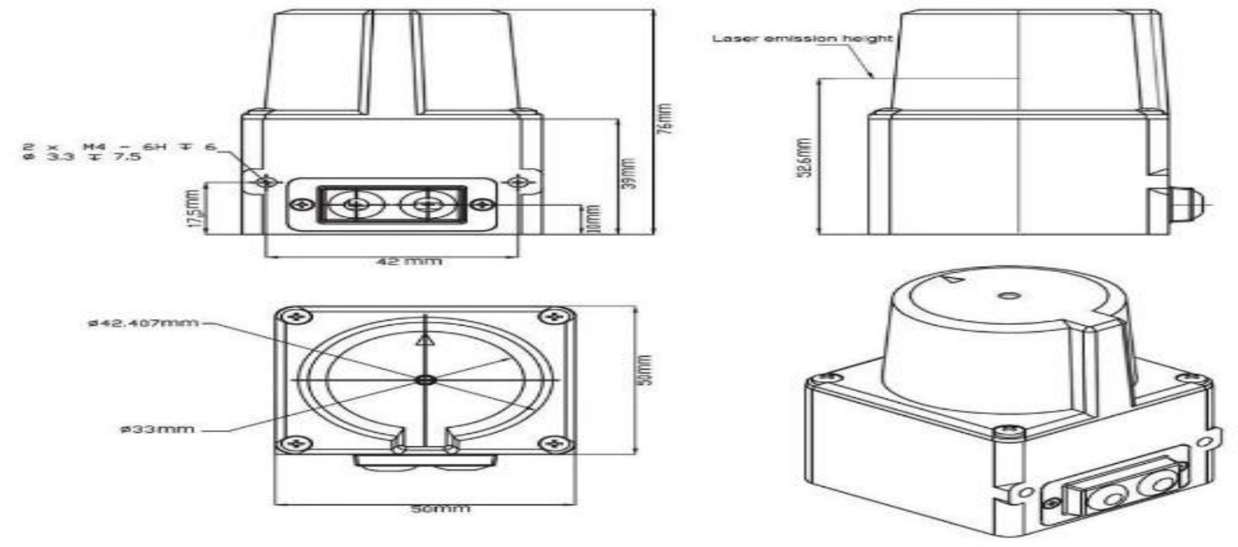


Figure. 3.5 LiDAR mechanical 2D sketches

Placing the LiDAR near the robot's geometric center improves scan consistency during rotations and helps reduce mapping/localization errors caused by asymmetric placement. A rigid mounting plate/bracket was used to maintain the sensor orientation, while the sensor cable was routed through a protected path to avoid strain on the connector.

TOF Sensor Mounting

TOF sensors were distributed around the robot perimeter to provide near-field obstacle detection. The sensor height and tilt were selected to detect typical indoor obstacles while minimizing interference from the floor surface. The mounts were designed to protect the sensors from impact and maintain fixed orientation during operation.

IMU Mounting

The IMU was installed close to the robot's center of mass on a rigid internal plate to reduce vibration-induced noise. This location improves the stability of inertial readings and supports more accurate attitude estimation.

Electronics Mounting and Cable Management

Electronic modules were mounted using standoffs to ensure electrical insulation and provide clearance for wiring. Cable routing was organized using tie points and guided paths to prevent contact with moving parts, reduce connector stress, and improve overall reliability and serviceability.

3.2.9 Stability and Center of Mass Considerations

Stability was a primary objective to ensure safe operation during turns and speed changes. The mechanical design follows a low center-of-mass approach by positioning heavier components near the base plate and close to the center.

Stability measures

- Battery installed in the lowest feasible location.
- Symmetrical left/right distribution of heavy modules to reduce lateral imbalance.
- Track width and wheelbase selected to resist tipping during turning.

Engineering rationale
A lower center of mass reduces the overturning moment during acceleration and turning, improving safety and navigation performance, especially in indoor environments where frequent turns occur.

3.2.10 Manufacturing

The mechanical structure was manufactured using available workshop processes, focusing on dimensional accuracy, structural rigidity, and ease of assembly. The fabrication stage included preparing the enclosure panels, drilling the mounting patterns, and producing the required brackets/plates to interface the chassis with the drivetrain and internal components.

As shown in Figure. 3.6, the main enclosure body and the drivetrain components were prepared before final integration. This step was used to verify physical fit, confirm wheel clearance relative to the body, and ensure that mounting points align with the intended assembly layout. The enclosure panels include pre-drilled holes distributed along the edges to support secure fastening and improve stiffness after assembly.

In parallel, the supporting mechanical parts (base plates, brackets, spacers/rods, and small mounts) were prepared and organized for assembly, as illustrated in Figure. 3.7. This preparation phase reduces installation errors and speeds up assembly, while ensuring consistent alignment between the chassis, drivetrain modules, and enclosure. Standard fasteners and brackets were used to achieve a modular structure that can be disassembled for maintenance.

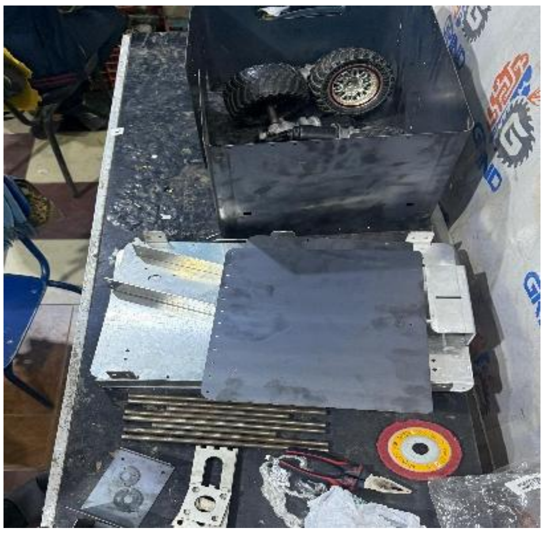


Figure. 3.6 mechanical parts before final integration

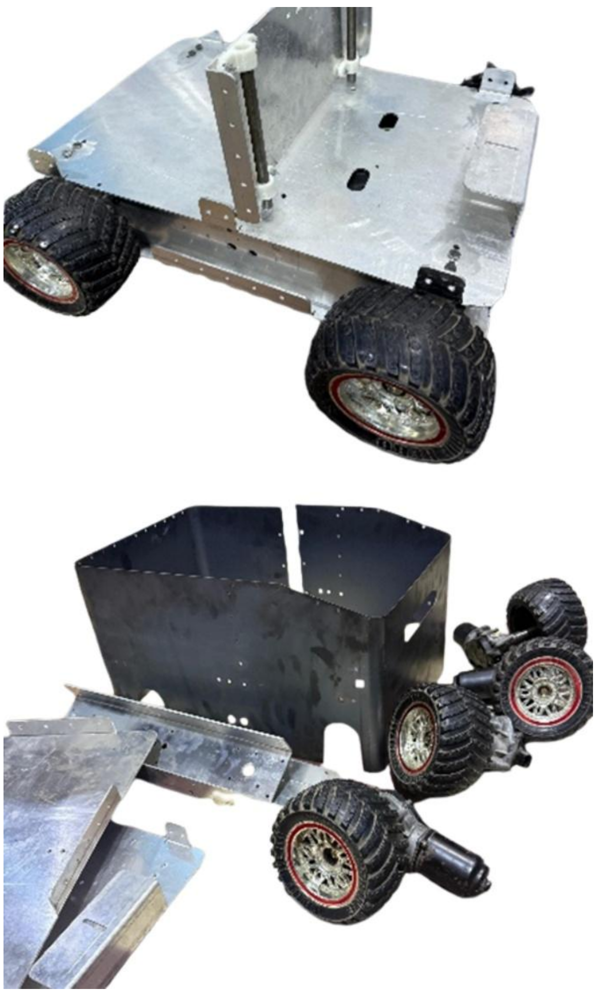


Figure. 3.7 base plates, brackets, spacers/rods, and small mounts Prepared for assembly

After completing the sub-assembly preparation, the chassis base plate, wheel modules, and supporting brackets were integrated to form the complete mechanical base. the assembled platform confirms correct wheel clearance, structural alignment, and rigidity of the mounting interfaces. The final assembly includes the main base plate with standardized slots/holes for internal mounting, and reinforced supports to maintain stiffness and minimize vibration during operation. This stage serves as a mechanical verification step before installing the enclosure panels and internal electronics.

3.2.11 Final Mechanical Specifications

Mechanical specifications (example template)

- Overall dimensions (L×W×H): **[95040] cm2**
- Total mass (no payload): **[20] kg**
- Payload capacity: **[15] kg**
- Wheel diameter / width: **[18] cm / [80] cm**
- Drive configuration: **Differential drive**
- Material: **[Aluminum & steel]**
- Manufacturing methods: **[laser cutting machine – 3d printing]**

Bill of Materials (BOM)

- Chassis plate/frame: **[aluminum, thickness]**
- Motor mounts/brackets: **[Chassis plate/frame: Mild steel sheet, 2 mm thickness (laser-cut and bent]**
- Wheels and hubs: **[4 × rugged rubber wheels on integrated drive-axle hubs; coupled to the motor/axle module via]**
- Fasteners: **[M3/M4/M5/M6]**
- Enclosure panels: **[Mild steel sheet, 0.5mm]**
- Sensor mounts: **[Combination of 3D-printed brackets (for LiDAR/TOF) and metal brackets (for rigid mounting), secured with M3]**

3.3 **Electrical and Power System Design**

The electrical and power system of the Autonomous Delivery Robot (ADR) is a fundamental subsystem that directly influences the platform's reliability, safety, and performance. The design focuses on delivering stable power to all electronic components, ensuring clean signal transmission, and maintaining electrical isolation between high-power and low-power domains. By using a structured multi-voltage architecture and robust grounding practices, the system minimizes electrical noise and interference while supporting modularity, scalability, and ease of maintenance in an outdoor mobile robot environment. The following sections detail the power architecture, sources, distribution, controller integration, communication interfaces, and safety considerations of the ADR's electrical design.

3.3.1 Power Architecture and Design Objectives

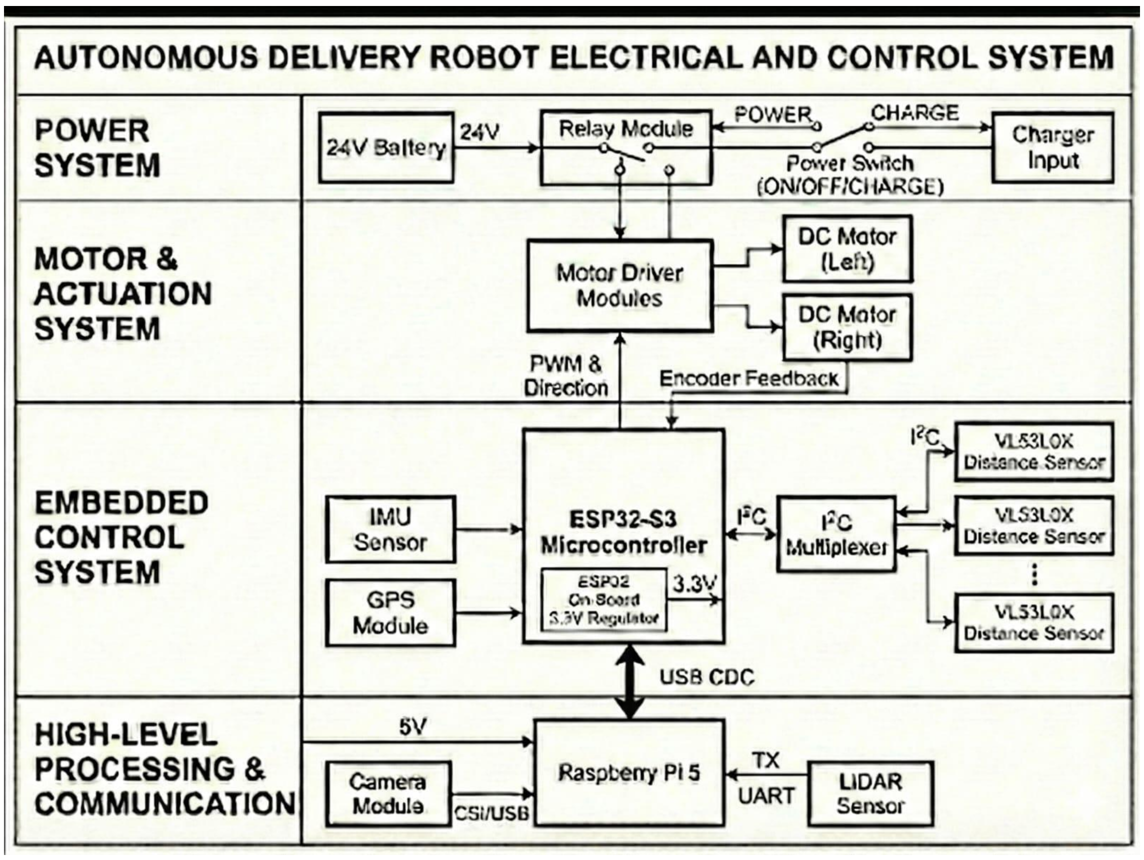


Figure. 3.8 High-level electrical and control system architecture of the Autonomous Delivery Robot

The ADR's electrical system is built on a hierarchical power architecture as shown in Figure. 3.8 that cleanly separates the high-power actuation circuitry from the low-power logic and sensing circuitry. In practice, this means the motors and their drivers operate in an isolated high-voltage domain, whereas processors and sensors reside in a low-voltage domain. By partitioning the power system in this way, the design minimizes coupling of noise from motors into sensitive electronics and improves overall robustness. High-current motor currents are routed on separate paths or planes from the logic ground to avoid injecting interference into the microcontrollers and sensors. All low-power subsystems share a common reference ground that ties to the high-power ground at a single point, preventing ground loops and ensuring a stable reference voltage throughout. This architectural approach enhances reliability since disturbances in one domain (e.g., motor voltage spikes) have limited impact on the other.

Design Objectives: The main objectives guiding the power architecture can be summarized as follows:

- Stable Voltage Levels: Provide regulated and stable voltages for all subsystems (e.g. 24 V for motor actuation, 5 V for single-board computing, and 3.3 V for embedded logic and sensors) under all load conditions. This prevents brown-outs of electronics during peak loads and keeps sensors and controllers operating within safe limits.
- Noise and EMI Reduction: Minimize electromagnetic interference and voltage noise in the system by isolating noisy high-power circuits, using proper grounding, and filtering power lines. This ensures clean signal transmission and reduces the chance of erratic behavior due to electrical noise.
- Support Multiple Devices: Accommodate numerous sensors, actuators, and communication interfaces by providing sufficient power capacity and proper interfacing (voltage-level compatibility, connectors). The architecture must be scalable to add future components without major redesign.
- Safe Power Distribution: Ensure that power is distributed safely with appropriate wire sizing, over-current protection, and thermal considerations. The system should tolerate varying load conditions (motor stall currents, inrush currents) without failure, using fuses or current limiters where necessary.
- Modularity and Simplicity: Use a modular design with well-defined power domains and connector interfaces, making it easier to debug issues and replace or upgrade parts. Simplicity in the power layout (clear separation of domains, straightforward wiring) improves reliability and maintainability.

Overall, the design emphasizes simplicity and reliability while maintaining compatibility between the low-level embedded control and the high-level computing units. By meeting these objectives, the electrical architecture provides a solid foundation that can be expanded or modified as the project evolves.

3.3.2 Power Sources and Voltage Domains

The robot utilizes two primary power source domains serving different functional roles. First, a 2 dedicated 12V batteries (high-voltage domain) as shown in figure is used exclusively for the propulsion system. This battery directly feeds the DC gear motors through high-power motor driver modules. Using a 24 V supply for motors is advantageous because, for a given power, higher voltage results in lower current draw, reducing resistive losses and allowing smaller, more efficient wiring and drivers. In fact, many mobile robots favor 24 V motors since they deliver the same power at half the current of an equivalent 12 V system, which improves efficiency and reduces heating in conductors. The 24 V domain is strictly confined to motor and motor driver circuitry, physically and electrically separated from the logic electronics to contain noise and voltage transients.

Secondly, a low-voltage domain supplies all logic, control, and sensor electronics. This domain operates primarily at regulated 5 V and 3.3 V levels. The Raspberry Pi 5 single-board computer (SBC) serves as the main logic power hub of the system. It is powered directly from a regulated 5 V power bank, which provides a stable and portable energy source suitable for mobile robotic applications. The Raspberry Pi, in turn, supplies a regulated 5 V rail through its GPIO header to power the ESP32-S3 control board and other low-power peripherals. The ESP32-S3 microcontroller board is connected to this 5 V rail and uses an onboard regulator to step it down to 3.3 V for its operation (details in Section 3.3.3). By having the Pi and microcontroller share the same 5 V supply and ground, a common reference is maintained across the computing units, which is important for reliable communication. All sensors and low-power devices are likewise tied into the 5 V/3.3 V domain. The result is a synchronized logic power system that ensures the Raspberry Pi, microcontroller, and sensors all see consistent voltage levels and ground references.

Crucially, the high-power and low-power electrical domains are interfaced only through logic-level control signals and a single-point ground reference. There is no direct power crossover between the two domains; for example, motor drivers receive control inputs from the ESP32-S3 microcontroller, while the motor supply voltage is confined exclusively to the propulsion subsystem and does not feed into the logic or sensing circuits. This functional isolation prevents large motor current transients and switching noise from propagating into the sensitive 5 V and 3.3 V logic supplies. In the proposed ADR design, the low-power electronics are powered from a regulated 5 V power bank, while the motors are supplied from a dedicated high-voltage battery, providing sufficient electrical separation without the need for additional isolation stages. A common ground reference is maintained to ensure reliable signal integrity; however, it is carefully structured to avoid ground loops and minimize potential ground voltage differences. Overall, this dual-domain power strategy achieves both high efficiency for motor actuation and stable operation for logic and sensing subsystems, satisfying the distinct electrical requirements of high-power actuators and low-power embedded electronics.

3.3.3 Voltage Regulation and Power Distribution

Most digital components and sensors in the Autonomous Delivery Robot operate at a 3.3 V logic level, which necessitates reliable voltage regulation from the available 5 V supply down to 3.3 V. The ESP32-S3 control board relies on the onboard low-dropout (LDO) voltage regulator integrated within the ESP32-S3 module to convert the 5 V input supply into a stable 3.3 V rail. This regulated 3.3 V output is used to power the microcontroller core as well as all connected low-power sensors. Local decoupling capacitors are placed close to the ESP32-S3 power pins and sensor connectors to ensure voltage stability, suppress noise, and maintain reliable operation during high processing load and wireless communication activity. By performing this regulation locally on the control PCB as shown in Figure. 3.9 and Figure. 3.10 , voltage drops and noise present on the 5 V supply—caused by cable resistance, load variations, or dynamic current consumption—are effectively filtered out. This ensures a clean and stable 3.3 V supply even during periods of high computational load or wireless communication activity. The Raspberry Pi 5 similarly integrates onboard power management circuitry that generates its required internal voltage rails, including 3.3 V, from the regulated 5 V input provided by the power bank. Overall, the staged regulation approach, in which a regulated 5 V source feeds localized 3.3 V regulation, improves voltage stability, enhances noise immunity, and ensures reliable operation of the logic and sensing subsystems.

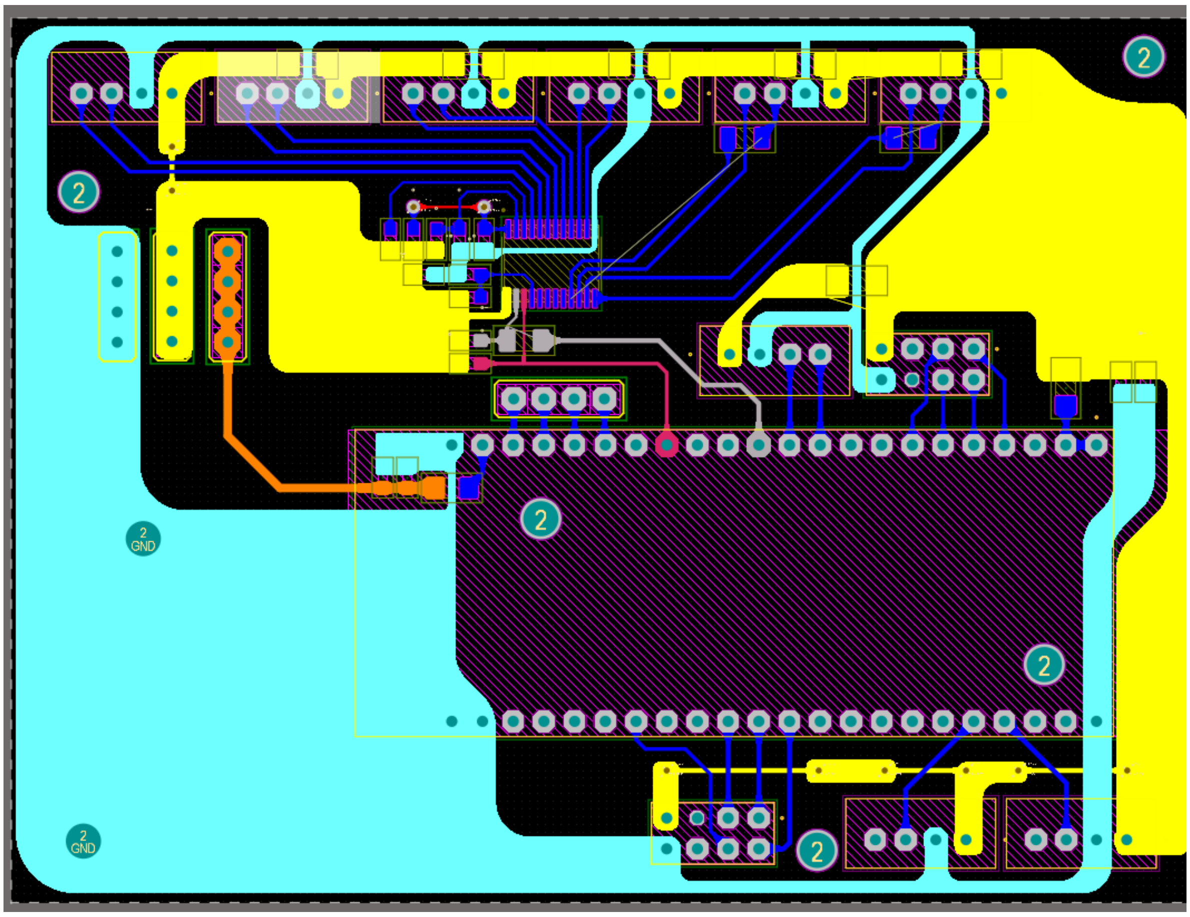


Figure. 3.9 PCB layout of the ADR controller board showing ground planes, power routing, and separation between high-current and low-power signal traces.

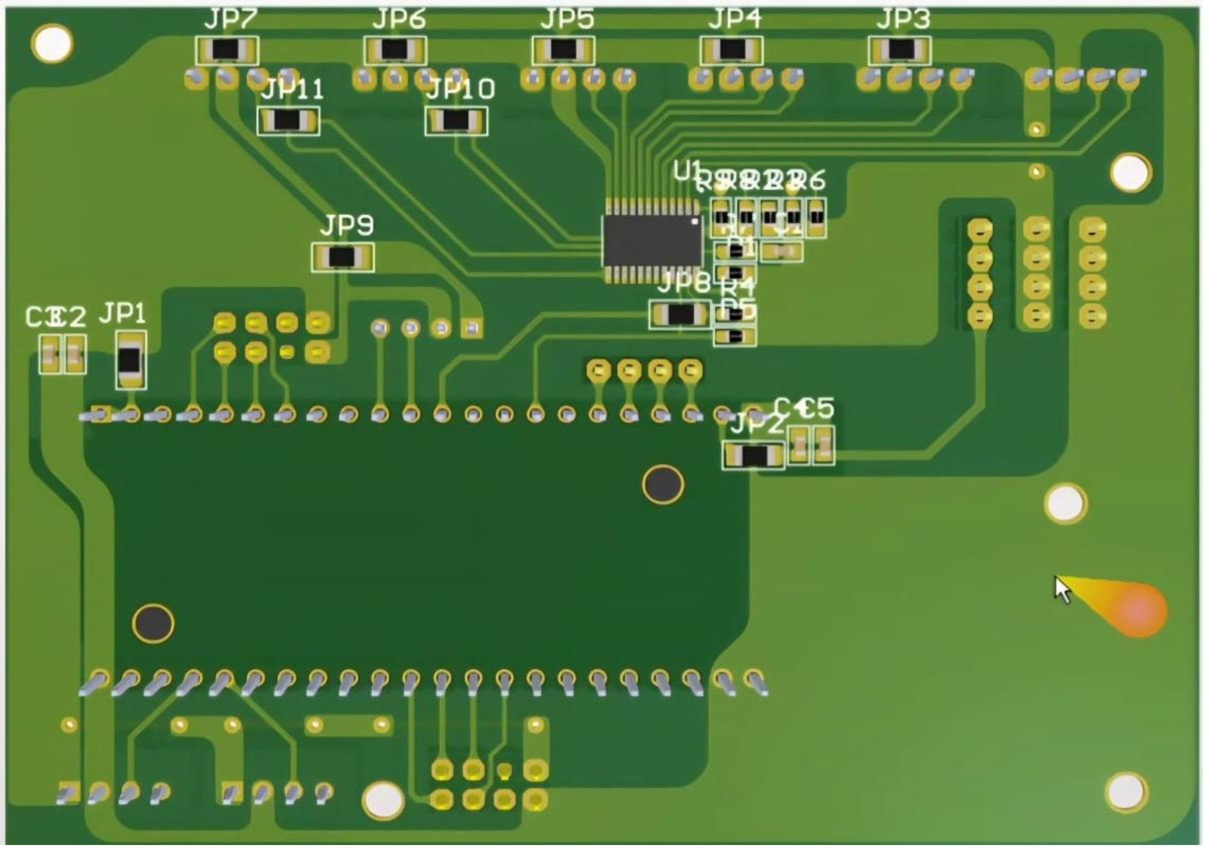


Figure. 3.10 3D view of the controller PCB highlighting component placement and internal routing for stable power distribution.

The power distribution network is carefully designed to safely route power from the sources to all subsystems with minimal loss and interference. PCB trace widths and wire gauges are chosen based on expected current levels in each path – for example, motor driver supply traces and battery leads are much thicker than sensor supply traces, to handle several amperes without overheating or significant voltage drop. Critical power nodes employ bulk capacitors (electrolytic or tantalum capacitors in the tens to hundreds of microfarads) placed at strategic locations such as the input of motor drivers and the 5 V regulator output. These bulk capacitors act as local energy reservoirs that smooth out transient dips or spikes in voltage when motors rapidly change speed or when digital circuits draw burst current. In addition, high-frequency decoupling capacitors (typically 0.1 µF ceramic capacitors) are placed as close as possible to the power pins of every integrated circuit (microcontroller, sensor modules, etc.). The decoupling capacitors shunt away high-frequency noise and stabilize the supply voltage at the IC by providing instantaneous current for fast switching events. This combination of bulk and decoupling capacitors across the power network maintains a low-impedance supply rail over a wide frequency range, preventing voltage ripple from affecting circuit performance.

From a grounding and layout perspective, the PCB uses a common ground plane that serves as the return path for all currents. A low-impedance ground plane is crucial for stable operation, as it ensures that ground potentials remain uniform across the board and that return currents can flow with minimal loop area. Special care is taken to route high-current returns (from motors and drivers) in such a way that they do not flow under or near sensitive analog/digital sections of the board. In practice, this might involve partitioning the ground plane or using star topology grounding for the motor controllers – bringing their ground returns to a single point (e.g. a ground lug or star point on the chassis) that then connects to the rest of the ground. By doing so, the design avoids creating overlapping ground loops between the high-power and low-power sections. All ground connections eventually tie together (since the system is battery-operated and shares a floating ground), but they meet at a common node to prevent circulating currents. This approach follows best practices in robotics, where typically a single ground point on the chassis is used for all returns to eliminate multiple ground path problems.

To further protect the system, the main battery positive line is typically fused near the source to prevent catastrophic currents in the event of a short-circuit or component failure. For example, a fast-blow fuse or a circuit breaker is placed immediately after the 24 V battery terminal. The fuse is rated just above the normal maximum current draw of the robot so that any unexpected surge (e.g. a stalled motor or wiring fault) will blow the fuse and isolate the battery. This prevents wiring damage or fire and protects the motor drivers and battery from excessive stress. Likewise, the 5 V regulator may have current limiting or a separate fuse on its input to protect the logic supply. These safety elements ensure that power distribution remains controlled and that failures in one branch do not cascade through the system.

In summary, through proper voltage regulation and power distribution design, the ADR's electronics receive stable power with minimal noise. Adequate trace sizing, localized decoupling, and solid grounding together create a robust electrical backbone. Even under dynamic loads like motors starting or stopping, the logic voltage stays within tolerance, and sensitive components are shielded from interference. This power integrity is essential for the robot's reliable operation, preventing issues like sensor misreads or processor resets that could be caused by power fluctuations.

3.3.4 Embedded Controller and Sensor Integration

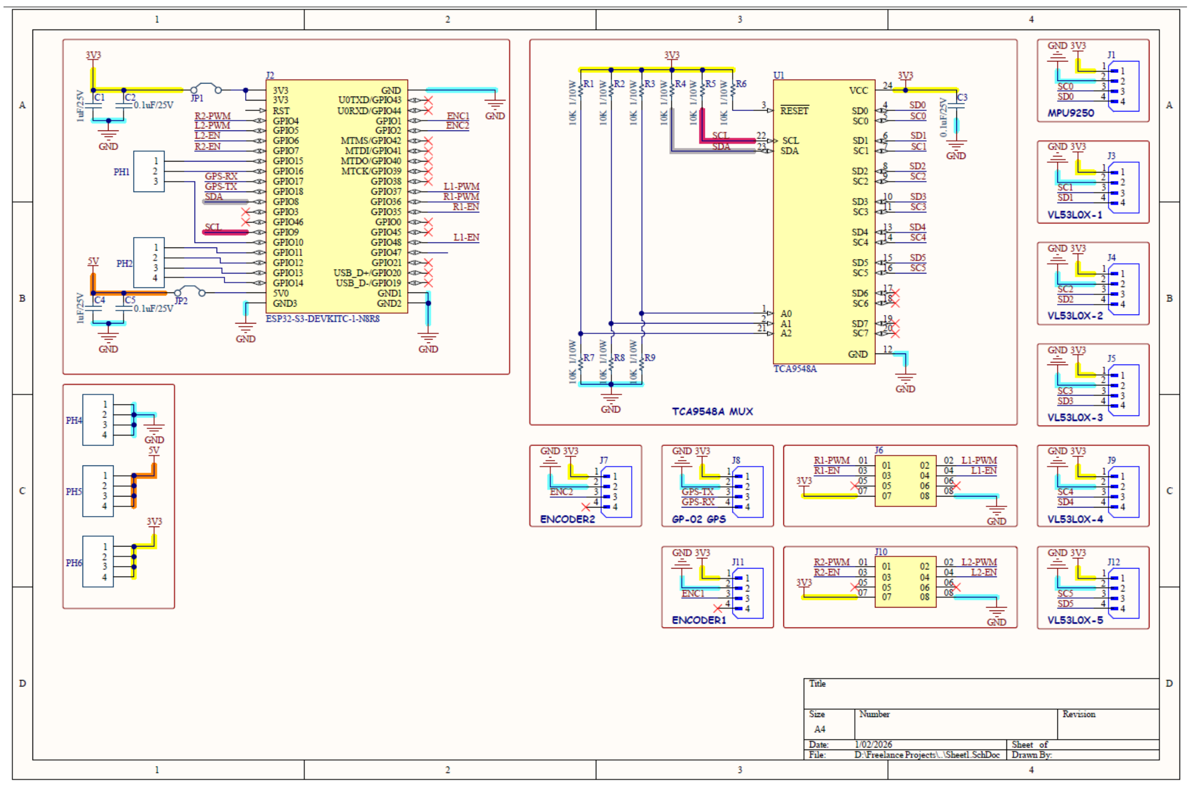


Figure. 3.11 The complete electrical schematic of the embedded control system

A custom controller board consolidates the low-power electronics. The ESP32-S3 microcontroller module (black board with silver antenna on the right) is mounted at the center, and numerous standardized connectors (white JST sockets along the edges, labeled J6, J7, J8, J10, J11, etc.) provide plug-and-play interfaces for sensors, motor drivers, and other peripherals. This modular layout simplifies integration and maintenance of the robot's electronics.

The ADR's control architecture shown in Figure. 3.11 employs an ESP32-S3 microcontroller as the low-level embedded controller responsible for real-time operations. This microcontroller is powered from the regulated 3.3 V rail discussed above, ensuring it has a clean and stable supply for its high-speed processor, wireless radio, and I/O lines. On the PCB, careful attention is given to the ESP32's power pins: multiple decoupling capacitors are placed adjacent to the module to prevent any supply dip during transmit bursts (Wi-Fi/Bluetooth) or when multiple GPIOs switch simultaneously. The ESP32-S3 handles tasks such as motor control (generating PWM signals and reading encoder inputs), sensor data acquisition, and communication with the higher-level computer (Raspberry Pi). It is essentially the real-time brain of the robot, offloading time-critical control loops from the Raspberry Pi. The reset circuitry and boot mode pins of the ESP32 are also tied into the design with proper pull-up/down resistors to ensure the microcontroller reliably starts up in the correct mode each time power is applied.

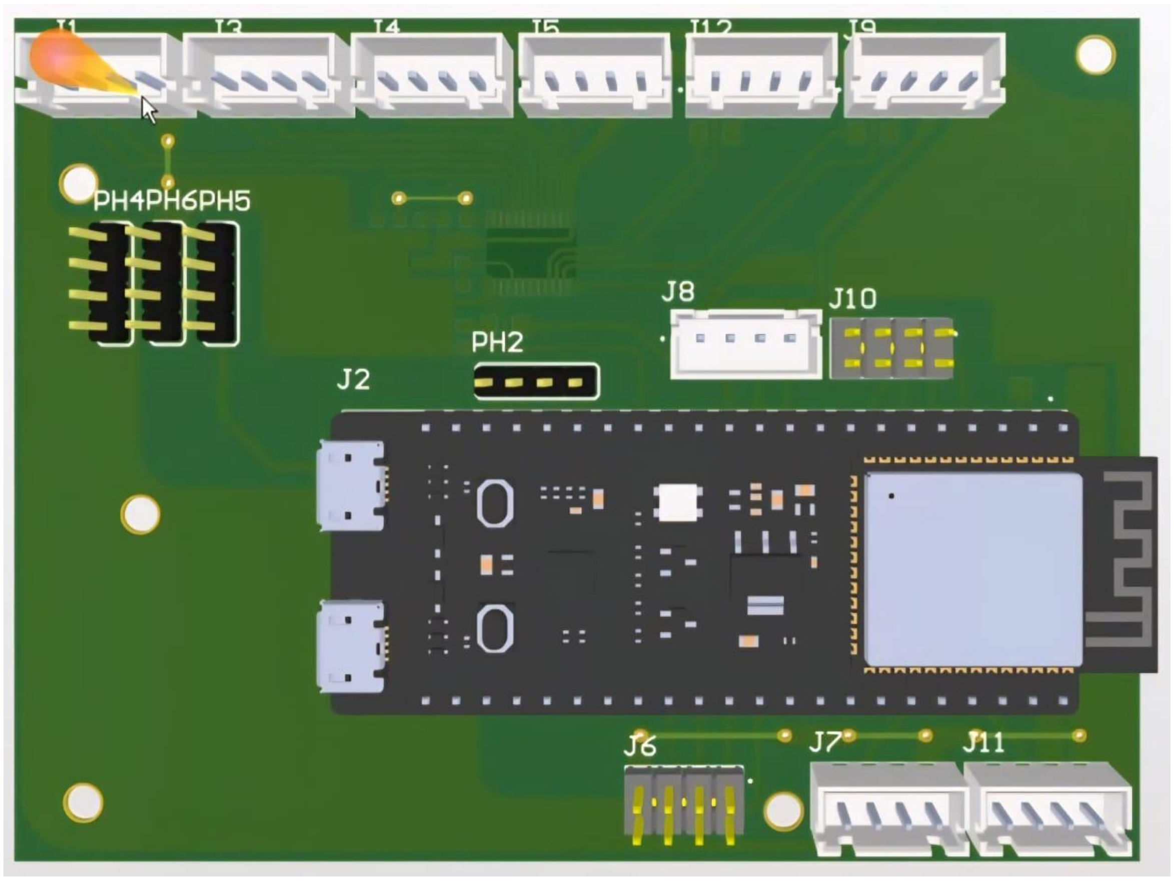


Figure. 3.12 A 3D Look on the assembled embedded controller board showing the

All sensing and auxiliary devices in the Autonomous Delivery Robot interface electrically with the ESP32-S3 control board and are powered from the same low-power logic domain. In the design shown in Figure. 3.12 , none of the sensors are mounted directly on the controller PCB; instead, all sensing modules are externally mounted on the robot body and connected to the control board through dedicated interfaces. The sensor suite includes an MPU9250 inertial measurement unit (IMU) for orientation and acceleration estimation, a GP-02 GPS module for global positioning, multiple Time-of-Flight (ToF) distance sensors (VL53L0X) distributed around the chassis for obstacle detection, and motor encoder sensors for wheel feedback. Each sensor module is electrically connected to the ESP32-S3 using standardized modular connectors. These connectors, such as 4-pin JST-PH connectors, provide regulated 3.3 V power, ground, and the required signal lines in a compact and keyed form factor. The use of uniform connectors and consistent pin assignments enables a plug-and-play integration approach, allowing sensors to be connected, replaced, or repositioned without custom wiring. For example, each VL53L0X distance sensor connects to a designated port on the controller board, receiving 3.3 V power and ground while exposing its I²C communication lines to the ESP32-S3. Similarly, the IMU and GPS modules interface through their respective I²C or UART connections. This modular wiring strategy simplifies system assembly, enhances resistance to vibration in outdoor operation, and significantly reduces the likelihood of wiring errors.

In laying out the PCB and cable harnesses, signal integrity for the sensor connections is given high priority. Sensitive signal lines (such as the I²C bus lines, UART RX/TX, and encoder outputs) are routed away from high-current traces and the motor driver areas to avoid picking up switching noise. Whenever a sensor cable runs in proximity to power wires, it is either kept short or shielded/twisted to reduce electromagnetic coupling. Additionally, all sensors share the common ground of the logic domain, so there are no ground potential differences between the ESP32 and any sensor. This prevents issues like ground offsets that could corrupt analog readings or communication signals. The uniform 3.3 V supply for all sensors means each sensor experiences the same reference voltage, again contributing to consistency in sensor readings and behavior.

By integrating the microcontroller and sensors in this cohesive manner, the ADR ensures that the low-level control loop has direct, efficient access to sensor data and can directly actuate motors. The ESP32's I/O capabilities (multiple ADCs, interrupts for encoders, I²C, SPI, UART, etc.) are fully leveraged to interface with the variety of peripherals on the robot. This tight integration is what enables real-time monitoring of the robot's state; for example, wheel encoders provide instantaneous speed feedback to the microcontroller, which can adjust motor PWM outputs on the fly to implement closed-loop speed control. Meanwhile, the IMU and ToF sensors feed into the microcontroller for tasks like balancing, obstacle detection or odometry, with the ESP32 doing pre-processing before relaying important information to the Raspberry Pi. Overall, the embedded controller and sensor integration are designed to be robust, modular, and responsive, forming a nerve center that reliably links the robot's hardware to its higher-level decision-making algorithms.

3.3.5 Communication Interfaces and Signal Integrity

To support autonomous operation, the electrical system incorporates multiple communication interfaces for data exchange between components. The two primary communication links in the ADR are an I²C bus (Inter-Integrated Circuit) for local sensor communications and a **USB CDC (USB-to-Serial, COM port)** link between the ESP32 microcontroller and the Raspberry Pi 5.

Sensor Communication (I²C): The I²C bus is used extensively to communicate with sensors such as the IMU (MPU9250) and the array of VL53L0X distance sensors. I²C is chosen for its simplicity and ability to support multiple devices on the same two-wire bus (SDA for data, SCL for clock). However, a known challenge arises when using multiple identical I²C devices like the VL53L0X rangefinders – by default, they share the same fixed I²C address, which would conflict on a single bus. The ADR design solves this by employing an I²C multiplexer chip (TCA9548A) to create separate channelled buses for each Time-of-Flight sensor. The TCA9548A acts like an electronic switch that can connect the master (ESP32) to one sensor at a time on command. This allows up to 8 devices with identical addresses to coexist, each on its own isolated channel of the multiplexer. The ESP32 selects the desired sensor's channel on the TCA9548A and then communicates normally; the multiplexer routes the data to only that sensor. This approach is far more scalable and less "fiddly" than trying to manually change sensor addresses by toggling pins or using separate I²C buses. Pull-up resistors are included on the I²C lines (either on the PCB or on the breakout boards) to ensure proper logic levels and signal integrity, with typical values in the 2.2–4.7 kΩ range for 3.3 V operation. The I²C wiring on the PCB is kept short and in a clean layout, often with SDA and

SCL routed in parallel and with ground nearby to provide a return path, minimizing interference. The communication speed of I^2C is set to a moderate rate (often 100 kHz or 400 kHz) to maintain reliability over the given trace lengths and connectors. With these practices, the I^2C sensor network operates reliably, allowing the microcontroller to poll multiple sensors in quick succession for environment awareness.

Processor Communication (USB CDC / Virtual COM Port): For data exchange between the ESP32 microcontroller and the Raspberry Pi 5 (which runs high-level autonomy software), the ADR uses the ESP32-S3's native USB interface configured as **USB CDC-ACM** (Communication Device Class – Abstract Control Model). In practice, this presents to the Raspberry Pi as a standard serial device (a "virtual UART" over USB, typically appearing as `/dev/ttyACM*`), while physically the two boards are connected using a USB cable. This link enables commands, status messages, and sensor/telemetry data to be exchanged in a simple, robust manner, while benefiting from USB's differential signaling and built-in error checking. Architecturally, the system still follows the common robotics pattern where a microcontroller handles low-level motor/sensor tasks and communicates with a higher-level computer through a serial-style channel[14][15]—the key difference is that the "serial link" is transported over USB rather than raw TTL UART pins. For example, the Raspberry Pi can send motion commands or high-level directives to the ESP32 (over the USB CDC channel), and the ESP32 responds with telemetry such as wheel speeds, battery voltage, IMU samples, and ToF readings. The effective throughput can be configured well beyond legacy 115200 bps-style limits, while remaining easy to interface from Linux and middleware layers. This is particularly practical when running micro-ROS, where the Pi can host the agent and the ESP32 can act as a USB device endpoint; the physical connection is plug-and-play, mechanically robust, and reduces wiring complexity compared with separate TX/RX/GND runs. Since the communication is via USB, there is no concern about mismatched logic levels between GPIO UART pins, and the short internal cable length inside the robot helps keep the system resilient against external EMI.

It's worth noting that alternative communication protocols were available (SPI, CAN, UART over GPIO, etc.), but **USB CDC** provided a straightforward solution with minimal additional wiring and strong signal robustness. SPI was not chosen for ESP32–Pi communication because it would require more wires plus stricter master/slave timing management, and CAN was unnecessary given the short-range and simplicity of the internal link. A raw UART connection (using TX/RX on the GPIO header) was also possible, but it would require dedicated wiring, careful level compatibility checks, and offers less inherent noise immunity than USB's differential physical layer. USB CDC also benefits from standardized host drivers on Linux and packet-level integrity mechanisms (CRC, retries), which helps reduce the probability of silent data corruption. At the application layer, the communication can still be framed with structured packets and optional checksums/sequence counters for additional robustness, especially when exchanging control commands and feedback at high update rates. In scenarios where cabling is very long or the environment is extremely noisy, designers might still opt for a differential field bus like RS-485 or CAN for inter-module links, but for this robot's internal Pi–ESP32 connection, USB CDC is both practical and adequately resilient.

Signal Integrity Measures: Across all communication interfaces, the design incorporates measures to maintain signal integrity. For I^2C, pull-ups are sized appropriately and trace lengths are controlled to limit edge degradation and susceptibility to noise. For the Pi–ESP32 link, USB uses differential D+/D− signaling, which inherently improves noise rejection compared with single-ended GPIO serial lines; using a short, good-quality USB cable with proper shielding further reduces EMI pickup. On the PCB, communication traces are routed away from high-current paths (motor supply, H-bridge switching nodes), minimizing inductive/capacitive coupling. Where connectors are used (as with sensors), pin assignments can place a ground adjacent to sensitive signal pins to provide a better return path and reduce crosstalk. If testing indicates ringing or excessive edge rates on any fast digital line, small series resistors or common-mode filtering can be added as needed, but only after evaluating actual signal quality. Unused MCU pins are configured to defined states to prevent floating inputs from injecting noise. The end result is that data communication in the robot remains reliable and free from corruption sensors report accurate readings, and commands between the Pi and microcontroller are received correctly. This reliable communication backbone is essential for autonomous operation, as the decisions made by the high-level software are only as good as the data received from the hardware.

3.3.6 Motor Driver Interface and Safety Considerations

The propulsion system of the ADR consists of multiple DC gear motors driven by high-power motor driver modules. Each motor driver module is powered from the **24 V traction power domain**, which in the ADR is formed by **two 12 V batteries connected in series** to provide the bus voltage required by the drivers and motors. This higher-voltage domain enables efficient delivery of power to the motor drivers while reducing current for a given mechanical output, which helps limit cable losses, reduce heating in wiring, and minimize voltage sag under load. The motor drivers (often H-bridge circuits or smart driver ICs) provide the necessary high-current switching to drive the motors forward or reverse and serve as the interface between the low-power control signals and the high-power motor actuators. The ESP32-S3 microcontroller generates logic-level control signals that connect to the motor drivers' inputs. In this design, for each motor the ESP32 provides a Pulse Width Modulation (PWM) signal to command speed and a direction or enable signal to control the motor's drive state (forward, reverse, brake, or coast). For example, the microcontroller outputs might include pins labeled L1_PWM and L1_EN for the left motor, R1_PWM and R1_EN for the right motor, etc., which feed into the driver modules. These signals are 3.3 V logic, which the motor driver modules are designed to accept (often they have 3.3 V or 5 V logic-compatible inputs). By keeping the control interface at logic level, the microcontroller can safely control motor operation without exposing its circuits to the 24 V supply. The motor drivers internally handle the high currents (often tens of amperes) and voltage, using MOSFET transistors and flyback diodes to switch the motor coils on and off. Flyback (freewheeling) diodes or equivalent circuitry in the motor driver are critical, as they safely dissipate the voltage spikes generated when motor coils are switched (back EMF), preventing those spikes from propagating into the rest of the power system.

A key safety feature of the ADR's power distribution is that the 24 V traction domain is **switched through a dedicated 4-channel relay module**, providing an electrically enforced "power cut" capability under fault or emergency conditions. In the implemented wiring, the relay module is used to control the series-battery power path such that the robot can isolate the traction bus quickly if a short circuit is detected, if the control electronics report a power emergency, or if the operator presses the shutdown button. Specifically, the relay channels are assigned to segment the series connection and bus endpoints in a controlled way: **Channel 1 carries the +12 V of the first battery and is considered the effective +24 V output**, **Channel 2 carries the − terminal of the first battery**, **Channel 3 carries the +12 V of the second battery**, and **Channel 4 carries the − terminal of the second battery and serves as the effective 24 V return (the − of the 24 V bus)**. With this arrangement, the relay module can break the series chain and/or disconnect the 24 V endpoints, ensuring that the high-power wiring to the motor drivers is physically opened when the system is placed in a safe state. All relay-controlled lines are designed to open (disconnect) if there is a detected short circuit on the board, or if the ESP32-S3 firmware flags any abnormal power condition (for example, unexpected current draw, undervoltage/overvoltage behavior, or a commanded emergency-stop condition). In addition, a **hardware switch button** is integrated as a user-facing safety control to "close" (enable) or "open" (disable) the vehicle power, providing a manual override that does not rely solely on software behavior and allowing the robot to be shut down immediately when required.

Importantly, the selected relay module includes **opto-couplers (opto-isolators)** on its input channels. This provides electrical isolation between the ESP32-S3's low-voltage GPIO control side and the relay coil/driver circuitry that interfaces with the high-power battery domain. The opto-couplers help protect the microcontroller from dangerous transients, ground bounce, wiring faults, or fault-induced voltage spikes that can occur during switching events or under abnormal power conditions. In effect, the ESP32 controls the relay state using light-coupled signals rather than a direct electrical connection, reducing the likelihood that a high-energy fault on the power side can propagate back into the logic domain and damage the controller or cause unintended resets. This isolation complements other protective measures (such as fusing, proper grounding strategy, and careful routing) and strengthens overall system robustness.

To monitor and close the loop on motor control, encoder feedback from the motors is fed into the ESP32. Each motor is equipped with an encoder (optical or magnetic hall-effect) that produces pulses in proportion to the wheel rotation. These encoder signals are routed via connectors (e.g., J7, J11 on the PCB) to the microcontroller's GPIO inputs, where they are typically processed by hardware timers or interrupt routines to measure speed and distance. With real-time encoder data, the microcontroller can implement closed-loop control algorithms (such as PID control) to adjust the PWM outputs and maintain the desired speed or to synchronize multiple motors. For instance, if the robot commands a certain speed and one wheel lags (perhaps due to terrain or load), the ESP32 can detect fewer encoder pulses from that wheel and increase its PWM duty cycle to compensate. This closed-loop control greatly improves the accuracy and stability of the robot's movements.

Safety and reliability considerations are integral to the motor driver interface design. One key aspect is the electrical separation and proper grounding between the motor drivers and the logic controller, as discussed earlier. The motor drivers share a common ground with the ESP32 (they must, so that the 3.3 V control signals have a reference), but their power supply and high-current ground paths are kept distinct beyond the single ground meet-up point. This prevents large motor currents from flowing through or near the microcontroller ground, which could introduce noise or ground shifts. The physical layout places motor drivers closer to the battery and motors, with short, thick wiring for the high-current loop, whereas the logic wiring to the drivers is kept apart. Additionally, to counteract electromagnetic interference (EMI) from the motors, the design may include snubber circuits or EMI filters on the motor terminals (some motor driver boards have built-in capacitors or RC snubbers). Motor leads are often twisted together and kept short to reduce the loop area that can radiate noise[5]. Such noise, if not managed, can wreak havoc on microcontrollers and sensors, leading to resets or erratic readings[5]. By following good practices—shielding, twisting, filtering—the ADR's design mitigates the impact of motor-induced noise on the rest of the system.

Another safety feature in the motor interface is the inclusion of current sensing or limiting. Many motor driver modules have an analog current sense output or over-current protection. The ADR's design can take advantage of this by reading the current sense (if available) with the ESP32's ADC to detect if a motor is straining or stalled. If a motor draws excessive current, the firmware can issue a shutdown or reduction of power to that motor to prevent damage (this acts as a secondary protection in addition to the fuse on the battery). Thermal considerations are also addressed: the motor drivers are heat-sinked or placed in well-ventilated areas on the robot, and the PCB has thermal relief where necessary to dissipate heat from the H-bridge MOSFETs. The connectors used for motors and drivers are rated for high current and lock in place to avoid disconnection under vibration.

Modularity in the motor interface is achieved with connectors that allow each motor or driver module to be disconnected easily. For instance, if a motor driver board needs replacement, it can be unplugged from the control board and power, without desoldering. This also aids in debugging – one can isolate a motor or driver to test it individually. LED indicators are often present on driver boards (showing status like power on, fault, direction, etc.), providing immediate visual feedback that helps in troubleshooting power issues or signal presence.

In summary, the motor driver interface is designed to translate the microcontroller's low-power commands into high-power action, with a strong emphasis on safety, controlled power distribution, and noise isolation. The ADR's 24 V traction bus is created by series-connected 12 V batteries and is actively controlled through a 4-channel relay module that can isolate the high-power domain under short-circuit or emergency conditions, complemented by a manual shutdown switch. The presence of **opto-coupler isolation** on the relay inputs further protects the ESP32-S3 and logic electronics from dangerous transients originating in the power stage. Through careful separation of power domains, robust interfacing, and protective monitoring, the high-power motor subsystem operates without compromising the stability of the logic circuits. Protective components and feedback loops are in place so that the system can detect and handle fault conditions (like motor stall or over-current) gracefully. These measures reduce the risk of electrical faults and ensure consistent performance even under demanding conditions, such as climbing a slope or navigating uneven terrain which puts varying load on the motors. By combining clear separation of power domains, robust switching-based safety isolation, and closed-loop feedback, the ADR's propulsion control achieves both responsiveness and reliability.

3.4 Sensor Suite and Hardware Components

The autonomous delivery robot is built on a **robust, modular hardware architecture (shown in** Figure. 3.13 **and** Table 1**)** that balances high-level computing with real-time control. A heterogeneous processing approach is employed, featuring a **Raspberry Pi 5** as the main computer and an **ESP32-S3 microcontroller** as a dedicated real-time controller. This design allows computationally intensive tasks (mapping, vision, AI planning) to run on the powerful Raspberry Pi 5, while time-critical control loops are handled separately on the microcontroller to ensure deterministic behavior. A **segregated power distribution** is used: dual lithium-ion battery packs separately supply the high-current drive system and the logic electronics, preventing motor-induced noise from affecting sensitive sensors and processors. All components are tightly integrated under the Robot Operating System (ROS 2) framework, enabling sensor fusion and coordinated control in real time.

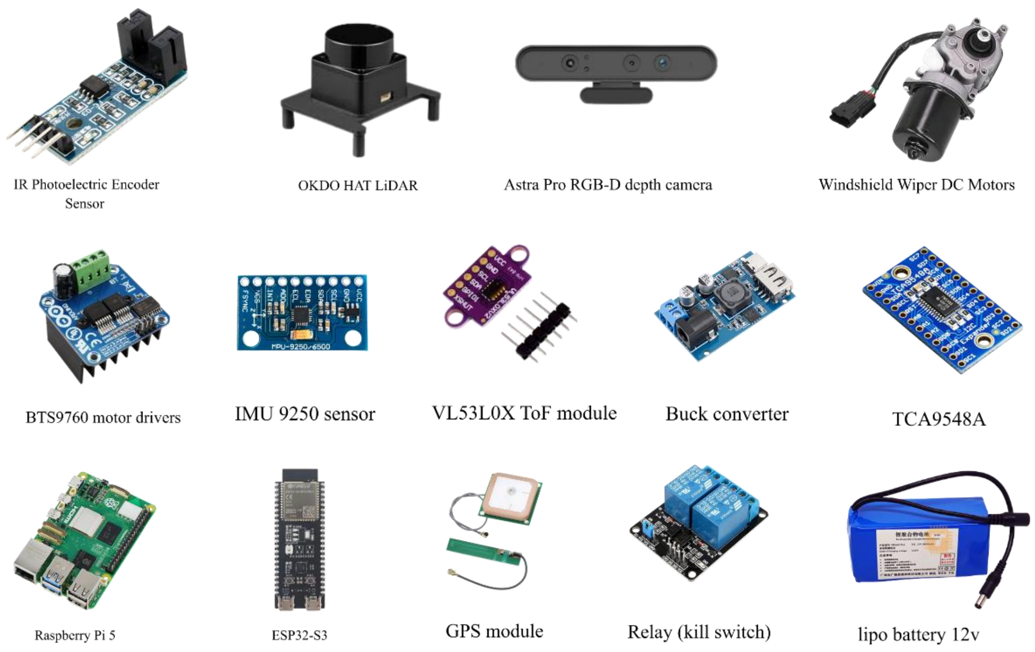


Figure. 3.13 Hardware components of the autonomous robot

Component	Function	Interface	Key Specifications
OKdo HAT LiDAR (LD06)	2D LiDAR scanner for obstacle detection and SLAM mapping. Provides distance measurements around the robot.	Serial/UART (via Pi HAT)	360° scanning; 0.02–12 m range; 5–13 Hz scan rate; ~3 cm accuracy.
Orbbec Astra Pro RGB-D Camera	RGB-D depth camera for vision-based perception, object recognition, and semantic mapping]. Captures color images and depth data.	USB 2.0	RGB output 1280×720 @30 fps; Depth output 640×480 @30 fps; Depth range ~0.6–8 m; FOV 60°×49° (73° diag).
MPU-9250 IMU	9-axis Inertial Measurement Unit for orientation (attitude) and motion sensing. Aids odometry and state estimation by measuring acceleration, rotation, and magnetic heading.	I²C (also SPI capable)	3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer; ±16 g, ±2000 °/s ranges (gyro/accel); up to 400 kHz I²C.
VL53L0X ToF Sensors	Short-range Time-of-Flight laser sensors for precise local proximity sensing and obstacle detection. Placed around the robot for blind-spot coverage and maneuvering in tight spaces.	I²C (via TCA9548A mux)	Range 3–2000 mm (≈2 m)[12]; ±3% accuracy; ~25° field of view; 20–30 ms per reading.
Ai-Thinker GP-02 GPS Module	Multi-constellation GNSS receiver for global positioning in outdoor environments. Provides latitude/longitude and velocity for long-range navigation.	UART (NMEA serial)	Supports GPS, GLONASS, BeiDou, Galileo (GNSS); -162 dBm sensitivity; up to 10 Hz update rate; built-in ceramic antenna.
Raspberry Pi 5	High-level processor running Linux and ROS 2 for AI and autonomy. Handles SLAM mapping, path planning, computer vision, and sensor fusion tasks.	Various (USB, CSI, Ethernet, etc.)	Quad-core CPU @ ~2.4 GHz + GPU; 4 GB RAM (typ.); 2 × USB 3.0, 2 × CSI camera ports; runs ROS 2 nodes for perception and planning.
ESP32-S3-DevKitC-1 (N8R8)	Real-time microcontroller board for low-latency motor control and sensor interfacing. Runs embedded firmware (FreeRTOS) to control actuators and read sensors deterministically.	GPIO, I²C, UART, SPI (to sensors & drivers)	240 MHz dual-core MCU; 512 KB RAM, 8 MB flash + 8 MB PSRAM; Wi-Fi/Bluetooth connectivity; <5 µs interrupt latency for control loops.
TCA9548A I²C Multiplexer	Eight-channel I²C switch expanding the I²C bus capacity. Allows multiple I²C sensors with identical addresses (e.g. multiple VL53L0X modules) to operate on one controller bus without address conflicts.	I²C control (0x70–0x77 addr)	1-to-8 bidirectional I²C channels; 1.65–5.5 V logic; up to 400 kHz I²C speed per channel; enables 8 same-address devices on one bus.
BTS9760 Dual H-Bridge Drivers	High-power motor driver modules (two boards) for the four drive motors. Each driver controls two DC motors (left or right side pair) with PWM signals, providing forward/brake/reverse control and protection features.	PWM + Direction inputs from MCU; Power output to motors	Up to 27 V supply; 43 A peak output per channel; 25 kHz PWM capable; built-in over-current, over-temp, short-circuit protection.
hield Wiper DC s	**hield Wiper DC Motors (with ers)**	**hield Wiper DC s (with Encoders)**	**hield Wiper DC Motors Encoders)**
Dual Li-ion Battery System	Two separate 18650 Li-ion battery packs (2S configuration) for power supply. One battery exclusively powers the motors and drivers, and the other powers logic (Pi,	Power (DC supply rails)	2 × 18650 cells per pack (7.4 V nominal each); ~2500–3000 mAh per pack; each pack with 15 A fuse

	MCU, sensors). This isolation prevents voltage sag and electromagnetic interference from the motors affecting the computing hardware		and cutoff relay for safety; separate ground reference connected at a single point (to avoid ground loops).
Buck Regulators (DC–DC)	High-efficiency step-down regulators converting battery voltage to stable 5 V and 3.3 V rails. Ensure a consistent supply for the Raspberry Pi, ESP32, and sensors despite varying battery output or motor load surges.	Power (DC output)	Synchronous buck converters (90%+ efficiency); 5 V @ 5 A for RPi and peripherals; 5 V @ 3 A for MCU board and sensors; <50 mV output ripple under load.
Battery Management System (BMS)	Protection and management circuit for the Li-ion batteries. Monitors cell voltages and current to prevent over-charge, over-discharge, and overload. Also handles cell balancing and provides a safe cut-off in fault conditions.	Power management (battery interface)	2 S Li-ion BMS, 5 A continuous rating (per pack); UVLO cut-off at ~3.0 V/cell, charge cutoff at 4.2 V/cell; balancing current ~50 mA; onboard temperature monitoring; status LEDs for fault indication.

Table 1 Hardware components with their functions, interfaces, and key specifications.

3.4.1 Sensor Suite

The robot is equipped with a **multi-modal sensor suite** to perceive its environment and track its motion. These sensors provide complementary data that is fused together for reliable navigation and obstacle avoidance:

- **OKdo HAT LiDAR (LD06)** – A 360° scanning LiDAR mounted as a Raspberry Pi HAT. It continuously measures distances to surrounding obstacles, producing a high-accuracy 2D point cloud of the environment. The LD06 LiDAR has a range of up to 12 meters and an accuracy on the order of a few centimeters, which is crucial for Simultaneous Localization and Mapping (SLAM) and global obstacle detection. By rotating at 5–13 Hz, it gathers thousands of distance points per second to build detailed occupancy maps. This LiDAR data forms the geometric basis for mapping and navigation algorithms.
- **Orbbec Astra Pro RGB-D Camera** – An RGB-D (color + depth) camera that provides rich visual information. The Astra Pro uses an infrared depth sensor paired with an RGB camera to output synchronized color images and depth maps. It operates over a range of roughly 0.6 m to 8 m, capturing depth at VGA resolution (~640×480 @30 fps) and color at 720p HD (1280×720 @30 fps). This sensor enables **object recognition and semantic perception** beyond the geometric data of LiDAR. For example, it can detect pedestrians, read signage, or recognize drop-off locations via computer vision, while the depth channel allows the robot to gauge object sizes and free space. The RGB-D camera's data is used in path planning to differentiate navigable areas from obstacles by their visual appearance in addition to geometry.
- **MPU-9250 IMU** – An Inertial Measurement Unit combining a 3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer in one package. The IMU provides continuous estimates of the robot's **attitude (orientation) and linear acceleration**, which are invaluable for odometry correction and state estimation. For instance, when the robot's

wheels slip or during fast maneuvers, the IMU can sense the actual rotation and tilt of the robot, helping to correct the wheel odometry. The MPU-9250 interfaces via I²C (or SPI) to the controller and outputs data at a high rate (hundreds of Hz). Its magnetometer also gives a compass heading, which can help reduce drift in heading during long runs. Overall, the IMU's nine-axis data is fused with wheel encoder and GPS information to yield a robust pose estimate for the robot.

- **VL53L0X Time-of-Flight Sensors** – Several VL53L0X laser-ranging modules are placed around the robot as **proximity sensors** for short-range distance measurement. These sensors emit eye-safe laser pulses and measure the return time to compute distance to the nearest object within a narrow field-of-view (~25°). Each VL53L0X can measure distances from ~3 cm up to about 2 meters with ~±3% accuracy, independent of target color or lighting. They are used to detect obstacles in the robot's immediate vicinity that might be below or outside the LiDAR's planar scan (for example, curbs, small objects, or obstacles very close to the robot's base). The ToF sensors enable **precise maneuvering** in tight spaces – for instance, detecting when the robot is right next to a wall or docking with a station. Multiple VL53L0X units are employed (e.g. front, sides, and rear), and they are managed via an I²C expander to avoid address conflicts (each sensor shares the same I²C address by default).
- **Ai-Thinker GP-02 GPS Module** – A precision GNSS receiver providing **global positioning** data for outdoor navigation. The GP-02 module tracks multiple satellite constellations (GPS, GLONASS, BeiDou, Galileo, etc.) concurrently for improved accuracy and coverage[14]. It features an integrated ceramic patch antenna and can output location updates at up to 10 Hz, with typical positional accuracy on the order of a few meters (augmented by SBAS/assisted-GPS when available). The module communicates via UART, streaming standard NMEA sentences (latitude, longitude, speed, etc.) to the robot's controller. In this autonomous delivery robot, the GPS is mainly used for **global reference** – for example, obtaining an absolute position fix when operating outdoors or to initialize/filter the robot's pose within a large-scale map. While GPS alone is not sufficiently accurate for precise navigation, when combined with the on-board sensors (IMU, wheel odometry, LiDAR), it helps bound long-term drift and enables rough navigation between distant waypoints or recovery if the robot gets lost.

Together, this sensor suite provides **comprehensive situational awareness**. The high-level ROS 2 software continuously fuses LiDAR scans, camera data, IMU readings, ToF distances, and GPS coordinates to form a reliable understanding of the robot's pose and its surroundings. This multi-sensor approach ensures that no single point of failure (e.g., GPS dropout or wheel slip) will compromise the robot's navigation accuracy, as the other sensors can compensate.

3.4.2 Sensor Models and Performance Calculations

To maximize perception and navigation reliability, each sensor stream is converted from raw measurements into physically meaningful quantities and then filtered to match the robot's speed, expected noise sources, and the computational limits of the Raspberry Pi 5 and ESP32-S3. The following models and practical calculations are used to tune each sensor for best performance within ROS 2.

OKdo HAT LiDAR (LD06)

- Polar-to-Cartesian conversion for scan points :
$$x\_i \ = \ r\_i \, cos(\theta\_i), \qquad y\_i \ = \ r\_i \, sin(\theta\_i).$$
- Rigid-body frame transform into the robot base frame:
$$p\_base \ = \ R\_(base < -lidar) \, p\_lidar \ + \ t\_(base < -lidar).$$
- Motion distortion estimate during one scan:
$$\Delta x \ \approx \ v \ \cdot \ T\_scan \ \ (keep \ \Delta x \ small \ relative \ to \ map \ resolution).$$
- Nearest-obstacle metric used by safety layers and planners:
$$d\_min \ = \ min\_i(r\_i), enforce \ d\_min \ > \ d\_safe.$$
- Filtering practice: reject invalid ranges outside $[r\_min, r\_max]$ and optionally apply median filtering to suppress outliers.

Orbbec Astra Pro RGB-D Camera

- Depth-to-3D (pinhole model):
$$X \ = \ (u \ - \ c\_x) \cdot Z \ / \ f\_x, \qquad Y \ = \ (v \ - \ c\_y) \cdot Z \ / \ f\_y, \qquad Z \ = \ Z.$$
- Projection (for validation and alignment):
$$u \ = \ f\_x \cdot X/Z \ + \ c\_x, \qquad v \ = \ f\_y \cdot Y/Z \ + \ c\_y.$$
- Depth gating for reliable perception: accept Z only within $[Z\_min, Z\_max]$ (e.g., the camera's stable range).
- Point-cloud decimation: $downsampling$ by factor k reduces point count by $\sim k\text{^}2$, improving real-time performance.

MPU-9250 IMU

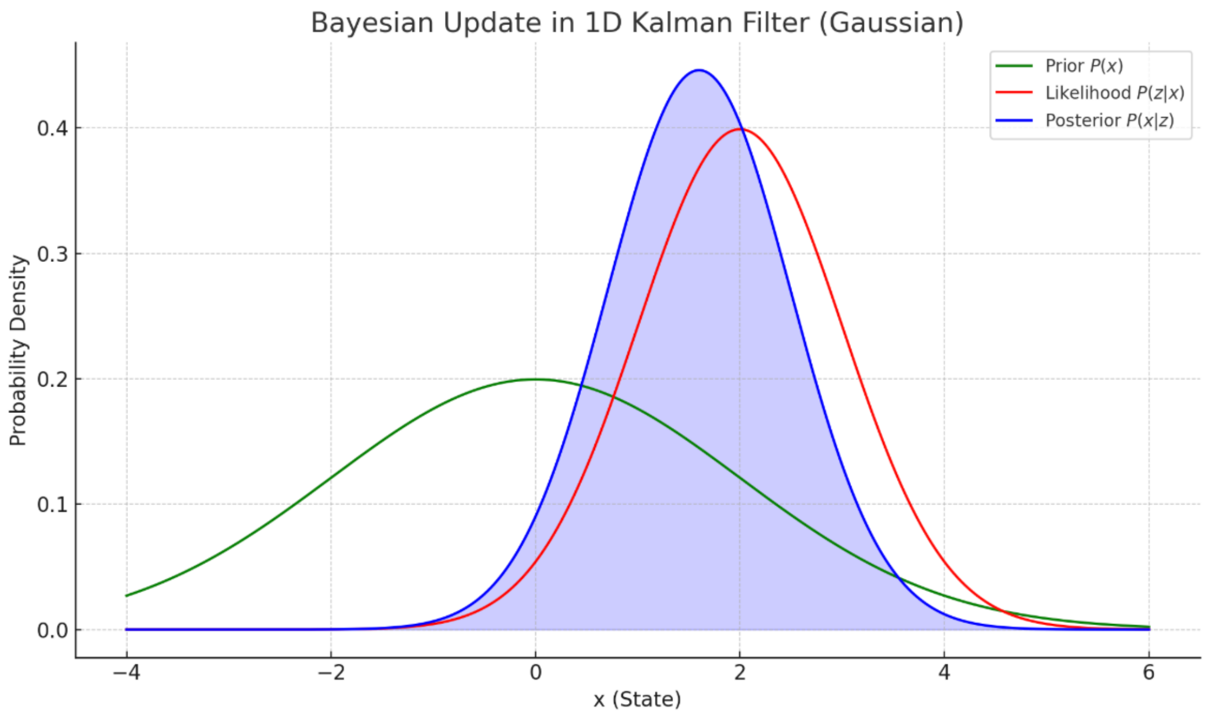


Figure. 3.14 kalman filter reducing error graph

- Measurement model:

$$\omega\_meas = \omega\_true + b\_g + n\_g, \ \ a\_meas = a\_true + b\_a + n\_a.$$

- Quaternion integration (conceptual):

$$q\_dot = 0.5 \cdot q \otimes [0, \omega]\text{^}T, integrated\ with\ \Delta t\ from\ time\ stamps.$$

- Complementary fusion idea (roll/pitch stabilization):

$$\theta\_est = \alpha(\theta\_est + \omega\Delta t) + (1-\alpha)\theta\_acc.$$

- Startup gyro-bias estimate while stationary:

$$b\_g \approx (1/N)\ \Sigma\_k\ \omega\_meas, k\ \ (average\ over\ N\ samples).$$

- Gravity compensation for linear acceleration:

$$a\_lin \approx a\_meas - R\text{^}T\ g, with\ g = [0,0,9.81]\ m/s\text{^}2\ in\ the\ world\ frame.$$

- Kalman filter (Figure. 3.14) (EKF) update equations (state fusion core):
- Prediction: $xk \mid k-1 = f(xk-1, uk), Pk \mid k-1 = FPk-1FT+Q.$
- Innovation: $yk = zk - h(xk \mid k-1), Sk = HPk \mid k-1HT+R.$
- Kalman gain: $Kk = Pk \mid k-1HTSk-1.$
- Update: $xk = xk \mid k-1 + Kkyk, Pk = (I-KkH)Pk \mid k-1.$

VL53L0X Time-of-Flight Sensors (Near-Field Safety Ring)

The ADR uses a short-range ToF safety ring to detect close obstacles that can sit below the LiDAR scan plane or inside the robot's immediate blind spots. In the current placement, one sensor is mounted at the front, two cover the sides, and two cover the rear to enable fast emergency-stop responses from the ESP32-S3 during low-speed docking and tight maneuvers.

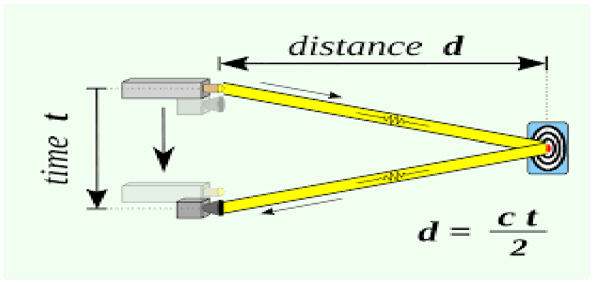


Figure. 3.15 distance calculation for time of flight

- Time-of-flight distance principle:

  $d \approx c \cdot \Delta t$ / 2 (the sensor internally returns d in millimeters) as shown in (Figure. 3.15).

- Multiplexed polling rate (N sensors): $f\_sensor \approx 1 / (N \cdot (T\_meas + T\_sw))$ where T_meas is timing budget and T_sw is mux switching overhead.

- Robust filtering: use median filtering over a small window to reject outliers:

  $$d\_filt = median(d\_\{t-k\} \dots d\_t).$$

- Emergency-stop threshold from braking physics: $d\_stop = v\hat{}2/(2a\_brake) + v \cdot t\_latency + d\_margin;\ stop\ if\ any\ d\_filt < d\_stop$ in the direction of motion.

- Direction-aware logic: emphasize rear sensors while reversing and side sensors during turning to reduce collision risk in tight spaces.

Ai-Thinker GP-02 GPS Module

- Great-circle distance between two fixes (Haversine):

  $$d = 2R \cdot asin(\ sqrt(\ sin\hat{}2(\Delta\varphi/2) + cos\varphi 1 \cdot cos\varphi 2 \cdot sin\hat{}2(\Delta\lambda/2)\ )\ ).$$

- Coarse velocity estimate:

  $v \approx d/\Delta t$ (useful for sanity checks against wheel odometry outdoors).

- Local-frame fusion: convert latitude/longitude to a local tangent frame (ENU) relative to an origin so it can be fused in meters with IMU and wheel odometry.

- Fusion weighting: set GPS measurement covariance to reflect meter-level uncertainty so the estimator does not over-trust GPS during short-term maneuvers.

Wheel Encoders (Odometry and Closed-Loop Control)

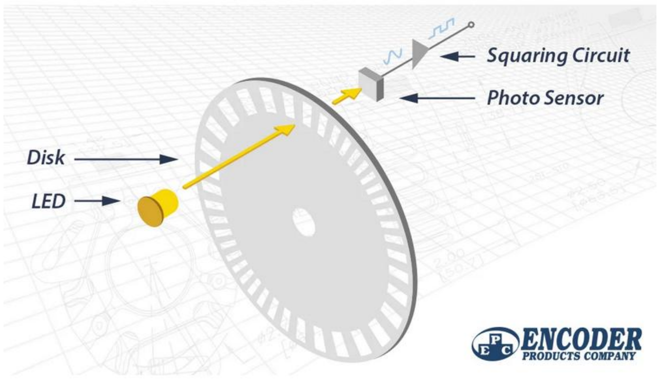


Figure. 3.16 **IR wheel encoder**

- Ticks to wheel angle: $\Delta\theta = 2\pi \cdot \Delta N / CPR$, where CPR is counts per wheel revolution.
- Wheel travel: $\Delta s = r \cdot \Delta\theta$.
- Differential-drive update: $\Delta\psi = (\Delta s\_R - \Delta s\_L)/L$, $\Delta s = (\Delta s\_R + \Delta s\_L)/2$, then update (x, y, ψ) using Δs and Δψ.
- Slip monitoring: compare yaw-rate from encoders ($\Delta\psi/\Delta t$) with IMU gyro yaw-rate to detect wheel slip and increase reliance on inertial/scan-based localization when needed.

These equations and tuning calculations provide a consistent basis for configuring ROS 2 parameters (frame transforms, filter covariances, update rates, and safety thresholds) so that each sensor contributes reliable information to the overall perception and navigation stack.

3.4.3 Processing Units

Effective autonomous operation is enabled by a two-tier computing system, separating heavy-duty processing from real-time control tasks and ensuring deterministic low-level behavior alongside high-level intelligence:

- **Raspberry Pi 5 (Main Computer):** The Raspberry Pi 5 single-board computer serves as the high-level brain of the robot. It runs a full Linux OS and the ROS 2 (Robot Operating System) middleware, hosting modules for perception, planning, and higher-level decision making. With a quad-core 64-bit processor and a built-in GPU, the RPi 5 is capable of running computationally intensive algorithms in real time. Key software components on the RPi 5 include SLAM (for mapping and localization), computer vision (object detection, traffic light/sign recognition) using deep learning models, and global path planning

algorithms. The RPi 5's GPU acceleration is leveraged for tasks such as neural network inference (e.g., YOLO for object detection) and accelerating image processing pipelines. By handling these complex workloads, the Raspberry Pi relieves the microcontroller of heavy computation and allows the robot to make intelligent decisions based on rich sensor data. The RPi 5 interfaces with sensors such as the LiDAR and RGB-D camera directly via USB/CSI connections and is network-enabled (Wi-Fi/Ethernet) for remote monitoring or cloud connectivity if required.
Importantly, the Raspberry Pi also acts as the **ROS 2 gateway** to the real-time controller: it runs the **micro-ROS agent** and bridges the microcontroller data streams into native ROS 2 topics and services. In the ADR implementation, the Pi communicates with the ESP32-S3 over a **USB CDC-ACM (USB-to-Serial virtual COM) link** (typically enumerating as `/dev/ttyACM*`). Although this appears as a "serial device" to Linux, electrically it is a USB differential link, which is generally **more robust and higher-throughput than a raw TTL UART connection**, and it simplifies wiring while improving noise immunity inside a motor-driven platform.

- **ESP32-S3 Microcontroller (DevKitC-1):** The ESP32-S3 is a 32-bit microcontroller that manages all real-time control loops and low-level hardware interfacing. It is responsible for tasks that require precise timing and fast response, such as reading wheel encoder ticks, computing motor-control PID loops, and generating Pulse Width Modulation (PWM) signals for the motor drivers. By using the ESP32-S3 for these tasks, the system achieves hard real-time performance: motor control updates occur at a fixed high frequency (on the order of a few kHz), largely unaffected by the non-deterministic workload on the Raspberry Pi. The ESP32 runs a lightweight Real-Time Operating System (FreeRTOS), which schedules a motor control task, a safety monitoring task, and a communication task in parallel. For example, one task continuously receives high-level velocity commands from the Pi (now carried over **USB CDC**), another reads current speed from motor encoders and executes PID control, and a safety task monitors power status and system health.
  In addition, the ESP32-S3 hosts the **micro-ROS client runtime**, enabling the microcontroller to publish and subscribe to ROS-style interfaces (topics/services) while keeping deterministic timing for control. Sensor and actuator data (e.g., encoder feedback, ToF safety distances, IMU samples, motor telemetry) can be published from the ESP32-S3 as micro-ROS messages and then forwarded by the micro-ROS agent on the Raspberry Pi into the main ROS 2 computation graph. This architecture reduces custom protocol overhead and produces a clean, modular interface between high-level autonomy and low-level control. The ESP32 interfaces with sensors directly: it connects to the IMU and ToF rangefinders over I^{2}C and reads wheel encoders via GPIO interrupts/timers. UART remains available on the ESP32 for peripherals such as GPS, but the **primary Pi↔ESP32 system link is USB CDC**, selected for its practical throughput, robustness, and simplified interconnect inside the robot.

This division of labor between the Raspberry Pi and ESP32 is critical: it guarantees that high-level processes (like SLAM and vision inference) do not interrupt or delay the low-level motor control and safety loops, maintaining stable, responsive, and safe operation of the robot at all times.

3.4.4 Actuation and Feedback

The robot's drive and steering mechanism is implemented through a set of DC motors with closed-loop control, overseen by robust motor driver hardware:

- **Windshield Wiper DC Drive Motors (with External Encoders)**

The locomotion system of the autonomous delivery robot utilizes **automotive-grade windshield wiper DC motors**, replacing the initially considered GM25-370 micro gear motors. This design decision was driven by the need for **higher torque, improved durability, and reliable operation in outdoor environments**, which are critical for a full-scale autonomous delivery platform.

Windshield wiper motors are specifically designed for **continuous operation under high mechanical load**, making them well-suited for mobile robotic applications that require sustained torque, smooth motion, and resistance to dust, vibration, and temperature variations. Each motor integrates a **high-ratio worm gear transmission**, providing substantial torque amplification while inherently preventing back-driving, which improves stability when the robot is stationary on slopes.The robot employs a **differential drive configuration**, where one motor drives the left wheel assembly and another drives the right wheel assembly. Rotational speed and traveled distance are measured using **external quadrature encoders** mounted on the motor shaft or wheel axis. These encoders provide real-time velocity and position feedback to the ESP32-S3 microcontroller, enabling **closed-loop speed control and precise odometry estimation**.

Motor actuation is handled by high-current H-bridge drivers, allowing bidirectional control, dynamic braking, and PWM-based speed modulation. This configuration ensures smooth acceleration, accurate trajectory tracking, and robust operation under varying payload and terrain conditions.

- BTS9760 Motor Driver Modules:

To drive the motors, the robot uses two high-current H-bridge motor driver modules based on the BTS7960/BTS9760 driver ICs. Each module can control two DC motor channels, so two modules cover the four motors (one module handles the left-side motors, the other the right-side motors). These drivers are chosen for their ability to handle the stall currents of the motors and their built-in protection features. Each BTS9760 H-bridge can operate at up to ~27 V and deliver peaks of 43 A, far above what the small gear motors typically draw – this overhead ensures reliability and keeps the driver running cool. The driver modules accept logic-level control signals from the ESP32: typically two PWM inputs (for forward/reverse drive) and an enable/disable pin for each motor channel. The integrated safeguards in the BTS9760 include over-current limiting, over-temperature shutdown, under-voltage lockout, and short-circuit protection. This means if a motor stalls or a wiring fault occurs, the drivers will automatically limit the current and prevent damage, improving the system's robustness during demanding maneuvers or error conditions. The motor drivers also provide a form of active braking (by dynamically adjusting the H-bridge outputs) which helps the robot decelerate quickly or hold a stopped position on a slope without back-driving the motors. The combination of these drivers with the feedback from encoders creates a reliable actuation system: the microcontroller can precisely modulate motor power and rapidly respond to any deviation or fault, ensuring the robot moves as commanded and can safely stop when needed.

3.4.5 Power Management

A stable and resilient power supply is vital for an autonomous robot, especially one with both high-power motors and sensitive electronics. The project implements a **dual-battery system with regulation and protection** to meet these needs:

- **Dual Battery System:** The robot uses two separate lithium-ion battery packs (each comprised of 18650 cells in series) to isolate the power domains. One pack exclusively supplies the **drive motors and motor drivers**, while the other pack feeds all **logic-level electronics** (Raspberry Pi, ESP32, sensors, etc.). By splitting the power this way, the large current spikes and noise from the motors (for example, when starting up or climbing a bump) are prevented from disturbing the voltage supplied to the computing components. The motor battery is a high-current source (nominal ~7.4 V or higher) capable of delivering several amps continuously. The logic battery is used to provide a clean and stable source for 5 V and 3.3 V regulators. Both battery packs are identical in voltage and capacity to allow balanced performance. This dual-battery architecture significantly improves **electrical noise isolation** and ensures that a dip in motor battery voltage (due to a heavy load) does not cause the Raspberry Pi to brown-out or reset. Additionally, each battery pack is equipped with appropriate **fusing and a cutoff switch** for safety: automotive blade fuses in series with each pack protect against short-circuits, and an emergency cutoff relay (triggered by a microcontroller or manual switch) can disconnect the motor battery in case the robot needs to be immediately halted.
- **Buck Regulators (DC–DC Converters):** The raw battery outputs are converted to the required supply rails by high-efficiency buck regulators. A primary buck regulator steps the ~7.4 V from the logic battery down to a stable 5 V which powers the Raspberry Pi 5 and USB peripherals. This regulator is rated for the Pi's peak current draw and provides a very flat 5.0 V output to avoid undervoltage issues on the Pi (which can cause performance throttling). Another regulator provides 5 V for the microcontroller board and sensors (or this may be shared with the Pi rail if appropriately sized), and on the ESP32 board further LDO regulators drop 5 V to 3.3 V for the MCU and I/O. Using switching buck converters with >90% efficiency minimizes heat dissipation and maximizes battery life. These converters are chosen to handle **dynamic loads** – for example, when the Pi's CPU peaks or multiple sensors draw current simultaneously, the regulator holds the 5 V line steady. The design also includes ample decoupling capacitors on the 5 V and 3.3 V lines near the major components. These capacitors smooth out any residual voltage ripple from the regulators and filter out noise from the motors or digital switching. As a result, the supply lines remain within tight voltage tolerance, which is crucial for the Pi and sensors to operate reliably.
- **Battery Management System (BMS):** Each lithium-ion battery pack is monitored and protected by a BMS circuit. The BMS performs several critical functions: it prevents **over-charge** by ensuring charging is cut off when any cell reaches 4.2 V, and avoids **over-**

discharge by shutting off the load if any cell falls below a safe threshold (around 3.0 V). It also balances the cells in the pack, equalizing their voltages during charging to prolong battery life and capacity. The BMS provides **over-current protection** as well, so if the current draw exceeds a set limit (indicating a possible short or stall), it will disconnect the output to protect the cells. Temperature sensors on the BMS board monitor the pack's heat, and if the pack temperature goes out of range (for instance, due to environmental conditions or heavy use), the system can take action to cool down or shut off the robot. Overall, the BMS ensures the **safety and longevity** of the battery packs, which is important in a robot that might operate for extended periods unsupervised. The power management design also includes status indicators (LEDs or a voltage/current sensor feeding into the microcontroller) so the robot can report its battery levels and warn when it needs recharging. In summary, through careful power segregation, robust regulation, and active battery management, the robot's electrical system maintains stable operation and safeguards both the hardware and users from power-related faults.

3.4.6 Communication and Integration

All the above components are integrated into a cohesive system through carefully planned communication channels and interfaces. The robot's architecture ensures that sensors, processors, and actuators can exchange information reliably, enabling the ROS-based autonomy software to coordinate everything. Key aspects of the communication and integration design include:

- **I²C Sensor Network with Multiplexer:** Many of the lower-level sensors (IMU, ToF rangefinders, etc.) use the I²C bus to communicate with the microcontroller. To accommodate multiple I²C devices especially several identical VL53L0X ToF sensors, which share the same fixed address the design incorporates a **TCA9548A I²C multiplexer**. This chip creates eight separate I²C channels from one master I²C bus. The ESP32 selects which channel (or combination of channels) to activate via the TCA9548A, effectively **time-multiplexing** access to sensors that would otherwise conflict on the bus. For example, each VL53L0X module can be wired to a different channel of the multiplexer, allowing the controller to query each sensor in turn without address collisions. The multiplexer itself is controlled by a simple I²C command (at address 0x70 by default) to switch channels. Using this approach, up to 8 I²C sensors with duplicate addresses can seamlessly coexist, greatly expanding the sensor suite. The I²C bus operates at 3.3 V logic (through the ESP32) and is pulled up with resistors on the multiplexer board. Short wiring runs and proper shielding are used to maintain signal integrity for the IMU and ToF sensor data lines. The result is a clean and organized sensor network where the microcontroller can reliably poll each sensor in a round-robin fashion, obtaining fresh IMU readings at ~100 Hz and distance readings from each ToF sensor at ~30 Hz, for instance.
- **High-Level to Low-Level Controller Link:** The Raspberry Pi 5 and the ESP32 microcontroller communicate over a dedicated serial link (UART) for command and data exchange. The ESP32 runs a communication task listening on this UART for incoming messages (e.g. velocity commands or high-level directives) from the Raspberry Pi. In the ROS 2 architecture on the Pi, a node publishes motion commands (such as wheel velocities

or drive torque commands) which are sent to the ESP32 over this serial interface. Conversely, the ESP32 can send feedback data to the Pi for example, current speed, battery status, or emergency stop flags. A simple packet-based protocol (or even ROS 2's rosserial/micro-ROS) is used to structure this communication. This ensures **bidirectional communication**: the high-level planner on the Pi can instruct the motors, and the low-level controller can report back and also signal any critical events (like if an obstacle is too close as detected by a ToF sensor, or if a motor current limit was reached). The UART link is configured at an appropriate baud rate (e.g. 115200 bps or higher) to handle the data throughput with minimal latency. By using a direct wired serial connection, the system avoids the unpredictability of network communications and achieves **deterministic messaging** between the brain and the motor control unit. In case of a communication failure (no heartbeat from the Pi), the ESP32 is programmed to stop the motors for safety.

- **ROS 2 Integration and Data Fusion:** On the integration side of software, ROS 2 serves as the backbone that ties all sensor and actuator data together in a coherent framework. Each sensor either has a dedicated ROS 2 node or driver on the Raspberry Pi (for high-bandwidth sensors like the camera and LiDAR) or is interfaced via the microcontroller and then relayed to ROS. For example, LiDAR scans are published as LaserScan topics at ~5–10 Hz, the Astra camera publishes synchronized image and depth topics, the IMU data is published at ~100 Hz, and the microcontroller (via its serial link) publishes wheel encoder odometry and ToF sensor distances into ROS topics. These diverse streams are then combined: a **sensor fusion node** (running an Extended Kalman Filter or similar algorithm) subscribes to IMU, wheel odometry, GPS, and possibly visual odometry, and outputs a filtered pose estimate of the robot. At the same time, a **localization and mapping node** (SLAM) uses the LiDAR scans (and IMU for motion prior) to build and update a map of the environment. All components synchronize through ROS 2's messaging, which handles the timing and data exchange so that, for instance, the planner always uses the latest available map and pose. The "Communication and Integration" is not just about moving bits of data, but ensuring that **all subsystems operate in unison**: the perception modules feed the planning modules, which in turn send commands to the low-level control, and any feedback or new sensor events loop back into the system. By leveraging ROS 2, the team benefits from a standard, flexible integration method – the LiDAR, camera, IMU, and other sensor drivers are largely off-the-shelf or based on open-source packages, which greatly simplifies development and ensures reliability.

In conclusion, the sensor suite and hardware components are **integrated through a combination of hardware multiplexing and structured communication protocols**. The ROS-based software layer unifies the data and control flows, enabling the autonomous delivery robot to operate as a coordinated whole. Each sensor and hardware module contributes its part from the LiDAR painting a 2D picture of obstacles, to the IMU and encoders tracking motion, to the batteries quietly supplying energy – and the integration fabric ensures these parts work in concert to achieve safe and intelligent autonomous navigation.

3.5 Embedded Control System

The embedded control system forms the real-time foundation of the ADR, bridging high-level autonomy running on the Raspberry Pi 5 with the physical robot hardware (motors, sensors, and safety mechanisms). While ROS 2 on the Raspberry Pi executes computation-heavy functions such as mapping, localization, perception, and global planning, time-critical operations must be handled deterministically at the hardware level. For this reason, the ADR adopts a two-tier architecture where a dedicated microcontroller performs low-latency control and monitoring, and the single-board computer focuses on high-level decision-making as shown in (Figure. 3.17).

At the center of this layer is the ESP32-S3 microcontroller, which is responsible for real-time motor actuation, encoder acquisition, sensor polling, and safety supervision. These functions require consistent timing, fast interrupt handling, and predictable execution that cannot be guaranteed by a general-purpose Linux environment under variable computational load. The embedded controller therefore maintains stable motion control loops, executes rapid obstacle and fault reactions, and ensures that the propulsion subsystem remains safe even when the high-level computer is busy or temporarily unavailable.

To integrate seamlessly with the ROS 2 software stack, the ESP32-S3 hosts a micro-ROS client, enabling structured communication with the Raspberry Pi through a USB CDC (serial-over-USB) link. This approach provides a robust, high-throughput communication channel while keeping wiring simple and noise-resilient in a motor-driven platform. Internally, FreeRTOS is used to organize firmware execution into prioritized tasks (control, sensing, communication, and safety) to avoid blocking delays and to preserve deterministic timing. Together, these design choices produce an embedded control layer that is responsive, reliable, and scalable, supporting autonomous operation while maintaining strong safety guarantees.

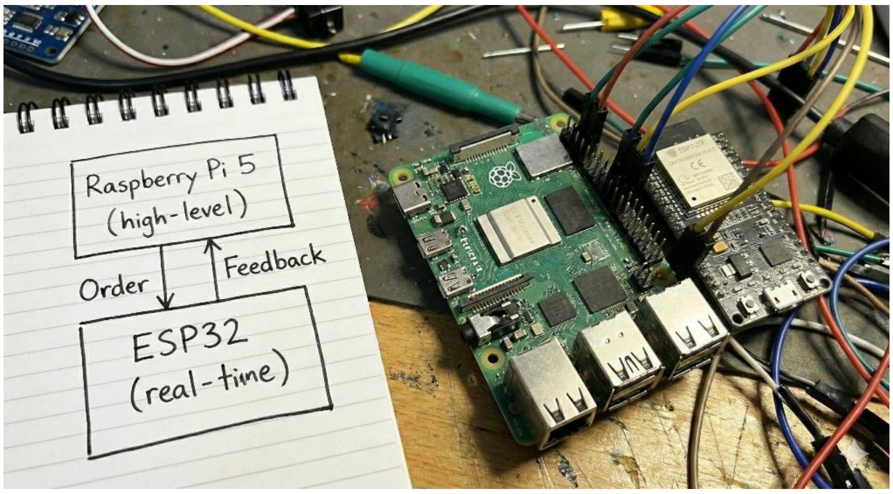


Figure. 3.17 micro-processor and micro-controller combination

3.5.1 Microcontroller Selection Rationale

Selecting the low-level controller for the ADR was driven by three primary constraints: real-time control performance, tight integration with the ROS 2 software stack, and future scalability toward connectivity and smart-service features. The controller must generate deterministic PWM for the motor drivers, process encoder feedback with minimal jitter, and continuously supervise safety conditions (power faults, communication loss, and emergency shutdown). At the same time, it must exchange structured data with the Raspberry Pi 5 efficiently to support a clean separation between low-level actuation and high-level autonomy.

Although STM32 microcontrollers are widely used in robotics due to their strong real-time behavior and mature peripheral set, they were not the optimal fit for this stage of the project. The main limitation was not raw capability, but development overhead and ecosystem alignment under the project's timeline and integration requirements. Achieving the same end-to-end workflow (rapid bring-up, stable ROS 2 communication, and straightforward debugging) typically requires additional tooling steps, board-specific firmware configuration, and transport integration effort. In contrast, the ESP32-S3 provides a highly practical path for fast iteration while still meeting the real-time control needs through dual-core processing, hardware timers, and FreeRTOS scheduling.

The ESP32-S3 was therefore selected because it balances deterministic control with strong system-level integration. It supports the micro-ROS client for ROS-compatible messaging and interfaces to the Raspberry Pi 5 through a USB CDC serial-over-USB connection, which is robust and convenient in a mobile robot environment. Beyond core control, the ESP32-S3 was chosen specifically to enable the ADR's IoT-oriented direction as a smart delivery platform. Its native Wi-Fi/BLE capability allows the robot to expose human-interface functions and client-facing services without adding extra radios or redesigning the electronics . In future operation, this enables features such as live delivery status, basic user authentication/interaction at the robot (e.g., phone-based access or short-range pairing), and lightweight telemetry exchange. The ESP32-S3 also supports practical integration of the GP-02 GPS module for position reporting, making it suitable for transmitting location and system state to a remote user interface or monitoring dashboard as shown in (Figure. 3.18). As a result, the microcontroller choice directly supports both current requirements (real-time control + ROS integration) and planned enhancements (connected operation, location updates, and user interaction) with minimal hardware changes.

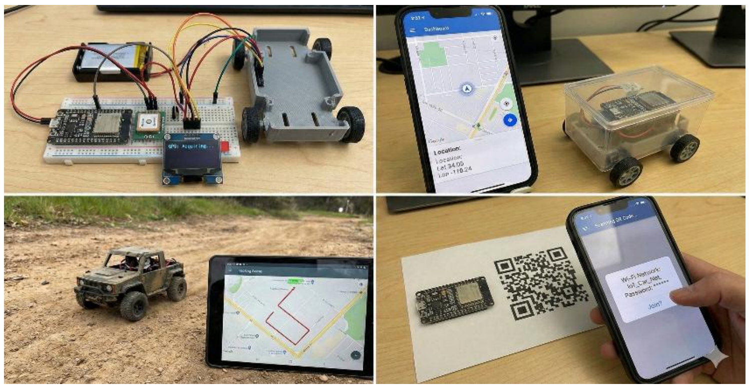


Figure. 3.18 ESP32-S3 with GP-02 sending location via IOT

3.5.2 Functional Requirements for the Low-Level Controller

The low-level controller is responsible for **deterministic actuation and sensing**, acting as the hardware-facing layer that guarantees predictable timing regardless of the Raspberry Pi workload. In the ADR, this controller must execute closed-loop drive control, continuously sample multiple sensors, and enforce safety actions within bounded latency. These requirements guided both the firmware structure and the physical I/O allocation on the ESP32-S3 control board as shown in (Figure. 3.19).

1) Deterministic motor actuation (drive interface requirements)
The controller must generate **stable PWM waveforms** and clean enable/direction logic for the motor driver modules, with update timing that remains consistent during motion (acceleration, turning, load changes). The design exposes dedicated motor-control headers carrying separate PWM and enable lines for each drive channel group—signals labeled **R1-PWM/L1-PWM and R1-EN/L1-EN**, and similarly **R2-PWM/L2-PWM and R2-EN/L2-EN**—to keep the power stage interface simple, low-latency, and independent from high-level computation.
From a control standpoint, this implies the firmware must support: (i) fixed-rate output updates (control loop period), (ii) safe direction transitions (dead-time / braking states as needed), and (iii) immediate "motor inhibit" on faults.

2) High-rate feedback capture (encoder requirements)
Accurate odometry and stable speed control require reliable encoder edge capture without missed pulses. The hardware provides dedicated encoder connectors **ENC1** and **ENC2** into the ESP32-S3 GPIO domain, enabling interrupt/timer-based counting and velocity estimation at high sampling rates.
This creates a firmware requirement for: (i) interrupt-safe counting, (ii) debounce/noise tolerance, and (iii) timestamped velocity computation to support PID control and diagnostics.

3) Multi-sensor acquisition under address constraints (I^2C + multiplexer requirements)
The ADR uses several I^2C sensors, including an IMU and multiple identical ToF modules. Because identical ToF sensors share the same default I^2C address, the low-level controller must support **channelized I^2C access**. The board implements a **TCA9548A I^2C multiplexer**, breaking the bus into multiple SDA/SCL channel pairs (SD0/SC0 … SD5/SC5) to connect the IMU and multiple VL53L0X sensors on isolated branches.
This drives firmware requirements for: (i) deterministic polling schedules (round-robin), (ii) channel selection and error recovery (sensor timeouts), and (iii) basic filtering before publishing (to avoid transient readings triggering false stops).

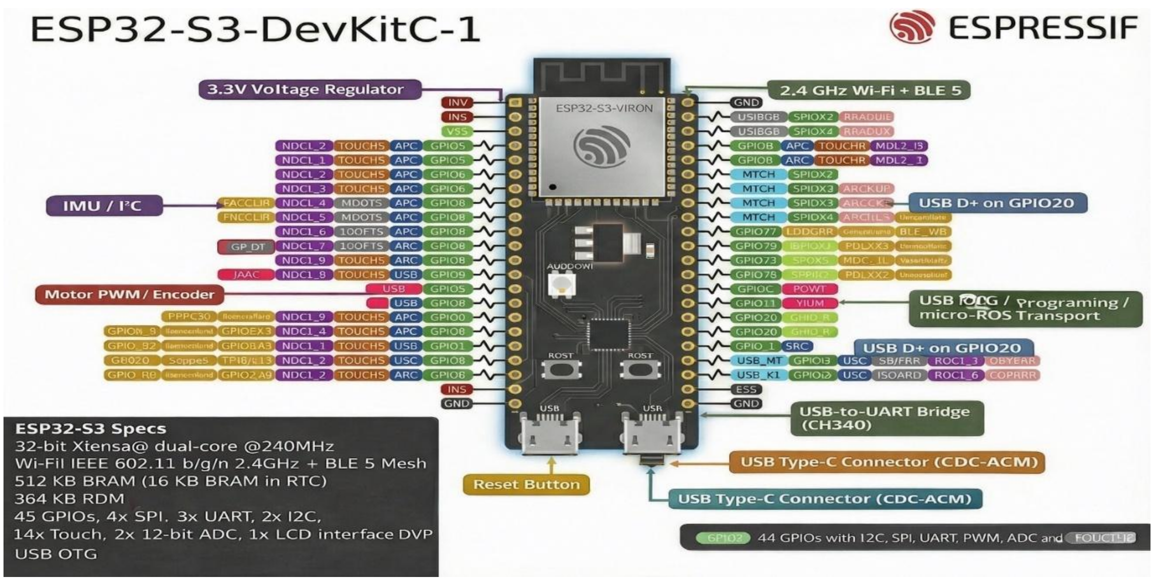
4) GPS integration for outdoor reference (UART peripheral requirement)
For location reporting and outdoor operation support, the controller must ingest NMEA streams from the GP-02 GPS module via a dedicated UART interface (GPS-TX/GPS-RX lines on the GPS header).
Practically, this requires non-blocking UART parsing and rate control so GPS handling cannot delay motor safety logic.

5) ROS-compatible messaging with bounded latency (micro-ROS requirement)
The low-level controller must present its data and control endpoints in a structured way to the ROS

2 system (commands in, telemetry out). This is achieved by running the **micro-ROS client on the ESP32-S3**, so wheel/IMU/ToF/GPS/diagnostic messages can be published as ROS topics and subscribed as command topics without custom packet formats. (Firmware requirement: deterministic publish periods, queue management, and graceful degradation if the link is interrupted.)

6) Robust Pi link without TTL wiring sensitivity (USB CDC requirement)
The Pi↔ESP32 transport must be resilient to noise and easy to integrate inside the enclosure. The ESP32-S3 board routes the native USB differential pair (**USB_D+ and USB_D−**) from the MCU pins (GPIO20/GPIO19) for a **USB CDC serial-over-USB** connection.
This creates a clean requirement set: (i) stable command/telemetry throughput, (ii) clear device enumeration on the Pi (CDC ACM device), and (iii) low overhead compared to adding separate external UART transceivers.

7) Real-time concurrency and non-blocking behavior (execution model requirement)
Because motor control, encoder capture, sensor polling, GPS parsing, and micro-ROS communication must run concurrently, the controller must support a preemptive scheduling model with prioritized tasks. The ESP32-S3's dual-core capability (and RTOS support) is used to isolate time-critical control from communication and background processing, ensuring the control loop remains stable even when telemetry load increases.



ESP32-S3-DevKitC-1
ESPRESSIF
3.3V Voltage Regulator
IMU / I²C
Motor PWM / Encoder
2.4 GHz Wi-Fi + BLE 5
USB D+ on GPIO20
USB JTAG / Programing / micro-ROS Transport
USB D+ on GPIO20
USB-to-UART Bridge (CH340)
USB Type-C Connector (CDC-ACM)
USB Type-C Connector (CDC-ACM)
Reset Button
44 GPIOs with I2C, SPI, UART, PWM, ADC and
ESP32-S3 Specs
32-bit Xtensa@ dual-core @240MHz
Wi-Fil IEEE 602.11 b/g/n 2.4GHz + BLE 5 Mesh
512 KB BRAM (16 KB BRAM in RTC)
364 KB RDM
45 GPIOs, 4x SPI. 3x UART, 2x I2C,
14x Touch, 2x 12-bit ADC, 1x LCD interface DVP
USB OTG

Figure. 3.19 *ESP32-S3-DEVKIT-1 I/O PINS*

3.5.3 ESP32-S3 Hardware Configuration and Key Specifications

The ADR low-level controller is implemented using the **ESP32-S3 (DevKit-class module)** as the primary embedded computing platform. This device provides an effective balance between real-time control capability, peripheral richness, and system integration for a mobile robot that must operate reliably in a mixed-signal environment (high-current motor switching alongside precision sensing). The ESP32-S3 integrates a **dual-core 32-bit processor** capable of operating at **up to 240 MHz**, which enables the firmware to separate time-critical control loops from background tasks such as sensor polling, diagnostics, and communications. This is a practical advantage in robotics, where deterministic timing is required for motor control and safety reactions, while communication and sensor workloads can vary dynamically during operation.

From an interface perspective, the ESP32-S3 configuration is organized to expose clear, dedicated I/O paths for the robot's main low-level functions. Motor driver control is provided through logic-level outputs that support **PWM speed commands** and **enable/direction control**, allowing the high-power driver stages to be commanded without routing power through the controller. Encoder inputs are allocated to GPIOs suitable for **interrupt and timer-based capture**, enabling accurate measurement of wheel speed and distance with low latency. In parallel, the controller supports multi-sensor acquisition using **I²C** as the primary sensor bus, which is ideal for compact wiring and multi-drop connectivity in embedded platforms.

Because the ADR employs multiple identical ranging sensors in the near-field safety layer, the I²C topology is designed to remain scalable and address-conflict-free. An **I²C multiplexer (TCA9548A)** is used to divide the sensor network into independent channels, allowing multiple Time-of-Flight sensors sharing the same default address to coexist. This architecture also improves maintainability: a fault or intermittent behavior on one branch is less likely to degrade the entire bus, and channel-by-channel polling can be scheduled deterministically. Additional peripherals such as the GPS receiver are supported through dedicated serial interfaces, ensuring that continuous streams (e.g., NMEA sentences) can be handled with buffered, non-blocking routines without affecting control-loop timing.

A key design choice in the ADR is the primary communication method between the ESP32-S3 and the Raspberry Pi 5. Instead of relying on raw TTL UART pins for the main inter-processor link, the system uses the ESP32-S3's **native USB interface configured as USB CDC (serial-over-USB)**. This link appears as a standard serial device on the Raspberry Pi while physically benefiting from USB's differential signaling, improved noise immunity, and practical throughput. It also simplifies wiring inside the robot and provides a robust transport for micro-ROS messaging, enabling the microcontroller to publish sensor telemetry and receive motion commands within a structured ROS-compatible communication model.

Overall, the ESP32-S3 hardware configuration delivers the essential capabilities required by the ADR embedded layer: high-frequency PWM generation for propulsion, reliable interrupt-driven encoder acquisition, scalable I²C-based sensor integration (including multiplexed ToF sensing), serial support for positioning sensors, and a robust USB CDC link to the ROS 2 host. These characteristics make the controller well-suited for a safety-critical, sensor-dense delivery robot, where predictable timing, clean interfacing, and reliable communications are mandatory for stable autonomous operation.

3.5.4 Processing Capability and Power/Energy Consumption Considerations

The ESP32-S3 was selected to provide **sufficient computational headroom for real-time control** while maintaining a power profile suitable for continuous battery operation. In the ADR architecture, the microcontroller executes time-critical workloads—PWM generation, encoder sampling, sensor polling, safety evaluation, and micro-ROS message handling—under a deterministic schedule. The ESP32-S3's dual-core processing capability (up to 240 MHz) enables practical workload partitioning, where control and safety functions remain responsive even when communication traffic or sensor activity increases. This separation is essential in a sensor-dense platform: the controller must remain stable under varying runtime conditions, rather than allowing background work to introduce control jitter or delayed fault response.

From an energy standpoint, the ESP32-S3 operates in a fundamentally different regime than the Raspberry Pi 5. The Pi is optimized for high-performance compute and runs a full Linux stack, which results in a comparatively higher and more variable power draw depending on CPU/GPU utilization. In contrast, the ESP32-S3 is designed for embedded efficiency: it can sustain real-time I/O and middleware tasks at a much lower power budget, and it provides flexible clocking and duty-cycling options when full performance is not required. This makes the microcontroller well-suited to "always-on" responsibilities such as monitoring, safety interlocks, and maintaining minimal control behavior even if the high-level processor is busy, rebooting, or temporarily unavailable.

A critical system-level requirement is **electrical and operational safety across power domains**. The ESP32-S3 effectively acts as a protection and control boundary between the **24 V traction subsystem** (motor drivers and motors) and the robot's low-voltage logic domain (microcontroller electronics and the Raspberry Pi 5). In addition to issuing stop commands through control signals, the ESP32-S3 is responsible for enforcing a hardware-level safe state by controlling the **24 V enable path through a relay-based disconnect**. This means the propulsion power is not treated as "always live"; it is **conditionally energized** only when the embedded controller confirms normal conditions (startup checks, valid command stream, and absence of detected faults). If a fault is detected—such as abnormal power behavior, short-circuit suspicion, emergency shutdown request, or loss of control integrity—the controller transitions the system into a safe mode and commands the relay stage to open, physically removing traction power from the drive electronics.

The relay interface itself is designed with additional protection features to ensure the embedded controller remains isolated from high-energy events originating in the traction domain. The **4-channel relay module** incorporates **opto-coupler input isolation as shown** in (Figure. 3.20), which decouples the ESP32-S3 GPIO control lines from the relay coil driver circuitry, reducing susceptibility to noise, ground bounce, and transient coupling during switching. Protective diode elements (for example, flyback suppression across coils and transient paths) further limit inductive kickback and switching spikes, improving reliability and reducing the risk of microcontroller resets or I/O damage during repeated enable/disable cycles. Together, these measures create a layered safety path: software-level decision logic plus a hardware-level power disconnect mechanism, with galvanic isolation that helps preserve the integrity of the control electronics.

In summary, the ESP32-S3 provides the ADR with a control layer that is both computationally capable and power-aware. It sustains deterministic embedded computation for motion and sensing while also serving as a safety authority that supervises and gates the 24 V propulsion domain. By combining efficient real-time processing with relay-controlled power isolation (including opto-coupling and suppression protection), the embedded controller supports stable operation, improved fault tolerance, and safer integration between high-power actuation hardware and the ROS 2 autonomy computer.

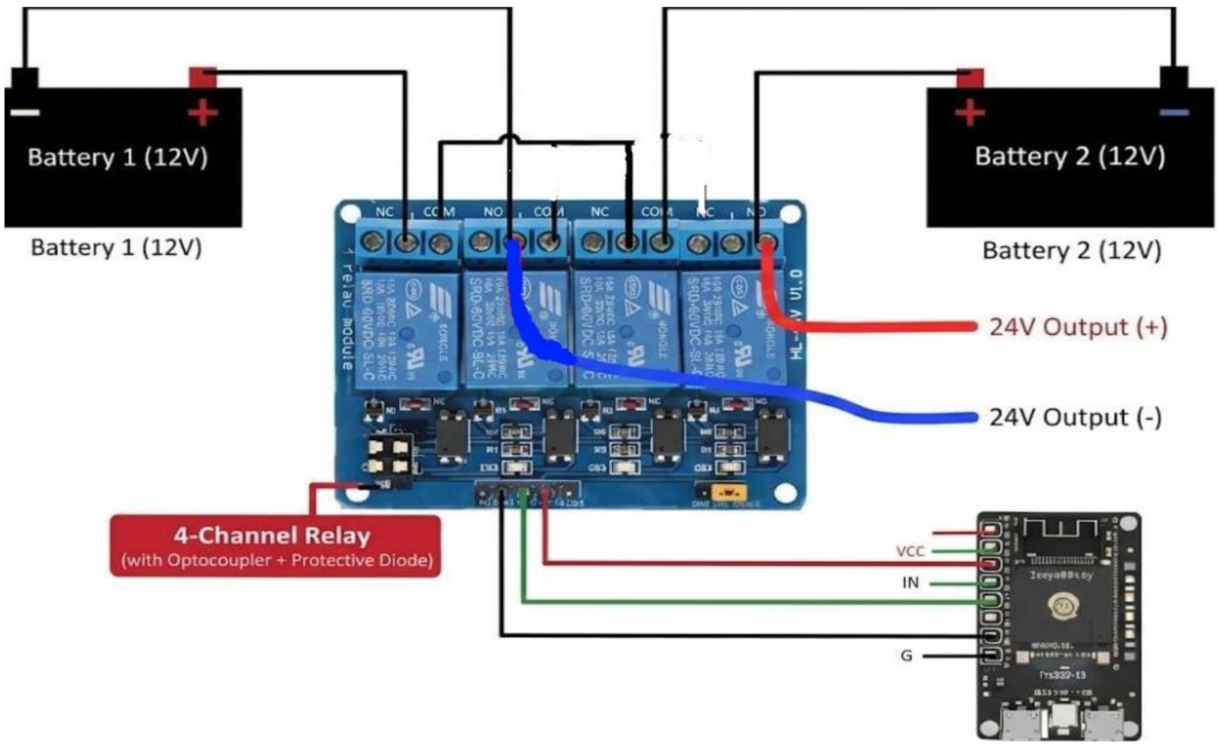


***Figure. 3.20 4 channel relay** opto-coupler module safety*

3.5.5 Real-Time Architecture Using FreeRTOS and micro-ROS Integration

The ESP32-S3 firmware is structured as a real-time system where **FreeRTOS provides deterministic scheduling**, and **micro-ROS provides ROS 2–compatible communication** without forcing the control layer into a "communication-driven" design. The main objective is to guarantee that **encoder capture, velocity estimation, motor/safety reactions, and near-field sensing** are executed on time, while still publishing high-rate telemetry (wheel motion, IMU, and ToF safety distances) to the Raspberry Pi 5 through micro-ROS over USB CDC.

3.5.5.1 FreeRTOS task model and priority strategy

The embedded workload is divided into a small set of bounded-time tasks, each with a clear timing contract:

1. **Encoder + Motion Estimation (Highest Priority)**

- Purpose: capture wheel encoder steps with minimal loss, compute wheel velocity, and update the motion state used for wheel TF/odometry publishing.
- Why highest priority: encoder edges can arrive at high frequency; missing pulses directly degrades velocity estimation and odom/TF stability.
- Execution model: the **encoder capture itself runs in interrupts**, while computation runs in a high-priority task.

2. **Safety + Near-Field Sensing (High Priority)**

- Purpose: periodically read the **5 ToF sensors** for blind-spot protection and enforce safe motion thresholds; read IMU to maintain stable attitude/motion feedback.
- Why high priority: ToF is used for immediate obstacle reactions (slow-down/stop), and IMU contributes to stable state estimation; both must remain responsive under load.

3. **micro-ROS Communication and Publish/Subscribe Handling (Medium Priority)**

- Purpose: publish wheel/IMU/ToF messages and receive velocity commands from the Pi, without ever delaying control or safety tasks.
- Why medium priority: communication can burst; it must be serviced regularly, but it is not allowed to preempt safety or encoder integrity.

This priority layering ensures that if the system becomes busy (e.g., heavy sensor polling + frequent publishing), **control integrity is preserved first**, then safety sensing, then communication throughput.

3.5.5.2 Interrupt-driven encoder capture with zero blocking

Encoders are handled using **GPIO interrupts (or hardware pulse counters/timers when available)**. The interrupt service routine (ISR) is intentionally kept extremely short:

- Increment a tick counter (or update direction-aware counts).
- Optionally latch a timestamp for rate computation.
- Notify a high-priority task using an ISR-safe primitive such as:
  - `vTaskNotifyGiveFromISR()` / `xTaskNotifyFromISR()`
  - or `xQueueSendFromISR()` (only for small fixed-size data)

This design prevents **long ISR occupancy**, which is critical because long ISRs create jitter and can delay other interrupts. All heavier work—velocity computation, filtering, odom delta updates—is performed inside the high-priority encoder/motion task, which wakes immediately after ISR notification. That task then produces the data that is later published to ROS 2 (wheel velocity + wheel TF/odometry), while ensuring no encoder edges are missed.

3.5.5.3 Sensor acquisition task for 5× ToF + IMU without stalling control

The ToF and IMU are polled in a dedicated high-priority sensing task with a fixed-rate schedule, typically implemented using:

- `vTaskDelayUntil()` for stable periodic timing (instead of `vTaskDelay()`)
- A bounded read sequence (no unbounded loops)

To avoid blocking behavior, sensor reads are designed with the following rules:

- **I²C transactions are time-bounded** (timeouts enabled). If a device does not respond, the task records a fault flag and moves on instead of waiting indefinitely.
- The ToF array is polled deterministically (round-robin through the multiplexer channels), so the worst-case cycle time is predictable.
- Basic filtering (median/EMA) is performed inline with strict upper bounds on compute time.

The sensing task outputs:

- **ToF distances** for safety motion constraints (stop/slow zones),
- **IMU measurements** (gyro/accel and orientation estimates if available),
- A status/health flag set (timeouts, out-of-range, stale sensor data).

Critically, **safety decisions are evaluated locally on the ESP32-S3** before publishing—so even if the ROS side is delayed, the microcontroller can still enforce safe behavior.

3.5.5.4 micro-ROS integration: executor-driven callbacks inside a controlled task

micro-ROS is integrated so that ROS communication never "takes over" the firmware timing. The typical pattern is:

- Create publishers (wheel odom/TF data, IMU, ToF, diagnostics)
- Create subscribers (e.g., `/cmd_vel` or equivalent)
- Run the **rclc executor** inside a dedicated FreeRTOS communication task

Inside that communication task:

- Subscriptions are processed via executor callbacks (command updates).
- Publishing is performed from timer-driven logic or periodic loops using prepared messages.

Two important implementation principles keep this non-blocking:

1. **Executor spin is time-sliced**
   Instead of blocking forever, the executor is advanced with a bounded time budget

(short spin period). This guarantees the communication task yields regularly and never starves the higher-priority control and safety tasks.

2. **Data handoff via queues/notifications, not shared blocking calls**
   Encoder and sensor tasks never wait on ROS. They push their latest data into lock-free or bounded structures:

- Single-producer/single-consumer queue
- Task notification + shared "latest sample" buffers guarded by lightweight synchronization (critical sections kept short)

The comm task pulls the newest available data and publishes it. If publishing is temporarily slower, the system can drop older samples (keeping the latest), which is the correct behavior for high-rate telemetry like IMU and ToF safety.

3.5.5.5 High-rate publishing for IMU and safety ToF without jitter

Your requirement ("callback is automated for publishing IMU frequently and ToF for safe movements") maps cleanly to **timer-based publish** in micro-ROS:

- IMU publish timer at a fixed rate (e.g., 50–200 Hz depending on bus limits and fusion needs)
- ToF publish timer at a fixed rate (bounded by multiplexer polling time)
- Wheel state/TF publish at a fixed rate (or triggered by encoder update windows)

The publish timers do not perform sensor reads. They publish the most recent stable measurements produced by the sensor/encoder tasks. This separation keeps publish timing consistent and prevents I²C latency from injecting jitter into control loops.

3.5.5.6 No blocking, no delay propagation: design guarantees

The firmware avoids "delay propagation" (one slow activity causing everything else to lag) through:

- Short ISRs + deferred processing
- Priority-based preemption (control/safety always wins)
- Bounded-time executor servicing
- Timeouts on all external buses (I²C/UART)
- Non-blocking inter-task communication (queues/notifications)
- "Latest-sample" publishing policy for high-rate topics (IMU/ToF), preventing backlog growth

As a result, wheel motion estimation remains accurate (encoder integrity preserved), ToF safety remains reactive (near-field stop logic stays local), and ROS integration remains clean (micro-ROS messages are published and subscribed reliably) without introducing blocking behavior that would compromise real-time control using freertos (Figure. 3.21).

Figure. 3.21 FREE - RTOS

3.5.5.7 ESP32-S3 ↔ Raspberry Pi 5 Communication via USB CDC (Serial over USB)

Communication between the low-level controller (ESP32-S3) and the high-level computer (Raspberry Pi 5) is a critical part of the ADR architecture, as it forms the bridge between real-time hardware control and ROS 2–based autonomy. In this system, the link is implemented using the **ESP32-S3 native USB interface configured as USB CDC (serial over USB)**. While this connection appears to the Linux host as a standard serial device, it is physically a USB differential link, offering improved noise immunity, stable throughput, and simplified wiring compared to raw TTL UART connections.

On the Raspberry Pi 5 side, ROS 2 Jazzy runs as the main middleware environment and hosts the **micro-ROS agent (**Figure. 3.22**)**. The agent acts as a bridge between the microcontroller and the ROS 2 DDS (Data Distribution Service) network. The ESP32-S3 runs the **micro-ROS client**, which allows it to participate in the ROS 2 ecosystem using the same publish/subscribe semantics as native ROS 2 nodes, despite running on a resource-constrained embedded platform. This architecture provides a **common communication model** between the Pi and the ESP32: both exchange data using ROS topics, services, and message definitions, rather than custom or ad-hoc serial protocols.

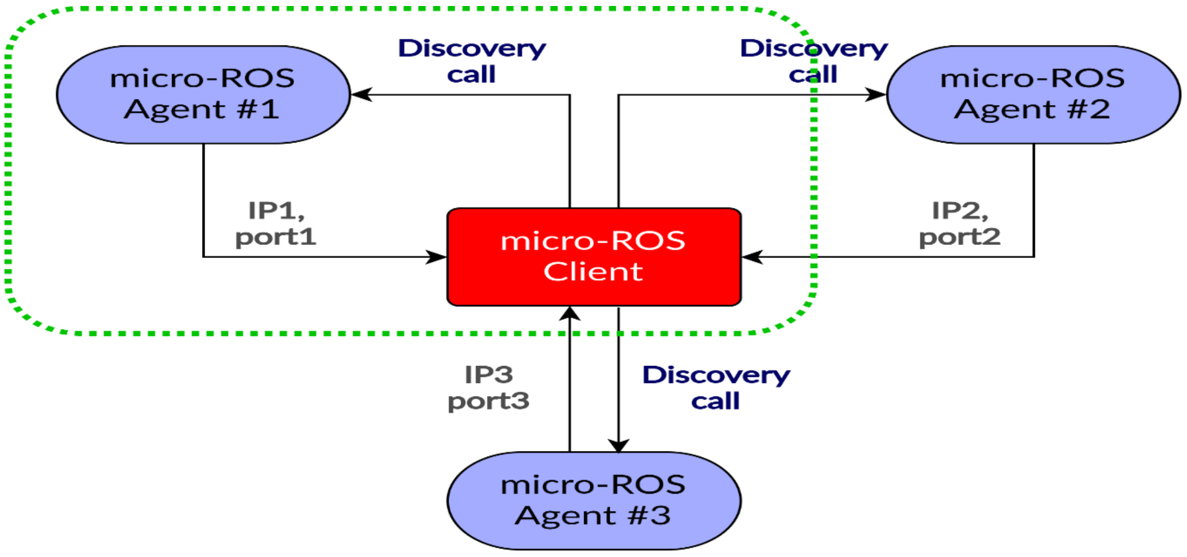


Figure. 3.22 micro-ros agent diagram

At the transport level, USB CDC provides a reliable and high-integrity channel for micro-ROS messages. At the middleware level, **DDS-based communication semantics** ensure structured, asynchronous data exchange with clearly defined Quality of Service (QoS) policies. Although the ESP32-S3 does not run DDS directly, the micro-ROS agent on the Raspberry Pi translates between the micro-ROS wire protocol and native DDS traffic. This allows messages published by the ESP32-S3 (such as wheel state, encoder-derived velocity, IMU data, and ToF safety distances) to appear as standard ROS 2 topics within the Jazzy environment, indistinguishable from data published by full ROS 2 nodes.

In the downstream direction, the Raspberry Pi 5 sends **motion commands** to the ESP32-S3 using ROS 2 topics (typically velocity-based commands). These commands encode *where to move* and *at what speed*, commonly expressed as linear and angular velocity setpoints. The micro-ROS client on the ESP32-S3 subscribes to these command topics, and the corresponding callbacks update internal control targets used by the motor control task. Importantly, the receipt of commands is decoupled from actuation timing: incoming messages update shared state, while the real-time control loop applies those setpoints at a fixed, deterministic rate. This prevents bursty communication or DDS scheduling effects from directly influencing motor timing.

In the upstream direction, the ESP32-S3 publishes real-time telemetry to ROS 2. This includes wheel motion data (used for odometry and wheel TFs), IMU measurements, ToF-based near-field safety distances, and diagnostic or status information. Publishing is handled periodically and deterministically within the microcontroller firmware, ensuring that the ROS 2 system receives fresh, time-consistent data without forcing the embedded controller into blocking or communication-driven execution. If communication is temporarily delayed or interrupted, the ESP32-S3 continues to operate safely using the last valid command or transitions to a predefined safe state, rather than stalling.

This USB CDC–based micro-ROS communication layer provides several architectural advantages. It establishes a **clean separation of responsibilities**: the Raspberry Pi 5 focuses on perception, planning, and global decision-making, while the ESP32-S3 enforces real-time control and safety. It eliminates custom binary protocols in favor of standardized ROS 2 message definitions, improving maintainability and debuggability. Finally, by relying on USB's electrical robustness and DDS-style asynchronous messaging, the system achieves reliable, scalable communication suitable for a mobile robot operating in an electrically noisy environment with tight real-time constraints.

Together, these design choices allow the ESP32-S3 and Raspberry Pi 5 to "speak the same language" within the ROS 2 Jazzy framework, enabling precise command delivery, high-rate telemetry exchange, and safe, coordinated autonomous motion.

3.5.6 Sensor and Peripheral Interfacing

The ESP32-S3 serves as the **primary peripheral controller** in the ADR, interfacing directly with all low-level sensors and feedback devices through a combination of digital communication buses and discrete GPIO signals. Each interface type (I²C, UART, and GPIO/interrupt inputs) is selected based on the **data rate, timing criticality, wiring complexity, and robustness requirements** of the connected peripheral. This layered peripheral design ensures accurate sensing, predictable latency, and reliable operation under dynamic motion and electrical noise conditions.

3.5.6.1 I²C Peripheral Interface (IMU and ToF Sensors)

The **Inter-Integrated Circuit (I²C)** bus is used for short-distance, low-to-moderate speed communication with onboard sensors that require synchronized sampling rather than continuous streaming. In the ADR, the I²C bus connects the **IMU** and the **Time-of-Flight (ToF) distance sensors** to the ESP32-S3.

I²C operates using two shared lines as shown in (Figure. 3.23):

- **SDA (Serial Data)**
- **SCL (Serial Clock)**

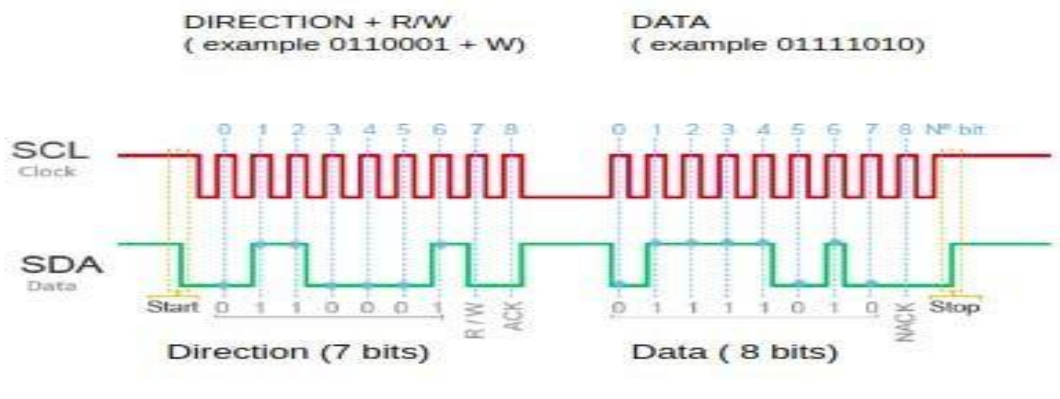


Figure. 3.23 *(I2C communication protocol*

Both lines are **open-drain**, meaning devices only pull the line low; pull-up resistors restore the high logic level. This architecture allows multiple devices to share the same bus safely.

The ESP32-S3 functions as the **bus master**, controlling clock timing and transaction sequencing.

Typical configuration parameters used in this system:

- **Bus speed:**
  - Standard Mode: 100 kHz
  - Fast Mode: 400 kHz (commonly used for IMU + ToF balance)
- **Bus length:** Short (on-robot PCB and harness only)
- **Voltage level:** 3.3 V logic

3.5.6.2 IMU over I²C

The IMU requires **periodic, time-consistent reads** of acceleration and angular velocity. I²C is well suited here because:

- Data packets are small and predictable.
- Sampling rate is controlled by the ESP32-S3.
- Latency is bounded and acceptable for attitude estimation.

Each IMU transaction consists of a register address phase followed by a burst read, ensuring minimal overhead per sample.

3.5.6.3 ToF Sensors and I²C Multiplexing

Multiple ToF sensors share the same default I²C address, which would normally cause bus contention. This is resolved using an **I²C multiplexer (TCA9548A)** controlled by the ESP32-S3. The multiplexer creates **isolated I²C branches**, allowing the controller to activate and read one ToF sensor at a time.

Key design advantages:

- No address conflicts.
- Predictable polling order.
- Known maximum sensor update cycle time.

ToF sensors are typically read at **20–50 Hz per sensor**, which is sufficient for near-field obstacle detection and emergency stopping logic. The maximum measurable distance (on the order of meters) and update rate are well matched to I²C bandwidth and FreeRTOS scheduling.

3.5.7 UART Peripheral Interface (GP-02 GPS)

The **Universal Asynchronous Receiver/Transmitter (UART)** interface is used for peripherals that produce a **continuous data stream** rather than discrete register-based reads. In the ADR, the GP-02 GPS module is connected via a dedicated UART as shown In (Figure. 3.24) channel on the ESP32-S3.

UART communication characteristics:

- **TX/RX point-to-point wiring**
- **No shared clock** (asynchronous)
- **Fixed baud rate**, commonly:
  - 9600 bps or 115200 bps for GPS modules
- **Logic level:** 3.3 V TTL

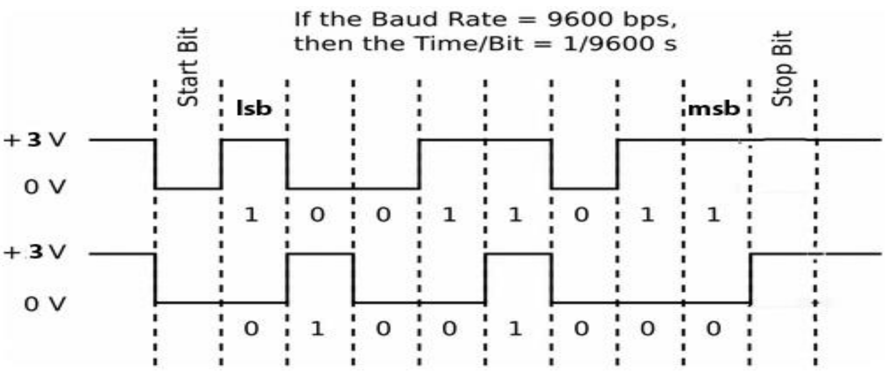


Figure. 3.24 UART PROTOCOL

GPS modules transmit **NMEA sentences** continuously, encoding latitude, longitude, speed, and fix quality. UART is ideal here because:

- Data arrives sequentially without polling.
- The ESP32-S3 can parse messages incrementally.
- Reception can be handled in a buffered, non-blocking task.

The UART receive pin is configured as a **normal digital input**, where logic-high and logic-low transitions represent serial bits. A hardware UART peripheral ensures precise timing and reliable decoding without CPU-intensive bit-banging.

3.5.8 GPIO and Interrupt Inputs (Wheel Encoders)

Wheel encoders provide **high-frequency, event-driven signals** that represent physical motion. These are connected to the ESP32-S3 using **digital GPIO pins configured as interrupt inputs**.

Encoder signal characteristics:

- **Digital pulses (HIGH/LOW transitions)**
- Frequency proportional to wheel speed
- Direction determined by phase relationship (for quadrature encoders)

The ESP32-S3 GPIOs detect rising or falling edges and trigger **hardware interrupts**. This mechanism ensures:

- Immediate response to motion.
- No polling delay.
- No loss of steps at high speed.

Logic-high and logic-low thresholds are defined by the 3.3 V GPIO standard, and encoder inputs are often conditioned (pull-ups, debouncing, or filtering) to ensure signal integrity. Distance traveled is computed from pulse counts, while velocity is derived from pulse frequency over time.

3.5.9 Signal Levels and Input Types

Across all peripherals, the ESP32-S3 uses **3.3 V logic**:

- **Normal digital inputs:**
  - Used for encoder pulses, limit signals, and status flags.
- **Peripheral inputs (I²C/UART):**
  - Managed by dedicated hardware blocks for timing accuracy and CPU efficiency.

This consistent logic level simplifies interfacing and reduces the risk of electrical mismatch.

3.5.10 Communication Speed, Distance, and Design Boundaries

Each interface is chosen according to its *speed and physical distance constraints*

Table 2:

Interface	Typical Speed	Distance	Use Case
I²C	100–400 kHz	Short (PCB / internal wiring)	IMU, ToF sensors
UART	9.6–115 kbps	Short–medium	GPS data stream
GPIO + Interrupts	Event-driven	Very short	Wheel encoders

Table 2 *speed and physical distance constraints*

By keeping I²C and GPIO wiring short and routing UART cleanly, the design minimizes noise susceptibility and timing uncertainty.

3.5.11 IoT and Telemetry: Location Reporting and Remote Monitoring

In addition to real-time control and onboard sensing, the ADR embedded system is designed to support **IoT-oriented telemetry with direct user awareness**, allowing a client or operator to know **exactly where the robot is located at any moment**. The ESP32-S3 plays a central role in this capability by directly interfacing with the GPS module, validating location data, and preparing it for transmission to higher-level systems or user-facing applications.

A key requirement of the smart delivery robot is **transparent location visibility for the end user**. The ESP32-S3 continuously receives positioning data from the GP-02 GPS module and maintains the latest valid estimate of the robot's geographic position (latitude, longitude, and speed). This information can then be forwarded—through the system's communication stack—to a user interface such as a mobile application, web dashboard, or monitoring station. As a result, a client waiting for a delivery can track the robot's progress in real time and know precisely when it approaches or reaches the destination.

3.5.11.1 GPS Positioning Principle and Satellite-Based Calculation

The GPS module determines position using **satellite trilateration**, where the receiver computes its location based on distance measurements from multiple satellites. Each GPS satellite broadcasts a time-stamped signal containing its precise orbital position. The GPS receiver measures the signal travel time and converts it into distance using the speed of light.

The basic distance (pseudo-range) to a satellite is given by:

$$\rho i = c \cdot (trecv - ttx, i)$$

where:

- ρi is the pseudo-range to satellite i,
- c is the speed of light,
- $trecv$ is the signal reception time,
- ttx, i is the signal transmission time from satellite i.

Using distance measurements from **at least four satellites**, the receiver solves for four unknowns: the three-dimensional position (x, y, z) and the receiver clock bias. The system of equations can be expressed as:

$$\rho i = (x - xi)2 + (y - yi)2 + (z - zi)2 + b$$

where (xi, yi, zi) is the known position of satellite i, and b is the receiver clock offset. Solving this system yields the receiver's position in Earth-centered coordinates, which are then converted into **latitude, longitude, and altitude**.

3.5.11.2 Location Reporting to the User

Once the ESP32-S3 receives valid GPS data, it performs basic validation (fix quality, number of satellites, and update freshness) before marking the location as reliable. The validated position is then used for **user-facing telemetry**, enabling features such as:

- Live position tracking on a map interface.
- Delivery progress visualization (distance remaining, arrival proximity).
- Status notifications when the robot reaches predefined waypoints or the delivery destination.

To support meaningful user feedback, distance to a target location can be computed using the **great-circle (Haversine) formula**:

$$d = 2Rarcsin(sin2(2\Delta\phi) + cos(\phi1)cos(\phi2)sin2(2\Delta\lambda))$$

where:

- ϕ is latitude,
- λ is longitude,
- R is Earth's radius,
- $\Delta\phi$ and $\Delta\lambda$ are the differences between current and target coordinates.

This calculation allows the system to determine how far the robot is from the user or delivery point, enabling accurate arrival estimation and clear communication to the client.

3.5.11.3 Telemetry Reliability and Integration

The ESP32-S3 ensures that location reporting is **non-intrusive to real-time control**. GPS acquisition and telemetry preparation run independently from motor and safety tasks, and outdated or invalid fixes are clearly flagged. If satellite coverage is temporarily lost, the system can indicate degraded accuracy rather than transmitting misleading information.

By combining satellite-based positioning, onboard validation, and structured telemetry, the ESP32-S3 enables the ADR to function as a **user-aware, connected delivery platform**. The robot not only navigates autonomously but also communicates its exact position to the user, reinforcing trust, transparency, and usability—essential characteristics of a real-world smart delivery system.

3.5.12 Reliability and Safety Firmware Features

Reliability and safety are fundamental requirements of the ADR embedded control system, as the robot operates with high-current motor drivers, moving mechanical parts, and continuous interaction with its environment. The ESP32-S3 firmware is therefore designed not only to execute control logic, but also to **continuously supervise system health, enforce safety constraints, and protect the carried cargo**. All safety mechanisms are implemented at the embedded level to guarantee fast and deterministic response times, independent of high-level software activity

3.5.12.1 Continuous ToF-Based Safety Monitoring Using FreeRTOS

A critical safety feature of the ADR is the **continuous operation of the Time-of-Flight (ToF) safety sensors**. Using the FreeRTOS multitasking environment, the ESP32-S3 runs a **dedicated high-priority ToF monitoring task** that executes continuously while the system is powered. This task polls all ToF sensors in a deterministic round-robin manner and evaluates distance measurements in real time.

Because this ToF task runs independently of communication, logging, or telemetry activities, **near-field obstacle detection is never paused or delayed**. Even under heavy system load, the RTOS scheduler ensures that the ToF safety task preempts lower-priority tasks if necessary. When an obstacle is detected within a predefined safety threshold, the firmware can immediately trigger protective actions such as speed reduction or a full stop. This design guarantees that safety sensing remains active at all times, providing a reliable last line of defense against collisions.

3.5.12.2 Watchdog Supervision and System Recovery

A core reliability mechanism in the embedded firmware is the use of a **watchdog timer**. The watchdog continuously monitors firmware execution and expects periodic refresh signals from healthy, time-critical tasks. If the firmware becomes unresponsive due to a software fault, deadlock, or unexpected blocking condition, the watchdog automatically resets the system, preventing the robot from remaining in an uncontrolled or unsafe state.

The watchdog refresh is explicitly tied to the successful execution of essential tasks such as motor control and safety monitoring. This ensures that the system is only considered healthy if critical real-time functions are actively running. Following a watchdog reset, the firmware enters a controlled startup sequence, initializing peripherals and holding the propulsion system in a disabled state until normal operating conditions are verified.

3.5.12.3 Fault Detection and Deterministic Handling

The ESP32-S3 firmware continuously evaluates multiple fault indicators to maintain safe operation, including:

- Loss or degradation of sensor data (e.g., missing encoder pulses, invalid or stale ToF readings, IMU timeouts).
- Abnormal system or power conditions (such as emergency stop triggers or unexpected shutdown signals).
- Control integrity issues, including loss of valid motion commands.

When a fault is detected, the firmware follows a **deterministic fault-handling path**. Depending on severity, this may involve limiting motion, inhibiting further commands, or transitioning the robot into a complete stop. Importantly, all fault responses are executed

locally on the ESP32-S3, without relying on responses from the Raspberry Pi or external systems.

For critical faults, the firmware commands the relay-based power control stage to physically disconnect the **24 V traction supply**, ensuring that motor power is removed at the hardware level. This mechanism remains effective even if higher-level software is delayed or unavailable.

3.5.12.4 Deterministic Control and Timing Guarantees

Deterministic execution is enforced by structuring all safety- and control-critical logic within **fixed-period FreeRTOS tasks**, combined with interrupt-driven mechanisms where appropriate (such as encoder pulse capture). Blocking delays, unbounded loops, and long-running operations are strictly avoided in these code paths.

Motor control updates, ToF sensing, IMU acquisition, and safety checks all execute at known and bounded rates. This allows worst-case reaction times to be calculated and validated, ensuring that emergency conditions—such as a sudden obstacle detected by the ToF sensors—are handled within a guaranteed response window.

3.5.12.5 Secure Cargo Lock Control and User Authorization

In addition to motion safety, the ADR incorporates **cargo security** as part of its embedded safety framework. The ESP32-S3 controls an electronic **cargo locking mechanism**, which remains locked by default during operation. The lock can only be actuated when the ESP32-S3 receives a **valid authorization command from the user** (for example, after delivery confirmation).

This logic is enforced entirely at the microcontroller level:

- The lock will **not open automatically** due to communication glitches or system resets.
- User authorization is explicitly checked before enabling the unlock action.
- If the system is in a fault or emergency state, the cargo lock remains secured.

By managing the lock directly within the real-time firmware, the system ensures that cargo access is controlled, intentional, and traceable, adding an important layer of trust and safety for delivery applications.

By combining continuous ToF-based safety sensing under FreeRTOS multitasking, watchdog supervision, deterministic fault handling, and secure cargo lock control, the ESP32-S3 firmware provides a comprehensive reliability and safety layer for the ADR. These mechanisms ensure that the robot maintains predictable behavior, protects both people and cargo, and reacts immediately to hazardous conditions—forming a trustworthy foundation for autonomous delivery operation.

3.6 ROS 2 integration

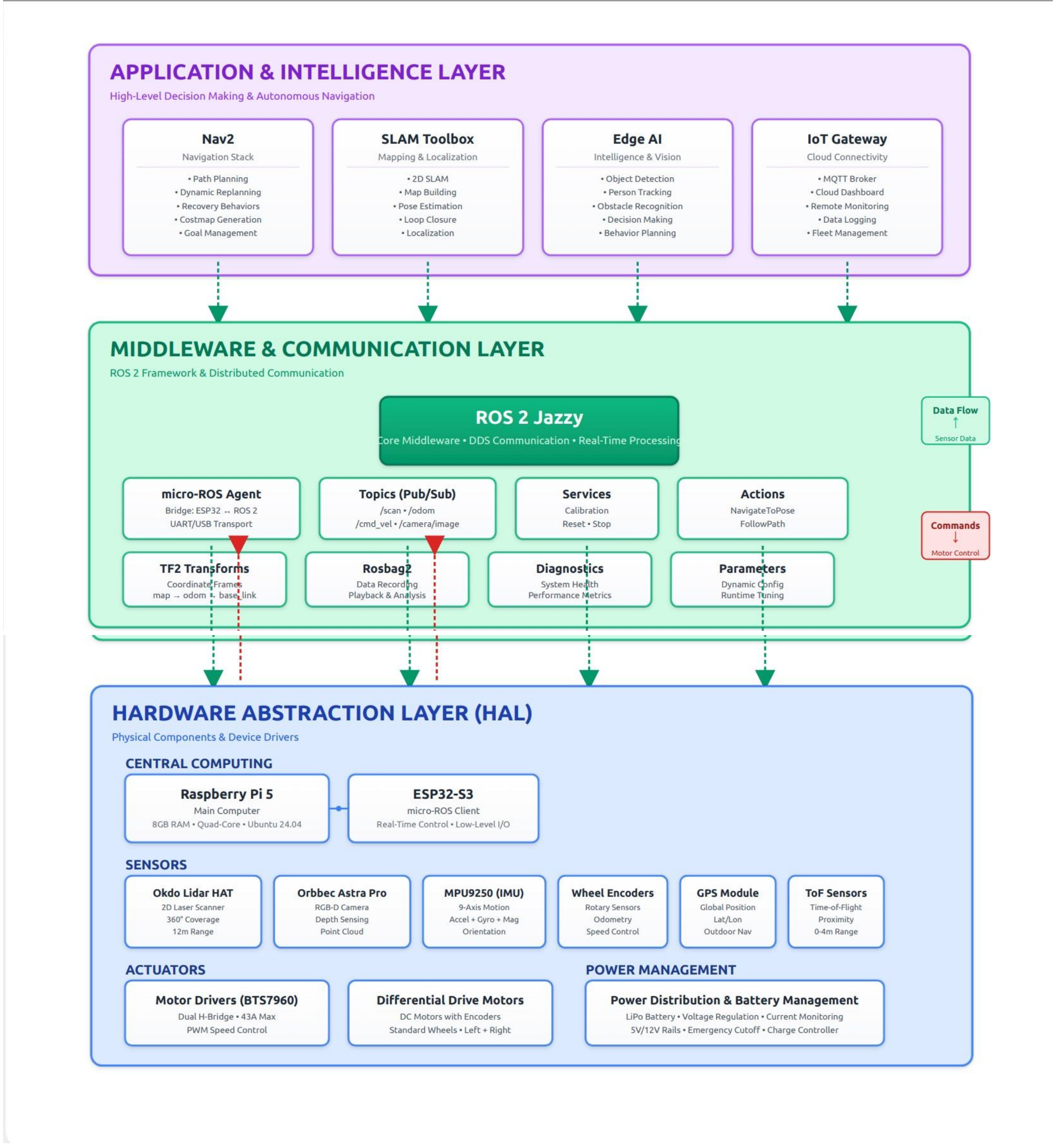


Figure. 3.25 ROS2 system architecture

3.6.1 ROS 2 as the System Integration Layer

The Robot Operating System 2 (ROS 2) is employed as the middleware layer that integrates all AMR subsystems into a unified distributed system. Each functional component is encapsulated as an independent node that exchanges data through typed interfaces.

Sensor drivers publish raw measurements, state estimation nodes compute pose and velocity, planning nodes generate motion references, and control interfaces transmit velocity commands to the embedded controller. Communication occurs via topics, services, and actions, enabling loose coupling and modular system design. ROS 2 nodes may execute on separate processors while remaining part of a single logical computation graph. ROS 2 was selected over custom communication stacks due to the following technical considerations:

- Native support for distributed, multi-process systems
- Deterministic communication via DDS Quality of Service (QoS) policies
- Standardized message definitions and tooling
- Compatibility with real-time execution and lifecycle management

These characteristics reduce integration complexity and improve system scalability without introducing application-specific middleware dependencies.

In this architecture, the MCU executes a lightweight ROS 2 client under a Real-Time Operating System (RTOS). A micro-ROS agent runs on the SBC and bridges the MCU into the ROS 2 network. Communication occurs over UART, USB, or UDP transports.

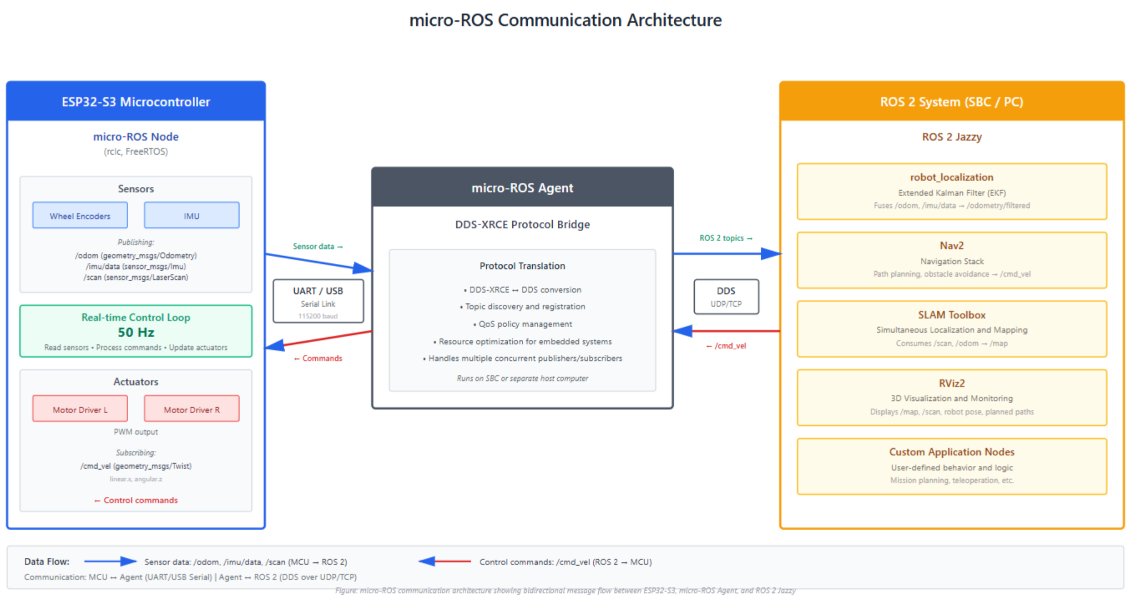

micro-ROS Communication Architecture
ESP32-S3 Microcontroller
micro-ROS Node
(rclc, FreeRTOS)
Sensors
Wheel Encoders
IMU
Publishing:
/odom (geometry_msgs/Odometry)
/imu/data (sensor_msgs/Imu)
/scan (sensor_msgs/LaserScan)
Real-time Control Loop
50 Hz
Read sensors • Process commands • Update actuators
Actuators
Motor Driver L
Motor Driver R
PWM output
Subscribing:
/cmd_vel (geometry_msgs/Twist)
linear.x, angular.z
← Control commands
Sensor data →
UART / USB
Serial Link
115200 baud
← Commands
micro-ROS Agent
DDS-XRCE Protocol Bridge
Protocol Translation
• DDS-XRCE ↔ DDS conversion
• Topic discovery and registration
• QoS policy management
• Resource optimization for embedded systems
• Handles multiple concurrent publishers/subscribers
Runs on SBC or separate host computer
ROS 2 topics →
DDS
UDP/TCP
← /cmd_vel
ROS 2 System (SBC / PC)
ROS 2 Jazzy
robot_localization
Extended Kalman Filter (EKF)
Fuses /odom, /imu/data → /odometry/filtered
Nav2
Navigation Stack
Path planning, obstacle avoidance → /cmd_vel
SLAM Toolbox
Simultaneous Localization and Mapping
Consumes /scan, /odom → /map
RViz2
3D Visualization and Monitoring
Displays /map, /scan, robot pose, planned paths
Custom Application Nodes
User-defined behavior and logic
Mission planning, teleoperation, etc.
Data Flow:
Sensor data: /odom, /imu/data, /scan (MCU → ROS 2)
Control commands: /cmd_vel (ROS 2 → MCU)
Communication: MCU ↔ Agent (UART/USB Serial) | Agent ↔ ROS 2 (DDS over UDP/TCP)
Figure: micro-ROS communication architecture showing bidirectional message flow between ESP32-S3, micro-ROS Agent, and ROS 2 Jazzy

Figure. 3.26 Micro-Ros communication architecture

3.6.2 Closed-Loop Control and System Operation

Autonomous behavior emerges from continuous feedback between perception, planning, and control subsystems.

When a navigation goal is issued, the planning stack computes a velocity reference (v, ω) and publishes it on the `/cmd_vel` topic. The MCU receives this command and regulates wheel velocities using a Proportional–Integral–Derivative (PID) controller.

3.6.2.1 Differential Drive Kinematics

For a differential drive robot with wheel radius r and axle length L:

$$v = \frac{r}{2}(\omega r + \omega l)$$

$$\omega = \frac{r}{L}(\omega r - \omega l)$$

3.6.2.2 PID Control Law

The control input u(t) applied to each motor is defined as:

$$u(t) = Kpe(t) + Ki\int^{e}(t)dt + Kd\ \cdot\ \frac{de(t)}{dt}$$

Where e(t) represents the velocity tracking error.

Safety mechanisms include watchdog timers, communication timeouts, and hardware emergency stop circuits. These layers ensure safe behavior under fault conditions.

3.6.3 ROS 2 Core Concepts for Autonomous Mobile Robots**

3.6.3.1 ROS 2 Jazzy Jalisco Overview

ROS 2 Jazzy Jalisco is a Long-Term Support (LTS) distribution designed for distributed, real-time robotic systems. It introduces deterministic communication, decentralized discovery, and native support for multi-machine deployments. Communication is implemented using the Data Distribution Service (DDS) standard, which provides peer-to-peer data exchange with configurable Quality of Service (QoS) guarantees.ROS 2 applications are composed of nodes, each implementing a single functional responsibility. Nodes exchange data over the ROS graph while remaining independent in execution and deployment. Namespaces are used to isolate node instances and prevent naming conflicts, enabling multi-robot operation using identical software stacks. Parameters are locally owned by nodes and strongly typed. This decentralized parameter model improves encapsulation and supports runtime reconfiguration. Rosbag2 enables recording and replaying system data for validation and debugging. Parameters allow dynamic tuning of controllers and planners without recompilation. Logging provides structured diagnostics with severity levels to support fault analysis.RViz2 provides real-time visualization of robot state, sensor data, and planning outputs. rqt aggregates system introspection tools into a unified graphical interface. ros2 doctor verifies runtime environment health and configuration consistency. Lifecycle nodes implement deterministic state transitions for controlled startup and shutdown. Executors define callback scheduling strategies, supporting both parallel and deterministic execution models.

Combined with QoS tuning and careful memory management, ROS 2 supports real-time AMR operation when deployed on suitable hardware and operating systems.

3.6.4 Mechanical-to-Software Transition: From CAD to URDF

This chapter presents a systematic workflow for converting a mechanical robot design into a ROS 2–compatible digital twin as shown Figure. 3.27 CAD Model To ROS2 Transformation. The process spans mechanical model preparation, extraction of physical properties, generation of a modular URDF/Xacro description, and verification of the resulting model in RViz2. The objective is to ensure geometric, kinematic, and physical consistency between the physical AMR and its software representation, thereby enabling reliable simulation, visualization, and downstream algorithm development.

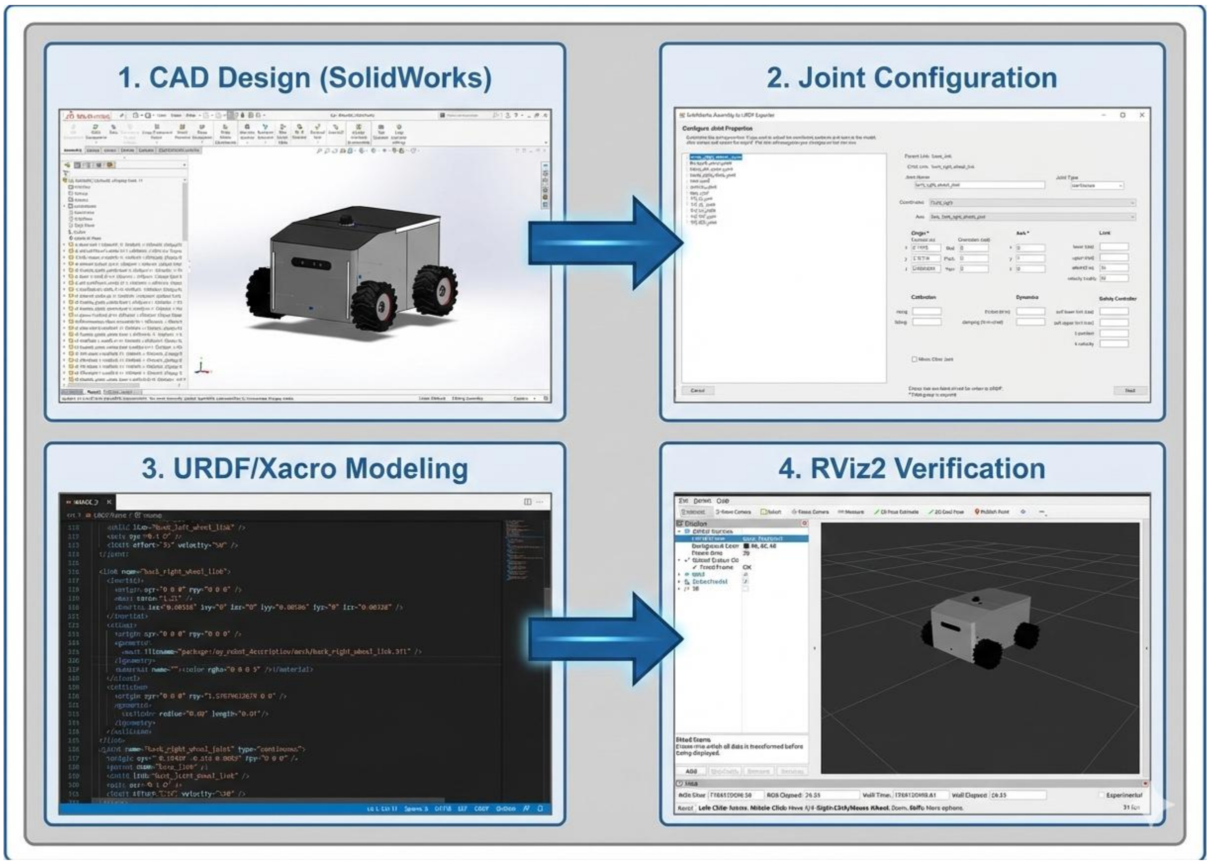


Figure. 3.27 CAD Model To ROS2 Transformation

3.6.4.1 Kinematic Decomposition and Link Hierarchy

The CAD assembly is decomposed into rigid bodies that directly correspond to URDF links. Components that exhibit no relative motion are grouped into a single subassembly, while mechanically independent components are separated and connected via explicit joints. This decomposition produces a kinematic tree rooted at the base link, typically representing the robot chassis.

Each joint is associated with a dedicated reference coordinate system defined in SolidWorks. These coordinate frames determine the relative transformation between parent and child links after export. The joint axis is aligned with the physical axis of rotation or translation, and ROS coordinate conventions (X-forward, Z-up) are enforced at the CAD level to prevent corrective transformations in the URDF. This disciplined structuring ensures a one-to-one mapping between mechanical subassemblies and URDF links, minimizing ambiguity during digital model generation.

3.6.4.2 Mass Properties and Inertia Extraction

Accurate physical modeling requires realistic mass and inertia parameters for each link. In SolidWorks, materials are assigned to all components prior to exporting mass properties. For each subassembly, the total mass, center of mass, and inertia tensor are computed using the built-in Mass Properties tools.

The inertia tensor for a rigid link is defined as:

$$I = \quad IxxIyxIzxIxyIyyIzyIxzIyzIzz$$

These parameters are transferred into the URDF <inertial> tag. While automated extraction via CAD-to-ROS tools provides reliable estimates, the resulting values are validated by verifying total robot mass and checking symmetry conditions where mechanically expected. Accurate inertial modeling is essential for stable physics simulation and controller performance.

3.6.4.3 Mechanical Validation of Sensor Placement

Sensor integration is evaluated at the CAD stage to ensure that the digital representation reflects realistic operational conditions. Sensor placement is optimized according to functional and mechanical constraints:

- LiDAR sensors are positioned to provide unobstructed 360° or forward-facing fields of view.
- Cameras are oriented to align optical axes with the robot's forward direction.
- IMUs are placed near the robot's center of mass to reduce rotational coupling.

Mechanical constraints such as vibration isolation, cable routing, accessibility, and collision clearance are verified using CAD visualization. Early resolution of these constraints ensures that sensor frames defined in the URDF correspond accurately to the physical robot configuration.

3.6.4.4 Digital Robot Model Generation

Each link's geometry is exported from SolidWorks as an STL or Collada (DAE) mesh. To ensure compatibility with ROS 2, reference coordinate systems aligned with ROS conventions are used during export. SolidWorks' native Y-up convention is avoided by exporting relative to custom coordinate frames.

URDF assumes SI units, with all dimensions expressed in meters. Export scaling is therefore verified to prevent unit mismatches. Mesh resolution is selected based on usage Meshes are organized within a dedicated meshes/ directory to maintain a clean robot description package structure.

3.6.4.5 URDF Generation and Xacro Refactoring

The SolidWorks-to-URDF (SW2URDF) exporter is used to generate an initial URDF model directly from the CAD assembly. Link definitions, joint types, transformations, and inertial parameters are derived automatically, ensuring geometric and physical consistency. The generated URDF is subsequently refactored into Xacro format to improve modularity and maintainability. Xacro macros are used to parameterize repeated structures, such as wheels or sensors, and to centralize constants such as dimensions and masses. This approach reduces redundancy and simplifies future mechanical revisions. Each URDF link is defined using three complementary components:

- **Inertial:** mass, center of mass, and inertia tensor
- **Visual:** mesh geometry and appearance for visualization
- **Collision:** simplified geometry for physics and collision detection

Consistent naming conventions (e.g.$, base\_link, wheel\_left\_link$) are applied to improve readability and facilitate debugging. Collision geometries are intentionally simplified to reduce computational overhead without compromising physical validity. Joints define the kinematic relationships between links as shown in Table 3 ROS2 Joints . The joint type is selected according to mechanical behavior:

Joint Type	Application
Fixed	Structural components and sensors
Continuous	Differential drive wheels
Revolute	Limited rotational joints
Prismatic	Linear actuators

Table 3 ROS2 Joints

Each joint specifies a parent link, child link, axis of motion, and optional limits on velocity and effort. Correct joint modeling ensures accurate kinematic behavior and prevents artificial constraints during simulation. Although visual materials do not affect physics, they enhance model interpretability in RViz2. Named materials with RGBA color definitions are used consistently across links. Transparency is selectively applied to sensors or internal components to reduce visual clutter. Where simulation is required, Gazebo-specific visual extensions may be added without affecting the core URDF structure.

3.6.4.6 Digital Twin Verification in RViz2

The `robot_state_publisher` node loads the URDF/Xacro description and publishes the robot's transform tree. When combined with `joint_state_publisher`, both fixed and movable joints can be visualized and manipulated. Successful initialization confirms syntactic correctness and complete link–joint connectivity.

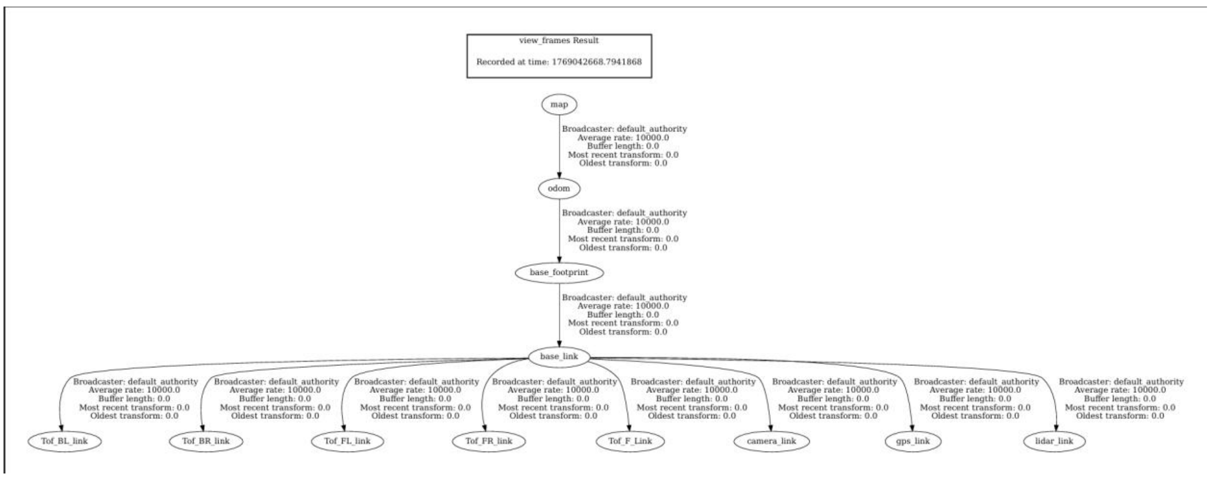


Figure. 3.28 ROS2 Publishers

3.6.4.7 Kinematic and Frame Validation

The digital twin is validated in RViz2 by inspecting:

- Link alignment and relative placement
- Joint axes and motion direction
- Sensor frame orientation and offsets

Interactive joint manipulation verifies kinematic correctness, while TF visualization tools confirm a single, coherent transform tree rooted at `base_link`. Any discrepancies identified during this stage are resolved by refining either the CAD export or the URDF/Xacro definitions.

3.6.5 Embedded Systems Architecture and Communication Bridge**

This chapter describes the embedded computing architecture of the Autonomous Mobile Robot (AMR) and the communication bridge that connects low-level real-time control with the high-level ROS 2 software ecosystem. A dual-layer architecture is adopted, consisting of a Single Board Computer (SBC) for autonomy and an embedded microcontroller for deterministic control. The chapter details hardware selection, the micro-ROS client–agent framework, real-time task scheduling, ROS 2 workspace organization, and robust device communication mechanisms.Autonomous mobile robots require heterogeneous computing resources to satisfy conflicting constraints, including computational throughput, real-time determinism, power efficiency, and physical footprint. To address these requirements, a two-tier architecture is employed:

- **High-level layer:** executes perception, localization, mapping, and navigation.
- **Low-level layer:** executes motor control, encoder processing, and time-critical sensing.

This separation isolates real-time constraints from computationally intensive algorithms, improving system robustness and predictability.

3.6.5.1 Primary Onboard Computer (Raspberry Pi 5)

The Raspberry Pi 5 is selected as the primary onboard computer due to its improved processing performance, I/O bandwidth, and ecosystem support. Its quad-core 64-bit Arm Cortex-A76 processor operating at 2.4 GHz provides sufficient computational capacity for ROS 2, the Nav2 navigation stack, SLAM algorithms, and moderate perception workloads.

Enhanced I/O capabilities, including dual MIPI CSI interfaces, PCIe 2.0 connectivity, and Gigabit Ethernet, support high-bandwidth sensors and storage expansion. Typical power consumption under moderate load remains within 3–5 W, making the platform suitable for battery-powered robotic operation.

Criterion	Raspberry Pi 5	Industrial SBC
ROS 2 support	Native	Native
Cost	Low	High
ower consumption	Low	Moderate–High
eal-time capability	Limited	Better
Ecosystem	Extensive	Limited

Table 4 Raspberry Pi 5 vs **Industrial SBC**

The Raspberry Pi 5 was selected due to its favorable balance between performance, cost, and software ecosystem support.

3.6.5.2 Real-Time Controller (ESP32-S3)

Low-level real-time control is implemented using an ESP32-S3 microcontroller operating as a micro-ROS client. The dual-core Xtensa processor at 240 MHz, combined with FreeRTOS, enables deterministic scheduling of time-critical tasks such as motor control, encoder counting, and high-rate sensor acquisition. Control loops execute at frequencies of 500–1000 Hz, while velocity commands from the SBC are received at lower rates (typically 30–50 Hz). This decoupling ensures smooth actuation even under variable computational load on the SBC.

3.6.5.3 Serial Transport Configuration

To avoid bandwidth limitations, the serial link operates at a baud rate of 921600 bps. This configuration supports high-frequency telemetry, including encoder updates and IMU measurements, without introducing significant latency. Reliable high-speed communication requires careful buffer and task management. On the microcontroller, UART FIFOs and RTOS-managed queues absorb burst traffic. On the SBC, the Linux serial driver and micro-ROS Agent process incoming frames asynchronously. Data integrity is ensured through XRCE-DDS cyclic redundancy checks (CRC) and optional reliable communication modes with retransmission. Task prioritization and non-blocking I/O prevent buffer overflows and ensure stable system operation. A modular ROS 2 workspace structure is adopted to improve maintainability and scalability. Each functional subsystem is encapsulated within a dedicated package, interacting with others exclusively through ROS 2 interfaces. This decomposition enables independent development, testing, and reuse across simulation and real-hardware deployments.

3.6.5.4 Real-Time Task Scheduling

Deterministic behavior is achieved by confining real-time tasks to the microcontroller. FreeRTOS schedules motor control, sensor acquisition, and micro-ROS communication using priority-based preemption. Encoder counting relies on hardware interrupts and timers to prevent missed events. The SBC executes only non-real-time tasks, such as planning and perception, ensuring predictable actuation despite operating on a general-purpose operating system. Persistent device naming is implemented using udev rules on the SBC. USB devices are identified by vendor and product identifiers and assigned stable symbolic links (e.g., /dev/ttyMicroROS). This eliminates ambiguity due to dynamic enumeration and simplifies deployment and maintenance.

3.6.6 Perception Layer and Sensor Integration

The perception layer provides environmental awareness through heterogeneous sensors, including LiDAR, depth vision, inertial measurement, wheel encoders, and GPS. Each sensor contributes complementary information that is fused to support localization, mapping, obstacle detection, and navigation.

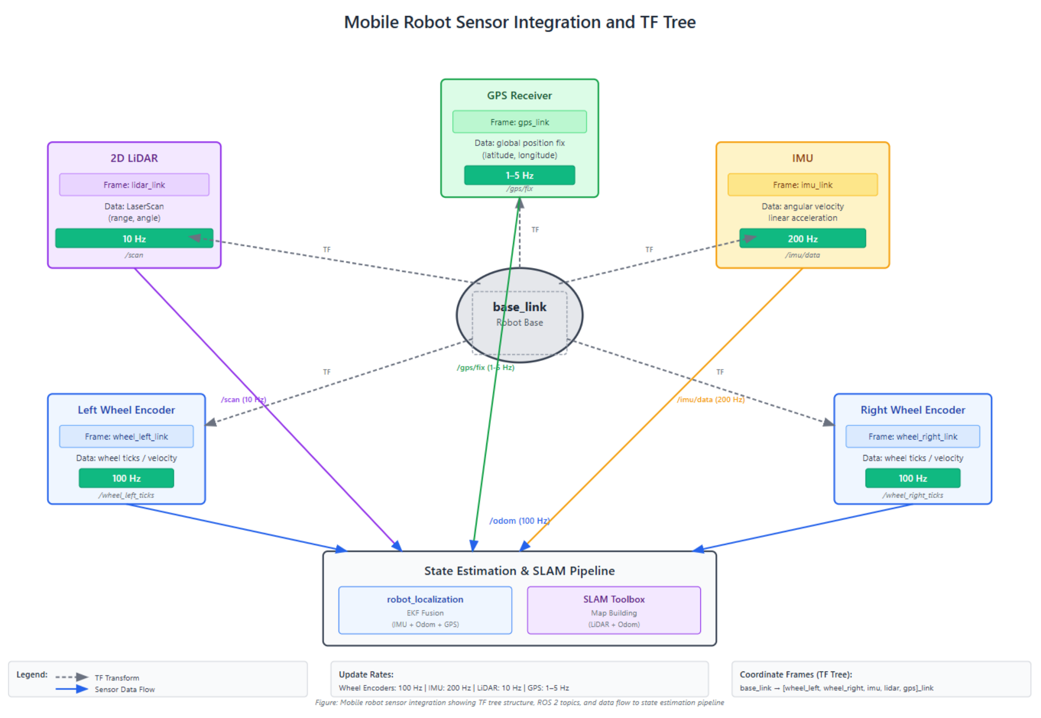

Mobile Robot Sensor Integration and TF Tree
GPS Receiver
Frame: gps_link
Data: global position fix (latitude, longitude)
1–5 Hz
/gps/fix
2D LiDAR
Frame: lidar_link
Data: LaserScan (range, angle)
10 Hz
/scan
IMU
Frame: imu_link
Data: angular velocity linear acceleration
200 Hz
/imu/data
TF
TF
TF
TF
TF
base_link
Robot Base
/gps/fix (1–5 Hz)
/scan (10 Hz)
/imu/data (200 Hz)
Left Wheel Encoder
Frame: wheel_left_link
Data: wheel ticks / velocity
100 Hz
/wheel_left_ticks
Right Wheel Encoder
Frame: wheel_right_link
Data: wheel ticks / velocity
100 Hz
/wheel_right_ticks
/odom (100 Hz)
State Estimation & SLAM Pipeline
robot_localization
EKF Fusion
(IMU + Odom + GPS)
SLAM Toolbox
Map Building
(LiDAR + Odom)
Legend:
TF Transform
Sensor Data Flow
Update Rates:
Wheel Encoders: 100 Hz | IMU: 200 Hz | LiDAR: 10 Hz | GPS: 1–5 Hz
Coordinate Frames (TF Tree):
base_link → [wheel_left, wheel_right, imu, lidar, gps]_link
Figure: Mobile robot sensor integration showing TF tree structure, ROS 2 topics, and data flow to state estimation pipeline

Figure. 3.29 ROS2 Perception Layer

A 2D LiDAR sensor provides planar range measurements for obstacle detection and mapping. A ROS 2 driver node parses raw sensor output and publishes `sensor_msgs/LaserScan` messages on the `/scan` topic. Each scan message includes angular bounds, angular resolution, and an array of range measurements in meters. The message is timestamped and associated with a dedicated sensor frame in the TF tree. Publishing frequency is matched to the sensor's rotational speed, typically 5–10 Hz.

Invalid measurements outside the sensor's operational range are discarded. Median or moving-average filters suppress spurious spikes caused by reflections or environmental noise. Blind sectors caused by mechanical obstructions are explicitly marked as invalid to prevent false obstacle detection.

A structured-light depth camera provides dense 3D information in the form of depth images and point clouds.

The ROS 2 driver publishes depth and color image streams, along with camera calibration data. When enabled, depth frames are converted into point clouds expressed in the camera optical frame. Static transforms align the camera with the robot base.

Depth values are projected into Cartesian coordinates using camera intrinsics. The resulting point clouds may be downsampled using voxel grid filtering and cropped to the robot's operating region. Depth measurements extend perception beyond planar LiDAR, enabling improved obstacle discrimination.

An IMU provides angular velocity, linear acceleration, and magnetic field measurements. Sensor fusion algorithms combine these signals to estimate orientation.

3.6.6.1 Sensor Fusion Model

An Extended Kalman Filter (EKF) or complementary filter fuses IMU signals to produce orientation in quaternion form:

$$ximu = [qw, qx, qy, qz, \omega x, \omega y, \omega z]T$$

Calibration removes bias and scale errors, while filtering suppresses vibration-induced noise.

3.6.6.2 Wheel Encoders and Odometry

Wheel encoders provide incremental motion estimates. For a wheel with circumference C and N counts per revolution, incremental travel per tick is:

$$\Delta s = \frac{C}{N}$$

For a differential drive robot with wheel separation W:

$$\Delta\theta = \frac{\Delta sr - \Delta sl}{W}$$

$$\Delta x = \frac{\Delta sr\ +\ \Delta sl}{2}\cos(\theta)$$

$$\Delta y = \frac{\Delta sr\ +\ \Delta sl}{2}\sin(\theta)$$

Interrupt-based counting and digital filtering ensure accurate odometry suitable for state estimation.

GPS provides absolute positioning in outdoor environments. NMEA sentences are parsed and published as `sensor_msgs/NavSatFix` messages.

3.6.6.3 Coordinate Transformation and Fusion

Geodetic coordinates are transformed into a local Cartesian frame (ENU or UTM). Covariance tuning reflects measurement uncertainty, allowing EKF-based fusion with wheel odometry and IMU data. Static transforms account for antenna offsets relative to the robot base.

All sensor messages are timestamped using a common clock and associated with consistent TF frames. Accurate temporal alignment enables correct data association during fusion and SLAM.

3.6.7 State Estimation and Localization

Wheel odometry estimates robot motion by integrating incremental wheel displacements over time, a process commonly referred to as dead reckoning. For a differential drive robot, odometry provides short-term, high-rate relative pose updates but lacks an absolute reference, resulting in unbounded drift.

Let ΔsL and ΔsR denote the incremental linear displacements of the left and right wheels, respectively, and let Wrepresent the wheelbase. The incremental robot motion is given by:

$$\Delta s = \frac{\Delta sR\ +\ \Delta sL}{2}$$

$$\Delta \theta = \frac{\Delta sR\ -\ \Delta sL}{W}$$

Assuming planar motion, the pose update in the odometry frame is:

$$\Delta x = \Delta s \cos\left(\theta + \frac{\Delta\theta}{2}\right),$$

$$\Delta y = \Delta s \sin\left(\theta + \frac{\Delta\theta}{2}\right)$$

The cumulative robot pose (x,y,θ) is updated by integrating these increments over time and published as a transform from `odom` to `base_link`.

3.6.7.1 Odometry Error Sources

Odometry errors grow monotonically due to both systematic and stochastic effects Even small heading errors cause significant lateral drift over distance. Consequently, wheel odometry is unsuitable as a standalone localization solution and must be corrected using absolute or drift-limiting sensors. To bound odometry drift and improve pose accuracy, sensor fusion is performed using an Extended Kalman Filter (EKF). The EKF estimates the robot state by combining proprioceptive and exteroceptive measurements within a probabilistic framework.

3.6.7.2 State Vector and Process Model

The estimated state vector is defined as:

$$x = [xy\theta v\omega]T$$

Where (x,y,θ) denote planar pose, and (v,ω) represent linear and angular velocities.

The nonlinear motion model is:

$$xk = f(xk - 1, uk) + wk$$

Where uk is derived from wheel odometry and $wk \sim N(0, Q)$ represents process noise.

3.6.7.3 EKF Prediction and Update Equations

Prediction:

$$x^k \mid k-1 = f(x^k - 1 \mid k-1, uk)$$

$$Pk \mid k-1 = Fk$$

Measurement Update:

$$yk = zk - h(x^k \mid k-1)$$

$$Kk = Pk \mid k - 1HkT(HkPk \mid k - 1HkT + R) - 1$$

$$x^k \mid k = x^k \mid k - 1 + Kkyk$$

$$Pk \mid k = (I - KkHk)Pk \mid k - 1$$

Here, Hk is the measurement Jacobian, and R is the measurement noise covariance.

3.6.7.4 6.3 Sensor Fusion Architecture and ROS 2 Integration

Localization is implemented using the `robot_localization` package in ROS 2. The EKF node fuses multiple sensor inputs into a single, filtered state estimate.

Subscribed topics typically include:

- `/odom` (nav_msgs/Odometry)
- `/imu/data` (sensor_msgs/Imu)
- `/gps/fix` (sensor_msgs/NavSatFix)

The EKF publishes:

- `/odometry/filtered`
- TF transform: `odom → base_link`

This configuration ensures smooth, locally accurate motion estimates while bounding long-term drift via absolute corrections. Filter performance depends critically on appropriate noise covariance selection. Static tuning is insufficient in dynamic environments; therefore, adaptive strategies are employed. Innovation residuals are monitored to detect inconsistency and prevent filter divergence. ROS 2 parameters allow runtime tuning without restarting nodes.

3.6.7.5 Uncertainty Representation and Drift Mitigation

The EKF explicitly maintains a covariance matrix P, representing pose uncertainty. This uncertainty grows during dead reckoning and contracts when reliable external measurements are incorporated.

Drift Mitigation Techniques

- GPS-based absolute corrections
- Zero-velocity updates (ZUPTs)
- Loop closures from SLAM
- Global re-localization triggers

Localization confidence directly informs planning and safety behavior. Elevated uncertainty may result in conservative velocity limits or reinitialization requests, improving overall system robustness.

3.6.8 Simultaneous Localization and Mapping (SLAM)

SLAM addresses the problem of estimating a robot trajectory $x1{:}k$ while simultaneously constructing a map M of an unknown environment using noisy sensor measurements.

$$p(x1{:}k, M \mid z1{:}k, u1{:}k)$$

Modern SLAM systems decompose this problem into:

- **Frontend:** Sensor processing and scan matching
- **Backend:** Pose graph optimization and loop closure

The selected SLAM approach is **2D LiDAR-based graph SLAM** due to:

- Structured indoor environment
- Reliable planar LiDAR data
- Moderate computational resources
- Direct compatibility with Nav2

SLAM Toolbox is selected over alternatives such as GMapping and Cartographer due to its native ROS 2 support, lifecycle management, and lifelong mapping capabilities.

3.6.8.1 Occupancy Grid Map Construction

The environment is represented as a 2D occupancy grid:

$$cell \in -1,0,100$$

Sensor rays are projected into the map frame, marking free space along beams and occupied cells at endpoints. Ray tracing and probabilistic updates reinforce confidence through repeated observations.

3.6.8.2 Static Map Storage and Reuse

Maps are saved using `map_server` as `.pgm` and `.yaml` files. During navigation, the static map provides a global reference, while dynamic obstacles are handled by local costmaps.

This separation reduces computational load and ensures repeatable localization performance across sessions.

3.6.8.3 Semantic Mapping and High-Level Representation

Semantic mapping augments metric maps with labeled entities, enabling higher-level reasoning.

Semantic Layers

- Object-level annotations (e.g., furniture)
- Region segmentation (rooms, zones)
- Topological graphs
- Ontology-based knowledge models

Benefits

- Human-readable navigation
- Task-aware planning
- Improved robustness in dynamic environments

Challenges

- Real-time object detection latency
- Consistency across sessions
- Computational overhead

Semantic SLAM represents an extension rather than a replacement of metric SLAM, enabling richer autonomy without sacrificing navigation reliability.

3.6.9 Autonomous Navigation Using the ROS 2 Nav2 Stack

Autonomous navigation is implemented using a hierarchical planning architecture consisting of **global path planning** and **local trajectory control**. The global planner computes an optimal path through a static or slowly changing environment, while the local controller generates dynamically feasible velocity commands that account for real-time obstacles and robot kinematics.

In ROS 2 Nav2 (Jazzy release), this separation is enforced through modular planner and controller plugins operating concurrently within a behavior tree–based execution framework.

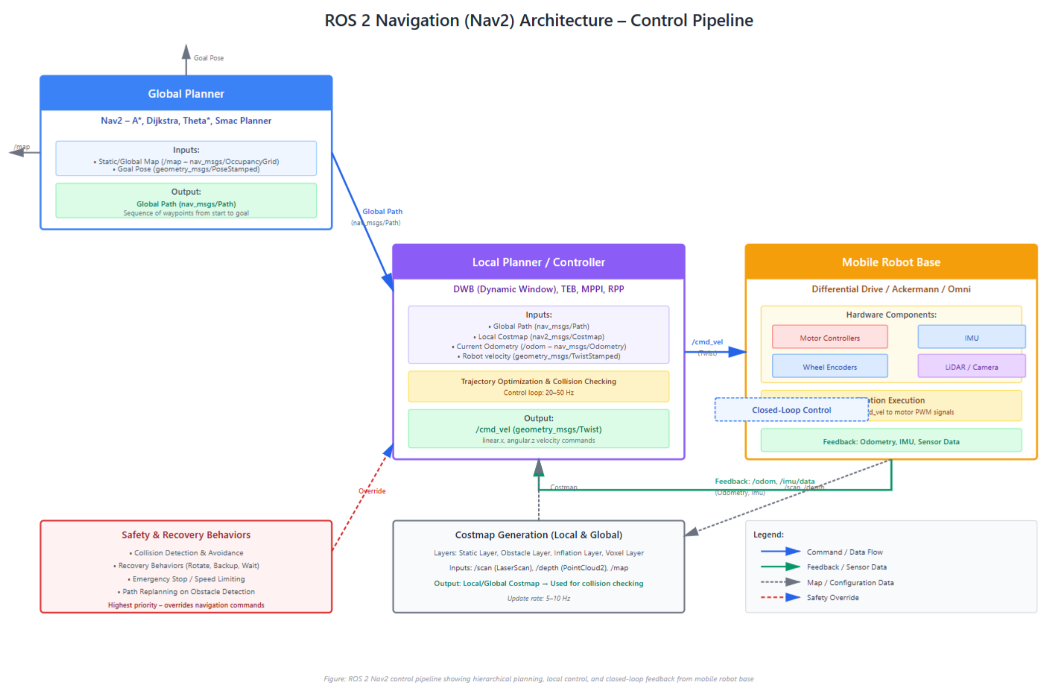

ROS 2 Navigation (Nav2) Architecture – Control Pipeline
Goal Pose
/map
Global Planner
Nav2 – A*, Dijkstra, Theta*, Smac Planner
Inputs:
• Static/Global Map (/map – nav_msgs/OccupancyGrid)
• Goal Pose (geometry_msgs/PoseStamped)
Output:
Global Path (nav_msgs/Path)
Sequence of waypoints from start to goal
Global Path
(nav_msgs/Path)
Local Planner / Controller
DWB (Dynamic Window), TEB, MPPI, RPP
Inputs:
• Global Path (nav_msgs/Path)
• Local Costmap (nav2_msgs/Costmap)
• Current Odometry (/odom – nav_msgs/Odometry)
• Robot velocity (geometry_msgs/TwistStamped)
Trajectory Optimization & Collision Checking
Control loop: 20–50 Hz
Output:
/cmd_vel (geometry_msgs/Twist)
linear.x, angular.z velocity commands
/cmd_vel
(Twist)
Mobile Robot Base
Differential Drive / Ackermann / Omni
Hardware Components:
Motor Controllers
IMU
Wheel Encoders
LiDAR / Camera
Closed-Loop Control
tion Execution
d_vel to motor PWM signals
Feedback: Odometry, IMU, Sensor Data
Feedback: /odom, /imu/data
(Odometry, Imu)
/scan, /depth
Costmap
Override
Safety & Recovery Behaviors
• Collision Detection & Avoidance
• Recovery Behaviors (Rotate, Backup, Wait)
• Emergency Stop / Speed Limiting
• Path Replanning on Obstacle Detection
Highest priority – overrides navigation commands
Costmap Generation (Local & Global)
Layers: Static Layer, Obstacle Layer, Inflation Layer, Voxel Layer
Inputs: /scan (LaserScan), /depth (PointCloud2), /map
Output: Local/Global Costmap → Used for collision checking
Update rate: 5–10 Hz
Legend:
Command / Data Flow
Feedback / Sensor Data
Map / Configuration Data
Safety Override
Figure: ROS 2 Nav2 control pipeline showing hierarchical planning, local control, and closed-loop feedback from mobile robot base

Figure. 3.30 ROS 2 Nav2 Stack

3.6.9.1 Grid-Based Search Formulation

The global planning problem is formulated as a shortest-path search on a discretized costmap. Each grid cell ci is assigned a traversal cost derived from obstacle proximity and inflation layers.

The objective is to minimize the cumulative cost:

$$J = i = 1 \sum^{N} (di + \lambda \cdot \cos t(ci))$$

where:

- di: is the Euclidean distance between successive cells,
- λ: weights obstacle proximity penalties.

3.6.9.2 Dynamic Window–Based Control

Local planning is performed using the **Dynamic Window–Based (DWB)** controller. Velocity commands are sampled within admissible bounds:

$$v \in [vmin, vmax],$$

$$\omega \in [\omega min, \omega max]$$

Each candidate trajectory is forward-simulated over a short horizon and evaluated using a weighted cost function:

$$J = wpJpath + woJobstacle + wgJgoal + wvJvelocity$$

The command minimizing J is selected and published to `/cmd_vel`.

3.6.9.3 Path Smoothing and Feasibility Optimization

Grid-based global paths often contain sharp turns unsuitable for nonholonomic robots. Nav2 applies constrained smoothing to reduce curvature and waypoint density.

- Optimization objectives include:
- Minimizing total path length
- Penalizing curvature

Preserving obstacle clearance

This produces smoother velocity profiles, reduced control effort, and improved actuator longevity.

3.6.9.4 Dynamic Obstacle Handling

Dynamic obstacles are addressed through continuous local replanning. If no valid trajectory exists, a global replan is triggered.

Velocity scaling and temporary halts are applied based on predicted collision time.

3.6.9.5 Latency Reduction

Latency is minimized through:

- Multi-threaded executors
- Intra-process communication
- Reduced costmap dimensions
- Frequency harmonization

3.7 AI Perception System

AI perception (shown in Figure. 3.31) in robotics refers to the process of interpreting raw sensor data to form an understanding of the environment. In an outdoor delivery scenario, the perception system enables the robot to detect obstacles (pedestrians, vehicles, curbs), recognize key objects (delivery targets, traffic signs), and navigate safely. It must handle complex outdoor constraints such as varying illumination (bright sunlight, shadows), weather conditions (rain, dust), and dynamic elements (moving people, animals). Real-time performance is essential: perception algorithms must process sensor inputs quickly to support immediate navigation decisions. At the same time, the system typically runs on resource-constrained hardware, requiring lightweight models and efficient processing. Overall, perception is a critical layer that bridges raw data (camera images, depth measurements) and higher-level tasks (path planning, control).

- **Role in outdoor delivery:** Identifying safe paths, detecting obstacles and delivery targets, and adapting to changing environments.
- **Key constraints:** Real-time operation, limited compute (e.g. embedded CPU/GPU), noisy and incomplete data (e.g. depth sensor noise in sunlit areas), and the need for robustness to weather and lighting changes.
- **Definition of perception:** Converting sensor streams into semantic information (object classes, locations, distances) that other modules (navigation, control) can use.

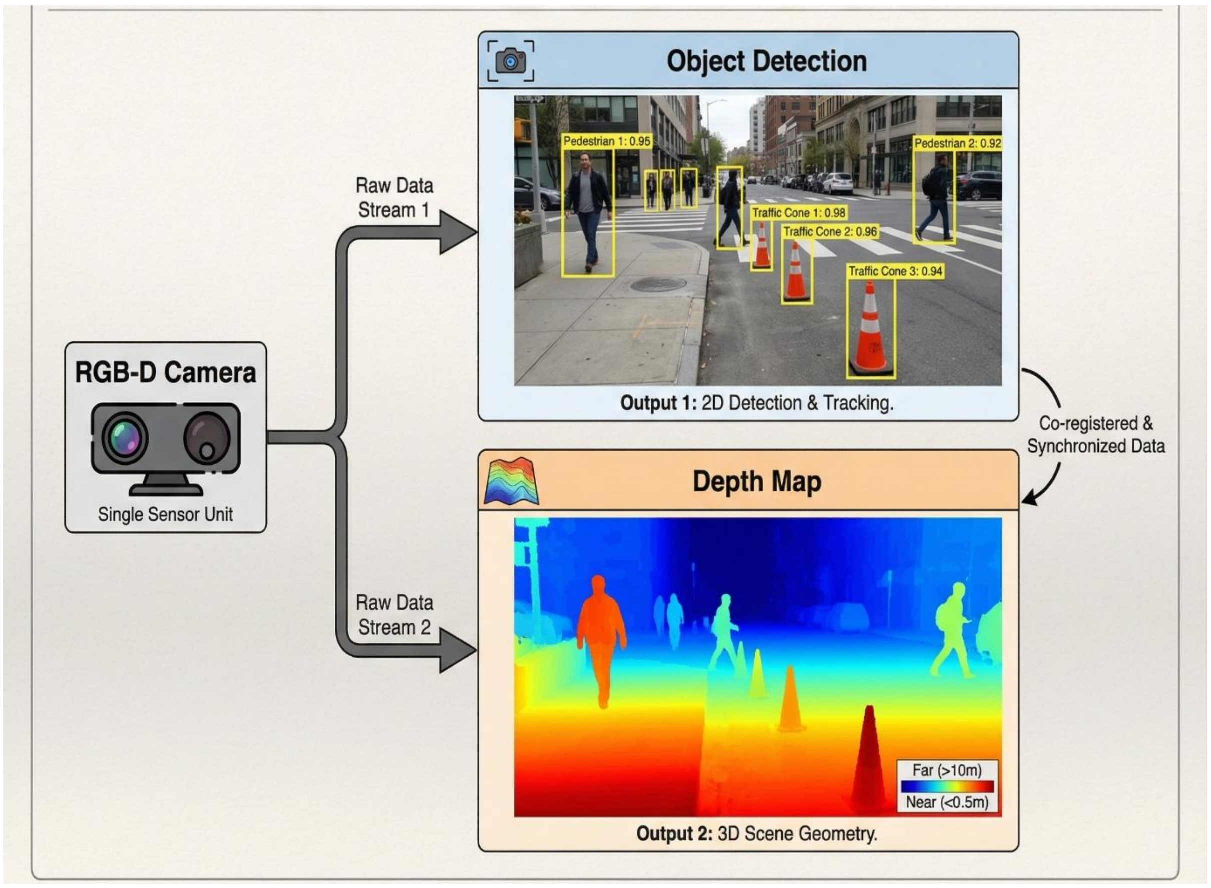


Figure. 3.31 perception pipeline

3.7.1 Perception System Architecture

The perception system integrates multiple sensors and processing stages in a pipeline. For this autonomous delivery robot, the primary sensors are an RGB-D camera (Orbbec Astra Pro) and any auxiliary RGB camera if available. The architecture typically consists of sensor drivers, data processing nodes, and a fusion layer:

- **Sensor inputs:** The Orbbec Astra Pro provides synchronized color (RGB) and depth data. The RGB image serves as input to the object detection network (YOLO11n), while the depth stream provides distance measurements. Optionally, a separate high-resolution RGB camera might be used solely for vision if the Astra's color resolution or frame rate is a limitation. Other common sensors (not detailed here) could include IMUs or wheel encoders, but the focus is on vision and depth.
- **Inference pipeline:** Raw images from the RGB camera are fed into the YOLO11n neural network, which outputs bounding boxes and class scores for detected objects. Simultaneously, the depth camera produces a depth map (a 2D array of distance values). These processes run in parallel, typically on the robot's onboard computing unit (e.g. a Raspberry Pi).
- **Sensor fusion:** After detection, the system fuses information by associating depth values with detected objects. For example, the depth value at the center of each bounding box (or an average depth within the box) is sampled to estimate the object's distance. This yields 3D coordinates (X, Y, Z) of each object relative to the camera. A fusion module may combine this information to output enriched object data (class, 2D bounding box, distance, estimated real-world position).
- **ROS 2 overview:** The system is implemented using ROS 2, a middleware framework for robotic applications. Each sensor and algorithm is encapsulated in a ROS 2 node. For instance, the Astra camera driver node publishes sensor_msgs/Image (RGB) and sensor_msgs/Image (depth) on topics like /camera/rgb and /camera/depth. The YOLO node subscribes to /camera/rgb, runs inference, and publishes detected bounding boxes on a topic (e.g. /detections) using a custom message (e.g. containing x, y, w, h, class_id, score). A depth processing node subscribes to /camera/depth. A fusion node subscribes to both detection and depth topics, performing depth lookup for each detection. The use of ROS 2 (shown in figure) allows modular development and easy integration of perception components with the robot's navigation stack.

Key components: The architecture shown in Figure. 3.32 can be summarized as: Camera Sensors → Preprocessing → YOLO11n Inference Node → Depth Processing Node → Fusion Node → Output (object positions). Each component communicates via ROS 2 topics.

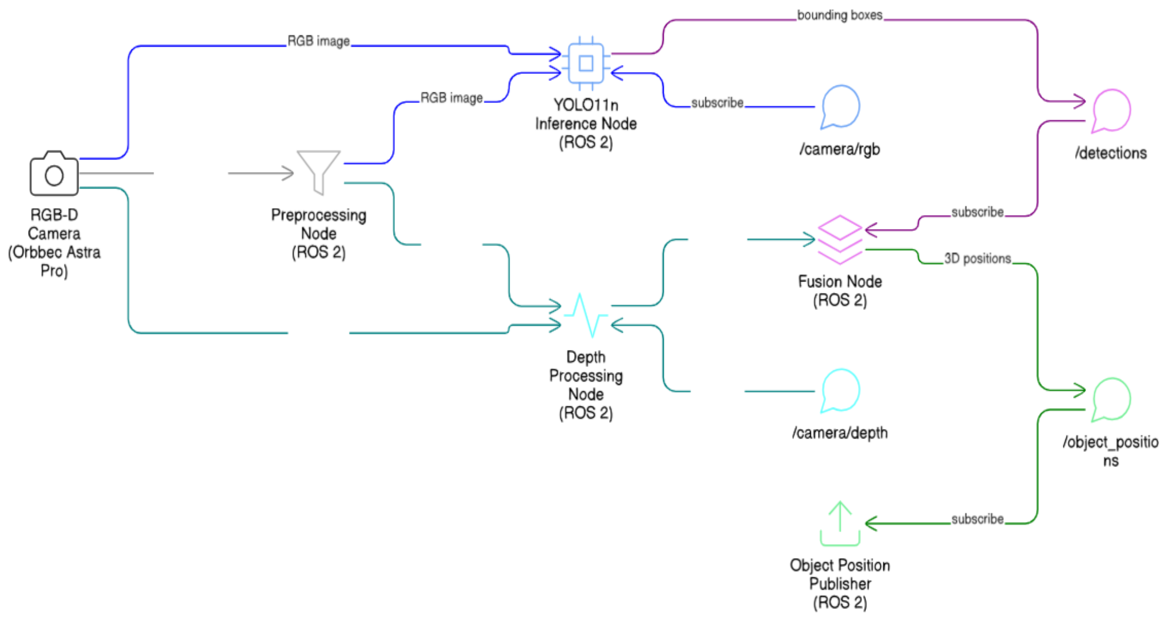

RGB image
RGB image
bounding boxes
YOLO11n Inference Node (ROS 2)
subscribe
/camera/rgb
/detections
RGB-D Camera (Orbbec Astra Pro)
Preprocessing Node (ROS 2)
subscribe
Fusion Node (ROS 2)
3D positions
Depth Processing Node (ROS 2)
/camera/depth
/object_positions
subscribe
Object Position Publisher (ROS 2)

Figure. 3.32 Perception System architecture

3.7.2 Object Detection Using YOLO11n

Object detection is formulated as finding a set of objects and their locations in an image. Mathematically, given an input image

$$I \in \mathbb{R}\hat{}(H \times W \times 3)$$

the goal is to predict N objects, each with a bounding box and class.
A common representation is

$$\{(b\_i, c\_i, p\_i)\}\_(i = 1)\hat{}N$$

where

$$b\_i = (x\_i, y\_i, w\_i, h\_i)$$

defines the center $(\boldsymbol{x\_i}, \boldsymbol{y\_i})$, width $\boldsymbol{w\_i}$, and height $\boldsymbol{h\_i}$ of the $\boldsymbol{i} - th$ box (typically normalized by the image dimensions).
$\boldsymbol{c\_i}$ is the class label (e.g., “person”, “chair”), and $\boldsymbol{p\_i}$ is a confidence score ($objectness$ and class probability).

The detection task can be framed as minimizing a loss L(θ) over network parameters θ, such as

$$L = \sum\_(i = 1)\hat{}N\ (L\_cls(c\_i, c\_i\hat{} *) + L\_box(b\_i, b\_i\hat{} *) + L\_obj(p\_i, p\_i\hat{} *))$$

where $(\boldsymbol{b\_i}\hat{} *, \boldsymbol{c\_i}\hat{} *, \boldsymbol{p\_i}\hat{} *)$ are ground-truth values and $\boldsymbol{L\_cls}, \boldsymbol{L\_box}, \boldsymbol{L\_obj}$ measure **classification**, **localization**, and $\boldsymbol{objectness}$ losses respectively.

3.7.2.1 Image Preprocessing in YOLO11n

Before image data is passed to the YOLO11n network for object detection, it undergoes a series of preprocessing steps designed to standardize and optimize the input. These preprocessing operations are critical to ensuring consistent inference quality, especially in dynamic outdoor environments where lighting and background conditions can vary significantly.

1. Image Resizing

YOLO11n, like other YOLO models, operates on fixed-size input tensors. The input image captured from the camera — regardless of its native resolution — is resized to a standard dimension such as **320×320**, **416×416**, or **640×640** pixels, depending on the deployment constraints and required accuracy. This resizing ensures that the input matches the dimensions of the convolutional filters in the network and maintains a consistent field of view during inference.

To maintain aspect ratio and avoid geometric distortion, **letterboxing** is commonly used. This method resizes the image while preserving its proportions and pads the remaining space with gray or black pixels.

2. Pixel Normalization

Once resized, the image undergoes pixel normalization, where raw pixel values in the range [0, 255] are scaled to a smaller range (e.g., [0, 1]). Additionally, the image is converted from BGR to RGB format as shown in Figure. 3.33, as most deep learning models, including YOLO11n, are trained on RGB images.such as **[0, 1]** or **[-1, 1]**, depending on the model's configuration and training settings. This normalization improves numerical stability during inference and speeds up convergence during training.

In the case of YOLO11n, which is optimized for embedded and edge inference, normalization is often handled by the inference engine (such as ONNX Runtime, TensorRT, or OpenCV DNN) and may be fused into the first layer of the model graph for efficiency

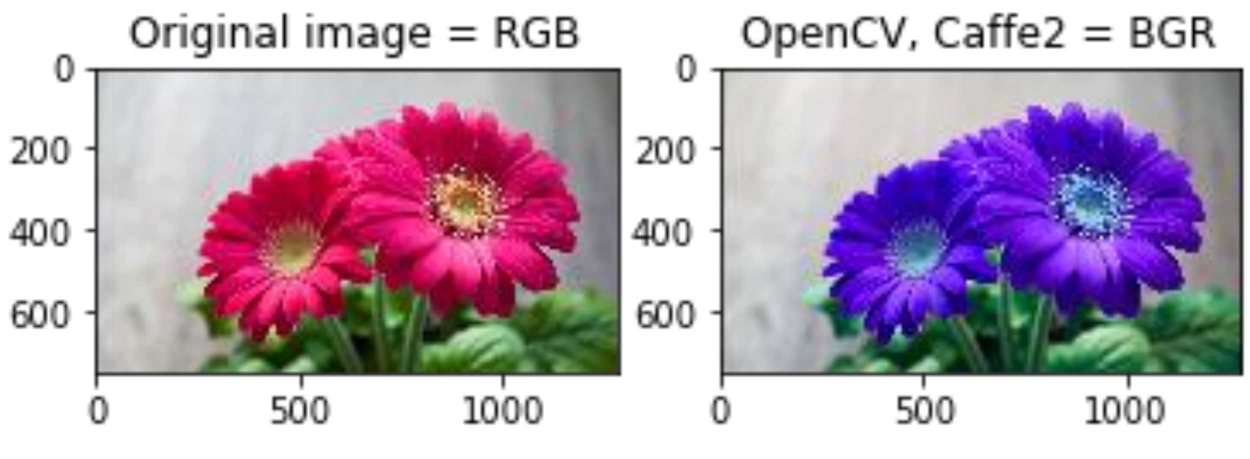


Figure. 3.33 Image processing from BGR to RGB Example

.

3. Channel Ordering

The standard input format for YOLO models is typically a **CHW** layout — that is, Channel × Height × Width — whereas images captured from cameras or read via OpenCV often follow an **HWC** format (Height × Width × Channels). The preprocessing step includes reordering these axes appropriately to match the network's input expectations.

Additionally, some inference backends require the image to be converted from **BGR to RGB** color ordering, as OpenCV captures images in BGR format by default, while most neural networks are trained on RGB data.

4. Batch Formatting

Although inference on a mobile robot often involves processing a single image at a time (batch size = 1), the input is still formatted as a **batch tensor** to satisfy the input structure expected by the model. This involves adding a leading dimension to the image tensor, resulting in a shape of (1, 3, H, W), where '1' represents the batch size.

While YOLO11n in its default form does not apply classical edge detection methods such as Canny or Sobel filters, some variants and research implementations introduce **edge-aware modules** or use **attention maps** that emphasize edge features. In the context of this project, however, standard YOLO11n is used without additional filters, relying on its learned convolutional features to detect object boundaries internally.

3.7.2.2 YOLO Detection Principle

YOLO (**You Only Look Once**) is a one-stage object detection algorithm. Its principle is as follows:

1. Single-pass prediction:
The image is divided into an

$$S \times S$$

grid. Each grid cell is responsible for detecting objects whose center falls within that cell.

2. Predictions per cell:
For each cell, the network predicts a fixed number B of bounding boxes.
Each predicted box has coordinates

$$(x, y, w, h)$$

relative to the cell, an $objectness$ score

$$p\_obj$$

and class probabilities.
In YOLO11n (anchor-free), bounding boxes are often regressed directly without predefined anchor shapes.

3. Joint classification and localization:
The network simultaneously predicts where objects are and what classes they belong to.
In one forward pass, YOLO outputs a tensor of shape

$$S \times S \times D$$

where

$$D = B \cdot (5 + C)$$

(5 numbers per box for $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}, \boldsymbol{h}, \boldsymbol{p\_obj})$ plus C class scores).

4. Loss functions:
YOLO11n is trained using a combination of three loss functions:

1. **Classification loss**: Measures how accurately the model predicts the correct class for each object.
2. **Localization loss**: Measures how well the predicted bounding boxes match the actual object locations in the image.
3. **Objectness loss**: Measures the confidence of the model in detecting the presence of any object at a given location.

5. Overall principle:
YOLO is designed for speed by avoiding a separate region proposal stage.
It performs detection in a fully convolutional fashion, with each grid cell contributing to the final detections.
Non-maximum suppression (NMS) is applied as post-processing to remove overlapping boxes.

3.7.3 YOLO11n Architecture

YOLO11n is the *nano-scale* variant of YOLO version 11, tailored for efficiency on edge devices. Its architecture shown in Figure. 3.34 follows the typical **Backbone – Neck – Head** pattern:

3.7.3.1 Backbone

A lightweight convolutional network extracts **multi-scale features** from the input image.
For YOLO11n, the backbone may be derived from a compact **CSP-Darknet** or other efficient feature extractor.
It consists of several layers of **convolutions**, **normalization** (e.g., BatchNorm), and **activations** (e.g., SiLU), organized in hierarchical blocks such as **C2f** or **BottleneckCSP**.
These blocks progressively **reduce spatial resolution** while **increasing channel depth**, thereby encoding increasingly abstract representations.

Formally, the input image is represented as:

$$I \in \mathbb{R}\hat{}(H \times W \times 3)$$

and the backbone produces a hierarchy of feature maps:

$$F\_k \in \mathbb{R}\hat{}(H\_k \times W\_k \times C\_k), \qquad k = 1,2,\dots,K$$

where each stage k extracts features at a distinct spatial resolution and depth.

3.7.3.2 Neck

The neck combines features from multiple backbone stages to enable **multi-scale detection**.
A typical design employs a **Path Aggregation Network (PAN)** or a **Feature Pyramid Network (FPN)**.
In YOLO11n, the neck fuses **high-level semantic features** with **low-level spatial details**, using **upsampling** and **concatenation** across stages:

$$F\_fused = concat(upsample(F\_high), F\_low)$$

The result is a set of enriched feature maps with complementary information—context and fine-grained detail—used for detecting objects of varying sizes.

3.7.3.3 Head

The detection head transforms the fused features into **final predictions**.
YOLO11n employs a **decoupled head architecture**, consisting of:

- a **regression/objectness branch**, and
- a **classification branch**.

At each detection scale, the head outputs a tensor of size:

$$S \times S \times B \times (5 + C)$$

- where S×S denotes the grid resolution,
- B is the number of boxes per cell,
- C is the number of classes
  - Each predicted element contains (x, y, w, h, p_obj) and C class probabilities.

Because YOLO11n is **anchor-free**, it predicts **offsets directly** relative to each grid cell rather than refining predefined anchors.

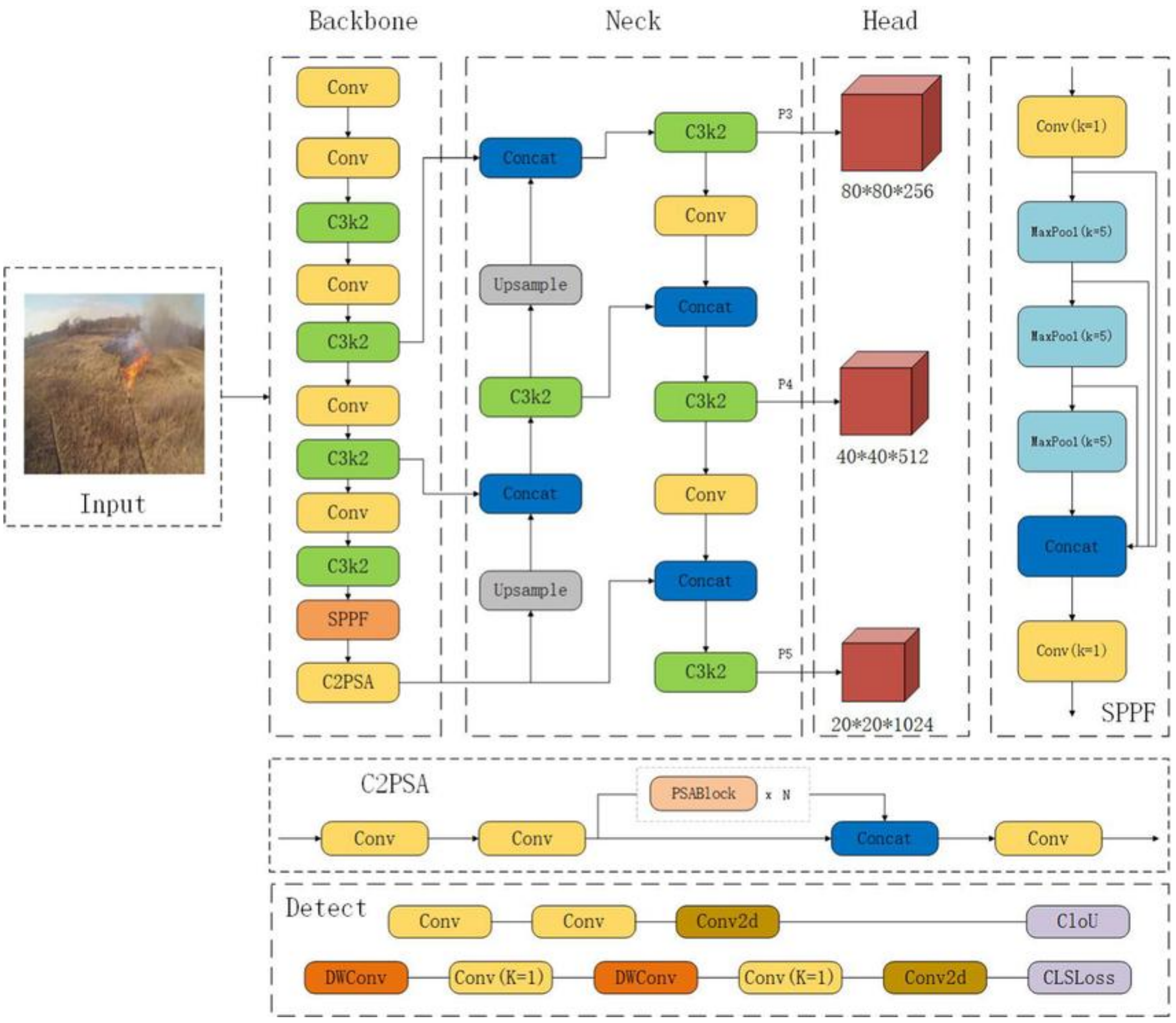


Figure. 3.34 The YOLO11n network

3.7.4 Depth Perception Using Orbbec Astra Pro

Depth perception enables the robot to understand the three-dimensional structure of its environment, which is essential for safe navigation, obstacle avoidance, and object localization. The Orbbec Astra Pro is an RGB-D sensor designed for this purpose, offering both color (RGB) and depth information in real time. It provides synchronized streams from a standard RGB camera and a depth-sensing unit, allowing accurate per-pixel measurement of distances to objects in the scene. See figure

3.7.4.1 Depth Sensing Principles

The Orbbec Astra Pro operates using a combination of a color imaging sensor and an infrared (IR) depth sensor. The depth sensing relies on either **structured light** or **active stereo vision** techniques:

- In **structured light**, the sensor projects a fixed infrared pattern (such as a grid or dot pattern) onto the environment. This pattern is then observed by an IR camera. As objects deform the pattern based on their shape and distance, the sensor analyzes these distortions to estimate how far each point is from the camera.
- In **active stereo vision**, the device uses two infrared cameras placed at a fixed distance from each other. These cameras view the same scene from slightly different angles. By comparing how features in the scene shift between the two views, the system can infer depth through triangulation.

Both methods allow the camera to generate a depth map—an image in which each pixel represents the distance from the sensor to the corresponding point in the scene.

An alternative technology used by some depth cameras is **Time-of-Flight (ToF)**, which estimates depth by measuring how long it takes infrared light to bounce back from surfaces. However, the Astra Pro typically uses structured light or stereo-based methods rather than ToF.

Importantly, the Astra Pro also provides **RGB-depth alignment**. This means that the depth data is calibrated to match the RGB image on a pixel-by-pixel basis. This alignment allows software systems to directly associate color and geometry, which is critical for tasks like colored point cloud generation or fusing object detection results with spatial measurements.

3.7.4.2 Depth Mapping and Coordinate Estimation

The Astra Pro generates a continuous stream of depth maps as shown in Figure. 3.35, 2D images where the intensity of each pixel corresponds to the distance from the camera to a point in the environment. These maps are produced in parallel with the RGB video stream and can be precisely aligned with the color image, enabling depth information to be extracted for any region of interest.

To estimate the real-world position of a pixel in three dimensions, the camera uses its internal calibration data. This includes information such as the position of the optical center and how the image is scaled. Using this data, software algorithms can convert pixel locations and their corresponding depth values into actual spatial coordinates (for example, in meters), allowing the robot to understand where objects are located in its field of view.

The calibration process is typically handled during the setup of the camera and uses standard camera models provided by vision libraries or the camera's SDK. The result is that every depth map not only gives information about how far objects are, but also enables accurate 3D reconstructions or measurements when needed.

3.7.4.3 Depth Map Representation and Characteristics

The depth output from the Astra Pro is typically represented as a grayscale image or numerical array, where each pixel holds a depth value. Some of the key characteristics of this depth data include:

- **Resolution and Frame Rate:** The Astra Pro can deliver depth maps at resolutions up to 640×480 pixels, with frame rates up to 30 frames per second. This makes it suitable for real-time applications such as mobile robot navigation.
- **Field of View and Range:** The camera has a horizontal field of view of approximately 60 degrees and is optimized for depth sensing in the range of about 0.6 to 5 meters. Depth accuracy is typically best within this range. Closer than 0.6 meters, the sensor may be unable to compute depth, and at longer distances, the depth values become less reliable or unavailable.
- **Noise and Artifacts:** Like all active depth sensors, the Astra Pro can exhibit noise, especially at greater distances or in complex lighting environments. Reflective, shiny, or transparent surfaces may result in invalid or missing depth values. External light sources, especially sunlight, can interfere with the infrared signals used by the sensor, reducing its effectiveness in outdoor settings.
- **Data Format:** Depth values are usually represented as 16-bit unsigned integers or floating-point values, depending on the software interface. These values represent the distance to the object in millimeters or meters. Depth data can be smoothed or filtered to reduce noise, using common image processing techniques like median or bilateral filtering.

Having access to structured depth data enables the perception system to build a spatial understanding of the robot's surroundings. It allows the system to estimate distances to obstacles, detect free space, and generate 3D representations of the environment. This is especially important for outdoor delivery tasks, where the robot must identify and avoid curbs, poles, vehicles, and other hazards in real time.

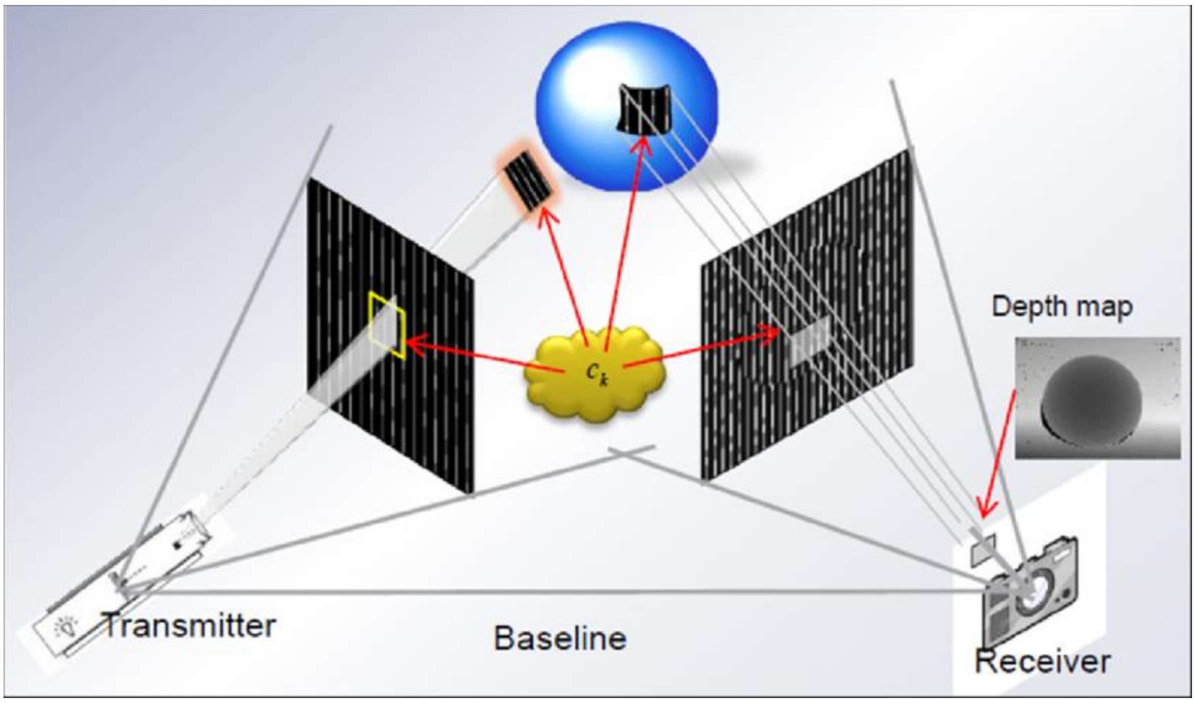


Figure. 3.35 how the Astra Pro uses infrared projection and camera sensing to generate a depth map

3.7.5 YOLO–Depth Sensor Fusion

Sensor fusion shown in Figure. **3.36** is the process of combining data from multiple sources to produce a more informative and robust perception of the environment. In the context of this system, fusion is performed between the outputs of the YOLO object detection algorithm and the depth data provided by the Orbbec Astra Pro camera. This integration allows the robot not only to identify objects but also to determine their spatial positions in three dimensions.

The fusion approach implemented here is efficient and well-suited for real-time applications on embedded hardware. It operates by associating each detected object in the RGB image with its corresponding depth information from the aligned depth map. The result is an enriched detection output that includes not only the object class and bounding box but also its estimated 3D position relative to the robot.

The fusion procedure follows these steps:

1. **Sampling depth at the center of detections:** Once an object is detected in the RGB image using YOLO, the system identifies the center point of the bounding box surrounding the object. It then retrieves the corresponding depth value from the aligned depth image at or near that center point. To reduce the effects of noise or missing data, the system may average the depth values within a small region around the center, rather than relying on a single pixel.
2. **Estimating 3D position:** With the depth value obtained, and using the camera's calibration parameters, the system estimates the 3D position of the detected object relative to the camera frame. This conversion transforms the 2D image coordinates into a three-dimensional location, typically expressed in real-world units such as meters. The resulting position can be interpreted as the approximate location of the object's center in space.
3. **Generating fused output:** Each object detection is now augmented with its estimated 3D position, the object's class label, and the detection confidence score. This information is published or logged as a unified data structure, which can be consumed by higher-level modules such as obstacle avoidance, path planning, or mapping.

This method provides a simple yet effective means of localizing objects in 3D, which is crucial for safe and intelligent robotic behavior. While it assumes that the detected object faces the camera and that the center of the bounding box is a reasonable approximation of its depth, these assumptions hold well in practice for many navigation scenarios. Moreover, because the process is computationally lightweight, it is compatible with the resource constraints of edge devices such as the Raspberry Pi.

Future enhancements could include more advanced techniques, such as fitting 3D bounding boxes using multiple depth samples across the detection area or incorporating object tracking across frames for temporal consistency. However, the current approach offers a reliable and practical foundation for real-time perception in delivery robotics.

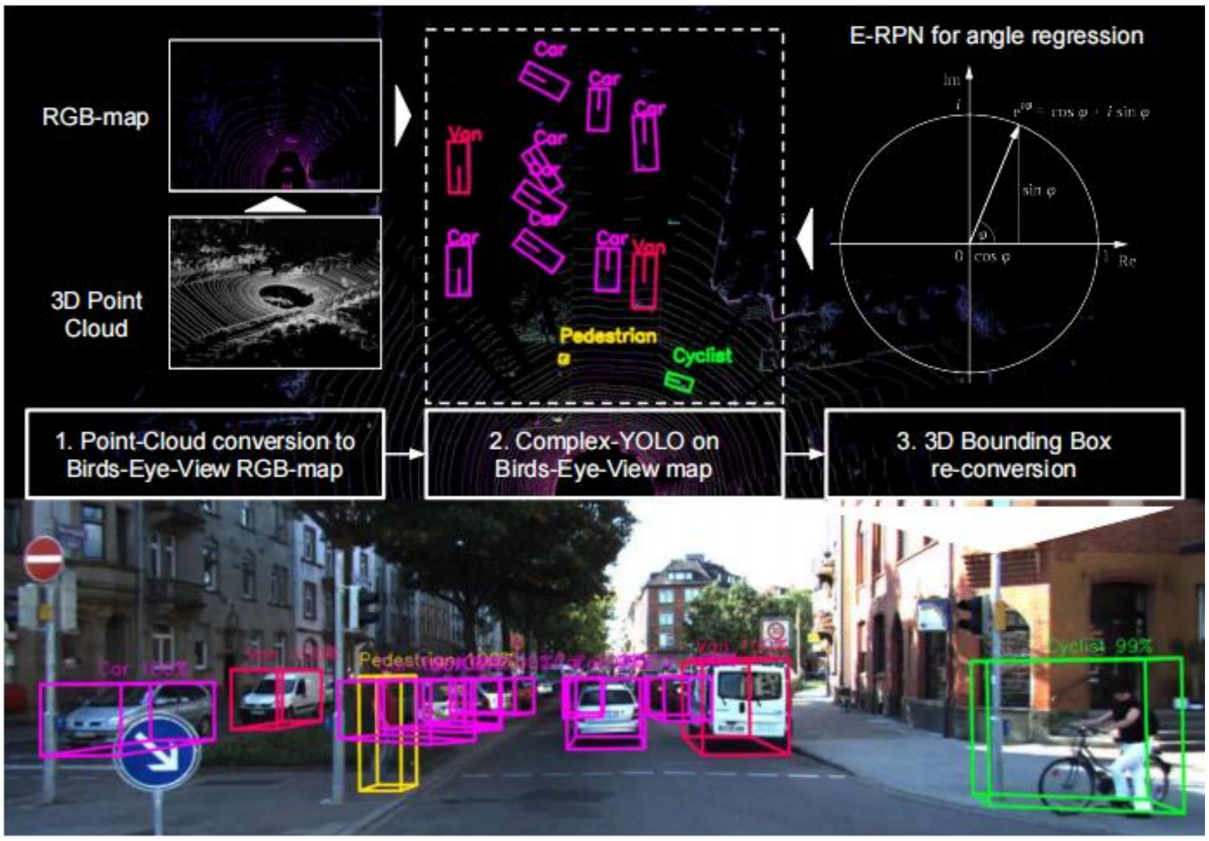

RGB-map
3D Point Cloud
E-RPN for angle regression
Car
Van
Pedestrian
Cyclist
Im
Re
$e^{i\varphi} = \cos\varphi + i\sin\varphi$
sin φ
cos φ
1. Point-Cloud conversion to Birds-Eye-View RGB-map
2. Complex-YOLO on Birds-Eye-View map
3. 3D Bounding Box re-conversion
Cyclist 99%

Figure.** 3.36 **camera view with detected objects marked by bounding boxes, alongside a 3D point cloud

3.7.6 Edge AI Implementation on Raspberry Pi

Deploying AI on the edge (on-device) has trade-offs compared to cloud processing. Key considerations for using a Raspberry Pi on a delivery robot include:

- **Edge AI vs. Cloud AI:** Edge AI means all computations run locally on the robot's processor. This minimizes latency (no network round-trip), increases reliability (no dependence on connectivity), and improves privacy/security (data stays on device). In contrast, cloud AI would offload processing to remote servers, requiring constant Wi-Fi/4G and suffering delays. For a delivery robot navigating unknown outdoor environments, local (edge) perception is preferable to react quickly to obstacles. However, edge hardware like a Raspberry Pi is far less powerful than data-center GPUs, so models must be smaller and more optimized.
- **Why Raspberry Pi:** The Raspberry Pi platform is widely used in robotics for prototyping. It is compact, low-cost, and has sufficient I/O (USB for camera, networking). The newer Pi models (Pi 4, Pi 5) have multi-core ARM CPUs (up to 4 cores @ ~2 GHz) and a GPU that can run OpenGL/OpenCL. While not as fast as dedicated AI boards (e.g. NVIDIA Jetson), a Pi can still run lightweight neural networks with optimizations. The community support and OS (Linux-based Raspbian/Ubuntu) make development easier. For this robot, Raspberry Pi strikes a balance between cost and capability for running YOLO11n in real time.

- **Model optimization strategies:** To enable YOLO11n on the Pi, several techniques are used:
    - *Model quantization:* Converting the network weights from 32-bit floating point to 16-bit or 8-bit integers can drastically reduce memory and speed up inference on CPUs and NPUs. For example, an 8-bit quantized model can run with vectorized integer instructions on the Pi's CPU.
    - *Pruning and simplification:* Removing redundant neurons/filters or using a smaller variant (YOLO11n is already the nano version) reduces computation.
    - *Efficient inference frameworks:* Using optimized libraries such as TensorFlow Lite, ONNX Runtime with ARM optimizations, or OpenCV's DNN module can improve speed. If available, hardware acceleration (e.g. utilizing the Pi 4's VideoCore GPU via OpenCL, or attaching a Google Coral USB TPU) can further boost throughput.
    - *Resolution and framerate adjustment:* Running the network on lower-resolution images (e.g. 320×320 instead of 640×640) or skipping frames can meet real-time constraints at the cost of some accuracy.
    - *Parallel processing:* The Pi's multiple CPU cores can run the vision pipeline alongside other tasks. For example, one core handles camera capture and depth processing while others run the neural network and fusion.

By combining these strategies, the ROS 2 perception nodes on the Pi can achieve inference speeds of several frames per second with YOLO11n. The edge implementation enables the robot to function entirely autonomously without offboard computation.

In this section, we have outlined the design of an AI perception system for an autonomous delivery robot, focusing on real-time object detection and depth estimation. We began by defining robotic perception and its critical role in outdoor navigation, emphasizing constraints like computation and environmental variability. The system architecture integrates an Orbbec Astra Pro RGB-D sensor and a YOLO11n neural network, all orchestrated within ROS 2. The object detection process was formalized mathematically and implemented via the YOLO11n model, whose backbone-neck-head architecture enables fast, one-shot detection. Depth perception was addressed using the camera's stereo-based depth sensing. By fusing YOLO's 2D bounding boxes with the depth map, the robot obtains 3D positions of obstacles.

We also discussed deploying this perception stack on edge hardware (a Raspberry Pi), highlighting the trade-offs of edge vs. cloud processing. Model optimization techniques and resource considerations ensure the system runs in real time. Finally, key performance factors (FPS, latency, power usage) and limitations (range, tracking, sensor robustness) were analyzed. Future improvements, such as adding neural accelerators or LiDAR, were suggested to enhance capability.

In conclusion, the described AI perception system combines state-of-the-art object detection (YOLO11n) with active depth sensing to provide an autonomous delivery robot with the necessary environmental awareness. The modular design allows further refinements, and the inclusion of suggestions for system and network diagrams will aid in visualizing and implementing the architecture.

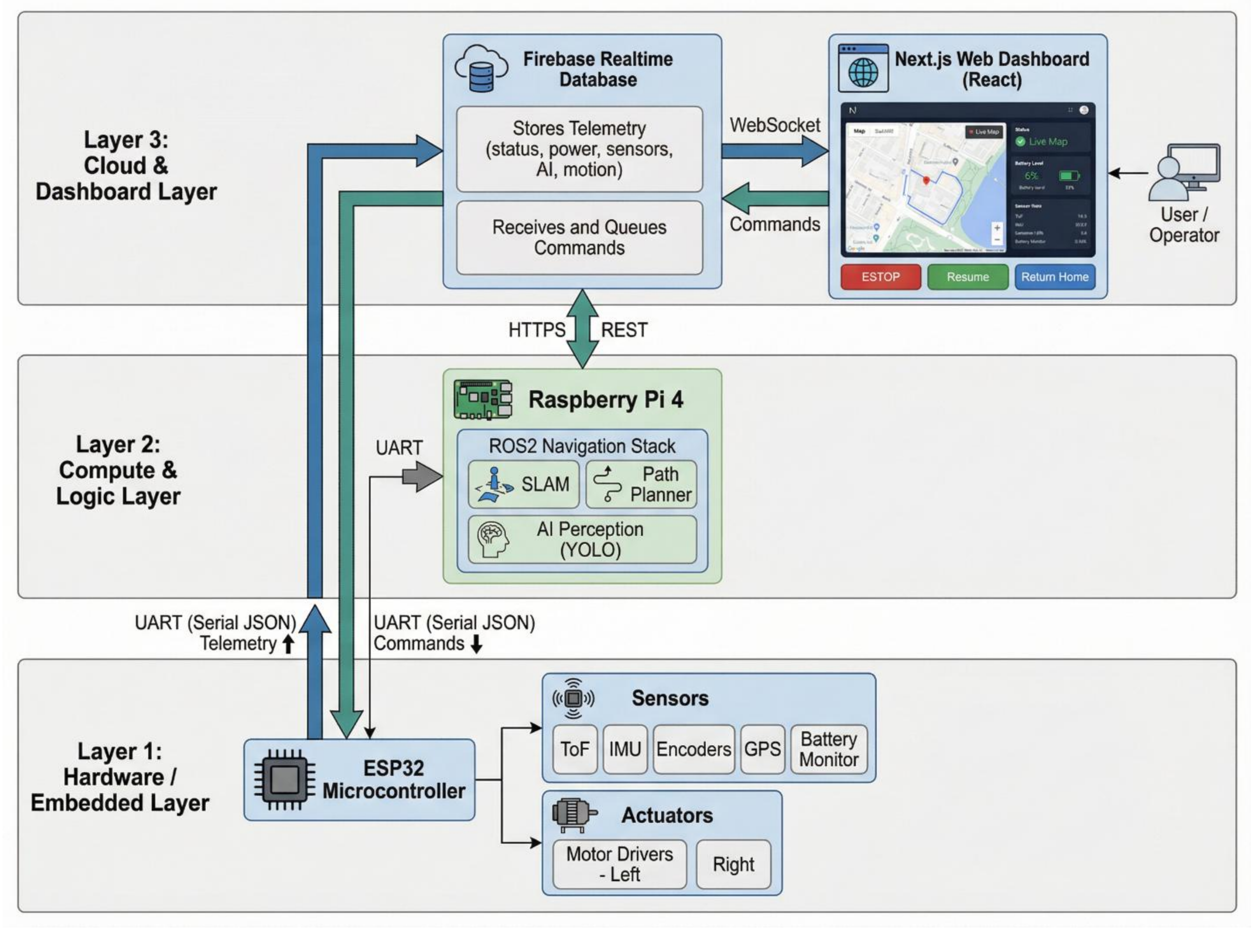
3.8 IoT Integration

This section describes the Internet-of-Things (IoT) integration of the autonomous mobile delivery robot system. The system employs a three-layer architecture comprising:

1. a low-level embedded layer (ESP32 microcontroller) for real-time motor control and sensor acquisition,
2. a high-level compute layer (Raspberry Pi 4) running the ROS2 navigation and AI perception,
3. a cloud and frontend layer (Firebase and a Next.js dashboard) for data aggregation,

monitoring, and operator control. This layered architecture is illustrated in Figure. 3.37(System Architecture Diagram). In this design, on-board sensors and actuators are interfaced with the ESP32; the Raspberry Pi performs sensor fusion, mapping, and high-level decision making; and the cloud database synchronizes state and command data with the web dashboard for real-time monitoring and control. The figure highlights the primary communication paths: bidirectional command flows (Dashboard ↔ ESP32 via the cloud) and largely unidirectional telemetry flows (sensors → ESP32 → Raspberry Pi → Firebase → Dashboard). These interconnections use common IoT protocols (UART serial for local links, HTTPS and WebSocket for network links).

Hardware and Software Components. The hardware comprises an ESP32 microcontroller, a Raspberry Pi 4, motor driver boards for the left and right drive motors, a 24 V nominal Li-ion battery pack, and a hardware emergency-stop button tied to the ESP32 (interrupt line). The sensor suite includes five Time-of-Flight (ToF) range sensors, a 9-axis IMU (accelerometer, gyroscope, magnetometer), a GPS receiver (NEO-6M), wheel encoder sensors, and circuits for battery voltage and current measurement. The ESP32 is responsible for real-time motor control and continuous acquisition of safety-critical telemetry (e.g., battery voltage/current, ToF distances, wheel RPM, etc.). The Raspberry Pi hosts the ROS2 navigation stack and computer vision or AI perception modules; it also aggregates data from the ESP32 and sensors to compute state estimates (pose, velocity, map). Cloud services are provided by Firebase: a Realtime Database (hosted in asia-southeast1 region) stores live telemetry and command queues. A custom web dashboard is implemented with Next.js (React/TypeScript) that uses the Firebase SDK (WebSocket-based) for real-time updates, with an integrated map display (using Mapbox) for navigation visualization.



Layer 3: Cloud & Dashboard Layer
Firebase Realtime Database
Stores Telemetry (status, power, sensors, AI, motion)
Receives and Queues Commands
WebSocket
Commands
Next.js Web Dashboard (React)
ESTOP
Resume
Return Home
User / Operator
HTTPS
REST
Layer 2: Compute & Logic Layer
UART
Raspberry Pi 4
ROS2 Navigation Stack
SLAM
Path Planner
AI Perception (YOLO)
UART (Serial JSON) Telemetry
UART (Serial JSON) Commands
Layer 1: Hardware / Embedded Layer
ESP32 Microcontroller
Sensors
ToF
IMU
Encoders
GPS
Battery Monitor
Actuators
Motor Drivers - Left
Right

Figure. 3.37 IOT System Architecture Diagram.

This figure shows the three-tier structure: the ESP32 (Layer 1) interfaces with hardware sensors and actuators, the Raspberry Pi (Layer 2) performs local processing, and the Firebase-backed web dashboard (Layer 3) provides remote monitoring and control. Communication protocols are annotated: a high-speed serial link (UART with JSON) connects the ESP32 to the Pi, while the Pi communicates with Firebase over HTTPS/REST (Wi-Fi) and the dashboard uses Firebase's WebSocket API. Telemetry flows from Layer 1/2 up to Layer 3, whereas operator commands flow from Layer 3 down to Layer 1.

3.8.1 Communication Infrastructure

Efficient, reliable communication is essential for coordinating the onboard controllers with the cloud and dashboard. Table 5 summarizes the main communication links and protocols. The ESP32 communicates with the Raspberry Pi via a USB-serial connection at 115200 baud, sending JSON-formatted telemetry packets at 10 Hz. This UART link exhibits typical latency under 2 ms and includes a simple heartbeat: if no data are received for more than 2 s, the Raspberry Pi sets an esp32_connected=false flag to indicate a failure. The Raspberry Pi pushes aggregated telemetry to the cloud using HTTPS PUT/GET REST calls (TCP/IP over Wi-Fi), with JSON encoding; typical round-trip latency is 50–200 ms. Failures on this link are detected via HTTP error codes and timeouts, with a retry mechanism (3× attempts, 5 s timeout).

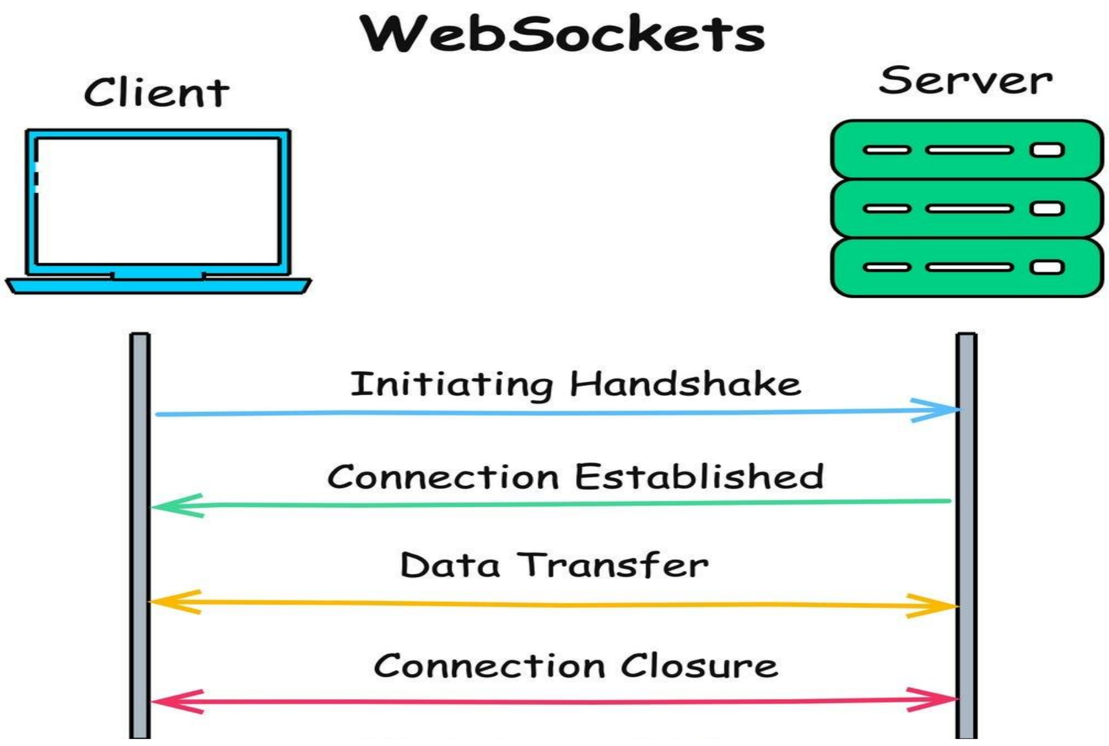


Figure. 3.38 WebSocket Communication

The dashboard subscribes to the Firebase Realtime Database using a WebSocket-based listener as shown in Figure. 3.38 , receiving pushed JSON updates on any changed values. After initial setup, this link yields updates in under 50 ms. Connection health is monitored via Firebase's .info/connected and websocket close events; if the WebSocket disconnects, the dashboard indicates offline status. Operator commands from the dashboard are sent to the cloud via HTTPS REST PUT requests (command payloads in JSON). These incur 50–200 ms latency and return promise rejections on failure, which are caught for retry or user warning. The Raspberry Pi polls Firebase for commands via periodic HTTPS GET requests (every 2 s). These GETs have similar latency and detect errors via HTTP codes. Once the Pi retrieves a command, it forwards motor or state commands to the ESP32 over the same serial link (UART, JSON, <2 ms latency). The ESP32 expects an acknowledgment from the Pi (if none is received, it assumes a serial failure). In summary, all telemetry flows upward (ESP32 → Pi → Firebase → Dashboard) and commands flow downward (Dashboard → Firebase → Pi → ESP32) via these layered links.

Link	Protocol	Transport	Data Format	Latency	Failure
ESP32 → Raspberry Pi	UART Serial	USB/UART	JSON strings	< 2 ms	No data >2 s ⇒ esp32_connected=false[9]
Raspberry Pi → Firebase (telemetry)	HTTPS REST	TCP/IP (Wi-Fi)	JSON	50–200 ms	HTTP error codes; retry (3×), 5 s timeout[10]
Firebase → Dashboard	WebSocket	TCP/IP (Internet)	JSON (push)	< 50 ms	.info/connected listener; WebSocket close[12]
Dashboard → Firebase (commands)	HTTPS REST PUT	TCP/IP (Internet)	JSON	50–200 ms	Promise rejection; error callbacks[13]
Firebase → Raspberry Pi (commands)	HTTPS GET (poll)	TCP/IP (Wi-Fi)	JSON	50–200 ms	HTTP error codes; poll interval 2 s[14]
Raspberry Pi → ESP32 (commands)	UART Serial	USB/UART	JSON strings	< 2 ms	No ACK ⇒ timeout/failure[15]

Table 5 Key IOT Communication Links

Each communication link is designed with redundancy and fault detection. For example, the Pi→Firebase link retries on HTTP failure, and the ESP32 monitors lack of UART input to signal connectivity loss. The WebSocket link from Firebase includes a built-in status endpoint to detect disconnections, enabling the dashboard to notify the operator (e.g. "Connection Lost" badge) if the cloud sync fails. In practice, normal operation yields sub-100 ms round-trip latencies end-to-end, ensuring near-real-time responsiveness for commands and telemetry.

3.8.2 Cloud Database and Data Architecture

The robot's cloud communication and data synchronization are managed through a **Firebase Realtime Database**, a cloud-hosted **NoSQL** solution optimized for hierarchical, path-based, real-time data exchange. In the context of embedded and robotic IoT systems, the selection of an appropriate database architecture directly impacts performance, consistency, latency, and scalability. This section explains the structural design of the data architecture, justifies the database model selection, and situates it within the taxonomy of modern database systems.

3.8.2.1 Database Classification and Rationale

Databases are typically classified into two main categories:

- **Relational (SQL)** databases: characterized by tabular data models, predefined schemas, and use of the Structured Query Language (SQL). Examples include MySQL, PostgreSQL, and SQLite.
- **Non-relational (NoSQL)** databases: designed for flexibility, hierarchical or document-based storage, and scalable distributed architectures. Examples include Firebase Realtime Database, MongoDB, and Cassandra.

The system implements **Firebase Realtime Database**, a NoSQL **hierarchical key–value store**. Unlike relational databases which rely on strict schema definitions and table-based normalization, Firebase allows dynamic data modeling using JSON-like structures and supports high-frequency updates via event-driven synchronization. This choice aligns with key IoT system requirements as shown in Table 6:

Requirement	NoSQL Advantage
Frequent, rapid updates	Optimized for high write throughput (e.g., 10 Hz)
Real-time synchronization	Built-in WebSocket push-based updates
Hierarchical sensor data	Supports deeply nested key-value trees
Schema flexibility	Adaptable to changing telemetry formats
Low-latency, lightweight transport	Designed for mobile, embedded, and constrained devices

Table 6 IoT system requirements for the ADR

Relational databases, while offering strong consistency and transaction integrity, are less suited to time-sensitive telemetry updates and schema-evolving sensor data typical of robotics. Moreover, the lack of JOIN operations in Firebase is not a limitation in this context, as the robot's telemetry is partitioned into independent logical domains and consumed as whole nodes in the dashboard.

Thus, Firebase Realtime Database is positioned as a **cloud-hosted, eventually-consistent, real-time NoSQL datastore**, ideal for IoT systems requiring continuous data streaming, moderate-scale retention, and low-latency access by web clients.

3.8.2.2 Hierarchical Schema Design

The Firebase database structure follows a **rooted tree topology**, with the root node /robot containing all mission-critical telemetry, status, perception, control state, logs, and commands. This design mirrors object-oriented decomposition, where each subtree maps to a functional module or sensor class. Data are stored as **flat key–value pairs or nested objects**, depending on their granularity.

Each key path supports **real-time updates**, and the web dashboard subscribes to only the necessary paths, enabling fine-grained and efficient monitoring. Subtrees are either **real-time (holding only the most recent value)** or **historical (log-structured entries)**. The main subtrees under /robot and their functional roles are as shown in Figure. 3.39:

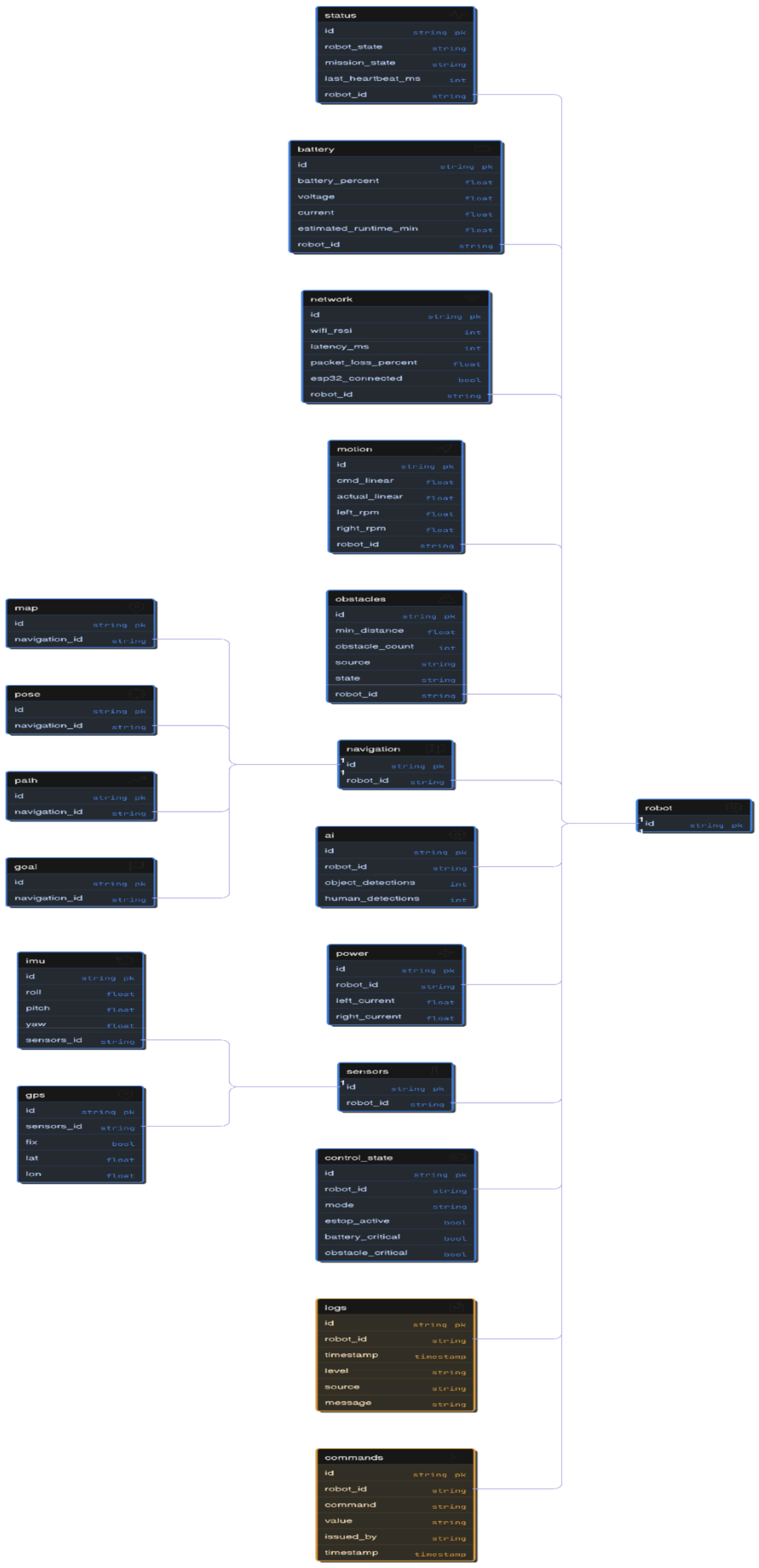


Figure. 3.39 Subtree Structure and Update Frequencies

In academic terms, the database satisfies core characteristics of **IoT-aware NoSQL design**:

- **BASE model (Basically Available, Soft state, Eventually consistent)** is acceptable in robotic supervision since brief desynchronization (<200 ms) does not compromise physical safety (E-STOP is handled on-board).
- **High Write Availability** supports concurrent 10–20 Hz updates from multiple subsystems.
- **Key–Value hierarchy** matches natural robotic modularity (e.g., separating motion, power, perception).
- **Subscription-based change listeners** (WebSocket) provide real-time updates without polling overhead.

Furthermore, Firebase's structure avoids complexity like foreign keys or multi-table joins, which are overkill in the time-sensitive, single-client context of real-time robotics. This reduces delay and logic overhead, enabling system reactivity.

3.8.2.3 Data Prioritization and Telemetry

Given the mix of safety-critical and non-critical data, the system prioritizes telemetry into four tiers:

- **Safety-Critical:** Highest priority. This includes emergency stop status (estop), robot operational state (robot_state), heartbeat (last_heartbeat_ms), battery level (battery_percent), nearest obstacle distance (min_distance), obstacle critical flag (obstacle_critical), ESP32 connectivity (esp32_connected), and human presence detections (human_detections). These values are transmitted and monitored with the greatest urgency; any change triggers immediate safety logic (e.g. cutting power or issuing alerts).
- **Real-Time Operational:** Next level. This covers motion commands and feedback (cmd_linear, cmd_angular, actual_linear), wheel speeds (left_rpm, right_rpm), robot pose (robot_pose), planned route (navigation_path), raw obstacle distances (tof_sensors[5]), and power telemetry (battery_voltage, battery_current, IMU roll/pitch/yaw). These data are used for closed-loop control and navigation; they are updated at high rates (10 Hz) to ensure smooth operation.
- **Diagnostic:** Lower priority. This includes network metrics (wifi_rssi, latency_ms, packet_loss_percent), motor load (left_current, right_current), localization confidence, mission phase (mission_state), system uptime (uptime_sec), etc.. These values help with troubleshooting and performance monitoring but are not critical for immediate control.
- **Historical:** Lowest priority. This encompasses logs, command history, and full environmental map (occupancy grid). These are stored for post-mission analysis and diagnostics, updated infrequently (event-driven or <1 Hz).

By classifying data, the system ensures that, under bandwidth or processing constraints, safety-related information is given preference. For example, while the map might update at 0.1 Hz and occupy bandwidth, a sudden obstacle reading (safety-critical) or a command to stop (safety-critical) would interrupt and take precedence. The cloud database schema and the dashboard UI also reflect this priority: safety-critical fields are prominently displayed with immediate updates, while historical logs are available but do not clutter the real-time display.

3.8.3 Control and Command Interface

Operators interact with the robot via the web dashboard, which allows issuing high-level commands through the cloud backend. The main commands are listed below, along with their targets, preconditions, safety priorities, and expected acknowledgments:

- **ESTOP:** Issued from the Dashboard to the ESP32 (hardware emergency stop). No preconditions (always available). Safety priority = *HIGHEST*. Expected effect: the ESP32 immediately disables the motor drivers (hardware interrupt) and sets estop = true in its telemetry (within ~100 ms). The dashboard then shows an "E-STOP" indicator (red) and prohibits further motion commands.

- **RESUME:** Issued from the Dashboard to the Raspberry Pi state machine. Preconditions: no active emergency stop (estop_active=false), battery not critical, network connected, no obstacle critical (obstacle_critical=false). Safety priority = *High*. Upon execution, the Pi transitions the robot state out of E_STOP or ERROR into MOVING or IDLE, resuming normal operation. The dashboard waits for robot_state to change accordingly as acknowledgment.

- **RETURN_HOME:** Issued to the Raspberry Pi's navigation stack. Preconditions: robot_state != E_STOP and battery percentage > 10%. Safety priority = *High*. This command sets the mission state to RETURNING and updates the navigation goal to the home/base coordinates. The acknowledgment is seen when mission_state becomes RETURNING and the goal node updates.

- **SET_MODE:** Issued to the Raspberry Pi state machine to switch operation mode (e.g. AUTONOMOUS vs MANUAL). Preconditions: robot_state != E_STOP. Safety priority = *Medium*. Acknowledgment is achieved when the /control_state/mode field in the database reflects the new mode.

- **RESET:** Issued to the Raspberry Pi (software reset). Preconditions: robot_state != MOVING (i.e. safe to restart). Safety priority = *Medium*. Expected effect: the Pi reboots, causing uptime_sec to reset to zero. The dashboard may observe a temporary loss of data until the system comes back online.

The command flow is depicted in Figure. 3.40 (Control Flow Diagram). When the user clicks a button, the dashboard writes a command entry to the /commands queue in Firebase. The Raspberry Pi periodically polls this queue and, upon retrieving a new command, validates its preconditions. It then forwards the action: for motor-related commands (RESUME, RETURN_HOME, etc.), it adjusts the ROS2 state or sends serial commands to the ESP32 as needed. Each subsystem acknowledges completion by writing status updates back to Firebase (e.g. the ESP32 sets estop = true, the Pi updates robot_state). The acknowledgment path thus goes ESP32 → Raspberry Pi → Firebase → Dashboard. At each stage, safety gates enforce preconditions: for instance, any command is ignored if estop is active, and RESUME is blocked unless the network is healthy and no obstacle is critical.

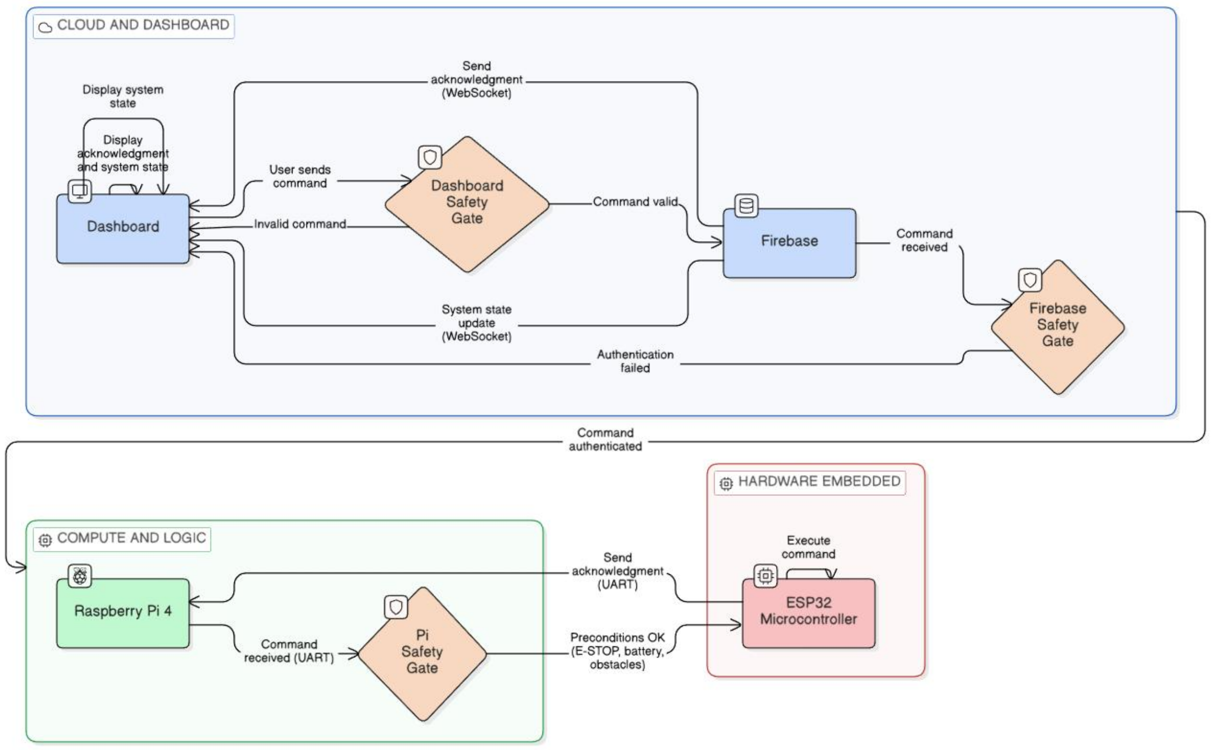


Figure. 3.40 Control Flow Diagram.

Commands originate at the dashboard (layer 3) and travel down through Firebase to the on-board controllers (layers 2 and 1). The return path shows how acknowledgments propagate back. Safety preconditions are checked at each handoff (e.g. Pi won't execute a command if estop is set).

3.8.3.1 Safety Mechanisms

Safety is woven into the IoT integration through multiple mechanisms: heartbeat monitoring, emergency-stop handling, threshold-based guards, and failure responses. The combined goal is to guarantee that the robot can be safely stopped or alerted under any fault or hazard.

Heartbeat Monitoring: Both the dashboard and the robot controllers monitor periodic "heartbeat" signals to detect disconnections. The Raspberry Pi publishes a last_heartbeat_ms timestamp (updated every 100 ms). If the dashboard observes that this exceeds 1000 ms, it triggers a "CONNECTION LOST" warning to the user. Similarly, the Pi expects regular serial data from the ESP32; if no data arrives for >2 s, it flags esp32_connected=false and assumes the ESP32 is offline. These timeouts ensure that loss of link is caught quickly and displayed as a critical alert on the UI.

Emergency-Stop (E-STOP) Logic: A dedicated GPIO-connected E-STOP button on the ESP32 provides a hardware-interrupt line. When pressed (or any condition sets estop=true), the ESP32 immediately cuts power to the motor drivers at the hardware level. It also updates its telemetry with estop = true within 100 ms. The Raspberry Pi and dashboard immediately pick up this flag: the UI shows a red “E-STOP” badge, and all motion commands are disabled until cleared. This ensures the highest-priority response to any critical situation.

Threshold-Based Protections: The system continuously evaluates telemetry against safety thresholds (Table 6.2 summarizes key thresholds). For example, battery charge <15% triggers a “RETURN TO BASE” warning on the dashboard. Battery state of charge also controls UI color: below 20% it turns red, below 40% yellow. Proximity sensors enforce obstacle avoidance: if any ToF reading falls below 0.5 m, the state enters CRITICAL and an immediate E-STOP is invoked; a moderate alert state (NEAR) is triggered below 2.0 m, causing automated speed reduction. If the robot tilts more than 15° (via IMU roll/pitch), a TIPPING_RISK warning appears, suggesting a potential tip-over. Excessive motor current (>0.5 A surge) raises an OVERLOAD warning. Network quality is monitored: packet loss >10% or latency >100 ms result in “NETWORK ISSUES” or yellow badges. These thresholds are enforced by both the firmware and the UI: when crossed, they update /control_state flags and trigger visual alerts.

Failure Handling: The system handles faults in communication and sensors gracefully. If the Firebase/WebSocket link drops (next.js .info/connected = false), the dashboard shows an offline indicator and continues displaying cached data with a “stale” label. For Wi-Fi loss on the robot side, the ESP32 continuously retries to reconnect; the Pi, seeing no new data, eventually flags esp32_connected=false but continues autonomous navigation with the last-known inputs. Sensor faults trigger fallbacks: if a ToF sensor fails, the system ignores it and computes min_distance from the others. If GPS fix is lost, navigation falls back to wheel odometry and a “GPS_LOST” warning is shown. An IMU failure yields degraded attitude estimates (and a warning). If an encoder fails, the controller disregards feedback and uses commanded velocity only, raising a MOTOR_FAULT flag.

Table 7 representative safety thresholds and their system responses lists representative safety thresholds and their system responses. Each trigger is monitored on-board or in the UI, ensuring that hazards (low battery, obstacle too close, loss of connectivity, etc.) prompt immediate, defined actions. This layered approach – combining hardware E-STOPs, software checks, and UI alerts – provides robust protection against both physical and cyber faults.

Threshold	Detection Method	Response	UI Feedback
Battery < 15%	Dashboard (DB battery node)	*return-to-base* warning; disable non-essential motions	“Battery Critical” red badge; progress bar red
Battery drop > 10%	Dashboard comparison	“POWER FAULT” warning	Battery card warning icon
Obstacle < 0.5 m	ESP32 ToF readings	Set obstacle_critical=true; trigger E-STOP	“Obstacle Critical” red badge
Obstacle < 2.0 m	ESP32 ToF	Enter NEAR state; auto slow down	Obstacle card warns proximity
IMU tilt > 15°	ESP32 IMU	Warn tipping risk; recommend slow or stop	“Tipping Risk” warning badge
Motor current spike > 0.5 A	ESP32 current sensors	Warn overload; possibly cut drive	Motor card overload warning

Packet loss > 10%	Pi network stats	"NETWORK ISSUES" warning	Network card warning badge
Latency > 100 ms	Pi network stats	Yellow connection badge	Network card latency warning
ESP32 offline (>2 s)	Pi (serial timeout)	esp32_connected=false; degraded mode	"MCU OFFLINE" critical alert
Wi-Fi offline	ESP32 Wi-Fi monitor	ESP32 retries; robot continues on previous inputs	"Connection Lost" warning
Firebase offline	Dashboard (websocket)	Show offline mode; disable new commands	Offline status indicator

Table 7 representative safety thresholds and their system responses

The combination of heartbeat monitoring, hardware E-STOP, and threshold checks ensures that any anomaly – be it mechanical, electrical, or network-related – results in a predictable and safe system response. The dashboard prominently displays all critical alerts (red badges for highest priority) so that the operator has immediate visibility into the robot's safety status.

3.8.3.2 Fault Responses and Failure Modes

Building on the safety mechanisms, this section outlines specific failure modes and automated responses. The system distinguishes between various fault classes:

- **ESP32 Disconnection:** If the Raspberry Pi detects no serial data for >2 s, it deems the ESP32 offline. In response, esp32_connected is set to false and the robot enters a "degraded" mode (autonomous functions like SLAM and planning may continue, but no new motor commands can be issued). The dashboard immediately shows a "MCU OFFLINE" critical alert (red). This lets the operator know that although the robot may coast safely, no further motion is possible until the low-level controller reconnects.
- **Network Connection Loss:** If the dashboard's heartbeat monitor sees last_heartbeat_ms > 1000 ms[68], it triggers a "CONNECTION LOST" warning (yellow) and disables operator controls. The UI displays a prominent badge indicating the lost link. Once connectivity is restored, normal operation resumes.
- **Battery Critical (<15%):** Detected by continuous checking of battery_percent. Falling below 15% immediately prompts a "RETURN TO BASE" critical warning on the dashboard. The UI shows a red color-coded alert and a red battery progress bar. The robot's navigation is programmed (via a mode or waypoint) to guide it back to the charging station automatically.
- **Emergency Stop Activated:** If the ESP32 interrupts estop=true, the robot goes into an E-STOP state. The dashboard shows the highest-priority E-STOP badge, all drive commands are disabled, and the robot immediately halts motion. The operator must explicitly clear the E-STOP and then issue a RESUME command to continue.

- **Robot Error State:** If the Raspberry Pi's ROS2 state machine enters an ERROR state (e.g., due to a planning failure), it sets robot_state = ERROR. The response is to stop all motion and display an "ERROR" badge on the dashboard. The status card shows the error state and logs any relevant messages.
- **High Packet Loss (>10%):** If the Pi measures packet loss above 10%, the dashboard flags "NETWORK ISSUES" (yellow). This warns the operator that control commands may be delayed or lost, suggesting caution.
- **Battery Voltage Drop:** If the dashboard detects a sudden voltage drop >10% between readings, it triggers a "POWER FAULT" warning. This could indicate a failing battery or connection and prompts investigation. A warning icon appears on the battery status card[61].
- **Firebase Disconnection:** When the dashboard's WebSocket closes (.info/connected = false), the dashboard shows an offline indicator and continues to display the last-known data (clearly marked as "stale" with a timestamp). This ensures the operator is aware data is not fresh.
- **Obstacle Critical (<0.5 m):** If the ESP32's ToF sensors report min_distance < 0.5 m, the flag obstacle_critical=true is set and an immediate E-STOP is triggered to prevent collision. The dashboard shows an "Obstacle Critical" red alert.
- **Motor Fault:** If the commanded linear velocity and actual linear velocity differ by more than 0.2 m/s for >1s, a MOTOR_FAULT is declared. The system may ignore new motion commands until reset, and a warning is logged. The operator sees a warning on the motion card and details in the log[81].
- **GPS Fix Lost:** If gps_data.fix goes false, navigation falls back to odometry and a "GPS_LOST" warning is shown. The operator is informed via the sensor status.
- **Tipping Risk:** If the IMU roll or pitch exceeds 15°, a TIPPING_RISK warning is raised, recommending the robot slow down or stabilize. A yellow warning appears in the sensor card. The system may optionally reduce speed automatically to regain stability.

Each of these conditions and responses is implemented in the robot's software (ESP32 firmware and ROS2 nodes) and/or in the dashboard logic. By formalizing these failure modes, the system ensures consistent, predictable behavior under faults, enhancing safety and maintainability.

3.8.4 Web Dashboard Design & Development

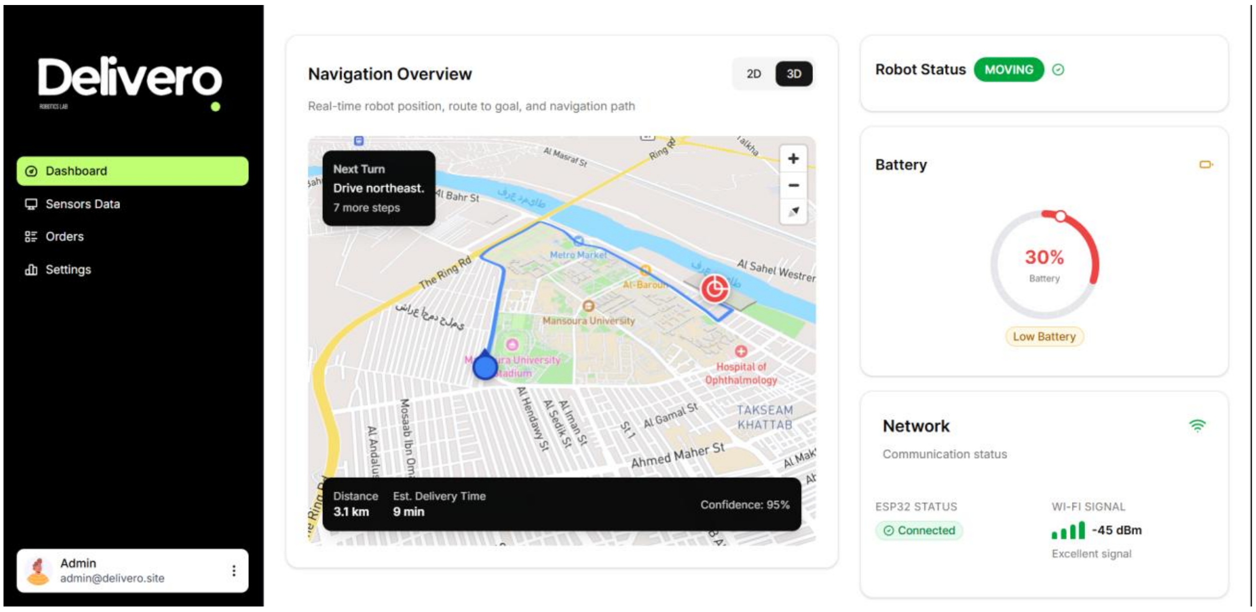
3.8.4.1 Framework Selection: Why Next.js

The web dashboard for the autonomous delivery robot is built using **Next.js** as shown in Figure. 3.41 , a modern React-based web application framework. Next.js was chosen for its hybrid rendering capabilities, built-in routing, developer productivity, and seamless compatibility with real-time Firebase data via client-side JavaScript.

The core motivations behind choosing this stack include:

- **React Foundation**: Enables declarative UI, reusable components, and rapid updates based on robot telemetry state.
- **Client-Side Rendering (CSR)** with optional server-side rendering (SSR): Allows the dashboard to function as a lightweight single-page application while enabling future scalability.
- **Firebase Integration**: Firebase's JavaScript SDK integrates natively into Next.js, offering real-time WebSocket listeners and seamless HTTPS calls for control commands.
- **Modular Development**: Next.js supports file-based routing and component-based design, enabling structured and maintainable code.

This architecture allows the robot's state and sensor data to be reflected in near real-time on the dashboard interface, supporting both observation and human-in-the-loop control.



Navigation Overview

Real-time robot position, route to goal, and navigation path

Next Turn
Drive northeast.
7 more steps

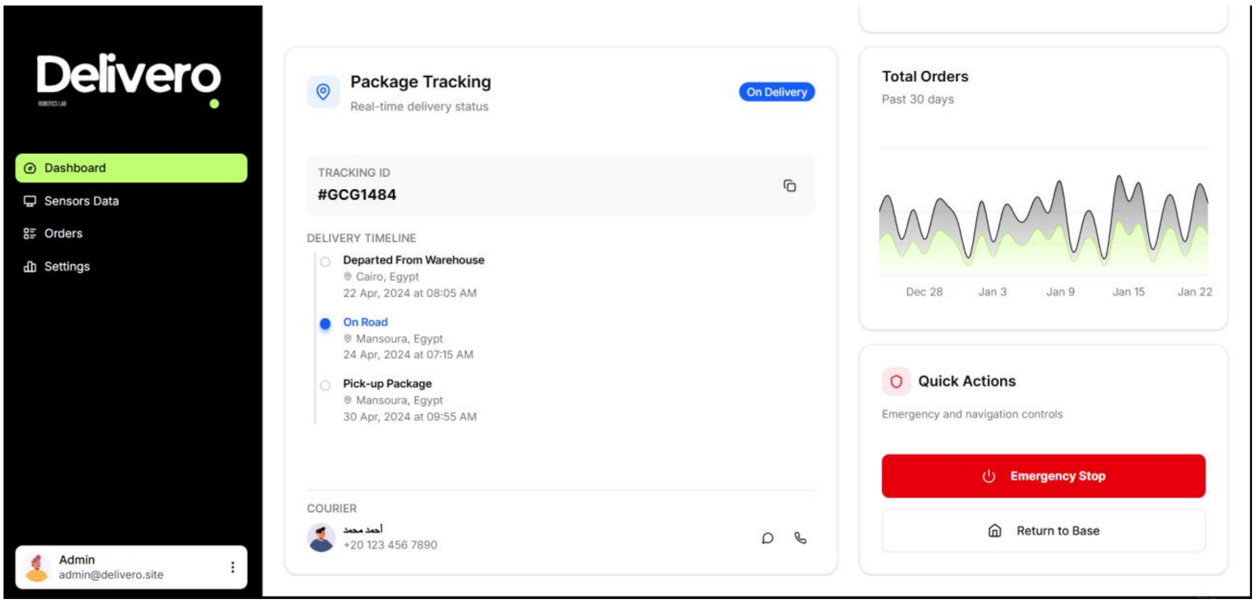
Distance 3.1 km | Est. Delivery Time 9 min | Confidence: 95%

Robot Status MOVING

Battery 30%

ESP32 STATUS: Connected

WI-FI SIGNAL: -45 dBm
Excellent signal



Package Tracking
Real-time delivery status
On Delivery

TRACKING ID
#GCG1484

DELIVERY TIMELINE

Departed From Warehouse
Cairo, Egypt
22 Apr, 2024 at 08:05 AM

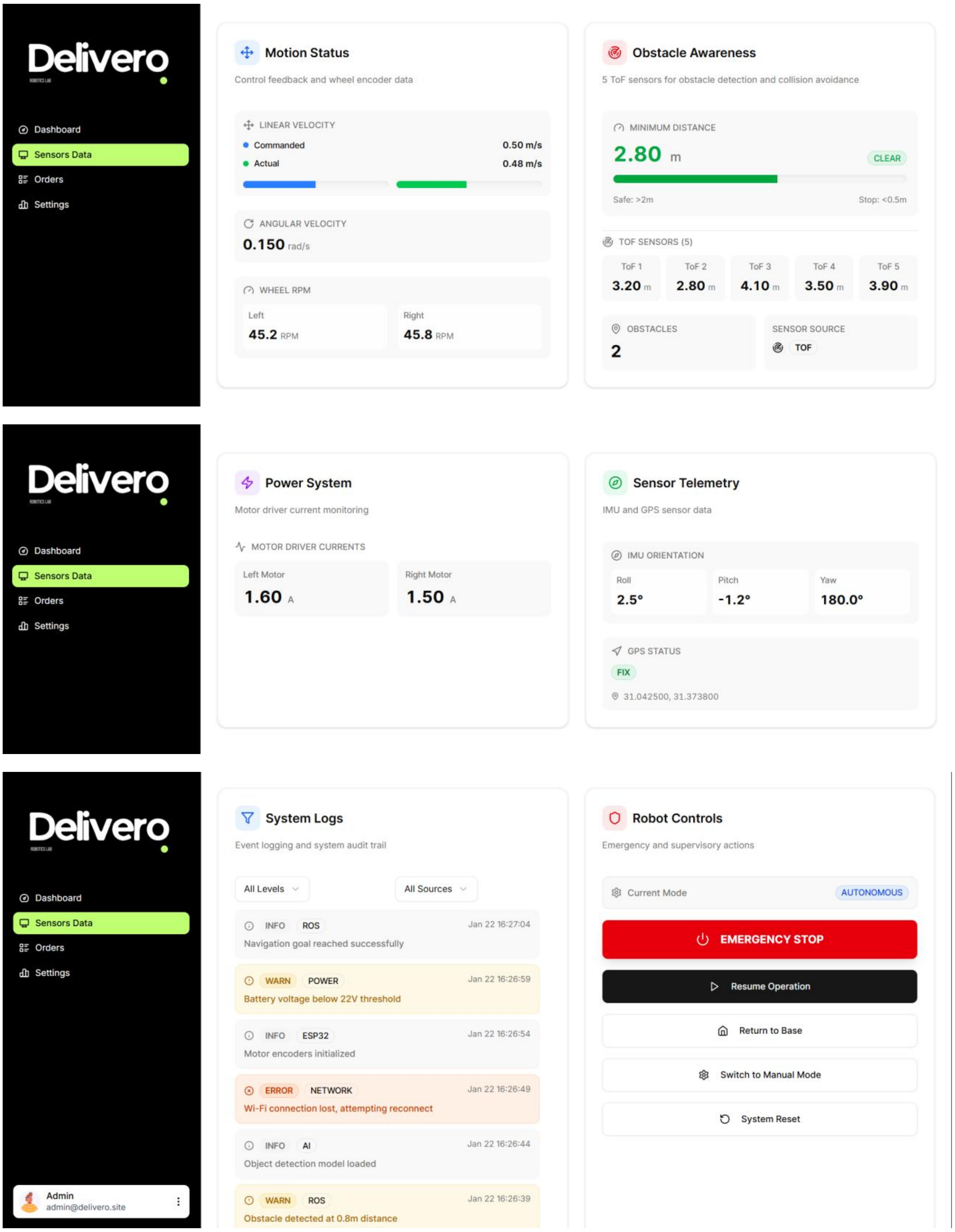
On Road
Mansoura, Egypt
24 Apr, 2024 at 07:15 AM

Pick-up Package
Mansoura, Egypt
30 Apr, 2024 at 09:55 AM

COURIER
احمد محمد
+20 123 456 7890

Total Orders
Past 30 days

Quick Actions
Emergency and navigation controls



Motion Status

Control feedback and wheel encoder data

LINEAR VELOCITY

Commanded 0.50 m/s

Actual 0.48 m/s

ANGULAR VELOCITY

0.150 rad/s

WHEEL RPM

Left 45.2 RPM

Right 45.8 RPM

Obstacle Awareness

5 ToF sensors for obstacle detection and collision avoidance

MINIMUM DISTANCE

2.80 m CLEAR

Safe: >2m Stop: <0.5m

TOF SENSORS (5)

ToF 1	ToF 2	ToF 3	ToF 4	ToF 5
3.20 m	2.80 m	4.10 m	3.50 m	3.90 m

OBSTACLES 2

SENSOR SOURCE TOF

Power System

Motor driver current monitoring

MOTOR DRIVER CURRENTS

Left Motor 1.60 A

Right Motor 1.50 A

Sensor Telemetry

IMU and GPS sensor data

IMU ORIENTATION

Roll 2.5°

Pitch -1.2°

Yaw 180.0°

GPS STATUS

FIX

31.042500, 31.373800

System Logs

Event logging and system audit trail

All Levels All Sources

```
INFO  ROS      Jan 22 16:27:04
Navigation goal reached successfully
WARN  POWER    Jan 22 16:26:59
Battery voltage below 22V threshold
INFO  ESP32    Jan 22 16:26:54
Motor encoders initialized
ERROR NETWORK  Jan 22 16:26:49
Wi-Fi connection lost, attempting reconnect
INFO  AI       Jan 22 16:26:44
Object detection model loaded
WARN  ROS      Jan 22 16:26:39
Obstacle detected at 0.8m distance
```

Robot Controls

Emergency and supervisory actions

Current Mode AUTONOMOUS

Figure. 3.41 the main dashboard interface

3.8.4.2 Developer Workflow

The frontend was developed using a modern full-stack web development workflow shown in Figure. 3.42, optimized for real-time IoT supervision:

- **Framework and Tools:**
  - **Next.js** for the application framework
  - **TypeScript** for type safety and code clarity
  - **Tailwind CSS** for utility-first responsive design
  - **Firebase SDK** for real-time data binding and cloud communication
- **Codebase Structure:**
  - /components/: UI cards, maps, control panels
  - /hooks/: Custom logic for Firebase listeners and data subscriptions (e.g., useRobotStatus, useAIPerceptionData)
  - /app/dashboard/: Page routing and dashboard layout
  - /lib/: Helper functions for data parsing and state logic
- **Firebase Integration:**
  - Firebase Realtime Database is configured in **region: asia-southeast1**
  - Uses **WebSocket subscriptions** for telemetry
  - Uses **HTTPS PUT** to push control commands (e.g., E-STOP, Return Home)

This modular and responsive development structure enables rapid iteration and extensibility. Critical states are visually encoded with dynamic UI elements and alert logic, empowering operators to act immediately in emergency scenarios.

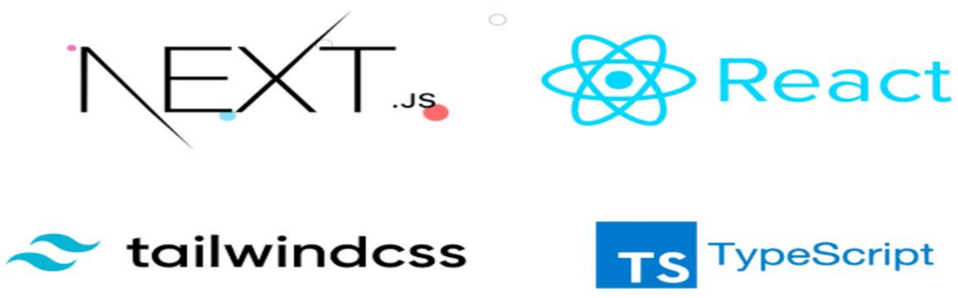


Figure. 3.42 web technologies used in dashboard design

4 Experiments and Results

This chapter presents the experimental validation of the Autonomous Delivery Robot (ADR) system, covering mechanical simulations, perception system evaluation, embedded deployment, and real-world testing. Both simulation-based and hardware-based experiments are conducted to assess structural integrity, perception performance, computational efficiency, and overall system robustness.

4.1 Experimental Setup

The experimental setup integrates mechanical, electrical, and software subsystems of the ADR platform. Mechanical validation was performed using finite element analysis (FEA) to evaluate stress, deformation, and load-bearing capability of the chassis, wheel assembly, and motor base. Perception experiments were conducted on the onboard compute platform using a monocular RGB camera and a Raspberry Pi-based processing unit.

Test Platforms:

- Mechanical CAD models developed in SolidWorks
- Structural simulations using SolidWorks Simulation
- Embedded compute: Raspberry Pi (Pi 4 / Pi 5)
- Camera: USB RGB camera (640×480)
- Software stack: Python, OpenCV, YOLO11n, Depth Anything V2

4.2 Finite Element Analysis (FEA)

To evaluate the structural integrity and mechanical reliability of the proposed Autonomous Delivery Robot (ADR), a comprehensive **finite element analysis (FEA)** was conducted as part of the experimental validation phase. Simulation-based testing was selected to verify the mechanical design prior to full-scale real-world deployment, reduce prototyping risks, and identify potential structural weaknesses early in the design cycle.

All simulations were performed using **SOLIDWORKS Simulation** under static loading conditions, which are suitable for preliminary validation of mobile robotic platforms operating at low to moderate speeds.

4.2.1 Simulation Methodology and Assumptions

Static structural analysis was employed to evaluate stress, deformation, strain, and factor of safety under representative operational loads. The following assumptions were applied across all simulation studies:

- Loads were applied gradually, neglecting dynamic and inertial effects.
- Material behavior was assumed to be **linear elastic and isotropic**.
- Deformations were considered small.

- Contacts between components were defined as **bonded**, representing welded or rigid mechanical joints.
- Gravity was included where applicable to simulate real operating conditions.
- Thermal effects were enabled with a reference temperature of **298 K**, without temperature gradients.
- Frictional contact and large displacement effects were neglected.

These assumptions are appropriate for early-stage structural verification of a wheeled mobile robot intended for campus and sidewalk delivery tasks.

4.2.2 Material Properties

Two primary materials were used in the mechanical structure and evaluated in the simulations:

1. **Aluminum Alloy 1060**

Used for the robot chassis and motor base due to its lightweight properties:

- Density: 2700 kg/m³
- Elastic Modulus: 69 GPa
- Poisson's Ratio: 0.33
- Yield Strength: 27.6 MPa
- Tensile Strength: 68.9 MPa

2. **Plain Carbon Steel**

Used for the driving wheels to withstand torque and ground contact stresses:

- Density: 7800 kg/m³
- Elastic Modulus: 210 GPa
- Poisson's Ratio: 0.28
- Yield Strength: 220 MPa
- Tensile Strength: 399 MPa

4.2.3 Simulated Components and Boundary Conditions

Three critical mechanical subsystems were analyzed:

3. **Complete Four-Wheel Assembly**
4. **Motor Base Structure**
5. **Individual Driving Wheel**

Boundary conditions were defined using fixed geometry constraints at mounting points, bolt interfaces, and chassis connections to accurately represent real-world attachment conditions.

Applied loads included:

- Gravity (9.81 m/s^2)
- Normal forces representing payload and component weight
- Applied torque on the wheel to simulate driving conditions

4.2.4 Mesh Generation

High-quality meshing was employed to ensure reliable numerical results:

- **Four-Wheel Assembly:**
  Solid mesh with 40,793 nodes and 20,040 elements.
  Over 97% of elements exhibited acceptable aspect ratios.
- **Motor Base:**
  Shell mesh using mid-surface extraction with 25,362 nodes and 12,103 elements.
  Curvature-based meshing was applied near mounting holes and stress-concentrated regions.
- **Wheel:**
  Solid curvature-controlled mesh with 6,903 nodes and 4,188 elements.
  More than 95% of elements had aspect ratios below 3, indicating excellent mesh quality.

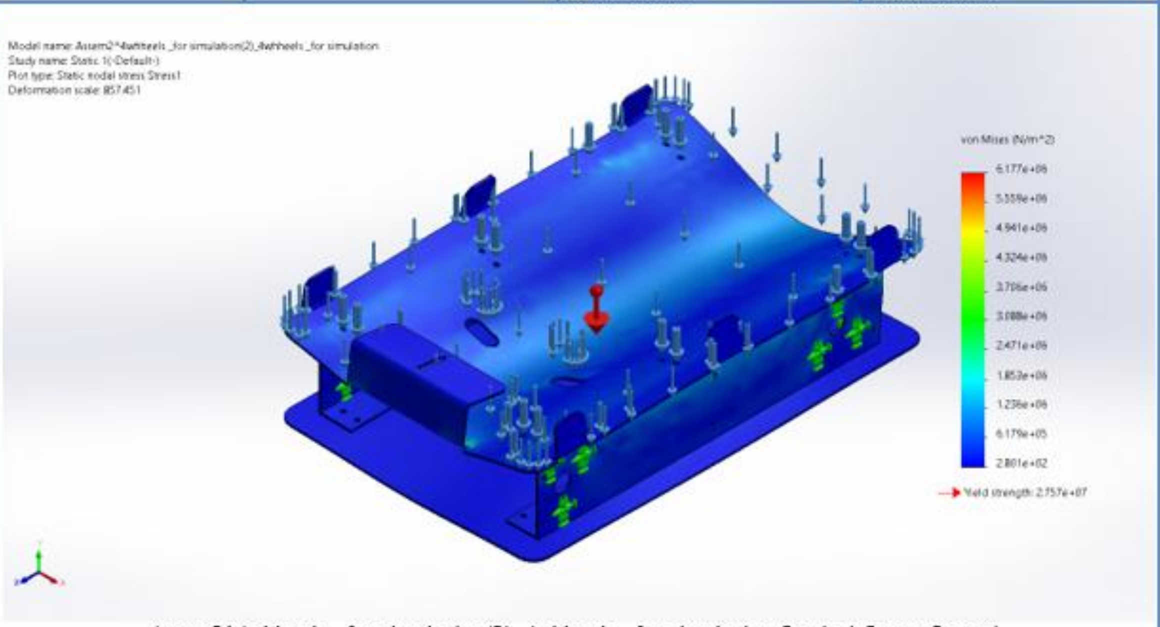
4.2.5 Simulation Results and Discussion

6. **A. Four-Wheel Assembly Results**

The full assembly shown in Figure. 4.1 was evaluated under gravitational and payload loading.

- Maximum von Mises stress: **6.18 MPa**
- Maximum displacement: **0.064 mm**
- Maximum equivalent strain: **4.43×10^{-5}**
- Minimum factor of safety: **4.46**

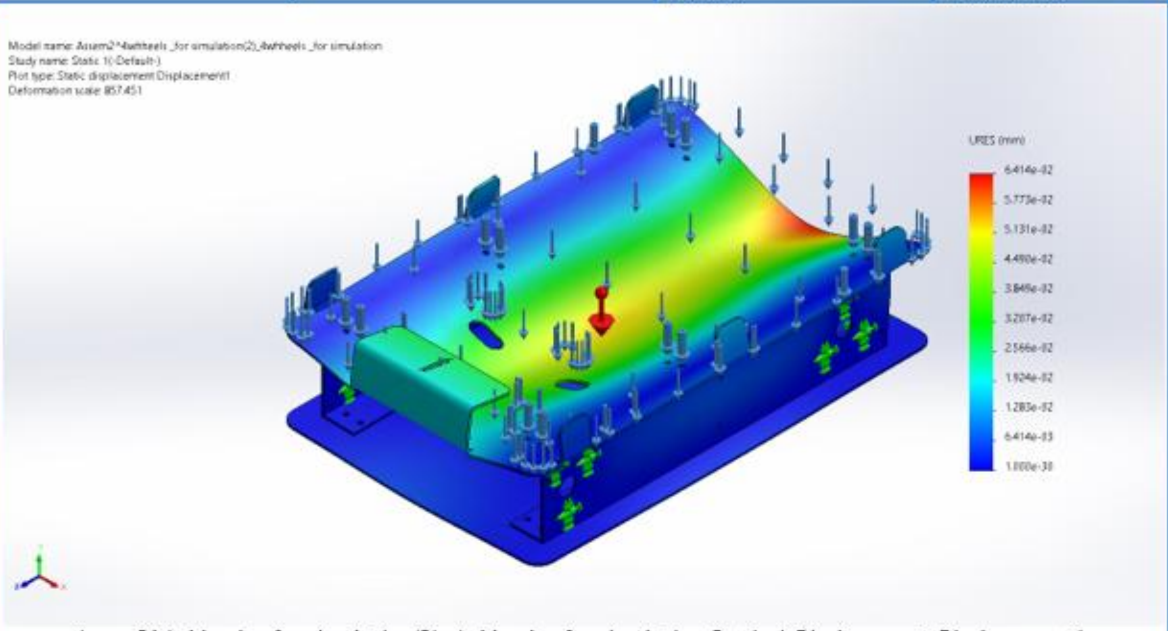
Name	Type	Min	Max
Stress1	VON: von Mises Stress	2.801e+02N/m^2 Node: 22011	6.177e+06N/m^2 Node: 38326


Model name: Assem2^4whheels _for simulation(2)_4whheels _for simulation
Study name: Static 1(-Default-)
Plot type: Static nodal stress Stress1
Deformation scale: 857.451
von Mises (N/m^2)
6.177e+06
5.559e+06
4.941e+06
4.324e+06
3.706e+06
3.088e+06
2.471e+06
1.853e+06
1.236e+06
6.179e+05
2.801e+02
Yield strength: 2.757e+07

Assem2^4whheels _for simulation(2)_4whheels _for simulation-Static 1-Stress-Stress1

Figure. 4.1 The von Mises stress distribution across the full four-wheel assembly

Name	Type	Min	Max
Displacement1	URES: Resultant Displacement	0.000e+00mm Node: 7	6.414e-02mm Node: 35167


Model name: Assem2^4whheels _for simulation(2)_4whheels _for simulation
Study name: Static 1(-Default-)
Plot type: Static displacement Displacement1
Deformation scale: 857.451
URES (mm)
6.414e-02
5.773e-02
5.131e-02
4.490e-02
3.849e-02
3.207e-02
2.566e-02
1.924e-02
1.283e-02
6.414e-03
1.000e-30

Assem2^4whheels _for simulation(2)_4whheels _for simulation-Static 1-Displacement-Displacement1

Figure. 4.2 illustrates the resultant displacement contour.

The results shown in Figure. **4.2** confirm that stresses remain well below material yield limits, indicating a structurally safe design.

7. B. Motor Base Results

The motor base was identified as the most structurally critical component due to load concentration at motor mounts.

- Maximum von Mises stress: **10.75 MPa**
- Maximum displacement: **1.15 mm**
- Maximum equivalent strain: **9.67×10^{-4}**
- Minimum factor of safety: **2.56**

Although stresses remain within safe limits, this component governs the structural safety of the robot and represents the primary candidate for future reinforcement.

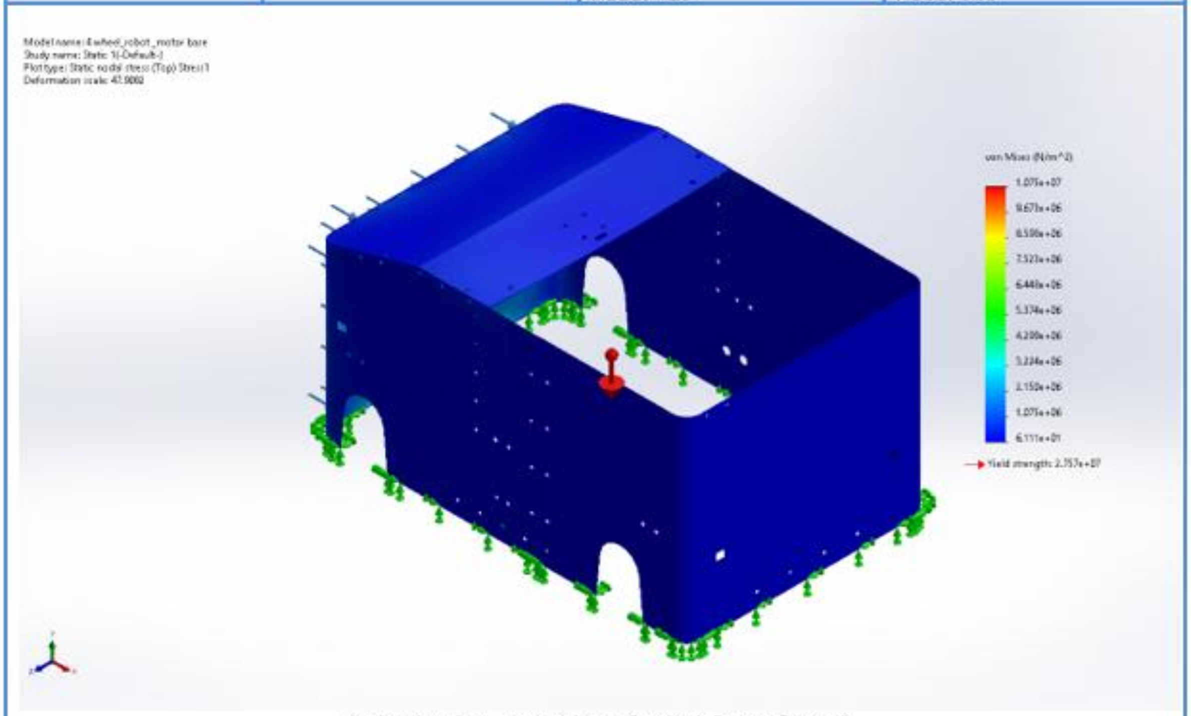
8. C. Wheel Structural Results

The driving wheel was analyzed under an applied torque of **5 N·m** to simulate traction forces.

- Maximum von Mises stress: **16.9 MPa**
- Maximum displacement: **0.0064 mm**
- Minimum factor of safety: **13**

Figure. 4.3shows the stress concentration near the wheel hub, while Figure. 4.4 presents the displacement contour.

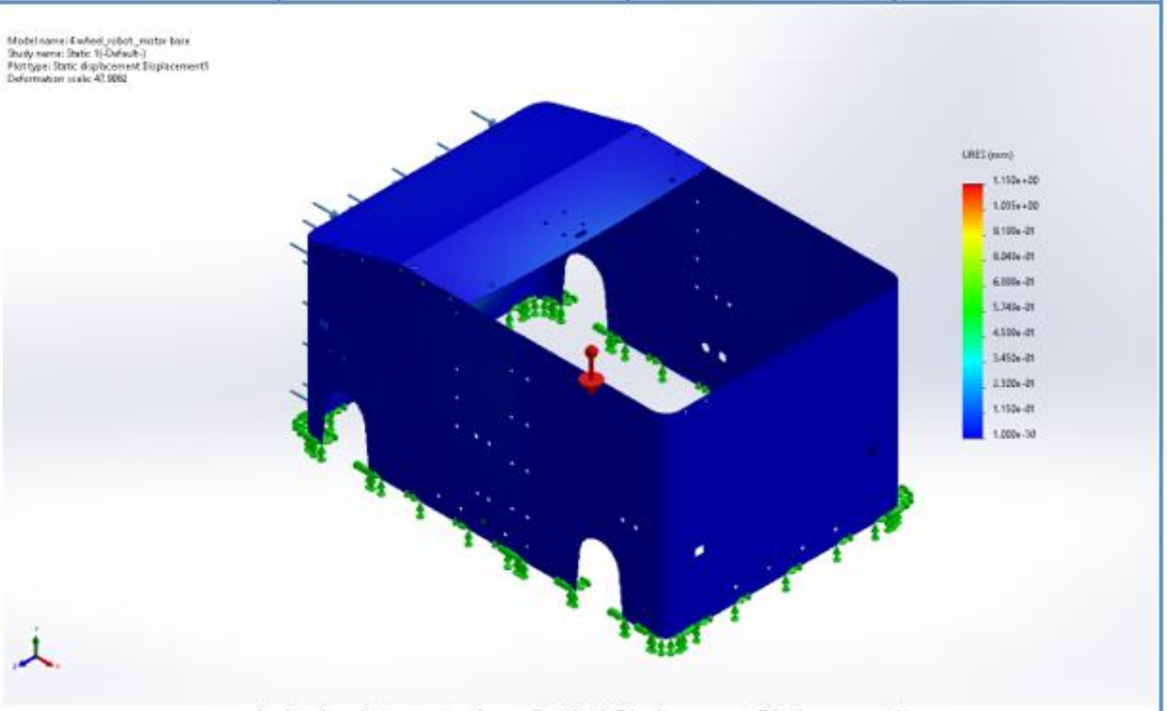
Name	Type	Min	Max
Stress1	VON: von Mises Stress	6.111e+01N/m^2 Node: 126	1.075e+07N/m^2 Node: 705


Model name: 4 wheel_robot _motor base
Study name: Static 1(-Default-)
Plot type: Static nodal stress (Top) Stress1
Deformation scale: 47.9002

4 wheel_robot _motor base-Static 1-Stress-Stress1

Figure. 4.3 Stress distribution in the driving wheel under applied torque

Name	Type	Min	Max
Displacement1	URES: Resultant Displacement	0.000e+00mm Node: 2	1.150e+00mm Node: 709


Model name: 4 wheel_robot _motor base
Study name: Static 1(-Default-)
Plot type: Static displacement Displacement1
Deformation scale: 47.9002

4 wheel_robot _motor base-Static 1-Displacement-Displacement1

Figure. 4.4 Resultant displacement of the driving wheel

The wheel demonstrates excellent structural robustness with a very high safety margin.

4.2.6 Comparative Evaluation

Component	Max Stress (MPa)	Max Displacement (mm)	Min FOS
Four-Wheel Assembly	6.18	0.064	4.46
Motor Base	10.75	1.15	2.56
Wheel	16.9	0.006	13

Table 8 FEA Comparative Evaluation

The motor base is identified as the most critical structural component and dictates the overall mechanical safety margin of the system.

4.2.7 Simulation Conclusions

The simulation results demonstrate that:

- All mechanical components operate safely below material yield limits.
- No plastic deformation is expected under nominal operating conditions.
- The structural design is suitable for prototype fabrication and real-world testing.

Recommended future improvements include:

- Reinforcing the motor base with stiffening ribs.
- Increasing thickness near motor mounting holes.
- Conducting dynamic and fatigue analyses for long-term durability.
- Evaluating full payload and acceleration load cases.

4.3 Real-World Testing

Real-world experiments were conducted in indoor and semi-controlled environments to validate perception, decision-making, and system responsiveness. The robot was tested under varying lighting conditions and object densities to simulate real delivery scenarios.

Test scenarios included:

- Static obstacles (chairs, boxes)
- Dynamic obstacles (walking persons)
- Mixed object environments

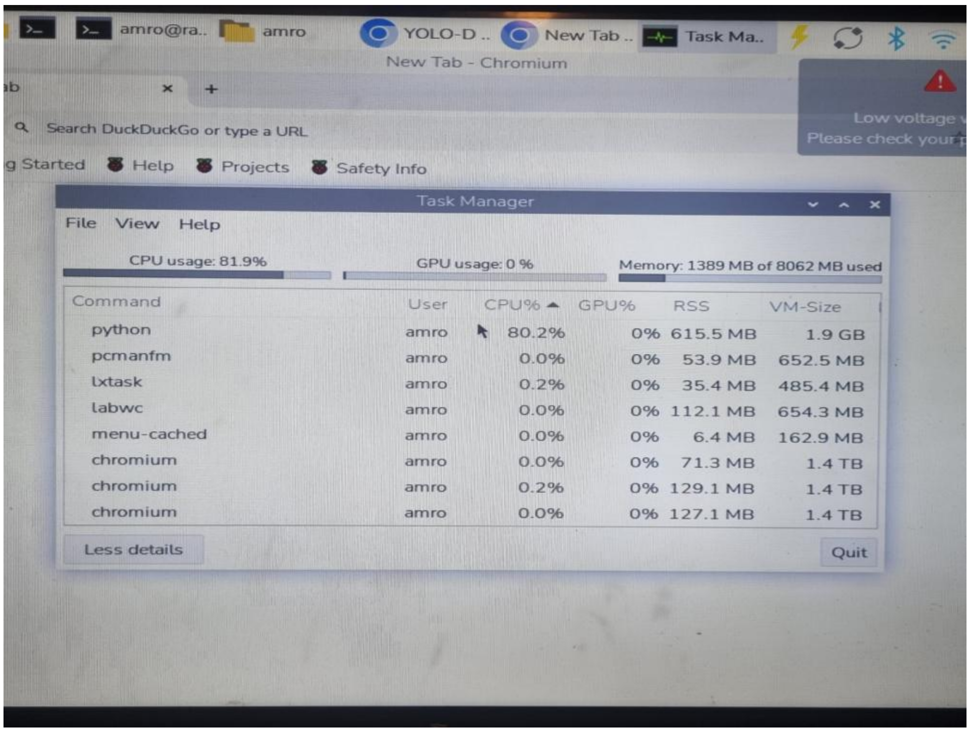
Observed behavior confirmed stable perception output and consistent action generation without system crashes or frame drops.

4.4 Performance Metrics

The perception system performance was evaluated using frame rate, latency, and memory usage as key metrics shown in Figure. 4.5.

4.4.1 YOLO3D Pipeline Performance

The YOLO3D pipeline combines YOLO11n for object detection with Depth Anything V2 for monocular depth estimation. Multiple optimization phases were experimentally evaluated.



Task Manager

CPU usage: 81.9% GPU usage: 0 % Memory: 1389 MB of 8062 MB used

Command	User	CPU%	GPU%	RSS	VM-Size
python	amro	80.2%	0%	615.5 MB	1.9 GB
pcmanfm	amro	0.0%	0%	53.9 MB	652.5 MB
lxtask	amro	0.2%	0%	35.4 MB	485.4 MB
labwc	amro	0.0%	0%	112.1 MB	654.3 MB
menu-cached	amro	0.0%	0%	6.4 MB	162.9 MB
chromium	amro	0.0%	0%	71.3 MB	1.4 TB
chromium	amro	0.2%	0%	129.1 MB	1.4 TB
chromium	amro	0.0%	0%	127.1 MB	1.4 TB

Figure. 4.5 YOLO3D performance on Raspberry Pi.

Key observations:

- Frame rate improved from 3–8 FPS to 8–15 FPS after optimization
- Frame latency reduced by approximately 40%
- Memory usage reduced by ~200 MB through resolution reduction

4.4.2 Data Flow and Processing Latency

The end-to-end data flow includes frame acquisition, preprocessing, detection, depth estimation, fusion, and action policy execution.

The rule-based action policy introduces negligible computational overhead (<1 ms per frame), making it suitable for real-time robotics applications.

4.5 Manufacturability Evaluation

The mechanical design emphasizes manufacturability using widely available fabrication methods such as CNC machining, laser cutting, and standard fasteners. Modular sub-assemblies (wheel modules, motor mounts, electronics enclosure) simplify assembly and maintenance.

Design considerations include:

- Standardized fastener sizes
- Minimal custom parts
- Easy disassembly for repair

4.6 Safety and Risk Assessment

Safety analysis focused on both mechanical and operational risks.

Identified risks:

- Structural failure under overload
- Collision with pedestrians
- Power system faults

Mitigation measures:

- Conservative factor of safety in mechanical design
- Distance-based STOP and SLOW_DOWN actions
- Emergency stop logic and software debouncing

4.7 Ethical and Social Considerations

The deployment of autonomous delivery robots introduces ethical and social considerations related to safety, accessibility, and workforce impact.

Key considerations include:

- Ensuring pedestrian safety through conservative behavior
- Transparency and explainability of robot decisions
- Supporting last-mile delivery efficiency without compromising public spaces

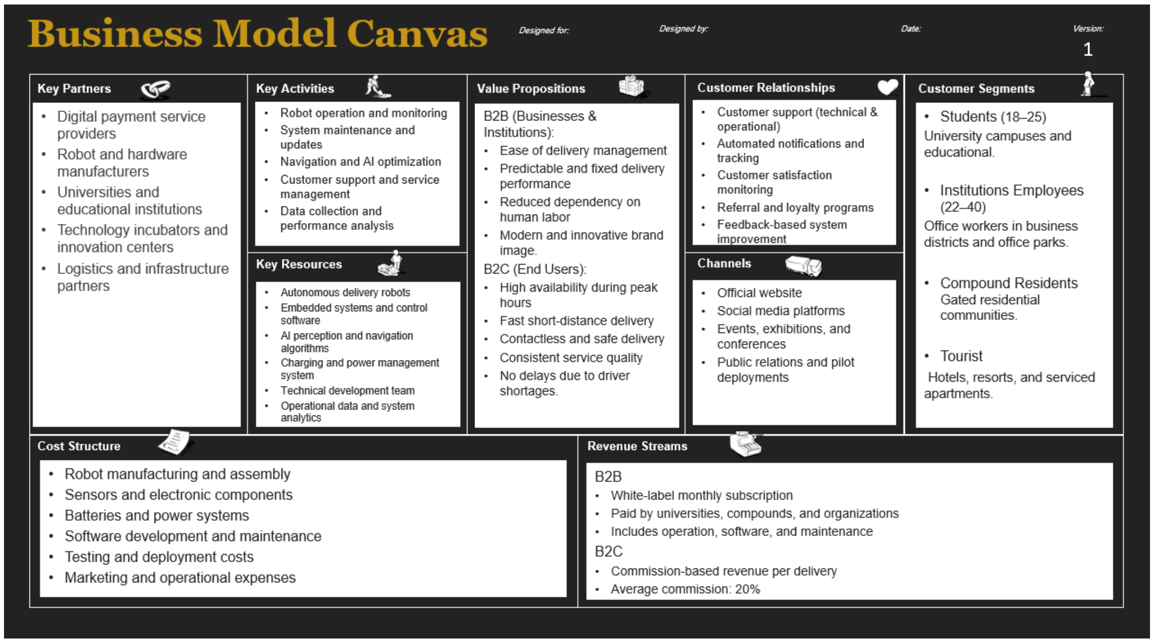
The ADR system prioritizes safety-first decision policies and interpretable rule-based logic to promote trust and responsible deployment.

The experimental results demonstrate that the ADR system satisfies mechanical safety requirements, achieves real-time perception on embedded hardware, and operates reliably in real-world environments. The combination of simulation-driven design and hardware validation confirms the feasibility of the proposed autonomous delivery platform.

5 Our Business Model

This chapter presents the proposed business model shown in Figure. 5.1 for the autonomous delivery robot system. The model is designed specifically for the Egyptian market, taking into consideration local economic conditions, delivery behaviors, and the suitability of robotic delivery in semi-controlled environments. The chapter explains the selected customer segments, revenue streams, key partners, and the value propositions offered to both businesses and end users. The Business Model Canvas is used as a structured framework to represent the overall business logic of the system.

5.1 Business Model



Business Model Canvas

Designed for: Designed by: Date: Version: 1

Key Partners
- Digital payment service providers
- Robot and hardware manufacturers
- Universities and educational institutions
- Technology incubators and innovation centers
- Logistics and infrastructure partners

Key Activities
- Robot operation and monitoring
- System maintenance and updates
- Navigation and AI optimization
- Customer support and service management
- Data collection and performance analysis

Key Resources
- Autonomous delivery robots
- Embedded systems and control software
- AI perception and navigation algorithms
- Charging and power management system
- Technical development team
- Operational data and system analytics

Value Propositions

B2B (Businesses & Institutions):
- Ease of delivery management
- Predictable and fixed delivery performance
- Reduced dependency on human labor
- Modern and innovative brand image.

B2C (End Users):
- High availability during peak hours
- Fast short-distance delivery
- Contactless and safe delivery
- Consistent service quality
- No delays due to driver shortages.

Customer Relationships
- Customer support (technical & operational)
- Automated notifications and tracking
- Customer satisfaction monitoring
- Referral and loyalty programs
- Feedback-based system improvement

Channels
- Official website
- Social media platforms
- Events, exhibitions, and conferences
- Public relations and pilot deployments

Customer Segments
- Students (18–25)
University campuses and educational.
- Institutions Employees (22–40)
Office workers in business districts and office parks.
- Compound Residents
Gated residential communities.
- Tourist
Hotels, resorts, and serviced apartments.

Cost Structure
- Robot manufacturing and assembly
- Sensors and electronic components
- Batteries and power systems
- Software development and maintenance
- Testing and deployment costs
- Marketing and operational expenses

Revenue Streams

B2B
- White-label monthly subscription
- Paid by universities, compounds, and organizations
- Includes operation, software, and maintenance

B2C
- Commission-based revenue per delivery
- Average commission: 20%

Figure. 5.1 The Business Model Canvas is illustrated

5.1.1 Customer Segments

The autonomous delivery robot system targets specific customer segments within the Egyptian market where robotic delivery can provide clear operational and economic advantages. These segments were selected based on delivery demand, environmental control, and user acceptance of smart technologies.

University students aged between 18 and 25 years represent a primary customer segment. This group frequently relies on delivery services, particularly for food and small parcels, and demonstrates a high level of familiarity with mobile applications and automated systems.

University campuses often impose restrictions on external delivery vehicles, making robotic delivery a practical and efficient solution.

Employees aged between 22 and 40 years form another key customer segment. This group includes office workers and professionals operating within business districts and office complexes. Their demand for delivery services is typically time-sensitive, especially during working hours. Robotic delivery offers reliable and predictable service while avoiding delays caused by traffic congestion.

Residents of gated residential compounds constitute an important customer segment due to security regulations that limit access for human delivery personnel. The controlled nature of these environments makes them highly suitable for autonomous delivery systems, offering residents a secure and organized delivery experience.

Tourists staying in hotels, resorts, and serviced apartments represent a secondary but high-value customer segment. This group values convenience and innovation and is more willing to adopt advanced delivery solutions as part of a premium service experience.

5.1.2 Revenue Streams

The revenue model of the autonomous delivery robot system is designed to ensure financial sustainability through diversified income streams. For B2B clients, the system operates under a white-label monthly subscription model. Institutional clients such as universities, residential compounds, and organizations pay a fixed monthly fee that covers system operation, software services, and maintenance.

For B2C users, revenue is generated through a commission-based model applied to each completed delivery. A fixed percentage, estimated at approximately 20%, is deducted from the service fee. This model allows revenue to scale proportionally with increased system usage.

5.1.3 Key Partners

Key partners are essential to the successful deployment and operation of the autonomous delivery robot system. Payment service providers enable secure and reliable financial transactions. Robot and hardware manufacturers support system production and scalability. Universities and educational institutions serve as both deployment environments and development partners, while technology incubators and innovation centers provide technical and business support.

5.2 Value Propositions

The value proposition of the proposed autonomous food delivery robot defines the unique advantages that the system offers to customers compared to conventional human-based delivery services. The robot targets controlled environments such as university campuses, residential compounds, and hotels, where it can operate safely and efficiently. The value proposition is based on delivering fast, reliable, and contactless food delivery while reducing operational costs and enhancing customer convenience.

The value proposition is represented using a **Value Proposition Canvas shown in** Figure. 5.2 , which links customer needs (jobs, pains, and gains) with the product's features (products & services, pain relievers, and gain creators).

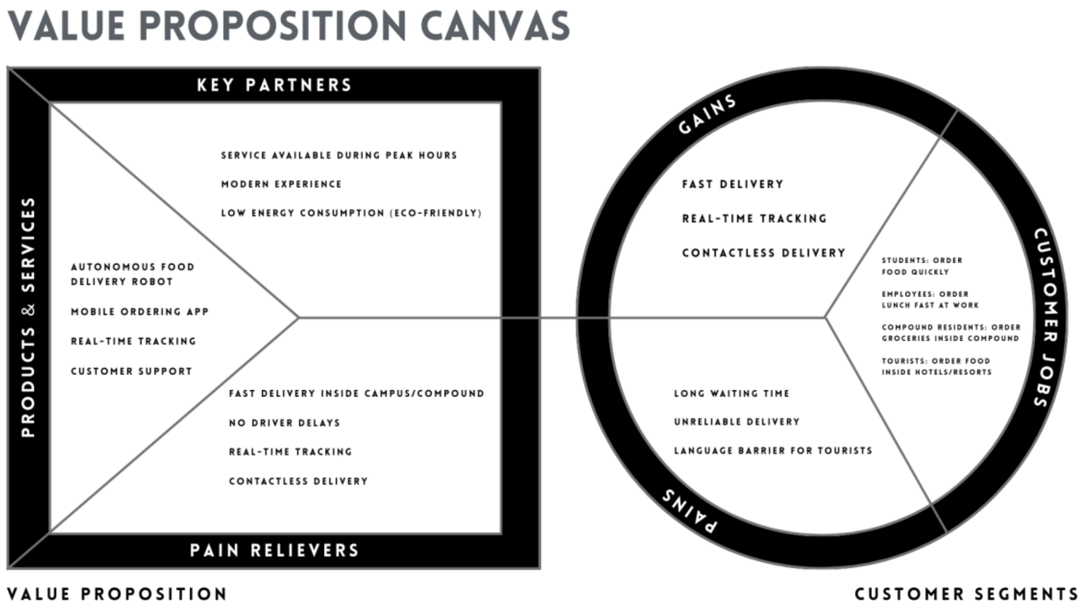


Figure. 5.2 Value Proposition Canvas illustrated

5.2.1 Customer Jobs

Customer jobs refer to the tasks that customers aim to accomplish when using the proposed system. The autonomous food delivery robot targets four primary customer segments: students, employees, compound residents, and tourists. Each segment requires a reliable and efficient method to order and receive food or groceries with minimal waiting time and effort.

Key customer jobs include:

- **Students:** Place and receive food orders quickly between lectures and campus activities.
- **Employees:** Order lunch or snacks during working hours without disrupting work schedules.
- **Compound residents:** Order groceries and daily necessities within the residential compound.
- **Tourists:** Order food in hotels or resorts without facing language barriers or navigation difficulties.

5.2.2 Customer Pains

Customer pains represent the negative experiences and challenges associated with traditional delivery services. These pains reduce customer satisfaction and increase the likelihood of order cancellations or complaints.

Primary customer pains include:

- **Long waiting time:** Delays due to traffic congestion, driver availability, or inefficient routes.
- **Unreliable delivery:** Inconsistent delivery times and potential order mishandling.
- **Language barrier (tourists):** Difficulty communicating order details, leading to errors and delays.

5.2.3 Customer Gains

Customer gains describe the benefits that customers expect from the system. The proposed solution aims to provide measurable and perceivable advantages that enhance the overall user experience.

Primary customer gains include:

- **Fast delivery:** Reduced delivery time within the controlled environment.
- **Real-time tracking:** Live updates on order status and estimated arrival time.
- **Contactless delivery:** Safe and convenient delivery without physical contact.

5.2.4 Products & Services

The proposed solution consists of integrated products and services that enable autonomous food delivery.

The main products and services are:

- **Autonomous food delivery robot:** A mobile robot capable of navigating controlled environments and delivering orders to customers.
- **Mobile ordering application:** A user-friendly interface for placing orders, tracking deliveries, and contacting support.
- **Real-time tracking system:** Provides live location updates and estimated arrival times.
- **Customer support:** A communication channel for handling inquiries, complaints, and special requests.

5.2.5 Pain Relievers

Pain relievers are features of the system that address the identified customer pains directly. The proposed system reduces customer pain by improving delivery speed, reliability, and transparency.

Main pain relievers include:

- **Fast delivery inside campus/compound:** Shorter routes and predictable navigation reduce waiting time.
- **No driver delays:** Autonomous operation eliminates dependency on human drivers.
- **Real-time tracking:** Enhances transparency and reduces uncertainty.
- **Contactless delivery:** Ensures safe and hygienic order handover.

5.2.6 Gain Creators

Gain creators are the features that generate additional value beyond addressing pains. These features aim to increase customer satisfaction and differentiate the service from traditional delivery methods.

Primary gain creators include:

- **Service available during peak hours:** The autonomous system can handle high demand without human limitations.
- **Modern experience:** Provides a novel and advanced delivery method, enhancing user perception.

- **Low energy consumption (eco-friendly):** Supports sustainability and reduces operational costs.

The proposed autonomous food delivery robot offers a comprehensive value proposition by addressing customer needs, reducing pain points, and delivering meaningful gains. The system enhances delivery speed, reliability, and customer convenience, making it a suitable solution for controlled environments such as campuses, compounds, and hotels.

5.3 SWOT Analysis

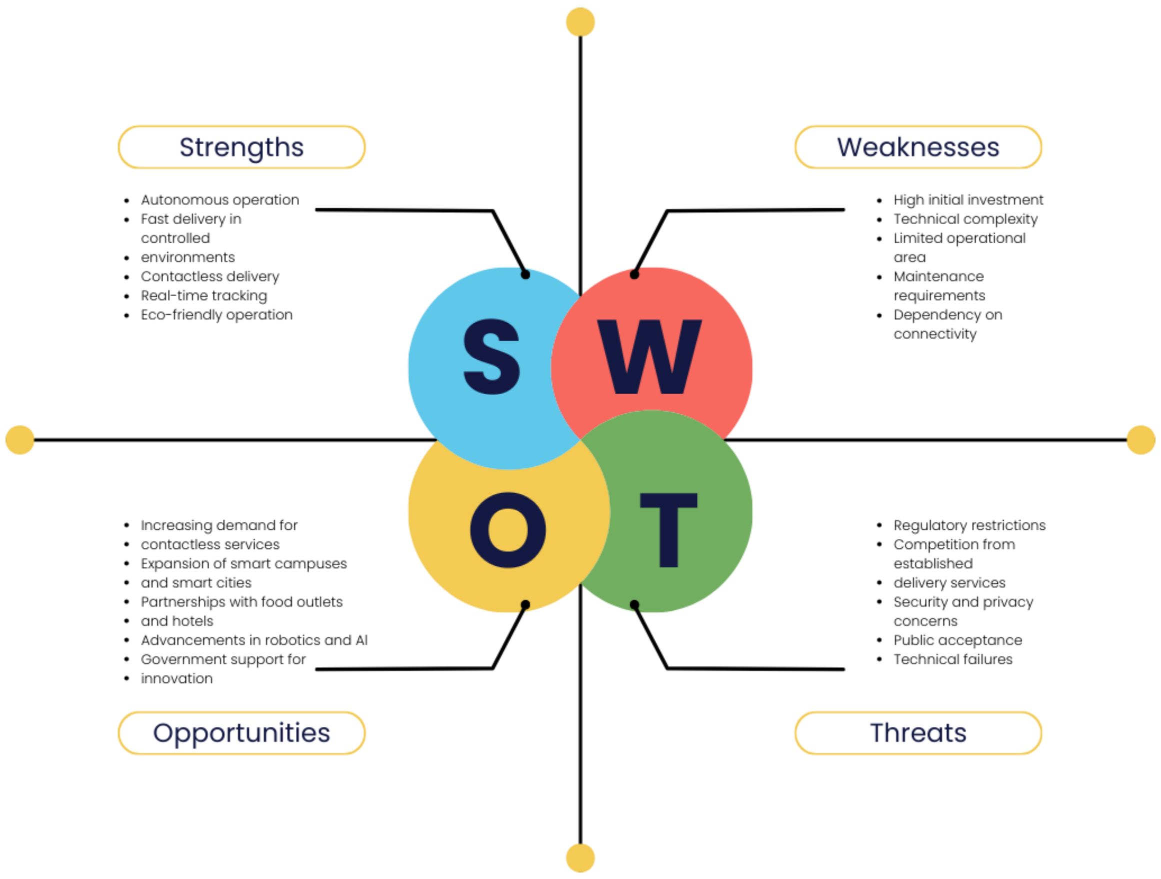


Figure. 5.3 SWOT Analysis Canvas illustrated

SWOT analysis (Figure. 5.3) is a strategic planning tool used to identify the internal strengths and weaknesses of a project, as well as external opportunities and threats that may affect its success. In the context of the proposed autonomous food delivery robot, SWOT analysis helps in evaluating the project's feasibility, market potential, and risks, and in developing strategies to maximize advantages while minimizing limitations.

5.3.1 Strengths

Strengths represent internal attributes and resources that provide the project with competitive advantages. The proposed autonomous food delivery robot has several strengths that enhance its performance and value proposition.

Key strengths include:

- **Autonomous operation:** Reduces dependency on human drivers and lowers labor costs, improving operational efficiency.
- **Fast delivery in controlled environments:** Predictable routes and limited areas enable shorter delivery times.
- **Contactless delivery:** Enhances safety and hygiene, especially in crowded environments.
- **Real-time tracking:** Provides transparency and improves customer trust.
- **Eco-friendly operation:** Low energy consumption and reduced carbon footprint.

5.3.2 Weaknesses

Weaknesses refer to internal limitations or challenges that may hinder the project's success. Identifying weaknesses helps in planning mitigation strategies and resource allocation.

Primary weaknesses include:

- **High initial investment:** The cost of robot development, sensors, and infrastructure may be significant.
- **Technical complexity:** Requires advanced navigation, obstacle avoidance, and integration with software systems.
- **Limited operational area:** Designed for controlled environments only, which limits scalability.
- **Maintenance requirements:** Regular maintenance and troubleshooting may increase operational expenses.
- **Dependency on connectivity:** Requires stable network connection for tracking and control.

5.3.3 Opportunities

Opportunities are external factors that can be leveraged to improve project success. These factors may include market trends, technological advancements, and strategic partnerships.

Key opportunities include:

- **Increasing demand for contactless services:** Customer preference for safe and convenient delivery continues to grow.
- **Expansion of smart campuses and smart cities:** Integration with IoT infrastructure and smart management systems.
- **Partnerships with food outlets and hotels:** Collaboration opportunities to expand service coverage and market reach.
- **Advancements in robotics and AI:** Technology improvements reduce costs and enhance system performance.
- **Government support for innovation:** Grants and incentives for smart and sustainable solutions.

5.3.4 Threats

Threats are external factors that may negatively impact the project's performance. Identifying threats helps in developing risk mitigation and contingency plans.

Primary threats include:

- **Regulatory restrictions:** Rules and regulations regarding autonomous robots in public or semi-public areas.
- **Competition from established delivery services:** Existing platforms with large user bases and strong brand recognition.
- **Security and privacy concerns:** Risks related to data protection and cyber threats.
- **Public acceptance:** Resistance to adopting robotic services due to trust or cultural factors.
- **Technical failures:** Sensor malfunctions or navigation errors that may cause service disruptions.

The SWOT analysis indicates that the proposed autonomous food delivery robot possesses strong internal advantages in terms of efficiency, safety, and innovation. However, the project faces challenges related to investment, technical complexity, and regulatory requirements. By leveraging external opportunities and addressing potential threats, the project can achieve successful implementation and sustainable growth in controlled environments.

5.4 Fundraising and Commercialization Strategy

While the proposed autonomous delivery robot system is developed and validated as an academic and technical prototype, its transition into a real-world solution requires a clear commercialization and funding strategy. This section outlines a structured fundraising approach aimed at supporting system scaling, pilot deployment, and long-term sustainability, without affecting the technical feasibility of the proposed solution.

Fundraising is considered a **growth-enabling mechanism** rather than a requirement for system operation. The prototype can function independently within controlled environments; however, additional funding would accelerate deployment, improve system robustness, and enable wider adoption.

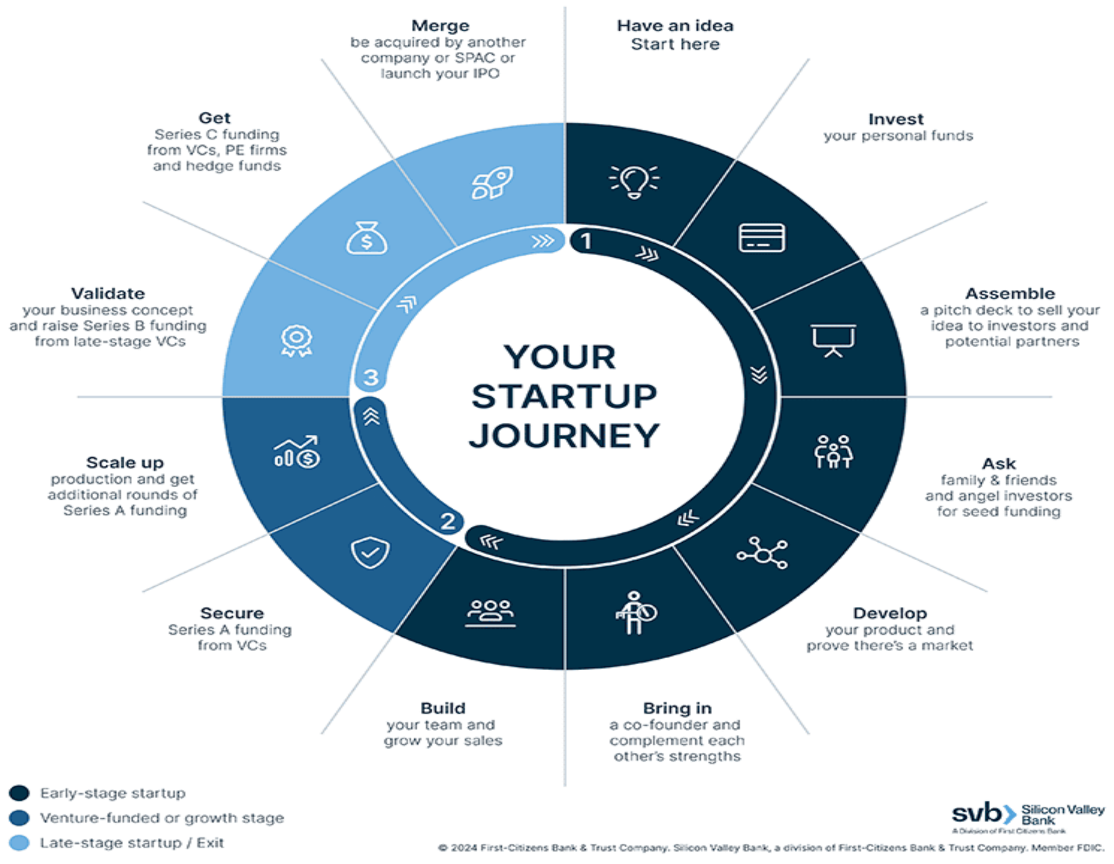
5.4.1 Purpose of Fundraising

The primary objective of fundraising is to support the transition from a single prototype to a deployable multi-robot system. Funding would be utilized to:

- Manufacture multiple robotic units for pilot programs
- Improve hardware reliability and durability
- Enhance software infrastructure and monitoring systems
- Establish charging and docking stations
- Support testing, certification, and operational validation

This approach ensures that the system can move beyond experimental deployment toward a scalable and commercially viable solution.

5.4.2 Funding Stages



YOUR STARTUP JOURNEY
Have an idea
Start here
Invest
your personal funds
Assemble
a pitch deck to sell your idea to investors and potential partners
Ask
family & friends and angel investors for seed funding
Develop
your product and prove there's a market
Bring in
a co-founder and complement each other's strengths
Build
your team and grow your sales
Secure
Series A funding from VCs
Scale up
production and get additional rounds of Series A funding
Validate
your business concept and raise Series B funding from late-stage VCs
Get
Series C funding from VCs, PE firms and hedge funds
Merge
be acquired by another company or SPAC or launch your IPO
1
2
3
● Early-stage startup
● Venture-funded or growth stage
● Late-stage startup / Exit
svb Silicon Valley Bank
© 2024 First-Citizens Bank & Trust Company. Silicon Valley Bank, a division of First-Citizens Bank & Trust Company. Member FDIC.

Figure. 5.4 Startup funding stages

The fundraising strategy is divided into progressive stages (shown in Figure. 5.4) aligned with system maturity. Each stage has clear technical and operational goals.

Prototype Stage
At this stage, the system is developed as a graduation project using university resources and student expertise. The focus is on technical feasibility, navigation accuracy, and system integration.

Pilot Deployment Stage
This stage targets limited real-world deployment in controlled environments such as university campuses or residential compounds. Funding at this stage may come from innovation grants, incubators, or institutional partners. The objective is to validate system performance under real operational conditions.

Scaling and Commercialization Stage
Once pilot deployment proves successful, larger-scale funding may be pursued to manufacture additional robots, expand service coverage, and improve operational efficiency. This stage focuses on long-term sustainability and revenue growth.

5.4.3 Potential Funding Sources

The proposed system aligns well with innovation-driven and technology-focused funding programs, particularly within developing markets. Potential funding sources include:

- University innovation and research support programs
- Technology incubators and entrepreneurship centers
- Government initiatives supporting smart cities and automation
- Strategic partnerships with universities, hotels, and residential compounds
- Corporate sponsorships and applied research collaborations

These funding sources reduce financial risk while supporting early adoption in semi-controlled environments.

5.4.4 Allocation of Funds

Funds obtained through fundraising would be allocated to technical and operational components directly related to system performance and scalability. Key allocation areas include:

- Robot hardware manufacturing and assembly
- Sensors, batteries, and embedded control units
- Charging infrastructure and docking stations
- Backend servers, cloud services, and data storage
- Maintenance tools, spare parts, and system monitoring

This allocation ensures that investment is focused on improving system reliability and operational readiness.

5.4.5 Risk Awareness and Academic Scope

It is important to emphasize that fundraising is presented as a **future commercialization strategy** rather than a dependency for system success. The autonomous delivery robot remains technically feasible as a standalone prototype. Financial planning is included to demonstrate real-world awareness and to support potential expansion beyond the academic scope.

6 Conclusion and Future Work

This graduation project presented the design, implementation, and experimental validation of an **Autonomous Delivery Robot (ADR)** intended for last-mile delivery applications in structured and semi-structured environments such as university campuses, residential compounds, and controlled urban zones. The work adopted a holistic engineering approach that integrates mechanical design, embedded systems, perception using artificial intelligence, system-level software architecture, and experimental evaluation.

From a mechanical engineering perspective, the robot was designed with a strong emphasis on structural reliability, modularity, and manufacturability. The chassis, motor base, and wheel assemblies were modeled using CAD tools and validated through finite element analysis (FEA). Static structural simulations demonstrated that all critical components operate within safe stress limits under expected operational loads, with acceptable factors of safety. The motor base was identified as the most critical structural element; however, even in this case, stresses and displacements remained below allowable thresholds, confirming the soundness of the mechanical design for prototype operation.

On the embedded and electrical side, the system architecture successfully separated high-power actuation from low-power logic and sensing domains, improving robustness and safety. The integration of a microcontroller-based low-level control layer with a high-level onboard computing unit enabled reliable motor control, sensor interfacing, and real-time communication. This layered architecture allowed time-critical tasks, such as motor PWM generation and encoder feedback processing, to be handled efficiently at the embedded level, while computationally intensive perception and decision-making tasks were executed on the onboard computer.

A major contribution of this project lies in the development and evaluation of the **YOLO3D perception pipeline**, which combines real-time 2D object detection with monocular depth estimation to provide depth-aware perception on resource-constrained hardware. By integrating a lightweight object detection model with an efficient monocular depth estimation network, the system was able to estimate relative object distances and classify obstacles into distance zones (Near, Medium, Far). This information was then used to generate interpretable, rule-based navigation actions such as STOP, SLOW_DOWN, and PROCEED.

Extensive performance optimization was applied to ensure real-time operation on embedded hardware. Techniques such as resolution reduction, depth estimation throttling, caching, and efficient per-object depth sampling significantly improved system throughput while maintaining acceptable perception quality. Experimental results demonstrated a clear improvement in frame rate, reduced latency, and lower memory usage compared to an unoptimized baseline. These results confirm that advanced AI-based perception can be deployed effectively on low-power platforms when system-level optimization is carefully considered.

Real-world testing further validated the robustness of the proposed system. The robot demonstrated stable perception output, consistent action generation, and reliable operation across varying environmental conditions, including different lighting levels and obstacle configurations. The rule-based action policy proved particularly effective in providing predictable and safe behavior, which is essential for autonomous systems operating in shared human environments.

Beyond technical performance, the project also addressed practical considerations such as cost, manufacturability, safety, and ethical implications. A preliminary cost analysis showed that the proposed design is economically feasible and competitive compared to commercial delivery robots, especially when targeting localized or campus-scale deployment. The modular mechanical design and use of standard components further enhance manufacturability and ease of maintenance. Safety and risk analysis highlighted potential hazards and corresponding mitigation strategies, while ethical considerations emphasized transparency, pedestrian safety, and responsible deployment of autonomous systems.

In summary, this project successfully demonstrates the feasibility of a low-cost, modular, and AI-enabled autonomous delivery robot. The integration of mechanical validation, embedded control, optimized perception, and experimental evaluation provides a strong foundation for future development and potential real-world deployment.

6.1 Future Work

While the proposed Autonomous Delivery Robot achieves its primary objectives, several opportunities exist for extending and enhancing the system. These future directions span mechanical design, perception, navigation, system integration, and broader deployment considerations.

6.1.1 Mechanical Design Enhancements

Future iterations of the robot can benefit from further mechanical optimization. Although static structural analysis confirmed the safety of the current design, dynamic and fatigue analyses should be conducted to evaluate performance under long-term operation and uneven terrain. Incorporating suspension elements or compliant mechanisms could improve stability and vibration isolation when operating on rough surfaces. Additionally, weight optimization through topology optimization or alternative materials may reduce overall mass and improve energy efficiency.

6.1.2 Advanced Power Management

Future work may focus on enhancing the power system through intelligent energy management. This includes battery state-of-charge estimation, adaptive power allocation based on task demands, and regenerative braking techniques. Solar-assisted charging or battery-swapping mechanisms could also be explored for extended deployment scenarios.

6.1.3 Sensor Fusion and Localization

Although the current system focuses primarily on vision-based perception, integrating additional sensors such as LiDAR, stereo cameras, or inertial measurement units can significantly improve robustness. Sensor fusion techniques can provide more accurate localization, better obstacle detection, and improved performance under challenging visual conditions such as low light or glare. Future work could also incorporate SLAM-based localization for autonomous navigation in large-scale environments.

6.1.4 Learning-Based Navigation and Control

The present system relies on deterministic, rule-based action policies to ensure interpretability and safety. While effective, future research could explore learning-based navigation strategies, such as reinforcement learning or imitation learning, to enable more adaptive and efficient behavior. Hybrid approaches that combine rule-based safety constraints with learning-based decision making represent a promising direction for balancing safety and adaptability.

6.1.5 Depth Estimation Improvements

Monocular depth estimation inherently suffers from scale ambiguity and limited accuracy. Future enhancements may include improved calibration techniques, temporal consistency models, or the integration of stereo or depth sensors for metric distance estimation. Additionally, exploring model quantization or alternative lightweight depth networks could further improve performance on embedded platforms.

6.1.6 System Optimization and Hardware Acceleration

Further performance gains may be achieved through hardware acceleration and optimized inference backends. Exploring advanced inference engines, model compression techniques, and dedicated AI accelerators could significantly increase throughput and reduce power consumption. Automated benchmarking tools could also be developed to dynamically select the optimal configuration based on available hardware resources.

6.1.7 Connectivity and Cloud Integration

Future versions of the system may include tighter integration with cloud services for fleet management, remote monitoring, and data analytics. Secure communication protocols, over-the-air updates, and centralized dashboards would enable scalable deployment of multiple robots. Cloud-assisted learning and data aggregation could also be used to continuously improve perception and navigation models.

6.1.8 Human-Robot Interaction

Improving human-robot interaction represents another important research direction. Visual, auditory, or haptic feedback mechanisms could be added to communicate robot intent to nearby pedestrians. User interfaces for operators and recipients can also be enhanced to improve usability, trust, and acceptance of autonomous delivery systems.

6.1.9 Large-Scale Deployment and Field Trials

Finally, extensive field trials in real urban environments are essential to validate system performance under diverse conditions. Long-term deployments would provide valuable data on reliability, maintenance requirements, and user acceptance. Such trials would also help identify regulatory, infrastructural, and societal challenges associated with large-scale deployment of autonomous delivery robots.

6.2 Final Remarks

The work presented in this project establishes a solid foundation for autonomous delivery robotics by demonstrating a practical, scalable, and experimentally validated system. Through continued development and research, the proposed platform can evolve into a mature solution capable of contributing meaningfully to future smart mobility and logistics ecosystems.

7 References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.

[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.

[3] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Springer, Cham, Switzerland, 2016.

[4] M. Quigley *et al.*, “ROS: An Open-Source Robot Operating System,” in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA) Workshop*, Kobe, Japan, 2009.

[5] Open Robotics, “ROS 2 Documentation,” 2024. [Online]. Available: https://docs.ros.org

[6] S. Macenski, T. Moore, D. Lu, and A. White, “The Navigation Stack for Mobile Robots in ROS 2,” *IEEE Robotics & Automation Magazine*, vol. 27, no. 2, pp. 96–106, Jun. 2020.

[7] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 4th ed. Pearson, 2017.

[8] P. Corke, *Robotics, Vision and Control*, 2nd ed. Springer, Cham, Switzerland, 2017.

[9] K. Konolige *et al.*, “Efficient Sparse Pose Adjustment for 2D Mapping,” in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2010.

[10] H. Durrant-Whyte and T. Bailey, “Simultaneous Localization and Mapping: Part I,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, Jun. 2006.

[11] T. Bailey and H. Durrant-Whyte, “Simultaneous Localization and Mapping: Part II,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, Sep. 2006.

[12] J. Redmon *et al.*, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv preprint arXiv:2004.10934*, 2020.

[14] Ultralytics, “YOLO Documentation,” 2024. [Online]. Available: https://docs.ultralytics.com

[15] NVIDIA, “Isaac Sim Documentation,” 2024. [Online]. Available: https://developer.nvidia.com/isaac-sim

[16] Open Source Robotics Foundation, “Gazebo Simulator,” 2024. [Online]. Available: https://gazebosim.org

[17] MathWorks, “Finite Element Analysis (FEA) Fundamentals,” 2023. [Online]. Available: https://www.mathworks.com

[18] R. Budynas and J. Nisbett, *Shigley's Mechanical Engineering Design*, 10th ed. McGraw-Hill, 2015.

[19] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Prentice Hall, 2002.

[20] Espressif Systems, “ESP32 Technical Reference Manual,” 2023. [Online]. Available: https://www.espressif.com

[21] Raspberry Pi Foundation, “Raspberry Pi 5 Documentation,” 2024. [Online]. Available: https://www.raspberrypi.com

[22] Firebase, “Firebase Realtime Database & Cloud Functions,” Google, 2024. [Online]. Available: https://firebase.google.com

[23] A. Koubaa *et al.*, *Robot Operating System (ROS) for Absolute Beginners*, Springer, 2021.

[24] ISO 3691-4, *Industrial Trucks — Safety Requirements and Verification — Part 4: Driverless Industrial Trucks*, International Organization for Standardization, 2020.

[25] A. Elfes, “Using Occupancy Grids for Mobile Robot Perception and Navigation,” *Computer*, vol. 22, no. 6, pp. 46–57, Jun. 1989.