People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA - Boumerdes

Institute of Electrical and Electronic Engineering

Department of Power and Control

Final Year Project Report Presented in Partial Fulfillment
of the Requirements of the Degree of

'Master'

In Control Engineering

Option: Control Engineering

Title:

Autonomous Navigation System for Mobile Robot In Dynamic Environment

Presented By:

Bouaiss Manar

Supervisor:

Dr. Belaidi Hadjira

Co-Supervisors: Della Djamel Kahlouche Shouhila

Registration Number:...../2023

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to **Allah**; the One to whom all greatness belongs; for his mercy and guidance that lightened my path during moments of darkness and despair; for giving me the strength to persevere through the most challenging times.“And my success is not but through Allah”

I would like to express my appreciation to my Supervisors for their valuable feedback and consistent encouragement throughout this project, for inspiring me to strive for excellence in my work.

I would also like to extend my thanks to Center for the Development of Advanced Technologies **(CDTA)** for providing me with the necessary resources, facilities, and opportunities to conduct my research.

To my friends and colleagues, who have been my steadfast companions, trusted confidants, and valuable collaborators on this arduous quest for knowledge. This work stands as a testament to our collective strength, resilience, and belief in the power of collaboration.

Lastly; I extend my heartfelt appreciation to **INELEC** for being my second home for the last five years, It has been a remarkable experience full of invaluable knowledge, excitement, and deadlines. I am deeply grateful to its exceptional professors, dedicated staff, and brilliant fellow students who have contributed to making INELEC cherished place.

DEDICATION

I wholeheartedly dedicate this work to my parents for their unwavering love and support along this journey.

To my sisters who were my source of inspiration, strength, and motivation.

And I proudly dedicate this work to myself, as a testament to my resilience, hard work invested and the sacrifices I have made along the way. It stands as a tribute to the commitment and the countless hours of effort poured into bringing this project to fruition.

DECLARATION OF AUTHORSHIP

I, Manar BOUAISS, solemnly declare that this report, titled Autonomous "Navigation System for Mobile Robot In Dynamic Environment", is entirely my own work and has not been previously submitted, either in its entirety or in part, for the fulfillment of any other academic degree or diploma.
The experimental work is entirely my individual effort; the collaborative contributions have been indicated clearly and acknowledged. Due references have been provided on all supporting literature and resources.

ABSTRACT

Mobile robots have found numerous applications across various domains, such as agriculture, mining, construction, and military. The versatility and high mobility of ground wheeled robots make them a popular choice for navigating complex and hazardous environments. However, achieving autonomous navigation is a challenging task as it relies on critical factors such as perceiving the robot's surroundings, create a representative map of the environment, and localize itself in real-time. In this project, we present an approach for achieving autonomous navigation for a two-wheeled differential ground robot utilizing the robot operating system (ROS) environment .The objective is to create a navigation system that is robust and dependable, capable of enabling the robot to navigate autonomously through an unknown environment and reach a specified destination. The proposed navigation system incorporates the use of the SLAM algorithm for mapping and localization, as well as the A* algorithm for path planning. The developed system is tested and evaluated within simulation environment, which demonstrates the navigation system's effectiveness and precision. Finally, the proposed navigation system is implemented on a mobile robot , and its performance is evaluated in real-world situations. The outcomes indicate that the developed navigation system can effectively enable the robot to navigate autonomously while avoiding obstacles and arriving at its destination with precision. This project topic is part of the research project to develop a mobile robot platform intended for surveillance purposes called ROSMI, which aims to adapt perception, localization, and navigation techniques to a surveillance robot that can perform tasks such as threatening and sensitive sites. These techniques are implemented on the robotic platform, equipped with a Raspberry Pi, a camera, and a Lidar. The developed application must facilitate the mobile robot's autonomous navigation while transmitting in real-time the acquired video stream from the camera to a remote control station.

Keywords: Navigation System, ROS, SLAM, Navigation Stack.

NOMENCLATURE

WMR	Wheeled Mobile Robot
UGV	Unmanned Ground Vehicle
UAV	Unmanned Aerial Vehicle
UUV	Unmanned Underwater Vehicle
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
LIDAR	Light Detection and Ranging
AMCL	Adaptive Monte Carlo Localization
AI	Artificial Intelligence
GUI	Graphical User Interface
API	Application Programming Interface
CPU	Central Processing Unit
RGB-D	Red Green Blue - Depth
PID	Proportional-Integral-Derivative
SSH	Secure Shell
VNC	Virtual Network Computing
PWM	Pulse Width Modulation
URDF	Unified Robot Description Format
GMAPPING	Grid-based FastSLAM Mapping

TABLE OF CONTENTS

LIST OF TABLES

TABLE **Page**

List of Figures

INTRODUCTION

The field of robotics has undergone remarkable transformations in recent years, leading to groundbreaking advancements that have revolutionized numerous industries and domains. This rapid progress has played a pivotal role in seamlessly integrating robots into various facets of human society. Moreover, as robots increasingly interact and collaborate with humans, the need for safe and reliable navigation systems becomes increasingly paramount. To effectively navigate their surroundings, robots require a robust and comprehensive navigation system. This system encompasses various components, including perception, planning, and control, working together to enable autonomous movement and interaction with the environment. The success of these missions relies heavily on the development of adaptable navigation systems that facilitate movement across vast and unfamiliar landscapes. By continually advancing navigation technologies and algorithms, research strives to enhance the capabilities of robots, enabling them to operate in complex and dynamic environments. This ensures the optimal performance of robots with diverse settings.

This report deals with the development and implemention of navigation system on a mobile robot, Hence, the remaining of this report is organized as follows: **Chapter 1** provides a comprehensive overview of robotics, with a specific emphasis on unmanned ground vehicles and their architectural aspects. The fundamental elements of robot design and structure are thoroughly examined with a detailed exploration on the navigation process within robots, encompassing various components such as perception, localization, and control. The significance of these components in facilitating autonomous movement and interaction with the environment is highlighted followed by an introduction to robot operating system (ROS) and its crucial role in the development of robotics systems.. The primary objective of this chapter is to establish a solid foundation of knowledge and comprehension in the field of robotics, laying the groundwork for further investigations and advancements in the subsequent chapters.

In chapter 2 the tools used in this study are presented, recognizing that the initial stage in establishing a functional navigation system is the precise setup of the robot. This involves meticulous configuration of both the hardware and software components to guarantee flawless integration and compatibility. Additionally, a comprehensive examination of the proposed methods and their evaluation techniques is provided, encompassing the configuration of essential software packages such as SLAM (Simultaneous Localization

and Mapping) and the navigation stack.

Chapter 3 covers the configuration of network communication, enabling remote control of the robot, as well as the incorporation of ROS within the Raspberry Pi. The chapter also addresses the setup of the navigation environment. Finally, we explore the practical implementation of navigation algorithms, including fine-tuning parameters and the launch files to optimize performance.

The experimental results of the implemented navigation system is presented in **Chapter 4**, followed by evaluation and the discussion of the effectiveness of the developed system. Finally the future work and further developments are presented in the conclusion.

CHAPTER 1

INTRODUCTION

This chapter presents overview of robotics, with a focus on unmanned ground vehicles including their overall architecture design and structure of a robot. Additionally, a comprehensive explanation of the navigation process of robots is presented, along with an introduction to the Robot Operating System (ROS) and its various tools and concepts that are essential for understanding and implementing robotic systems.

1.1 Overview of robotics

Robotics is a multidisciplinary area of research that focuses on the design, construction, operation, and utilization of robots. It is a fusion of various fields, such as computer science, electrical engineering, and mechanical engineering, to create machines that can perform tasks typically carried out by humans. The field of robotics has seen significant advancements in recent years, resulting in the development of robots for diverse applications, including manufacturing, healthcare, and education [1]. The goal of robotics research is to create intelligent and autonomous machines that can sense, reason, and act in the physical world and interact seamlessly and safely with humans. Achieving this requires the implementation of various techniques, algorithms, and machine learning strategies that enable robots to learn from their environment, make decisions, and execute actions based on perceived conditions. . Robots are classified into various types based on their intended function and application. The most common types include industrial robots, humanoid robots, and autonomous vehicles. Each type of robot possesses unique capabilities and characteristics, making them suitable to perform specific industry task. The flexibility and adaptability of robotics design allows robots to be developed for a wide range of applications in various industries.

1.2 Autonomous mobile robots

Autonomous mobile robots are robotic systems that operate independently in their environment and can move without continuous human intervention or control. Equipped with various sensors, perception systems and embedded intelligence, these robots perceive their surroundings, make decisions, plan actions and perform tasks without the need for constant human guidance [2]. By moving autonomously and interacting with their environment, these robots can perform a variety of functions such as exploration, transport, inspection, surveillance and support with a high degree of autonomy and adaptability.

There are many different types of autonomous mobile robots, each designed for specific applications and environment (Figure 1.1). Some of the most common types are:

1. **Wheeled robots (WMR/UGV):** These robots are usually equipped with 2-4 wheels and can move efficiently on flat surfaces. They are often used indoors in warehouses, hospitals, and offices for tasks such as material handling, transportation, and surveillance.

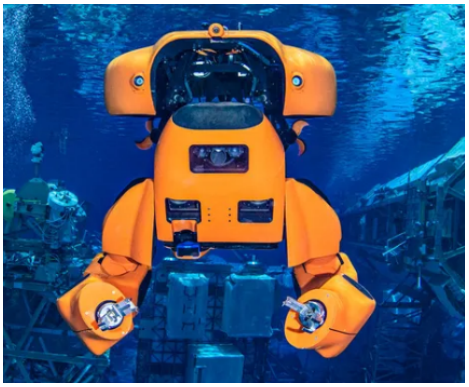
2. **Legged Robots:** Legged robots mimic the movements of animals and insects with their legs and limbs. These robots have the advantage of being able to navigate uneven terrain, stairs, or rough surfaces that are difficult for wheeled robots to traverse. Legged robots are used for search and rescue missions, reconnaissance, or military operations [3].

3. **Aerial Robots (UAV):** These robots are designed to fly and navigate through the air. They typically use rotors or wings for propulsion and can be used in a variety of environments, including indoors and outdoors. Aerial robots are widely [4] used for aerial photography, surveillance, delivery services, and environmental monitoring .

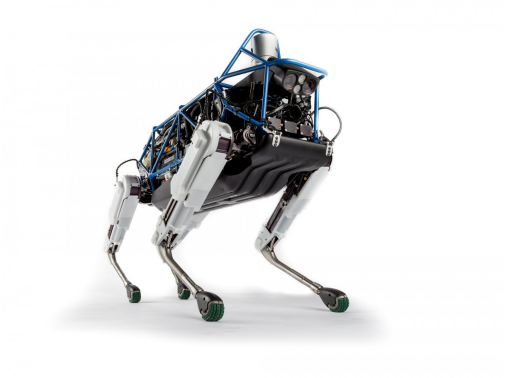
4. **Underwater Robots (UUV):** These robots are specifically designed for remotely operation or autonomous underwater (AUV) operations. They are used for marine research, underwater exploration, pipeline inspection, and deep-sea research [5].

This work focuses on the use of Wheeled Mobile Robot(WMR), which are preferred over other types of robot designs because they are simpler in design, production, and programming processes for moving on flat terrain and easier to control [6]. Wheeled robots can use different locomotion methods depending on their number of wheels. Generally, two-wheeled robots control their heading by differential steering method, which is applicable to not only three-wheeled but also four-wheeled robots.

Aerial Robot [7]



Underwater Robot [8]



Legged Robot [9]

Wheeled robot [10]

Figure 1.1: Different type of Autonomous mobile robots

1.2.1 Robot Architecture

1.2.1.1 Perception

The concept of perception in mobile robotics refers to a system's capacity to gather, interpret, and coordinate data that will let the robot operate and interact with its environment accurately and in real-time despite complexities such as noise, occlusions, and dynamic changes. Additionally, the robot must have various sensors that assess both its internal status and the environment in which it evolves the extraction of meaningful information in order for the robot to perform its task.

1.2.1.2 Localization

Localization for robots refers to the ability of a robot to determine its own position and orientation within a given environment. The most widely used technique for relative localization is **odometry**. It involves initialization relative to the surroundings and optical encoders attached on the axis motors to determine the basic wheel rotations. The approach is based on establishing the robot's location and orientation in relation to an absolute reference frame.

The following diagram (Figure 1.2) illustrates the localization process:

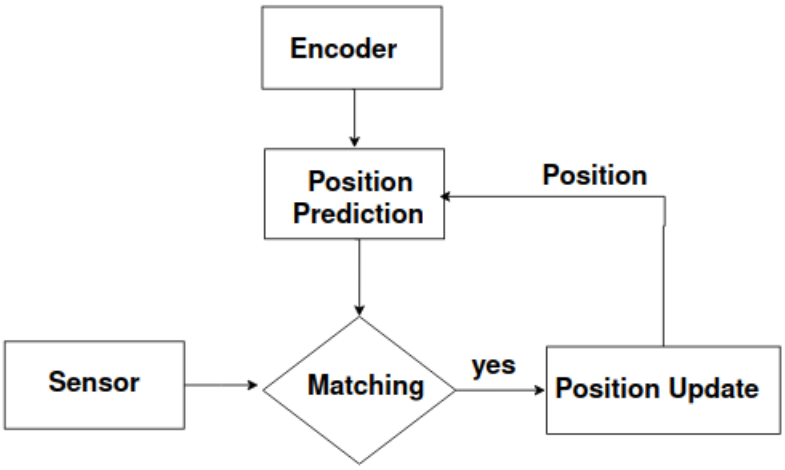


Figure 1.2: Localization system diagram [11]

1.2.1.3 Cognition

Cognition is a critical aspect of mobile robotics, involving the ability of the robot to gather data from its sensors, analyze it, and utilize it to make decisions based on its objectives and tasks. Perception is the foundation of cognition, as the robot's sensors collect data which is subsequently processed to construct maps of the environment, plan paths, and execute tasks [11]. Moreover, developing cognitive robots presents significant challenges, such as the development of robust and scalable algorithms, the integration of complex sensory inputs, and the requirement for high computing and storage capacity. Achieving cognitive capabilities in robots could enable them to operate autonomously and interact intelligently with their environment.

1.2.1.4 Motion Control

Motion control of a mobile robot refers to the process of controlling the movement and direction of the robot by adjusting the speed and direction of the wheels. This involves the use of sensors to measure the current status of the robot, as well as the desired motion , and then calculating the necessary wheel speeds to make the action. The control scheme typically includes a trajectory generator, a motion control law, and a steering control law. An efficient motion control system enables the robot to move while ensuring that it accomplishes its intended task precisely.
The diagram in Figure 1.3 provides a visual representation of the interconnected of the four key components in robotic systems:

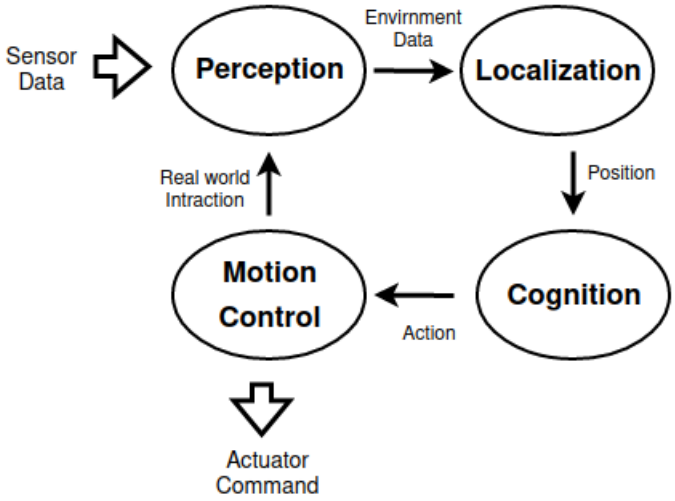


Figure 1.3: Architecture of Robotic System

1.3 Wheeled robotic locomotion:

The Differential Drive Wheeled Mobile Robots (DDWMRs) usually have two independently driven wheels and one or more unpowered wheels at the rear as a balance; as represented in Figure 1.4 The two fixed wheels are mounted on the same axis, a longitudinal axis, and a center wheel. The two fixed wheels' velocity and the steerable wheel's orientation determine the robot's direction of travel. The axis that corresponds with the fixed wheels and the axis of the steerable wheel meet at the point where its center of rotation is situated. In this configuration, the velocity of each wheel is controlled separately , this what gives the robot the flexibility to perform rolling motion. The robot rotates about a point that lies along their common left and right wheel axis, this point is known as the ICC - (Instantaneous Center of Curvature) The trajectories that the robot follows can be varied by varying the velocities of the two wheels [12]. The following equations describe the velocities of the right and left wheel respectively:

$$\omega(R + l/2) = Vr \tag{1.1}$$

$$\omega(R - l/2) = Vl \tag{1.2}$$

Where w is the rate of rotation about the ICC, where l is the distance between the centers of the two wheels, and R is the signed distance from the ICC to the midpoint between the wheels.

1.If Vl = Vr, The robot exhibits forward linear motion by moving in a straight line.

2. If Vl = -Vr, the robot rotates about the midpoint of the wheel axis (the robot rotates in place).

3. If Vl < Vr the robot deviates to the left , and if Vl > Vr the robot deviates to the right.

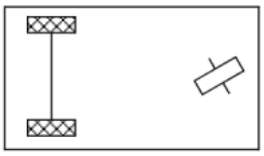


Figure 1.4: Mobile robot with tricycle kinematics [11]

1.4 Navigation

Navigation ability is the most crucial element of a mobile robot design. The objective for the robot is to move from one place to another taking into account the sensor data to achieve its goals, whether the environment is known or unknown. To do so , the robot should be able to combine the four perspectives, mentioned in the first chapter : Perception, localization, cognition, and motion control . The use of sensors to collect environment data this is important because it allows the robot to preform fundamental requirements for navigation such as mapping and obstacle detection;furthermore, to operate effectively, the robot needs to perform several tasks. It must accurately estimate its position and orientation within the environment. This requires cognitive abilities such as path planning and obstacle avoidance. Finaly, the robot must execute the planned algorithms and convert the intended path into control signals. These signals are then transmitted to the robot's actuators, enabling it to execute the desired path.

1.5 The Robotics Operating System (ROS)

The Robotics Operating System (ROS) is a free and open source robotics software framework that is used in both commercial and research applications.[13] In addition to being the first large-scale collaborative robotics project to offer a set of time-saving software tools for the development of a robot or robotic system, The major goal of ROS is to support code reuse in robotics research and development. It has quickly become the standard development process for many robotics research institutions and companies.The flexible ROS framework may be used to create software for robots. It consists of a collection of tools that make it simpler to create complex, reliable robot behaviors on a variety of robotic systems.

1.5.1 ROS Tools

1.5.1.1 ROS Concepts

ROS provides a set of powerful concepts and tools which plays the role of building blocks of ROS applications . In this context, it is essential to understand the core concepts of ROS and their applications, as they form the foundation of any ROS project.

Node: Process that use ROS APIs to perform computations.
ROS master: an intermediate program that connects ROS nodes.
ROS parameters server: program that runs with ROS master where parameters stored there can be accessed by all nodes.
Ros topics: named buses through which nodes can publish or subscribe.
ROS massages: the information send through topics.
ROS services: request and reply mechanism connection server node and client node.
ROS bag: Save and play back ROS topics and log data from robot to process it later.

Sending and receiving messages is the primary means through which ROS nodes communicate. Topics are the channels through which the messages are exchanged . Nodes can subscribe to a topic to receive information or publish messages on it. This process is illustrated in the diagram 4.2 :

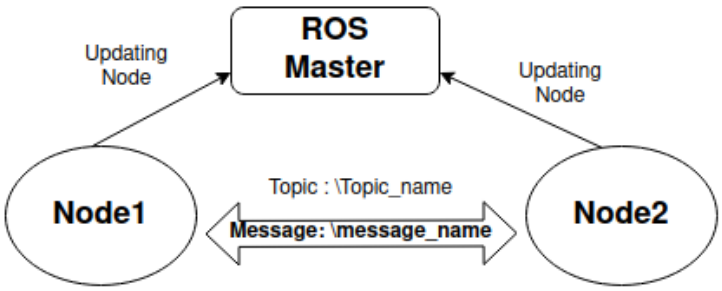


Figure 1.5: ROS communication block diagram

1.5.1.2 RViz

Rviz, is a visualization tool specifically designed for the Robot Operating System (ROS), possesses the capacity to effectively visualize and present data derived from ROS topics in both 2D and 3D formats. Within the ROS projects, Rviz plays a critical role in facilitating the visualization and analysis of simulated environments, as well as real-time hardware applications. Its extensive range of visualization features and interactive controls renders it an invaluable asset for the purposes of debugging, monitoring, and gaining

profound insights into the behavior and performance of ROS-based systems. By providing a comprehensive view of sensor data, robot models, and environmental elements, Rviz significantly enhances the development and analysis processes in the realm of robotics applications.

1.6 Conclusion

In conclusion, this chapter provided an overview of mobile robots and its various aspects, including the fundamental aspects of the robots to be able to perceive and interact with its environment, and navigation. The chapter also discussed the Robot Operating System (ROS) and its importance in robotics to preform navigation. The subsequent chapter will provide a more comprehensive analysis of the robot setup and navigation stack.

CHAPTER 2

METHODOLOGY

This chapter delves into the various aspects of setting up a robust navigation system for robotic applications. It begins by discussing the network configuration and communication setup which provides the remote control of the robot. Afterwards, the configuration of the Raspberry Pi for seamless integration with the Robot Operating System (ROS) is explained. Moreover,it demonstrates the navigation environment setup and the map building technique. Finally, the chapter addresses the practical application of navigation stack, including algorithms of the launch files and fine-tuning the parameters.

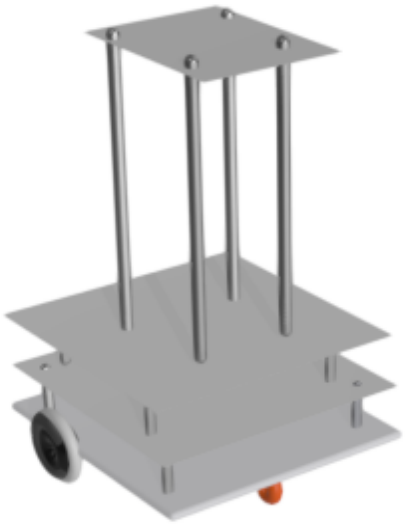
2.1 Robot setup

A detailed understanding of the robot's physical is essential for creating a robot that meets the specific requirements and facilitates the integration of software components. This includes aspects such as the robot's mechanical design, as well as, the used hardware components.

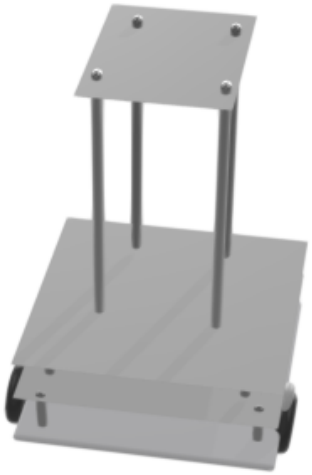
2.1.1 Mechanical design

The robot is made up from metal which provides a low cost prototype and provides a strong and solid architecture for the robot to be able to work in outdoor environment.
The robot platform is made up of three layers: The base layer, medium layer and top layer as showen in Figure 2.1.
For the hardware connection on the base layer consists of two wheels with one on each side of the chassis; they are linked to the DC motors and encoders which are required to provide odometry information. A third wheel is centered at the chassis to make the movement of the robot possible.

The platform's specifications and measurements are illustrated in Table 2.1:



Back View of the Model



Front View of the Model

Figure 2.1: 3D Model for the robot

Table 2.1: Robot platform specifications

Specification	**Value (cm)**
Chassis length	45
Chassis Width	36
Wheel Width	2
Wheel Diameter	7
Distance between Wheels	51

2.1.2 Hardware Setup

The Rplidar: RP-LiDAR (Light Detection and Ranging) showen in Figure 2.2 is laser scanner sensor that provides 360-degree scan field, rotating at 5.5 Hz/10 Hz. It is based on laser triangulation ranging principle and uses high-speed vision acquisition and processing. The sensor utilize a sequential process to detect obstacles and produce laser scan data. Initially, the LiDAR sensor emits laser beams in multiple directions, covering a 360-degree horizontal field of view; upon encountering objects in the environment; these laser beams reflect back towards the sensor. Through precise time-of-flight measurements, the sensor calculates the distances between itself and the detected objects. By combining these distance measurements, a comprehensive point cloud representation of the environment is formed.[14] Algorithms can then be applied to analyze the point cloud data, identifying clusters of points that correspond to physical obstacles. Furthermore, the laser scan data can be visualized through 3D point representations or top-down maps, providing an intuitive display of the detected obstacles and their spatial relationship with the LiDAR sensor.

Figure 2.2: RPLIDAR A2M8 360 Degree 2D Laser [15]

The data acquisition and analysis serves as a foundation for several key functionalities, including the creation of highly accurate maps that depict the surroundings in detail. As the RPLIDAR sensor scans the surroundings and generates a point cloud, the processed data is utilized to identify and delineate obstacles or occupied regions. These areas are then represented as black edges in the map to indicate their presence and outline their boundaries. Conversely, the remaining unoccupied or free spaces are displayed as white, providing a visual representation of the accessible areas within the environment. Additionally, robots can leverage laser scan data to detect obstacles in real-time, allowing them to promptly respond and navigate safely through their environment. Furthermore, laser scan data facilitates the process of localization, enabling robots to determine their precise position relative to the surrounding environment. By incorporating laser scan data into their decision-making processes, robots can plan optimal paths for navigation, ensuring efficient and effective movement towards their goals.

DC motor with an integrated encoder: The Pololu 12 V 37D Metal Gearmotor is used in this project , which is a powerful brushed DC motors paired with 37mm-diameter gearbox with ratio of 1:50 . Notably, this gearmotor includes an integrated 64 CPR quadrature encoder situated on the motor shaft enabling precise feedback on position and speed.[16]

Figure 2.3: Pololu 12 V 37D Gearmotor with encoder [16]

Batteries for Energy Storage

220v 110v AC Output Portable UPS Battery compact and portable backup battery that offers uninterrupted power supply (UPS) capabilities. It has been specifically designed to provide continuous AC power output options of either 220V or 110V, with a capacity of 150 watt-hours (WH). The device incorporates multiple input and output features, including a DC input for charging within the range of 12V to 20V, with a maximum power input of 65W.

Rechargeable Lipo Battery: the battery shown in Figure 2.5 features a high energy density, allowing it to store a large amount of energy in a small and lightweight package. Additionally, this battery incorporates built-in protection circuitry, effectively safeguarding against potential risks such as overcharging, over-discharging, and short-circuiting. These safety measures ensure dependable and secure operation of the battery.

Figure 2.4: Versatile Portable UPS Battery

Figure 2.5: 25000mAh Rechargeable LiPo Battery [17]

Wheels: lightweight wheels; shown in figure 2.6are used with an inner portion composed of a durable plastic material, providing structural integrity. The outer layer of the wheel is coated with high-quality rubber, which enhances traction during operation. These wheels exhibit the capability to withstand heavy loads. With a radius of 5cm, the wheels are directly driven by gearmotors, which provide the necessary rotational force for locomotion. These wheels are designed to be steerable, allowing precise control over the robot's direction of movement. The steering mechanism integrated with the wheels enables the robot to navigate and maneuver in various directions, enhancing its mobility and versatility.

Figure 2.6: Rubber-Coated Plastic Wheels

The RoboClaw: A motor controller designed for mobile robots that can be used to control the speed and direction of two brushed DC motors. It provides advanced control features, such as acceleration and deceleration control, stall detection, and speed control.

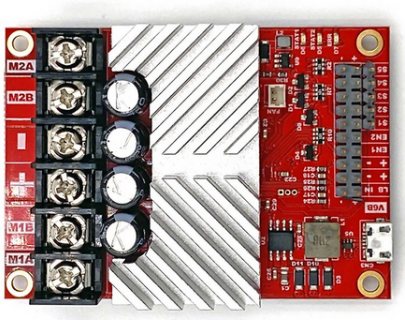


Figure 2.7: RoboClaw 2x30A Motor Controller [18]

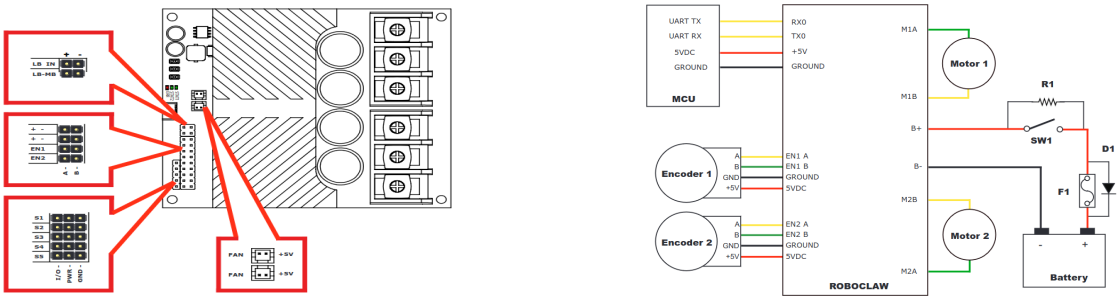


Figure 2.8: Roboclaw and Motor connection [19].

Raspberry Pi : is a versatile single-board computer , with its compact size, low power consumption, and GPIO pins, the Raspberry Pi can interface with various sensors, actuators, and motor controllers due to its connectivity options such as USB, Ethernet, and wireless interfaces (Wi-Fi, Bluetooth). In this project the Raspberry Pi 4 presented in Figure 2.9 is used , which features a powerful quad-core ARM Cortex-A72 processor running at high clock speeds. This processor configuration offers notable advantages, including High-speed Data Transmission and high-performance capabilities.

Figure 2.9: Raspberry Pi 4G [20]

Raspberry Pi Camera: The Raspberry Pi Camera Module V2 (Figure 2.10)is a camera accessory designed specifically for the Raspberry Pi single-board computer. It is a compact and lightweight camera module with high quality 8 megapixel Sony IMX219 image sensor custom . The Connection to the Raspberry Pi is through a small sockets on the top of the board. This interface uses a dedicated CSI interface specifically designed for the camera.

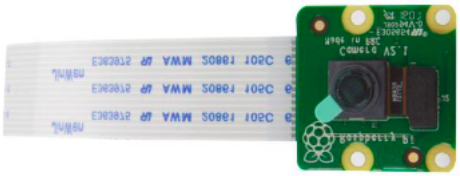


Figure 2.10: Raspberry Pi 4G [21]

2.1.2.1 Hierarchical Layering of Robot Components:

The foundational layer of the robot's architectural framework provides space for the placement of the Roboclaw motor controller and gearmotors, which establish the vital connection to the wheels. Above this base layer, the subsequent layer is specifically allocated for the optimal positioning of batteries, strategically designed to optimize weight distribution across the robot's structure. This deliberate configuration aims to enhance the robot's stability and maneuverability as a result. Moving up to the third layer, the Rplidar sensor and the Raspberry Pi are placed, synergistically contributing to the robot's sensory perception and processing capabilities(see Figure 2.11). Finally, the topmost layer is reserved for accommodating the camera module, facilitating visual perception and further expanding the robot's sensing capabilities.

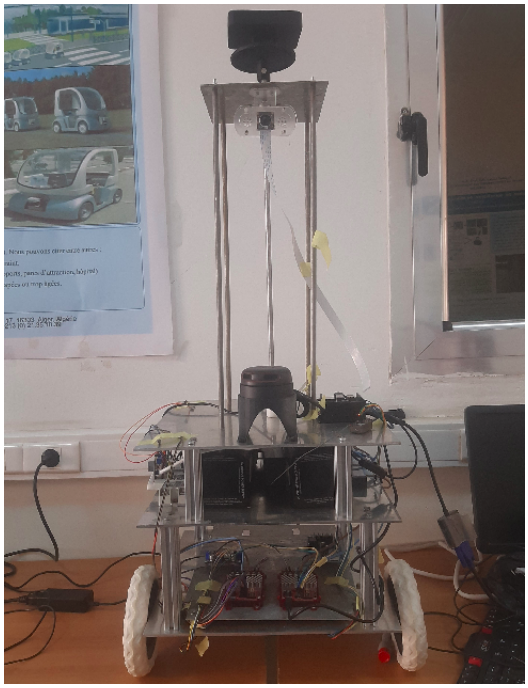


Figure 2.11: Robot Architecture: Component Placement

2.1.2.2 Data Transmission and Component Connection Process

The Raspberry Pi sends the prepared motor control commands to the RoboClaw via USB serial.Upon receiving the commands, the RoboClaw interprets the data and extracts the relevant information, such as motor speed and direction. Then the RoboClaw processes the commands and adjusts the motor's PWM signals accordingly to achieve the desired control parameters. After that the motor responds to the adjusted PWM signals and changes its speed and direction accordingly, hence,t he motor's movement, based on the control signals received, will lead to a change in the robot's position or orientation. The process of sending commands from the Raspberry Pi to the motors involves the implementation of a control loop on the Raspberry Pi, which continuously reads the desired position and orientation values from the user or application. The RoboClaw extracts the relevant encoder data, such as motor position, speed, or other feedback parameters, from the processed encoder signals.This data represents the current state of the motor as measured by the encoders. The loop calculates the necessary adjustments required to achieve the desired position and orientation. To ensure accuracy and precision, a feedback control algorithm, such as a PID controller, is utilized to compare the desired position and orientation with the actual readings from the encoders. The control signals generated by the feedback control algorithm are used by the Roboclaw to adjusts the motor's PWM signals to achieve the desired position and orientation. This iterative process is repeated continuously until the desired position and orientation are reached.
The Raspberry Pi Camera Module uses the Camera Serial Interface (CSI) connector to transfer image data to the Raspberry Pi. This dedicated interface enables high-speed data

transfer between the camera module and the Raspberry Pi, where the camera module uses the image sensor to capture images or video, and the CSI interface facilitates the transfer of raw image data. The CSI protocol defines data formats and synchronization mechanisms for transmission. On the Raspberry Pi side, the CSI receiver converts the received differential signals into digital data. The Raspberry Pi can process, analyze, and store image data according to specific requirements, allowing for flexible utilization of the captured data.

2.2 Software Setup

2.2.1 Simultaneous Localization and Mapping (SLAM)

SLAM is the process by which a mobile robot can build a map of its environment and at the same time estimates its own position within that map.[22] It works by integrating sensor measurements, such as odometry, visual data, or range measurements, with probabilistic estimation techniques. As the robot moves through the environment, it collects sensor data and uses it to update its belief about its own position and the surrounding environment. This iterative process allows the robot to incrementally build an accurate map and maintain a reliable estimate of its location, even in the presence of uncertainty and dynamic changes in the environment. SLAM enables robots to autonomously navigate and operate in unknown or partially known environments by combining perception and localization [23], thus facilitating tasks that require knowledge of both the robot's location and the surrounding environment. SLAM algorithms use probabilistic estimation techniques to estimate the robot's position based on sensor measurements and a map of the environment. Some common probabilistic estimation techniques used in SLAM include Markov Localization, Kalman Localization, Monte Carlo Localization(AMCL), Probabilistic Principal Component Analysis (PCA), and Conditional Probability Density Functions (PDFs). SLAM algorithms often use a combination of these techniques to estimate the robot's position and build an accurate map of the environment. The choice of technique depends on the specific requirements of the application, such as the type of sensors used, the level of accuracy required, and the available computational resources.

2.2.2 Path Planing

Path planning, a fundamental aspect of the navigation of robots, it relies on representing the environment and potential paths as a graph data structure.
This representation involves partitioning the environment into distinct locations, denoted as nodes N, and connections between these nodes represent possible transitions or movements. Each node in the graph typically corresponds to a specific location or

configuration in the environment, while the edges capture the permissible connections or paths between these locations. By representing path planning as a graph, the problem is transformed into a search task. Algorithms possessing the capability to systematically and effectively explore the graph G in order to determine the optimal path from an initial node A to a destination node B are referred to as admissible algorithms. [24] The graph representation allows for the integration of various factors such as obstacles, terrain characteristics, and other constraints, enabling the algorithm to make informed decisions on the most suitable and efficient paths to navigate through the environment. Various path planning algorithms have been developed, each with its own strengths and limitations. Among the most widely used algorithms are Dijkstra's algorithm, and A-star algorithm.

Dijkstra's algorithm,is a classic path planning algorithm,which explores the entire graph, considering all possible paths and their associated costs [25]. While this ensures finding the shortest path but can be computationally expensive, for that reason the **A* (A-star)** algorithm was developed improving upon Dijkstra's algorithm by incorporating a heuristic to guide the search process, the heuristic approach uses special knowledge about the domain of the problem being represented by a graph to improve the computational efficiency of solutions to particular graph-searching problems. During the exploration, the algorithm updates the f-scores and parent pointers of the nodes to keep track of the optimal path. It uses a heuristic function, such as the Euclidean distance or Manhattan distance, to estimate the remaining cost from each node to the goal [26].

The cost of A start is given by:

$$f(v) = h(v) + g(v) \tag{2.1}$$

where h is the heuristic and g is the cost that already incurred from initial state to 'n' state. A good heuristic function provides approximate cost that is close to actual cost.

In terms of algorithmic complexity, Dijkstra's algorithm exhibits a time complexity of $\Theta(V^2)$, but it can be optimized by using adjacency lists and efficient priority queues such as binary heaps, pairing heaps, or Fibonacci heaps, resulting in a time complexity of $O((E+V)\log V)$ where E - number of edges, V - number of vertices. While A-Star algorithm has a time complexity that depends on the chosen heuristic[27], but it is generally considered more efficient than Dijkstra's algorithm.[28] [27]

Overall, A-star algorithm balances efficiency and optimality; making it widely applicable in various real-world scenarios. For this, A-Star algorithm was chosen over Dijkstra's algorithm to plan the path for our navigation system implementation.

Table 2.2: Efficiency Comparison: A* Algorithm vs Dijkstra's Algorithm

Comparison	**A* Algorithm**	**Dijkstra's Algorithm**
Basic Idea	Informed search algorithm using heuristics	Uninformed search algorithm
Optimality	Optimal (when heuristic is admissible)	Optimal
Memory Usage	More memory-efficient	Less memory-efficient
Time Complexity	Depends on the heuristic	$O((E+V)\log V)$
Heuristic Function	Requires admissible heuristic Considers both cost and estimated remaining cost	No heuristic function required Considers only the cost

2.2.3 Navigation stack

The Navigation Stack is a set of software packages in ROS that provides a complete solution for robot navigation.
The Navigation Stack includes several built-in packages that provide essential functionality for autonomous robot navigation. These packages are:
map_server: This package provides a map server that reads a map from a file and serves it to other ROS nodes in the Navigation Stack.
amcl: This package provides Adaptive Monte Carlo Localization (AMCL), which is a probabilistic algorithm for estimating the robot's position in the map.
move_base: This package provides a high-level interface for commanding the robot to move to a target location while avoiding obstacles. It uses the map, AMCL, and a path planner to generate a collision-free path to the target.
costmap_2d: This package provides a 2D costmap that represents the obstacles and free space in the robot's environment. It is used by the path planner to generate a collision-free path for the robot.
global_planner and **local_planner**: These packages provide path planning algorithms that generate a global and local path, respectively, for the robot to follow. The global planner generates a path from the robot's current location to the goal location, while the local planner generates a path that is optimized for the robot's current position and velocity.
base_local_planner: This package provides a low-level interface for generating velocity commands that move the robot along a path while avoiding obstacles.[29]

The diagram in Figure 2.12 illustrates the overview of navigation stack configuration. The white components are required components that are already implemented, the gray components are optional components, and the blue components must be created for each robot platform.[30]

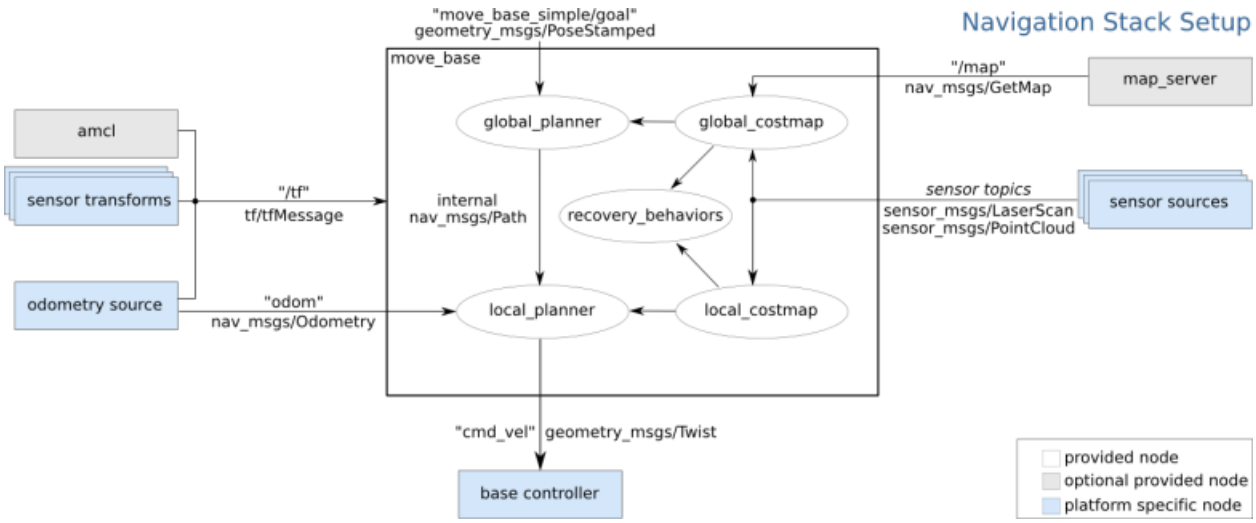


Figure 2.12: Navigation Stack Configuration[31]

The Navigation Stack relies on several input sources, including:

Map: In order to navigate in an environment, the navigation stack needs a map of the environment. This map is usually published in the **map** section. However, if no map is available, the navigation stack may still work, but it may not be able to avoid obstacles or plan optimal paths.

Odometry: The robot should publish its odometry data, which describes its position and orientation over time. This information is usually published in **odom**.

Sensor data: The robot should publish the sensor data that is used by the navigation stack to build a map of its environment. This data is usually published to a topic specific to the type of sensor used, such as a lidar or laser data topic.

Transformations: The robot must publish transformations that define the relationship between its various links on the **tf** topic. This information is usually generated by a tool such as robot_state_publisher.

The following sequence denotes the key stages of information transformation:

map -> odom -> base_footprint -> base_link -> laser

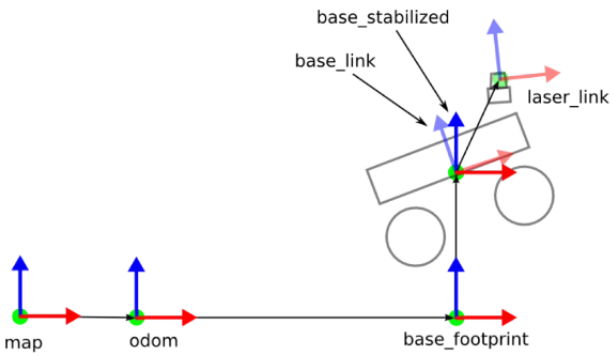


Figure 2.13: Frames of Navigation Stack

In the context of our robot configuration, it is noteworthy that the terms "base_footprint" and "base_link" are identical or refer to the robot chassis. So, In our implementation, the term "base_link" will be exclusively utilized to refer to the fundamental component within the robot configuration.
In the transform sequence(see Figure 2.13), the initial mapping data ("map") is subsequently processed through odometry ("odom"), leading to the movement reference frame ("base_link"). Finally, this chain culminates in the laser sensor ("laser"), which enables the robot to accurately perceive its environment.

The outputs of the navigation stack include: **cmd_vel** message, which is a geometry_msgs Twist message containing speed commands for the robot. These commands are typically sent to the robot's actuators, such as motors, and are received on the **cmd_vel** topic. By receiving these velocity commands, the robot can move through its environment and navigate to its goal.
Figure 2.14 illustrates the navigation process:

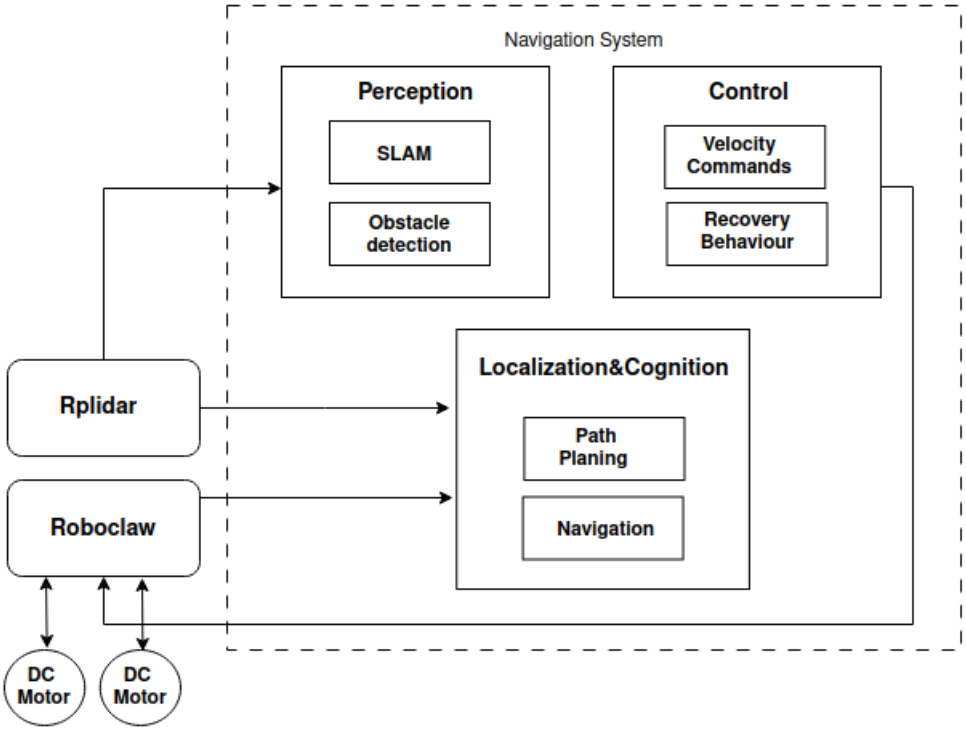


Figure 2.14: navigation System Architecture

The navigation stack can be customized and configured for better design and implementation of robotic systems that can effectively navigate their environment with greater efficiency and effectiveness.

2.3 Conclusion

In conclusion, this chapter has provided a comprehensive overview of the methodology employed in setting up the robotic system. By thoroughly addressing the essential aspects of the robot setup, encompassing mechanical design, hardware configuration, and software components, this chapter establishes a solid foundation for the subsequent chapter, which will focus on setting up the navigation system.

CHAPTER 3

NAVIGATION SYSTEM

The objective of this chapter, is to examine the development of a navigation system for a robot. The first step towards creating a functional navigation system is the setup of the robot. This involves the careful configuration of both hardware and software components to ensure that they work together seamlessly, enabling autonomous navigation. The setup process includes connecting sensors, actuators, and other hardware components to the robot and configuring essential software packages, such as SLAM and the navigation stack. By completing this process, the navigation system can be customized to meet the specific requirements of the robot's intended application. A well-executed robot setup is critical in ensuring consistent and accurate navigation, enabling the robot to navigate its environment safely and efficiently.

3.1 Network Configuration and Communication Setup

The network configuration between the user's PC and the Raspberry Pi involves establishing a connection for remote access and control. Both devices need to be connected to the same network, either through a wired Ethernet connection or a wireless connection. The Raspberry Pi should be assigned a valid IP address within the network . This IP address acts as the unique identifier for the Raspberry Pi, enabling the PC to locate and communicate with it. Once the network connection is established, the user can utilize network protocols such as Secure Shell (SSH) which is a cryptographic network protocol that provides a secure and encrypted method for remote login, command execution, and data communication, it can be utilized to establish connection between the user's PC and the Raspberry Pi, allowing remote access and execution of commands. OR for graphical interface-based remote control and improved monitoring capabilities, the Virtual Network Computing (VNC) protocol is employed. By installing VNC Server on the Raspberry Pi and VNC Viewer on the user's PC, a connection can be established to remotely access and

control the Raspberry Pi's graphical user interface. This network configuration, which combines SSH and VNC, facilitates secure remote management, control, and enhanced monitoring of the Raspberry Pi from the user's PC.
The diagram shown in Figure 3.1 illustrates the connection between a PC and Raspberry Pi and its physical connection to the hardware used:

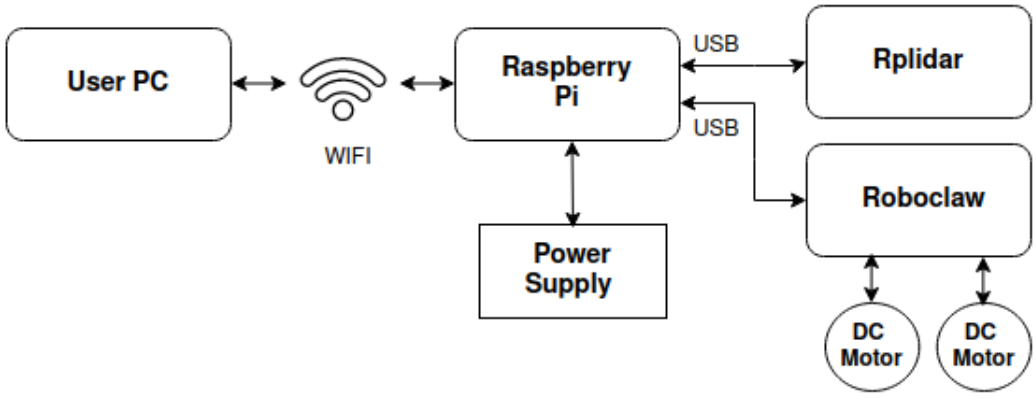


Figure 3.1: PC , Raspberry Pi and Hardware interfacing

3.2 Raspberry Pi Setup for ROS Application

To set up the Raspberry Pi for software development and efficient data processing, the Raspbian operating system is typically installed. Raspbian, being the official operating system for Raspberry Pi, is a preferred choice due to its compatibility and extensive community support. It is based on the Debian distribution, providing a stable foundation for managing files, executing programs, and processing data. By installing the Raspbian operating system, the Raspberry Pi becomes equipped with a reliable platform that facilitates software setup and enables efficient data processing capabilities. Once the setup is complete, the Raspberry Pi is connected to the internet, either through an Ethernet cable or Wi-Fi. At this stage, ROS can be installed on the Raspberry Pi by following the official ROS installation instructions for the specific desired ROS version. This typically involves adding the ROS repository to the package sources, installing the necessary dependencies, and then installing the ROS packages. Once ROS is successfully installed, the Raspberry Pi is ready to be utilized for developing and running ROS applications, enabling the integration of robotic systems and the execution of various ROS-based functionalities.

3.3 Navigation Setup

3.3.1 ROS environment

In ROS, a workspace is a directory where you can build and manage your ROS packages. A workspace can contain one or more packages, each of which consists of one or more nodes, libraries, or other components that perform specific tasks within a robot system.

A workspace provides an isolated environment for your ROS packages, allowing you to manage dependencies and build your packages independently of other packages or system libraries. Within a workspace, you can use ROS tools and commands to build, run, and test your packages, as well as manage their dependencies and build configurations. The workspace in ROS is named catkin by convention, this folder contains the following 3 folders (as shown in 3.2):

Devel space: space: contains setup files for the project ROS environment.

Build space: contains the compiled binary files.

Src space: contains source code, and is the main work folder which contains all the packages, launch files, scripts,...

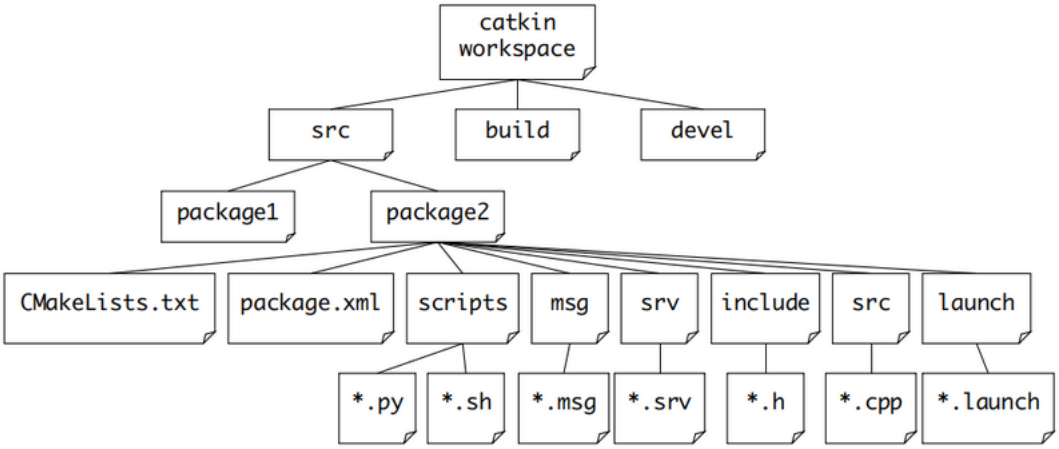


Figure 3.2: Catkin work space file

In order to execute a node successfully, it is essential to include the corresponding node package within the source file of the catkin_ws. By doing so, the necessary dependencies, libraries, and configurations associated with the node can be properly resolved and accessed during run time. This ensures that the node is integrated seamlessly into the ROS (Robot Operating System) environment, enabling its execution and interaction with other nodes within the system.

3.3.2 Map building

To enable the robot to navigate its environment, a static map must be created, this is done by using SLAM (Simultaneous Localization and Mapping) which allows the robot to build a map of an unknown environment while simultaneously estimating its own position within that environment

Although several SLAM algorithms are available, the Hector SLAM was selected for this project since it delivered good results and an accurate map.
HectorSLAM is a mapping method that creates maps of the surroundings only from laser scan data unlike Gmapping which requires the odometry information. It estimates the robot's movement and establishes its position in the environment by comparing the most recent laser scan with the most recent scan using scan matching techniques. The robot's linear or angular displacement in HectorSLAM triggers the map update process, keeping the map in syncronization with the robot's movement.
HectorSLAM provides additional adjustable parameters includeing: map size, the number of particles utilized in the scan matching process, and the names of the ROS topics employed for data communication, which enhance its configurability and adaptability to different applications. With this flexibility, users may precisely tailor the mapping process to meet their unique needs, guaranteeing that the desired map quality and performance standards are reached.

In order to execute HectorSLAM, the HectorSLAM file was downloaded and placed in the workspace directory. This file contains the necessary code and dependencies for running the algorithm. Alongside this, the execution of HectorSLAM also requires the presence of an RPLIDAR sensor. Figure 3.3 represents the block diagram of Hector Slam.

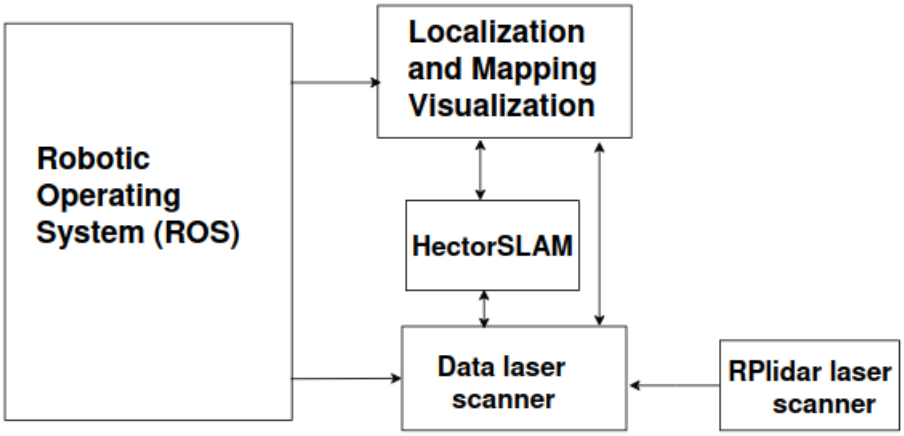


Figure 3.3: Block Diagram of Hector SLAM System[32]

Construction of precise and trustworthy maps for the navigation system is made possible by integrating the RPLIDAR sensor with the HectorSLAM algorithm.

3.4 Navigation implementation:

The first step to execute the navigation stack is to install the navigation stack packages within the ROS framework, it can either be cloned from official repositories or conducting a direct installation from the terminal in an Ubuntu OS environment. The navigation package comes with the necessarily tools to make the robot move, localize itself and save and display the map, among other things. install Navigation Stack Packages into **src** directory either by cloning the necessary Navigation Stack packages from the official repositories ,or install it directly from the terminal if Ubuntu OS is installed into the raspberry.
The creation of a Navigation Stack package involves a systematic process to ensure compatibility and customization according to unique research requirements. This process begins with the specification of navigation-related dependencies, ensuring that the package integrates seamlessly with the Navigation Stack framework. Moreover, meticulous adjustments to the **package's XML** and **CMakeLists.txt** files are necessary to precisely tailor them to the specific research objectives. These modifications allow for the inclusion of essential components and functionalities within the package.
The navigation file is comprised of three essential components, namely the **launch** file that executes all launch files, the static map files stored in the **map** file, and the parameter files found in the **param** section. These parameter files determine the behavior and performance of the robot's navigation system and contain various settings, such as obstacle avoidance, path planning, and goal selection, that dictate how the robot responds to different situations in its environment.

The navigation launch file (as described in **algorithm 1**) is structured as follows:

1. Display the previously build map using map server node
2. RViz configuration file
3. The rplidar and the roboclaw nodes
4. Transform between the map frame and odom frame, as well as the transform from base_link to laser
5. The move_base node

Algorithm 1 Navigation Launch File Algorithm

Start
Set the value of the argument **model** to "$(find package_name)robot_model_file.urdf.xacro" (default value).
Set the value of the argument *gui* to **true** (default value).
Execute the command "$(find xacro)/xacro $(arg model)" and store the output in the parameter **robot_description**.
Set the value of the argument **map_file** to "$(find package_name)/maps/map_file.yaml" (default value).
Start the map server node with the following specifications:
    • Node name: **map_server**
    • Package: **map_server**
    • Node type: **map_server**
    • Arguments: Use the value of the argument **map_file** as an argument for the node.
    • Set the value of the parameter **frame_id** to **map** for the map server node.
Include and launch the **rplidar.launch** file from the package **rplidar_node**.
Include and launch the **roboclaw.launch** file from the package **roboclaw_node**.
Include and launch the **move_base.launch** file.
Include and launch the **amcl.launch** file.
Start the rviz node with the following specifications:
    • Node name: **rviz**
    • Package: **rviz**
    • Node type: **rviz**
    • Arguments: Use the "-d $(find package_name)/rviz_configuration_file.rviz"
End

The move_base node comes with pre-defined parameters. However, in order to address the specific robot requirements, it is crucial to make adjustments to these parameters and load the corresponding parameter files (as specified in **algorithm 2**). This process is essential for customizing and finely tuning the move_base functionality to achieve optimal performance.

The following algorithm snippet illustrates the launch file configuration for the move_base package. This launch file sets up various parameters, including the choice of the local planner and the configuration files for the global and local costmaps. Additionally, it specifies the use of the dwa_local_planner, a popular choice for dynamic window-based local planning. The launch file also defines the remapping of certain topics.

Algorithm 2 move_base Launch File Algorithm

Start
Set the value of the argument **cmd_vel_topic**
Set the value of the argument **odom_topic**
Set the value of the argument **move_forward_only** to **false**.
Start the **base_laser_broadcaster** node with the following specifications:
- Package: **tf**
- Node type: **static_transform_publisher**
- Arguments: **"x y z roll pitch yaw *parent frame child frame* time stamp"**
  tf: map odom
  tf: base_link laser

Start the **move_base** node with the following specifications:
- Package: **move_base**
- Node type: **move_base**
- Respawn: **false**
- Arguments: Output to the screen
- Set the parameter
  base_local_planner to dwa_local_planner/**"DWAPlannerROS"**
- Load the parameters from the file:
  "costmap_common_params" for the namespace "global_costmap"
  "costmap_common_params" for the namespace "local_costmap"
  Load: rosparam "local_costmap_params"
  Load: rosparam "global_costmap_params"
  Load: rosparam "move_base_params"

- Remap the topic "cmd_vel" to "(arg cmd_vel_topic)"
- Remap the topic "odom" to "(arg odom_topic)"

End

The tuned parameters for the Move Base component are presented in the Tables 3.1, 3.2, 3.3, 3.4 and 3.5. A comprehensive overview is provided by these tables, showcasing the various configurations and settings that have been carefully determined to ensure optimal performance of the navigation system.
Each parameter plays a crucial role in defining the behavior and functionality of the Move Base, including aspects such as obstacle avoidance, path planning, and motion control.

Table 3.1: Common_costmap Parameters

Parameter	**Value**
footprint	[[0.225, 0.15], [0.225, -0.15] [-0.225, -0.15], [-0.225, 0.15]]
robot_radius	0.2
observation_sources	scan
scan	{ data_type: LaserScan, topic: /scan, marking: true, clearing: true }
cost_scaling_factor	2.0
inflation_radius	0.05
obstacle_range	2.5
raytrace_range	3.0
observation_persistence	1.0

Table 3.2: Local_costmap parameters

Parameter	**Value**
global_frame	odom
robot_base_frame	base_link
update_frequency	10.0
publish_frequency	10.0
rolling_window	true
static_map	false
width	6.0
height	6.0
resolution	0.05

Table 3.3: Global_costmap parameters

Parameter	**Value**
global_frame	map
robot_base_frame	base_link
update_frequency	5.0
static_map	true

Table 3.4: DWAPlannerROS Parameters

Parameter	**Value**
max_vel_x	0.05
min_vel_x	-0.05
max_vel_y	0.0
min_vel_y	0.0
max_vel_trans	0.26
min_vel_trans	0.13
max_vel_theta	1.82
min_vel_theta	0.9
acc_lim_x	1.0
acc_lim_y	0.0
acc_lim_theta	3.2
xy_goal_tolerance	0.002
yaw_goal_tolerance	0.002
latch_xy_goal_tolerance	false
sim_time	2.0
vx_samples	20
vy_samples	0
vth_samples	40
controller_frequency	25.0
path_distance_bias	32.0
goal_distance_bias	20.0
occdist_scale	0.02
forward_point_distance	0.325
stop_time_buffer	0.2
scaling_speed	0.25
max_scaling_factor	0.2
oscillation_reset_dist	0.1
publish_traj_pc	true
publish_cost_grid_pc	true

Table 3.5: Move_base Parameters

Parameter	**Value**
shutdown_costmaps	false
controller_frequency	10.0
planner_patience	5.0
controller_patience	15.0
conservative_reset_dist	3.0
planner_frequency	10.0
oscillation_timeout	10.0
oscillation_distance	0.2

To enable real-time visualization of the robot's navigation and enhance the execution of the navigation stack, the RViz configuration was customized to provide an improved

representation of the robot's navigation process.
Table 3.6T represents the key components incorporated into the RViz configuration.

Table 3.6: Rviz Configuration

Component	**Topic**	**Visualization Type**
LaserScan	/scan	LaserScan
Map	/map	OccupancyGrid
Planner Plan	/move_base/NavfnROS/plan	Path
Global Map	/move_base/global_costmap/costmap /move_base/DWAPlannerROS/global_plan	OccupancyGrid
Local Map	/move_base/local_costmap/footprint	Footprint
Polygon	/polygon	Polygon
Pose	/pose	Pose

The subsequent illustration presented in Figure 3.4 showcases the interconnection of all utilized nodes via topics using the rqt_graph tool. It provides a visual representation that allows for a comprehensive understanding of the communication flow within the system.

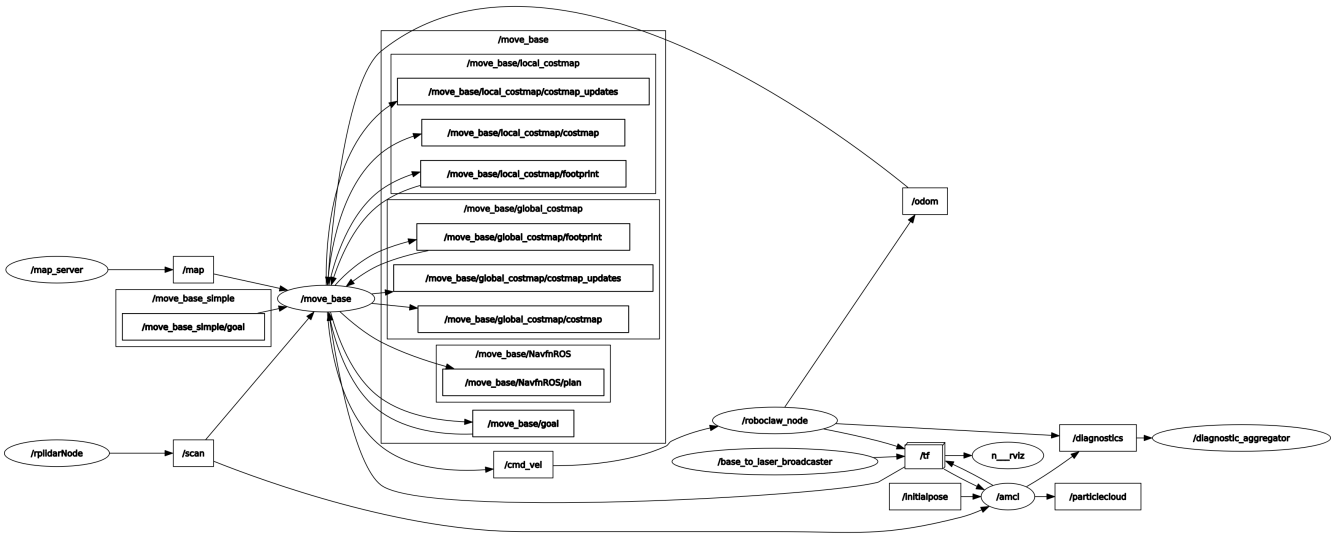


Figure 3.4: Connection of nodes graph

This graphical representation illustrates the relationships and interactions among the nodes, offering valuable insights into the coordination and exchange of information between different components.

The ROS frame tree for the navigation system is illustrated in Figure 3.5, which is a hierarchical structure that defines the spatial relationships between different frames in the robot environment. It serves as a fundamental component to ensure accurate and consistent data representation and transformation across nodes within a ROS system.

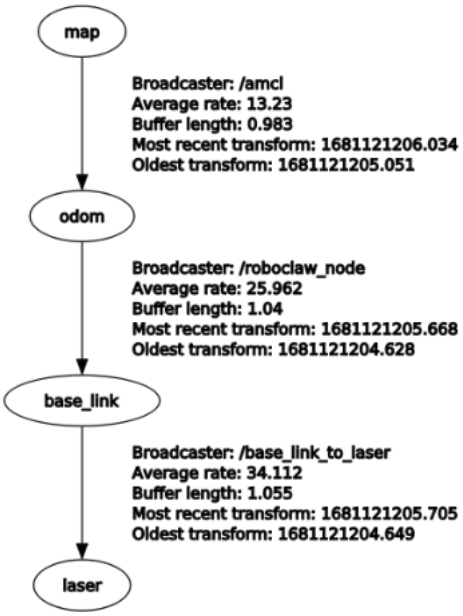


Figure 3.5: Frames graph

By examining the ROS frame tree along with the rqt_graph visualization, a comprehensive understanding can be gained regarding how the nodes and frames are collectively employed to enable effective navigation. The ROS frame tree provides a framework for defining the spatial relationships between different frames, thereby facilitating seamless data exchange and transformation. A thorough comprehension of this hierarchical structure is essential in order to ascertain the flow of information within the navigation system and the manner in which the robot interacts with its environment.

3.5 Conclusion

This chapter examined the software implementation of navigation, highlighting the set up of the work environment including setting up the remote control , creating the catkin work space,and the mapping process. It also provided the algorithms and the move base configuration used in the navigation system. The implementation of this work along with a comprehensive analysis and discussion will be presented in the next chapter.

CHAPTER 4

RESULTS AND DISCUSSION

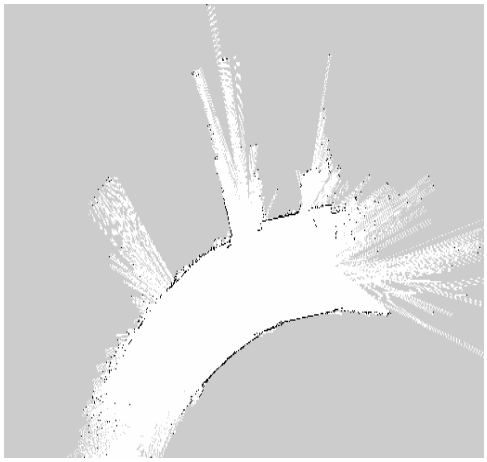
This chapter presents the results of the implementation of the ROS navigation stack into a mobile robot platform in a real-world setting. The navigation system is evaluated and tested for four key aspects: localization, obstacle avoidance, path following, and goal reaching. A comprehensive analysis of the results is provided, as well as a discussion and examination of the implications of the findings.

4.1 Mapping

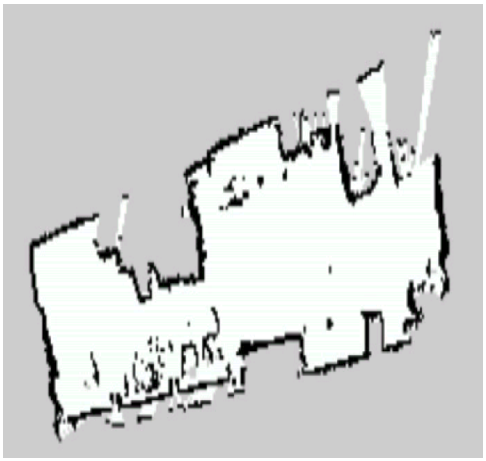
Executing the HectorSLAM package and connecting the RPLIDAR sensor, allow the start of the mapping process .Using scan matching algorithms, the RPLIDAR continually processes laser scan data to build and update the map. The generated map can be visualized and monitored using RViz, the displaying on Rviz shows the constructed map along with the trajectory of a moving robot.

To create a map of the environment, the robot is driven around using a joystick or keyboard. This allows the robot to explore the surrounding area and create a map. Once the mapping is complete, the resulting map is saved using the map server package. The saved map is in the form of an occupancy map, which represents the occupied and unoccupied areas of the environment represented as:
-An image showing the blueprint of the environment.
-A configuration file (yaml) that gives meta information about the map (origin, size of a pixel in real world).

Figure 4.1 represents the constructed maps of the space in which the robot will navigate. The map provides a visual representation of the environment, including walls, obstacles, and other features.



Outdoor Environment Map



Indoor Environment Map

Figure 4.1: Maps Constructed using Hector SLAM

During the map construction process, it was observed that certain objects were not detected due to the low placement of the LIDAR sensor on the robot. Consequently, certain areas were mistakenly considered unoccupied or empty, despite the presence of objects at elevated positions. This limitation highlights the importance of careful consideration of sensor location and its impact on the accuracy and comprehensiveness of the generated map.
Moreover, the suitable position to place the LIDAR is near the robot's center of mass which minimizes blind spots closer to the ground and higher up providing a balanced perspective covering a wider area of the environment. This reduces the possibility of missing an obstacle or misjudging available space.

4.2 Evaluation of Implemented Navigation System

Subsequently, we will proceed with launching the navigation stack. With the move_base and RViz parameters configuration specified in subsection 3.4 and setting the necessary nodes. Launching the navigation file opens Rviz with the map , costmaps, and displays the robot model . This comprehensive representation in Rviz shows that all the necessary components have been set up correctly and the robot is ready to start traversing the environment autonomously, and reach the set goals.

The navigation process begins by setting an approximate initial position for the robot needs to be set using the "2D Pose Estimate" in Rviz to facilitate self-localization within the map and match its sensor data with the static map. The navigation process starts with setting the goal position using the "2D Nav Goal" feature with specified position and orientation; these inputs are published to the /move_base_msgs/MoveBaseActionGoal

topic. The move_base package receives the approximate location of the robot through particle filtering from the amcl package and the goal data . After that, it plans a path to the goal using the global_planner, which generates a global plan and the local_planner which generates a local plan to follow the global plan, taking into account the robot's current pose as estimated by amcl. Then, The move_base node sends motor commands to the robot's motors using the /cmd_vel topic to follow the planned path, while the costmap and obstacle_detector nodes detect and avoid obstacles in the robot's environment.

4.2.1 Motion Control:

The desired velocities are published as velocity commands on "/cmd_vel"ROS topic. These velocity commands are represented as linear and angular velocities, in the form of Twist messages. Roboclaw generates PWM signals to encode information of the desired linear and angular velocities received from the move base in the form of a varying pulse width. These signals are sent to the appropriate motor channels in Roboclaw. Which then interprets its PWM signal and activates the appropriate H-bridge circuit incorporated in the roboclaw, to control the rotation of the motor.
The H-bridge selectively turns its four switches on or off, applies a positive or negative voltage to the motor terminals, resulting in forward or backward motion.
In case of the desired velocity is positive, indicating forward motion, Roboclaw responds by generating a high duty cycle PWM signal. This high duty cycle signal activates the upper switches in the H-bridge circuit, allowing current to flow in a particular direction through the motor, resulting in forward rotation. Conversely, when the desired velocity is negative, indicating reverse motion, the Robolaw generates a PWM signal with a low duty cycle. This low duty cycle signal activates the bottom switch of the H-bridge, causing current to flow in the reverse direction through the motor. To rotate the robot, Roboclaw modulates the left and right motor PWM signals separately. By lowering the duty cycle of one motor and keeping the duty cycle of the other motor high, Roboclaw will actuate the appropriate switches in the H-bridge circuit accordingly. This creates a speed difference between the left and right wheels, causing the robot to turn.
By adjusting the duty cycle of the PWM signal, Roboclaw controls the direction and speed of the motors, allowing the robot’s motion to be precisely controlled.
On the other hand, feedback control is needed to provide information about the actual movement of the robot to fine-tune its performance. Encoders serve as the source of this feedback by measuring the real-time velocity of the robot, and according to that robot movements can be precisely controlled and regulated using PID control algorithms. A PID controller that produces a control output. This control output is typically applied as a motor command to the wheels of the robot, adjusting the motor command to the desired speed. This iterative control loop is executed continuously with feedback from the encoder and recalculation of the control output at each iteration.

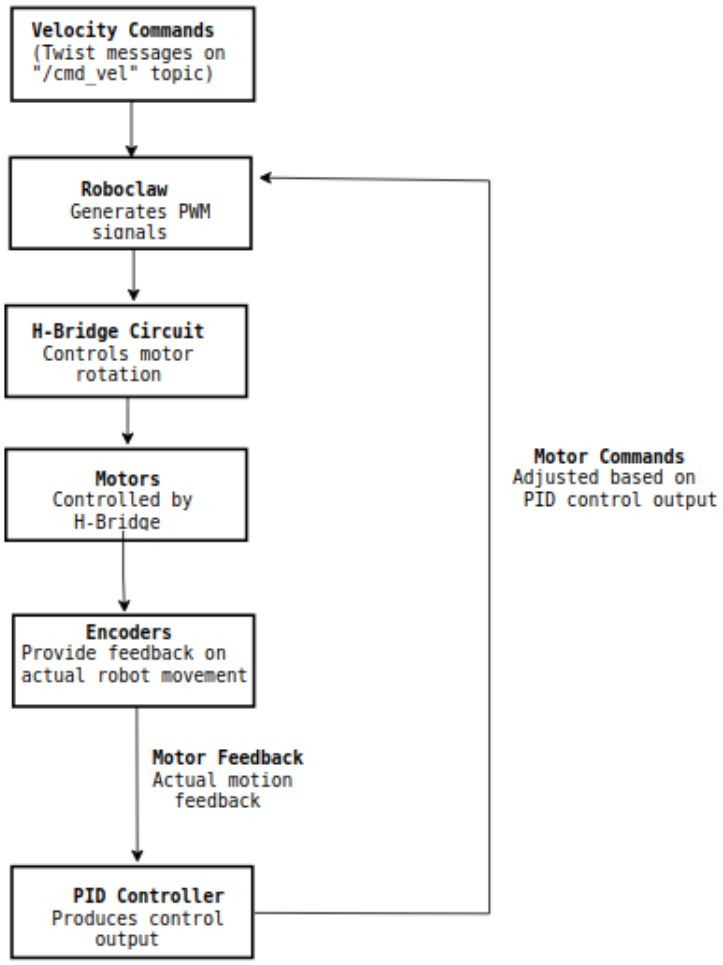
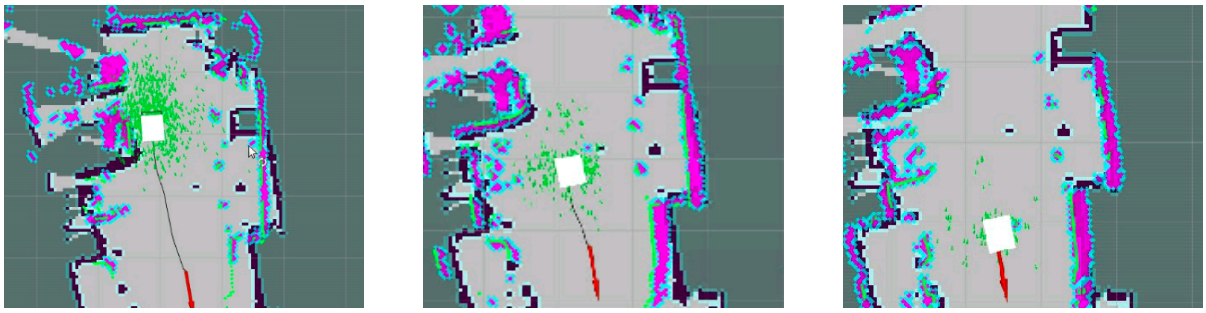


Figure 4.2: System Architecture for Robot Motion Control with Feedback Loop

4.2.2 Localization Performance

As discussed before the ROS Navigation Stack requires the use of AMCL (Adaptive Monte Carlo Localization). AMCL is used to track the pose of a robot against a known map. It takes as input a map, LIDAR scans; transform messages, and outputs an estimated pose. In executing the ROS navigation stack, RViz displays the particle cloud used by the robot's localization system. The particle cloud represents the probability distribution of the robot's position; the spread of the cloud represents the localization system's degree of certainty about the robot's position: a cloud that is very spread out reflects high uncertainty, while a condensed cloud represents low uncertainty. As the robot navigates through the environment, the particle cloud undergoes dynamic changes; initially, when the robot starts its motion, the particle cloud is typically spread out across the entire map. Figure 4.3 (a), representing a wide range of potential robot poses. As the robot moves and receives sensor measurements, the particles are updated based on the observations, leading to a refinement of their spatial distribution Figure 4.4 (b). By the end of the robot's

journey, the particle cloud becomes significantly concentrated around the true robot pose, indicating a high level of confidence in the localization accuracy as shown in Figure 4.4 (c).



(a) Initial Particle Distribution

(b) Refinement of Particle Distribution

(c) Converged Particle Distribution

Figure 4.3: Evolution of AMCL Particle Distribution during Robot Path Movement

The accuracy of localization was evaluated by comparing the actual position of the robot in the environment with the estimated position displayed in Rviz. Then the RMSE is calculated and the results are represented in Figure 4.4:

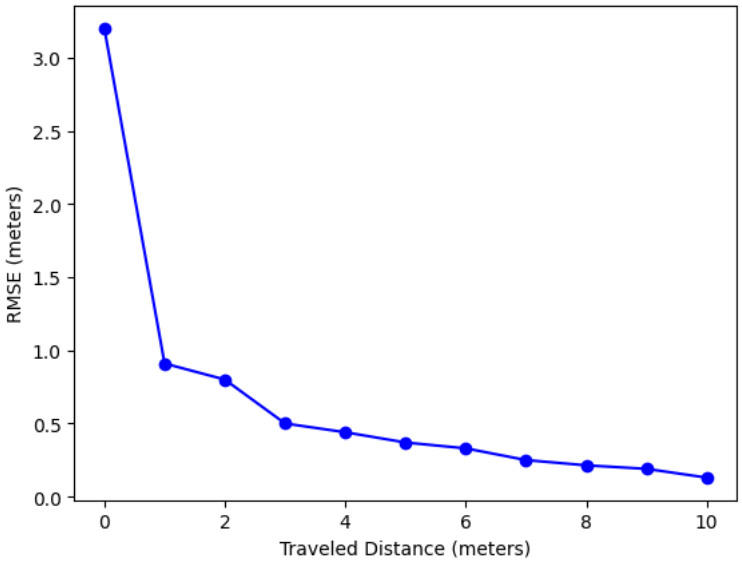


Figure 4.4: Robot position RMSE error

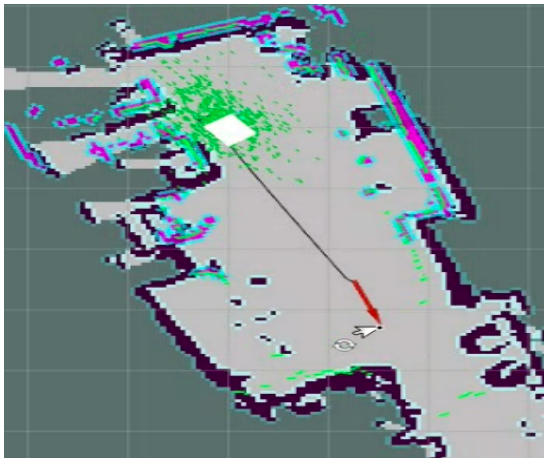
The Root Mean Square Error (RMSE) was utilized as a metric to quantify the robot position error. As the robot traversed from its initial position to the goal, the RMSE value was computed at regular intervals of the positional deviations at different stages of the navigation process. These results are represented in Figure 4.3 to illustrate the relationship between the traveled distance and the corresponding RMSE values. This representation enables a visual understanding of how the localization error changes as

the robot covers different distances along its path.

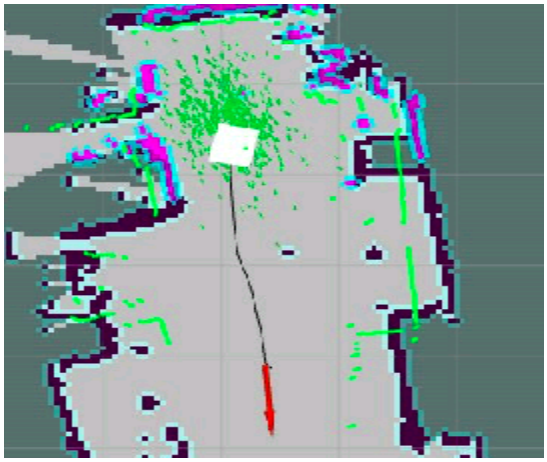
The observation of the results demonstrates a pattern: the localization error exhibits an apparent decrease as the robot progresses over greater distances during its movement. This observation strongly suggests that the accuracy of the Adaptive Monte Carlo Localization (AMCL) algorithm, implemented in the navigation stack, significantly enhances the localization performance as the robot explores the environment. This can be explained by the fact that as the robot moves further away from its initial position, the accumulation of sensor data and the integration of multiple observations over time contribute to a more accurate estimation of its actual position. The AMCL algorithm utilizes a particle filter-based approach that dynamically adjusts the belief of the robot's pose based on sensor measurements and motion updates. As the robot explores the environment and collects more information, the algorithm can refine its estimate, leading to a reduction in localization error.

4.2.3 Path Execution

In Section 2.2.2; the path planing was discussed. In this section, we aim to evaluate the efficiency of two path planning algorithms: A* and Dijkstra's algorithms; within the implemented navigation stack. The objective is to provide evidence that supports the earlier statement regarding the choice of path planning algorithm. Moreover, the execution of the path planned in the real environment is evaluated. By analyzing these performance measurements, it is possible to assess the system's ability to find optimal paths and successfully execute it.
In order to evaluate the performance of the path planing algorithms within the navigation stack, a comparative analysis is conducted. The navigation stack is executed twice, with each run starting from the same initial point and aiming for the same goal. In one execution, the A* algorithm is employed, while in the other, the Dijkstra's algorithm is utilized. This allows for a direct comparison between the two algorithms in terms of their efficiency and effectiveness in generating the optimal path as illustrated in Figure 4.5.



Dijkstra Algorithm



A* Algorithm

Figure 4.5: Path Planning Comparison: A* Algorithm vs Dijkstra's Algorithm

During the conducted testing of implementing the navigation stack using A* and Dijkstra algorithms, it was observed that the A* algorithm exhibited comparatively faster execution times, ranging between 3 to 6 seconds, while Dijkstra's algorithm took approximately 5 to 9 seconds to complete. However, it is important to note that these observed execution times may include not only the computation time of the algorithms but also the potential time delay caused by data transmission between the master PC and the Raspberry Pi. The navigation stack heavily relies on efficient data exchange between these components, and any delays in data transmission can impact the overall execution time.

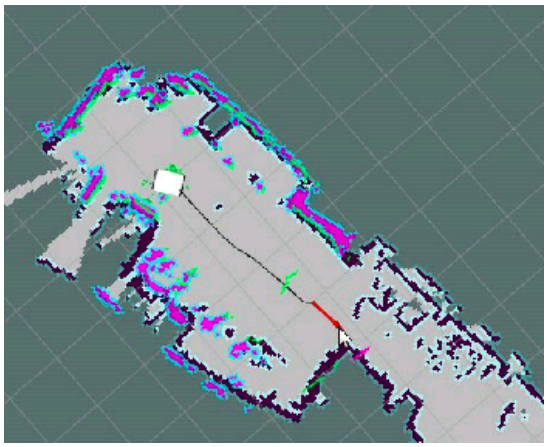
While Dijkstra's algorithm guarantees the discovery of the shortest path. The conducted experiments revealed it typically generates straighter paths in comparison to A*. On the other hand, A* has the ability to dynamically adjust its path based on the changing environment. By considering the heuristic and incorporating it into the cost estimation, A* algorithm can adapt to obstacles and make more informed decisions for path planning, resulting in better obstacle avoidance capabilities.
Overall, despite the slightly less short paths generated by Dijkstra's algorithm compared to A*, the improved time response and adaptive nature of A* make it a favorable choice for our implementation, especially in scenarios where real-time responsiveness and obstacle avoidance are crucial.
Subsequently, an analysis and evaluation are conducted on the execution of the path that has been planned using the A* algorithm.

The evaluation of the robot's path-following capabilities yielded a high level of adhering to the desired trajectory in linear movement; however, the robot occasionally exhibits fast and aggressive rotations which cause the robot to slightly deviate from its

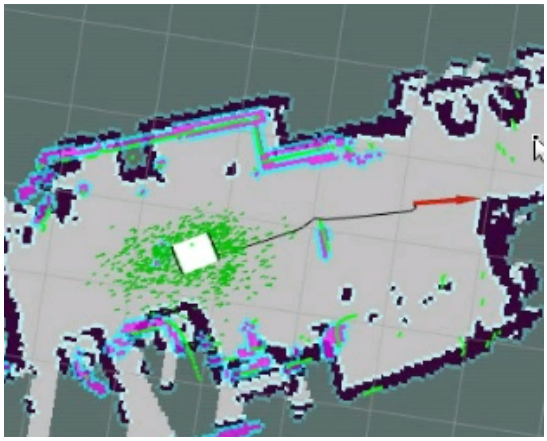
intended trajectory; notwithstanding the efforts to ensure smooth rotation transition. This deviation is primarily attributed to the instability of the robot's wheels used in the robot model, which was observed to slip on the ground. This inherent instability leads to slight variations in the robot's movements, resulting in deviations from the planned path. However, the navigation system continuously monitors these deviations and employs corrective measures to minimize their impact and realign the robot with the desired trajectory. Despite the challenges posed by wheel instability, the system endeavours to achieve dependable path following, enhancing the robot's overall navigation capabilities.

4.2.4 Obstacle Avoidance

In this section, the focus is on studying the system's capability to detect and avoid dynamic obstacles. The navigation stack is designed to automatically avoid static obstacles that are present in the preloaded map. When initially given a goal; The system plans a path that avoids these static obstacles. However, this behavior differs when it comes to dynamic obstacles. In this case, if an obstacle is relatively distant from the robot, beyond the specified obstacle range defined in the move base parameters, the initial path planning does not consider the obstacle. Only when the obstacle comes within close proximity to the robot that will be detected, prompting the robot to re-plan its path dynamically. This adaptive behavior enables the robot to avoid obstacles and reach its intended goal safely as shown in Figure 4.6.



Initial path planning without considering distant obstacle

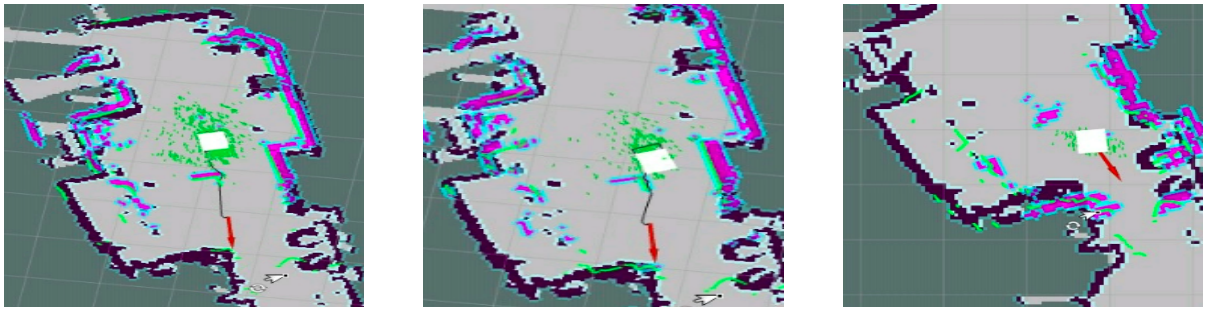


Dynamic path re-planning upon detecting close obstacles

Figure 4.6: Path planning and obstacle detection in the navigation system

Figure 4.7: Obstacle Encountered in the Robot's Pathway

The experiment was conducted in two different scenarios, an indoor environment and an outdoor environment, to assess the robot's adaptability and performance in varying conditions. In the indoor scenario, the robot encountered known obstacles that were strategically placed ahead of its path (Figure 4.7), simulating controlled conditions for obstacle detection and avoidance. On the other hand, the outdoor scenario introduced the challenge of a dynamic obstacle, represented by a human being standing in the robot's path, as depicted in Figure 4.11. This scenario aimed to assess the robot's ability to detect and respond to unexpected obstacles in a real-world setting. By conducting experiments in both scenarios, a comprehensive evaluation of the robot's obstacle avoidance performance was achieved, taking into account various challenges and potential limitations.

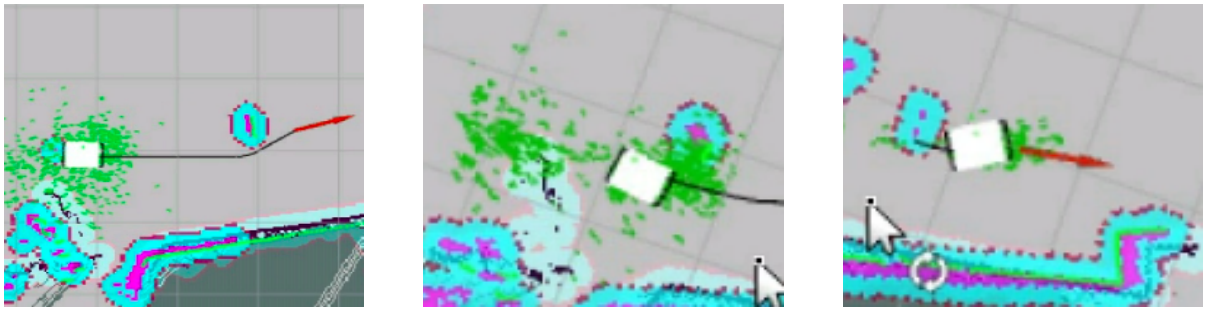


(a) (b) (c)

Figure 4.8: Simulated Obstacle Avoidance in RViz (Indoor Environment)

(a) (b) (c)

Figure 4.9: Real-Time Obstacle Avoidance in Dynamic Indoor Environment



(a) (b) (c)

Figure 4.10: Simulated Obstacle Avoidance in RViz (Outdoor Environment

(a) (b) (c)

Figure 4.11: Real-Time Obstacle Avoidance in Dynamic Outdoor Environment

In the controlled indoor environment, the robot demonstrated efficient obstacle avoidance capabilities by successfully detecting and navigating around predetermined obstacles without any instances of collision as shown in Figure 4.8 and 4.9. However, the outdoor scenario presented a more dynamic and demanding setting, featuring a human obstructing the robot's path. The results indicated that the robot effectively detected the presence of the human and promptly halted its motion to prevent potential collisions. However, due to the continuous movement of the human, which affected the path re-planning process, the robot exhibited slight deviations and rotations in its trajectory. Nonetheless, the robot demonstrated its adaptability by effectively adjusting its trajectory to circumvent obstacles and ultimately reaching its intended destination as demontrated in Figure 4.11 and 4.10.

By observing the behavior of the robot, it was able to detect the presence of an obstacle and promptly stop as it approached it. However, the robot encountered challenges in re-planning the path when the robot encounters a sudden obstacle positioned very close to its current location. This proximity restricts the robot's ability to re-plan its path smoothly in real-time. To compensate for this, the robot initiates a rotation in place to explore alternative paths and avoid immediate collisions.

when the obstacle is relatively close, the robot stops when approach it; the act of stopping the robot in place when encountering a dynamic obstacle is a deliberate strategy employed

to adapt to the changing dynamics of the environment. This pause allows the robot to update its local map, providing an opportunity to explore alternative paths and make informed decisions for safe path planning. By stopping and re-evaluating the situation, the robot ensures that it can navigate around obstacles effectively and choose the most secure path.
The successful avoidance of the obstacle indicates that the transformation between the laser sensor and the robot's base was accurately established. The precise alignment and calibration of the sensor's position relative to the robot's base ensures that the sensor readings correspond to the robot's actual position and orientation, enabling reliable obstacle detection. Another crucial aspect is the real-time updates of the sensors data ; this ensures that the LIDAR is continuously inquiring the data at a sufficiently high rate, allowing dynamic detection. Furthermore, the results shows that the ROS navigation stack employs reactive path planning techniques that dynamically adjust the robot's trajectory based on real-time sensor data. This allows the robot to respond promptly to obstacles and navigate around them effectively.

4.2.5 Evaluation of Goal Reaching Effectiveness

The primary objective of the navigation system is to allow the robot to reach its desired goal with utmost accuracy and precision. Achieving this objective necessitates the three key factors evaluated before. First, the robot must maintain continuous and reliable localization within the map, ensuring it has an accurate understanding of its own position in relation to the goal and surrounding environment. Second, the navigation system should enable effective collision avoidance, allowing the robot to dynamically respond to obstacles and navigate around them to reach the goal safely. Lastly, the robot should closely adhere to its planned path, minimizing deviations and executing the intended trajectory as closely as possible. By emphasizing these aspects, the navigation system aims to optimize the robot's goal-reaching capabilities and enhance its overall performance in its environment.
This section examines the navigation system's accuracy in achieving the designated goal. This is accomplished by comparing the intended goal assigned to the robot within the map to its actual stopping position indicating "Goal Reached" in the move base status topic.

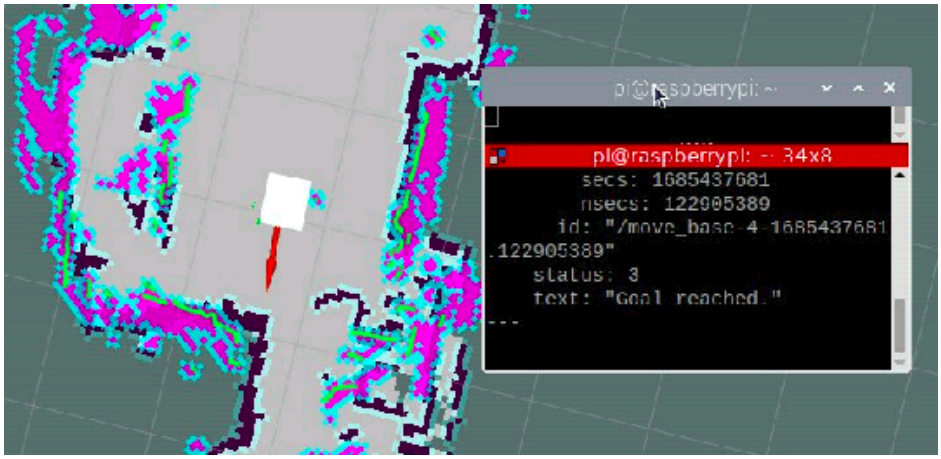
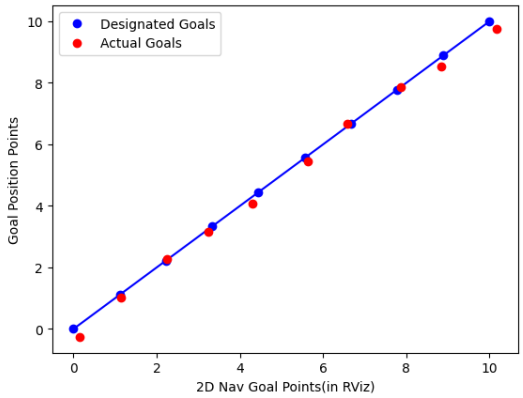
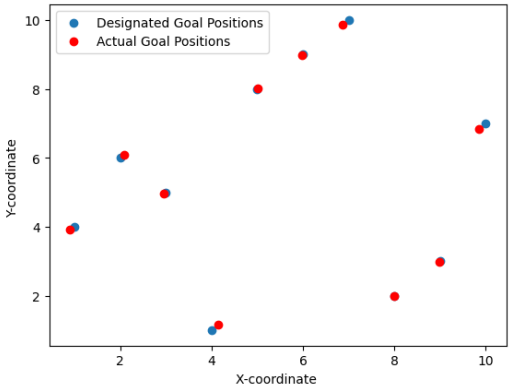


Figure 4.12: Reaching the Targeted Goal

To evaluate the preformance; a series of experiments were conducted, and the outcome of the experiments is visually presented in the accompanying figures.



Deviation of Actual Positions from Designated Goals



Designated Goals and Actual Positions on the Map

Figure 4.13: Evaluation of Navigation System Accuracy in Reaching Designated Goals

Overall, the robot demonstrated a high level of success in reaching its intended goal during the conducted tests; the total error resulting from summing the root mean square error (RMSE) for each measurement of the ten goal positions analyzed in indicates a value of RMSE= 0.1813.
It is important to note that the goal reaching results heavenly depend on the model of the robot defined in the URDF and the footprint dimensions set in the parameter files. The precise representation of the real robot's dimensions and characteristics within the simulation is essential to ensure the congruity between the simulated and actual robot movements.

In the presence of dynamic obstacles, such as humans or objects placed in front of the robot , the navigation system demonstrated its capability to detect and respond promptly. Nevertheless, the process of re-planning the robot's path necessitated rotational adjustments, which introduced an average time delay of 4 seconds. Despite this delay;

the robot presented safe and efficient collision avoidance. Throughout the experiments, the robot successfully followed the planned trajectory, although small oscillations were detected when the robot tried to adhered its trajectory consistently. Where the oscillation in our case where augmented due to the wheels instability of our robot mechanical design,however in case of a more stable structure; the slight oscillations in the path execution process is normal result of the robot's adaptive behavior to maintain a good overall planning performance.
Overall, the assessment of the navigation system's performance highlights several noteworthy strengths pertaining to its robustness, adaptability, and efficacy in various critical aspects. The system consistently exhibited the capability to adhere to the designated path and effectively implement both global and local planning strategies, thereby ensuring reliable collision avoidance. These findings offer valuable insights into the system's capabilities and lay a foundation for future research and development efforts aimed at further enhancing and optimizing its performance.

4.3 Conclusion

In conclusion, this chapter has presented the implementation of the ROS navigation stack in a real-world mobile robot platform. The navigation system was subjected to rigorous testing and in-depth analysis, with a focus on critical aspects such as localization, obstacle avoidance, path following, and goal reaching. The results of these tests were thoroughly examined and discussed, providing insights into the performance and effectiveness of the navigation system. Additionally, the implications of the findings were carefully considered, highlighting both the strengths and limitations of the implemented solution.

CONCLUSION AND FUTURE WORK

The aim of this project was to develop a robust and reliable navigation system for a mobile robot, It has reported the research carried out in order to accomplish this objective presenting the implementation of the navigation system on a mobile robot. The intricacies of mapping and navigation were thoroughly explored and deployed. The system underwent a rigorous and comprehensive evaluation covering a wide range of aspects, and the results of implementing the system real mobile robot were thoroughly discussed and carefully analyzed. The results demonstrated reliable localization, accurate path execution, efficient obstacle avoidance, and effective goal-reaching capabilities. These outcomes validate the feasibility and potential of using ROS as a framework for developing advanced navigation systems for unmanned ground vehicles.
Overall The mobile robot successfully navigated from its initial position to the goal position. Moreover, this research prioritized the implementation of the navigation stack with reduced complexity and minimal parameter adjustment, led to satisfactory outcomes through precise testing and tuning.

The findings of this work demonstrate the potential of ROS as a framework for developing efficient navigation systems for unmanned ground vehicles. These findings provide valuable insight into the capabilities of the system and lay a foundation for future research and development aimed at further improving and optimizing its performance.
While this research primarily focused on the application of the navigation system for surveillance purposes. The proposed system has important implications for integration into various industrial applications such as warehouses, exploration and transportation.

This work holds significant potential for future expansion and enhancement. One area with potential for development is the exploration of perception challenges. By addressing these challenges, the system can improve its ability to understand and respond to dynamic environments. A specific approach to consider is the utilization of the visual SLAM enabling the robot to navigate using the camera, resulting in more precise localization and mapping. To achieve this, the powerful Intel RealSense (RGBD) camera can be employed for perception, depth-sensing, mapping, and SLAM navigation.
Furthermore, the incorporation of the NVIDIA Jetson Nano is being considered to provide enhanced computational capabilities. This integration would enable the incorporation

of machine learning-based perception into the developed navigation system. This enhancement would empower the system to make informed decisions, navigate effectively in complex environments, improve safety, and deliver a more efficient and reliable navigation experience.

Additionally, we aim to integrate a local monitor to facilitate human-robot interaction and simplify the goal-selection process. By doing so, we eliminate the need for remote access from the master PC. The local board will directly display the simulation, enabling the navigation process to be entirely managed at the robot level. This addition seeks to enhance the efficiency of goal picking and improve the overall interaction between humans and the robot.

BIBLIOGRAPHY

[1] S. Serholt, S. Ljungblad, and N. Ní Bhroin, "Introduction: Special issue‚äîcritical robotics research", *AI & Soc*, vol. 37, pp. 417–423, 2022. DOI: 10.1007/s00146-021-01224-x.

[2] S. S. Ge and F. L. Lewis, *Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications*. CRC Press, 2006.

[3] J. A. T. Machado and M. F. Silva, "An overview of legged robots", in *MME 2006 International Symposium on Mathematical Methods in Engineering*, Polytechnic Institute of Porto, Cankaya, Ankara, Turkey, Apr. 2006.

[4] F. Ahmed, J. Mohanta, A. Keshari, *et al.*, "Recent advances in unmanned aerial vehicles: A review", *Arab J Sci Eng*, vol. 47, pp. 7963–7984, 2022. DOI: 10.1007/s13369-022-06738-0.

[5] Y. Petillot, G. Antonelli, G. Casalino, and F. Ferreira, "Underwater robots: From remotely operated vehicles to intervention autonomous underwater vehicles", *IEEE Robotics & Automation Magazine*, pp. 1–1, 2019. DOI: 10.1109/mra.2019.2908063.

[6] G. Oriolo, *Wheeled robots*, 2014. DOI: 10.1007/978-1-4471-5102-9_178-1.

[7] E. Ackerman. "Meet aquanaut, the underwater transformer", IEEE Spectrum. (Sep. 2021), [Online]. Available: https://spectrum.ieee.org/meet-aquanaut-the-underwater-transformer.

[8] FlyMotion. "Flymotion law enforcement drones". (2022), [Online]. Available: https://flymotionus.com/law-enforcement/.

[9] Boston Dynamics. "Boston dynamics", Boston Dynamics. (2023), [Online]. Available: https://www.bostondynamics.com/legacy.

[10] C. Robotics, *Turtlebot 4*, https://clearpathrobotics.com/turtlebot-4/, Accessed on April 28, 2023.

[11] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. Cambridge, Massachusetts London, England: The MIT Press, 2011.

[12] H.-H. Lee, "Modeling and trajectory control of a forklift-like wheeled robot", in *Proceedings of the ASME 2014 International Mechanical Engineering Congress and Exposition*, Nov. 2014. DOI: 10.1115/IMECE2014-37081.

[13] R. Community, *ROS wiki*, http://wiki.ros.org/, Accessed on April 28, 2023.

[14] *LD108 SLAMTEC RPLIDAR Datasheet A1M8 v3.0*, Online, Accessed: May 12, 2023.

[15] S. Corporation, *Slamtec a3 lidar*, https://www.slamtec.com/en/Lidar/A3, Accessed on April 28, 2023.

[16] P. Corporation, *Pololu 37d metal gearmotors*, https://www.pololu.com/file/0J1736/pololu-37d-metal-gearmotors-rev-1-2.pdf, Accessed on April 28, 2023.

[17] L. Batteries, *Lp9858102-3p 25000mah 3.7v rechargeable lipo battery with connector jst-phr-2b*, https://www.lipobatteries.net/hot-selling-lipo-battery/lp9858102-3p-25000mah-3-7v-rechargeable-lipo-battery-with-connector-jst-phr-2b/, Accessed on April 28, 2023.

[18] Basicmicro, *Roboclaw 2x30a motor controller*, https://www.basicmicro.com/RoboClaw-2x30A-Motor-Controller_p_9.html, Accessed on April 28, 2023.

[19] S. Electronics, *Roboclaw 2x60a motor controller datasheet*, https://cdn.sparkfun.com/assets/c/2/1/7/1/DS-16479-RoboClaw_2x60A_Motor_Controller.pdf, Accessed on April 28, 2023.

[20] R. P. Foundation, *Raspberry pi 4 model b*, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, Accessed on April 28, 2023.

[21] Raspberry Pi. "Raspberry pi camera module v2". (), [Online]. Available: https://www.raspberrypi.com/products/camera-module-v2/.

[22] S. Thrun and J. Leonard, "Simultaneous localization and mapping", in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., Berlin, Heidelberg: Springer, 2008. DOI: 10.1007/978-3-540-30301-5_38. [Online]. Available: https://doi.org/10.1007/978-3-540-30301-5_38.

[23] U. Frese and G. Hirzinger, "Simultaneous localization and mapping - a discussion", in *Proceedings of the IJCAI Workshop on Reasoning with Uncertainty in Robotics*, Seattle, 2001, pp. 17–26.

[24] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. DOI: 10.1109/TSSC.1968.300136.

[25] A. Javaid, "Understanding dijkstra algorithm", *SSRN Electronic Journal*, Jan. 2013. DOI: 10.2139/ssrn.2340905.

[26] G. E. Mathew, "Direction based heuristic for pathfinding in video games", *Procedia Computer Science*, vol. 47, pp. 262–271, 2015. DOI: 10.1016/j.procs.2015.03.206.

[27] L. W. Santoso, A. Setiawan, and A. K. Prajogo, *Performance analysis of dijkstra, a* and ant algorithm for finding optimal path - case study: Surabaya city map*, https://core.ac.uk/download/pdf/32452834.pdf, Informatics Department, Faculty of Industrial Engineering, Petra Christian University, Jl. Siwalankerto 121-131 Surabaya, 60236, 2016.

[28] A. Name, "Dijkstra's and a-star in finding the shortest path: A tutorial", in *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, IEEE, 2020. DOI: 10.1109/DATABIA50434.2020.9190342.

[29] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*, 2nd. Springer, 2016.

[30] R. N. Team, *Robotsetup*, http://wiki.ros.org/navigation/Tutorials/RobotSetup, Accessed on April 28, 2023.

[31] R. Community, *ROS wiki - robotsetup*, http://wiki.ros.org/navigation/Tutorials/RobotSetup, Accessed on May 16, 2023.

[32] S. Saat and et al., "Hectorslam 2d mapping for simultaneous localization and mapping (slam)", *J. Phys.: Conf. Ser.*, vol. 1529, p. 042 032, 2020. DOI: 10.1088/1742-6596/1529/4/042032.