Faculty of Engineering – Ain Shams University

Mechanical-Mechatronics Department

Graduation Project

Multi-Agent Warehouse Co-Bot

Under the supervision of

Prof. Dr. Mohamed Ibrahim Mohamed Hassan Awad

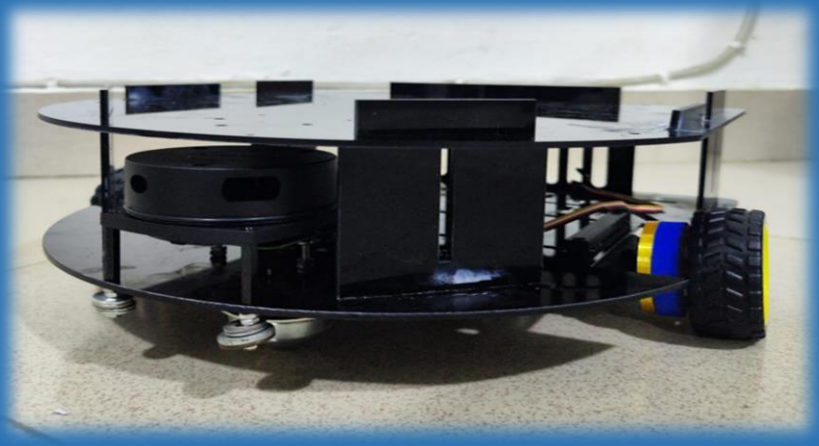
Presented by

- Thomas Medhat Naeem Fatooh 1900987
- Ruqaya Hamdy Awam 1804844
- Kareem Elsayed Mohamed Mahmoud 1806549
- Adham Walaa Mohamed Salah El-Din 1900747
- Mena Rober Arnest Elkoms 1900522

Multi-Agent Warehouse CO-Bot

Submitted By:

Thomas Medhat naeem Fatooh	1900987
Ruqaya Hamdy Zawam	1804844
Kareem Elsayed Mohamed Mahmoud	1806549
Adham Walaa Mohamed Salah El-Din	1900747
Mena Rober Arnest Elkoms	1900522



Program Name:

Mechatronics

Graduation Project Report Supervisor(s):

Prof/ Mohamed Ibrahim Mohamed Hassan Awad

Date:

24/6/2024

QR

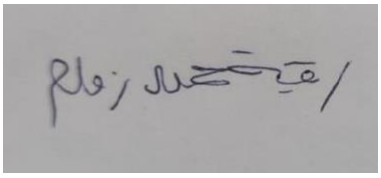
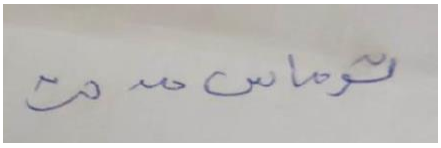
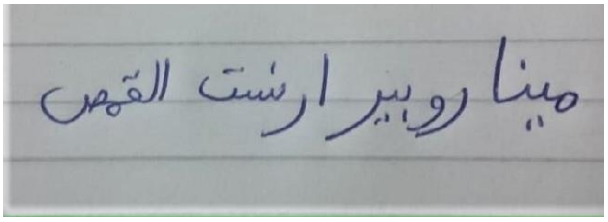
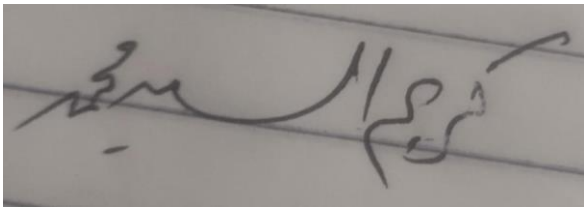
1 Elsarayat St., Abbaseya, 11517 Cairo, EgyptFax: (+20 2)26850617

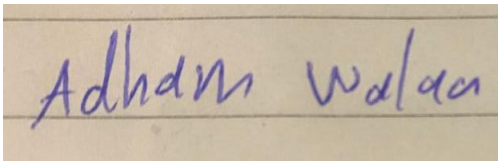
www.eng.asu.edu.eg

Declaration

We/I hereby certify that this Project submitted as part of our/my partial fulfilment of BSc in ***(Bachelor)*** is entirely our/my own work, that we/I have exercised reasonablecare to ensure its originality, and does not to the best of our/my knowledge breach any copyrighted materials, and have not been taken from the work of others and to theextent that such work has been cited and acknowledged within the text of our/my work.

Signed: by all students

Ruqaya Hamdy Zawam	
Thomas Medhat Naeem	
Mena Rober Arnest	
Kareem Elsayed Mohamed Mahmuod	

Adham Walaa Mohamed Salah El-Din	

Date: 24, June 2024.

Acknowledgment

With deep appreciation, we extend our gratitude to the esteemed faculty members for their guidance, knowledge, and unwavering support. Your collective commitment to education has shaped our academic journey and contributed to the success of thisendeavor.

we wish to express our sincere gratitude to our supervisor, **Prof/ Mohamed Ibrahim Mohamed Hassan Awad**, for giving us the chance to go through this challenge beside the immense knowledge, profound experience and professional expertise has enabled usto complete this project successfully. We wish to express our sincere thanks to Ain Shams University for accepting this project as a graduation project.

Thanks for all your encouragement!

Abstract

In this project, we dive into the transformative of Multi-Agent Robots to revolutionize warehouse efficiency. Employing cutting-edge robotics and artificial intelligence, our system offers autonomous agents within the warehouse system. These agentscollaborate intelligently, optimizing tasks like inventory management, order fulfillment, and navigation. Through rigorous analysis, this study unveils the impact of Multi-Agent Robots on enhancing warehouse settings and gets into the engineering guide of implementing robust cooperative warehouse robotic system.

This project is prepared, designed, and manufactured by Mechatronics students, this robot represents an integration between Mechanical, Electrical and Control systems in developing the robot.

Abbreviations

FEA – Finite Element Analysis

SLAM – Simultaneous localization and mapping

AGV – Automated Guided Vehicles

AMCL – Adaptive Monte Carlo Localization

IMU – Inertial measurement unit

NDT – Normal distributions transform

ICP – Iterative closest point

EKF – Extended Kalman Filter

DDS – Data Distribution Service

ROS – Robot Operating System

LIDAR – Light detection and ranging

RViz – ROS visualization

Table of Contents

Acknowledgment 1

Abstract 2

Abbreviations 3

1. Introduction 13

1.1 Problem Statement 14

1.1.1 Suboptimal Resource Utilization: 14

1.1.2 Increased Operational Costs 14

1.1.3 Delays in Order Fulfillment 14

1.1.4 Limited Adaptability to Change 14

1.1.5 Inefficiencies in Error Handling 14

1.2 RESEARCH QUESTION 15

1.3 RESEARCH QUESTIONS ANSWERS 15

1.4 Solution Abstraction 16

1.5 Project Objectives 17

1.6 Social Impact and Demand 18

1.7 Scholarly Context 18

2. Warehouse Case Study 19

2.1 Overall Situation 19

2.2 Scenarios In Numbers: 20

2.3 System Deployment 20

2.4 Scenario-Specification Alignment 21

2.5 PRE-POST ANALYSIS (ANNUALLY) 22

2.5.1 Warehouse Automation Market in the Middle East 23

2.6 THE MIDDLE EAST E-COMMERCE MARKET 24

2.7 WAREHOUSE AUTOMATION IS CRITICAL FOR E- COMMERCE FULFILMENT 26

2.8 KEY PLAYERS AND DEVELOPMENTS IN THE MIDDLE EAST 27

2.9 MARKET PENETRATION 29

2.9.1 Target Segment 29

2.1.1 Marketing Channels 29

2.1.2 Marketing Strategy 29

3. Autonomous Warehouse Co-bots Case Study 30
3.1 Theoretical Framework 30
3.1.1 Implementation of Co-bots: Planning and Integration 30
3.1.2 Workflow Optimization with Co-bots 31
3.1.3 Impact on Efficiency and Productivity 31
3.1.4 Safety Measures and Risk Mitigation 31
3.1.5 Future Prospects and Scalability 32
3.2 Autonomous System Design 32
3.2.1 mechatronics system design 32
3.2.2 Autonomous system use case 33
3.2.3 Components: 34
3.2.4 Workflow: 34
4. Theories and Methodologies 36
4.1 Differential Drive 36
4.1.1 Mathematical Modelling 39
4.1.2 Differential Steering Concept 40
4.2 Odometry 41
4.2.1 Servo Theory 41
4.2.2 How Encoders Contribute to Odometry 42
4.3 Perception 44
4.3.1 LIDAR 44
4.4 SLAM 46
4.4.1 Lidar SLAM 47
4.4.2 Mapping 50
4.4.3 EKF in SLAM 51
4.4.4 Particle Filter 52
4.5 Localization 53
4.5.1 AMCL 53
4.6 Sensor Fusion 54
4.6.1 sensors' limitations 54
4.6.2 EKF 54

4.7 Navigation 57
4.7.1 Dijkstra Algorithm 57
4.7.1 A* Algorithm 57
5. Co Bot Project Resources 59
5.1 Software tools 59
5.1.1 ROS 2 59
5.1.2 Micro-Ros 61
5.1.3 Gazebo 63
5.1.4 RVIZ 63
5.2 Electrical Hardware Tools 64
5.3 Mechanical Hardware Components 65
5.4 Wiring Diagram 66
5.5 Power Specifications 67
6. Mechanical Design 68
6.1 Design 68
6.2 Manufacturing 72
6.3 Payload 74
6.4 Stress Analysis 75
6.4.1 1'st and 3'd principal 75
6.4.2 Stresses of all the (x, y, z) planes 76
6.4.3 Displacement in the 3 directions 77
7. Software Implementation 78
7.1 URDF (Universal Robot Description Format): 78
7.2 Xacro (XML, Macros): 79
6.2.1 Key Features of Xacro: 79
6.2.2 Benefits of Using Xacro: 79
6.3 Install Important ROS 2 Packages: 80
6.4 Results 81
6.5 ROS 2 Navigation Stack 84
6.5.1 Navigation Concepts: 86
6.5.2 Robot configuration: 89

7. Hardware Implementation 98
7.1 Hardware Interface Diagram 98
7.2 Implementing ROS2 Control in a Differential Drive Robot 98
7.2.1 Hardware Interface 98
7.2.2 Controller Configuration 99
7.2.3 Implementation 100
7.2.4 ros2 control terminal launch 100
7.3 Motors Implementation code on ESP32 102
7.4 CPP ros2-control diff-drive Hardware Interface Code 110
7.5 HPP Header File for Hardware Interface 113
7.6 ROS 2 Driver MPU-6050 114
7.6.1 Node Structure 114
7.6.2 Example Workflow 114
7.7 How the IMU Translates its Readings into /imu Message 114
7.7.1 Mathematical Explanation and Theory 114
7.7.2 Linear Acceleration 115
7.7.3 Angular Velocity 115
7.8 Calibration Process 115
7.8.1 Steps in Calibration 115
7.8.2 Practical Implementation 116
7.9 System Architecture 117
7.10 Slam in Real Time 117
7.11 Communication Protocols 118
7.11.1 Esp32 and Raspberry-Pi (Serial Communication) 118
7.11.2 IMU MPU6050 and Raspberry-Pi (I2C) 118
7.11.3 LIDAR and Raspberry-Pi (Serial Communication) 119
7.11.4 Application and Raspberry-Pi (WIFI Communication) 119
8. Mobile Application 120
8.1 Introduction 120
8.2 User Interface and Workflow 120
8.3 Screens and Functionalities 120
8.3.1 Entering ID and Location 120

8.3.2 Checking Robot Arrival Status 121
8.3.3 QR Code Scanning 122
8.3.4 Choosing the Destination 123
8.3.5 Final Delivery Check 124
9. Cost 126
10. Future Work 127
10.1 Integration with IoT Devices 127
10.2 Machine Learning for Predictive Maintenance 127
10.3 Energy Efficiency 127
10.4 Security Enhancements 127
10.5 Collaboration with External Systems 127
11. Conclusion 128
12. References 129

List of Figures

Figure 1 Autonomous warehouse robot transporting goods. 13
Figure 2 A robot sensing a human moving around. 16
Figure 3 The Middle East Warehouse Automation Market 23
Figure 4 E-Commerce, Fashion & Beauty and Grocery will be the biggest end-consumers. 24
Figure 5 Middle East e-commerce in 2025 25
Figure 6 E-commerce penetration as a percentage of total retail (2017) 26
Figure 7 Co Bots 30
Figure 8 Autonomous Driving System Design with Application 32
Figure 9 multi-agent co-bot use case flowchart 33
Figure 10 Real Routes VS. Theoretical Route 39
Figure 11 Differential Steering 40
Figure 12 Servo Theory 41
Figure 13 Servo Encoders 43
Figure 14 LIDAR principle 44
Figure 15 slam process diagram 46
Figure 16 Grid Map 47
Figure 17 grid map occupancy values 48
Figure 18 Cartographer 48
Figure 19 Cartographer SLAM method diagram 49
Figure 20 Mapping Types 50
Figure 21 EKF in SLAM 51
Figure 22 Particle Filter 52
Figure 23 AMCL ROS framework 53
Figure 24 Kalman Fiter. 55
Figure 25 EKF Vs EK 56
Figure 26 Equations Behind EKF 56
Figure 27 Dijkstra Algorithm 57
Figure 28 A* Algorithm 57
Figure 29 comparison between ROS & ROS2 59
Figure 30 Communication in ROS 2 59
Figure 31 ROS2 Real-time Capabilities 60
Figure 32 Micro-ROS 61
Figure 33 comparison between micro-ros and ros2 62

Figure 34 GAZEBO with ROS block diagram 63
Figure 35 Lidar 64
Figure 36 Raspberry pi 4 64
Figure 37 Servo Motor 64
Figure 38 Power bank 65
Figure 39 Esp32 board 65
Figure 40 IMU degrees of freedom 65
Figure 41 Caster Wheel 65
Figure 42 wheel 65
Figure 43 Acrylic Sheet 66
Figure 44 Wiring Diagram 66
Figure 45 side view of co-bot 68
Figure 46 frontal view of co-bot 69
Figure 47 Upper view of co-bot 70
Figure 48 view of components on the co-bot 71
Figure 49 upper view of co-bot 72
Figure 50 frontal view of co-bot 72
Figure 51 co-bot interior assembled 73
Figure 52 1’st and 3’d principal 75
Figure 53 6.5.2 Stresses of all the (x,y,z) planes 76
Figure 54 Displacement in the 3 direction 77
Figure 55 Robot Links and Joints 78
Figure 56 mobile robot with LIDAR 78
Figure 57 Launching the Robot's URDF in RVIZ 80
Figure 58 Launching the Robot's URDF in Gazebo 80
Figure 59 TF Tree 81
Figure 60 Multi-robot Launching on RVIZ 82
Figure 61 Multi-robot Launching on Gazebo 82
Figure 62 Viewing Robots Frames 83
Figure 63 rqt_graph 84
Figure 64 AMCL Localization 85
Figure 65 Navigation concepts 86
Figure 66 Behavior Trees 87
Figure 67 Nav2 Task 88

Figure 68 Slam 90
Figure 69 Results_1 96
Figure 70 Results_2 97
Figure 71 Hardware, Simulation and Controller Interface 98
Figure 72 Command Control Loop 99
Figure 73 Controller Configuration 100
Figure 74 Controllers Implementation 100
Figure 75 ros2 control terminal launch_1 100
Figure 76 ros2 control terminal launch_2 101
Figure 84 Hardware Interface Code_1 110
Figure 85 Hardware Interface Code_2 110
Figure 86 Hardware Interface Code_3 111
Figure 87 Hardware Interface Code_4 111
Figure 88 Hardware Interface Code_5 112
Figure 89 Header File of Hardware Interface_1 113
Figure 90 Header File of Hardware Interface_2 113
Figure 91 IMU Test 116
Figure 92 See, Think, Act system Architecture 117
Figure 93 Real Time SLAM 117
Figure 94 raspberry-pi and esp32 serial connection 118
Figure 95 raspberry-pi and IMU I2C connection 118
Figure 96 raspberry-pi and LIDAR serial connection 119
Figure 97 raspberry-pi WIFI connection 119
Figure 98 Application Layout_1 121
Figure 99 Application Layout_2 122
Figure 100 Application Layout_3 123
Figure 101 Application Layout_4 124
Figure 102 Application Layout_5 125

List of Tables

Table 1 Human Resources Calculations 20
Table 2 Quality Calculations 20
Table 3 Productivity Calculations 20
Table 4 Robot Specification 21
Table 5 Scenario Specification alignment 21
Table 6 post annually analysis 22
Table 7 Key Players in the middle East 29
Table 8 Project Cost 126

1. Introduction

Figure 1 Autonomous warehouse robot transporting goods.

The rapid advancement in robotics and automation has propelled the evolution of warehouse management systems towards increased efficiency and precision. In this context, our project aims to contribute to the field through the development of a multi- agent warehouse robotic system featuring differential two-wheel robotic cars. These intelligent agents are designed to perform diverse tasks within a warehouse environment, including lifting weights, path planning, and precise localization. The integration of cutting-edge technologies, that enables seamless communication and coordination between the robots, facilitating an intelligent and adaptive approach to navigating and executing tasks.

As we delve into the intricacies of this project, this chapter provides an introductory overview, articulating the problem statement, delineating the project's objectives, and establishing a scholarly context. The subsequent chapters will delve deeper into the theoretical foundations, methodology, results, and conclusions, unraveling the intricacies of our innovative multi-agent warehouse robotic system.

1.1 Problem Statement

The modern logistics landscape is constantly evolving, demanding efficient and agile solutions for warehouse management. Traditional methods struggle to keep pace with the dynamic nature of today's supply chain, leading to challenges such as suboptimal resource utilization, increased operational costs, and delays in order fulfillment.

1.1.1 Suboptimal Resource Utilization:

Traditional systems lack adaptability, leading to inefficient resource allocation. Human- operated processes may struggle to adjust personnel and equipment in real-time, resulting in underutilized staff or equipment in some areas and bottlenecks in others.

1.1.2 Increased Operational Costs

The rigidity of traditional methods contributes to higher operational costs, especially with manual labor. Lack of automation in repetitive tasks leads to inefficiencies, drivingup labor expenses. Stringent labor regulations in certain regions further compoundfinancial burdens on warehouses.

1.1.3 Delays in Order Fulfillment

Traditional systems struggle to keep pace with dynamic supply chain, impacting order fulfillment pace. Manual and inflexible processesresult in delays, dissatisfied customers, potential loss of business, and reduced overall supply chain efficiency.

1.1.4 Limited Adaptability to Change

Traditional warehouse systems are designed with specific configurations, making them less adaptable to changes in inventory, product variety, or warehouse layout. Shifting market demands or the introduction of new products may necessitate significant reconfiguration, leading to downtime and interruptions.

1.1.5 Inefficiencies in Error Handling

Manual processes are prone to errors, Traditional systems lack robust error detection and correction mechanisms. This leads to incorrect shipments, misplaced inventory, and need for manual intervention, contributing to operational inefficiencies and customer dissatisfaction.

1.2 RESEARCH QUESTION

- How much will it cost when production rate doubles by using human and robots?
- How much effort can robots provide more than human?
- How much time can be saved by using robotics system instead of using only human?
- How heavy can robots carry and transport?
- How many robots can we use to keep the work safe for each other and the surrounding environment?
- How Controllable are the robots? Can they be monitored and send feedback of their status?
- What will be the additional requirements and modifications in sites for installing such system?
- What kind of maintenance is required for the robots and for what periods?

1.3 RESEARCH QUESTIONS ANSWERS

- **How much time can be saved by using robotics system instead of using onlyhuman?**

Robots cost only the operation cost in the power source, that can be provided nowadays from many cheaper renewable sources in addition to continuous uninterrupted working hours that can reach 24hrs per day and more, where some automatic systems can work with almost no humans in site using monitoring systems. For example, At SEAT's Martorell production plant in Spain, 125 AGVs help to shift 23,800 parts every day and cover someof the 400,000 Km travelled by the site's entire fleet every year, saving their human counterparts many millions of steps, then estimates that its blended fleet of AGVs has reduced production times by 25%.

- **How heavy can robots carry and transport?**

Robots can carry up to tons of materials. And some of the automated guided vehicles are in the shape of Forklifts vehicles that do the same job with less size and less cost than typical forklifts.

- **How many robots can we use to keep the work safe for each other and the surrounding environment?**

The robots can be provided with obstacle avoidance systems that allow them to avoid crashing with each other and with humans and environment.

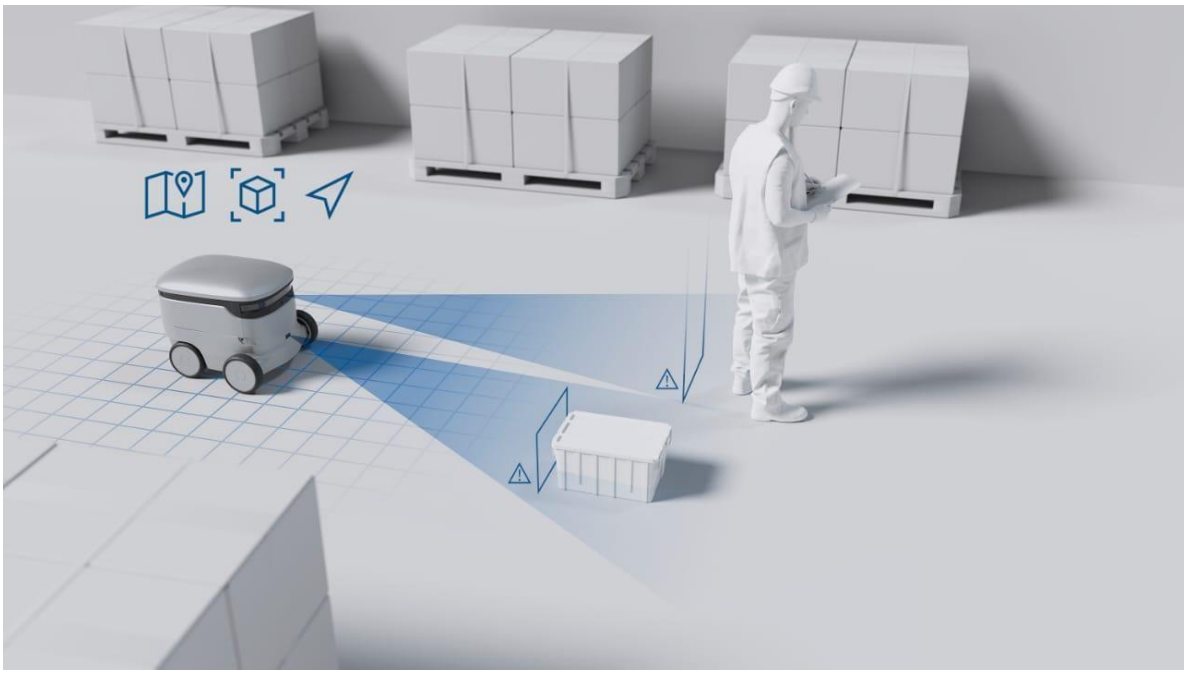


Figure 2 A robot sensing a human moving around.

- **How Controllable are the robots?**

Can they be monitored and send feedback of their status? Robots all can be connected to a control station that can send orders for the robots via wireless connections also robots can send their current position and destination to the station in addition to the alerts and messages they need to send for a critical decision- making situation.

- **What will be the additional requirements and modifications in sites for installing such system?**

The mapping techniques of the autonomous warehouse robotic systems uses a 2D floor-based map to navigate the robots' motion thus requires a straight-line paths and rectangular arrangement for the surroundings which is almost not an additional requirement for sites.

1.4 Solution Abstraction

Robotics, Automation and Artificial Intelligence sciences can be merged to develop a robots handle the problem in a more efficient way than before, Here're points that make the robot choice is efficient for the problem solution:

- Provides a Streamline Processes by Doing Repetitive that takes up employee's time that could be better spent on projects that require critical thinking. Robots, on the other hand, are designed to handle routine tasks while maintaining accuracy for long periods of time.

- Easier Scheduling: No Breaks Necessary Traditional workers require breaks and lunch periods to rest and rebuild their energy levels. Yet, no breaks are necessary for robots. They can continue working, uninterrupted, throughout a long shift.
- Fewer Injuries on the Job Robots don't require people to be with them 24/7. Programmed with sensors, they can perform their work while easily avoiding other robots and objects. This benefit lowers the chances of serious accidentsthat could impact a human worker's health and safety.

1.5 Project Objectives

The primary goal of our project is to address the limitations of conventional warehouse by introducing an innovative multi-agent robotic system. Key objectives include:

Enhanced Efficiency: Develop a system that significantly improves the efficiency of warehouse operations, reducing time required for operational tasks.

Cooperation with workers: The primary advantage to warehouse co-robots is their ability to increase productivity hand in hand with the ability to safely co-exist with workers, by working collaboratively with them.

Optimized Resource Utilization: Implement intelligent algorithms and coordination mechanisms to ensure optimal use of robotic agents, minimizing idle time and maximizing overall throughput.

Adaptability and Scalability: Design a system that can adapt to changing warehouse environments and easily scale to accommodate varying workloads and inventory sizes

Error Reduction: Integrate robust error detection and correction mechanisms to minimize mistakes in order fulfillment, leading to improved customer satisfaction and trust.

Safety Concerns: providing a safe warehouse environment is a main priority for working labor and productivity.

1.6 Social Impact and Demand

Robots will enhance factory logistics by taking over repetitive and dangerous tasks, allowing humans to focus on creativity, social skills, and emotional intelligence. This collaborative workforce marks the Fourth Industrial Revolution, redefining worker roles and environments. However, automation may impact jobs negatively, potentially dehumanizing work if humans become too subservient to machines. Despite their benefits, robots still require significant reprogramming for new tasks and can't learn new skills from humans or experience. The automation debate should consider how task automation changes the nature of work, not just job loss.

1.7 Scholarly Context

Our project is situated within the broader context of autonomous robotics, artificial intelligence, and logistics management. Building upon existing research in these domains, we aim to contribute to academic discourse by introducing a robust approachto multi-agent systems for warehouse environments.

Using theories from robotics, artificial intelligence, and operations research, we strive tocreate a comprehensive solution that not only addresses the immediate challenges of warehouse management but also contributes valuable insights to the academic community. As we delve into the subsequent chapters, we will explore the theoretical foundations, elaborate on the methodology employed, present the results obtained, and draw meaningful conclusions that shed light on the intricacies of our innovative multi- agent warehouse robotic system. Autonomous Warehouse Co-bots Case Study

2. Warehouse Case Study

2.1 Overall Situation

X-commerce they are online shopping companies that allow their customers to buy goods online and they deliver it. Their edge is next day delivery. They promise their customers if they ordered anything before 9pm, they guarantee the delivery next day. Their warehouse needs to have a very efficient process to make sure they fulfill all orders on the promised time.

Currently they are implementing a strategy that depends on hiring many workers, so Everything gets done at the right time. They have 4 main stages to fulfill any order, after The last stage delivery is handled by an outsourced company. The 4 stages are:

-Inventory

- Handling
- Preparation
- Cashier

X-commerce is experiencing delays due to human errors and issues with workers. The management is thinking about introducing automation solutions to avoid all issues with workers and have optimized, efficient operations. In addition to the main delay challenge there are more challenges that most e-commerce businesses are dealing with that is related to:

- Fluctuating basket size
- Many SKUs and low inventory
- High peak times/days
- Low picking efficiency of orders with multiple products

2.2 Scenarios In Numbers:

Human Resources calculations:

No. warehouse workers No. warehouse Managers	60 8
Workers annual salary	2,880,000 (4000*12*60)
Managers annual salary	576,000 (6000*12*8)
Total annual HR Cost	3,456,000 (48k+72k)

Table 1 Human Resources Calculations

- Quality calculations

Cost of missing 1 order	100
No. orders missed / year	1800 (5*30*12)
Total annual cost of missing orders	180,000
AVG. Cost of returns and replacement/day	100
Total annual cost of returns and replacement	36,500 (365*100)

Table 2 Quality Calculations

- Productivity calculations

Working hours/day	9
Total annual working hours	2920 (365*8)
Cost of power consumptions/year	120,000 (12*10,000)

Table 3 Productivity Calculations

2.3 System Deployment

- Robot specifications:

Battery lifetime	**1.5-2 years**
Battery cycle time	**4.43 hrs**
Recharge time	**3 hrs**
Working hours/cycle/robot	**1.43 hrs**
Recharge time/cycle	**3 hrs**
Robot to workers ratio	**1:4**
work efficiency improved due to robots	**25%**

Table 4 Robot Specification

2.4 Scenario-Specification Alignment

According to the working hours per day and the robot specifications here is how the scenario and the specifications the following new arrangement are defined as follows:

No. workers **No. manager**	**10** **5**
No. robots required/shift	**10**
Total number of robots required	**20**
Cost /robot	**30,000**
Total Robot Initial cost	**600,000**
Total annual HR Cost w/robots	**1,020,000 (12*(10*5000+5*7000))**

Table 5 Scenario Specification alignment

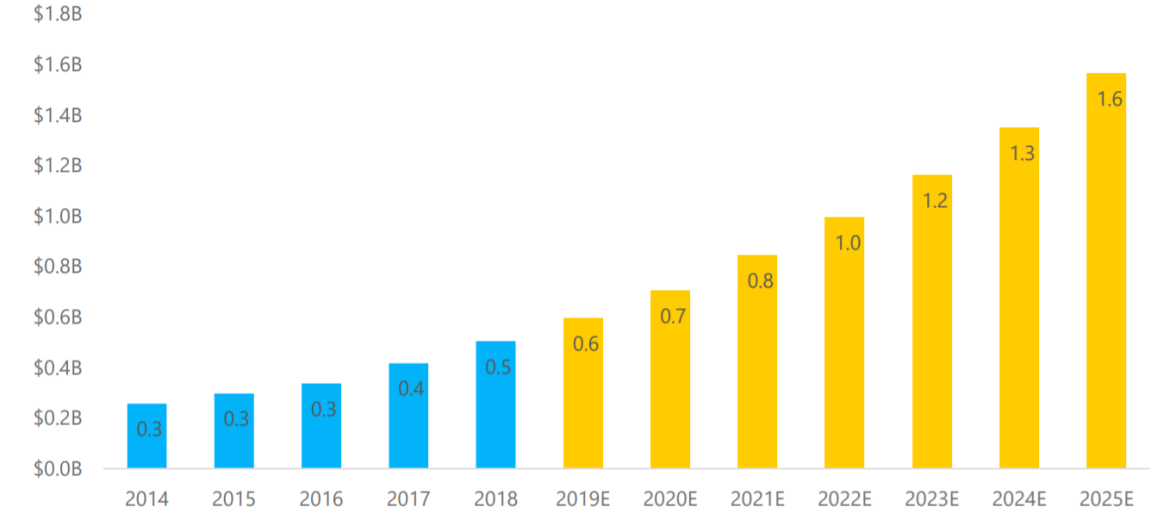
2.5 PRE-POST ANALYSIS (ANNUALLY)

	Pre	Post
Human resources cost	3,456,000	**1,020,000**
Quality Cost	217,000	**22,000**
Power consumptions Cost	120,000	**140,000**
Maintenance Cost	100,000	**400,000**
Total Annual Cost	**3,893,000**	**1,613,000**

Table 6 post annually analysis

2.5.1 Warehouse Automation Market in the Middle East

E-Commerce is the most important growth engine for warehouse automation technology throughout the world, and the same is true in the Middle East. Demand for increased warehouse space, as well as the use of automation solutions in the warehouse, is rising because of e-Commerce fulfilment. According to our estimations, the Middle East warehouse automation industry was worth USD 0.5 billion in 2015. It is expected to increase at a CAGR of 17.5 percent to USD 1.6 billion by 2025.



Sources:,LogisticsIQ

Figure 3 The Middle East Warehouse Automation Market

Note that The Middle East Warehouse Automation Market was worth $0.5 billion in 2018 and is expected to nearly triple to $1.6 billion by 2025.

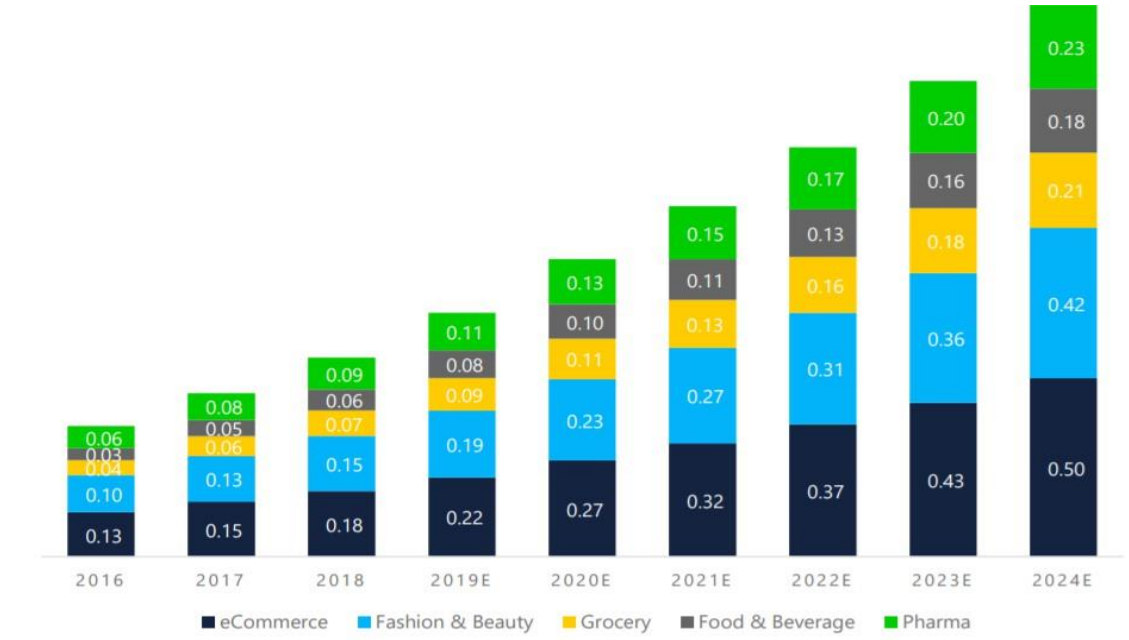
The major end-users of warehouse automation systems in the Middle East will be eCommerce, fashion and cosmetics, and groceries. Saudi Arabia's retail spending is expected to continue strong.

More competitiveness, better access to know-how, and increased access to funding are some of the primary factors driving Saudi Arabia's internet penetration growth. In Europe, traditional brick-and-mortar online grocery players have built successful and competitive online grocery operations. Tesco online has a 40% market share in the UK, whereas traditional food retailers control two-thirds of the online grocery market.

Traditional grocery businesses' success may be due to their foresight in entering the industry early to avoid internet development and choosing the proper entrance strategy. The fundamental pillars of an online grocery strategy are selection, quality assurance, pricing, and

delivery. Perishables are a prominent component of the online assortment for established players, and a typical online grocery platform contains between 20,000 and 50,000 Stock Keeping Units. The three biggest cost drivers in online grocery retailing are store distribution, picking costs, and last-mile delivery.

The most significant variable component, as well as the easiest to handle, is order picking and the challenge of overcoming the rise in labor expenses that results from having own staff perform the picking rather than the customers. Order picking is also an area where the industry is focusing its efforts in terms of automated and robotic solutions.



Sources:,LogisticsIQ

Figure 4 E-Commerce, Fashion & Beauty and Grocery will be the biggest end-consumers.

2.6 THE MIDDLE EAST E-COMMERCE MARKET

Global e-commerce sales totaled USD 2.9 trillion2 in 2012, accounting for 12.2 percent of all retail sales globally. E-commerce sales are expected to surpass USD 5 trillion by 2021, accounting for 22 percent of all retail sales by 2023. As the internet becomes more widely used and consumers move online, digital adoption is gaining traction in the Middle East, and the digital ecosystem is gradually developing in industries such as media and e- commerce

In 2017, the e-commerce market in the Middle East surpassed USD 10.3 billion. Saudi Arabia and the United Arab Emirates are the largest eCommerce economies, with a combined share of more than 50% in 2017 and expected to contribute more than 60% in 2018.

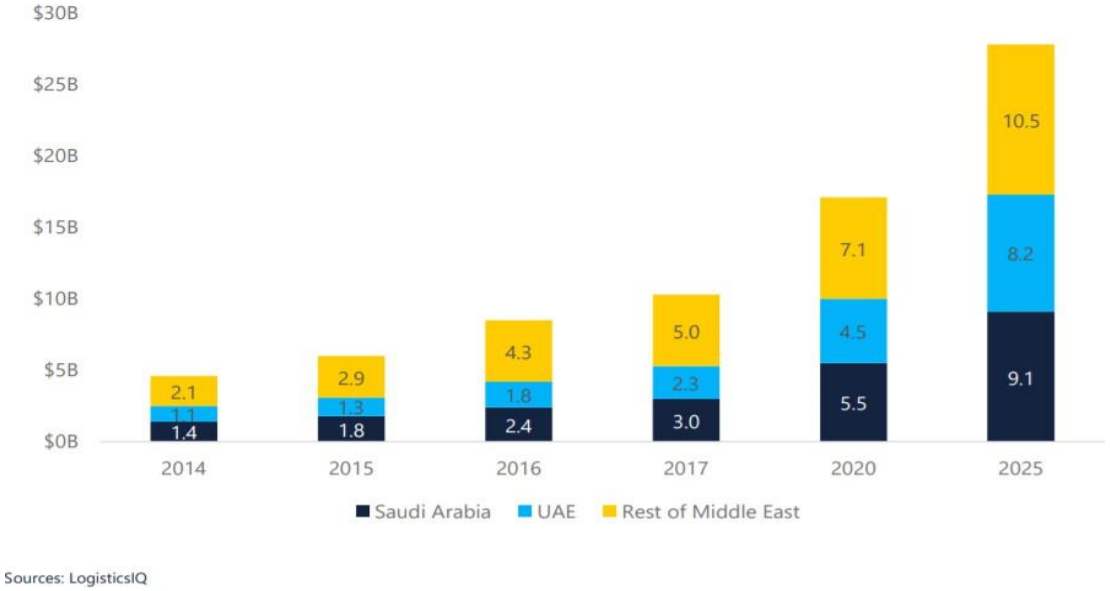


Figure 5 Middle East e-commerce in 2025

Amazon announced the purchase of Souq, the biggest MENA e-commerce firm, for USD 580 million in March 2017. Later same year, Mohamed Alabbar, chairman of Emaar Properties, launched noon, a new e-commerce company sponsored by the Saudi Arabian Public Investment Fund, with USD 1 billion in capital. In the Middle East eCommerce market, this signaled the start of a new era.

E-commerce development varies per country throughout the world. E-commerce penetration of total retail sales in China and the most sophisticated Western countries, such as the UK, US, France, and Germany, has reached 10%, with the UK reaching over 16%. In several other areas, such as Brazil, Turkey, and India, however, e-commerce penetration of total retail sales is still less than 5%. In 2017, e-commerce penetration of overall retail sales in the Middle East averaged 1.9 percent, with the GCC reaching 3 percent.

With a penetration rate of 4.2 percent, the United Arab Emirates (UAE) is the region's most advanced e-commerce market, comparable to Turkey and Brazil. The UAE is closely followed by the Kingdom of Saudi Arabia (KSA), which has a 3.8 percent share. Egypt's ecommerce penetration of total retail sales is comparable to India and Indonesia, at 2.5 percent.

E-commerce penetration in the Middle East is relatively low compared to other regions, therefore there is still a lot of opportunity for development. Considering the global backdrop,

the Middle East e-commerce market has a lot of space to develop in the next few years.

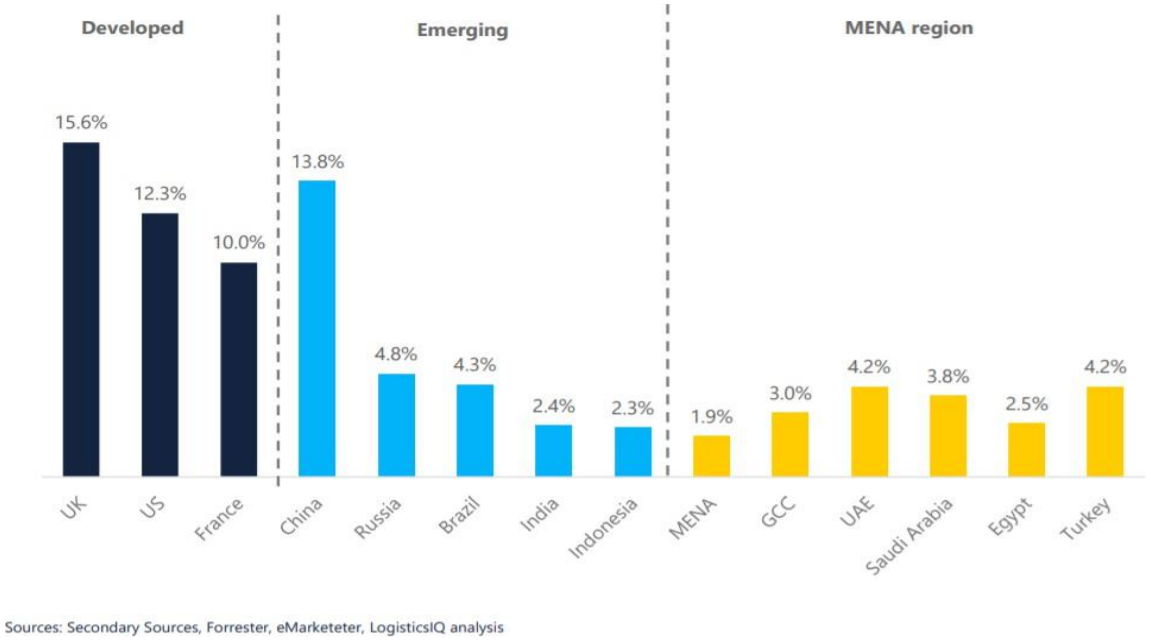


Figure 6 E-commerce penetration as a percentage of total retail (2017)

2.7 WAREHOUSE AUTOMATION IS CRITICAL FOR E- COMMERCE FULFILMENT

The surge in e-commerce is exacerbating the USD 5 trillion global logistics industry's workforce shortages. The number of shipments is quickly increasing, and internet shopping needs more logistical labor per item than brick and store retail. Online purchases need individual selection, packaging, and shipping, as contrast to traditional brick and mortar retail's bulk transportation strategies. Jack Ma, Alibaba founder famously said in 2013 that “For E-Commerce firms, the three most important infrastructure items are information flow, cash flow, and delivery.”

Over 90% of warehouse picking is now done by hand, and US firms employ 2 million people only to conduct stock and order fulfilment labor. When compared to pick-to- conveyor operations, automated picking results in productivity benefits of 2x–3x, and 5x– 6x when compared to human pick-to-pallet fulfilment centers. When compared to store- based fulfilment, online business models require 300 percent more warehouse space.

Euromonitor estimates that by 2035, approximately 2.3 billion square feet of extra warehouse space would be required due to worldwide eCommerce development.

Meanwhile, the $22 trillion conventional retail business is undergoing a dramatic transformation as it tries to adapt to changes in customer buying behavior brought on by the Internet, social media, and mobility. Omni-channel marketing is the most popular trend today, and it aims to connect physical and digital channels to provide a unified consumer experience and satisfy demand across all channels (webstore, ERP, point-of-sale, call center, mobile app, etc.). All brick-and-mortar shops, as well as online retailers growing their brick-and-mortar presence, are improving their supply-chain operations to improve inventory visibility and give a high degree of consumer experience.

2.8 KEY PLAYERS AND DEVELOPMENTS IN THE MIDDLE EAST

كارفور Carrefour ماجد الفطيم MAJID AL FUTTAIM	In 2018, Majid Al Futtaim introduced the Carrefour digital platform in Egypt, Jordan, Lebanon Saudi Arabia, and the United Arab Emirates, based on the SAP Hybris system, currently known as SAP Commerce Cloud. There is also a plan for an automated warehouse.
المراعي Almarai swisslog Mai Dubai	- Almarai, a Saudi Arabian food manufacturer, has placed a $47 million contract with Swisslog to automate its distribution logistics for its plants in Al Kharj, south of Riyadh. - Swisslog has started work on a $21 million warehouse expansion project for UAE water bottler Mai Dubai, which will see the company have a fully automated storage system by 2018.
DUBAI INDUSTRIAL CITY	- Dubai Industrial Park is leasing 70 warehouses ranging in size from 464 to 929 square meters. The warehouses are suitable for cold, chemical, and general storage, as well as light industrial usage.

	- Lootah Real Estate Development, based in the United Arab Emirates, has completed the W10 Warehouses, which are part of Senaeyat, a pre- built modern industrial manufacturing development in Dubai Investment Park. - Dubai Industrial Park has signed a long-term agreement with Lootah Real Estate Development to construct Senaeyat, a 28- hectare industrial project.
 	In Saudi Arabia, KNAPP is working on the project with local partner ALS Logistic Solutions for Healthcare Business. The new distribution center is expected to open in mid-2020.
	In late 2016, Kinedyne consolidated engineering, production, quality control, supply chain management, customer support, and government contract activities in its 200,000-square-foot facility to increase overall speed, efficiency, and productivity.
 	DHL Global Forwarding has agreed to construct a distribution center with KIZAD, Abu Dhabi's industrial hub and a subsidiary of Abu Dhabi Ports, to service the needs of its clients.
	DB Schenker mobilizes throughout the Gulf Cooperation Council (GCC) for the Ramadan logistics rush, moving more than 100,000 tons of consumer and retail products into the UAE. This can rise to 130,000 tons during Ramadan.

infor DANUBE	Danube Home has invested in Infor SCE, a renowned warehouse management system (WMS), and its Infor OS (operating services platform), which will be deployed by Vinculum, a warehouse management and logistics firm.

Table 7 Key Players in the middle East

2.9 MARKET PENETRATION

2.9.1 Target Segment

The target segment would be big corporate and lead manufacturers who has big warehouse and fulfill the following criteria:

- willingness to high initial investment
- has operations that deal with repetitive tasks.
- Decreased flexibility of operations.

Examples would be E-commerce businesses, Delivery companies and FMCG manufacturers.

2.1.1 Marketing Channels

Reaching out to their warehouse managers on LinkedIn or through personal connections. Ads account with paid ads that is targeted to the decision makers of our target segment. Webinars and events can take place to familiarize industry leaders about the benefits of implementing such technology in their warehouses and how it's going to make big difference based on real life data and case studies.

2.1.2 Marketing Strategy

We can offer a demo duration of working with robots for a short period of time could be one month or two. During the demo period small adjustments to their warehouse would be implemented or it can be through a rental warehouse of ours to show how things work and how robots can be very efficient.

3. Autonomous Warehouse Co-bots Case Study

3.1 Theoretical Framework

3.1.1 Implementation of Co-bots: Planning and Integration

Co-operative mobile robots: Robots can be used for different fulfilling industrial tasks including pick and place, packing, and transport of product between staging areas, our project main focus is co- operative bots. The primary advantage to co-bots is their ability to increased productivity hand in hand with the ability to safely co-exist with workers, by working collaboratively with them.

Figure 7 Co Bots

There are 2 types of co-bots:

Meet-Me Bots: These robots travel through a pick area, stopping and waiting ata pick location for a worker who performs the pick. Workers move from co-bot to co-bot within a dynamic zone, rather than traverse the full expanse of the warehouse.

Follow-Me Bots: These bots travel to a worker and lead them from picking location to location for some or all the picks on the bot. When full, the bot travels to a packing station and another co-bot are deployed to the picking worker's location.

In addition to the labor and time savings associated with reducing travel, the benefits of these robots include increased productivity and the ability to safely co-exist with workers. These robots are deployed for inventory replenishment tasks. There is also a potential application where a robotic arm is added to the mobile platform to eliminate the need for a human to execute the pick.

3.1.2 Workflow Optimization with Co-bots

The impact of autonomous warehouse co-bot mobile cars on efficiency and productivity is transformative. These co-bots streamline operations by automating repetitive and time-consuming tasks such as sorting, picking, and transporting goods. This automation significantly reduces the time taken to complete these processes, leading to faster order fulfillment and reduced lead times. Co-bots operate with high precision, minimizing errors and ensuring consistent quality in tasks. Additionally, they can work continuously without fatigue, maintaining high productivity levels around the clock. By handling routine tasks, co-bots free up human workers to focus on more complex and strategic activities, enhancing overall workforce productivity.

3.1.3 Impact on Efficiency and Productivity

Co-bots seamlessly integrate into existing workflows, enhancing coordination between different tasks and departments. They can be programmed to follow optimized paths, reducing travel time and congestion within the warehouse. By automating the movement of goods, co-bots ensure that items are delivered to their next processing point just in time, minimizing delays and bottlenecks. Their ability to operate in sync with human workers and other automated systems creates a harmonious workflow, where resources are utilized to their maximum potential. Additionally, co-bots can adapt to real-time changes in workflow, such as sudden increases in order volume, ensuring continuous and efficient operations. This dynamic optimization leads to smoother processes, faster turnaround times, and a more agile warehouse environment capable of meeting the demands of modern supply chains.

3.1.4 Safety Measures and Risk Mitigation

Safety measures and risk mitigation for autonomous warehouse are paramount to ensure seamless and secure operations. These systems are equipped with advanced sensors, such as LIDAR, ultrasonic, and infrared sensors, which provide real-time detection of obstacles and enable precise navigation. Safety protocols include programmed emergency stop functions and collision avoidance algorithms that prevent accidents and ensure the safety of human workers. Regular maintenance and software updates are crucial to maintain the efficiency and reliability of such systems. Additionally, implementing secure communication networks. Training employees to interact safely with co-bots and fostering a culture of safety awareness further minimizes risks, ensuring a harmonious and productive warehouse environment.

3.1.5 Future Prospects and Scalability

The future of autonomous warehouse co-bot mobile cars is highly promising, driven by advancements in artificial intelligence, machine learning, and sensor technologies. As technologies continue to evolve, co-bots will become more efficient, capable of handling complex tasks with increased accuracy and speed.

The scalability of these systems allows warehouses of all sizes to adopt automation, from small operations to large-scale distribution centers. Future developments may include enhanced interoperability between different robotic systems and integration with Internet of Things (IoT) devices. Additionally, as costs decrease and technology becomes more accessible, widespread adoption is expected to transform the landscape of warehouse management. This shift will not only improve operational efficiency but also create new job opportunities in tech support and maintenance.

3.2 Autonomous System Design

3.2.1 mechatronics system design

The Autonomous Driving System Design: The steps constitute of a continuous loop that forms the core of the SLAM process. As the robot continues its exploration, it gathers new sensor data, updates the map based on this data, refines its location estimate, and makes informed navigation decisions. This persistent cycle of perception, mapping, localization, loop closure, and refinement empowers the SLAM system to construct a comprehensive map and navigate through unknown environments with remarkable autonomy.

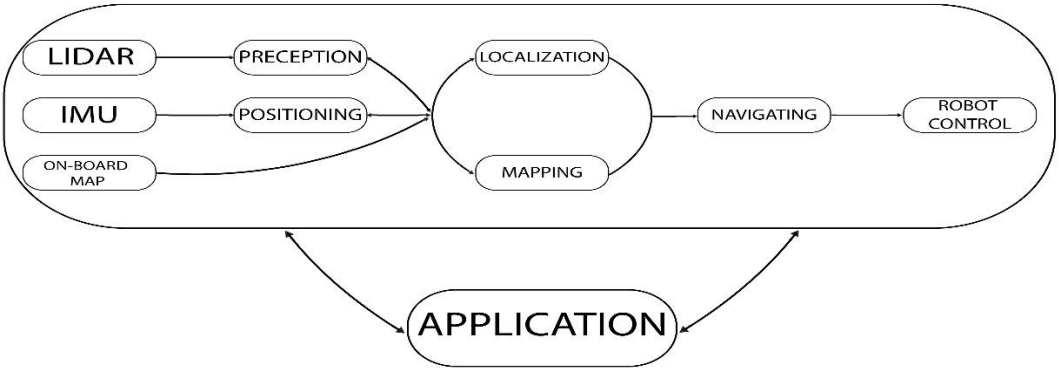


Figure 8 Autonomous Driving System Design with Application

3.2.2 Autonomous system use case

The diagram below outlines the step-by-step process of how a user interacts with the mobile app to manage warehouse logistics, including order placement, robot dispatch, and order delivery, with the backend server and ROS facilitating communication and control of the robots.

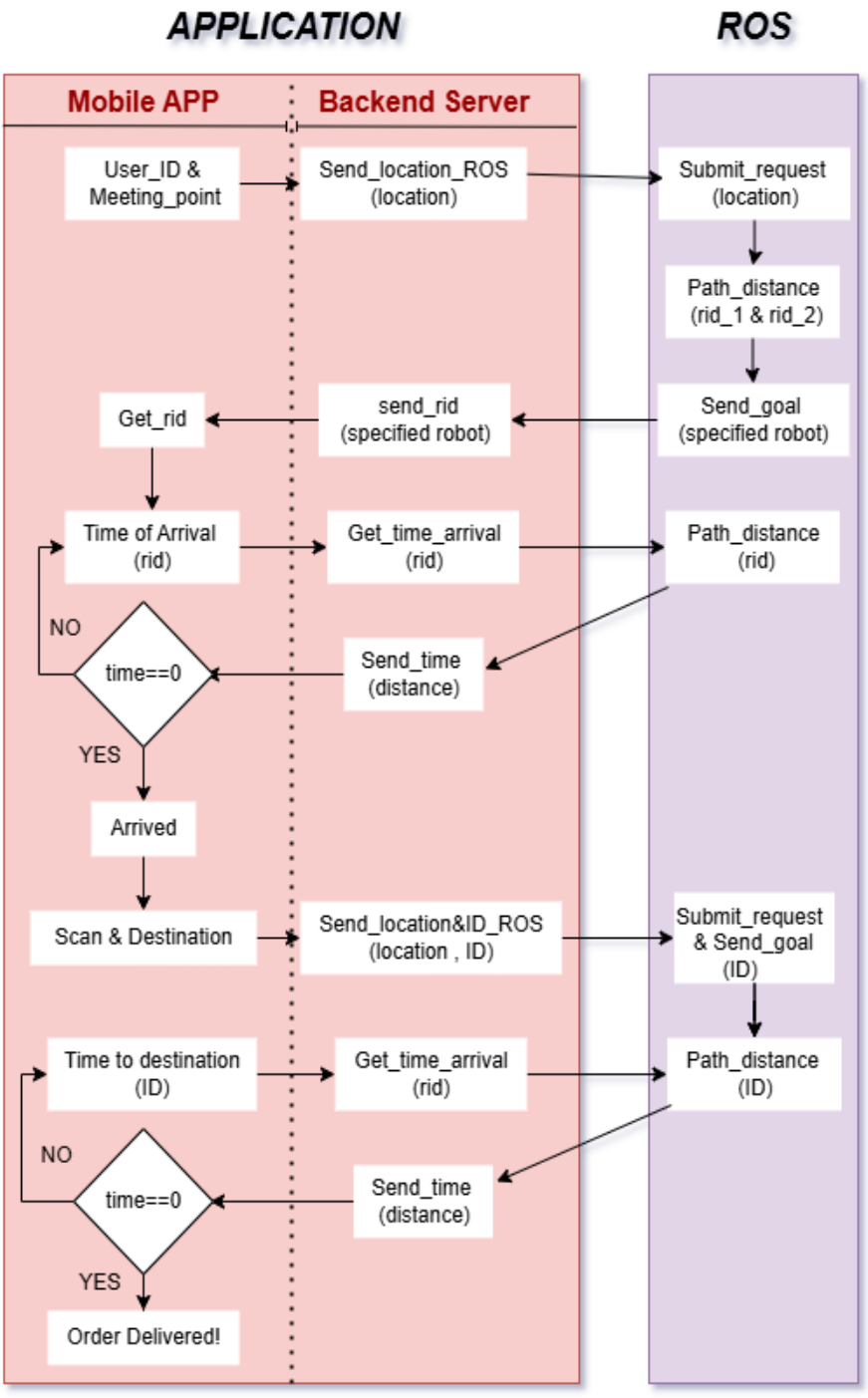


Figure 9 multi-agent co-bot use case flowchart

3.2.3 Components:

1. **Mobile App**: The front-end interface used by users to manage orders and track deliveries.
2. **Backend Server**: Manages communication between the Mobile App and the ROS.
3. **ROS (Robot Operating System)**: Controls the robots used for transporting items within the warehouse.

3.2.4 Workflow:

1. **User_ID & Meeting_point (Mobile App)**:

The user enters their ID and selects a meeting point (location).

2. **Send_location_ROS (Backend Server)**:

The backend server sends the location information to the ROS.

3. **Submit_request (ROS)**:

ROS receives the location and processes the request.

4. **Get_rid (Mobile App)**:

The mobile app requests a robot ID (rid) from the backend server.

5. **send_rid (Backend Server)**:

The backend server sends the robot ID to the mobile app.

6. **Path_distance (ROS)**:

ROS calculates the distance for the robot to travel to the specified location.

7. **Time of Arrival (Mobile App)**:

The mobile app checks the estimated time of arrival (ETA) for the robot.

8. **Get_time_arrival (Backend Server)**:

The backend server fetches the time of arrival from ROS.

9. **Send_time (Backend Server)**:

The backend server sends the time of arrival to the mobile app.

10. **Arrival Check (Mobile App)**:

The mobile app checks if the time is zero (robot arrived). If not, it keeps checking.

11. **Scan & Destination (Mobile App)**:

Once the robot arrives, the user scans the QR code on the robot and selects the destination.

12. **Send_location&ID_ROS (Backend Server)**:

The backend server sends the new location and ID information to ROS.

13. **Submit_request & Send goal (ROS)**:

ROS processes the new request and sends the goal (destination) to the robot.

14. **Time to destination (Mobile App)**:

The mobile app checks the ETA to the destination.

15. Get_time_arrival (Backend Server):

The backend server fetches the new time of arrival from ROS.

16. Send_time (Backend Server):

The backend server sends the updated time to the mobile app.

17. Arrival Check (Mobile App):

The mobile app checks if the time is zero (delivery completed). If not, it keeps checking.

18. Order Delivered (Mobile App):

Once the robot arrives at the destination, the order is marked as delivered.

4. Theories and Methodologies

4.1 Differential Drive

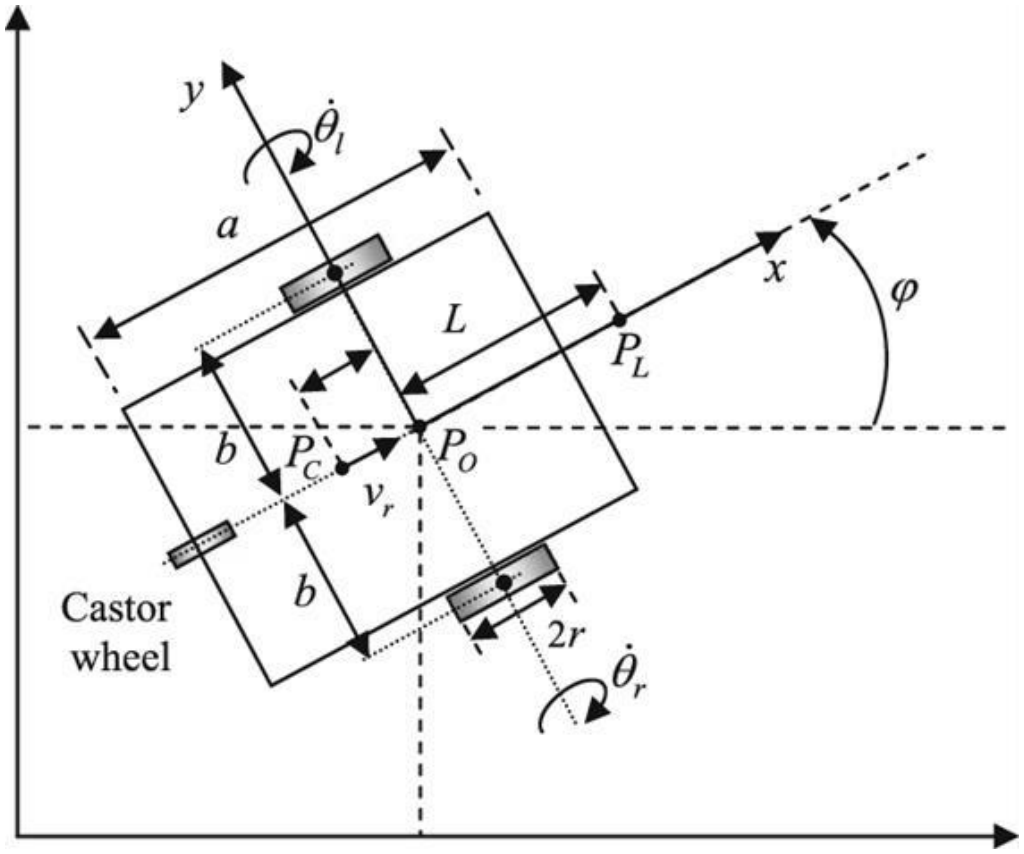


Figure 11 Differential Drive Robot Model

Differential drive is a system commonly used in mobile robots and, characterized by two independently driven wheels on a common axis. This design allows for differential control of the wheels, offering vehicle various maneuvers, like forward and backward motion, and rotation around its axis. The theoretical aspects of differential drive systems: Coordinate system of mobile robot to the track line is shown in **Fig. 9.** Linear velocities can be calculated refer to the wheel radius *R* and angular velocity *w* each wheel and:

$v_R = w_R * R$

$vL = wL * R$

A linear velocity v for the mobile robot calculated by average of the angular velocity of right and left driven wheel. It called by direct kinematic model of TWD linear velocity.

$$v = v_R + v_L$$
$$v = \frac{(w_R + w_l)}{2} * R$$

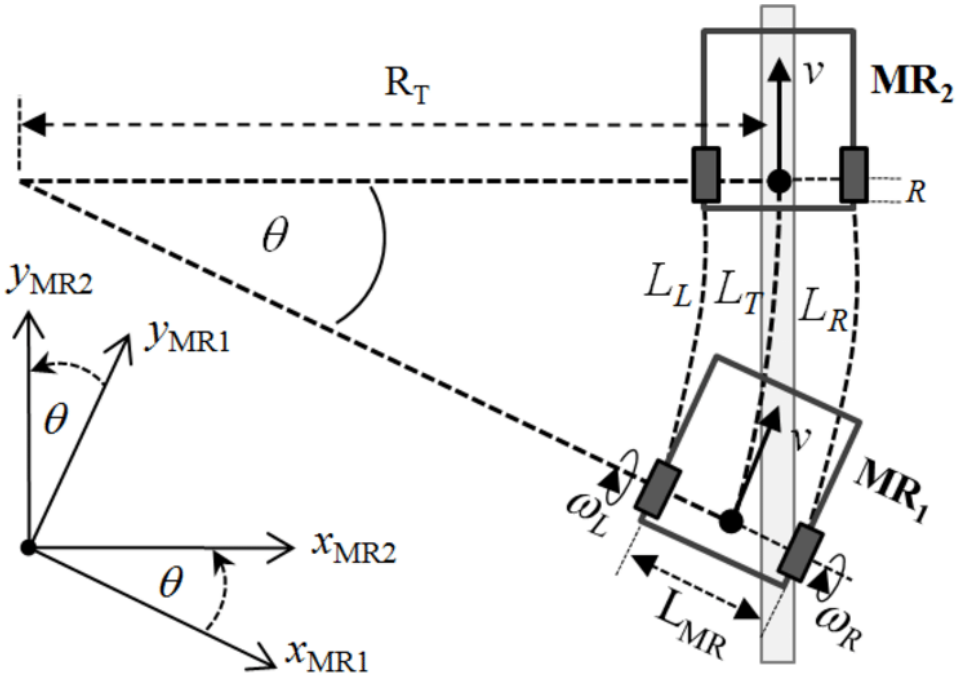
V is (Component of linear velocities $\dot{x}$ and $\dot{y}$ derived from the coordinate system consistof x and y position.)

$$\dot{x} = v * sin\,\theta$$
$$\dot{y} = v * \cos\theta$$

Substitution of linear velocity to the component velocities and imply that function of angular velocities to be:

$$\dot{x} = \frac{w_R + w_l}{2} * R * sin\,\theta \text{ [1]}$$
$$\dot{y} = \frac{w_R + w_l}{2} * R * cos\,\theta \text{ [2]}$$

Integrating and applying the initial position of the robot $\boldsymbol{x_0}$& $\boldsymbol{y_0}$, Assuming slip free motion (ignoring geometrical impossibilities, tires with finite width, inconsistent front and rear wheel speed, and slippery ground conditions). the equations confirms that when the wheels turn at fixed velocities, the robot follows a circular path.



Derive a differential equation describing the change in orientation as with respect to time of time. the robot's orientation as a function of wheel velocity and time

$$\theta_t = \frac{(v_R - v_L)t}{b} + \theta_0$$

As noted above, this change in orientation also applies to the absolute frame of reference. Shifting our attention back to the absolute frame. We combine this fact with what we know the about orientation as a function of time from the differential equations [1] and [2]

$$y(t) = y_0 + \frac{b(v_R + v_L)}{(v_R - v_L)}\left[\cos(\frac{(v_R - v_L)t}{2b} + \theta_o) - \cos(\theta_o)\right]$$

$$x(t) = x_0 + \frac{b(v_R + v_L)}{(v_R - v_L)}\left[\sin(\frac{(v_R - v_L)t}{2b} + \theta_o) - \sin(\theta_o)\right]$$

Note that the resulted equations confirm the earlier assertion that, when the wheels turn at fixed velocities, the robot follows a circular path. The term $\frac{\boldsymbol{b(v_R+v_L)}}{\boldsymbol{(v_R-v_L)}}$ is actually the turn radius for circular trajectory of the robot's center.

When using these equations in a computer application, it is necessary to implement special handling for cases where the wheel speeds are nearly equal, the robot will travel in a nearly straight line. The problem arises due to the small denominator in the division. Specifically, and as this difference approaches zero, the equations involve divisions by a very small number, which can lead to large numerical errors or undefined behavior. Using ***L'Hospital's*** rule from elementary calculus, we can show that the equations do have limits.

$$\lim_{\alpha \to 0} \sin(\theta_0 + \alpha t) - \sin(\theta_0) = t\cos(\theta_0)$$

where $\alpha = \frac{v_R - v_L}{2b}$

4.1.1 Mathematical Modelling

The differential drive system is often mathematically modeled to predict the vehicle's motion accurately. The relationships between wheel speeds, vehicle velocity, and turning radius are expressed through equations that facilitate precise control and navigation.

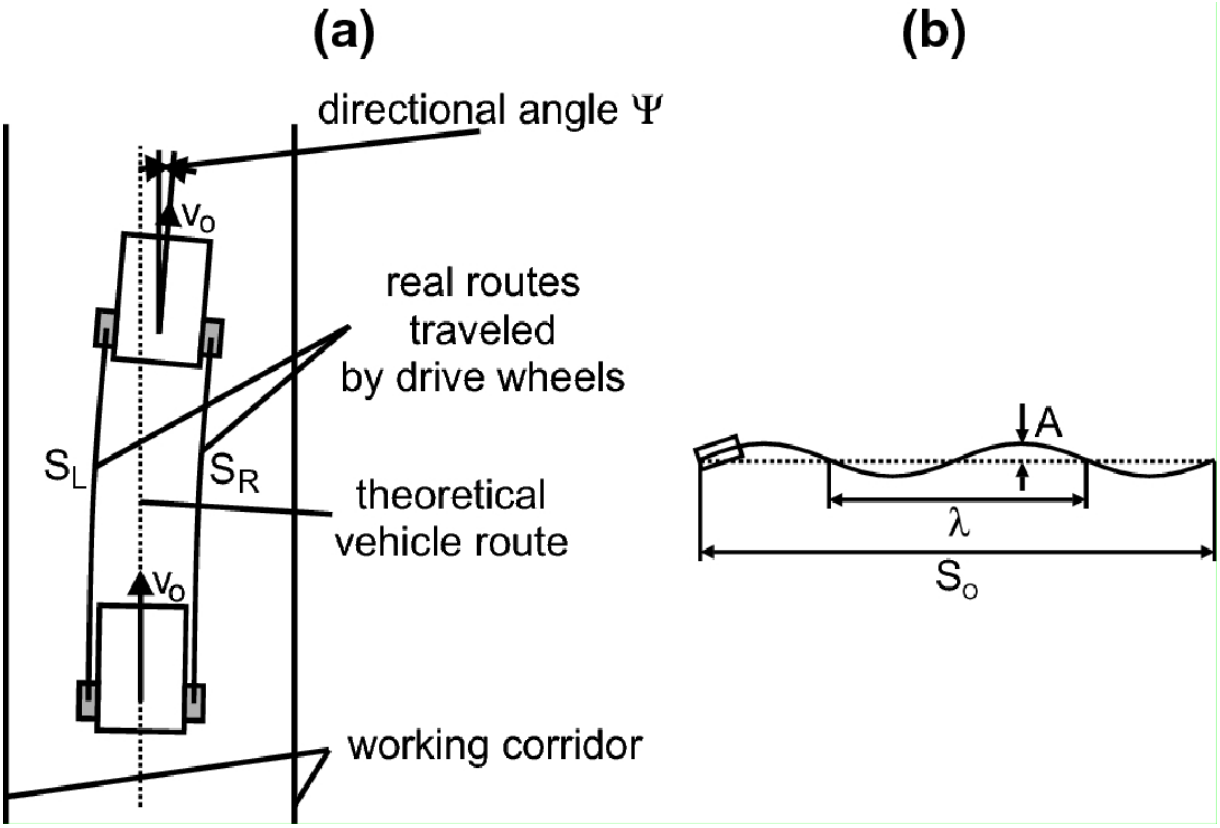


Figure 10 Real Routes VS. Theoretical Route

Limitations

While differential drive systems are versatile, they have limitations. One commonchallenge is accurately controlling the vehicle's trajectory, especially at higher speeds orin environments with varying traction. Additionally, abrupt changes in speed or direction can lead to wheel slippage

4.1.2 Differential Steering Concept

The key principle behind differential drive is differential steering. By driving the two wheels at different speeds or in opposite directions, the vehicle can achieve different types of motion. For example, moving both wheels forward at the same speed results in straight-line motion, while driving one wheel faster than the other causes the vehicle to turn.

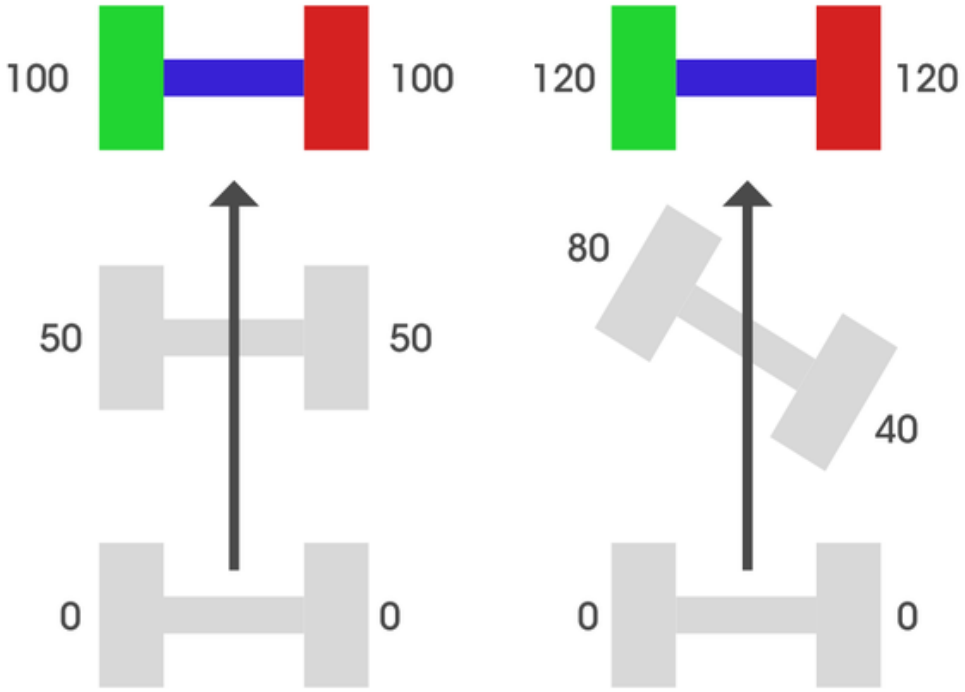


Figure 11 Differential Steering

4.2 Odometry

Differential drive systems use odometry to estimate the robot's position and orientation based on wheel encoders' readings. This information is crucial for navigation and mapping algorithms.

4.2.1 Servo Theory

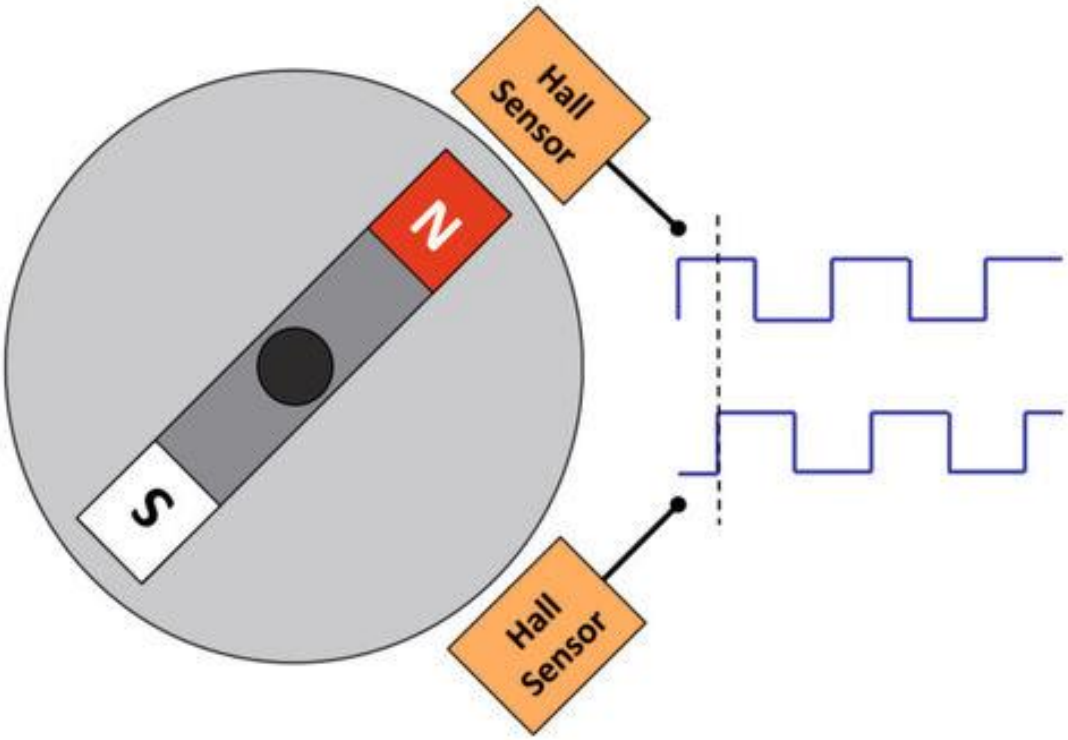


Figure 12 Servo Theory

Magnetic encoders are devices that convert the position, velocity, or direction of a magnetic field into an electrical signal, which can be interpreted by electronic systems. They are widely used in various applications due to their robustness and reliability. Here are the key theories and principles behind magnetic encoders:

Magnetoresistance:

Magnetic encoders often utilize the magneto resistive effect, where the resistance of a material changes in response to an external magnetic field. Magneto resistive sensors detect changes in resistance as a magnet moves past them, allowing precise measurement of position and rotation.

Hall Effect:

Some magnetic encoders use Hall effect sensors. The Hall effect occurs when a current-carrying conductor is placed in a magnetic field, generating a voltage perpendicular to both the current and the magnetic field. This voltage is proportional to the strength of the magnetic field, allowing for the determination of the magnet's position.

Magnetic Field Interpolation:

Magnetic encoders often interpolate the magnetic field changes to produce high-resolution position data. By placing multiple sensors around a magnet, the encoder can detect small changes in the magnetic field, resulting in fine-grained position measurements.

Incremental and Absolute Encoding:

Incremental Encoders: These encoders generate pulses as the magnet moves, with each pulse representing a fixed amount of movement. The system counts the pulses to determine the position relative to a starting point.

Absolute Encoders: These encoders provide a unique code or position value for each position of the magnet, ensuring that the exact position is always known, even after a power loss.

Signal Processing:

The raw signals generated by magneto resistive or Hall effect sensors are often processed to filter noise and enhance accuracy. Advanced algorithms can improve resolution and compensate for any non-linearities in the magnetic field.

Magnetic encoders are highly valued for their durability, resistance to environmental contaminants, and ability to function in harsh conditions, making them ideal for a wide range of industrial and automotive applications.

4.2.2 How Encoders Contribute to Odometry

- **Distance Traveled**: By calculating the number of ticks per time interval and knowing the wheel diameter, you can compute the distance traveled.
- **Angular Displacement**: For differential drive robots (with two wheels), the difference in wheel rotations helps determine the robot's change in orientation.

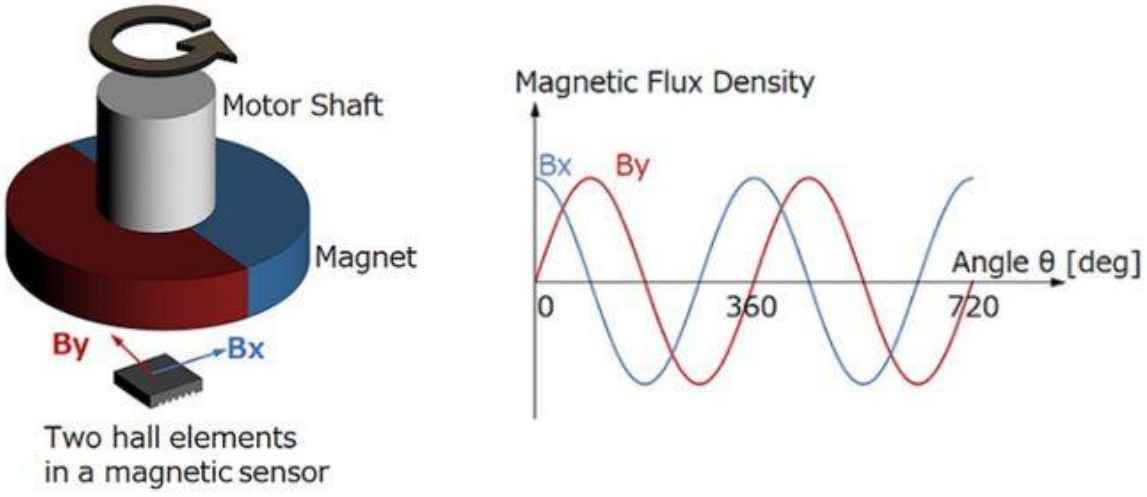


Figure 13 Servo Encoders

To transform encoder ticks to degrees per second, you'll need to consider the number of ticks per revolution of the encoder and the time interval over which you're measuring. Here is the general formula:

$$\omega_{\frac{deg}{sec}} = \left(\frac{\Delta analog\ value}{4096}\right) \times 360 \times \left(\frac{1}{\Delta t}\right)$$

Where:

ticks per revolution is the total number of encoder ticks per full revolution of the wheel or shaft.

Δt is the time interval in seconds over which the ticks are counted.

4.3 Perception

4.3.1 LIDAR

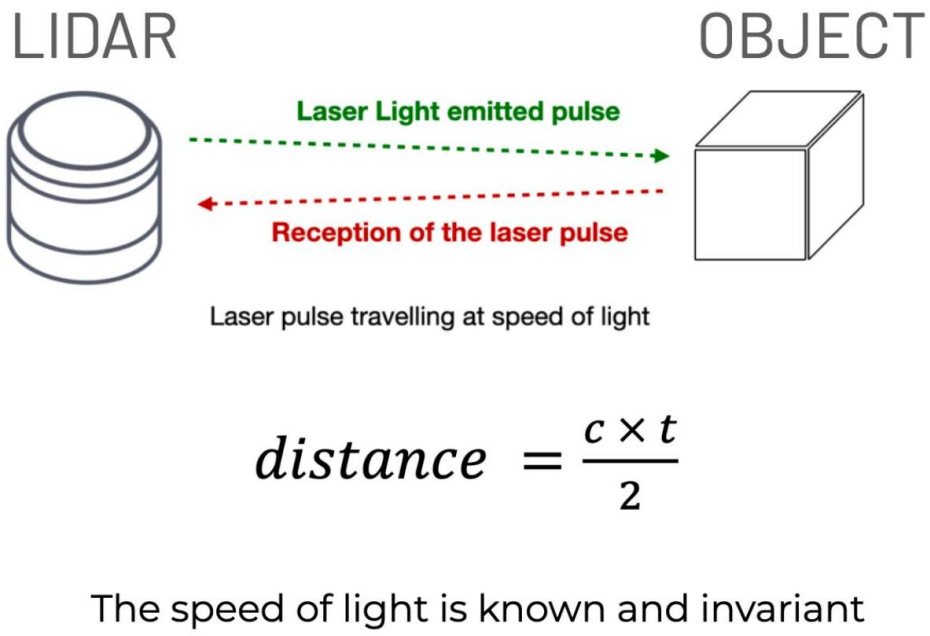


Figure 14 LIDAR principle

2D LiDAR (Light Detection and Ranging) works by emitting laser pulses and measuring the time it takes for each pulse to bounce back after hitting an object. The device rotates the laser emitter and receiver to scan the environment in a 2D plane. By calculating the time delay, the LiDAR determines the distance to objects in various directions, creating a 2D map of the surroundings.

- Components

Laser Emitter:

The laser emitter generates laser pulses, typically in the infrared spectrum, which are emitted into the environment.

Rotating Mirror or Scanner:

To achieve 2D scanning capability, the LiDAR sensor often incorporates a rotating mirror or scanner. This component rotates rapidly, directing the laser pulses in different directions across the field of view.

Optical Components:

Lenses and mirrors are used to focus and steer the laser pulses emitted by the laser emitter and the rotating mirror or scanner. These components ensure that the laser pulses are directed accurately and efficiently.

Receiver/Photodetector:

After the laser pulses are emitted and reflected off objects in the environment, the receiver or photodetector detects the reflected pulses. It measures the time it takes for each pulse to return (time of flight) and the intensity of the returned light.

Timing and Control Electronics:

Timing circuits are crucial for precisely measuring the time delay between pulse emission and reception. Control electronics manage the rotation of the mirror or scanner and coordinate the emission and reception of laser pulses.

Signal Processing Unit:

The signal processing unit processes the data received from the photodetector. It calculates distances to objects based on the time-of-flight measurements and constructs a two-dimensional map of the surroundings.

Data Interface:

Data interface:

(such as Ethernet, USB, or serial communication) to transfer the processed data to the host system (e.g., a robot's onboard computer or controller).

4.4 SLAM

SLAM is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. Popular approximate solution methods include the particle filter, extended Kalman filter, covariance intersection.

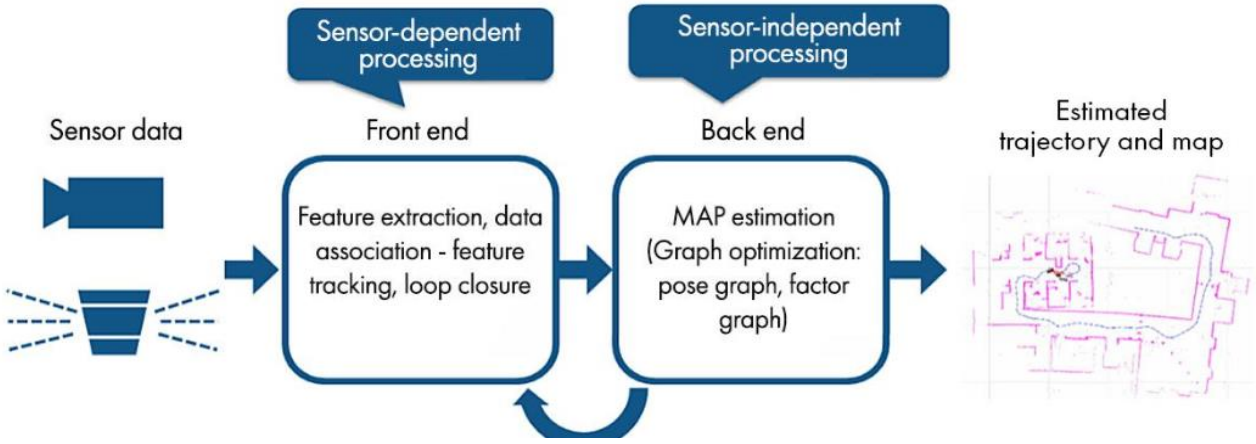


Figure 15 slam process diagram

SLAM algorithms are based on concepts in computational geometry and computer vision, used in robot navigation, robotic mapping and odometry. Given a series of controls $\boldsymbol{u_t}$ and sensor observations $\boldsymbol{o_t}$ over discrete time steps , the SLAM problem is to compute an estimate of the agent's state $\boldsymbol{x_t}$ and a map of the environment $\boldsymbol{m_t}$. All quantities are usually probabilistic, so the objective is to compute:

$$P(m_{t+1}, x_{t+1} | o_{1:t+1}, u_{1:t})$$

Applying byes 'rule gives a framework for sequentially updating the location posteriors, given a map and a transition function:

$$P(x_t | x_{t-1}),$$

The function, $\boldsymbol{P}$, depends on the system used and the control input of the system. If the control input parameters are the changes in positions, the function, $\boldsymbol{P}$ will be an addition to the prior position $\boldsymbol{x_{k-1}}$ with noise, The predicted measurements are then fused with the current measurements using the ***Bayes' theorem*** to give the estimated measurements. In order to keep the prior map information for fusion, the transformation equation is required to transform the prior map to the same reference coordinate system as the current map due to movement of sensors.

$$P(x_t|o_{1:t}, u_{1:t}, m_t) = \sum_{m_{t-1}} P(o_t|x_t, m_t, u_{1:t}) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}|m_t, o_{1:t-1}, u_{1:t})/Z$$

Similarly, the map can be updated sequentially by

$$P(m_t|x_t, o_{1:t}, u_{1:t}) = \sum_{x_t} \sum_{m_t} P(m_t|x_t, m_{t-1}, o_t, u_{1:t}) P(m_{t-1}, x_t|o_{1:t-1}, m_{t-1}, u_{1:t})$$

4.4.1 Lidar SLAM

Compared to cameras and other sensors, lasers are significantly more precise. The output values from our laser sensor are a $\mathbf{2D_{x,y}}$ point cloud data. It works effectively for map construction with SLAM algorithms. The calculated movement estimated sequentially registering the point clouds is used for localizing the vehicle. To estimate the relative transformation between the point clouds, you can use registration algorithms such as **ICP** and **NDT**. Due to these challenges, localization for autonomous vehicles may involve fusing other measurements such as wheel odometry, and IMU data.

4.4.1 Grid Map

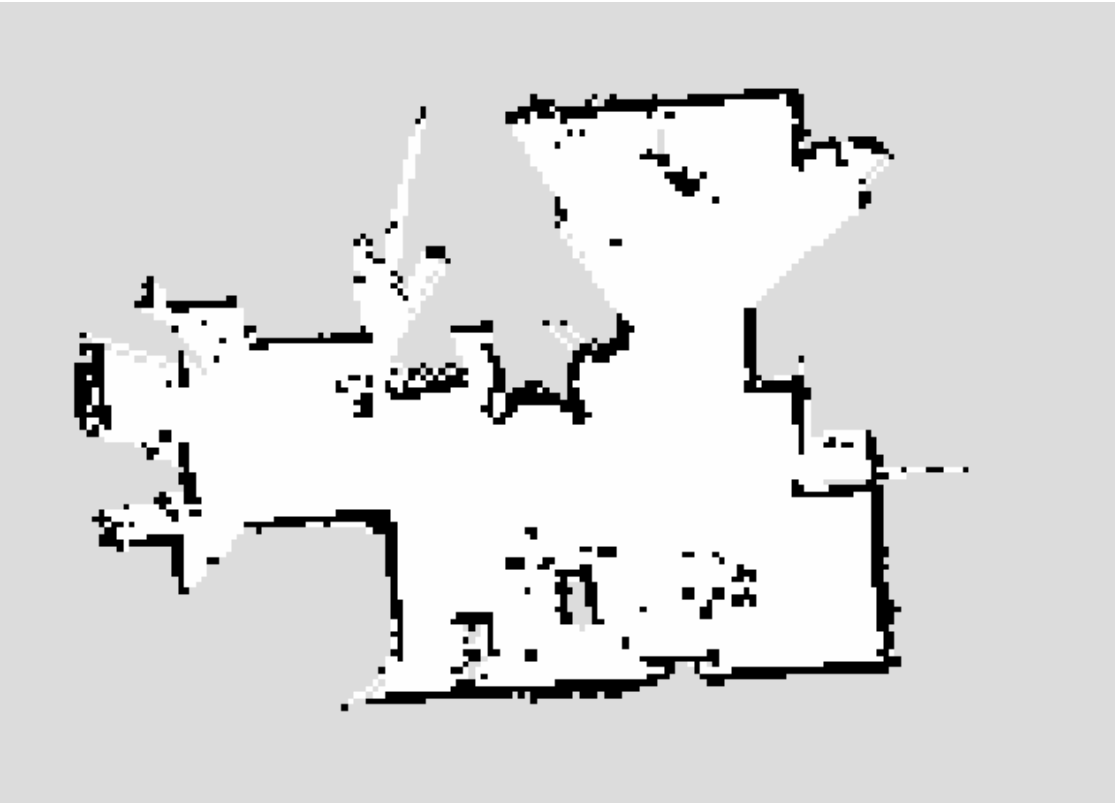


Figure 16 Grid Map

In an occupancy grid map, each cell is marked with a number that indicates the likelihood the cell contains an object.

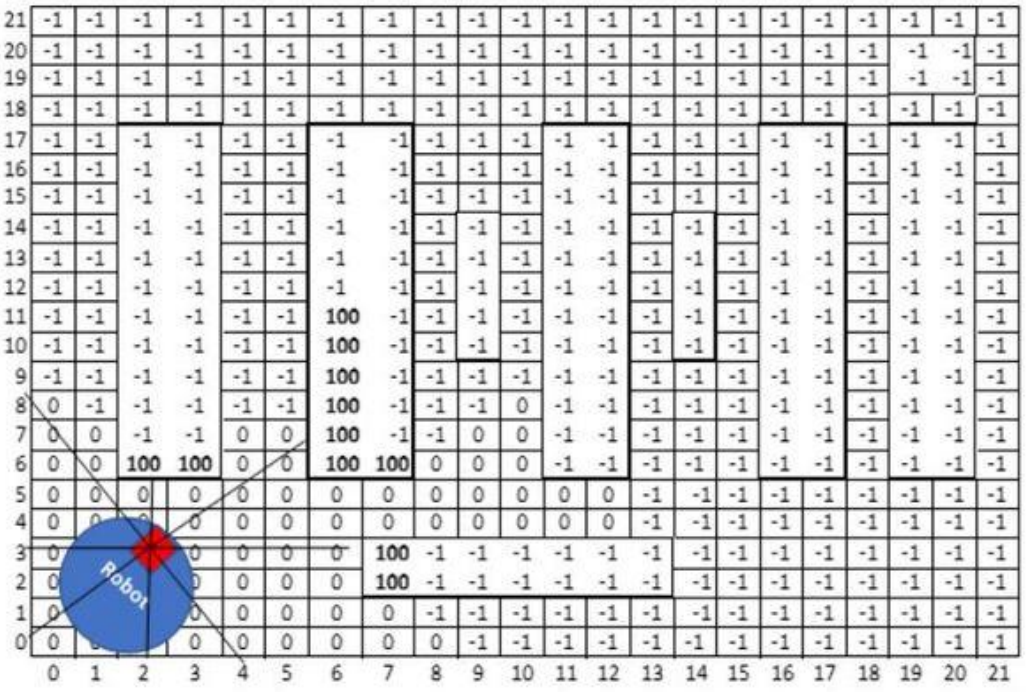


Figure 17 grid map occupancy values

The number is often **0** (free space) to **100** (100% likely occupied). Unscanned areas (i.e. by the LIDAR, ultrasonic sensor, or some other object detection sensor) would be marked **-1**.

4.4.1 Cartographer

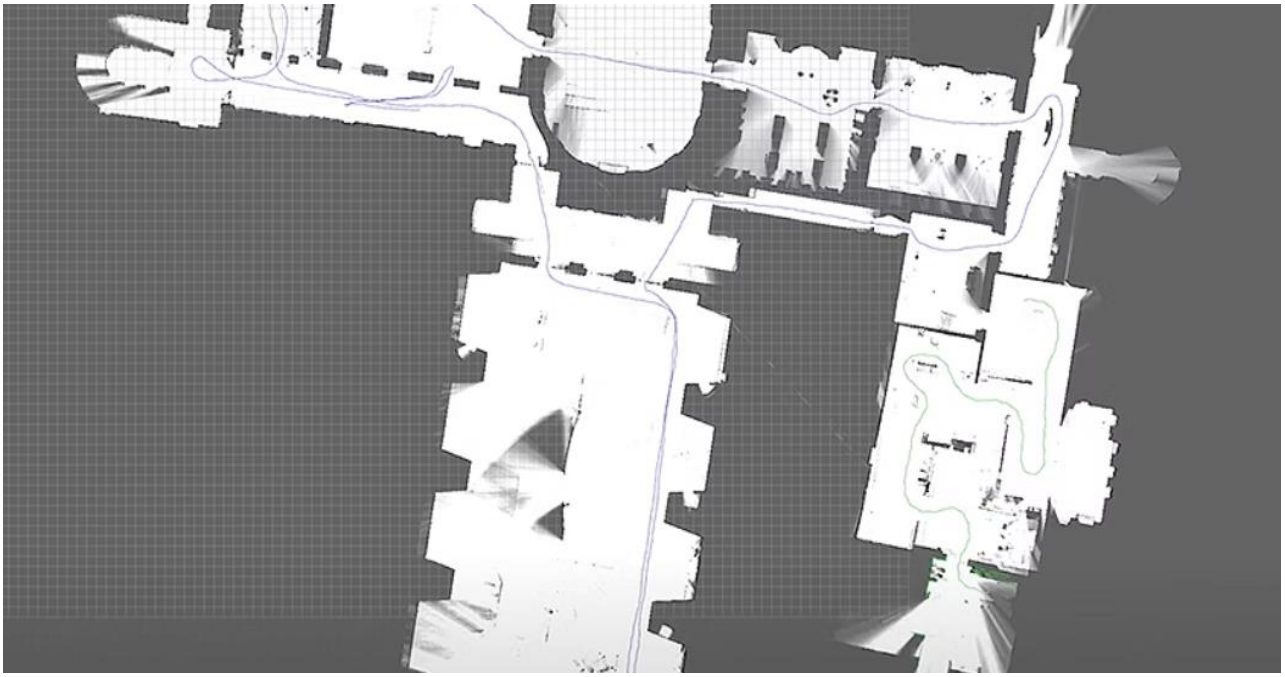


Figure 18 Cartographer

As sensor data comes in, the state of Cartographer algorithm evolves to stay the current best estimate of a robot's trajectory and surroundings. It serialize its internal state in a **.pbstream** file format which is then compressed into a **protobuf** file containing a snapshot of the data structures used by Cartographer internally. To run in real-time, it throws most of its sensor data away immediately and only works with a small subset of its input, the mapping used internally (and saved in **.pbstream** files) is then very rough. However, when the algorithm finishes and a best trajectory is established, it can be recombined a posteriori with the full sensors data to create a high-resolution map. Cartographer makes this kind of recombination possible using **cartographer_assets_writer**. The assets writer takes as input the original sensors data that has been used to perform SLAM in a ROS **.bag** file, a cartographer state captured while performing SLAM on this sensor data saved in a **.pbstream** file, the sensor extrinsic TF data from the bag or an URDF description, and a pipeline configuration, which is defined in a **.lua** file. The assets writer runs through the **.bag** data in batches with the trajectory found in the **.pbstream**. The pipeline can be used to color, filter and export SLAM point cloud data into a variety of formats. There are multiple of such points processing steps that can be interleaved in a pipeline - several ones are already available from **cartographer/io.**

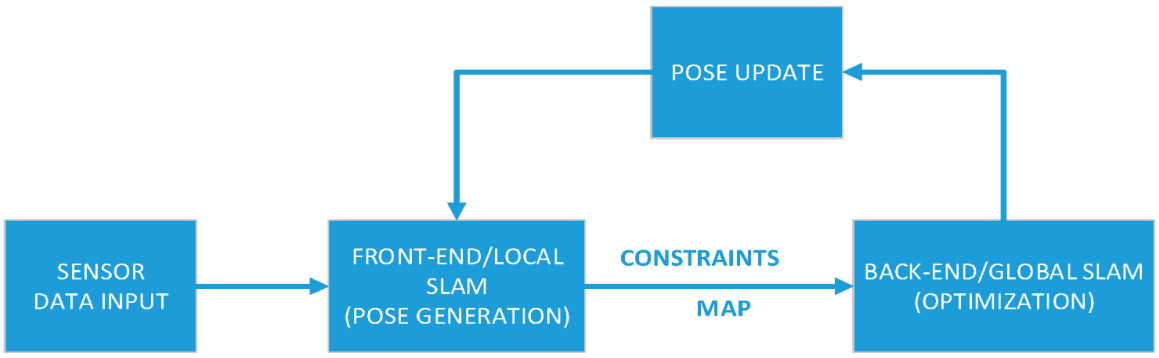


Figure 19 Cartographer SLAM method diagram

4.4.2 Mapping

- Global Map:

Scope: Entire environment

Purpose: Long-term navigation and consistent mapping.

Update: Incrementally as the agent explores.

- Local Map:

Scope: Immediate surroundings

Purpose: Real-time navigation and obstacle avoidance.

Update: Frequently as the agent moves.

- Integration Concept:

Local to Global: Local maps are aligned and merged into the global map for a cohesive representation.

- Global to Local: The global map helps refine and validate the local map for accuracy.

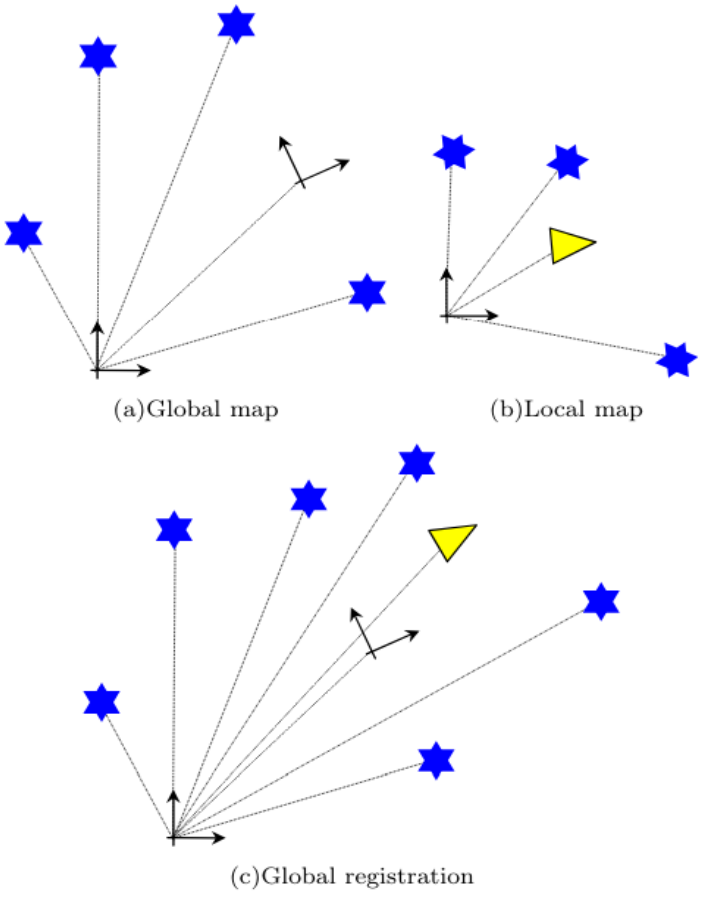


Figure 20 Mapping Types

4.4.3 EKF in SLAM

EKF

EKF is a state estimation algorithm that accounts for nonlinearities. used in SLAM to estimate the robot's position and map features simultaneously. By Predicting the state of a system and updates this prediction based on sensor measurements.

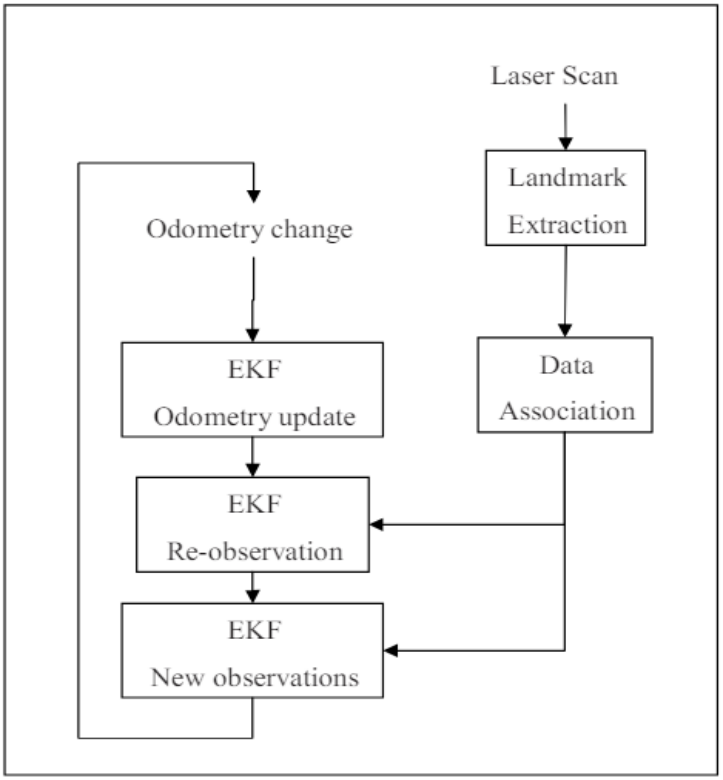


Figure 21 EKF in SLAM

Process

1. Prediction Step: Estimates the new state of the robot based on motion models.
2. Update Step: Corrects the estimate using sensor measurements to refine both the robot's position and the map.
3. Integration in SLAM
4. EKF-SLAM: A specific implementation of SLAM where EKF is used to handle the uncertainties in robot motion and sensor measurements.
5. Global Map: EKF maintains a global map by updating the state vector, which includes both the robot's pose and the landmarks' positions.
6. Consistency: Helps in maintaining the consistency of the map and the robot's location over time.

4.4.4 Particle Filter

Particle Filter: A sequential Monte Carlo method used for estimating the state of a system from noisy observations. It uses a set of samples (particles) to represent the probability distribution of the state. Each particle represents a possible state of the system, and the algorithm updates these particles based on motion models and sensor data. Particle filters are used in SLAM to estimate both the robot's pose (localization) and the map simultaneously.

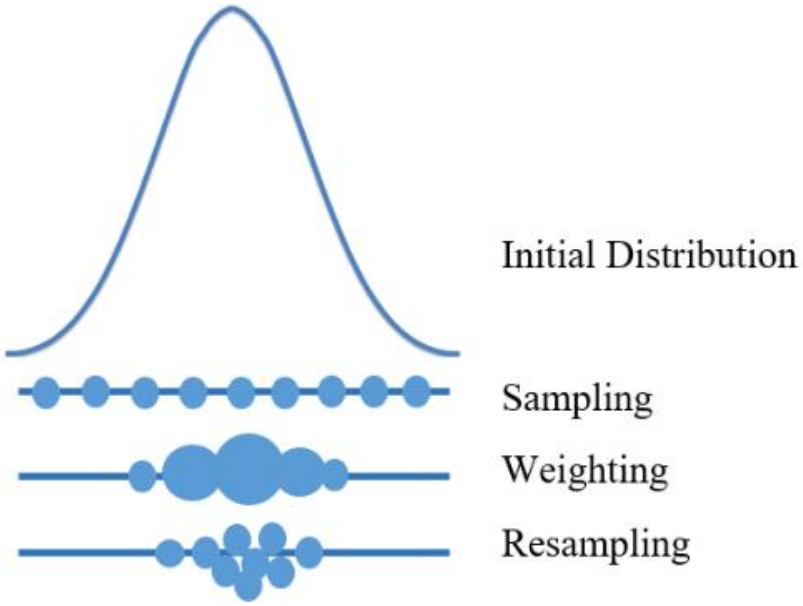


Figure 22 Particle Filter

Process

1. Prediction Step: Each particle is updated based on the robot's motion model.
2. Update Step: Particles are weighted according to how well the predicted observations match the actual sensor measurements.
3. Resampling: Particles with higher weights are more likely to be selected for the next iteration, focusing the computational effort on the most probable states.
4. Integration in SLAM
5. Particle Filter-SLAM: A specific implementation of SLAM where Particle Filters are used to represent the distribution of the robot's pose and the map.
6. Global Map: Particles collectively help in building and maintaining a global map, each particle containing a hypothesis of the map and the robot’s pose.
7. Adaptability: Handles non-linearities and multi-modal distributions more effectively than EKF, making it suitable for more complex environments.

4.5 Localization

4.5.1 AMCL

AMCL is an algorithm based on particle filter to help robots determine its precise location within a known map.

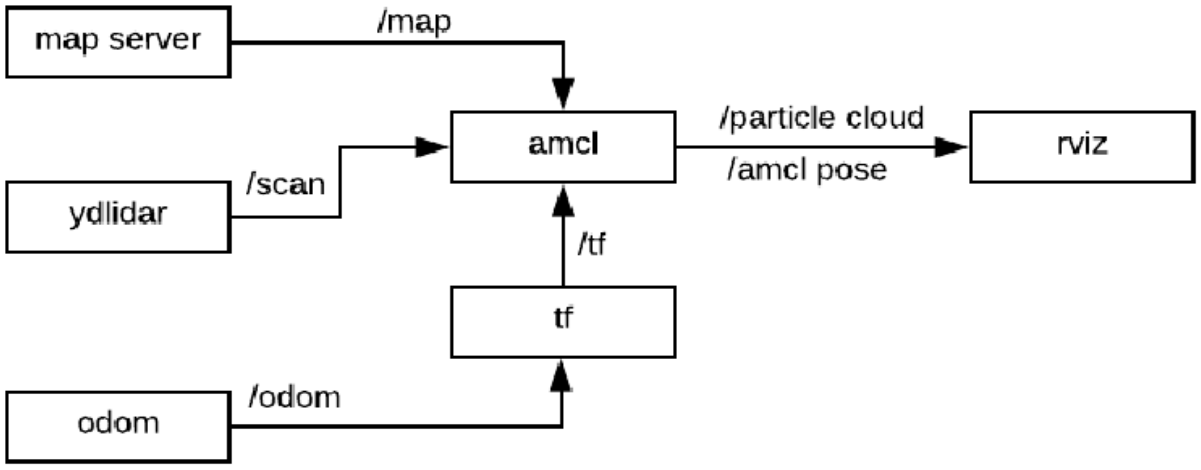


Figure 23 AMCL ROS framework

4.5.1 Practical Description of AMCL:

1. Initialization:

AMCL starts with an initial belief of the robot's pose and orientation on the map. This belief is represented using a set of particles, each particle being a hypothesis of where the robot might be.

2. Sensing and Motion Updates:

Motion Update: When the robot moves, it predicts its new pose using odometry information. Each particle is updated according to this predicted motion.

Sensing Update: After sensing the environment using a laser scanner to detect landmarks or features, each particle's weight is updated based on how well the robot's sensor readings match what is expected in the map. Particles that better match the sensor data are assigned higher weights.

3. Resampling:

Particles with higher weights (i.e., better pose estimates) have a greater chance of being selected in the resampling process. This ensures that the particle set evolves over time to better represent the robot's actual pose.

4. Adaptive Particle Management:

AMCL dynamically adjusts the number of particles based on the uncertainty in the robot's pose. When uncertainty is high, more particles are used to cover a wider range of possible poses. Conversely, when uncertainty is low, fewer particles suffice to maintain accurate localization.

4.6 Sensor Fusion

Sensor fusion is a technique of integrating multiple sensors data to obtain accurate reliable understanding of the environment or state of a system. It aims to mitigate individual sensor limitations to enhance performance. This concept is crucial where precise data is essential.

4.6.1 sensors' limitations

- **IMU**

Drift: IMUs accumulate errors over time, leading to drift in pose and orientation estimates.
Noise: IMUs are susceptible to noise, especially in high-vibration environments.
Bias: Gyroscopes and accelerometers can have biases that change over time or with temperature, requiring frequent calibration.

- **Encoder**

Slip and Skid: encoders can be inaccurate when the wheels slip or skid, as the actual distance traveled is not equal to the wheel rotations.
Resolution: Encoders provide discrete measurements, which can limit accuracy.

- **LiDAR**

Line-of-Sight: it requires a clear line of sight and can be obstructed by, dust, fog, or rain.
Limited Range: The effective range of LiDAR can be limited, affecting its ability to detect distant objects.

4.6.2 EKF

The algorithm works in a two-step process Prediction and Estimation. In the (prediction step), the filter produces an estimate of the current state, with its error probabilities. At the beginning, initial conditions are used. Once the next sensor reading has entered the filter, Estimates are updated using a weighted average of the raw readings. **k** represents the present

state and **k-1** represents the previous state**.** $\boldsymbol{X(k)}$ denotes current position, $\boldsymbol{X(k-1)}$ denotes previous position.

$\boldsymbol{u(k)}$ represents the previous velocity and the acceleration, $\boldsymbol{B(k)}$ represents the directions, $\boldsymbol{F(k)}$ represents the orientation, $\boldsymbol{w(k)}$ is used to indicate the unknown forces like friction.

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n\left(z_n - \hat{x}_{n,n-1}\right)$$
$$p_{n,n} = (1 - K_n)p_{n,n-1}$$

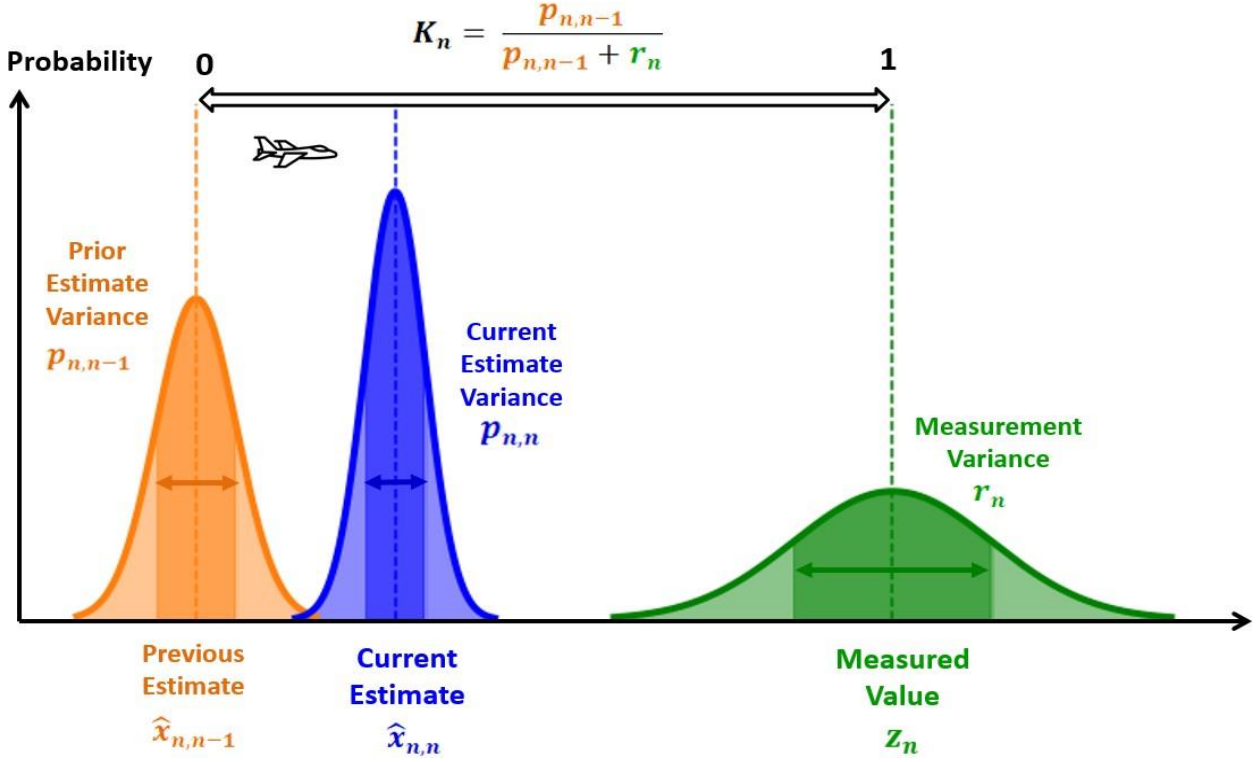


Figure 24 Kalman Fiter.

Algorithm Integration: Sensor fusion algorithms, such as Kalman filters, integrate data from Combining Lidar with encoders or IMU with encoders to obtain an accurate estimation of the robot's position and motion through sophisticated algorithms enables robots to navigate in their environment more accurately, overcoming the limitations of individual sensors.

4.6.2 What's the difference between EKF and KF?

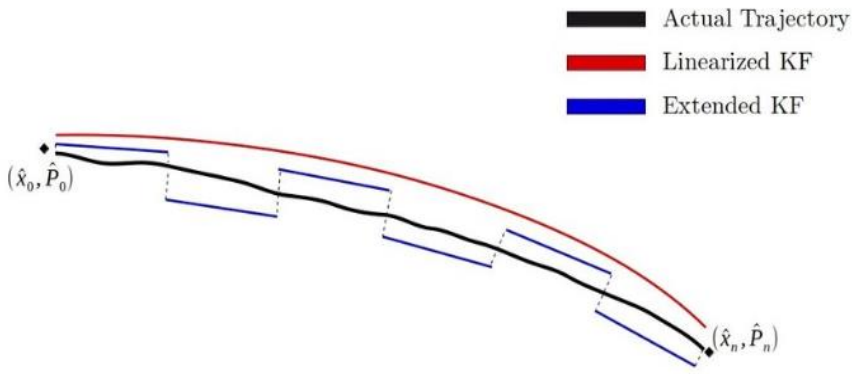


Figure 25 EKF Vs EK

KF and the EKF are both used for state estimation, The Kalman Filter is ideal for systems with linear dynamics, it provides optimal efficient estimates and commonly used in applications like object tracking with linear models. The EKF extends the Kalman Filter to handle nonlinear systems by approximating them with linear models around the current estimate, using Jacobian matrices. This makes the EKF suitable for complex nonlinear applications, such as autonomous navigation, Unfortunately, it's more computationally intensive due to the need for linearization.

4.6.2 What's the equations behind the EKF?

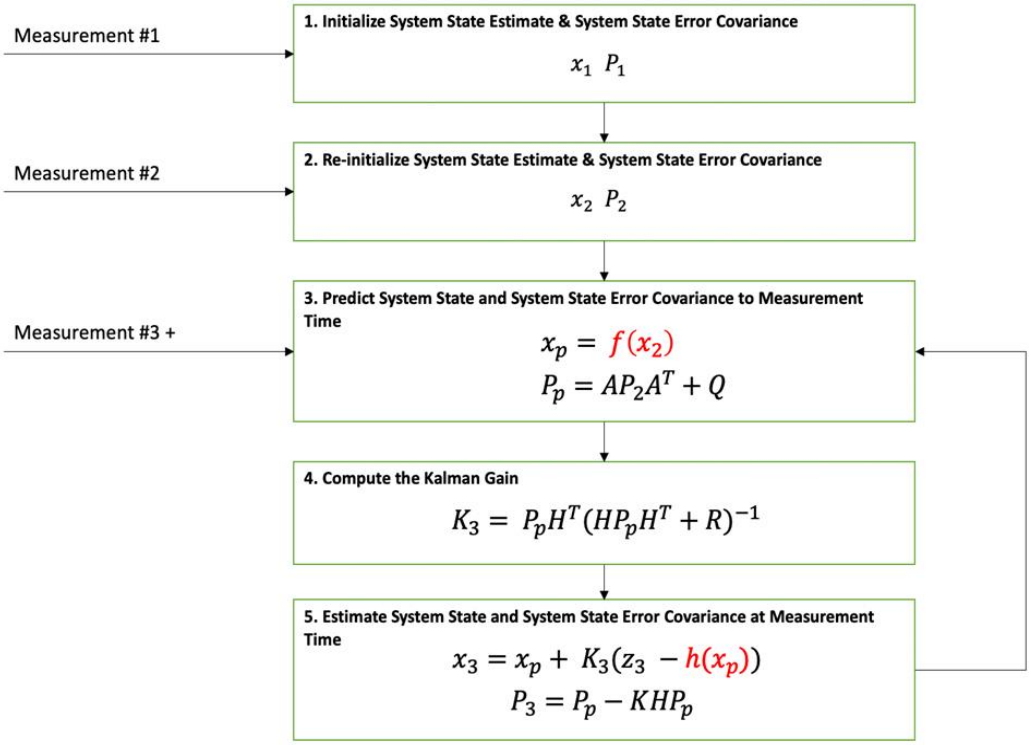


Figure 26 Equations Behind EKF

4.7 Navigation

4.7.1 Dijkstra Algorithm

Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph. The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.

Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path. The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

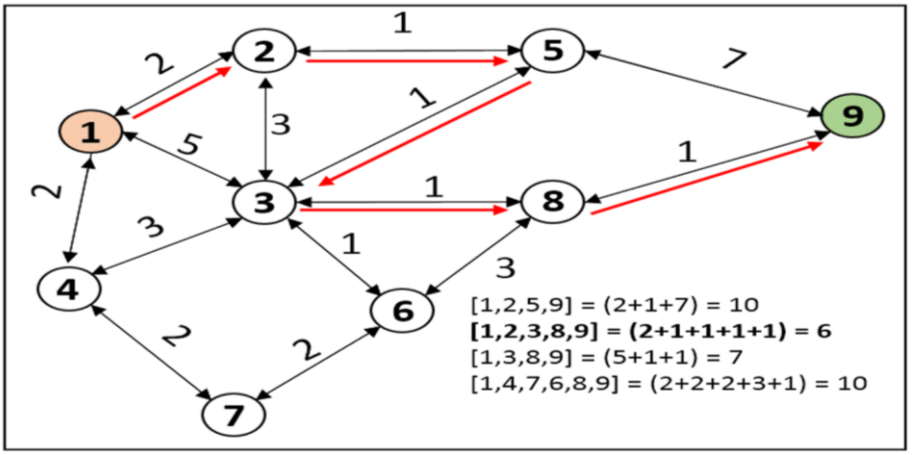
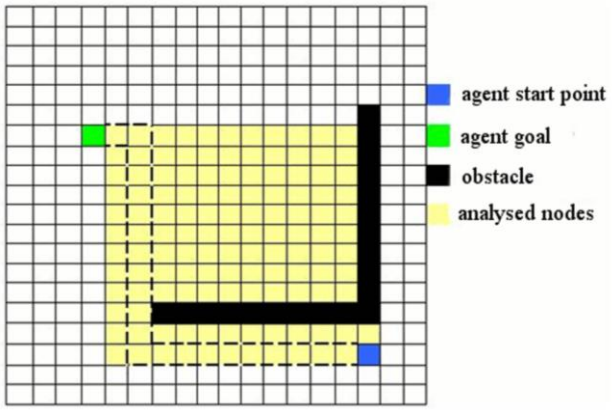


Figure 27 Dijkstra Algorithm

4.7.1 A* Algorithm

Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. This fact is cleared in detail in below sections. And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).



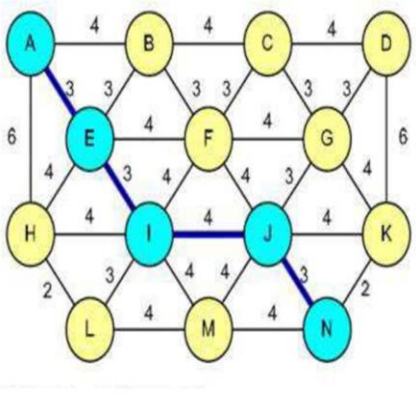


Figure 28 A Algorithm*

Explanation Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue. What A* Search Algorithm does is that at each step it picks the node according to a value- 'f' which is a parameter equal to the sum of two other parameters – 'g' and 'h'. At each step it picks the node/cell having the lowest 'f', and processes that node/cell. We define 'g' and 'h' as shown below…

- **g:** the movement cost to move from the starting point to a given square on the grid, following the path generated to get there
- **h:** the estimated movement cost to move from that given square on the grid to the destination. This is often referred to as heuristic, which is nothing but a kind of smart guess. We really do not know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.).

5. Co Bot Project Resources

5.1 Software tools

5.1.1 ROS 2

Robot Operating System (ROS) is an open-source middleware framework. It provides a set of tools and libraries for building and managing complex robot applications. It includes a middleware layer **(DDS)** for communication It supports multiple programming languages.

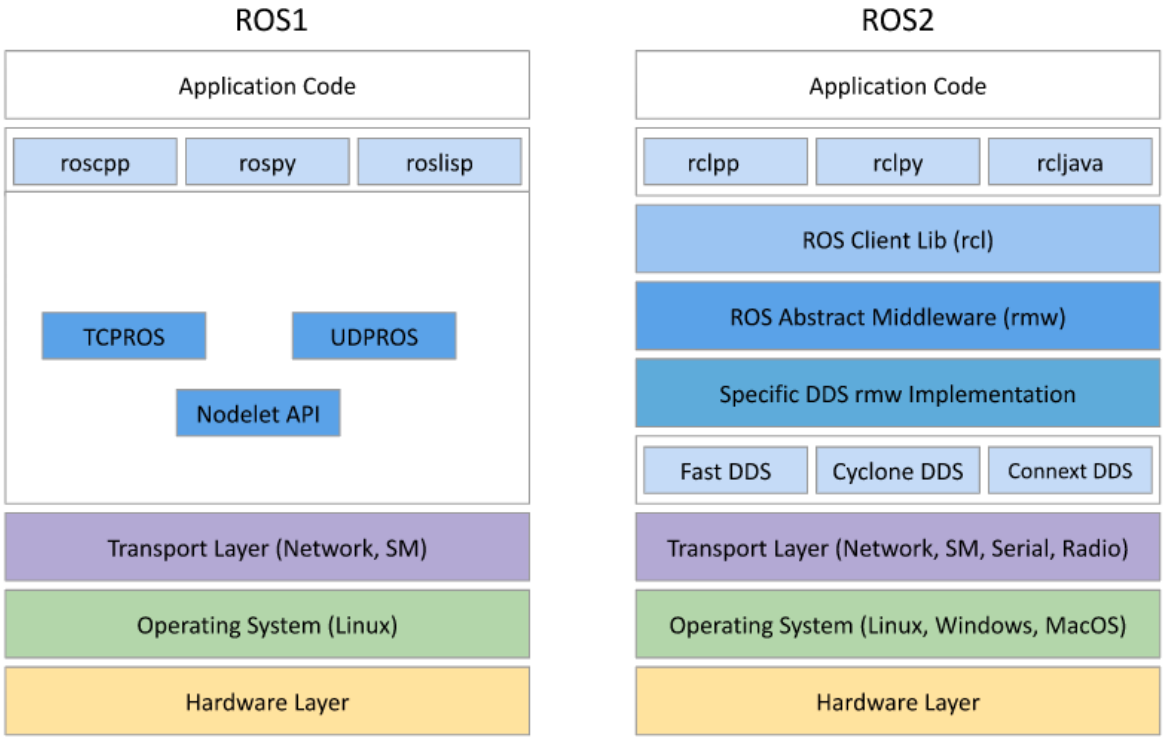


Figure 29 comparison between ROS & ROS2

Communication: ROS 2 utilizes the DDS middleware for communication with multiple applications, providing a flexible and scalable messaging server.

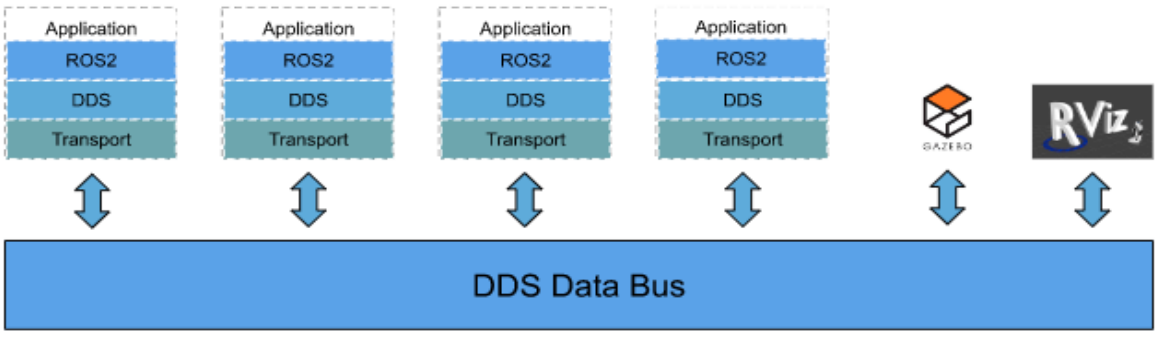


Figure 30 Communication in ROS 2

Real-time Capabilities: ROS 2 aims to improve real-time capabilities with stricter timing requirements to control, predict and visualize motion of robotics.

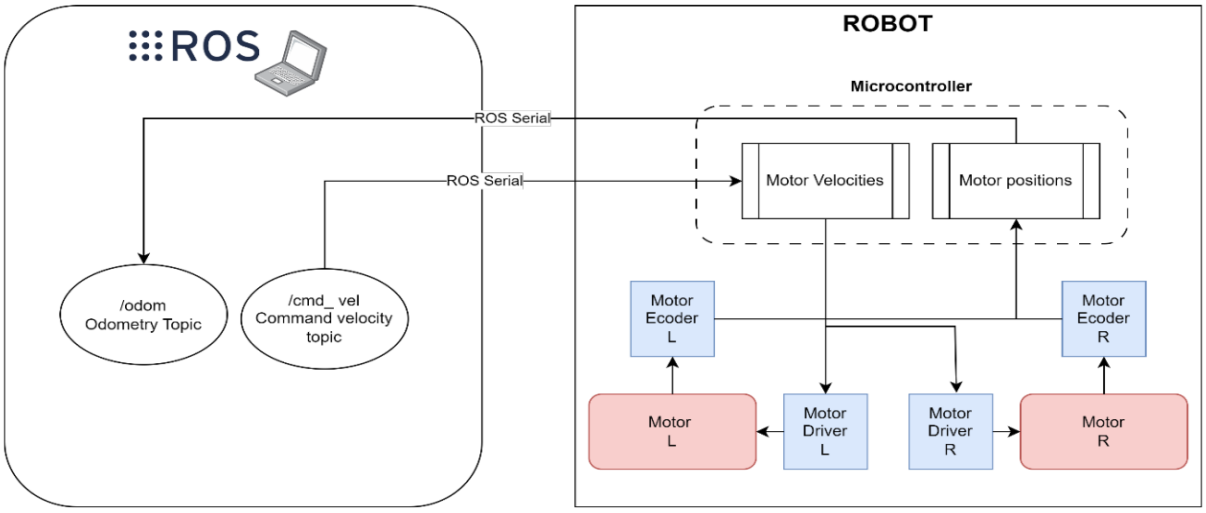


Figure 31 ROS2 Real-time Capabilities

Multilingual: ROS 2 supports multiple programming languages, including C++, Python, and others, making it accessible to a broader audience.

Development: ROS 2 is actively developed and maintained by the Open Robotics community, and it benefits from a large ecosystem of packages and libraries.

5.1.2 Micro-Ros

micro-ROS is a robotic framework targeting embedded robot components with extremely constrained computational resources. micro-ROS follows the ROS 2 architecture and makes use of its middleware pluggability to use **DDS-XRCE**, suitable for microcontrollers. **micro-ROS** empowers these devices to become first-class participants of ROS ecosystem.

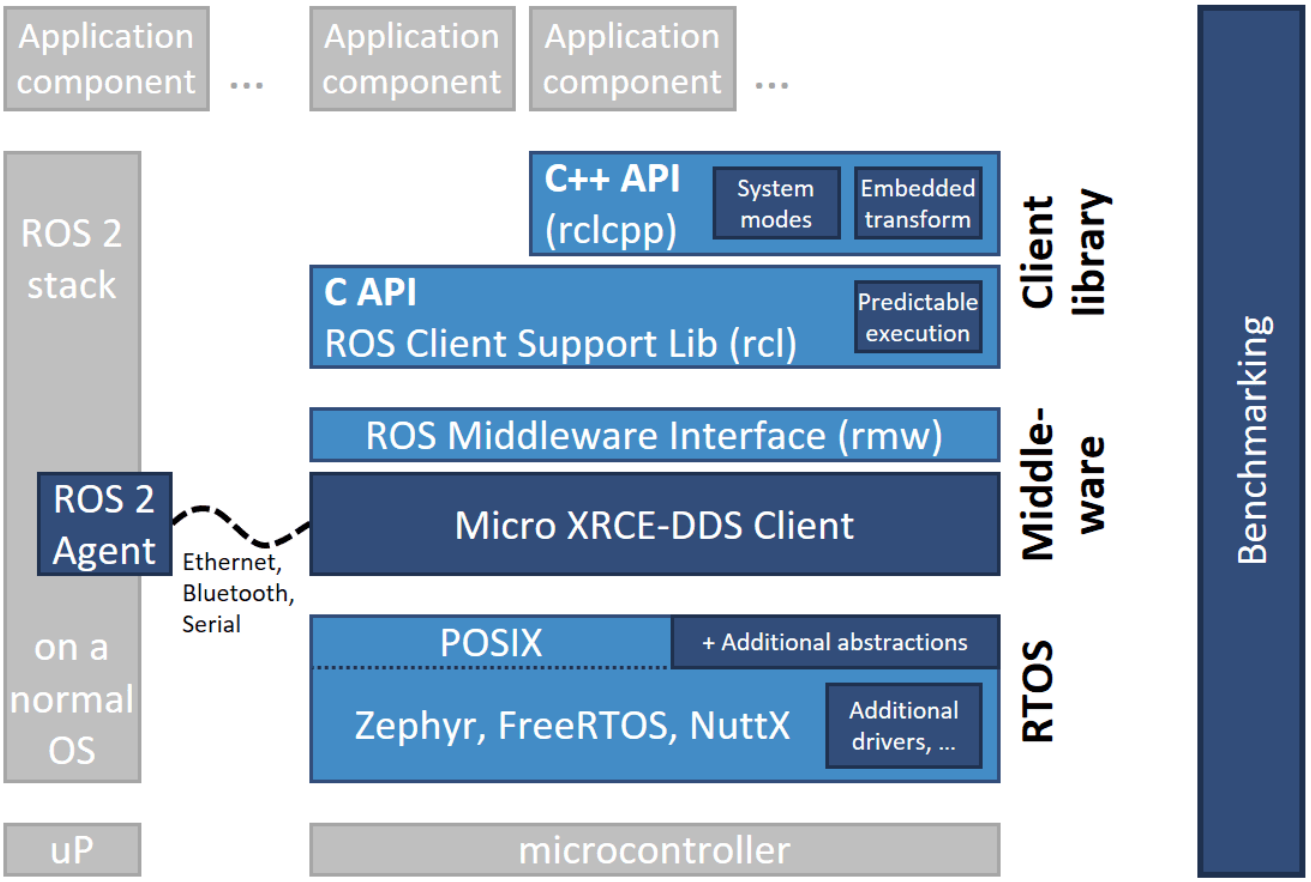


Figure 32 Micro-ROS

- The dark blue components shown in the architecture are developed specifically for micro-ROS.
- The light blue components, on the other hand, are taken from the standard ROS 2 stack.
- micro-ROS is supported by the RTOSes FreeRTOS, Zephyr, NuttX, in addition to Linux and Windows. All three RTOSes are downloaded natively with the micro-ROS build system, and can be chosen when creating a new firmware workspace.

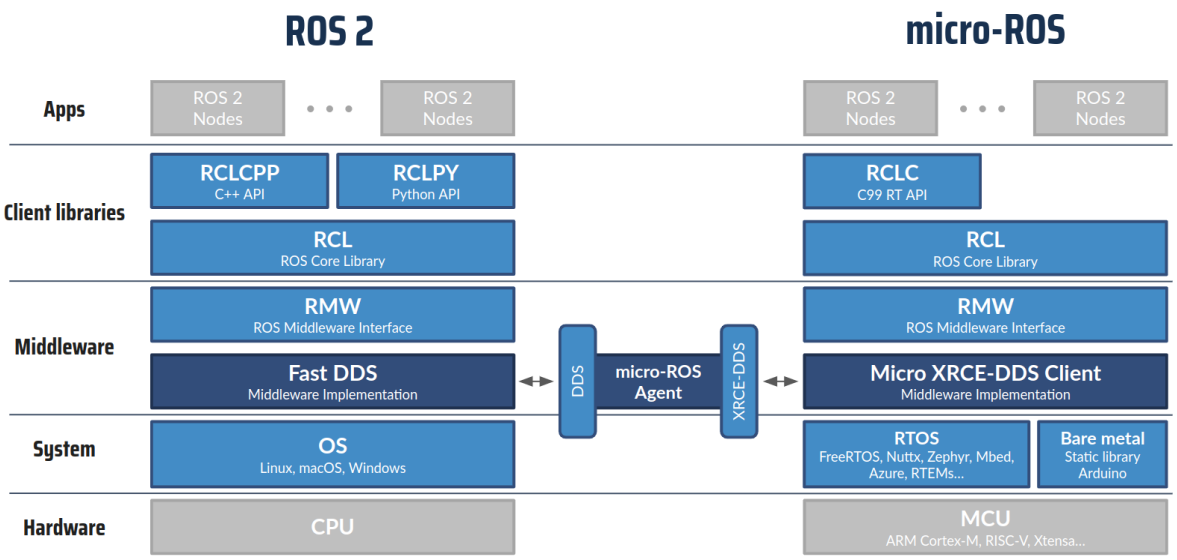


Figure 33 comparison between micro-ros and ros2

- ROS 2 is designed for general-purpose robots and aims to provide a flexible and scalable framework for developing robotic applications. It offers robust communication tools, real-time capabilities, and enhanced security features compared to its predecessor, ROS 1. ROS 2 is suitable for a wide range of robots, from simple hobbyist projects to complex industrial automation systems, and it can run on powerful hardware such as desktop computers and embedded systems. On the other hand, micro-ROS is tailored for microcontroller-based environments, which are typically resource-constrained with limited processing power, memory, and storage. It brings the power of ROS 2 to these small devices, allowing developers to implement ROS 2 functionalities on microcontrollers. micro-ROS is ideal for applications where size, power consumption, and cost are critical factors, such as in embedded systems, IoT devices, and wearable technology.
- In order to better understand the differences between micro-ROS and other approaches, the project has conducted an analysis of several related approaches like **ROSSerial**. In order to provide an accurate comparison. The whole stack that is necessary to implement ROS features in microcontrollers that are lacking an operating system like Linux or Windows, which are prerequisites to operate **ROS2**.

- In summary, while ROS 2 targets general-purpose and more powerful robotic systems, micro-ROS focuses on extending ROS 2 capabilities to resource-limited microcontroller environments, enabling the development of highly integrated and efficient robotic solutions.

5.1.3 Gazebo

A robotic simulation environment integrated with ROS, designed to simulate robots, sensors, and environments in 3D. It provides a platform for testing and developing robot algorithms without the need for physical hardware. It's an essential tool for developers to evaluate robotic virtually.

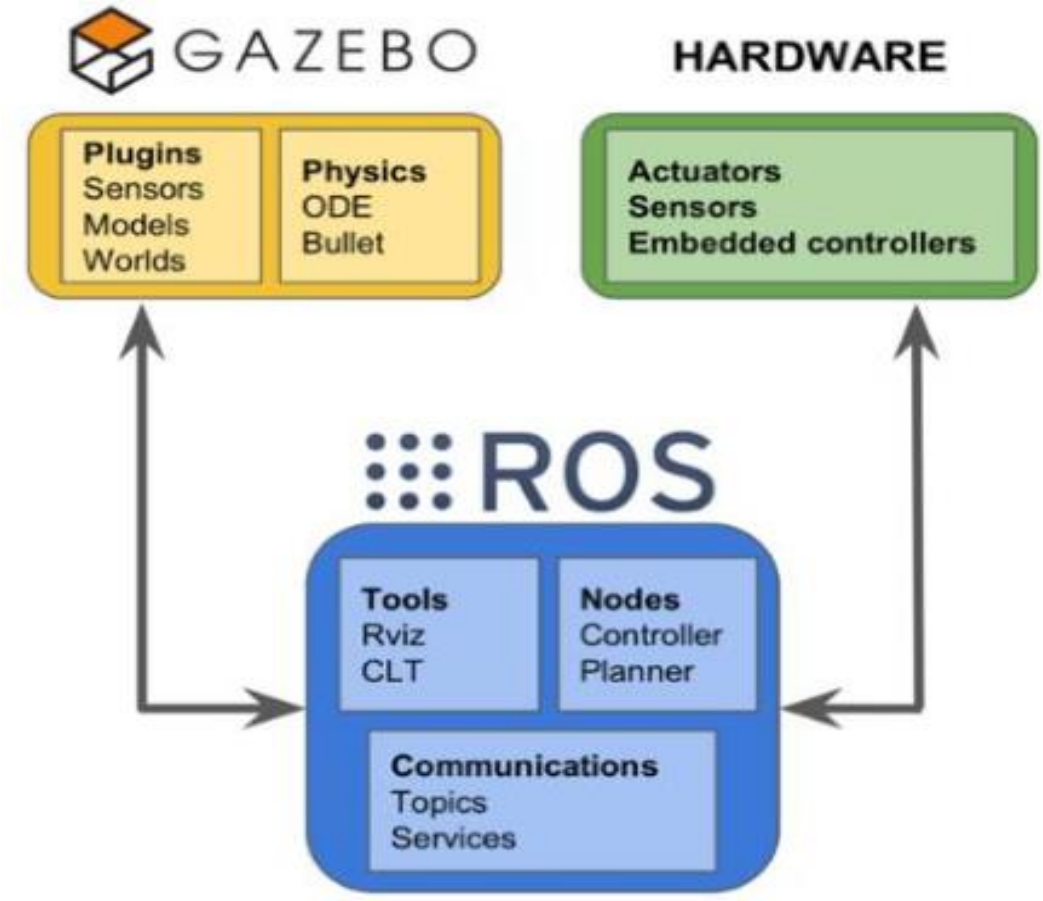


Figure 34 GAZEBO with ROS block diagram

5.1.4 RVIZ

A visualization tool for ROS that displays sensor data, robot model states, and environment maps. Helps in debugging, monitoring, and visualizing the robot's perception and state. It features Interactive markers, 3D rendering, plugin support for various data types (e.g., laser scans)

5.2 Electrical Hardware Tools

Each mobile co-bot car has these components individually:

- PR-LIDAR

Figure 35 Lidar

- Raspberry pi 4

Figure 36 Raspberry pi 4

- 2 digital continuous servo motors 15kg.cm with encoders

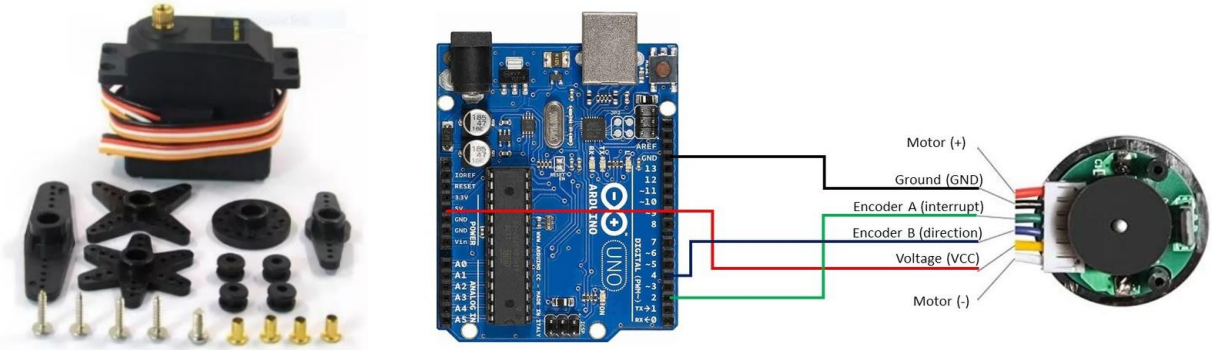


Figure 37 Servo Motor

- Power bank (10 Ah, 5V)

Figure 38 Power bank

- Esp32

Figure 39 Esp32 board

- Imu MPU5060 (Inertial Measurement Unit)

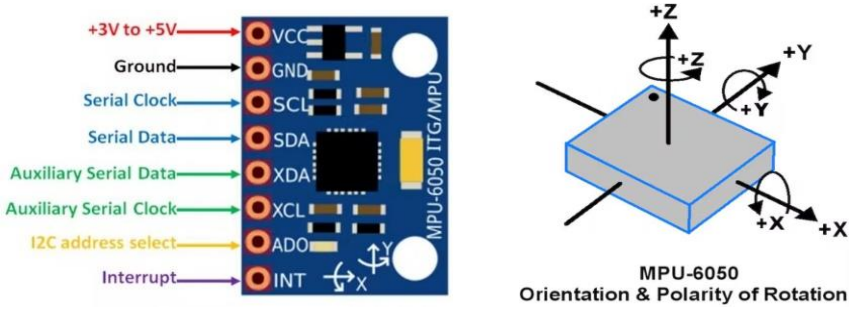


Figure 40 IMU degrees of freedom

5.3 Mechanical Hardware Components

- 2 caster wheels

Figure 41 Caster Wheel

- 2 wheels

Figure 42 wheel

- Black Acrylic 3mm sheet for Chassis Body

Figure 43 Acrylic Sheet

5.4 Wiring Diagram

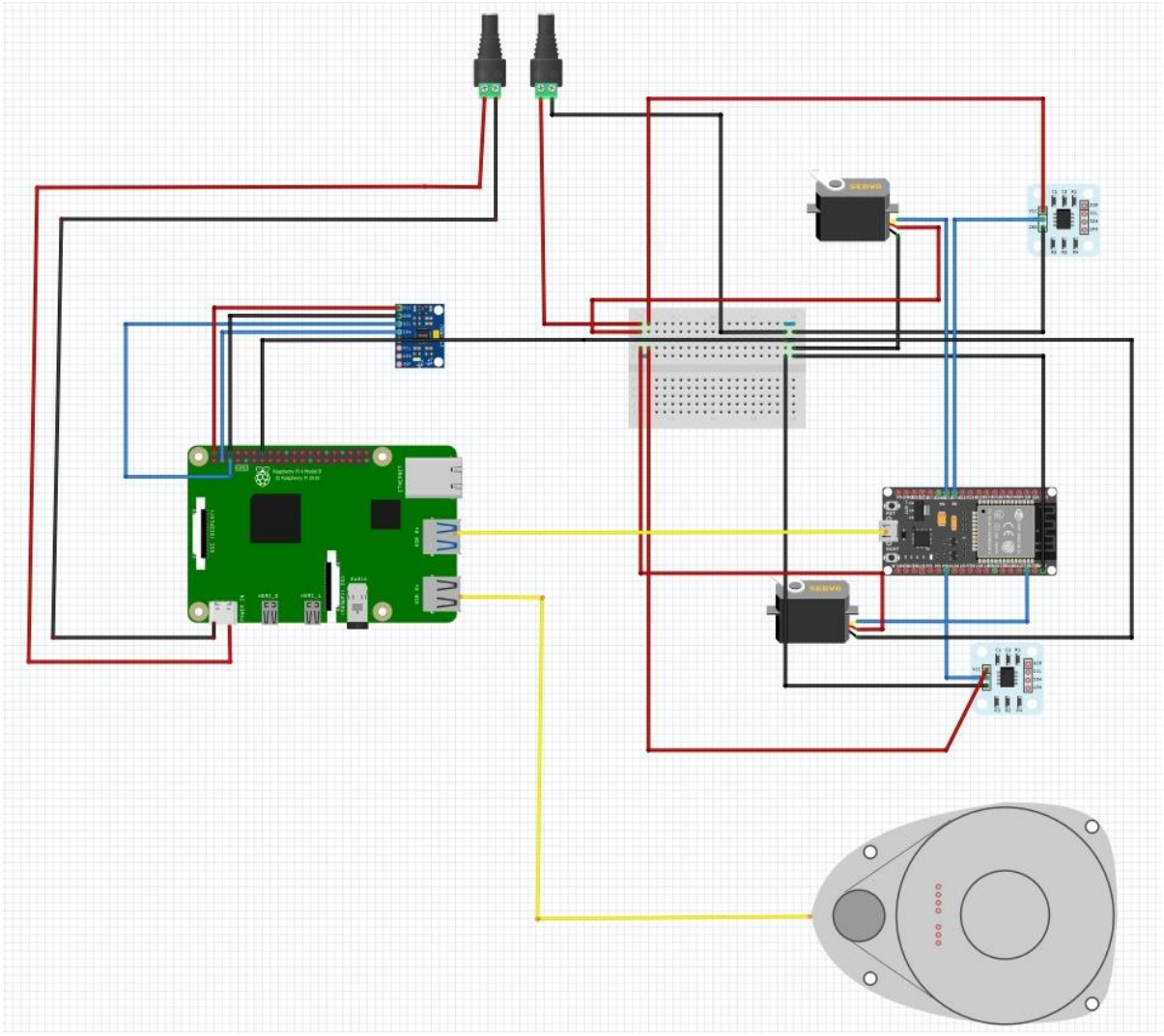


Figure 44 Wiring Diagram

5.5 Power Specifications

- Our Power source is a 10 Ah, 37 Wh Power Bank for each car

Rated Capacity: 10000 mAh, 5V, 2.1 A

USB1/USB2: 5 V, 2 A

Type C input: 5 V, 2.1 A

Total Output: USB1 + USB2

- We use 1 USB output for the servo motor and the other USB output for the Raspberry-Pi, it's recommended operating conditions are:

1. Raspberry-Pi

Recommended Voltage: 5V

Recommended Current: 3A

2. Servo Motor

Operating Voltage: 4.8V

Input Voltage: 4V – 4.8V

Rated Current: 450 mA

6. Mechanical Design

6.1 Design

Solid 3D visualization

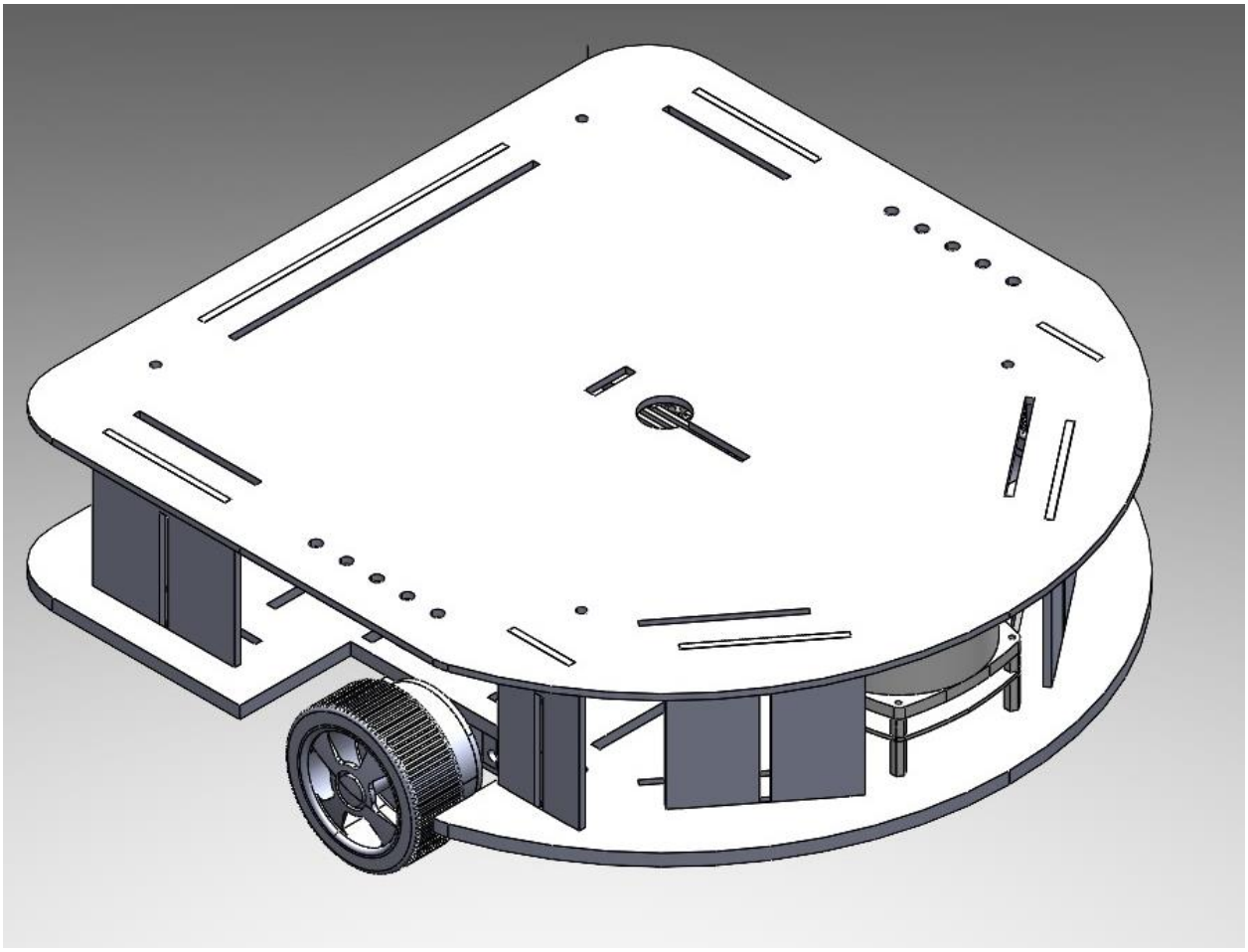


Figure 45 side view of co-bot

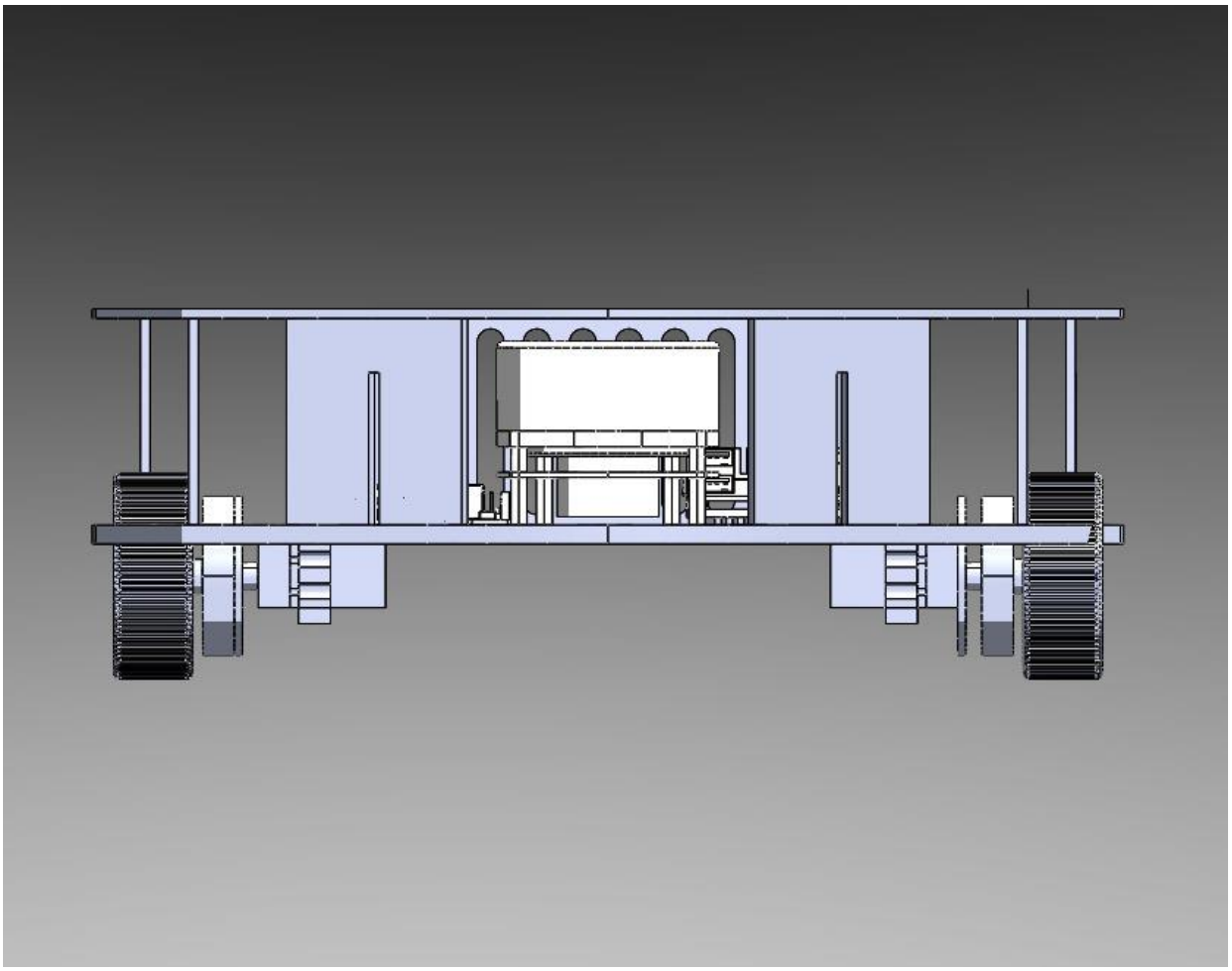


Figure 46 frontal view of co-bot

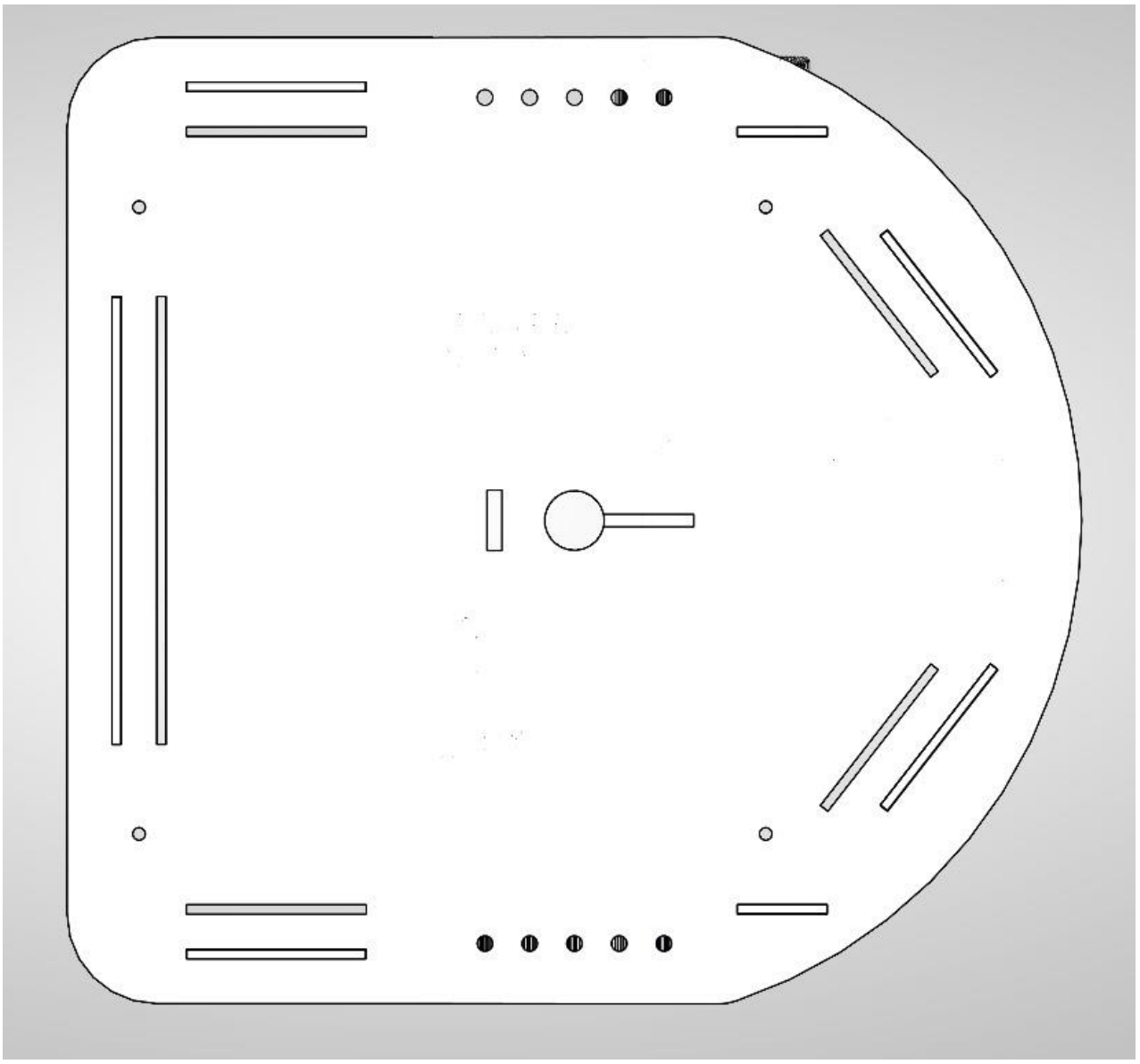


Figure 47 Upper view of co-bot

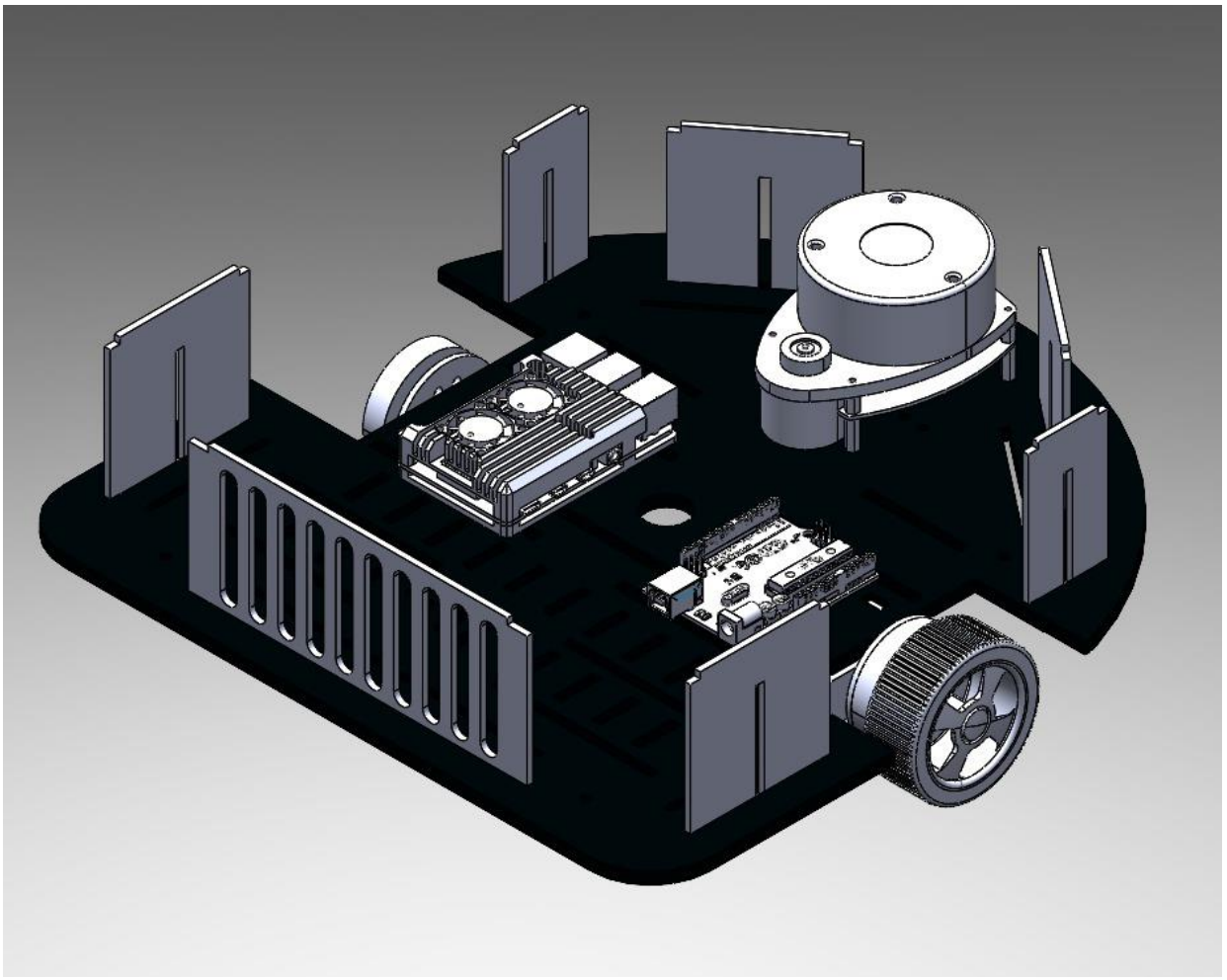


Figure 48 view of components on the co-bot

6.2 Manufacturing

1- Using laser cutting on acrylic to manufacture plates and corners

2- 3D printing to make servo motor connecting shafts with wheels

Figure 49 upper view of co-bot

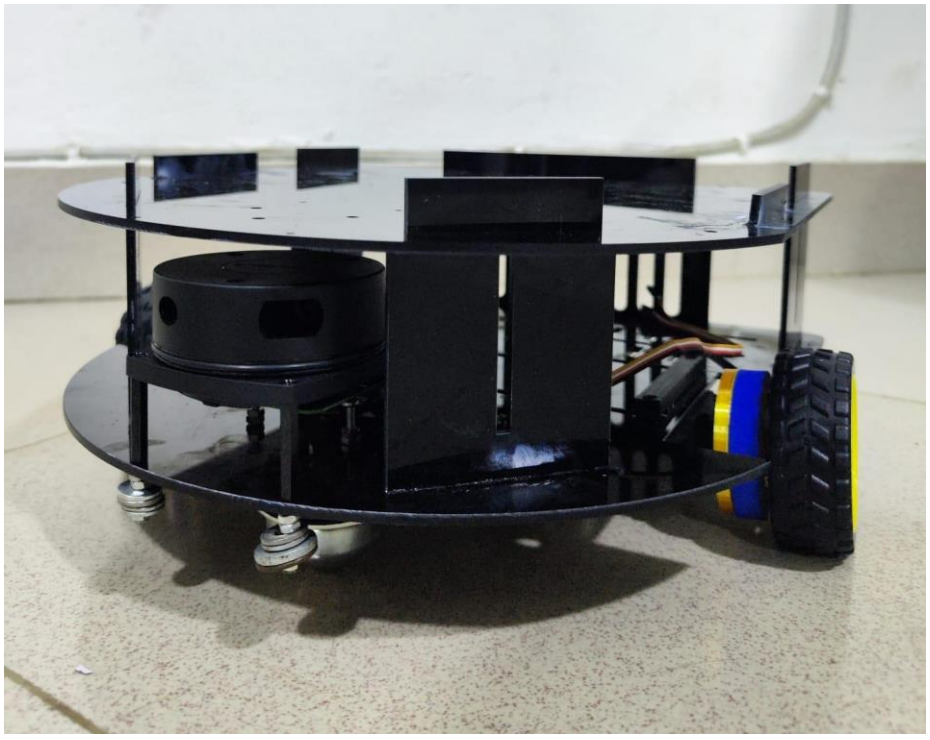


Figure 50 frontal view of co-bot

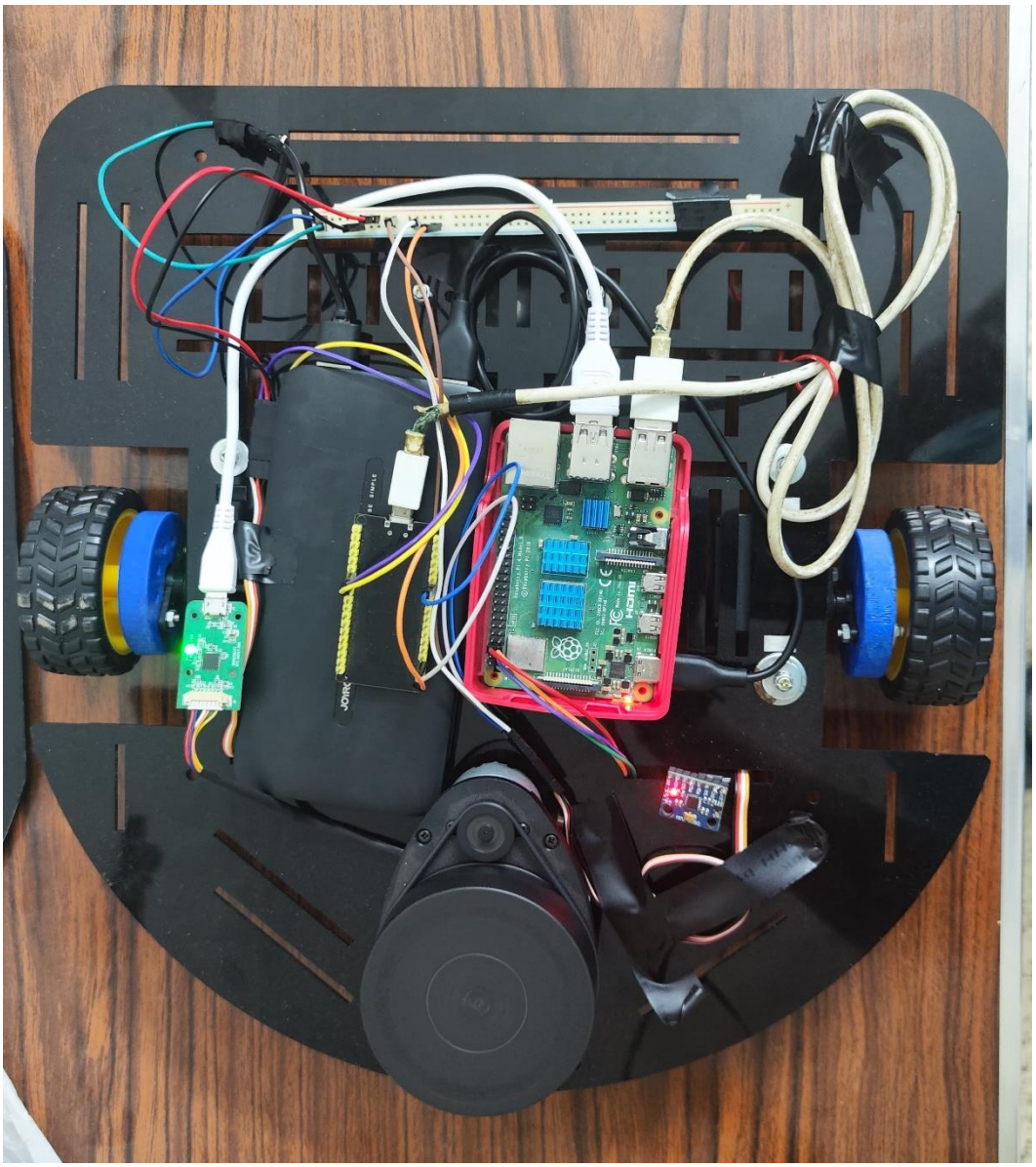


Figure 51 co-bot interior assembled

6.3 Payload

Payload refers to the part of a vehicle's load, It is essentially the "useful" load that a vehicle is designed to transport, excluding its own weight and the fuel or energy it needs for propulsion. understanding and optimizing payload capacity is crucial for the economic, efficient, and successful operation of transportation vehicles.

- In order to measure payload, you need to go through the following steps

1. First weigh the car with all its essential electrical and mechanical components.
Our car estimated weight was 1.5 kg.

2. Run the car forward with a constant speed and load a specific weight on it, try (3 kg), (4k) and (5 kg)

3. Measure the maximum weight the car can carry without significant loss of speed or control. This will involve trial and error, adding different weights and testing performance at each weight added. Ensure the car can start, stop, and turn with the payload.
Max efficient weight was **4 kg**

4. Payload = total car weight (loaded) + car weight without (unloaded)
Our Payload is equal **4 kg**

5. To consider the effect of friction and wheel slipping divide by a safety factor
Payload after a safety factor of (**1.15**) equals **3.47 kg**

6.4 Stress Analysis

After estimating the payload, using the **SolidWorks** software we apply our stress analysis to our co-bot. Stress analysis is crucial because it ensures the safety, durability, and reliability of structures and materials by identifying potential points of failure and optimizing design to withstand loads and stresses.

Here's a more detailed explanation of the stress analysis process

- **First Principal Strain** This is the largest normal strain in the material. It occurs along the axis where the material experiences the maximum elongation or compression.
- **Second Principal Strain** This is the intermediate normal strain. It is neither the maximum nor the minimum but falls in between. In two-dimensional stress states (plane stress or plane strain), this component may not be present or may be zero.
- **Third Principal Strain** This is the smallest normal strain in the material. It occurs along the axis where the material experiences the least elongation or compression.
- **Negative stress** stands for **compression stress** of the material, while **Positive stress** stands for **strain stress**

6.4.1 1’st and 3’d principal

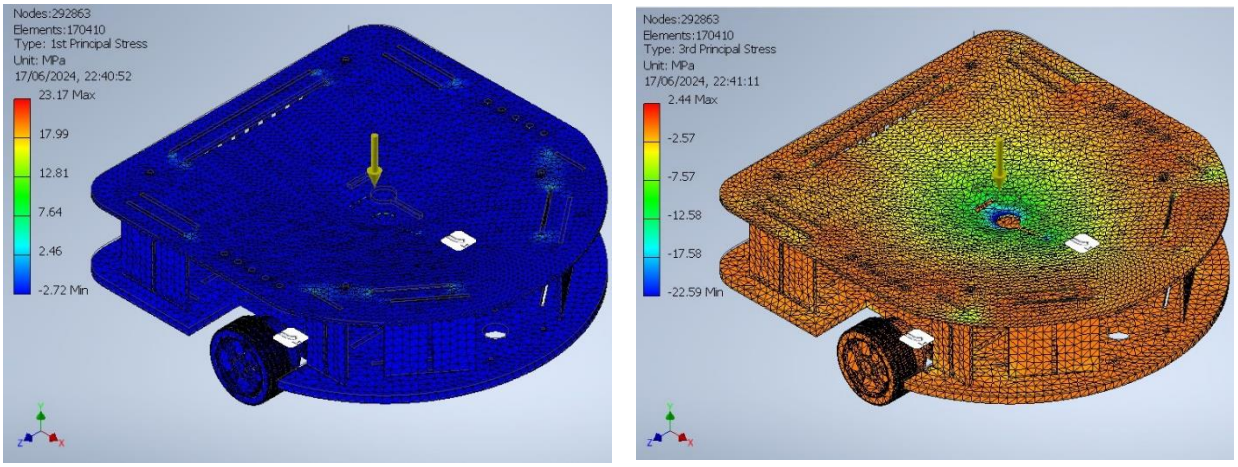


Figure 52 1’st and 3’d principal

6.4.2 Stresses of all the (x, y, z) planes

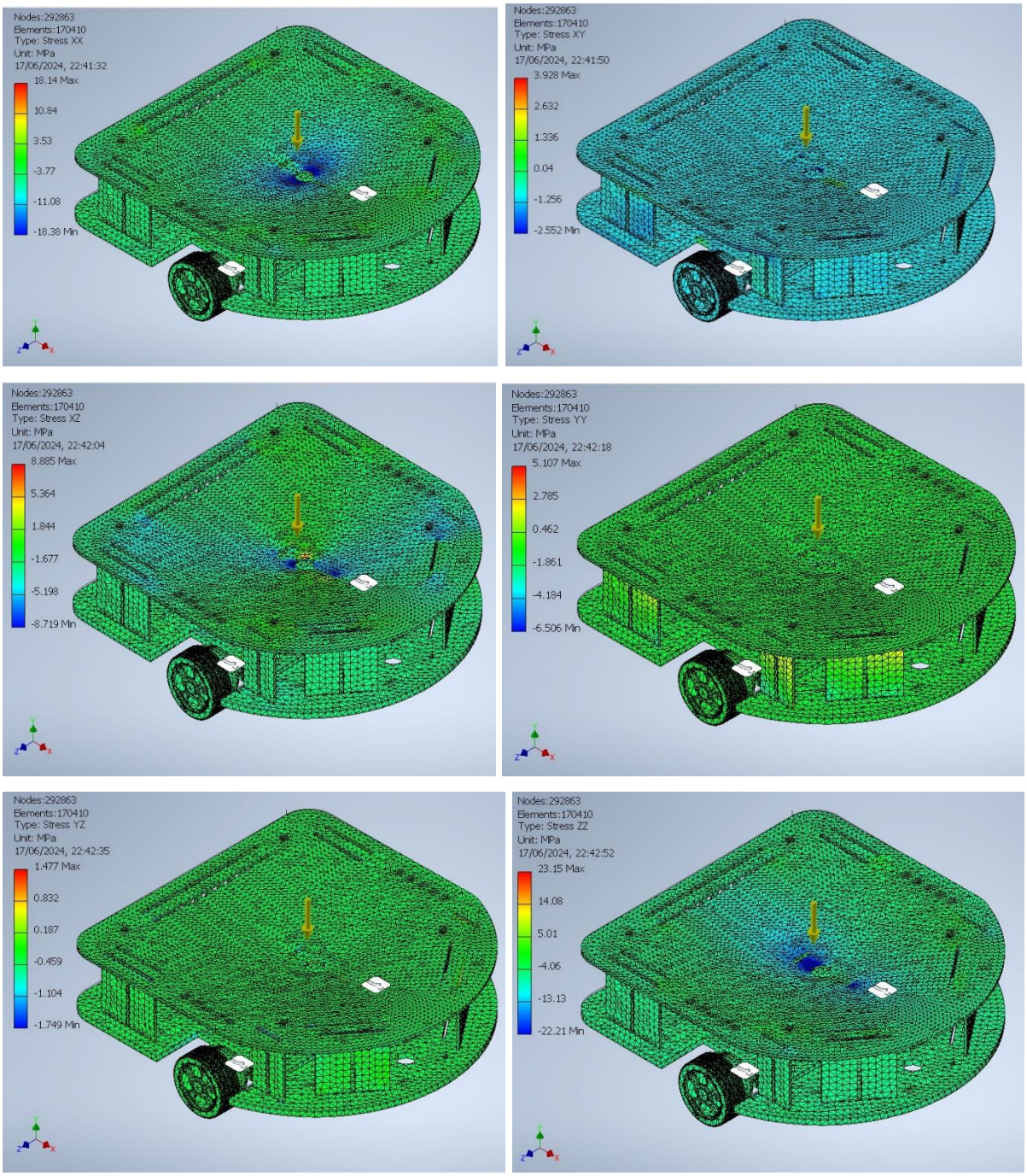


Figure 53 6.5.2 Stresses of all the (x,y,z) planes

6.4.3 Displacement in the 3 directions

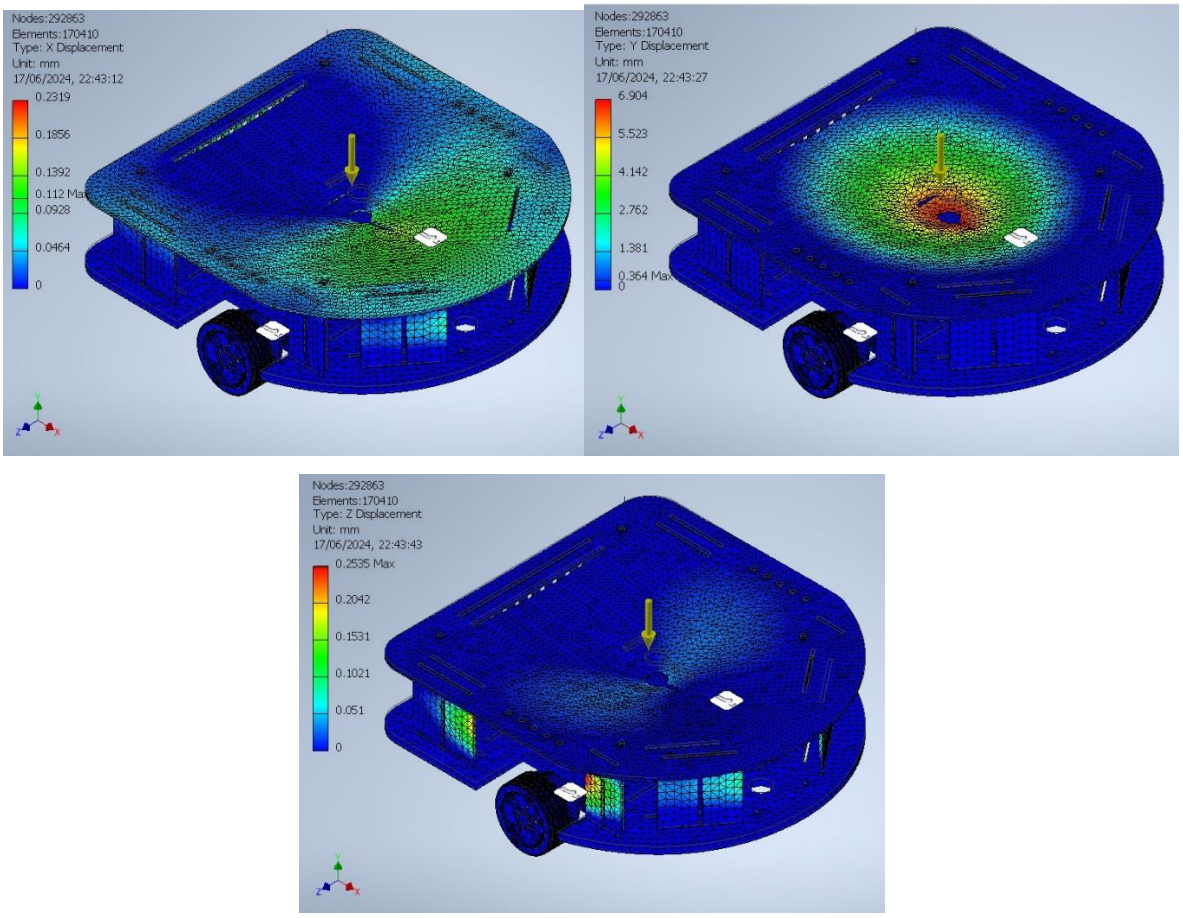


Figure 54 Displacement in the 3 direction

Commentary

The stress analysis results indicate that the design successfully supported the applied load, demonstrating excellent structural integrity. The analysis shows that the material and design can handle the expected stresses, ensuring safety and reliability for its intended use. This confirms that the design is both robust and efficient, meeting all required performance criteria.

7. Software Implementation

7.1 URDF (Universal Robot Description Format):

A URDF (Universal Robot Description Format) file is an XML file that describes what a robot should look like in real life. It contains the complete physical description of the robot.

The body of a robot consists of two components:

1. Links
2. Joints

Links are the rigid pieces of a robot. They are the “bones”.

Links are connected to each other by joints. Joints are the pieces of the robot that move, enabling motion between connected links.

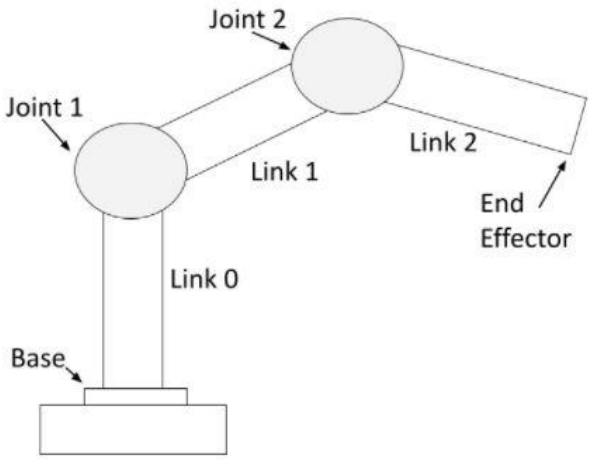


Figure 55 Robot Links and Joints

For a mobile robot with LIDAR, links and joints look like this:

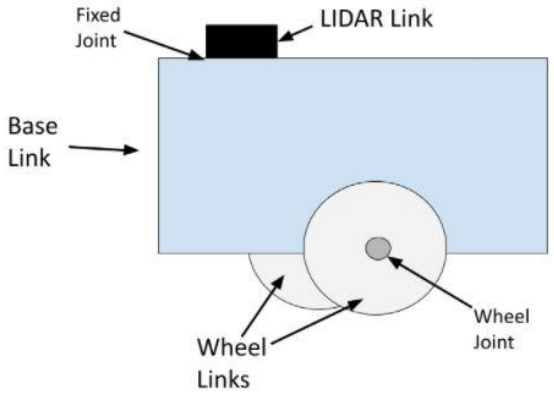


Figure 56 mobile robot with LIDAR

The wheel joints are **revolute joints**. Revolute joints cause rotational motion. The wheel joints in the photo connect the wheel link to the base link.
Fixed joints have no motion at all. You can see that the LIDAR is connected to the base of the robot via a fixed joint.

7.2 Xacro (XML, Macros):

Xacro (XML Macros) is a tool used in the Robot Operating System (ROS) for generating parameterized XML files. It is specifically designed to simplify the creation and maintenance of complex XML files, which are commonly used in robot description formats like URDF (Unified Robot Description Format).

6.2.1 Key Features of Xacro:

- **Parameterization**: Xacro allows you to use parameters within XML files. This means you can define variables and reuse them throughout the file, making it easier to change values in one place rather than multiple locations.
- **Macros**: You can define reusable blocks of XML code called macros. These macros can be instantiated multiple times with different parameters, reducing redundancy and simplifying the XML structure.
- **Inclusion**: Xacro supports the inclusion of other Xacro files, enabling modularity and reuse of code across different robot descriptions.

6.2.2 Benefits of Using Xacro:

- **Reduced Redundancy**: By defining macros and parameters, you can significantly reduce the amount of repeated XML code.
- **Easier Maintenance**: Parameterization makes it easier to update values across the file. Changing a parameter in one place updates all instances where it's used.
- **Modularity**: Inclusion of other Xacro files supports modular design, allowing you to break down the robot description into manageable parts.

6.3 Install Important ROS 2 Packages:

```
sudo apt install ros-humble-joint-state-publisher-gui
sudo apt install ros-humble-xacro
```

The output after launching the robot:

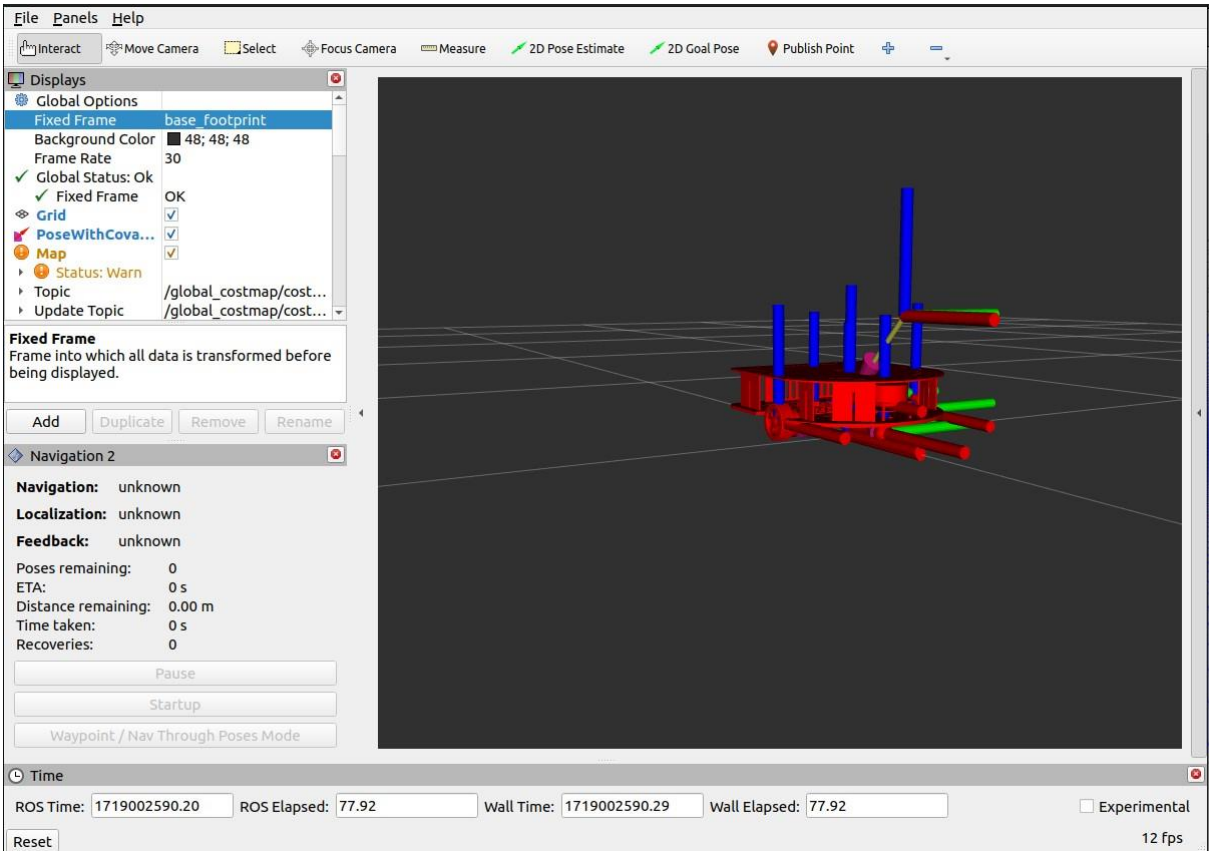


Figure 57 Launching the Robot's URDF in RVIZ

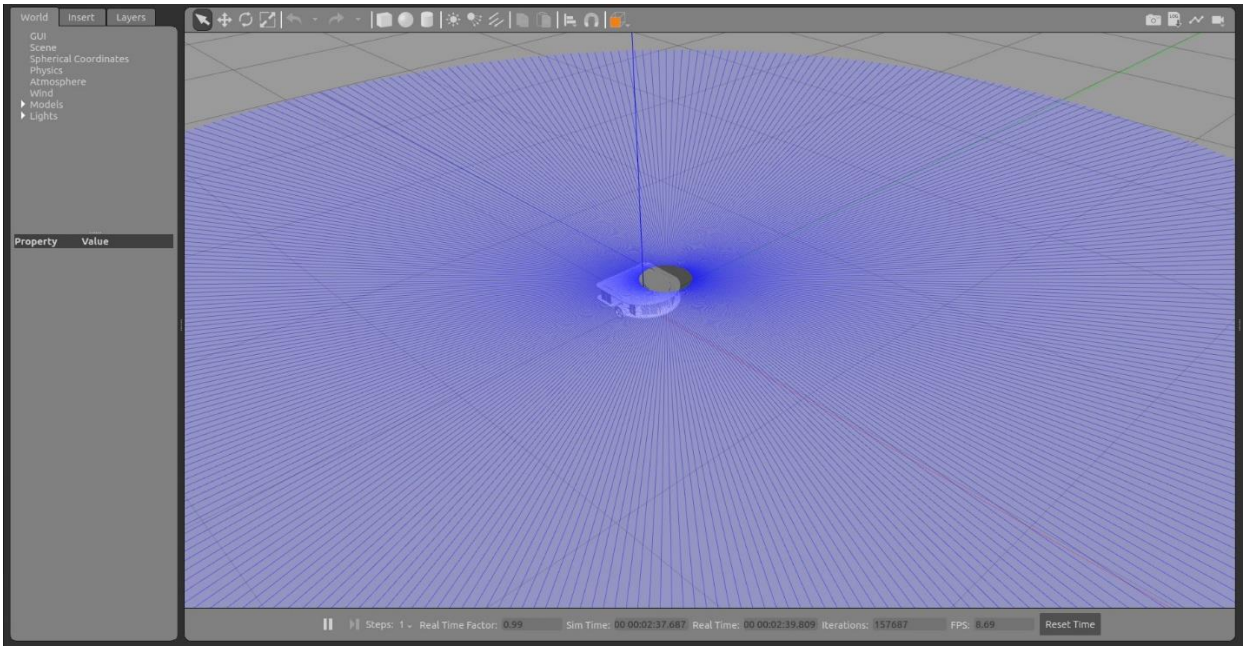


Figure 58 Launching the Robot's URDF in Gazebo

View the Coordinate Frames:

- install the necessary software.

```
sudo apt install ros-foxy-tf2-tools
```

- Check out the coordinate frames:

```
ros2 run tf2_tools view_frames.py
```

6.4 Results

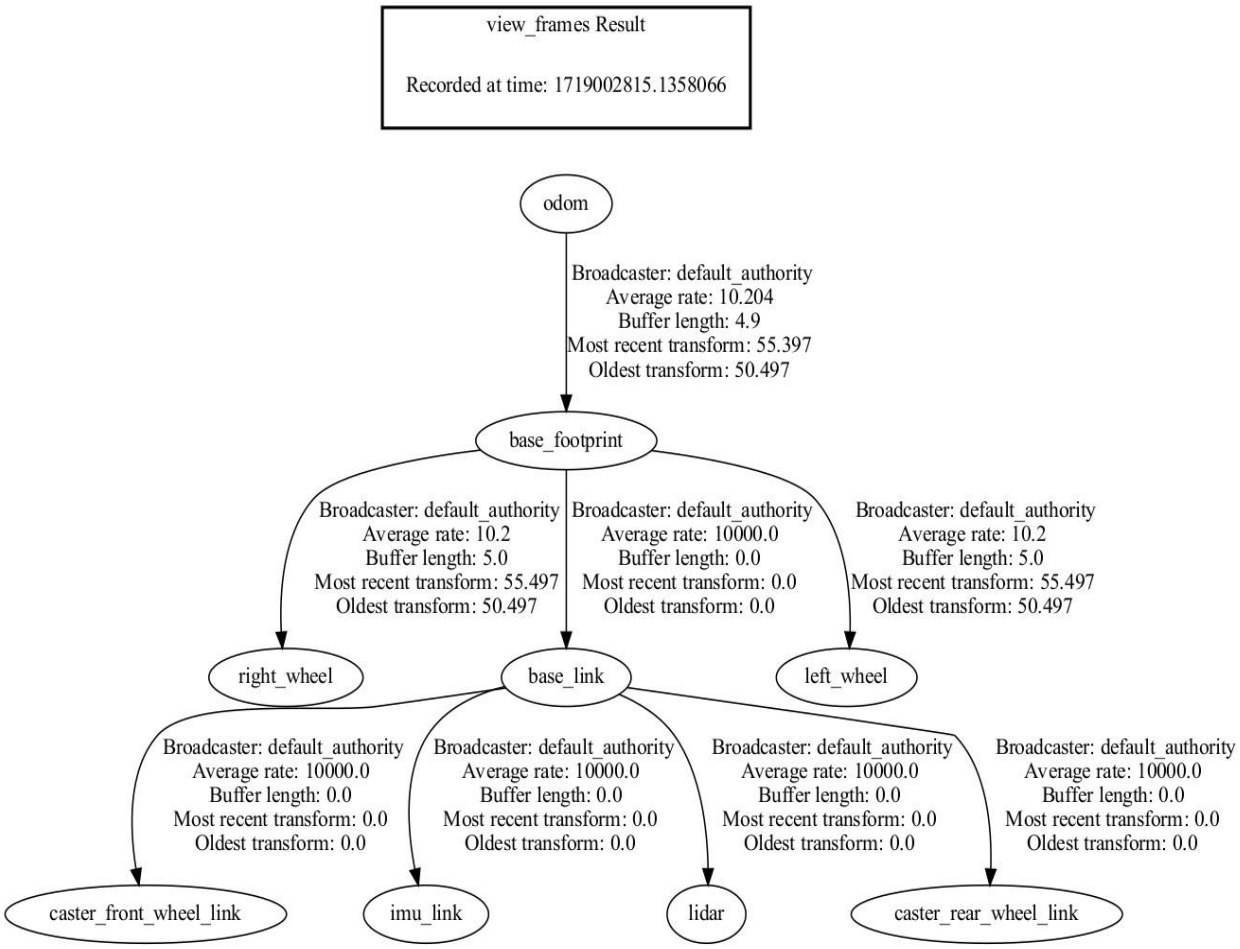


Figure 59 TF Tree

The output after launching the Multi-robot :

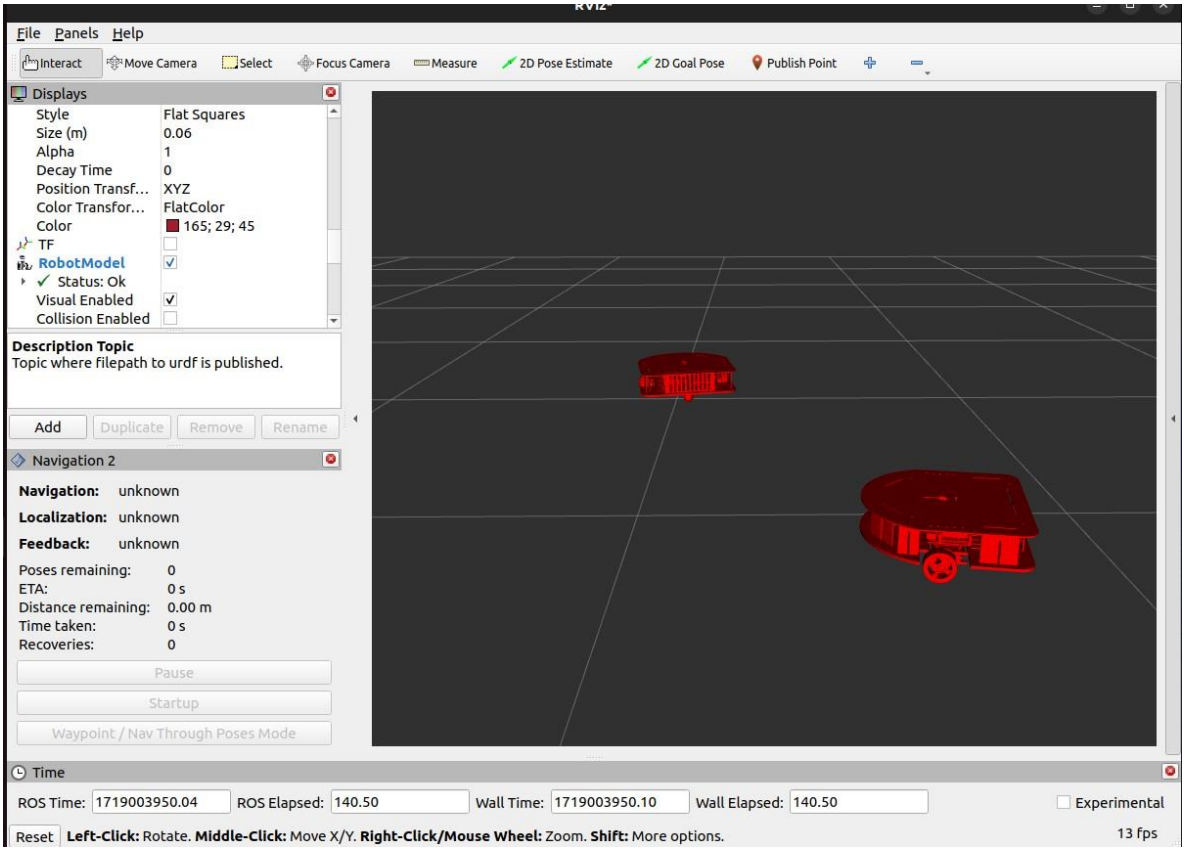


Figure 60 Multi-robot Launching on RVIZ

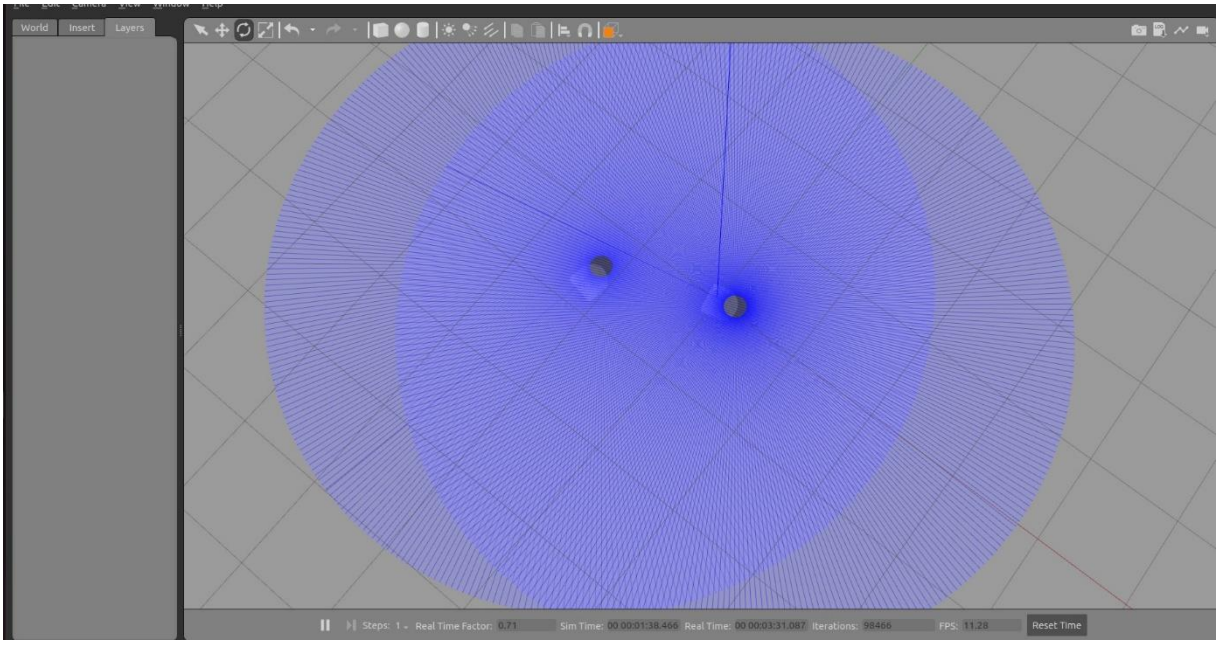


Figure 61 Multi-robot Launching on Gazebo

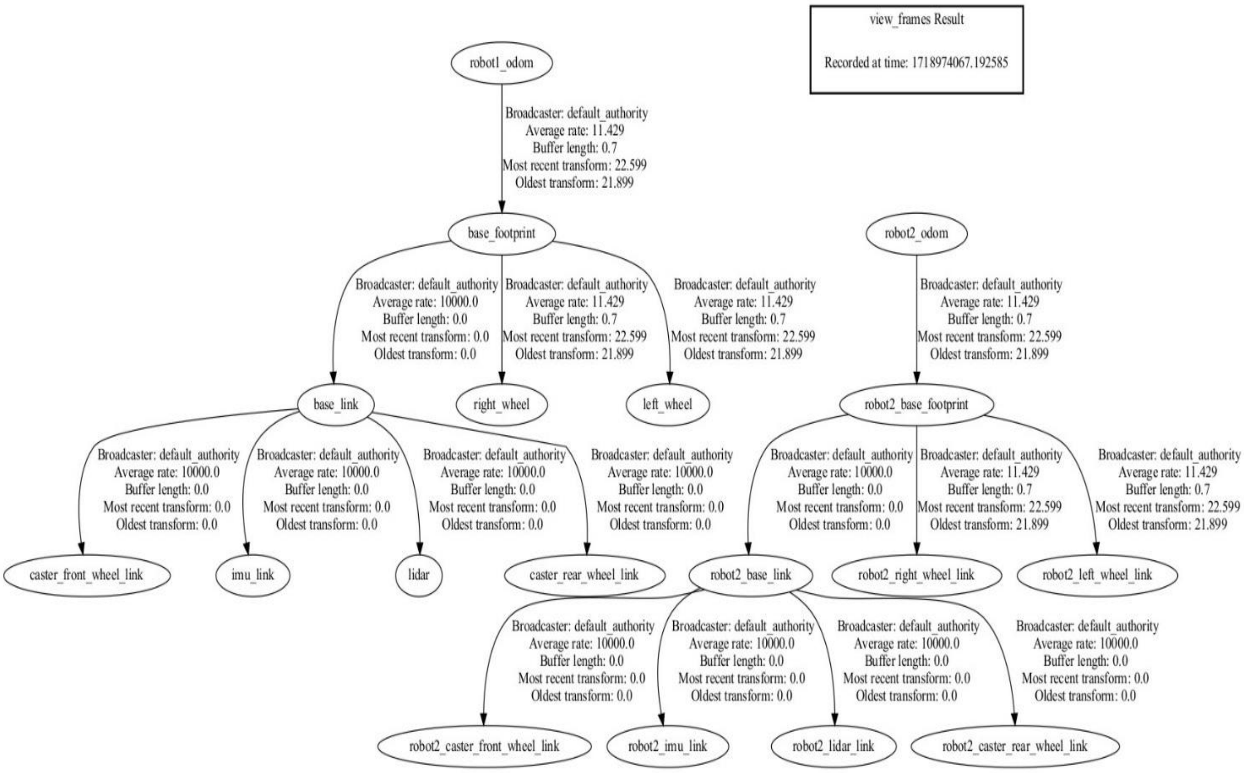
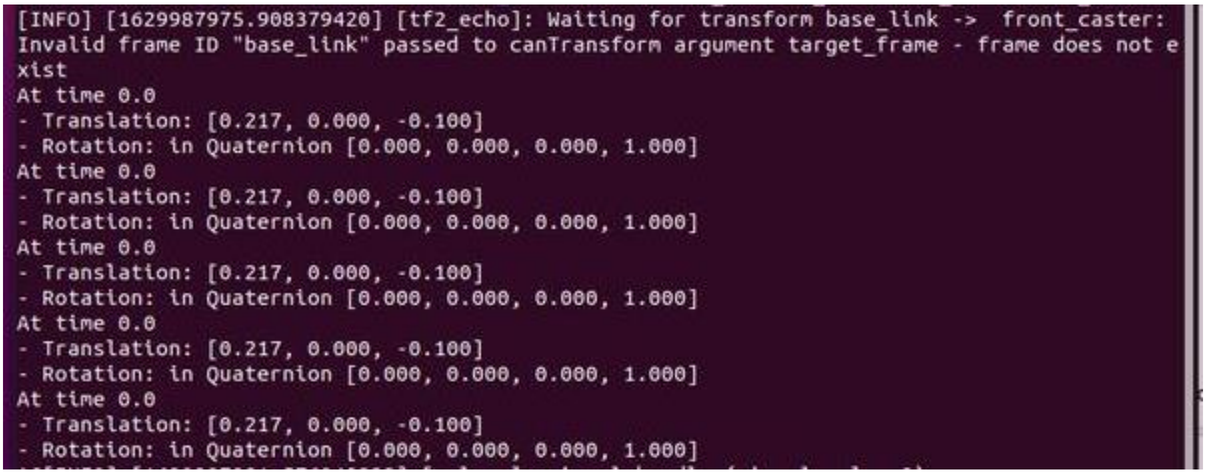


Figure 62 Viewing Robots Frames

If the coordinate transformation from one link to another need to be observed:

```
ros2 run tf2_ros tf2_echo base_link caster _front_wheel_link
```



```
[INFO] [1629987975.908379420] [tf2_echo]: Waiting for transform base_link ->  front_caster: Invalid frame ID "base_link" passed to canTransform argument target_frame - frame does not e
xist
At time 0.0
- Translation: [0.217, 0.000, -0.100]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
At time 0.0
- Translation: [0.217, 0.000, -0.100]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
At time 0.0
- Translation: [0.217, 0.000, -0.100]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
At time 0.0
- Translation: [0.217, 0.000, -0.100]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
At time 0.0
- Translation: [0.217, 0.000, -0.100]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
```

To see an image of the architecture of our ROS system:

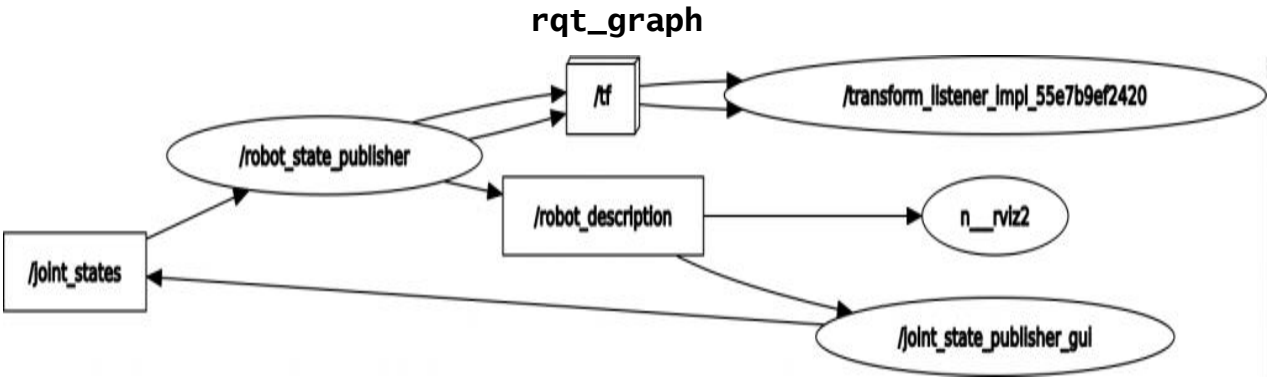


Figure 63 rqt_graph

6.5 ROS 2 Navigation Stack

The information obtained from LIDAR scans are used to build a map of the environment and to localize on the map. The purpose of doing this is to enable our robot to navigate autonomously through both known and unknown environments.

The two most used packages for localization are the **nav2_amcl** package and the **slam toolbox**. Both packages publish the **map -> odom** coordinate transformation which is necessary for a robot to localize on a map.

```
sudo apt install ros-humble-navigation2
sudo apt install ros-humble-nav2-bringup
```

For the ROS 2 Navigation Stack., The parameters enable you to do all sorts of things with the ROS 2 Navigation Stack.

The most important parameters are for the **Costmap 2D package,** a costmap is a map made up of numerous grid cells. Each grid cell has a "cost". The cost represents the difficulty a robot would have tried to move through that cell.

For example, a cell containing an obstacle would have a high cost. A cell that has no obstacle in it would have a low cost.

The ROS Navigation Stack uses two costmaps to store information about obstacles in the world.

- **Global costmap:** This costmap is used to generate long term plans over the entire environment….for example, to calculate the shortest path from point A to point B on a map.
- **Local costmap:** This costmap is used to generate short term plans over the environment…. for example, to avoid obstacles.

The AMCL (Adaptive Monte Carlo Localization) algorithm is used for localizing the robot in the world and for publishing the coordinate transform from the **map to odom** frame. AMCL localizes the robot in the world using LIDAR scans. It does this by matching real-time scan information to a known map

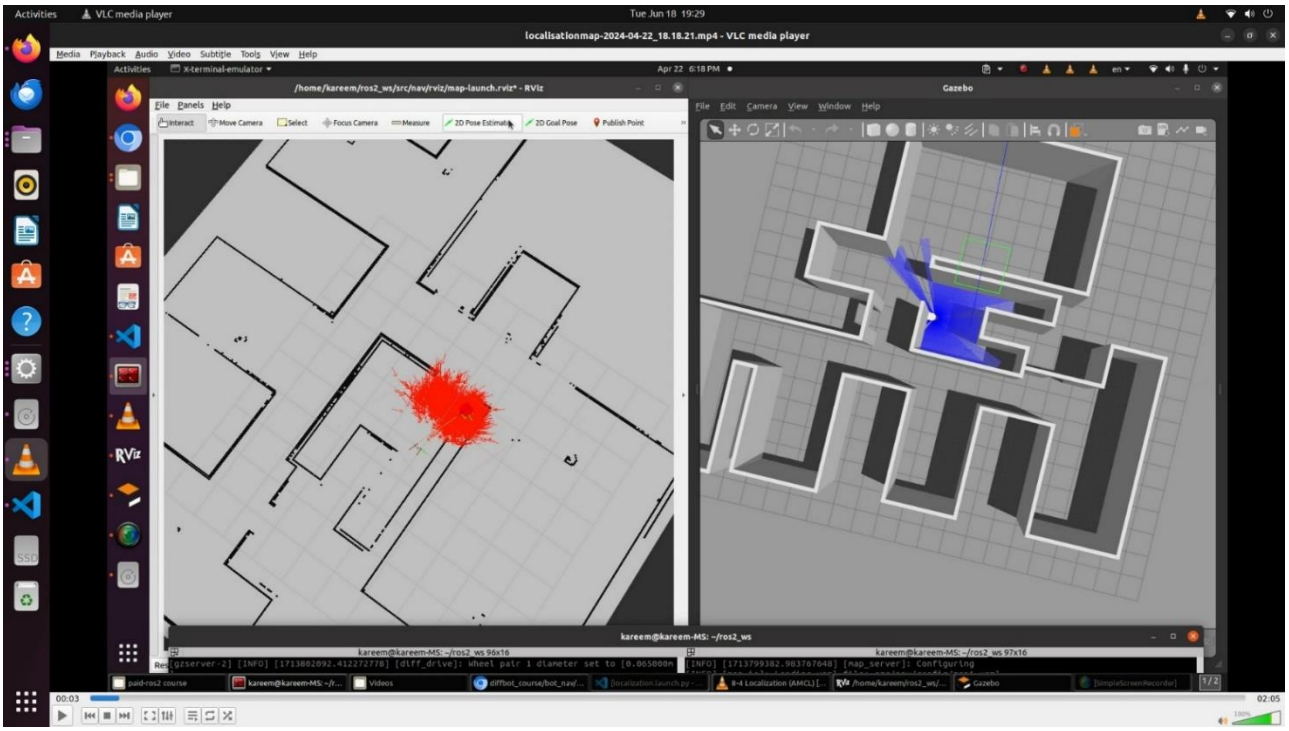


Figure 64 AMCL Localization

6.5.1 Navigation Concepts:

6.5.1.1 Action Server

Action servers in ROS 2 are used to control long-running tasks like navigation,

Action servers are like a canonical service server. A client will request some tasks to be completed, except, this task may take a long time. An example would be moving the shovel up from a bulldozer or asking a robot to travel 10 meters to the right.

In practice, action servers and clients allow these long-running tasks to be executed in separate processes or threads, returning a future result. This approach permits the client to either block until the task is complete or to periodically check on the task's status while continuing other work in the client thread. Since these tasks are long-running, action servers also provide feedback to their clients throughout the process. This feedback can be anything defined in the. action file along with the request and result types.

Feedback and results from action servers can be gathered synchronously by registering callbacks with the action client, or asynchronously by requesting information from shared future objects. Both methods require the client node to spin in order to process callback groups.

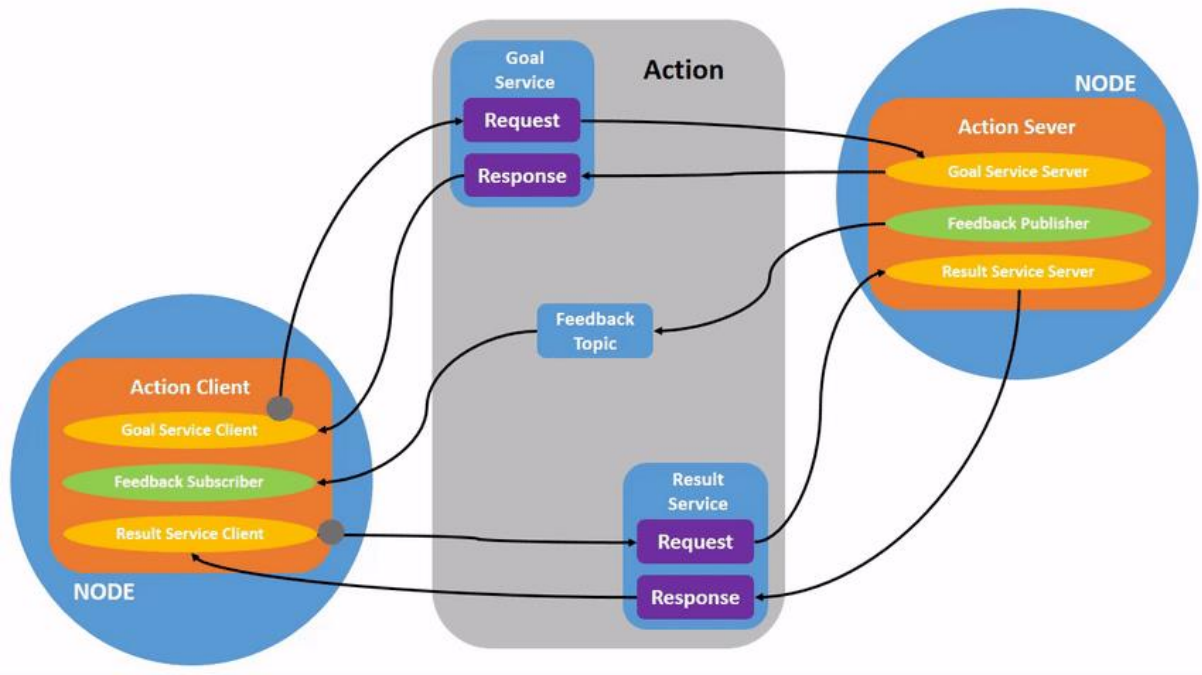


Figure 65 Navigation concepts

6.5.1.2 Behavior Trees:

Behavior Trees (BT) are increasingly used in complex robotics tasks due to their scalable and human-understandable structure. Unlike finite state machines (FSM), which can become unwieldy with many states and transitions, BTs provide a clear framework for defining multi-step applications. For example, embedding soccer game logic into an FSM would be challenging, while a BT can easily reuse basic primitives like “kick,” “walk,” and “go to ball” for various behaviors.

BTs offer a formal structure for navigation logic, enabling the creation of complex systems that can be verified and validated for correctness. Centralizing application logic in the behavior tree, with independent task servers communicating data over the tree, allows for formal analysis.

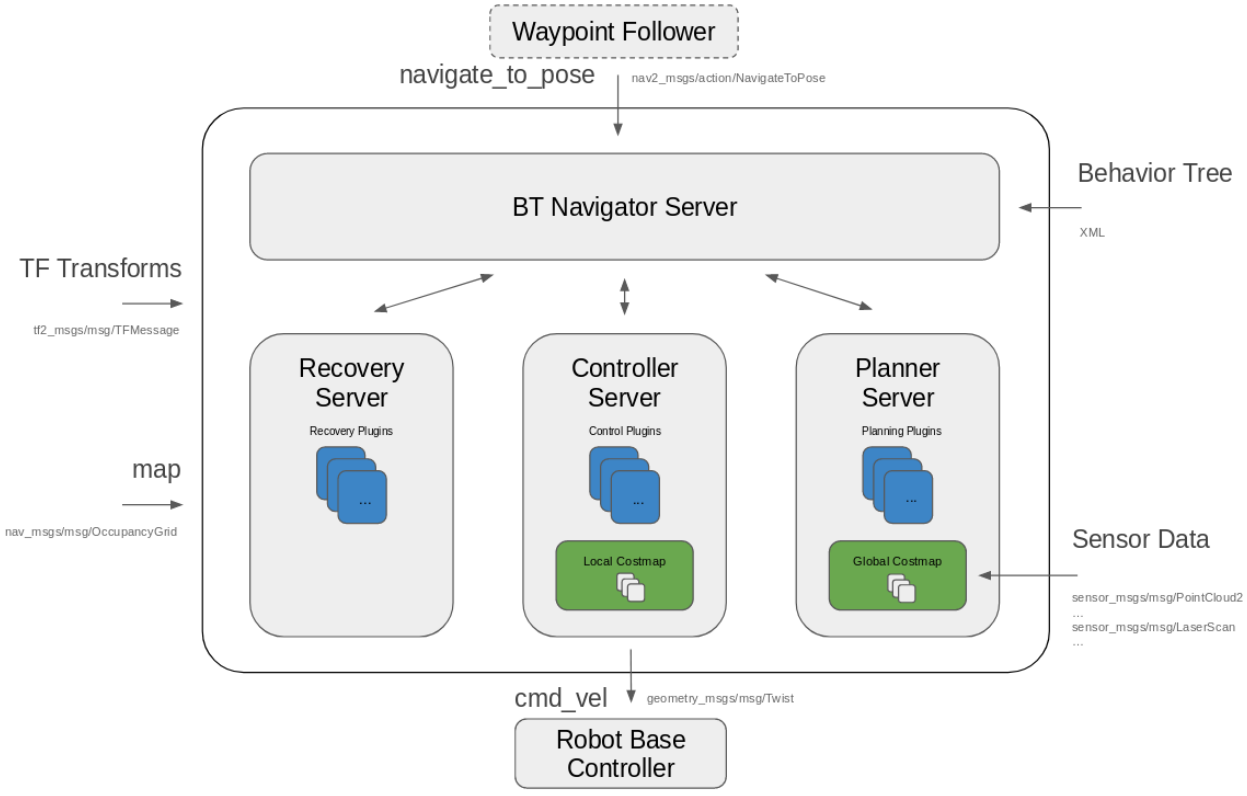


Figure 66 Behavior Trees

6.5.1.3 Planners

The task of a planner is to compute a path to complete some objective function. The path can also be known as a route, depending on the nomenclature and algorithm selected. Two canonical examples are computing a plan to a goal (from current position to a goal) or complete coverage (plan to cover all free space). The planner will have access to a global environmental representation and sensor data buffered into it. Planners can be written to:

- Compute shortest path
- Compute complete coverage path
- Compute paths along sparse or predefined routes

The general task in Nav2 for the planner is to compute a valid, and potentially optimal, path from the current pose to a goal pose. However, many classes of plans and routes exist which are supported.

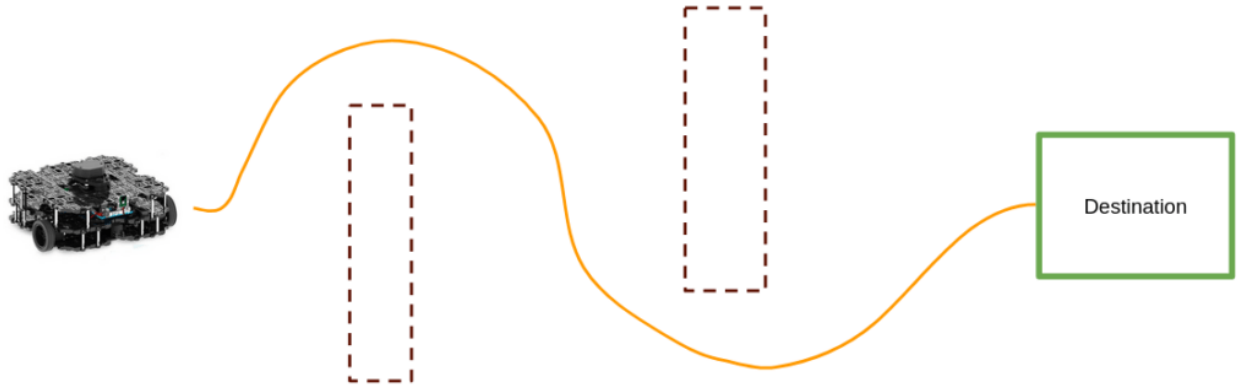


Figure 67 Nav2 Task

6.5.1.4 Controllers

Controllers are the way we follow the globally computed path or complete a local task. The controller will have access to a local environment representation to attempt to compute feasible control efforts for the base to follow. Many controllers will project the robot forward in space and compute a locally feasible path at each update iteration. Controllers can be written to:

- Follow a path
- Dock with a charging station using detectors in the odometrical frame
- Board an elevator
- Interface with a tool

The general task in Nav2 for a controller is to compute a valid control effort to follow the global plan. However, many classes of controllers and local planners exist. It is the goal of this project that all controller algorithms can be plugins in this server for common research and industrial tasks.

6.5.1.5 Costmaps and Layers

The current environmental representation is a costmap. A costmap is a regular 2D grid of cells containing a cost from unknown, free, occupied, or inflated cost. This costmap is then searched to compute a global plan or sampled to compute local control efforts.

Various costmap layers are implemented as pluginlib plugins to buffer information into the costmap. This includes information from LIDAR, RADAR, sonar, depth images, etc. It may be wise to process sensor data before inputting it into the costmap layer, but that is up to the developer.

Costmap layers can be created to detect and track obstacles in the scene for collision avoidance using camera or depth sensors. Additionally, layers can be created to algorithmically change the underlying costmap based on some rule or heuristic. Finally, they may be used to buffer live data into the 2D or 3D world for binary obstacle marking.

6.5.2 Robot configuration:

6.5.2.1 ROS 2 Cartographer:

The main problem in mobile robotics is localization and mapping. To estimate the position of the robot in an environment, you need some kind of map from this environment to determine the actual position in this environment. On the other hand, you need the actual position of robots to create a map related to its position. Therefore, you can use SLAM – Simultaneous Localization and Mapping.

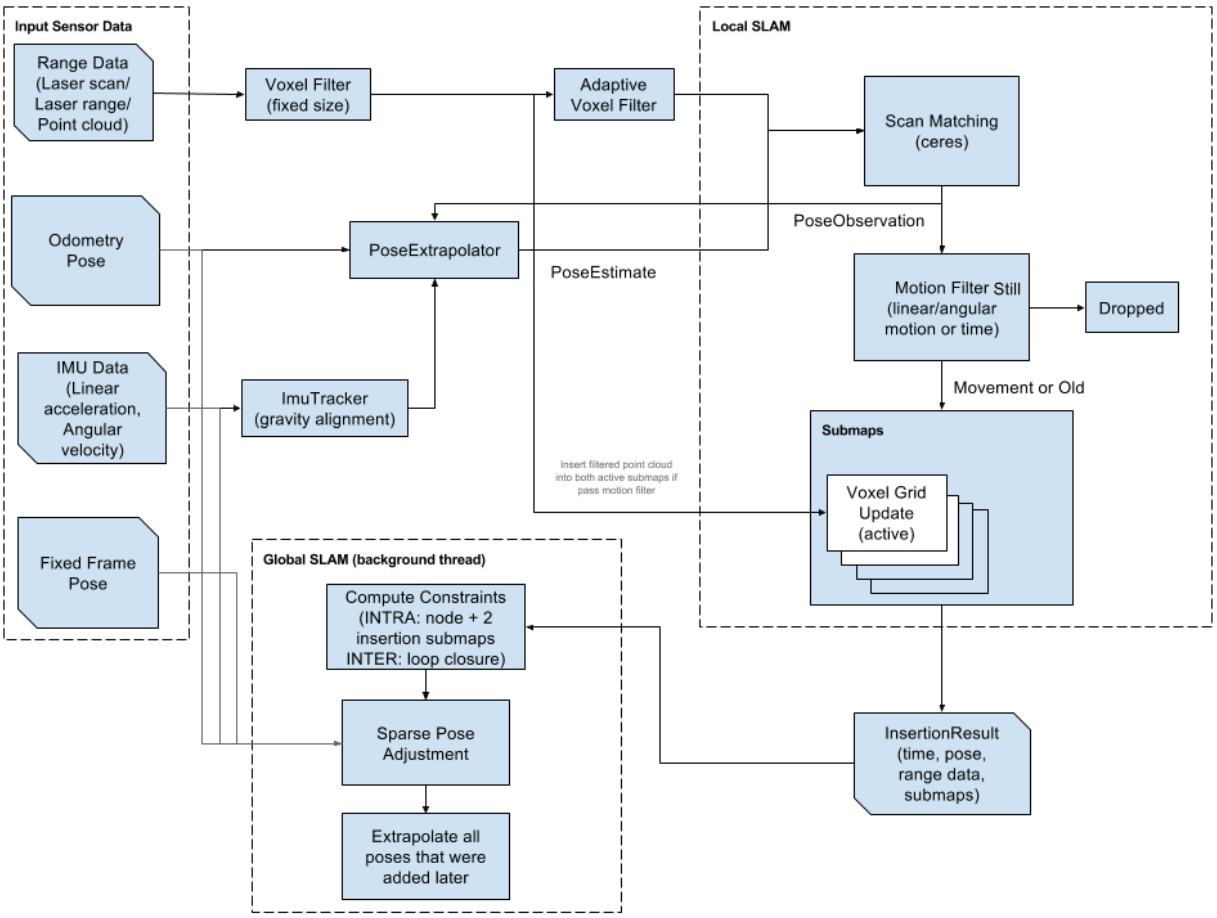


Figure 68 Slam

6.5.2.2 Installation

```
$ sudo apt install ros-humble-cartographer
```

6.5.2.3 Parameters explained:

Cartographer parameters:

- **Frames:**

- map_frame: The frame ID of the map.

- tracking_frame: The frame ID of the robot's base.

- published_frame and odom_frame: The frame ID for odometry data.

- **Sensor Usage:**

- use_odometry, use_nav_sat, use_landmarks: Specify whether to use odometry, GPS, and landmarks.

- **Sensor Data:**

- num_laser_scans, num_multi_echo_laser_scans, num_subdivisions_per_laser_scan, num_point_clouds: Number and types of sensors used.

- **Timing Parameters:**

- lookup_transform_timeout_sec, submap_publish_period_sec, pose_publish_period_sec, trajectory_publish_period_sec: Define various timeouts and publishing periods.

- **Sampling Ratios:**

- rangefinder_sampling_ratio, odometry_sampling_ratio, fixed_frame_pose_sampling_ratio, imu_sampling_ratio, landmarks_sampling_ratio: Define sampling ratios for different data sources.

- **Map Builder Configuration:**

- Specifies that a 2D trajectory builder will be used.

- **Trajectory Builder Configuration:**

- Defines the range for the laser scanner.
- Specifies whether to use IMU data.
- Enables online correlative scan matching for improved mapping.
- Configures motion filter to discard small movements.

- **Pose Graph Configuration:**

- Sets the minimum score for adding constraints to the pose graph.
- Configures the minimum score for global localization.
- Optionally disables optimization after a certain number of nodes.

AMCL Parameters:

- **alpha1 to alpha5:**

- Noise parameters for the motion model, these parameters represent different types of noise (e.g., rotational, translational) and help AMCL to model the uncertainty in the robot's movements.

- **Frames**:

- base_frame_id: The robot's base frame.
- global_frame_id: The map frame.
- odom_frame_id: The odometry frame.

- **Beam Skipping Parameters:**

- Parameters related to the beam skipping technique used to speed up the processing of laser scans by ignoring some beams.

- **Laser Parameters:**

- **laser_likelihood_max_dist:** Maximum distance for considering the likelihood field model.
- **laser_max_range:** Maximum range of the laser scanner.
- **laser_min_range:** Minimum range of the laser scanner.
- **laser_model_type:** Type of model used for the laser scan.
- **max_beams:** Maximum number of beams to consider.

- **Particle Filter Parameters:**

- **max_particles:** Maximum number of particles used in the filter.
- **min_particles:** Minimum number of particles.
- **pf_err:** Threshold for the particle filter error.
- **pf_z:** Threshold for the particle filter's effective sample size.

- **Recovery Parameters:**

- Parameters for controlling the recovery behavior of the particle filter.

- **Resresample_interval: Frequency of resampling the particle filter.**

- robot_model_type: Type of motion model used

- **save_pose_rate: Rate at which to save the pose.**
- **tf_broadcast: Whether to broadcast the transform.**
- **transform_tolerance: Tolerance for the transform.**
- **update_min_a and update_min_d: Minimum update thresholds for angle and distance.**
- **z_hit, z_max, z_rand, z_short: Parameters for the likelihood field model.**

bt_navigator parameters:

- **Behavior Tree Loop Duration: The duration for each loop of the behavior tree, in milliseconds.**
- **Plugin Libraries: These plugins correspond to specific actions or conditions used in behavior trees. Examples:**

- nav2_compute_path_to_pose_action_bt_node: Computes a path to a given pose.
- nav2_follow_path_action_bt_node: Follows a given path.
- nav2_clear_costmap_service_bt_node: Clears the costmap.
- nav2_is_stuck_condition_bt_node: Checks if the robot is stuck.
- nav2_goal_reached_condition_bt_node: Checks if the goal has been reached.
- nav2_recovery_node_bt_node: Handles recovery actions.
- nav2_navigate_to_pose_action_bt_node: Navigates to a specific pose.

controller_server parameters:

- **controller_frequency: The frequency at which the controller runs, in Hz.**
- **Velocity Thresholds: Minimum velocity thresholds for the x, y, and theta directions to avoid negligible movements.**
- **Failure Tolerance: The tolerance level for considering a movement failure.**
- **Progress Checker: Specifies the plugin to use for checking the robot's progress.**

- **required_movement_radius:** The radius within which the robot must move to be considered making progress.
- **movement_time_allowance:** The time allowed for the robot to make the required movement.

- **Goal Checker: Specifies the goal checker plugins to use.**

- **stateful:** If true, the goal checker maintains state between checks.
- **xy_goal_tolerance:** The tolerance for reaching the goal in the x and y directions.
- **yaw_goal_tolerance:** The tolerance for reaching the goal orientation.

- **FollowPath Controller: configure the DWB (Dynamic Window Approach) local planner, including:**

- **debug_trajectory_details:** Whether to output detailed trajectory debugging information.
- **min_vel_x, min_vel_y, max_vel_x, max_vel_y:** Minimum and maximum velocities in the x

and y directions.

- **max_vel_theta:** Maximum rotational velocity.
- **acc_lim_x, acc_lim_y, acc_lim_theta:** Acceleration limits.
- **decel_lim_x, decel_lim_y, decel_lim_theta:** Deceleration limits.
- **vx_samples, vy_samples, vtheta_samples:** Number of samples for velocity in x, y, and theta.
- **sim_time:** Simulation time for trajectory evaluation.
- **linear_granularity, angular_granularity:** Granularity of linear and angular sampling.
- **transform_tolerance:** Tolerance for transforms.
- **xy_goal_tolerance:** Tolerance for reaching the goal in x and y directions.
- **trans_stopped_velocity:** Threshold for considering the robot stopped.
- **short_circuit_trajectory_evaluation:** Whether to short-circuit trajectory evaluation if an invalid trajectory is found.
- **critics:** List of trajectory critics to use for scoring trajectories.
- **BaseObstacle.scale, PathAlign.scale, etc.:** Scaling factors for the critics.

6.5.2.4 Local Costmap

The local_costmap is responsible for creating a dynamic, short-range map around the robot to aid in local navigation and obstacle avoidance.

- *General Parameters:*

- **update_frequency:** The rate at which the costmap updates, set to 5 Hz.
- **publish_frequency:** The rate at which the costmap is published, set to 4 Hz.
- **global_frame:** The frame in which the costmap operates, set to "map".
- **robot_base_frame:** The robot's base frame, set to "base_footprint".
- **use_sim_time:** Whether to use simulated time or real-time, set to False.
- **rolling_window:** Whether the costmap moves with the robot, set to True.
- **width & height:** The dimensions of the costmap, set to 2 meters.
- **resolution:** The resolution of the costmap, set to 0.02 meters per cell.

- *Robot Footprint:*

- footprint: Defines the shape of the robot as a polygon for obstacle avoidance.

- *Plugins:*

- **inflation_layer: Expands obstacles to create a buffer around them.**
  - **cost_scaling_factor:** Determines how fast the cost increases with distance from an obstacle.
  - **inflation_radius:** The radius within which obstacles are inflated.
- **voxel2d_layer: Handles 3D obstacles and marks/clears obstacles based on sensor data.**
  - **enabled:** Whether this layer is active.
  - **z_resolution:** The resolution in the z-axis for voxel grid.
  - **z_voxels:** Number of voxels in the z-axis.
  - **max_obstacle_height:** Maximum height of obstacles to consider.
  - **observation_sources:** Sources of obstacle data, in this case, laser scans from /scan.
- **static_layer: Uses a static map layer.**
  - **map_subscribe_transient_local:** If the map data should be transient local.
- **always_send_full_costmap: Always send the entire costmap.**

6.5.2.5 Global Costmap

The global_costmap is used for long-term path planning over the entire map.

- *General Parameters:*

- **update_frequency:** The rate at which the costmap updates, set to 5 Hz.
- **publish_frequency**: The rate at which the costmap is published, set to 2 Hz.
- **global_frame:** The frame in which the costmap operates, set to "map".
- **robot_base_frame:** The robot's base frame, set to "base_footprint".
- **use_sim_time:** Whether to use simulated time or real-time, set to False.
- **resolution:** The resolution of the costmap, set to 0.05 meters per cell.
- **track_unknown_space:** Whether to consider unknown space in the map.

- *Robot Footprint:*
  - **footprint:** Defines the shape of the robot as a polygon for obstacle avoidance**.**

- *Plugins:*

- **static_layer: Uses a static map layer.**
  - **plugin:** The specific plugin used for the static layer.
  - **map_subscribe_transient_local:** If the map data should be transient local.
- **voxel2d_layer: Handles 3D obstacles based on sensor data**.
  - **enabled:** Whether this layer is active.
  - **observation_sources:** Sources of obstacle data, in this case, laser scans from /scan.
- **inflation_layer: Expands obstacles to create a buffer around them.**
  - **cost_scaling_factor:** Determines how fast the cost increases with distance from an obstacle.
  - **inflation_radius:** The radius within which obstacles are inflated.

- **always_send_full_costmap: Always send the entire costmap.**

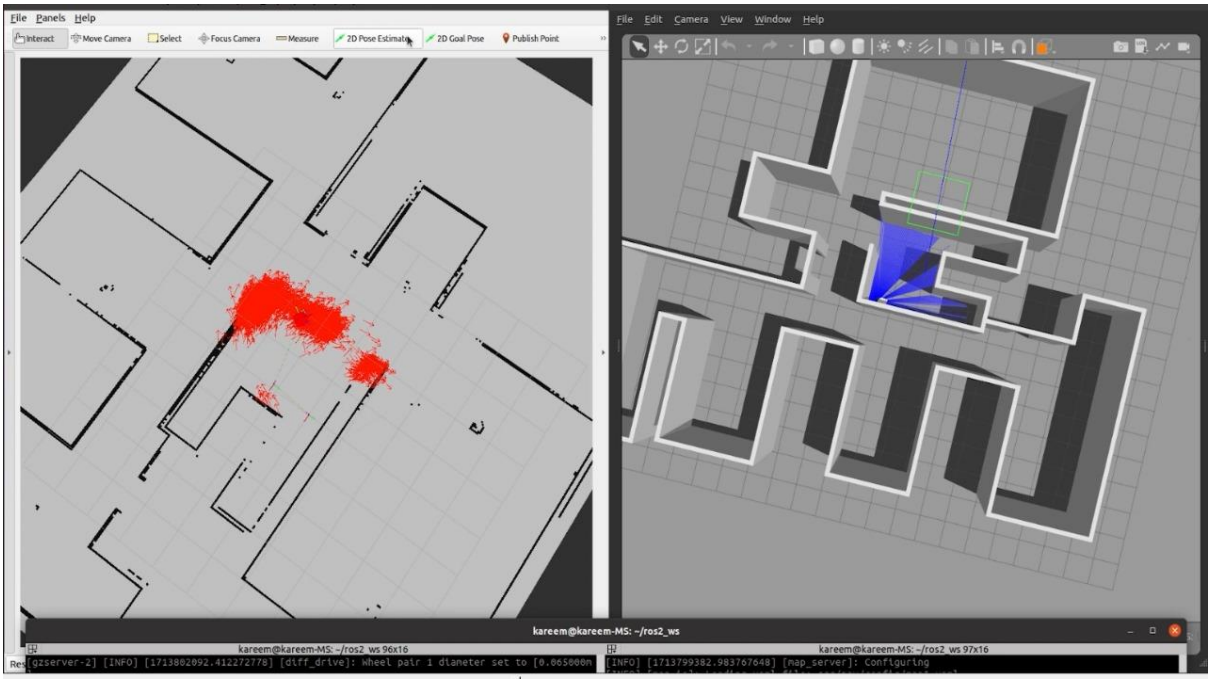


Figure 69 Results_1

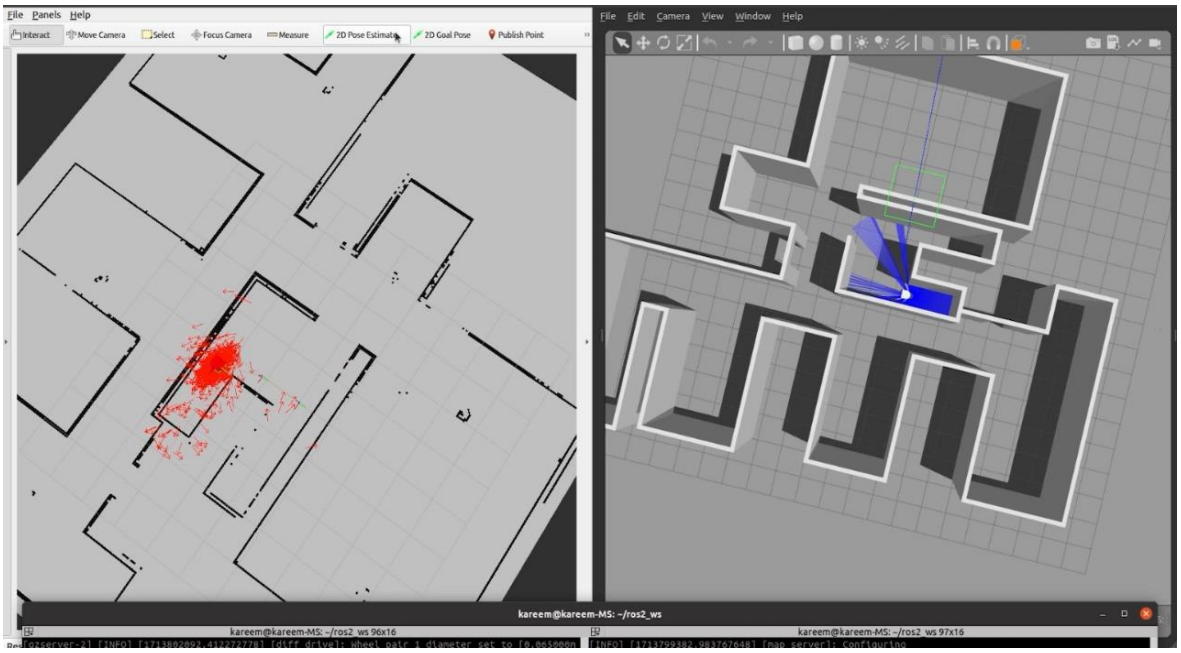


Figure 70 Results_2

7. Hardware Implementation

7.1 Hardware Interface Diagram

The ROS2 `ros2_control` package provides a powerful framework for implementing control in differential drive robots. By abstracting hardware interfaces and utilizing a well-defined kinematic model, this framework enables precise control and accurate odometry estimation. The integration of these components in ROS2 facilitates the development of advanced autonomous robotic systems, supporting tasks such as navigation and localization with high reliability and accuracy.

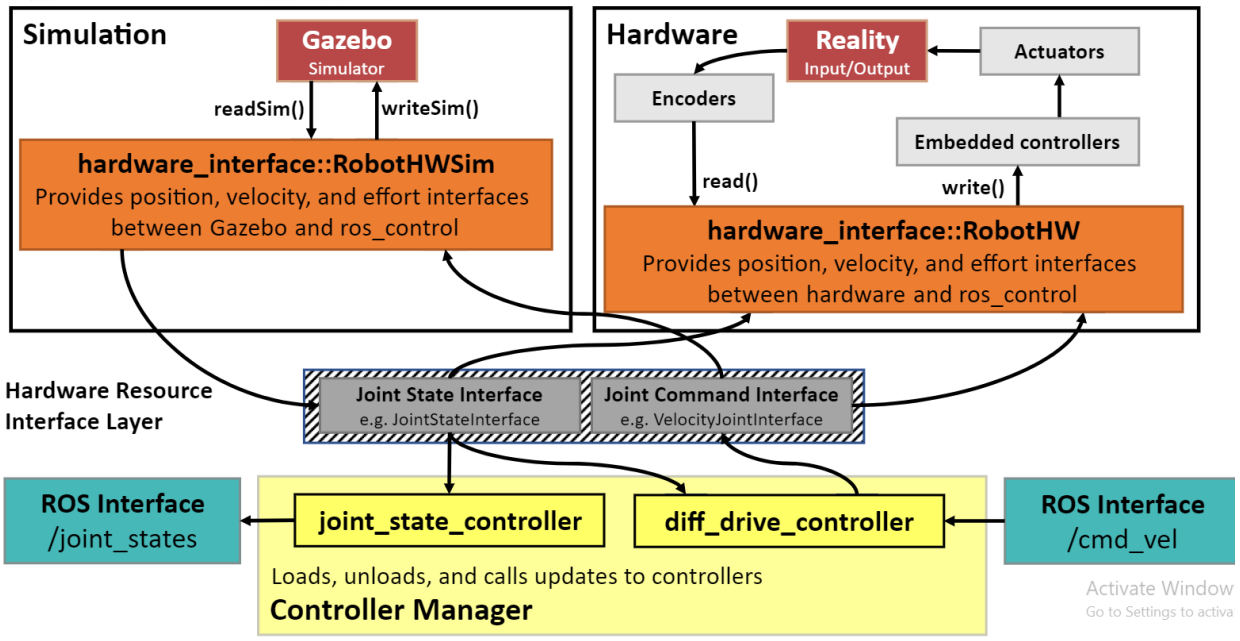


Figure 71 Hardware, Simulation and Controller Interface

7.2 Implementing ROS2 Control in a Differential Drive Robot

The implementation of `ros2_control` for a differential drive robot in this project involves several key components: the hardware interface, controller configuration, and the conversion of the kinematic model to odometry data. These components work together to ensure accurate and reliable control of the robot.

7.2.1 Hardware Interface

The hardware interface acts as the intermediary between ROS2 and the robot's physical actuators. This interface manages the state and command data for the robot's wheels. It subscribes to motor feedback and publishes velocity commands, enabling real-time control of

the robot's movements. The interface handles initialization, activation, deactivation, and real-time reading and writing of data to and from the robot's actuators.

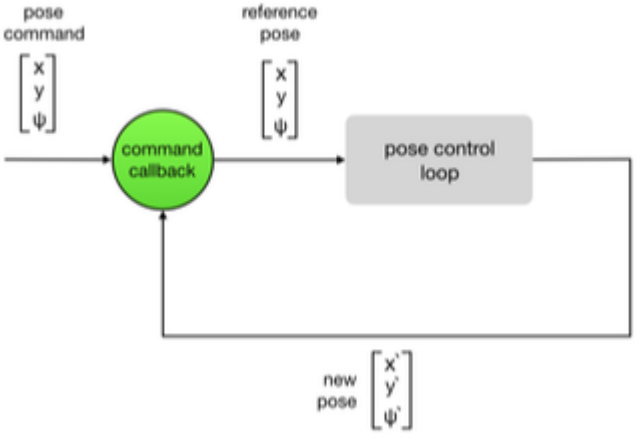


Figure 72 Command Control Loop

7.2.2 Controller Configuration

Configuring the controllers in ROS2 involves defining parameters that dictate the robot's movement management. For the differential drive robot, the `DiffDriveController` is used to handle velocity command computation and odometry. Key configuration parameters include:

- **Wheel Separation and Radius**: These define the physical dimensions of the robot, crucial for accurate kinematic calculations.
- **Velocity and Acceleration Limits**: These ensure the robot operates within safe ranges, preventing hardware damage and maintaining control stability.
- **Feedback Mechanisms**: Position and velocity feedback from the wheels are essential for closed-loop control, allowing the controller to adjust commands based on the robot's actual state
- **Command Timeout**: This specifies the maximum duration for which velocity commands are valid, ensuring the robot stops if no new commands are received within the specified time.

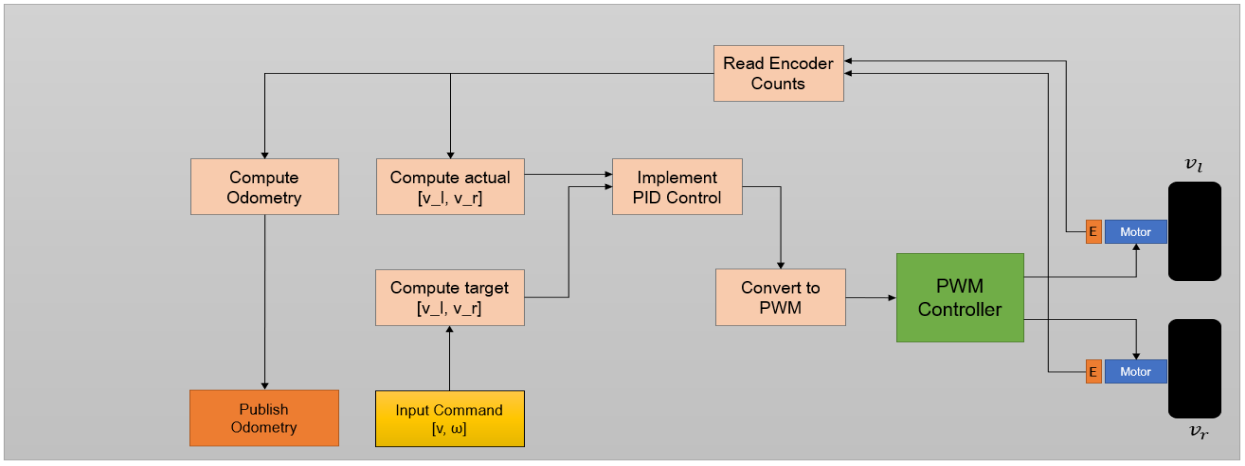


Figure 73 Controller Configuration

7.2.3 Implementation

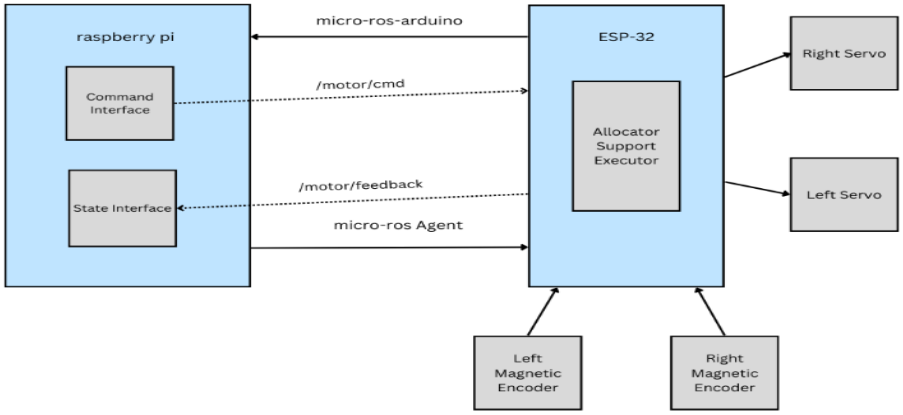


Figure 74 Controllers Implementation

7.2.4 ros2 control terminal launch

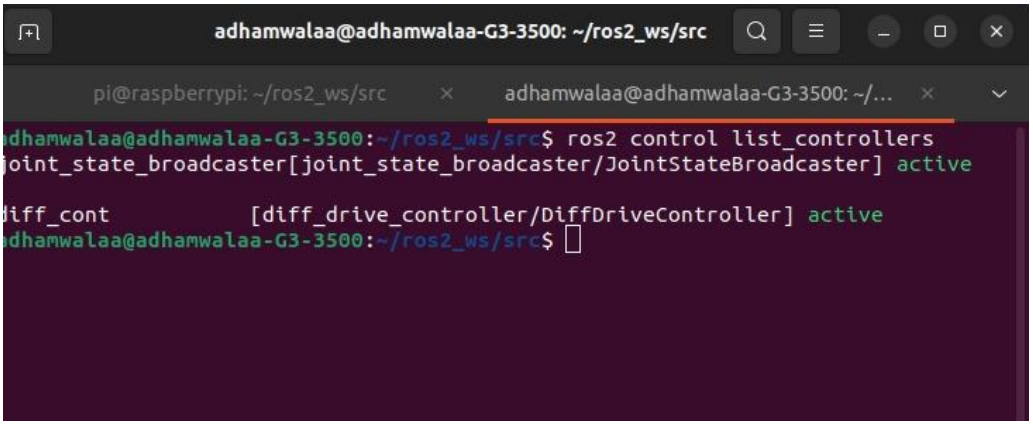


Figure 75 ros2 control terminal launch_1

Figure 76 ros2 control terminal launch_2

7.3 Motors Implementation code on ESP32

```
#include <micro_ros_arduino.h>  // Library for micro-ROS on Arduino

#include <stdio.h>
#include <rcl/rcl.h>            // Core ROS 2 client library
#include <rcl/error_handling.h>  // Error handling in ROS 2 client library
#include <rclc/rclc.h>          // Convenience layer for ROS 2 client
#include <rclc/executor.h>      // Executor for managing callbacks
#include <esp_system.h>         // ESP32 specific system functions

#include <std_msgs/msg/int64_multi_array.h>  // Message type for motor feedback
#include <std_msgs/msg/int32.h>              // Message type for motor command

// Declare ROS 2 entities
rcl_subscription_t rightmotor_sub;
rcl_subscription_t leftmotor_sub;

rcl_publisher_t motorfeedback_pub;
std_msgs__msg__Int64MultiArray encoderMsg;
std_msgs__msg__Int32 leftMsg;
std_msgs__msg__Int32 rightMsg;

rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;


//----------------------
```

```
//servo
// Control pin (white), signal pin (yellow)

#include <ESP32Servo.h>
Servo servo_R;
Servo servo_L;



// right motor
#define SRVPIN_R 12  // Control pin (white)
#define ENCPIN_R 14  // Signal pin (orange)

// left motor
#define SRVPIN_L 22  // Control pin (white)
#define ENCPIN_L 4  // Signal pin (orange)



#define PULSES_PER_REV 4095
int current = 0;
int zero = 0;



long position_r = 0;
long position_l = 0;



//**********************

// Macros for error checking
#define RCCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) { error_loop(); } \
  }
```

```
#define RCSOFTCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) {} \
  }

int pwm = 0;  // PWM value for motor control


// Function to handle errors by blinking the LED and restarting the ESP32
void error_loop() {
  int loop_counter = 0;
  while (1) {
    loop_counter += 1;
    delay(100);
    if (loop_counter >= 50) {
      esp_restart();
    }
  }
}



// Callback for right motor subscription
void rightsub_callback(const void *msgin) {
  const std_msgs__msg__Int32 *msg = (const std_msgs__msg__Int32 *)msgin;
  pwm = msg->data;
  if (pwm > 0) {
    int vel = map(pwm, 0, 255, 90, 180);  //ccw
    servo_R.write(vel);
```

```
  } else {
    int vel = map(pwm, 0, -255, 90, 0);  //cw
    servo_R.write(vel);
  }
}

// Callback for left motor subscription
void leftsub_callback(const void *msgin) {
  const std_msgs__msg__Int32 *msg = (const std_msgs__msg__Int32 *)msgin;
  pwm = msg->data;
  if (pwm > 0) {
    int vel = map(pwm, 0, 255, 90, 0);  //ccw
    servo_L.write(vel);

  } else {
    int vel = map(pwm, 0, -255, 90, 180);  //cw
    servo_L.write(vel);
  }
}



class MagneticEncoder {
public:
  MagneticEncoder(int maxValue) {
    previous_value = -1;  // Use -1 to indicate uninitialized state
    continuous_position = 0;
    max_value = maxValue;
  }
```

```
  long update(int current_value) {
    if (previous_value == -1) {
      // First update, just initialize the previous value
      previous_value = current_value;
      return continuous_position;
    }

    // Calculate the difference (delta)
    int delta = current_value - previous_value;

    // Handle wrap-around
    if (delta > max_value / 2) {
      // Wrapped around in the negative direction
      delta -= (max_value + 1);
    } else if (delta < -max_value / 2) {
      // Wrapped around in the positive direction
      delta += (max_value + 1);
    }

    // Update the continuous position
    continuous_position += delta;

    // Update the previous value
    previous_value = current_value;

    return continuous_position;
  }

private:
  int previous_value;
  long continuous_position;
  int max_value;
};
```

```
// Initialize the encoder with the maximum value of 4095
MagneticEncoder encoder_r (PULSES_PER_REV);
MagneticEncoder encoder_l (PULSES_PER_REV);



void setup() {
  pinMode(ENCPIN_R, INPUT);
  servo_R.attach(SRVPIN_R);

  pinMode(ENCPIN_L, INPUT);
  servo_L.attach(SRVPIN_L);

  //stop servo motors
  servo_R.write(90);
  servo_L.write(90);

  setzero();
  set_microros_transports();  // Initialize micro-ROS transport
  delay(2000);                // Wait for initialization

  allocator = rcl_get_default_allocator();  // Get default allocator

  // Initialize ROS 2 support
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // Create ROS 2 node
  RCCHECK(rclc_node_init_default(&node, "esp32_node", "", &support));

  // Initialize publisher for motor feedback
  RCCHECK(rclc_publisher_init_default(
    &motorfeedback_pub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int64MultiArray),
    "motor/feedback"));
```

```
  RCCHECK(rclc_subscription_init_default(
    &leftmotor_sub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "motor/left_cmd"));


  RCCHECK(rclc_subscription_init_default(
    &rightmotor_sub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "motor/right_cmd"));

  const unsigned int timer_timeout = 100;

  // Initialize timer for periodic callback
  RCCHECK(rclc_timer_init_default(
    &timer,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback));

  // Initialize executor to handle callbacks
  RCCHECK(rclc_executor_init(&executor, &support.context, 3, &allocator));
  RCCHECK(rclc_executor_add_timer(&executor, &timer));
  RCCHECK(rclc_executor_add_subscription(&executor, &rightmotor_sub, &rightMsg,
&rightsub_callback, ON_NEW_DATA));
  RCCHECK(rclc_executor_add_subscription(&executor, &leftmotor_sub, &leftMsg,
&leftsub_callback, ON_NEW_DATA));
}

// Timer callback to publish encoder values
void timer_callback(rcl_timer_t * timer, int64_t last_call_time){
```

```
  RCLC_UNUSED(last_call_time);
  if (timer != NULL) {
    static int64_t data_array[2] = {0};
    data_array[0] = position_r;
    data_array[1] = position_l;

    encoderMsg.data.data = data_array;
    encoderMsg.data.size = 2;
    encoderMsg.data.capacity = 2;
    RCSOFTCHECK(rcl_publish(&motorfeedback_pub, &encoderMsg, NULL));
  }
}

void setzero() {
  while (abs(zero - current) != 0) {
    current = analogRead(ENCPIN_R);
    servo_R.write(180);
    current = analogRead(ENCPIN_L);
    servo_L.write(180);
  }
  zero = current;  // Set zero to the current position after reaching it
}

void loop() {
  // Execute ROS 2 callbacks
  RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(10)));
  int current_value_r = analogRead(ENCPIN_R);
  position_r = encoder_r.update(current_value_r);

  int current_value_l = analogRead(ENCPIN_L);
  position_l =  -encoder_l.update(current_value_l);

}
```

7.4 CPP ros2-control diff-drive Hardware Interface Code

```cpp
#include <std_msgs/msg/float32_multi_array.hpp>

#include <chrono>
#include <cmath>
#include <cstddef>
#include <limits>
#include <memory>
#include <vector>

#include "hardware_interface/types/hardware_interface_type_values.hpp"
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/int64_multi_array.hpp"
#include <std_msgs/msg/float32_multi_array.hpp>
namespace robot_firmware
{
DiffBotSystemHardware::DiffBotSystemHardware() : node_(std::make_shared<rclcpp::Node>("diffbot_interface_node"))

{
  feedback_subscription_ = node_->create_subscription<std_msgs::msg::Int64MultiArray>(
    "/motor/feedback", 10,
    [this](const std_msgs::msg::Int64MultiArray::SharedPtr msg) {
      this->processFeedback(msg);
    });
```

Figure 77 Hardware Interface Code_1

```cpp
  cmd_publisher_ = node_->create_publisher<std_msgs::msg::Float32MultiArray>("/motor/cmd", 10);
}

DiffBotSystemHardware::~DiffBotSystemHardware() {}

hardware_interface::CallbackReturn DiffBotSystemHardware::on_init(
  const hardware_interface::HardwareInfo & info)
{
  if (hardware_interface::SystemInterface::on_init(info) != hardware_interface::CallbackReturn::SUCCESS) {
    return hardware_interface::CallbackReturn::ERROR;
  }

  hw_positions_.resize(info_.joints.size(), std::numeric_limits<double>::quiet_NaN());
  hw_velocities_.resize(info_.joints.size(), std::numeric_limits<double>::quiet_NaN());
  hw_commands_.resize(info_.joints.size(), std::numeric_limits<double>::quiet_NaN());

  return hardware_interface::CallbackReturn::SUCCESS;
}

std::vector<hardware_interface::StateInterface> DiffBotSystemHardware::export_state_interfaces()
{
  std::vector<hardware_interface::StateInterface> state_interfaces;
  for (auto i = 0u; i < info_.joints.size(); i++) {
    state_interfaces.emplace_back(hardware_interface::StateInterface(
      info_.joints[i].name, hardware_interface::HW_IF_POSITION, &hw_positions_[i]));
    state_interfaces.emplace_back(hardware_interface::StateInterface(
      info_.joints[i].name, hardware_interface::HW_IF_VELOCITY, &hw_velocities_[i]));
  }

  return state_interfaces;
}
```

Figure 78 Hardware Interface Code_2

```cpp
std::vector<hardware_interface::CommandInterface> DiffBotSystemHardware::export_command_interfaces()
{
  std::vector<hardware_interface::CommandInterface> command_interfaces;
  for (auto i = 0u; i < info_.joints.size(); i++) {
    command_interfaces.emplace_back(hardware_interface::CommandInterface(
      info_.joints[i].name, hardware_interface::HW_IF_VELOCITY, &hw_commands_[i]));
  }

  return command_interfaces;
}

hardware_interface::CallbackReturn DiffBotSystemHardware::on_activate(
  const rclcpp_lifecycle::State & /*previous_state*/)
{
  RCLCPP_INFO(rclcpp::get_logger("DiffBotSystemHardware"), "Activating ...please wait...");

  // set some default values
  for (auto i = 0u; i < hw_positions_.size(); i++) {
    if (std::isnan(hw_positions_[i])) {
      hw_positions_[i] = 0;
      hw_velocities_[i] = 0;
      hw_commands_[i] = 0;
    }
  }

  RCLCPP_INFO(rclcpp::get_logger("DiffBotSystemHardware"), "Successfully activated!");

  return hardware_interface::CallbackReturn::SUCCESS;
}
```

Figure 79 Hardware Interface Code_3

```cpp
hardware_interface::CallbackReturn DiffBotSystemHardware::on_deactivate(
  const rclcpp_lifecycle::State & /*previous_state*/)
{
  RCLCPP_INFO(rclcpp::get_logger("DiffBotSystemHardware"), "Deactivating ...please wait...");

  RCLCPP_INFO(rclcpp::get_logger("DiffBotSystemHardware"), "Successfully deactivated!");

  return hardware_interface::CallbackReturn::SUCCESS;
}

void DiffBotSystemHardware::processFeedback(const std_msgs::msg::Int64MultiArray::SharedPtr msg)
{
  if (msg->data.size() >= 2) {
    hw_positions_[0] = ((msg->data[0] * 2.0 * M_PI) / 4096.0) * 0.033;  // left_wheel_joint position cumilative
    hw_positions_[1] = ((msg->data[1] * 2.0 * M_PI) / 4096.0) * 0.033;  // right_wheel_joint position cumilative
  }
}

void DiffBotSystemHardware::processVelFeedback(const std_msgs::msg::Int64MultiArray::SharedPtr msg)
{
  if (msg->data.size() >= 2) {
    hw_velocities_[1] = ((msg->data[0] * 2.0 * M_PI) / 60.0)*0.033;  // left_base_joint speed
    hw_velocities_[0] = ((msg->data[1] * 2.0 * M_PI) / 60.0)*0.033;  // right_base_joint speed
  }
}
```

Figure 80 Hardware Interface Code_4

```cpp
hardware_interface::return_type DiffBotSystemHardware::read(
  const rclcpp::Time & /*time*/, const rclcpp::Duration & /*period*/)
{
  rclcpp::spin_some(node_);
  return hardware_interface::return_type::OK;
}

hardware_interface::return_type DiffBotSystemHardware::write(
  const rclcpp::Time & /*time*/, const rclcpp::Duration & /*period*/)
{
  auto cmd_msg = std::make_shared<std_msgs::msg::Float32MultiArray>();
  cmd_msg->data.push_back(hw_commands_[0]);  // left motor command
  cmd_msg->data.push_back(hw_commands_[1]);  // right motor command
  cmd_publisher_->publish(*cmd_msg);

  return hardware_interface::return_type::OK;
}

}  // namespace robot_firmware

#include "pluginlib/class_list_macros.hpp"
PLUGINLIB_EXPORT_CLASS(robot_firmware::DiffBotSystemHardware, hardware_interface::SystemInterface)
```

Figure 81 Hardware Interface Code_5

7.5 HPP Header File for Hardware Interface

```hpp
#include <hardware_interface/system_interface.hpp>
#include <rclcpp_lifecycle/state.hpp>
#include <rclcpp_lifecycle/node_interfaces/lifecycle_node_interface.hpp>
#include <std_msgs/msg/float32_multi_array.hpp>
#include <memory>
#include <vector>
#include <string>
#include "std_msgs/msg/int64_multi_array.hpp"
#include "rclcpp/subscription.hpp"

namespace robot_firmware
{

using CallbackReturn = rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn;

class DiffBotSystemHardware : public hardware_interface::SystemInterface
{
public:
  DiffBotSystemHardware();
  virtual ~DiffBotSystemHardware();

  // Implementing rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface
  virtual CallbackReturn on_activate(const rclcpp_lifecycle::State &) override;
  virtual CallbackReturn on_deactivate(const rclcpp_lifecycle::State &) override;

  // Implementing hardware_interface::SystemInterface
  virtual CallbackReturn on_init(const hardware_interface::HardwareInfo &hardware_info) override;
  virtual std::vector<hardware_interface::StateInterface> export_state_interfaces() override;
  virtual std::vector<hardware_interface::CommandInterface> export_command_interfaces() override;
```

Figure 82 Header File of Hardware Interface_1

```hpp
  // Implementing hardware_interface::SystemInterface
  virtual CallbackReturn on_init(const hardware_interface::HardwareInfo &hardware_info) override;
  virtual std::vector<hardware_interface::StateInterface> export_state_interfaces() override;
  virtual std::vector<hardware_interface::CommandInterface> export_command_interfaces() override;
  virtual hardware_interface::return_type read(const rclcpp::Time &, const rclcpp::Duration &) override;
  virtual hardware_interface::return_type write(const rclcpp::Time &, const rclcpp::Duration &) override;

private:
  std::shared_ptr<rclcpp::Node> node_;
  rclcpp::Subscription<std_msgs::msg::Int64MultiArray>::SharedPtr feedback_subscription_;
  rclcpp::Subscription<std_msgs::msg::Int64MultiArray>::SharedPtr velocity_subscription_;
  rclcpp::Publisher<std_msgs::msg::Float32MultiArray>::SharedPtr cmd_publisher_;

  void processFeedback(const std_msgs::msg::Int64MultiArray::SharedPtr msg);
  void processVelFeedback(const std_msgs::msg::Int64MultiArray::SharedPtr msg);

  double hw_start_sec_;
  double hw_stop_sec_;
  std::vector<double> hw_commands_;
  std::vector<double> hw_positions_;
  std::vector<double> hw_velocities_;

};  // namespace autonav_firmware
}
#endif  // DIFFBOT_SYSTEM_HPP
```

Figure 83 Header File of Hardware Interface_2

7.6 ROS 2 Driver MPU-6050

The ROS 2 driver for the MPU-6050 facilitates the integration of the sensor into the ROS 2 ecosystem, enabling the sensor to publish data that other ROS 2 nodes can subscribe to and utilize. The driver handles the following tasks:

- **I2C Communication Setup**: Establishes communication with the MPU-6050.
- **Sensor Data Retrieval**: Continuously reads accelerometer and gyroscope data from the sensor.
- **Message Publishing**: Converts the raw sensor data into ROS 2 messages and publishes them to a specified topic.

7.6.1 Node Structure

- **Publisher Node**: The node reads the sensor data, processes it, and publishes it as an `Imu` message on the `imu/data` topic.

7.6.2 Example Workflow

- **Initialize Node**: The ROS 2 node initializes the I2C communication with the MPU-6050.
- **Read Sensor Data**: Raw data from the accelerometer and gyroscope is read at regular intervals.
- **Publish Messages**: The raw data is converted into `sensor_msgs/Imu` messages and published for other nodes to use.

7.7 How the IMU Translates its Readings into /imu Message

The MPU-6050 provides raw accelerometer and gyroscope data. This data is then translated into an `Imu` message in ROS 2, which includes orientation, angular velocity, and linear acceleration.

7.7.1 Mathematical Explanation and Theory

7.7.1.1 Orientation Estimation

Orientation is estimated using sensor fusion algorithms that combine accelerometer and gyroscope data to determine the device's orientation in space. Common algorithms include the Complementary Filter, Kalman Filter, and Madgwick Filter.

- **Gyroscope Integration**: The gyroscope measures angular velocity (ω). Over a small-time interval (Δt), the change in orientation ($\Delta\theta$) can be approximated as:

$$\Delta\theta = \omega \cdot \Delta t \Delta\theta = \ \omega \cdot \Delta t \Delta\theta = \omega \cdot \Delta t$$

Integrating these changes over time gives orientation. However, gyroscope data alone suffers from drift.

- **Accelerometer**: Measures linear acceleration including gravity. It provides an absolute reference for orientation but is noisy.
- **Complementary Filter**: Combines gyroscope and accelerometer data:

$$\theta = \alpha(\theta gyro) + (1 - \alpha)(\theta acc)$$

Where $\theta gyro$is the integrated gyroscope data, θacc is the orientation from the accelerometer, and α\alphaα is a tuning parameter.

7.7.2 Linear Acceleration

The accelerometer provides raw acceleration data along the X, Y, and Z axes. To obtain the linear acceleration, the gravitational component must be subtracted. The gravity component is determined using the orientation estimated from the sensor fusion algorithm.

7.7.3 Angular Velocity

The gyroscope provides raw angular velocity data around the X, Y, and Z axes, which is used directly in the `Imu` message.

Translating to `sensor_msgs/Imu`

The `sensor_msgs/Imu` message includes:

- **Orientation**: Quaternion representation derived from the sensor fusion algorithm.
- **Angular Velocity**: Direct readings from the gyroscope.
- **Linear Acceleration**: Accelerometer readings with gravity compensated.

7.8 Calibration Process

Calibrating the MPU-6050 involves determining the offsets for both the accelerometer and gyroscope to ensure accurate readings.

7.8.1 Steps in Calibration

- **Setup and Initial Readings**
  - **Hardware Setup**: Connect the MPU-6050 to the microcontroller or single-board computer via I2C.
  - **Software Setup**: Initialize the sensor and read the raw accelerometer and gyroscope data.
- **Collect Raw Data**
  - Place the sensor on a flat, stable surface to ensure it is not moving.
  - Collect a series of readings for both the accelerometer and gyroscope over a period (e.g., 1000 readings).

- **Calculate Offsets**
    - **Gyroscope Offsets:** Since the sensor is stationary, the gyroscope readings should be close to zero. Average the collected gyroscope data to determine the offsets.
    - **Accelerometer Offsets:** The accelerometer should read approximately 0g on X and Y axes and 1g on the Z-axis when lying flat. Average the accelerometer data and determine the offsets accordingly.
- **Apply Offsets**
    - Adjust the raw sensor readings by subtracting the calculated offsets to get the calibrated values.

7.8.2 Practical Implementation

The practical implementation involves:

- Reading Raw Data: Continuously read data from the accelerometer and gyroscope.

- Calculating Average: Compute the average value for a large number of samples while the sensor is stationary.

- Subtracting Offsets: Subtract the calculated offsets from subsequent sensor readings to obtain calibrated data.

Calibrating the MPU-6050 ensures accurate measurements by compensating for any inherent biases and offsets, thereby improving the reliability and accuracy of the motion and orientation data provided by the sensor.

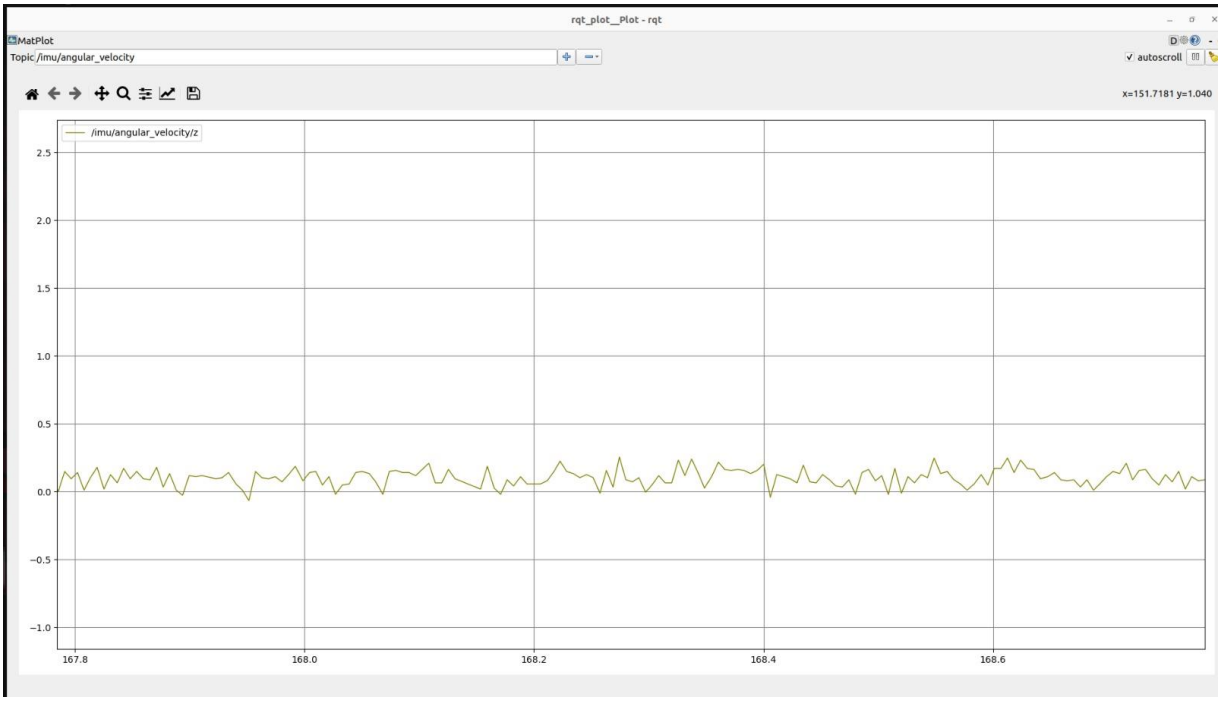


Figure 84 IMU Test

7.9 System Architecture

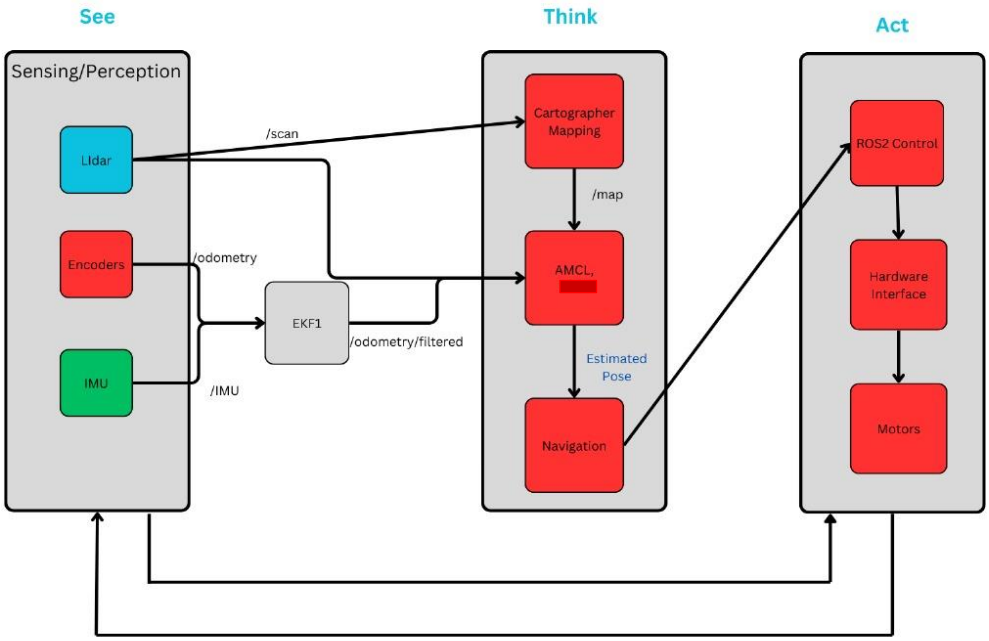


Figure 85 See, Think, Act system Architecture

7.10 Slam in Real Time

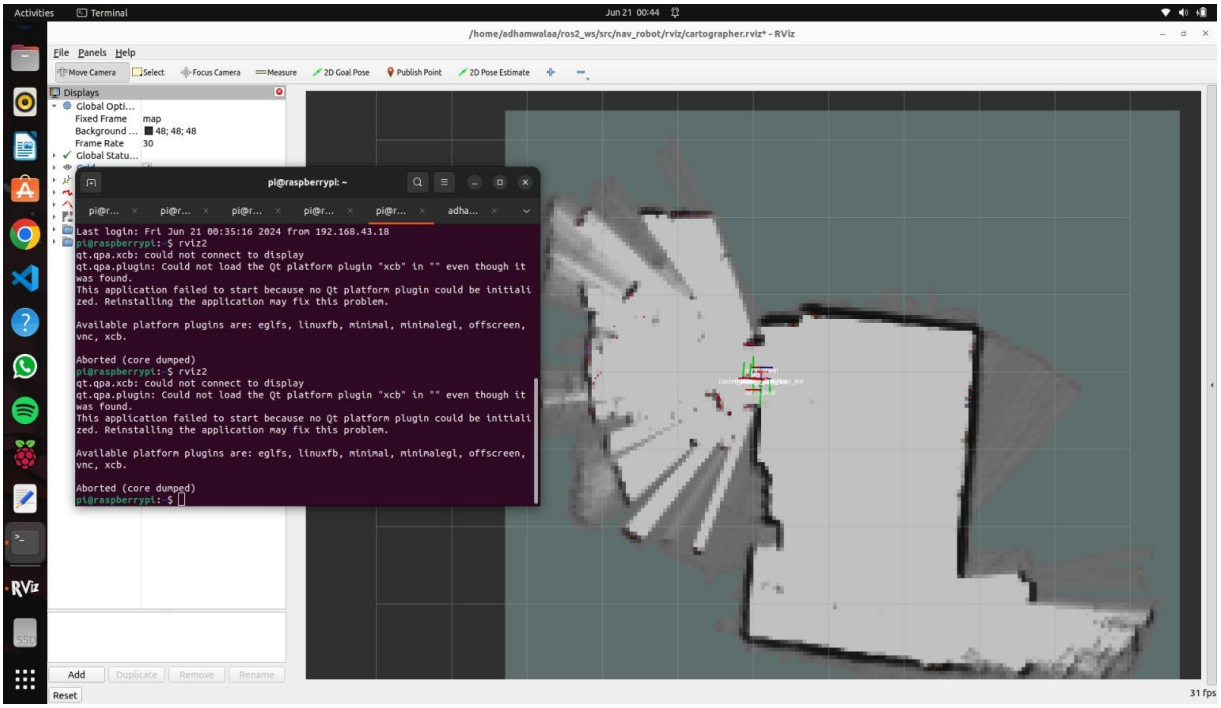


Figure 86 Real Time SLAM

7.11 Communication Protocols

7.11.1 Esp32 and Raspberry-Pi (Serial Communication)

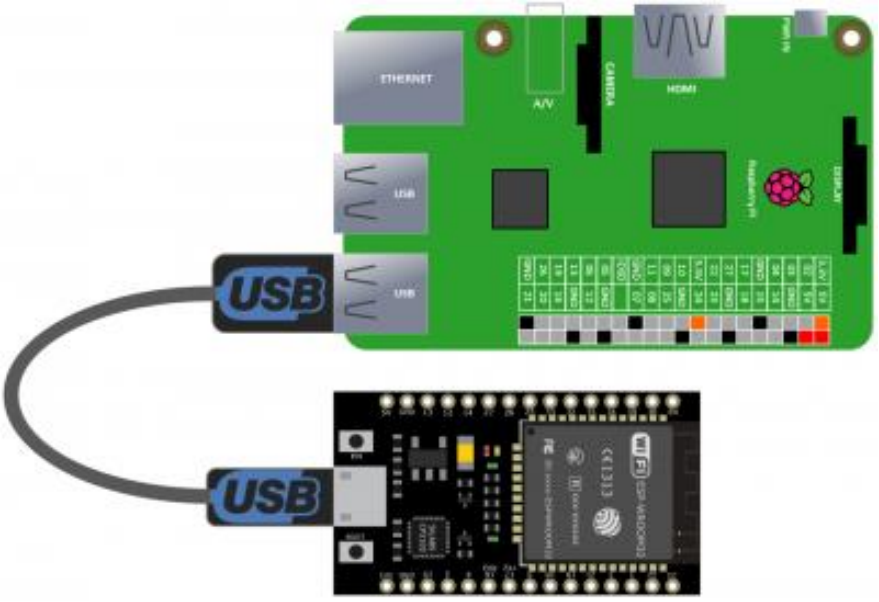


Figure 87 raspberry-pi and esp32 serial connection

7.11.2 IMU MPU6050 and Raspberry-Pi (I2C)

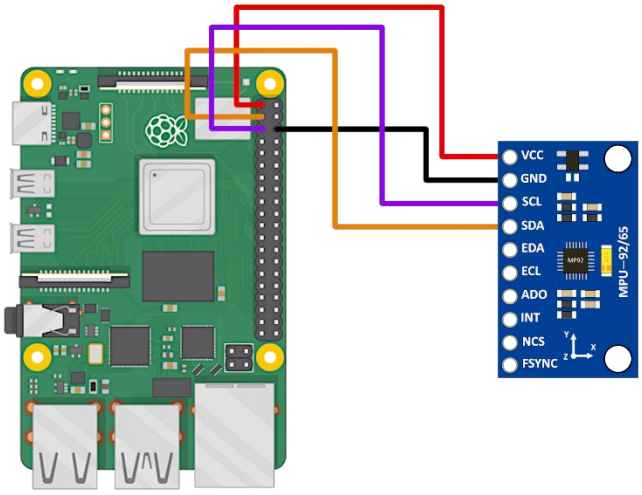


Figure 88 raspberry-pi and IMU I2C connection

7.11.3 LIDAR and Raspberry-Pi (Serial Communication)

Figure 89 raspberry-pi and LIDAR serial connection

7.11.4 Application and Raspberry-Pi (WIFI Communication)

Figure 90 raspberry-pi WIFI connection

8. Mobile Application

8.1 Introduction

The Mobile App revolutionizes the logistics and delivery processes within a warehouse setting by harnessing the power of automation and robotics. This application leverages Raspberry Pi-powered robots to transport items efficiently and reliably from one location to another. Users can manage orders, track deliveries, and ensure that items are moved to their intended destinations seamlessly. This chapter delves into the intricacies of the app, exploring its user interface, functionality, and the detailed workflow that guides users from order initiation to delivery confirmation.

8.2 User Interface and Workflow

The Mobile App is designed with a user-centric approach, ensuring that the interface is intuitive and easy to navigate. The app comprises five primary screens, each dedicated to a specific aspect of the order and delivery process. This section provides an in-depth look at each screen, highlighting their purposes and functionalities.

8.3 Screens and Functionalities

8.3.1 Entering ID and Location

- **Purpose:**

  The initial screen serves as the gateway to the app, allowing users to authenticate and set their location within the warehouse.

- **Functionality:**

- ID Input: Users are prompted to enter their valid ID, which the app verifies against a secure database to ensure authenticity.

- Location Selection: Users choose their current location from a predefined list of well-known points within the warehouse. This ensures that the robot knows where to pick up the items.

- **Detailed Steps:**

- Open the App: Launch the Mobile App on your Android device.

- Enter Valid ID: Input your ID in the designated field. This step is crucial for authentication

and ensuring that only authorized personnel can place orders.

- **Select Location:** Choose your current location from the dropdown menu. The locations are predefined to match the layout of the warehouse, ensuring accuracy in pickup points.

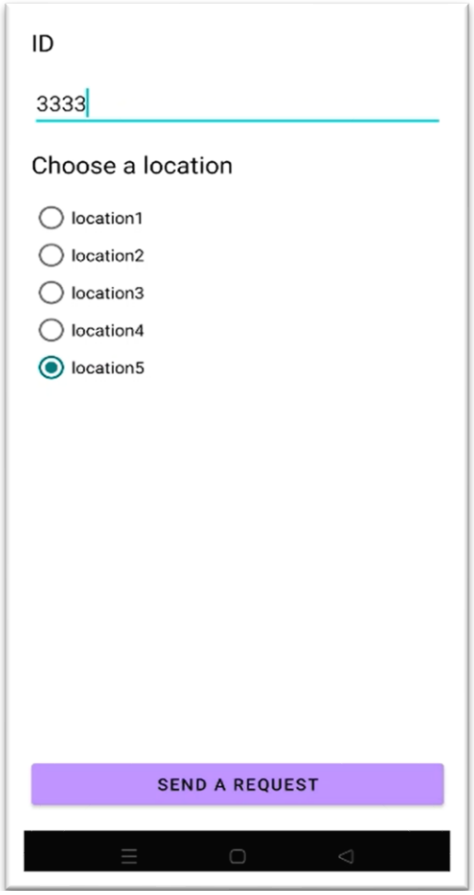


Figure 91 Application Layout_1

8.3.2 Checking Robot Arrival Status

- **Purpose:**

This screen is designed to keep the user informed about the robot's current status and whether it has arrived at the specified pickup location.

- **Functionality:**

- **Interval Checks:** The app checks the robot's status every 30 seconds.

- **Remaining Time Display:** If the robot has not yet arrived, the screen shows the remaining time until the next check.

- **Navigation:** If the robot has arrived, the app automatically shows a button to allow user to scan the QR Code.

- **Detailed Steps:**

- Monitor Status: After entering your ID and location, the app displays the robot's status, indicating whether it is in route or has arrived.

- Check Interval: The app updates the status every 30 seconds.

- Remaining time: If the robot does not yet arrive, the screen shows the remaining time until the next status check.

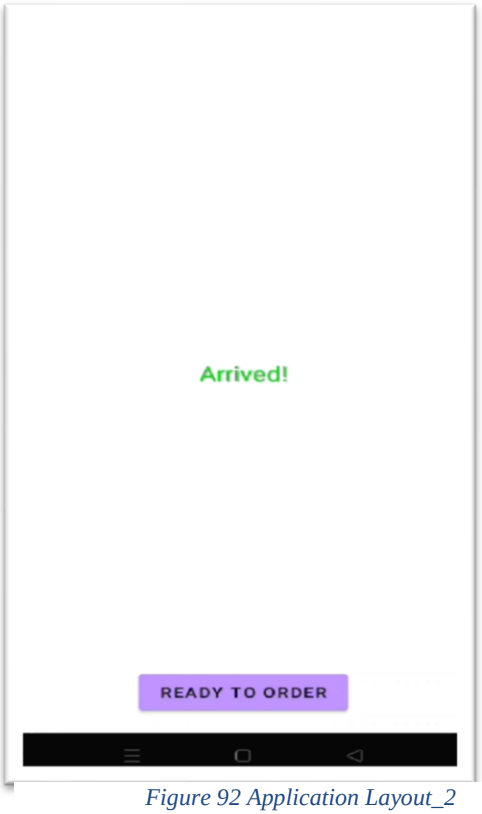


Figure 92 Application Layout_2

8.3.3 QR Code Scanning

- **Purpose:**

This screen enables users to scan the QR code on the robot, thereby linking it to their order and confirming its readiness to receive items.

- **Functionality:**

- QR Code Scanner: The app utilizes the device's camera to scan the QR code placed on the robot. This ensures that the correct robot is being used for the delivery.

- Verification: Upon scanning, the app verifies the robot's identity and prepares it for the delivery task.

- **Detailed Steps:**

- Initiate Scan: On the QR code scanning screen, press the scan button to activate the camera.

- Scan QR Code: Align the camera with the QR code on the robot. Ensure that the code is clearly visible and within the scanning frame.

Figure 93 Application Layout_3

- Verification: The app verifies the robot's ID and confirms its readiness to accept the order.

8.3.4 Choosing the Destination

- **Purpose:**

This screen allows users to specify the delivery destination for the robot, ensuring that items are transported to the correct location within the warehouse.

- **Functionality:**

- Destination Input: Users select the destination from a predefined list of warehouse locations.

- Dispatch Confirmation: Once the destination is set, the app confirms the order and dispatches the robot.

- **Detailed Steps:**

- Select Destination: From the list of available destinations, choose the location where the items need to be delivered.

- Confirm and Dispatch: After selecting the destination, confirm the order. The app will then dispatch the robot to the specified location.

Figure 94 Application Layout_4

8.3.5 Final Delivery Check

- **Purpose:**

This screen tracks the robot's journey to the destination and notifies users upon successful delivery, allowing them to place new orders if needed.

- **Functionality:**

- Interval Checks: The app checks the robot's status every 30 seconds.

- Remaining Time Display: If the robot has not yet arrived, the screen shows the remaining time until the next check.

- Progress Tracking: Users can monitor the robot's progress, seeing updates on whether it is in route or has arrived at the destination.

- **Detailed Steps:**

- Monitor Delivery: The app displays the robot's journey status, indicating its progress towards the destination.

- Check Interval: The app updates the status every 30 seconds.

- Remaining Time: If the robot has not yet arrived, the screen shows the remaining time until the next status check.

- Place New Order: After confirming the delivery, users have the option to initiate a new order, starting the process anew from the ID and location input screen.

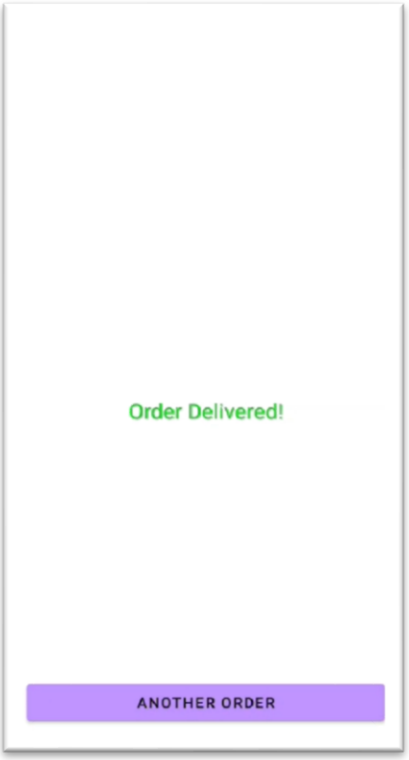


Figure 95 Application Layout_5

9. Cost

Name Of Product/Process	Cost	Quantity	Total Cost
LIDAR	8000	2	16000
Raspberry-Pi	6000	2	12000
ESP32	400	2	800
Servo Motor (fb5311m-360)	1000	4	4000
IMU (MPU 6050)	150	2	300
Power Bank	450	2	900
Wheels	50	4	200
Laser Cutting	1400	-	1400
Wires and Bolts	75	-	75
3D Printing	500	-	500
Total	-	-	36,175 EGP

Table 8 Project Cost

10. Future Work

10.1 Integration with IoT Devices

Investigate the integration of the system with Internet of Things (IoT) devices to enable real-time monitoring and data analytics. This will help optimize warehouse operations by providing insights and improving decision-making processes.

10.2 Machine Learning for Predictive Maintenance

Apply machine learning techniques to anticipate maintenance needs for the robots. This approach aims to reduce downtime and enhance operational efficiency by predicting and addressing potential issues before they occur.

10.3 Energy Efficiency

Explore methods to improve the energy efficiency of robots. This could involve researching better battery technologies or developing more efficient routing algorithms to minimize energy consumption during operations.

10.4 Security Enhancements

Implement additional security measures to safeguard the system against cyber threats. This will help ensure the integrity and confidentiality of the data, protecting the warehouse operations from potential security breaches.

10.5 Collaboration with External Systems

Examine the potential for integrating the warehouse system with external systems such as supply chain management software and ERP systems. This integration would facilitate better coordination and efficiency across various operational processes.

11. Conclusion

In conclusion, this report aims to explore the impact of integrating a robot-assisted delivery system within a warehouse setting to improve item transportation efficiency and accuracy. Through a detailed analysis of the implemented system, user interactions, and performance metrics, we found that using Raspberry Pi-powered robots and a user-friendly mobile application significantly enhances the logistics processes in the warehouse.

Our findings indicate that key factors contributing to this improvement include reduced manual handling, improved accuracy in item delivery, and an intuitive interface for order management. While there are challenges such as the initial setup and the need for regular maintenance, the overall effect on operational efficiency and accuracy is positive. Users reported higher satisfaction and more streamlined workflows, leading to improved overall performance.

Based on these findings, we recommend that warehouses consider adopting such robot-assisted systems to enhance their logistics processes. This approach can help mitigate inefficiencies while leveraging the benefits of robotic assistance. Additionally, investing in continuous training for users and regular maintenance can further optimize system performance and longevity.

In closing, the integration of a robot-assisted delivery system in warehouse logistics represents a significant advancement in operational efficiency and accuracy. As technology continues to evolve, it is essential to address the challenges and harness the advantages of such systems to ensure sustained productivity and user satisfaction in warehouse operations.

12. References

1. https://drive.google.com/file/d/10OQW2MljqwRQbcF-q8vISXSG9Nmi636q/view?usp=sharing,

2. https://drive.google.com/file/d/1AAfrGgJxnODkHvbikryxXKObvlM0jvgO/view?usp=sharing,

3. https://drive.google.com/file/d/1hSRGVGumnz2J4hWOJk1w2ZjwzUe3LCVD/view?usp=sharing,

4. https://drive.google.com/file/d/1o9j0KEnRstbKL1V7b1oTCUmfaq7aYCzr/view?usp=sharing,

5. https://drive.google.com/file/d/1tSbqTtirukF-_jzqAJhlawBitr1uYdTB/view?usp=sharing

6. https://www.youtube.com/watch?v=T3if0aPj0Eo&ab_channel=ClearpathRoboticsbyRockwellAutomation

7. Development of an AGV system using MBSE method and multi-agents' technology (hal.science)

8. A Concise Introduction to Robot Programming with ROS2 book by Francisco Martin Rico

9. ROS documentation (https://docs.ros.org/)

10. https://docs.nav2.org/index.html

11. https://docs.ros.org/en/humble/index.html

12. "AGV Cost Estimation, how much does an automated guided Vehicle System Cost", by AGV network (http://www.agvnetwork.com/)

13. "How does AGV Safety System work?" by AGV Network & SICK, (http://www.agvnetwork.com/)