PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

University of Amar Telidji - Laghouat

Faculty of Technology

Department of Electronics

MASTER THESIS

DOMAINE: Science & Technology

FIELD: Electronics

SPECIALTY: Embedded Systems

OTMANI Nour&MAKHLOUFI Maroua

Theme

Investigating and Implementing ROS-based Autonomous Navigation for Mobile Robots using SLAM Algorithms

Jury members:

ROUGAB Ilyas	MCB	President
REGUIEGUE Mourad	MCB	Examiner
OUBBATI Brahim Khalil	MCB	Supervisor
TOUHAMI Imane	PhD Student	Co-supervisor

2023 / 2024

ABSTRACT

Navigation is one of the most challenging tasks for a mobile robot. Mapping and navigation are becoming crucial in areas like exploration and automated vehicles, especially in places difficult for humans to access. To aid navigation, a Light Detection and Ranging (LiDAR) sensor is essential for creating maps of unknown environments. Our project focuses on mapping an unknown indoor space using Hector SLAM and GMapping techniques within the Robot Operating System (ROS). We used the Rviz tool in ROS to display the scanning results, and the maps created with LiDAR and ROS accurately match the real environment, making them ideal for indoor navigation. Additionally, we developed a navigation application where the robot first maps its current location. Then, we direct the robot to a new spot, and it moves there while updating and completing the map. This process repeats until the entire area is mapped. Our final results show that the robot can build maps and determine its location in real-time.

Keywords: Simultaneous Localization and Mapping (SLAM), Operating System ROS, Mobile Robot, Hector SLAM, GMapping, Navigation .

Résumé

La navigation est l'une des tâches les plus difficiles pour un robot mobile. La cartographie et la navigation deviennent cruciales dans des domaines tels que l'exploration et les véhicules automatisés, en particulier dans les endroits difficiles d'accès pour les humains. Pour faciliter la navigation, un capteur de détection et de télémétrie par ondes lumineuses

(LiDAR) est essentiel pour créer des cartes d'environnements inconnus. Notre projet se concentre sur la cartographie d'un espace intérieur inconnu en utilisant les techniques Hector SLAM et GMapping dans le Robot Operating System (ROS). Nous avons utilisé l'outil Rviz dans ROS pour afficher les résultats du balayage, et les cartes créées avec LiDAR et ROS correspondent exactement à l'environnement réel, ce qui les rend idéales pour la navigation intérieure. En outre, nous avons développé une application de navigation dans laquelle le robot cartographie d'abord sa position actuelle. Ensuite, nous dirigeons le robot vers un nouvel endroit, et il s'y déplace tout en mettant à jour et en complétant la carte. Ce processus se répète jusqu'à ce que toute la zone soit cartographiée. Nos résultats finaux montrent que le robot peut construire des cartes et déterminer sa position en temps réel.

Mots-clés: Simultaneous Localization and Mapping (SLAM), Système d'exploitation ROS, Mobile Robot, Hector SLAM, GMapping, Navigation.

ملخص

التنقل يعتبر واحد من أصعب المهام بالنسبة للروبوت المتنقل. رسم الخريطة والتنقل يصبحان مهمين للغاية في مجالات مثل الاستكشاف والمركبات الآلية، خاصة في الأماكن التي يصعب على البشر الوصول إليها. لمساعدة في التنقل، يتطلب الأمر استخدام مستشعر الليدار لإنشاء خرائط للبيئات غير المعروفة. يتمركز مشروعنا على رسم خريطة لمساحة داخلية غير معروفة باستخدام تقنيات هيكتور سلام و جي مابينغ ضمن نظام التشغيل للروبوت . استخدمنا أداة أرفيز في روس لعرض نتائج المسح، وتتطابق الخرائط التي تم إنشاؤها باستخدام ليدار و روس بشكل جيد مع البيئة الحقيقية، مما يجعلها مناسبة للتنقل داخل المباني. كما قمنا بعمل تطبيق للتنقل حيث يقوم الروبوت أولاً برسم خريطة لموقعه الحالي. ثم نوجه الروبوت إلى موقع جديد، ويتحرك إليه مع تحديث واستكمال الخريطة. يتكرر هذا العملية حتى يتم رسم الكامل للمنطقة. تظهر نتائجنا النهائية أن الروبوت قادر على بناء الخرائط والعثور على موقعه في الوقت الحقيقي.

الكلمات المفتاحية: تحديد الموقع المتزامن ورسم الخرائط سلام ، نظام التشغيل للروبوت روس ، هيكتور سلام ، جي مابينغ ، الروبوت المتنقل ، التنقل .

ACKNOWLEDGEMENTS

We would like to express our heartfelt gratitude to all those who have supported and inspired us throughout this journey.

First, we want to acknowledge Wall-E, the robot from the popular animated movie. Wall-E sparked our interest in robotics and made us believein the potential of robots to shape our future . Watching Wall-E was a transformative experience that ignited our passion for robotics and convinced us that they will play a pivotal role in the world ahead.

Our sincere appreciation goes to supervisor, Mr. OUBBATI Brahim Khalil.His guidance, encouragement, and unwavering support were instrumental in the successful completion of our Master's thesis. We are deeply grateful for his insights and advice, which have been invaluable throughout this process.

We also thank our co-supervisor, Ms. TOUHAMI Imane , for her valuable inputs and suggestions. Her feedback has improved our work, and her support has kept us motivated.

To our families, we owe our deepest thanks. Your endless love, understanding, and support have been our strength throughout this academic journey. You have been our pillar in tough times and our cheerleaders in good times. The sacrifices you made, the patience you showed, and the faith you placed in us have been the foundation of our success. We are eternally grateful for your belief in our dreams and for always being there with a steady hand and a loving heart. This achievement is as much yours as it is ours, and we dedicate this thesis to you with all our love and appreciation.

OTMANINour&MAKHLOUFI Maroua

Laghouat University

July 2024

ACRONYMS

2D Two Dimension

3D Three Dimension

AGV Automated Guided Vehicle

AMR Autonomous Mobile Robot

BoW Bag-of-Word

CNN Convolutional Neural Network

EKF Extended Kalman Filters

GNSS Global Navigation Satellite System

GPS The Global Positioning System

GMapping Grid-based Fast SLAM

IMU Inertial Measuring Unit

INS Inertial Navigation System

LiDAR Light Detection and Ranging

PID Proportional–integral–derivative controller

ROVs Underwater Robots

ROS Robot Operating System

RViz ROS Visualization

SLAM Simultaneous Localization and Mapping

UAV Unmanned Aerial Vehicle

CONTENTS

LIST OF FIGURES

GENERAL INTRODUCTION

Robotics has greatly advanced technology, transforming industries and aiding humans in tasks that are risky, repetitive, or too difficult. Autonomous Mobile robots, especially, are versatile machines that can move independently in various environments. They use sensors and advanced software to navigate, perform inspections, deliver items, and assist in emergencies. These robots play a vital role in improving safety and efficiency in many fields, making them essential tools in our evolving world.

The goal of this thesis is to navigate a two-wheeled robot in various environments and create a map using SLAM(simultaneous localisation and mapping). This operation focuses on several essential components, including Jetson NANO and various sensors. For this project, we utilized the basic SLAM algorithm known as gmapping This algorithm generates a 2D map.

our master thesis Divided into four chapters:

- **chapter one:** In this chapter, we explored autonomous robots from a theoretical perspective. We provided a detailed definition of autonomous robots, discussed their primary types, and reviewed navigation systems employed before SLAM technology. Lastly, we talked about ros, how it works, and the most important archetectur that works with it.
- **chapter two:** We started by presenting a mathematic model for a two-wheeled robot, then we moved on to Salam's study, where we initially addressed the most important problems facing Salam in its development. We also provided a comprehensive overview of the various Salam algorithms, so that we studied the theory of two basic algorithms (Hector Salam and gmapping). Then we explored an important

part of Salam, which is the loop closur, and its role in improving the professionalism and quality of the map. Finally, we saw how navigation is done for robots in the Salam system, starting from Determine the location and generate a map.

- **chapter three:**

  We will explain the most important components of the robot, mentioning the characteristics of each component and its cost. Then we will communicate between the robot and the computer and vice versa to ensure and enhance the robot's independence, then we will work to calibrate the robot to perform its functions accurately and professionally. In addition, we will activate the lidar node. Finally, we will draw maps using the Hector Slam algorithm and the gmapping algorithm in two different environments, in addition moving the robot using a joystick.

- **chapter four:** In summary, this chapter has detailed our methodology for implementing automated navigation, with an emphasis on map building and route planning using cutting-edge technologies such as LIDAR. Working closely with our colleagues, we have seamlessly integrated PID controllers to achieve precise and efficient navigation from origin (point A) to destination (point B) while simultaneously detecting and avoiding obstacles.

CHAPTER 1

THE STATE OF THE ART OF AUTONOMOUS ROBOT

1.1 Introduction

Robotics has achieved significant success, particularly in the realm of industrial manufacturing. However, addressing human needs solely within this domain has proven to be insufficient. The vision of personal robots being commonplace in homes, workplaces, and as substitutes for human-performed tasks has driven further advancements in robotics. This broader perspective has led to the development of autonomous robots designed for diverse environments and applications[1].

This chapter focuses on autonomous robots, offering a comprehensive overview of their various types. We delve into different autonomous navigation systems, highlighting that more advanced systems typically improve robot performance. Additionally, we discuss how the Robot Operating System (ROS) facilitates robotics development. ROS provides an extensive suite of tools, libraries, and communication structures that support modular applications, making the creation and management of complex robot behaviors more straightforward.

1.2 Autonomous Robots

1.2.1 Definition

Autonomous robots are advanced machines that operate independently without constant human guidance. They can navigate and determine their position on their own, making them useful for various tasks, especially those that are dangerous or physically demanding. Higher levels of autonomy lead to better performance.

These robots are versatile and can work in different environments, including underwater, underground, in space, and in the air. This adaptability makes them valuable for various applications, from exploration to industrial tasks. The development of autonomous robots is a significant milestone in robotics, offering new capabilities and advancing human efforts[2].

1.3 Types of autonomous robots

1.3.1 AGV

An **Automated Guided Vehicle (AGV)** is a robotic system that moves around a facility on its own to do tasks like transporting goods. It uses sensors like lasers or cameras to navigate, detect obstacles, and work safely. AGVs improve accuracy, reduce errors, and make material handling safer[3].

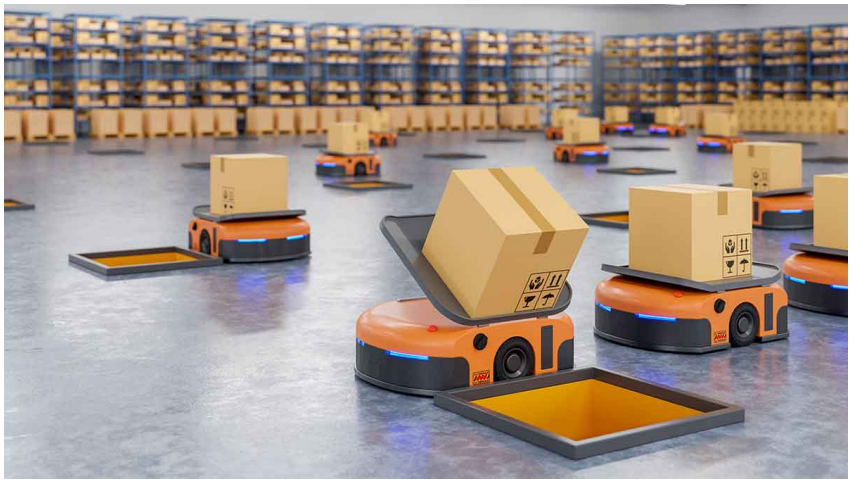

Figure 1.1: Automated Guided Vehicle (AGV)

1.3.2 AMR

An **Autonomous Mobile Robot (AMR)** is an improved version of an Automated Guided Vehicle (AGV). Unlike AGVs, which rely on physical guides or markers, AMRs use on-board sensors and processors to move items independently. The main advantage of AMRs is their flexible navigation, as they plan their own routes using a map and can dynamically change paths to avoid obstacles. This makes AMRs more versatile and intelligent than AGVs. Additionally, AMRs are more economical due to their adaptability and easy setup[3].

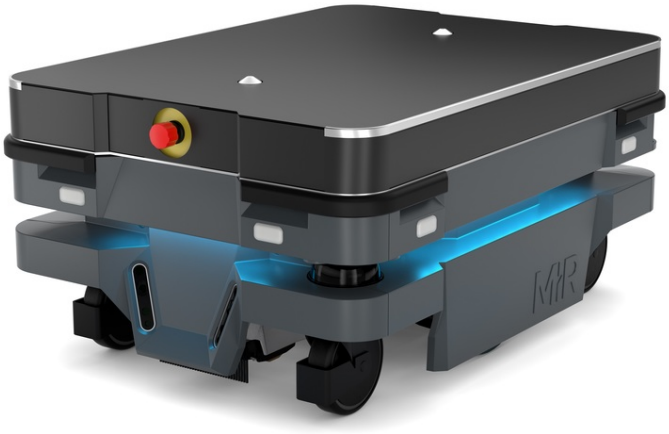

Figure 1.2: Autonomous Mobile Robot

1.3.3 Unmanned Aerial Vehicle (UAV)

Unmanned aerial vehicles (UAVs) are pilotless aircraft equipped with advanced communication components such as a ground control station and sensors. Initially used for tasks like search and rescue, mapping, and surveillance, UAVs have greatly evolved with modern technology. Today, they are also used for emergency evacuations during natural disasters like storms and floods.[4].

Figure 1.3: Expected reach of UAVs in various applications.

1.3.4 Underwater Robots (ROVs)

Large-scale projects like seabed research, archeology, and hull wreck searches have been the main uses of underwater imaging technology in recent years. AUVs, or autonomous underwater vehicles, are used for underwater operations involving prolonged, regular or hazardous features, such as maintenance and inspections, which emphasize the need for using underwater vision technology on AUVs[5].

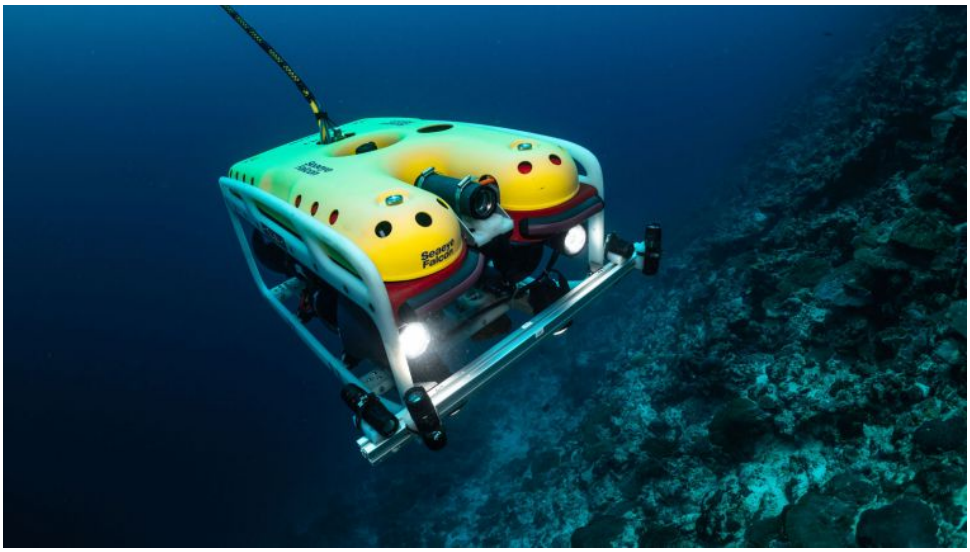

Figure 1.4: Underwater Robots (ROVs).

1.4 Autonomous Navigation System

1.4.1 definition

Autonomous navigation is the ability of a robot to move independently within its environment. The complexity of the navigation technique depends on the characteristics of the operating environment, such as whether it is indoors or outdoors, level or uneven terrain, and the workspace arrangement. It also depends on how familiar the robot is with the environment, whether it is dynamic or static[6].

1.4.2 Navigation Systems

- **GNSS:** Global Navigation Satellite System (GNSS) receivers get positioning and timing data from satellites in Medium Earth Orbit (MEO). These satellites help locate both stationary and moving objects and are used in robotics, geography, surveying, and navigation[7]

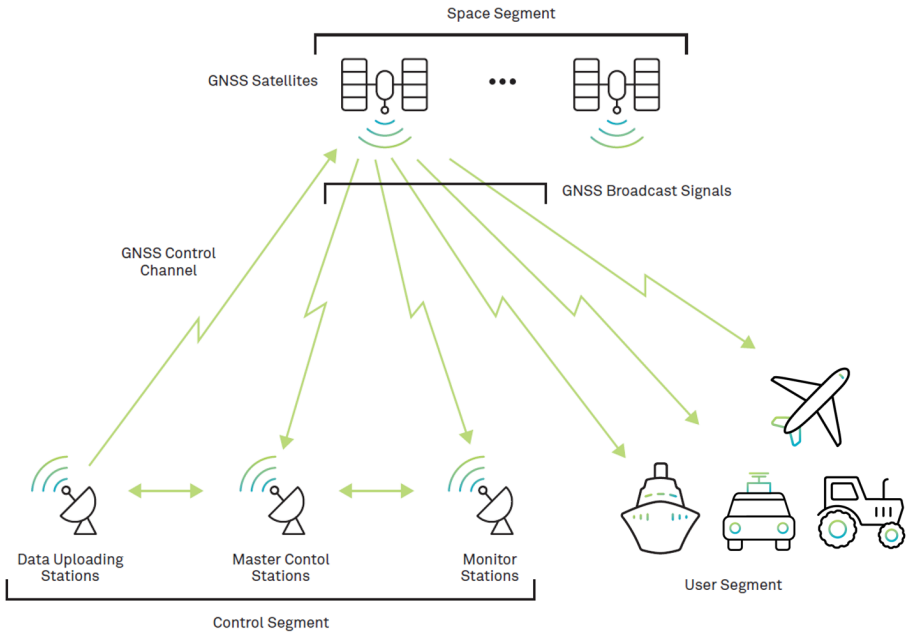

Figure 1.5: GNSS components.

- **GPS:**The Global Positioning System (GPS) has 24 satellites owned by the US government. These satellites are positioned around 20,200 km above the Earth in Medium Earth Orbit (MEO). Users can access Position, Navigation, and Time (PNT) services worldwide from the GPS system[7].

- **IMU:**The acronym for Inertial Measuring Unit is IMU. Gyroscopes and accelerometers are sensors used in IMUs to measure an object's angular position and displacement in three dimensions[8].

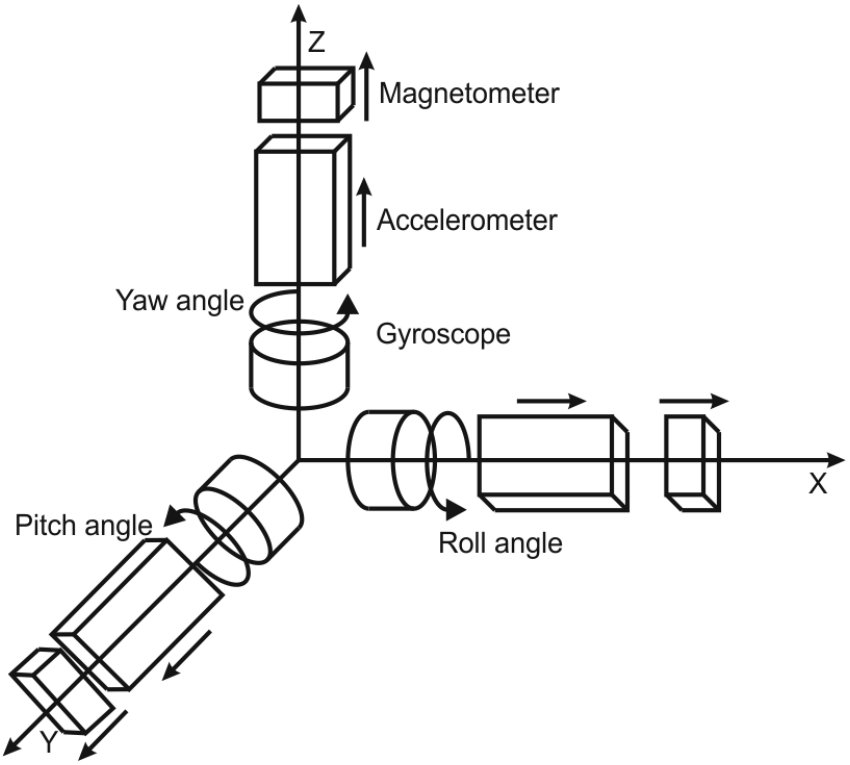

Figure 1.6: Inertial Measuring Unit.

- **INS:** The acronym for an inertial navigation system is INS. It combines gyroscopes and accelerometers from IMU sensors with a GNSS receiver for absolute position data. Magnetometers may also be used to measure magnetic fields. INS utilizes advanced data processing like Kalman filtering to calculate an object's real-time location and vector from its known starting position. It's used in vehicles like cars, submarines, and aircraft for 3D navigation[8].

1.5 Robot Operating System (ROS)

1.5.1 Definition

ROS, or the **Robot Operating System**, is a software framework used to develop robot applications. It offers tools and libraries that make it easier to create, simulate, and deploy robotic systems. With ROS, developers can easily connect different hardware

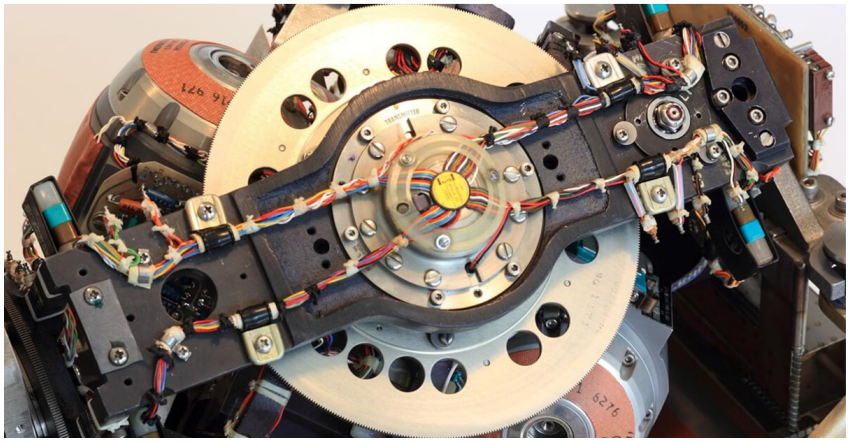

Figure 1.7: Inertial navigation systemst.

parts and focus on programming tasks like perception and navigation. It's widely used in robotics research, education, and industrial applications[9].

Notable versions of ROS include ROS Kinetic, ROS Melodic, and ROS Noetic. Each version brings improvements and works with different operating systems and hardware. ROS Kinetic, released in 2016, supports Ubuntu 16.04 and has long-term support until 2021. ROS Melodic, released in 2018, supports Ubuntu 18.04 with long-term support until 2023. ROS Noetic, released in 2020, supports Ubuntu 20.04 and offers improvements in usability, performance, and compatibility with newer software libraries.

1.5.2 Architecture of ROS

The architecture of ROS (Robot Operating System) is designed to simplify the development of robot applications. Here are the main components:

- **ROS Master:** The central manager that keeps track of all the nodes and manages communication between them.
- **ROS Nodes:** Independent programs that perform specific tasks. For example, one node might control a robot's wheels, while another node processes camera images.
- **ROS Topics:** Channels through which nodes communicate. Nodes can publish messages to topics or subscribe to topics to receive messages.
- **Messages:** Data packets sent between nodes via topics. They can contain various types of information, like sensor readings or control commands.

- **Launch Files:** XML files that start multiple nodes at once and set parameters for them.

1.5.3 Example of Two Nodes Communicating

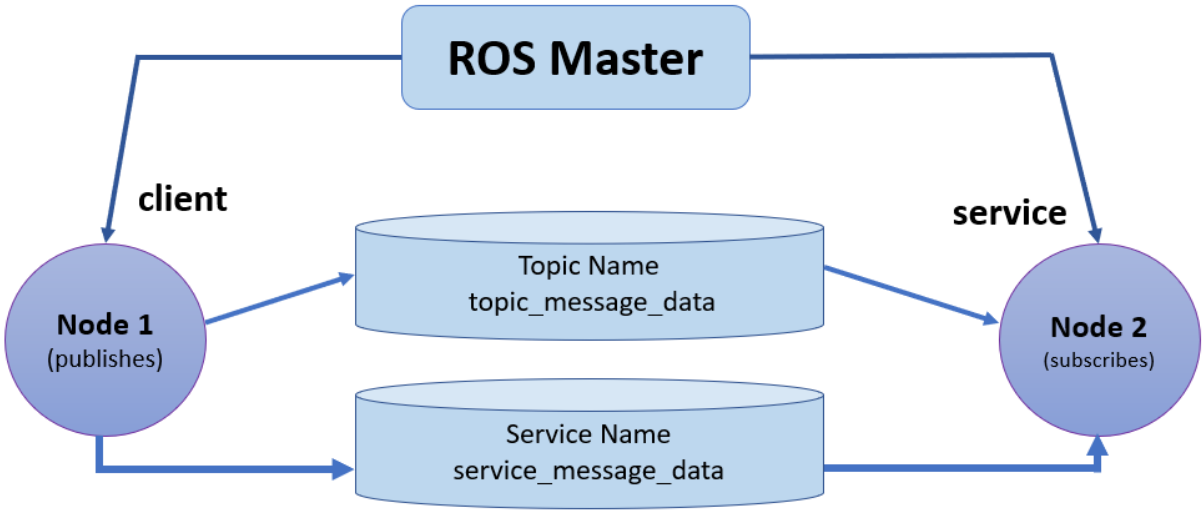

Figure 1.8: ROS Communication of Two Nodes diagram.

- **ROS Master:**
  - Manages communication between nodes.
  - Keeps track of Node 1 and Node 2.
- **ROS Nodes:**
  - **Node 1 :**

    Sends messages (publishes) to a topic.

    Requests information (client) from Node 2.
  - **Node 2 :**

    Receives messages (subscribes) from the topic.

    Provides information (service) to Node 1.
- **Topic Name:**
  - A named service for request and reply.
  - Node 1 requests the service.

  - Node 2 responds to the service request.

How It Works

- **Publishing and Subscribing:**
  - Node 1 publishes (sends) messages to the topic.
  - Node 2 subscribes (receives) those messages.
- **Service Communication:**
  - Node 1 asks for a service (client).
  - Node 2 answers the service request (service).

1.6 Conclusion

In the previous chapter, we gained insights into the world of autonomous robots, exploring their evolution and the array of navigation systems they utilize, including GNSS, GPS, IMU, and INS. We also delved into the significance of the Robot Operating System (ROS) in simplifying robotics development through its comprehensive suite of tools and communication structures.

In the upcoming chapter, our focus shifts to SLAM (Simultaneous Localization and Mapping), a groundbreaking advancement in robotics. SLAM is pivotal for robots, enabling them to effectively navigate and operate by estimating their position and orientation in real-time within their surroundings. By integrating data from various sensors, SLAM empowers robots to autonomously map and navigate dynamic environments, marking a significant milestone in the journey toward sophisticated autonomous systems.

CHAPTER 2

SIMULTANEOUS LOCALIZATION AND MAPPING(SLAM)

2.1 Introduction

Mobile robots rely on simultaneous localization and mapping (SLAM) to create maps of their surroundings and accurately determine their own position. Recent advancements have made significant progress in solving the SLAM problem, resulting in several effective solutions. Improving computational efficiency and ensuring precise mapping and localization estimations have been primary research objectives.

This chapter aims to provide a comprehensive introduction to SLAM, starting from basic concepts and progressing to more advanced algorithms and navigation methods. Before delving into various SLAM algorithms, we begin by examining the mathematical model for mobile robots (specifically for two-wheel robots). We then discuss the challenges and complexities associated with SLAM.

Following this, we explore LiDAR-based SLAM algorithms such as Hector SLAM, GMapping, and pre-mapping, known for their accuracy. We also cover loop closure and its significance in SLAM systems, followed by an overview of SLAM-based navigation systems that encompass localization, path planning, navigation, motion planning, and mapping.

2.2 Mathematical model for mobile robot (for two-wheel)

To study a two-wheeled mobile robot practically, we first need to understand its mathematical model. This model uses math equations to describe the robot's motion, dynamics, and control methods. By modeling the robot mathematically, we can simulate and predict its behavior in different situations and improve its performance. The model usually includes kinematics, dynamics, control theory, and optimization concepts.

2.2.1 Scenarios of robot's mouvements

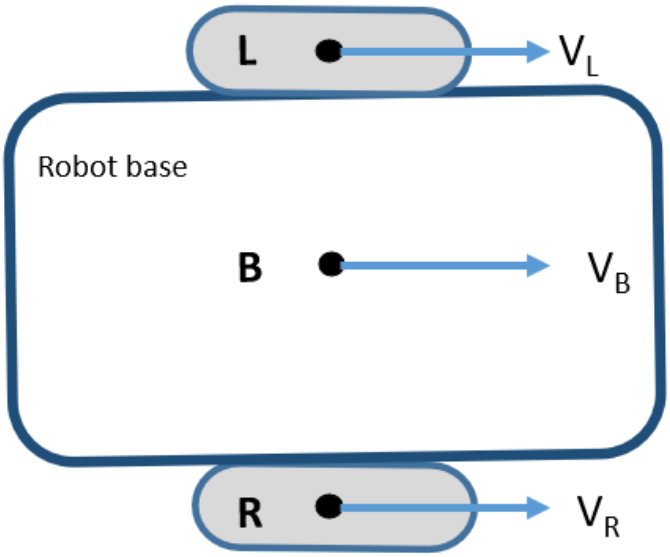

Figure 2.1: mobile robot moving straight.

In the figure 2.1, the robot is moving straight ,this because the intensities of the velocities V_L and V_R are equal . consequently , the instantaneous center of rotation is at infinity and all the points describe straight lines.

The robot is turn left , this is because the intensity of the velocity V_R is large than the intensity of the velocity V_L (see Figure 2.2).

When the obot is turn right because the intensity of the velocity V_L is large than the intensity of the velocity V_R (see Figure 2.2).

In this figure 2.3 the robot turn around the point B. the rotation is due the fact duo the

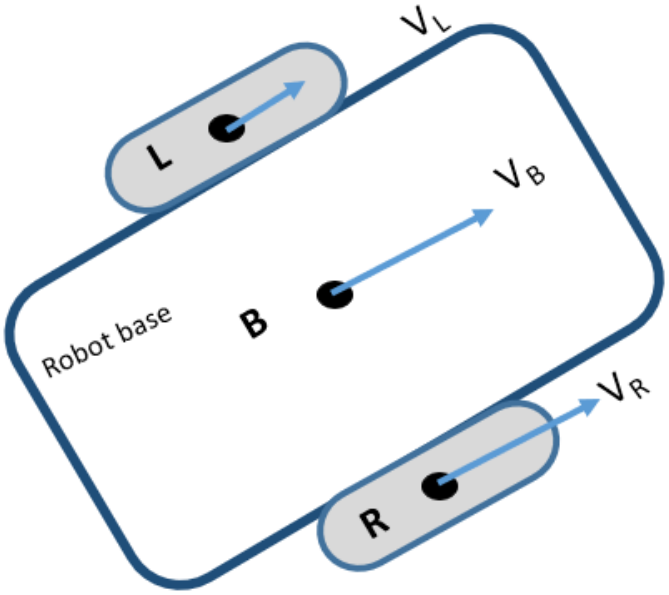

Figure 2.2: mobile robot turn.

velocities have identical intencities and opposite direction .

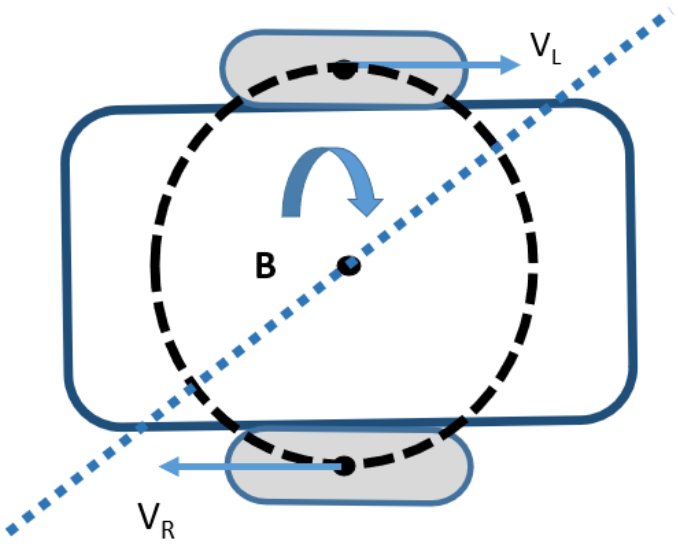

Figure 2.3: mobile robot turn around .

Summary

We may regulate the roboot's motion by varying the angular velocities of the two wheels, or alternatively, the velocities of the two wheel centers.

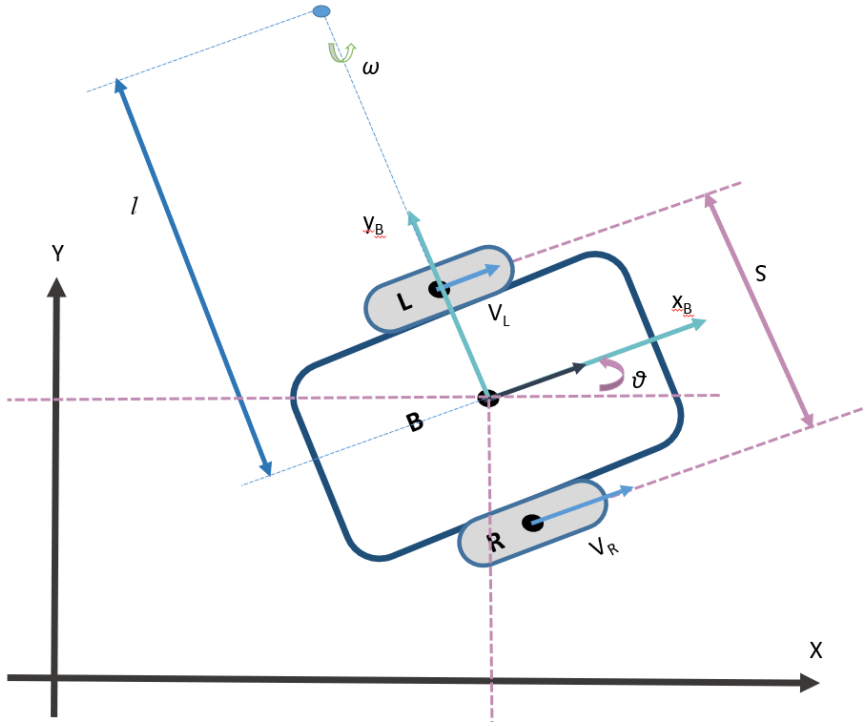

Figure 2.4: Kinematic Model.

Variables and definitions

- x: Position of the robot's center along the x-axis in the global coordinate frame.
- y: Position of the robot's center along the y-axis in the global coordinate frame.
- θ: Angle of rotation of the robot.
- x_b and y_b: Coordinates of the body frame.
- ω: Instantaneous angular velocity of the robot body.
- l: Distance between point B and point C.
- S: Distance between point R and point L.
- L: Center of the left wheel.
- R: Center of the right wheel.
- B: Point on the line connecting L and R (MIDDLE).
- V_R: Velocity of the center point of the right wheel.
- V_L: Velocity of the center point of the left wheel.
- V_B: Velocity of the center point of the line.

mathematic equations

extraction the equation of the Velocity of the center point of the right whee , Velocity of the center point of the left wheel and Instantaneous angular velocity of the robot body.

$$V_L = \omega \left(l - \frac{S}{2} \right), V_R = \omega \left(l + \frac{S}{2} \right) \tag{2.1}$$

$$\omega = \frac{V_L}{l - \frac{S}{2}} \tag{2.2}$$

$$V_R = \frac{V_L}{l - \frac{S}{2}} \left(l + \frac{S}{2} \right) \tag{2.3}$$

From (2.3) we obtain :

$$l = \frac{S(V_R + V_L)}{2(V_R - V_L)} \tag{2.4}$$

Now by substituting equation (2.4) in the first equation we get :

$$\omega = \frac{V_R - V_L}{S} \tag{2.5}$$

$$\begin{aligned} \dot{x} &= v \cos(\theta) \\ \dot{y} &= v \sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{2.6}$$

The final equation (2.6) can be expressed succinctly

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V_B \\ \omega \end{bmatrix} \tag{2.7}$$

In the other hand , we have :

$$V_B = \omega \cdot l \tag{2.8}$$

by substituting equation l and ω in the last equation (2.8) we get :

$$V_B = \frac{V_R + V_L}{2} \tag{2.9}$$

By combining this equation (2.9) with the equation of ω from (2.5) we get:

$$\begin{aligned} V_B &= \frac{V_R + V_L}{2} \\ \omega &= \frac{V_R - V_L}{S} \end{aligned} \tag{2.10}$$

The last equation can be written :

$$\begin{bmatrix} V_B \\ \omega \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ -\frac{1}{S} & \frac{1}{S} \end{bmatrix} \begin{bmatrix} V_L \\ V_R \end{bmatrix} \tag{2.11}$$

by substituting equation (2.11) in the equation (2.8) we get :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{\cos(\theta)}{2} & \frac{\cos(\theta)}{2} \\ \frac{\sin(\theta)}{2} & \frac{\sin(\theta)}{2} \\ -\frac{1}{S} & \frac{1}{S} \end{bmatrix} \begin{bmatrix} V_L \\ V_R \end{bmatrix} \tag{2.12}$$

Here is how the previous system can be extended

$$\begin{aligned} \dot{x} &= \frac{V_L}{2}\cos(\theta) + \frac{V_R}{2}\cos(\theta) \\ \dot{y} &= \frac{V_L}{2}\sin(\theta) + \frac{V_R}{2}\sin(\theta) \\ \dot{\theta} &= -\frac{1}{S}V_L + \frac{1}{S}V_R \end{aligned} \tag{2.13}$$

2.3 SLAM algorithms

SLAM, or **Simultaneous Localization and Mapping**, enables robots to map their surroundings and determine their location using onboard sensors, despite challenges such as sensor inaccuracies and robot drift. While computationally demanding, SLAM is crucial for various applications like autonomous driving, search and rescue missions, underwater exploration, and collaborative robotics. Over the past two decades, SLAM techniques have evolved from using Extended Kalman Filters (EKF) to more advanced methods like particle filters and graph-based approaches. These methods, exemplified by algorithms such as HECTOR-SLAM, Gmapping, Cartographer, KARTO-SLAM, and RTAB-Map, aim to produce accurate maps, track robot positions effectively, and minimize computational resources.[10][11].

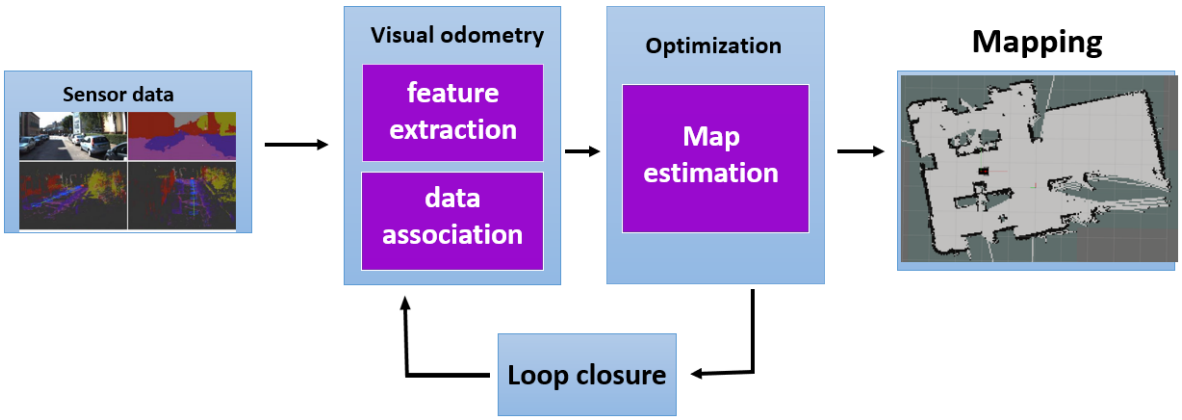

Figure 2.5: The typical visual SLAM system framework.

2.3.1 Difficulties and Challenges of SLAM.

Despite great advancements, these challenges still make it hard to use SLAM effectively in autonomous robots. More research is needed to solve these problems and make SLAM algorithms more reliable and efficient[12].
Here are some main problems SLAM faces[13]:

- **Accuracy and precision** One major challenge is achieving high accuracy in both the map and the robot's position, which is crucial for navigating tight spaces or performing precise tasks.
- **Dynamic Environments** In changing environments, where objects constantly move, SLAM algorithms struggle, leading to inaccurate maps and robot positions.
- **Limited Sensors** Many SLAM systems rely on sensors like cameras or laser scanners, which can be expensive, heavy, and consume a lot of power. This limits their use in small or low-power robots.
- **Computational Complexity** SLAM algorithms can be demanding on computational resources, making them hard to use in robots with limited processing power. This is particularly challenging for real-time applications where the robot must continuously update its map and position.
- **Sensor Noise and Data Association** One big challenge in SLAM is matching sensor readings from different times and places to the same real-world objects, known as the Data Association Problem. Getting this right is essential for accurate

SLAM. It's especially hard in changing environments where objects can appear or disappear, and sensor noise makes it even tougher[14].

2.4 LiDAR-based SLAM alghrithmes

lidar-based SLAM (Simultaneous Localization and Mapping) algorithms use laser sensors to create real-time maps of an environment while tracking the device's location within that environment. This technology is essential for autonomous vehicles, drones, and robots, enabling them to navigate accurately and safely in unfamiliar terrain.

2.4.1 Hector SLAM's scan matching algorithm

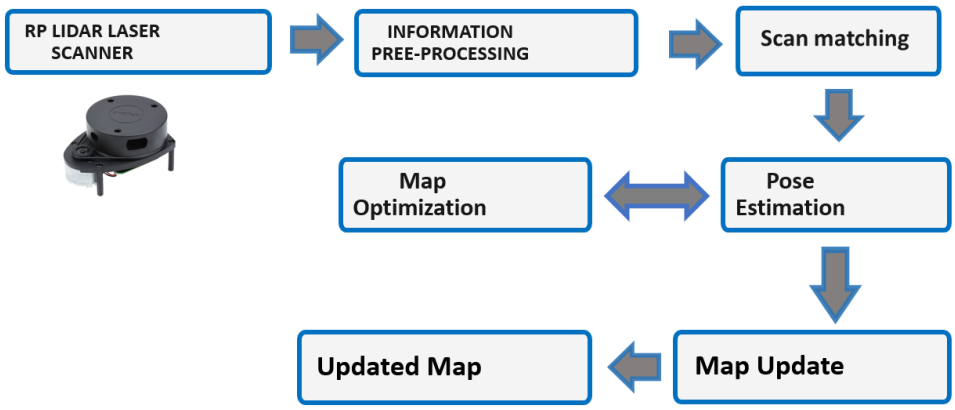

Figure 2.6: Diagram of Hector SLAM Algorithm .

A mapping algorithm, like the one we are employing, Hector SLAM, must "scan match" in order for the mapping process to succeed.An illustration using a multidimensional (2D or 3D) tessellation of space into individual cells

where a probabilistic approximation of each cell's state is stored. "The state variable s(C) is defined as a discrete random variable with two states, occupied and empty, associated with a cell C of the occupancy grid."
Let's take a look at a point, p_m, on the continuous map. The occupancy value, $M(P_m)$, will give the gradient the following shape when we talk about the scan match[15]:

$$\nabla M(Pm) = (\frac{\partial M(Pm)}{\partial x}, \frac{\partial M(Pm)}{\partial y}) \tag{2.14}$$

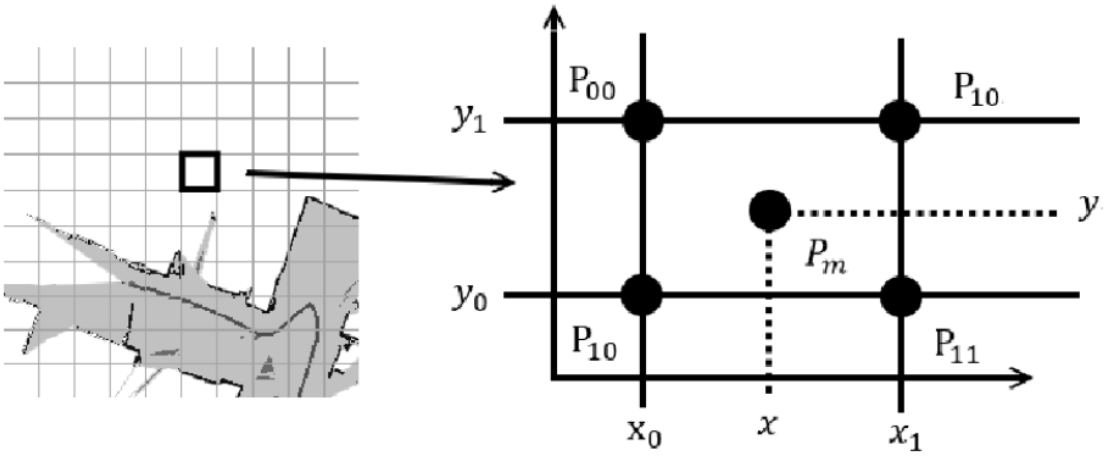

Figure 2.7: Occupancy grid cell representation using Bilinear method .

Sub-grid cells are interpolated using bilinear filtering in order to derive occupancy derivatives & probability. In this manner, the surface's discrete value on the map will be continuous at any given location[15]. Additionally, by utilizing this linear interpolation technique with closest You may display the integer coordinates P_{00}, P_{01}, P_{10}, andP_{11} by:

$$M(P_m) \approx \frac{y-y0}{y1-y0}(\frac{x-x0}{x1-x0}M(P_{11}) + \frac{x1-x}{x1-x0}M(P_{01}))$$

$$+\frac{y1-y}{y1-y0}(\frac{x-x0}{x1-x0}M(P_{10}) + \frac{x1-x}{x1-x0}M(P_{00}) \tag{2.15}$$

Also, the derivatives of the map at any certain point can be shown as[15]:

$$\frac{\partial M(Pm)}{\partial x} \approx \frac{y-y0}{y1-y0}(M(P_{11}) + M(P_{01})) + \frac{y1-y}{y1-y0}(M(P_{11}) + M(P_{01})) \tag{2.16}$$

$$\frac{\partial M(Pm)}{\partial x} \approx \frac{x-x0}{x1-x0}(M(P_{11}) + M(P_{01})) + \frac{x1-x}{x1-x0}(M(P_{11}) + M(P_{01})) \tag{2.17}$$

Scan matching involves aligning the ends of beams from a map with those from the scanned environment. To predict the next position without extra data, a Gauss-Newton method is used. This method adjusts a transformation, denoted by $\xi = (p_x, p_y, \psi)^T$,

where ψ = yaw.The aim is to minimize this transformation to achieve the best alignment between the scanned data and the map. This is achieved by minimizing:S_i

$$\xi^* = \arg\min_{\xi} \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2 \tag{2.18}$$

This is where the Gauss-Newton method steps in to solve this non-linear least squares problem. We define a target function r_i to guide our solution.

$$r_i = 1 - M(S_i(\xi)) \quad (2.5) \tag{2.19}$$

The function $M(S_i(\xi))$ gives us the value of the map at the coordinates provided by $S_i(\xi)$. With our initial guess being ξ, we have to compute $\nabla\xi$[16]. According to Newton's method, the iterative process to minimize the target function is given by:

$$\mathbf{\Delta}\xi = \xi_{t-1} - \mathbf{H}^{-1}\mathbf{G} \tag{2.20}$$

Here, G represents the gradient vector, which shows the direction and magnitude of change, while H stands for the Hessian matrix. This matrix is calculated by ignoring the second-order derivative terms. We assume that the robot has moved very slightly, so the change $\nabla\xi$, is so small that we can disregard it:

$$G = \sum_{i=1}^{n} r_i \frac{\partial r_i}{\partial \xi} \tag{2.21}$$

$$= \sum_{i=1}^{n} \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]^T \left[1 - M(S_i(\xi)) - \nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \nabla\xi \right] \tag{2.22}$$

$$= \sum_{i=1}^{n} \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]^T [1 - M(S_i(\xi))] \tag{2.23}$$

As mentioned earlier, H, can be denoted as:

$$H = \sum_{i=1}^{n} \frac{\partial r_i}{\partial \xi} \frac{\partial r_i}{\partial \xi} = \sum_{i=1}^{n} \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]^T \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right] \tag{2.24}$$

And the derivative of $S_i(\xi)$ can be represented as[16]:

$$\frac{\partial S_i(\xi)}{\partial \xi} = \begin{pmatrix} 1 & 0 - \sin(\psi) & -\cos(\psi) s_{i,x} \\ 0 & 1 \cos(\psi) & \sin(\psi) s_{i,y} \end{pmatrix} \tag{2.25}$$

Now, solving for $\nabla\xi$, minimizes our problem to be:

$$\nabla\xi = H^{-1} \sum_{i=1}^{n} \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]^T [1 - M(S_i(\xi))] \tag{2.26}$$

For each endpoint to converge correctly, Hector SLAM uses scan matching with a point coordinate on the map. This depends on the non-smooth linear approximation mentioned earlier.

2.4.2 Gmapping algorithm

This algorithm, inspired by the particle filter with Rao-Blackwellization, helps figure out where a robot is by using probabilities from past data. It improves its position estimates by looking at previous information and maps. It also corrects itself by considering how it's moved (odometry data) and updating its guesses and the map. GMapping is a popular tool in robots, especially indoors, because it can figure out where the robot is and make maps using inexpensive sensors.

MAPPING WITH RAO-BLACKWELLIZED PARTICLE FILTERS

The Rao-Blackwellized particle filter for SLAM estimates the joint posterior $p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1})$,which includes the map m and the robot's trajectory $x_{1:t} = x_1, \ldots, x_t$, based on sensor observations $z_{1:t} = z_1, \ldots, z_t$ and odometry measurements $u_{1:t-1} = u_1, \ldots, u_{t-1}$. This estimation uses the factorization[17]:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) = p(m \mid x_{1:t}, z_{1:t}) \cdot p(x_{1:t} \mid z_{1:t}, u_{1:t-1}) \tag{2.27}$$

First, the robot's trajectory is estimated, then the map is computed based on this trajectory. This approach, called Rao-Blackwellization, allows efficient computation. The posterior over maps $p(m \mid x_{1:t}, z_{1:t})$ can be calculated analytically using "mapping with known poses."

A particle filter is used to estimate the posterior $p(x_{1:t} \mid z_{1:t}, u_{1:t-1})$. Each particle represents a possible trajectory, and an individual map is associated with each particle. The process includes four main steps[17]:

1 **Sampling:** Generate the next set of particles from the current set using a probabilistic odometry motion model.

2 **Importance Weighting:** Assign a weight to each particle based on the importance sampling principle:

$$w_t(i) = \frac{p(x_{1:t}^{(i)} \mid z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)} \mid z_{1:t}, u_{1:t-1})} \tag{2.28}$$

3 **Resampling:** Draw particles with replacement proportional to their weights to maintain a finite number of particles approximating a continuous distribution.

4 **Map Estimation:** For each particle, compute the corresponding map estimate $\cdot p(m(i) \mid x_{1:t}^{(i)}, z_{1:t})$ based on the particle's trajectory and the observation history.

The weights are computed recursively to avoid inefficiency, using the following formulation:

$$w_t^{(i)} \propto \frac{p(z_t \mid m_{t-1}^{(i)}, x_t^{(i)}) p(x_t^{(i)} \mid x_{t-1}^{(i)}, u_{t-1})}{\pi(x_t \mid x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1})} \cdot w_{t-1}^{(i)} \tag{2.29}$$

The implementation must accurately compute the proposal distribution and adaptively perform resampling to ensure accurate and efficient mapping.

2.5 Loop Closure

An essential part of SLAM systems is loop closure detection, which seeks to identify previously visited locations (true loops) with high accuracy in order to improve mapping precision and minimize cumulative drift. Occlusion, fluctuating ambient conditions, and computing complexity are problems for current approaches. While convolutional neural network (CNN) features and Bag-of-Words (BoW) features are explored in recent works, classic point feature-based techniques like SIFT and SURF are computationally costly. BoW techniques are effective yet memory-intensive, whereas CNN-based techniques have

good potential for resilience. The difficulty of identifying real loops is still unresolved despite progress. Although there are taxonomies for classifying loop closure detection algorithms, this review mainly concentrates on visual and LiDAR-based techniques, classifying them as vision-based, LiDAR-based, and deep learning-based methods[18].

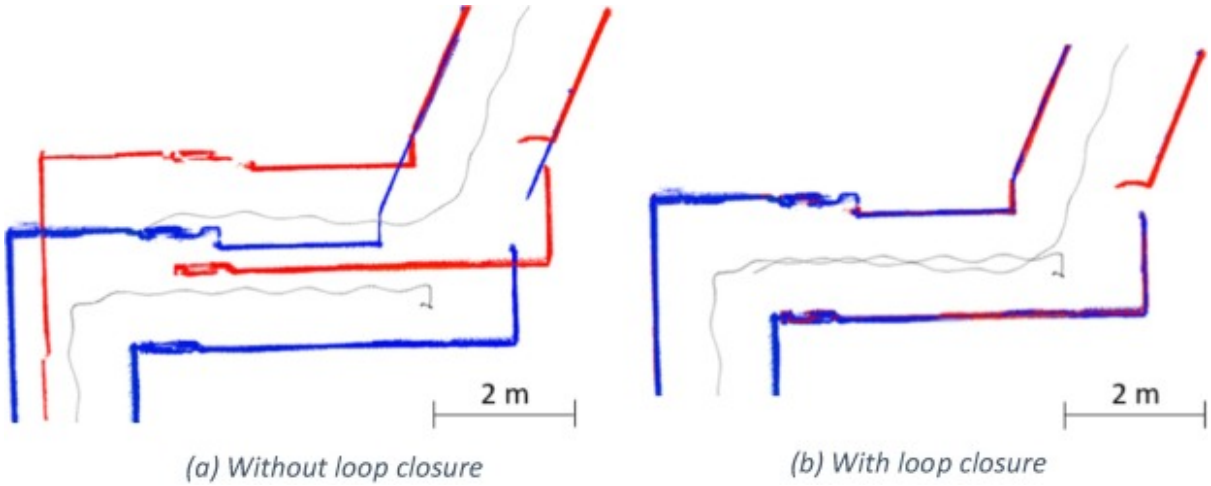

Figure 2.8: The first-floor loop in the ITC building (ITC f1 largeLoop) which is the largest loop in our experiments. (a) Top view of the walls' points with trajectory (black) at the start (blue) and end (red) of the loop without loop closure. (b) The walls' points and the trajectory after loop closure correction [20].

2.6 Navigation system based on SLAM

2.6.1 Localization

Localization based on SLAM is a dynamic process that enables robots to navigate unknown environments by simultaneously constructing maps and estimating their own positions within them. It involves data collection using sensors, feature extraction, map creation, and iterative pose estimation. This iterative cycle ensures real-time accuracy in localization, allowing robots to navigate autonomously.[19][20].

2.6.2 Path Planning and Navigation

SLAM helps robots map and locate themselves, vital for navigating new areas. It uses sensors like cameras and lasers to gather data and create maps. The goal is to accurately determine the robot's position in real-time, improving as it recognizes landmarks.

Challenges include dealing with sensor errors and matching data to the map. Advanced techniques like loop closure fix mistakes and keep maps accurate for better navigation.[21] [22].

2.6.3 Motion Planning

Motion planning in robotics, a vital field since the late 1960s, involves choosing the best motion for a robot while considering uncertainties. It aims to meet all constraints, such as avoiding collisions and obeying speed limits. There are two types of algorithms: explicit and implicit. Explicit planning breaks the process into route planning, trajectory planning, and robot control to find a path for the robot to complete its tasks efficiently[23].

2.6.4 Mapping

A mobile robot uses SLAM to create a map and understand its location in relation to that map. It gathers data from onboard sensors like cameras and LiDAR while exploring uncharted areas. SLAM systems use probabilistic techniques to handle sensor and movement errors, improving map accuracy over time. Mapping based on SLAM is vital for robotic tasks like navigation and surveillance in unfamiliar or changing environments[24].

2.7 Conclusion

In this chapter, we looked at the theories and methods behind Hector SLAM and GMapping algorithms, focusing on their mathematical models and how they work. Hector SLAM is ideal for low-noise environments because it updates quickly, while GMapping works well in complex, noisy settings with its particle filter approach. We also talked about how these algorithms are used within the ROS framework, showing its flexibility for advanced robotic applications. This theoretical foundation sets the stage for the practical implementations we'll cover in the next chapter

CHAPTER 3

BUILDING 2-D MAPS

3.1 Introduction

In this chapter, we will delineate the various components of the robot, detailing their characteristics and functionalities. Subsequently, we will address the communication protocols between the robot and the computer to enhance the robot's autonomy. Additionally, we have calibrated the robot to ensure its precision and proficiency in executing tasks. Finally, we initiated the implementation of algorithms such as Gmapping and Hector SLAM, creating corresponding maps within a virtual machine. Furthermore, we will discuss the application of joystick-based control for robot navigation. All of these applications are a prelude to navigation.

3.2 Mobile Robot

The mobile robot that we use in our project is JetBot ROS (Robot Operating System) is an advanced version of the JetBot, an educational robot kit powered by NVIDIA's Jetson Nano. It integrates with the ROS framework, which is widely used in robotics. This integration allows users to utilize a large collection of software libraries and tools for developing complex robotic applications. JetBot ROS enables the robot to perform tasks such as autonomous navigation, object detection, and real-time data processing. It supports communication between different components, making it easier to test and

implement algorithms. This combination provides a powerful platform for learning and experimenting with robotics and AI [25].

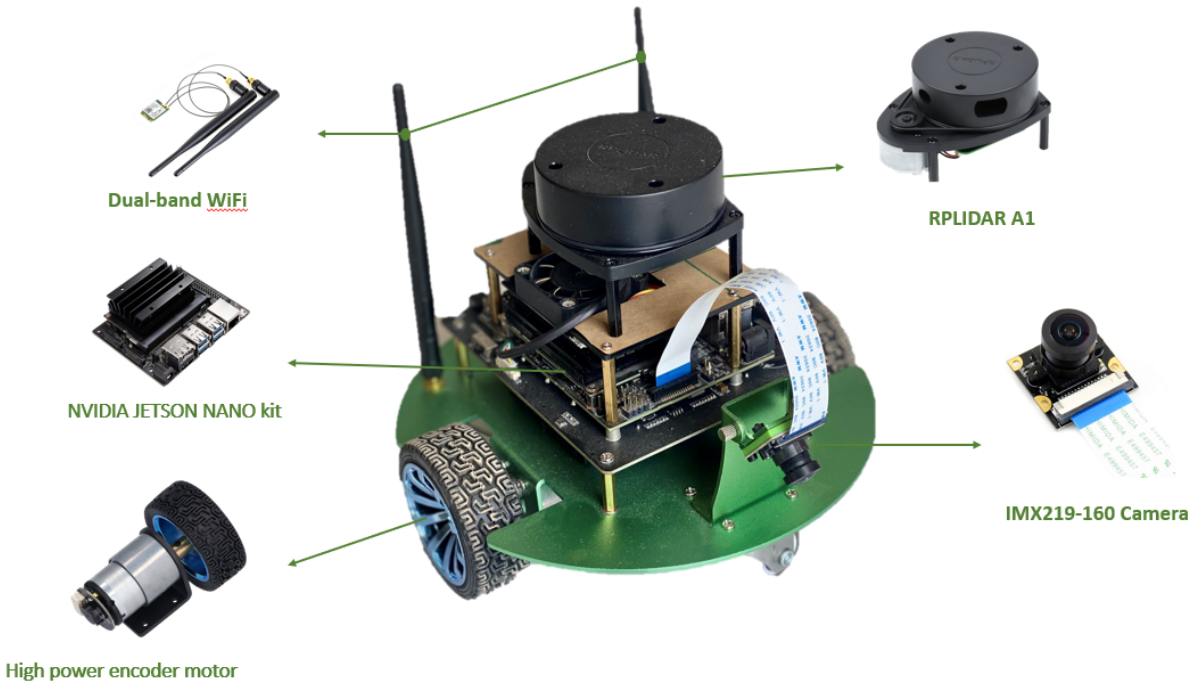

Figure 3.1: JetBot ROS Structure .

3.3 Robot Components

3.3.1 NVIDIA JETSON NANO kit

The NVIDIA Jetson Nano kit is a small, powerful computer designed for AI development and robotics. It includes a Jetson Nano module, which has a quad-core ARM processor and a 128-core NVIDIA GPU. This combination allows it to run multiple neural networks in parallel for applications like image recognition, object detection, and speech processing. The kit also comes with various ports for connecting peripherals such as cameras, displays, and sensors, making it versatile for different projects. It's an ideal platform for beginners and professionals to create AI-powered projects, enabling easy prototyping and development of smart devices and robots.

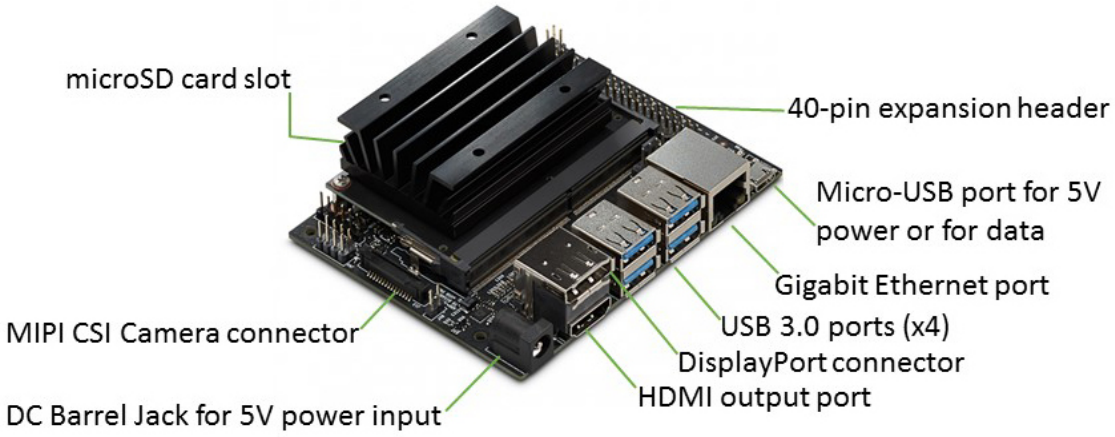

Figure 3.2: Jetson Nano Dev Kit .

Table 3.1: Jetson Nano Dev Kit Parameter

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	16 GB eMMC 5.1
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Camera	12 (3x4 or 4x2) lanes MIPI CSI-2 D-PHY 1.1
Connectivity	Gigabit Ethernet
Display	HDMI 2.0, eDP 1.4, DP 1.2 (two simultaneously)
USB	1x1/2/4 PCIe Gen2
Others	1x USB 3.0, 3x USB 2.0

3.3.2 RpLidar A1

Mobile robots are getting better at their jobs because they can easily use sensors that give them helpful data. This helps them work more efficiently. Even robots at home are becoming more independent because they can use these sensors well. In our project, we're using a Lidar to map the area around us, which is very important for what the robot needs to do. The most important sensor for us is the RP Lidar, specifically the RPLIDAR A1. It's a low-cost 2D laser scanner made by SLAMEC. It can scan all around

the robot in a 12-meter range. Using the data from this scanner helps us make maps, figure out where we are, and understand the surroundings.

Figure 3.3: RpLidar A1 360-degree laser.

```
header:
  seq: 314
  stamp:
    secs: 1717613476
    nsecs:  78522727
  frame_id: "laser"
angle_min: -3.12413907051
angle_max: 3.14159274101
angle_increment: 0.00871450919658
time_increment: 0.000185220851563
scan_time: 0.133173793554
range_min: 0.15000000596
range_max: 12.0
```

Figure 3.4: Key characteristics of RPLIDAR A1.

The RPLIDAR A1 has a small and safe infrared laser that uses low power, around 5mW. It emits quick pulses of light. This laser is safe for people and pets, so it's practical to use in different situations [26].

The RPLIDAR A1 sends out an infrared laser signal, which bounces off objects and comes

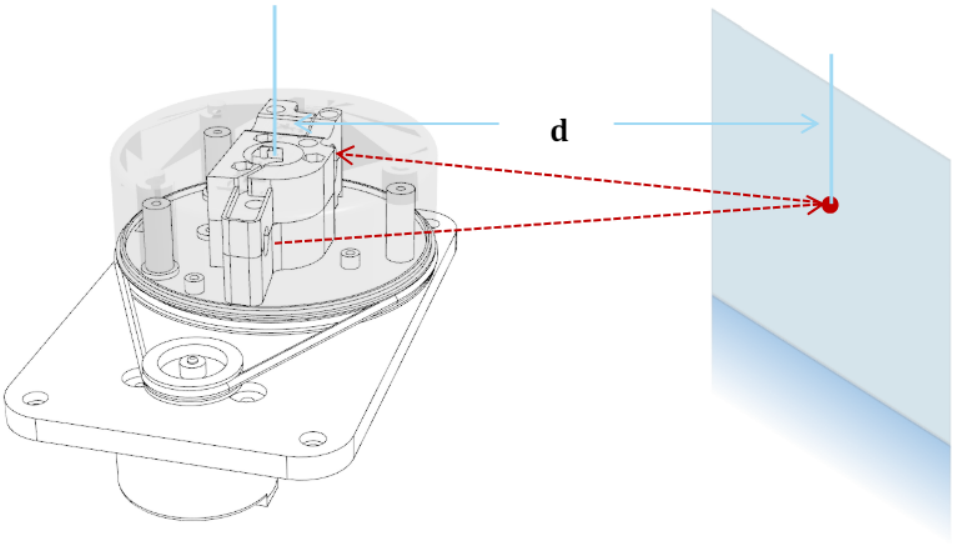

Figure 3.5: Schematic of the RPLIDAR mechanism [26].

back. Then, the RPLIDAR A1's vision system collects this returned signal. The device also has digital processing that deals with the data, like the distance to objects and the angle between the lidar and the object [26].

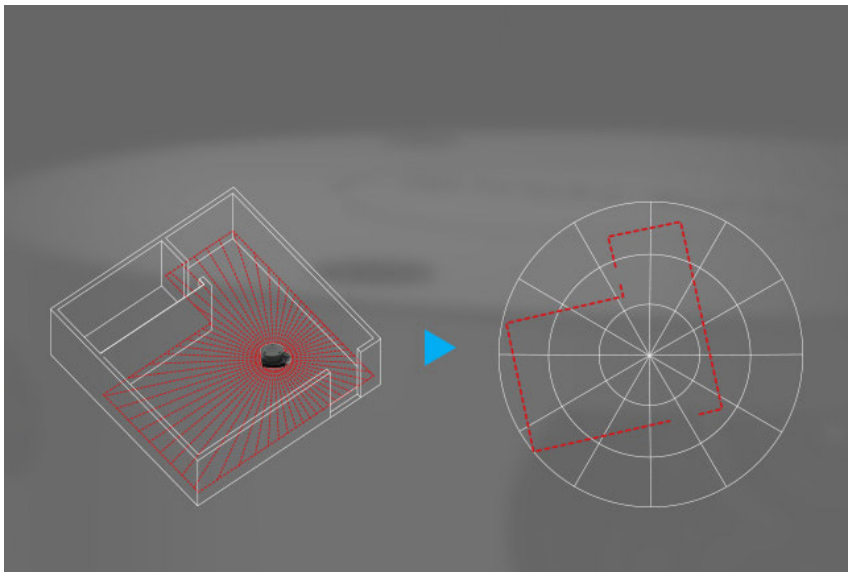

Figure 3.6: Example of RPLIDAR visualization using a frame grabber software [26].

Theoretical Framework of RPLIDAR A1

The RPLIDAR A1 operates on the principle of Light Detection and Ranging (LIDAR) technology. It emits laser pulses and measures the time it takes for these pulses to return after hitting an object (time-of-flight). The distance to the object is calculated using the

speed of light and the measured time. The laser emitter and receiver are mounted on a rotating platform driven by a motor, enabling continuous 360-degree scanning. The angle at which each distance measurement is taken is recorded, forming a set of polar coordinates. This data is then processed to create a 2D point cloud representing the environment. The point cloud can be converted from polar to Cartesian coordinates for use in mapping and navigation.

Distance Calculation (Time-of-Flight):

$$d = \frac{c \cdot t}{2} \tag{3.1}$$

- d is the distance to the object.
- c is the speed of light (approximately 3×10^8 meters per second).
- t is the time-of-flight of the laser pulse.

The factor of 2 accounts for the round-trip travel of the pulse (to the object and back) [27].

Polar to Cartesian Conversion:

$$\begin{aligned} x &= d \cdot \cos(\theta) \\ y &= d \cdot \sin(\theta) \end{aligned} \tag{3.2}$$

- x and y are the Cartesian coordinates.
- d is the distance to the object.
- θ is the angle at which the distance was measured.

3.3.3 High power encoder motor

333 RPM motors can be used in different areas. What interests us is the area of Automation and Robotics. The motor RPM function uses electromagnetic principles to convert electrical energy into rotating motion in a specific way, allowing for revolutions per minute (RPM).

These standard gear motors are incredibly sturdy and feature full metal gears to help you control wheels, gears, or almost anything else that needs to rotate. They have a transmission ratio of 30:1 and operate up to 12 volts, additionally, they have a maximum speed of 333 RPM. Each standard gear motor has a 6 mm diameter D shaft.

Table 3.2: Specifications of the Gear Motor

Specifications	**Details**
Voltage	6 - 15 Volts
Reduction Ratio	30:1
Maximum Torque	4.5 Kg-cm / 12V
Speed	333 RPM @ 12V
Maximum Current	1.2A / 12V
Shaft Size	6mm Diameter x 15mm Long
Motor Size	35mm Diameter x 50mm Long
Gear Motor Model	JGB37-3530 (DC: 12V, 333RPM)

Figure 3.7: High power encoder motor.

3.3.4 IMX219-160 Camera

IMX219-160 camera module is a product from Sony Semiconductor Solutions Corporation. It's known for its good low-light performance and high-quality image output. Many single-board computers and development boards, like the Raspberry Pi, utilize this camera module due to its compatibility and versatility. Also the IMX219 camera is natively supported by the Jetson Nano out of the box.

Table 3.3: Camera Specifications

Specifications	**Details**
Photosensitive Chip	IMX219
Resolution	3280 x 2464 pixels
CMOS Size	1/4 inch
Aperture (F)	2.35
Focal Length	3.15mm
Diagonal Field of View (FOV)	160°
Distortion	$< 14.3\%$
Lens Size	6.5mm x 6.5mm

Figure 3.8: camera IMX219-160 .

3.3.5 IMU MPU9250

The MPU-9250 is a small 9-axis Motion Processing Unit used in devices like smartphones and wearables. It's very compact, measuring just 3x3x1mm, and it uses only 9.3 µA of power. This unit includes a gyroscope, accelerometer, and compass, offering improved performance and range. It helps save battery life and makes it easier to create motion-based features. Additionally, its high precision and low power consumption make it ideal for applications requiring accurate motion tracking, such as fitness tracking, gaming, and augmented reality.[28].

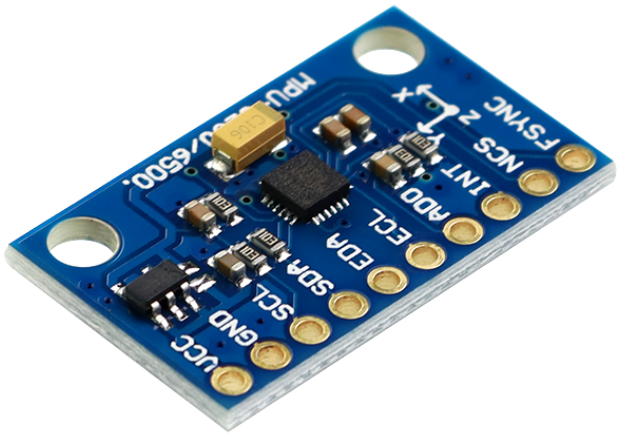

Figure 3.9: IMU MPU9250.

3.3.6 Table of Costs

JetBot ROS AI Kit Materials

Item	**Cost (USD)**
Jetson Nano Developer Kit B01	$150.00
Micro SD Card 64GB	$14.99
5V 4A DC power adapter	$9.99
2-Axis Servo Driver HAT	$12.99
Waveshare IMX219-160 Camera	$25.99
Wireless Dual-band Network Card	$18.99
LiDAR	$129.00
Motor	$15.99
Encoder Motor	$19.99
Battery Pack	$29.99
JetBot Chassis	$24.99
Other accessories (cables, screws, etc.)	$19.99

3.4 Configuring communication between robot and PC

In order for us to implement navigation applications on a robot, the robot must be autonomous. To achieve this autonomy, the robot must not be connected to any external equipment such as a screen. To do this, we need to set up communication between the robot and a computer. This way, we can see the robot's inputs and control it.

There are various methods for this communication process. In our project, we will work on entering the **IP address and hostname**of the robot into the computer, using a virtual machine. We will also perform the reverse process, such that we enter the **IP address and hostname** of the computer into the robot(in our case we connect the robot and the computer in the same WiFi). We will see how this process is done .

First of all, we must know the robot's hostname by using the command :

```
$ hostname
```

We will now review the results.

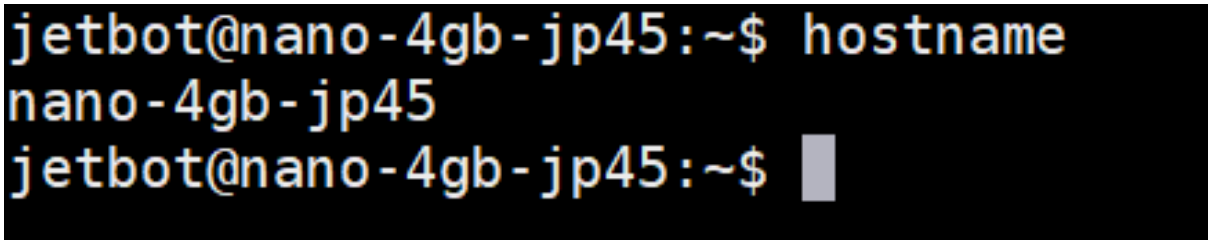

Figure 3.10: hostname of the robot .

To get the IP address of the robot , we input the command

The result obtained is (see Figure 3.11):

and also we can see the IP address in the screen (see Figure 3.12).

We will now input the robot's hostname and IP address into the computer, utilizing the command provided on the computer interface.

```
$ sudo nano /etc/hosts
```

next we will see the result of applying this command (see Figure 3.13).

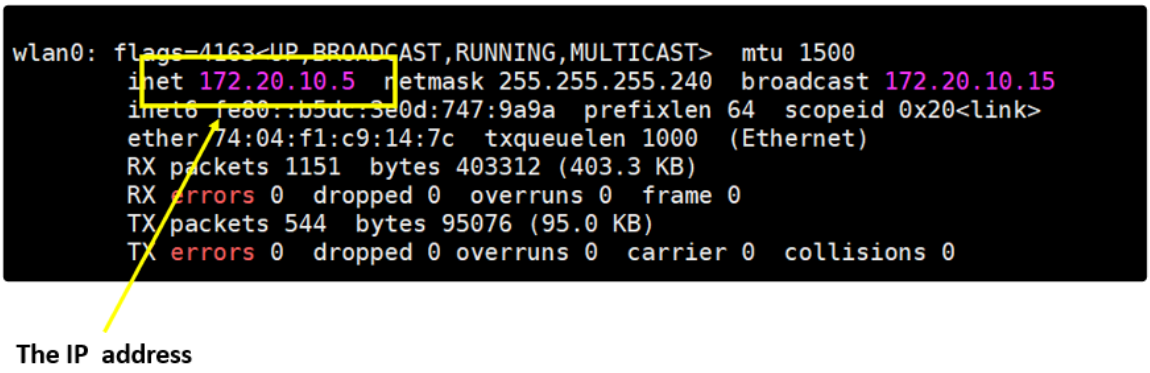

Figure 3.11: IP address of the robot .

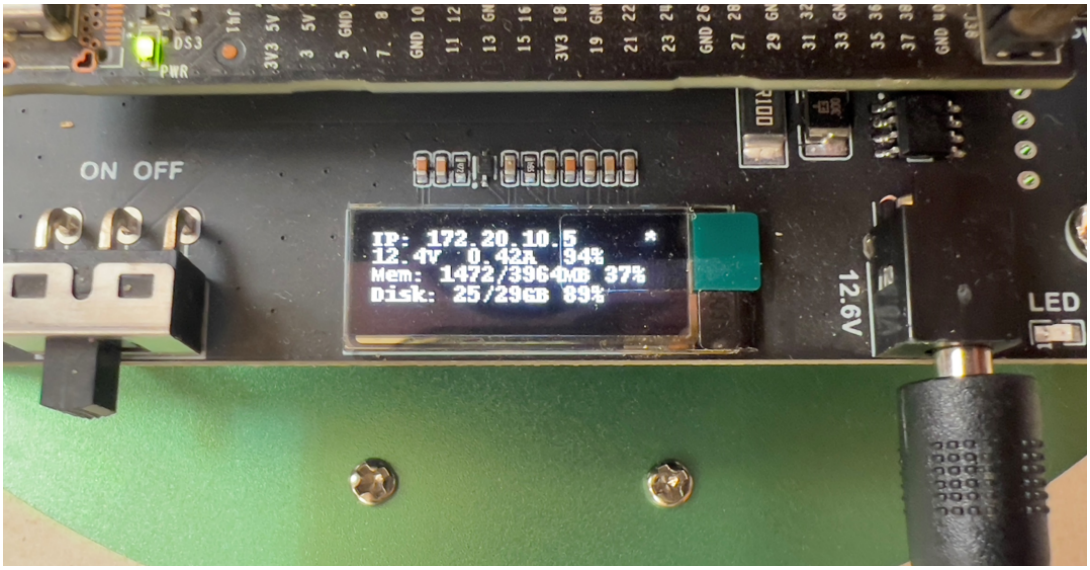

Figure 3.12: IP address of the robot .

3.5 Robot Calibration

Robot calibration is essential to ensure a robot performs its tasks accurately. This process involves adjusting sensor settings and control software parameters to improve precision and overall performance. Calibration focuses on fine-tuning sensor alignment, sensitivity, and error correction to optimize tasks like navigation and object manipulation. By refining these settings, robots can operate reliably in different environments, enhancing their efficiency and functionality.

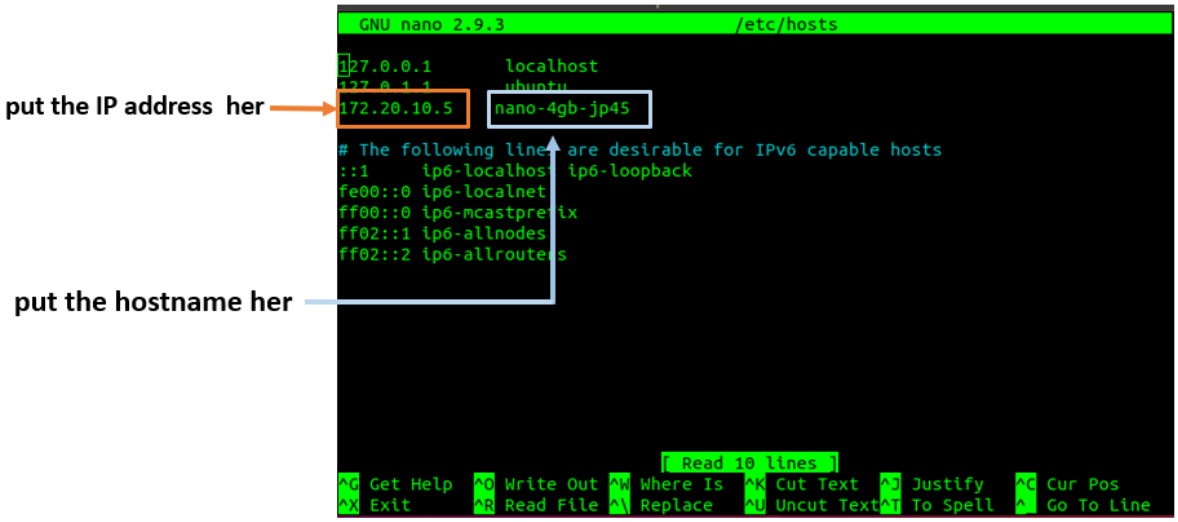

Figure 3.13: the modification of the hostname.

linear calibration

The robot was calibrated to move accurately over a distance of one meter. One meter was marked on the ground and the robot was tested to see if it moved exactly that distance .

Figure 3.14: linear trajectory .

next step is moving to the computer and opn the vertual machine ,then throw the terminal and define the command:

```
$ roslaunch jetbot_pro calibrate_linear
```

it was run to start the calibration process. after that we throw an auther terminal and define the command:

```
$  rosrun aqt_reconfigure aqt_reconfigure
```

it was run to open the RQT Reconfigure tool. In RQT, settings were adjusted to optimize the robot's movement.

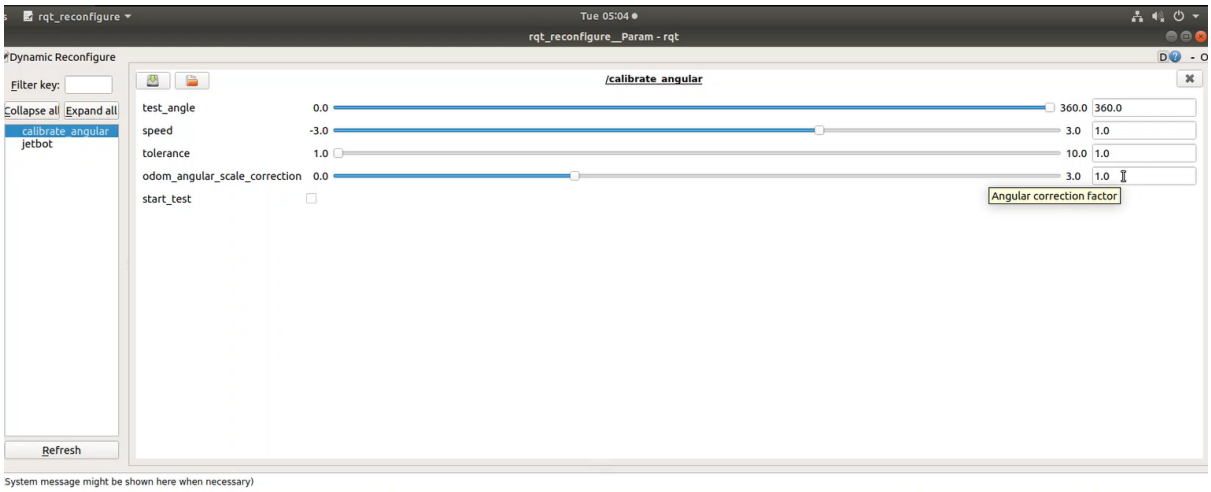

Figure 3.15: rqt widow

The robot was tested multiple times, and the distance it moved was measured and adjusted until it moved exactly one meter. The final settings were saved and checked to ensure the robot always moved the correct distance.

calibrate angular

To calibrate the robot's angular movement, place the phone on top of the robot and open a compass app, ensuring the angle reads 0 (see Figure 3.16).

Then, go to the computer, open a terminal, and type:

```
$ roslaunch jetbot_pro calibrate_angular
```

pressing Enter to start the calibration process. Open another terminal window and type:

Figure 3.16: angle reads 0.

```
$ rosrun aqt_reconfigure aqt_reconfigure
```

pressing Enter to launch a tool for adjusting the robot's angular settings. Use this tool to change the angular value while checking the compass app on your phone to maintain the angle at 0.

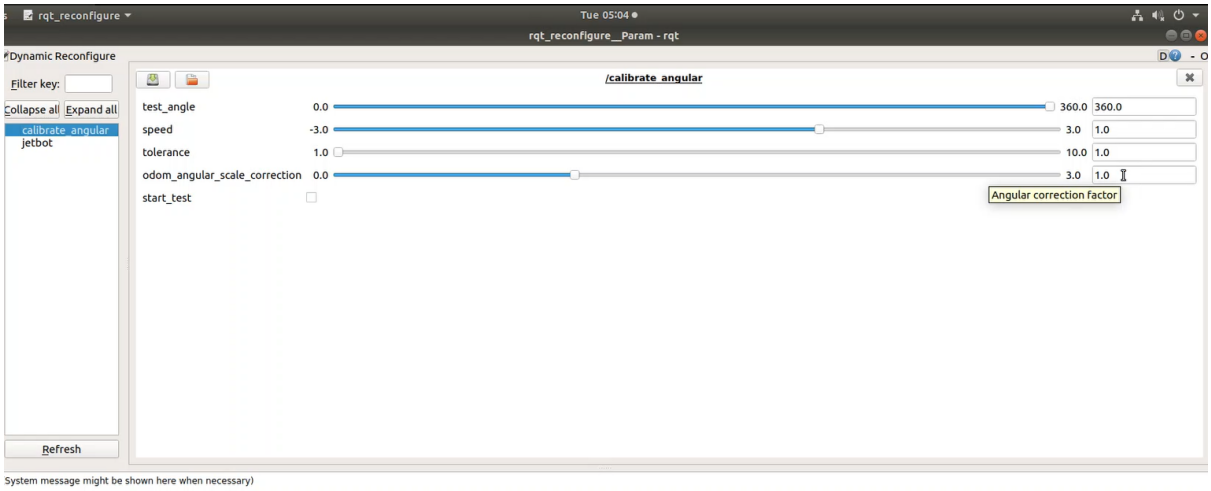

Figure 3.17: rqt window.

Continue adjusting and checking until the robot's angle is perfectly calibrated to 0.

3.6 LiDar Node activation

To start, connect the RPLIDAR A1 to the USB port of the NVIDIA Jetson Nano using a USB adapter and the provided communication cable. After connecting, open a terminal window. Then, to ensure proper communication, we need to change the permission settings by executing the following command:

```
$  sudo chmod 666 /dev/ttyUSB0
```

Now we can read and write data with the LiDAR device using the USB port. To verify the connection, we can check the device list with the following command:

```
$  ls -l /dev | grep ttyUSB
```

Now, we navigate to the directory:

```
$ cd ~/catkin_ws/src
```

and download the RPLIDAR ROS package. This package contains the necessary drivers and tools to interface with the RPLIDAR A1. We do this by entering the following command:

```
$  git clone https://github.com/robopeak/rplidar_ros.git
```

This command fetches the RPLIDAR ROS package from GitHub and places it in our workspace. Once downloaded, we can build the package and integrate it into our ROS environment, allowing our Jetson Nano to communicate with the RPLIDAR A1 and process the data it collects.

Next, we head back to the main workspace directory by running:

```
$  cd ..
```

Once we are back in the main **catkin_ws** directory, we compile our catkin workspace using the command:

```
$ catkin_make
```

This command will build all the packages in the workspace, including the RPLIDAR ROS package we just downloaded. Compiling the workspace ensures that all dependencies are resolved and the ROS nodes can run correctly. After the compilation is complete, our system will be ready to interface with the RPLIDAR A1 and process the LiDAR data.

After compiling the workspace, we need to source the environment with our current terminal to make the changes take effect. We do this by running the following command:

```
$ source devel/setup.bash
```

Sourcing the setup file ensures that the ROS environment variables are correctly set for our terminal session. This allows ROS commands and nodes to be executed without any issues. After sourcing the setup file, we can leave the terminal open without closing it, and ROS commands will work as expected until the terminal session is closed.

Finally, we need to open a new Terminal window and start the ROS core using the command:

```
$ roscore
```

Leaving this Terminal window open, we return to the previous Terminal where we sourced the environment and run the following command:

```
$ roslaunch rplidar_ros view_rplidar.launch
```

After launching the RPLIDAR ROS node, an instance of RViz will open automatically. RViz is a visualization tool that displays the LiDAR data in a map format, allowing us to see the surroundings detected by the RPLIDAR A1. The map will show the environment around the LiDAR sensor, including obstacles, walls, and other objects. This visual representation helps us understand the LiDAR's perception of the surroundings ,Below is a representation of how the map may look in RViz:

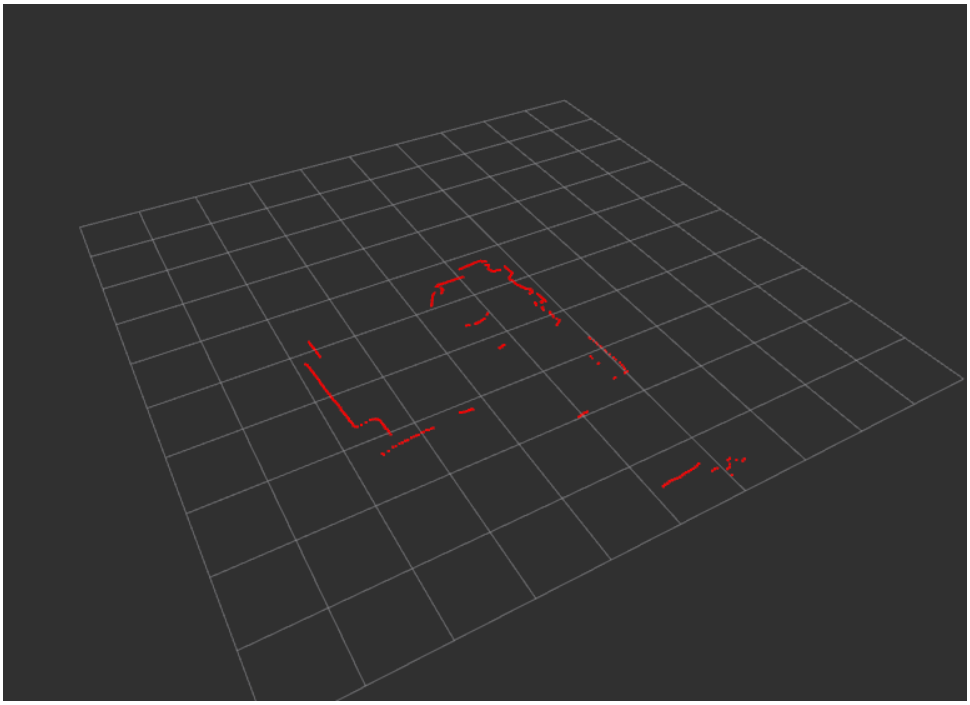

Figure 3.18: Visualize LiDAR sensor data using RVIZ.

3.7 Goystick application

In our project We applied the navigation with joystick because joystick in mobile robots allows for intuitive control over their movement. By manipulating the joystick, operators can steer the robot in various directions and adjust its speed, facilitating precise navigation in dynamic environments.

Figure 3.19: goystick.

3.8 First Application Hector SLAM

To integrate mapping capabilities into our system, we utilize the Hector-SLAM package. This tool enables the creation of visual maps that represent the environment scanned by

the LiDAR. Hector-SLAM utilizes LiDAR data and employs a scan matching technique to build accurate maps.

To install the Hector SLAM package:

Navigate to the **catkin_ws/src** directory using the command:

```
$ cd ~/catkin_ws/src
```

Clone the Hector SLAM source files from its GitHub repository:

```
$  git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git
```

This command downloads the necessary files into your ROS workspace for further installation and configuration.

Since we lack a "base_footprint" frame, we must modify two files to designate our "base_link" frame as the odometry frame. Initially, we need to adjust Hector SLAM's launch file. Locate it using the following commands:

Navigate to the directory where Hector SLAM's launch file is located:

```
$ cd ~/catkin_ws/src/hector_slam/hector_mapping/launch
```

Edit the launch file using a text editor:

```
$  nano mapping_default.launch
```

Update the following lines:

```
<arg name="base frame" default="base link"/>
<arg name="odom frame" default="base link"/>
```

Replace the commented-out static transform node:

```
<!-- <node pkg="tf" type="static_transform_publisher" name="map_nav"
args="0 0 0 0 0 0 map nav 100"/> -->
```

With:

```
<node pkg="tf" type="static_transform_publisher" name="base_to_laser"
args="0 0 0 0 0 0 base_link laser 100"/>
```

Move to the Hector SLAM launch folder:

```
$ cd ~/catkin_ws/src/hector_slam/hector_slam_launch/launch
```

Open the tutorial launch file:

```
$  sudo nano tutorial.launch
```

Modify the parameter for simulated time:

```
<param name="/use_sim_time" value="true"/>
```

Change it to:

```
<param name="/use_sim_time" value="false"/>
```

Return to the main ROS workspace directory:

```
$ cd ~/catkin_ws
```

Source the environment setup script:

```
$ source devel/setup.bash
```

Launch the Hector SLAM for mapping:

```
$ roslaunch jetbot_pro slam.launch map_type:=hector
```

Leave this terminal open and to view the map run this command:

```
$ roslaunch jetbot_pro view_slam.launch
```

we see that when launching Hector SLAM, it's important to consider the robot's speed during operation. If the robot moves at high speeds or makes sudden turns, the generated map may become distorted or inaccurate, a phenomenon often referred to as "map crush." This occurs because the SLAM algorithm struggles to match LiDAR scans effectively under fast motion conditions, leading to inconsistencies in the mapped environment. Conversely, moving the robot at lower speeds or with smoother motions ensures that Hector SLAM can accurately align consecutive LiDAR scans and maintain the integrity of the map. This approach results in more reliable and detailed maps of the environment, which are crucial for effective navigation and localization in robotics applications.

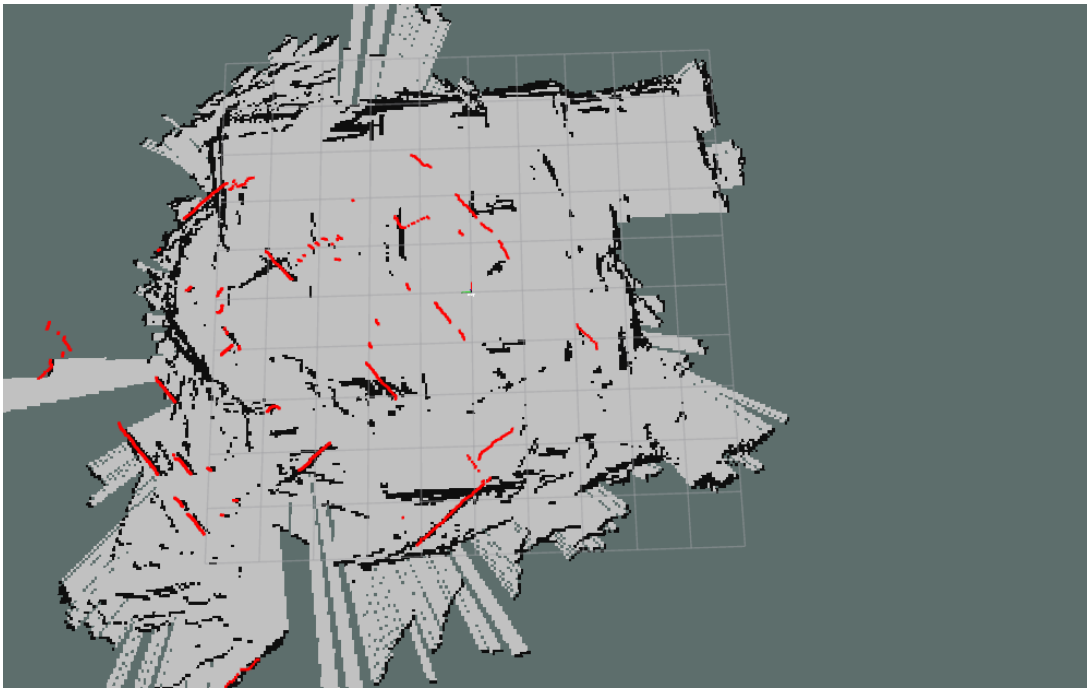

Figure 3.20: moving the robot at HIGH speed.

To mitigate this issue, we use techniques like GMapping. GMapping is a popular SLAM algorithm that incorporates additional sensor data and advanced algorithms to handle high-speed movements more effectively. By integrating data from odometry and other sensors, GMapping can compensate for the challenges posed by fast robot movements, resulting in more robust and accurate mapping outcomes. This approach ensures that even during high-speed operations, the generated maps remain reliable and suitable for navigation and localization tasks in robotics.

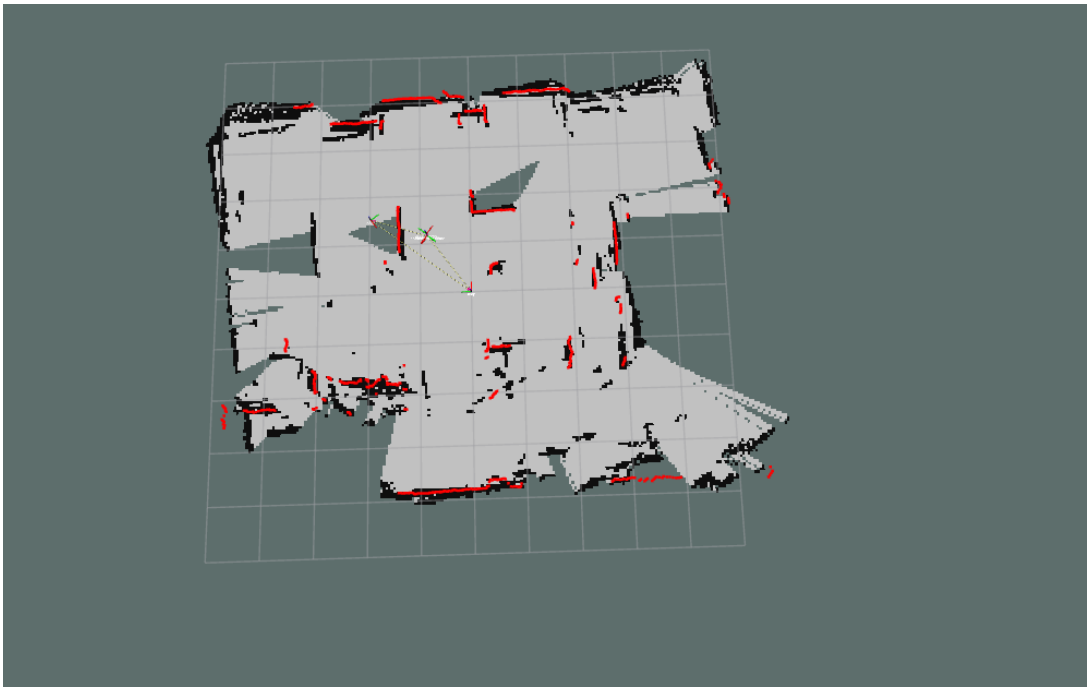

Figure 3.21: moving the robot at LOW speed.

Figure 3.22: The Real Environment.

3.9 The Second Application GMapping

When we use GMapping with loop closure in our process, we employ a technique that improves the accuracy of our mapping. We start by launching:

```
$ roslaunch jetbot_pro slam.launch
```

in a terminal on our computer. This command starts GMapping, which uses the robot's sensors like cameras, lidar, and odometry to create a detailed map of its surroundings. Loop closure is an important part of this process. It helps correct errors by recognizing previously visited locations in the robot's path, ensuring the map remains accurate and coherent as the robot moves. After starting the mapping, we use :

```
$ roslaunch jetbot_pro view_slam.launch
```

in another terminal to visualize the map in real-time. This tool shows us what the robot sees, including obstacles and structures, and demonstrates how loop closure maintains map accuracy throughout the mapping process. Integrating GMapping with loop closure ensures our robot builds and maintains a reliable map essential for navigating complex environments effectively.

Figure 3.23: Real-time Mapping with GMapping and Loop Closure.

3.10 Conclusion

In Chapter 3, we implemented and evaluated Hector SLAM and GMapping algorithms in ROS for autonomous navigation of mobile robots in indoor environments. Both algorithms successfully generated accurate maps, with Hector SLAM excelling in low-noise

data scenarios and GMapping in more complex environments. Our results confirm their effectiveness, highlighting the importance of selecting the right SLAM algorithm based on operational needs. This work establishes a strong foundation for future advancements in robotic navigation that .In the next chapter, we will conduct practical experiments with the robot, focusing specifically on its navigation abilities. This involves programming the robot to move from point A to point B, avoiding obstacles on its own while simultaneously creating a map of its surroundings environment.

CHAPTER 4

AUTONOMOUS NAVIGATION

4.1 Introduction

In this chapter, we will start to implement robot navigation so that the robot is in its first position and we move it by specifying a path for it, that is, it moves from point A to point B (the one responsible for this process is the PID CONTROLLER) and building the map simultaneously In order for the robot to perform this navigation professionally, we studied how the robot builds its map using lidar technology and how this technology deals with data to draw a two-dimensional map. In addition, we studied how the robot determines its location correctly using the AMCL algorithm and writing all the equations related to it.Finally, we explained all the results we obtained by performing robot navigation. In our thesis, we focus on building and studying the map . the PID controller, is used to guide the robot to navigate accurately and efficiently to the target position (Move from point A to point B by selecting goals) , In fact, to do this challenge task, , we collaborate with colleagues working under the same supervisor on a project titled "AI-Driven Mobile Robot Navigation:Precision in Road Following and Objective Pursuit." Our colleagues, Assia Benkouider and Manel Benaidja, have already completed the part of the project .

4.2 Mapping

In this thesis, we rely on the gmapping algorithm to create detailed maps essential for navigating robots using SLAM technology. This method uses data from a LiDAR sensor to build 2D maps of the environment where the robot operates. It handles frequent updates from the sensor by processing approximate data quickly. The algorithm converts sensor data into a point cloud.

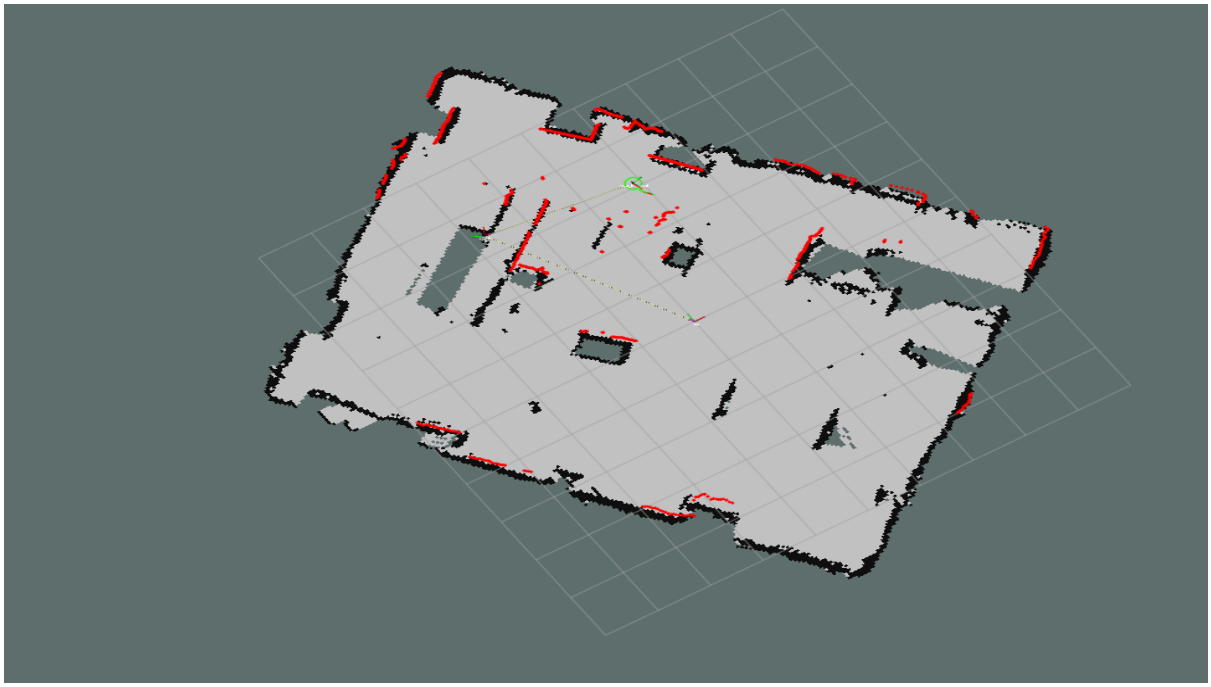

Figure 4.1: 2D map of the environment .

Navigation is challenging because the mobile robot must find a valid global path from its starting point on the map to its destination and also react to its local environment. For example, it must handle uncharted obstacles or navigate through narrow gaps. Therefore, navigation involves two planners: a local planner and a global planner.

The global route is calculated by the global_planner of the navigation_stack library in ROS using the Dijkstra algorithm, which finds the shortest path from point A to point B in a graph. The process is straightforward: start with the entire graph, set one node as the starting point, and set the distance to all other nodes as ∞. The algorithm uses two queues: one for visited nodes (initially empty) and one for all other nodes except the starting node. As long as the queue of unvisited nodes isn't empty, the algorithm selects the node with the minimum distance, marks it as visited, and checks for new shortest

paths. If a shorter path is found, it updates the shortest path value.

```
dist[s] ← 0                                   (distance to source vertex is
   zero)
for all v ∈ V − {s}
     do dist[v] ← ∞                (set all other distances to infinity)
S ← ∅                              (S, the set of visited vertices is initially
   empty)
Q ← V                                    (Q, the queue initially contains all
   vertices)
while Q ≠ ∅                               (while the queue is not empty)
     do u ← mindistance(Q, dist)       (select the element of Q with the
        min. distance)
        S ← S ∪ {u}          (add u to list of visited vertices)
        for all v ∈ neighbors[u]
             do if dist[v] > dist[u] + w(u, v)    (if new shortest path
                found)
                then dist[v] ← dist[u] + w(u, v) (set new value of
                   shortest path)
                     if desired, add traceback code


return dist
```

The Dijkstra algorithm is suitable for graphs without negative edges. In the context of a 2D grid map generated by the SLAM algorithm, each pixel represents a node in the graph. Transitions between these pixels have consistent values, except those leading to obstacles, which have no transitions. The starting point is considered the initial node, and the destination serves as the endpoint. This approach allows for finding the shortest path for the mobile robot across the entire map.

4.3 Localization

To achieve accurate navigation, a mobile robot must be localized correctly within the map from the beginning and throughout its journey. Initially, the robot's odometry is calculated based on velocity commands such as speed and steering angle. However, since the robot starts from an estimated point, its odometry is not entirely accurate. Therefore,

a localization algorithm is essential to precisely determine the robot's position within the map. This is crucial for optimizing the construction of an accurate route by the local planner.

For this purpose, the adaptive Monte Carlo localization (amcl) algorithm is employed. Implemented as a particle filter in ROS, amcl uses a known map generated by a laser sensor, typically facilitated by libraries like hector_slam. The goal is to estimate the robot's pose within the map based on its movement and sensor data. The amcl algorithm begins with an initial belief of the robot's pose, often the estimated starting point. At each time-step, it updates the robot's state, addressing the Bayesian filtering problem to construct the posterior density $p(x_k \mid Z^k)$ of the current state conditioned on all measurements, where x represents the state vector[29].

$$p(x_k \mid Z^k) \qquad x = [x, y, \theta]^T \tag{4.1}$$

In the specific application of the Monte Carlo filter, density is represented using a collection of N random particles[29]:

$$S_k = \{s_k^i; i = 1..N\} \tag{4.2}$$

For accurate localization, it's essential to recursively compute the density at each time step, which involves two phases: prediction and update.

During the prediction phase, a motion model is employed to estimate the robot's current position. The state x at time step k depends solely on the state at the previous time step $k-1$, given a known control input u_{k-1} . The motion model is defined as a conditional density[29]:

$$p(x_k \mid x_{k-1,u_{k-1}}) \tag{4.3}$$

In Bayesian filtering, the predictive density over the state vector x_k is computed through integration:

$$p(x_k \mid Z^{k-1}) = \int p(x_k \mid x_{k-1,u_{k-1}})p(x_{k-1} \mid Z^{k-1})dx_{k-1} \tag{4.4}$$

In Monte Carlo localization, the process begins with the set of particles S_{k-1} from the previous time-step, where each particle s^i_{k-1} undergoes the motion model to sample from the density $p(x_k \mid s^i_{k-1}, u_{k-1})$. This generates a new sample s'_{i-k} for each particle s_{i-k} .

During the update phase, a measurement model integrates sensor information to derive the density function described in equation (4.1). Each measurement z_k is conditionally independent of previous measurements, and the measurement model provides a likelihood that the robot observes z_k at the location x_k[29].

$$p(z_k \mid x_k) \tag{4.5}$$

The posterior density over x_k is now obtained using Bayes' theorem.

$$p = (x_k \mid Z^k) = \frac{p(z_k \mid x_k)p(x_{k-1} \mid Z^{k-1})}{p(z_{k-1} \mid Z^{k-1})} \tag{4.6}$$

The Monte Carlo localization algorithm considers the measurement z_k and assigns a weight m^k_i to each sample s'_k created in the first phase:

$$m^k_i = p(z_k \mid s'^i_k) \tag{4.7}$$

Then, for each $j = 1 \ldots N$, a sample s^k_j is drawn from $\{s'^i{}_k, m^i_k\}$. The entire algorithm is computed recursively. It starts at $k = 0$ with random samples $S_0 = \{s^i_0\}$ rom the prior $p(x_0)$.

4.4 Results and discussion

In this application, we will see the steps for building a map while the mobile robot performs navigation simultaneously.

To start the navigation, we use the following command:

```
$ roslaunch jetbot_pro slam_nav.launch
```

We can view the map in RViz by using this command:

```
$ roslaunch view_nav.launch
```

This setup allows us to monitor the robot's navigation and mapping in real-time, providing a clear visual representation of the environment and the robot's path.

This figure (see Figure 4.2) represents the initial map before the robot starts navigation.

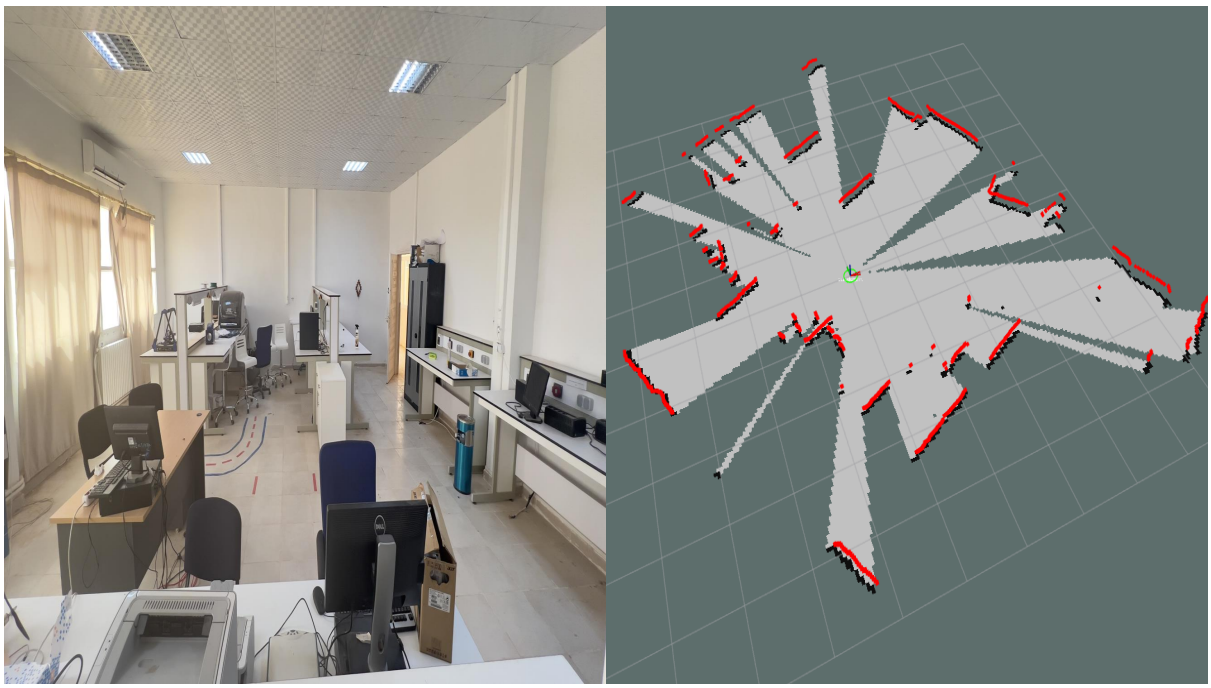

Figure 4.2: the initial map.

With the initial map ready, we can begin navigation. We select a goal using the 2D Nav Goal tool (see Figure 4.3). The goal must stay within the map's edges. By defining the goal, we set a path for the robot to follow from point A to point B (see Figure 4.4 and Figure 4.5), while simultaneously building the map.

This process ensures the robot can navigate accurately and update the map in real-time, enhancing its understanding of the environment.

The red line, or direct path (see Figure 4.6), represents the route the robot navigates. In this section, we will explain the role of the PID controller. Without a PID controller,

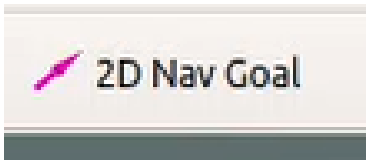

Figure 4.3: 2D Nav Goal tool.

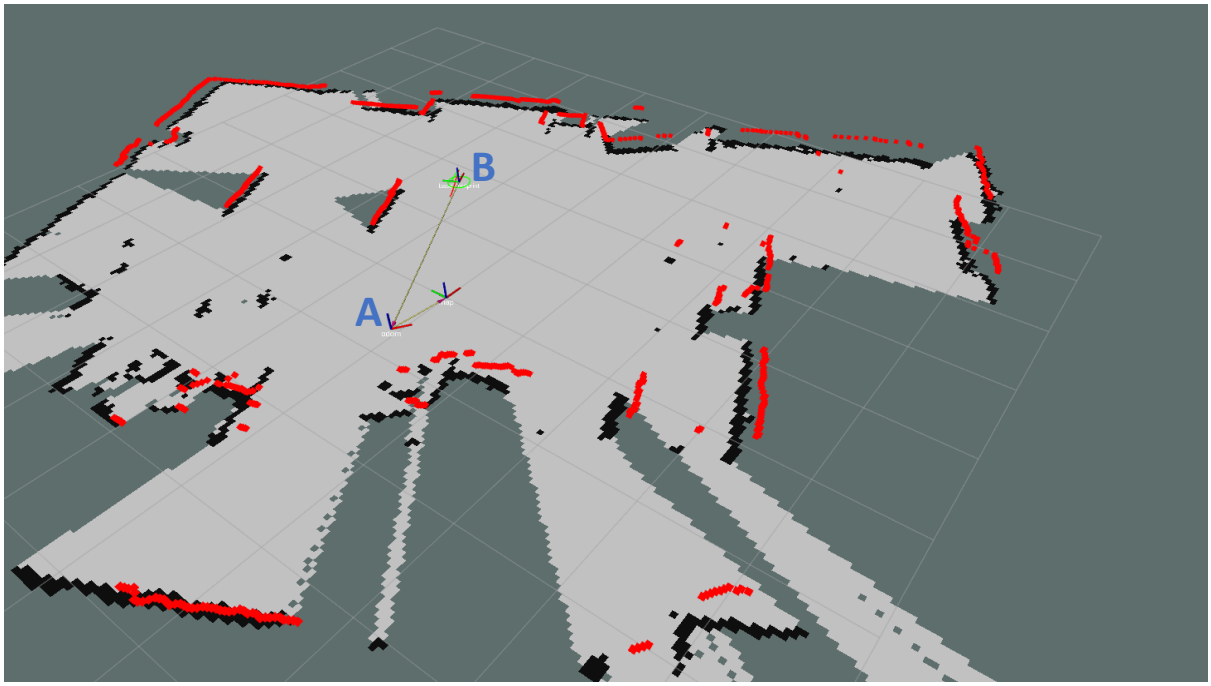

Figure 4.4: First path case .

the robot moves at various angles, making navigation inaccurate. By using the PID controller, we maintain a zero angle, ensuring the robot follows the intended path precisely. This enhances the robot's ability to stay on course and accurately map its environment.

To test the robustness of our control strategy, we conducted an experiment with a different path that included obstacles (see Figure 4.7). The robot successfully navigated around dynamic obstacles, demonstrating the effectiveness of our approach in maintaining a clear path while continuously mapping its environment. This experiment highlights the ability of our system to adapt to changing conditions in real time, ensuring reliable performance even in complex scenarios.

The results show that both motors closely track their target values when employing the PID controller. This was observed in two scenarios: one path contained obstacles, while

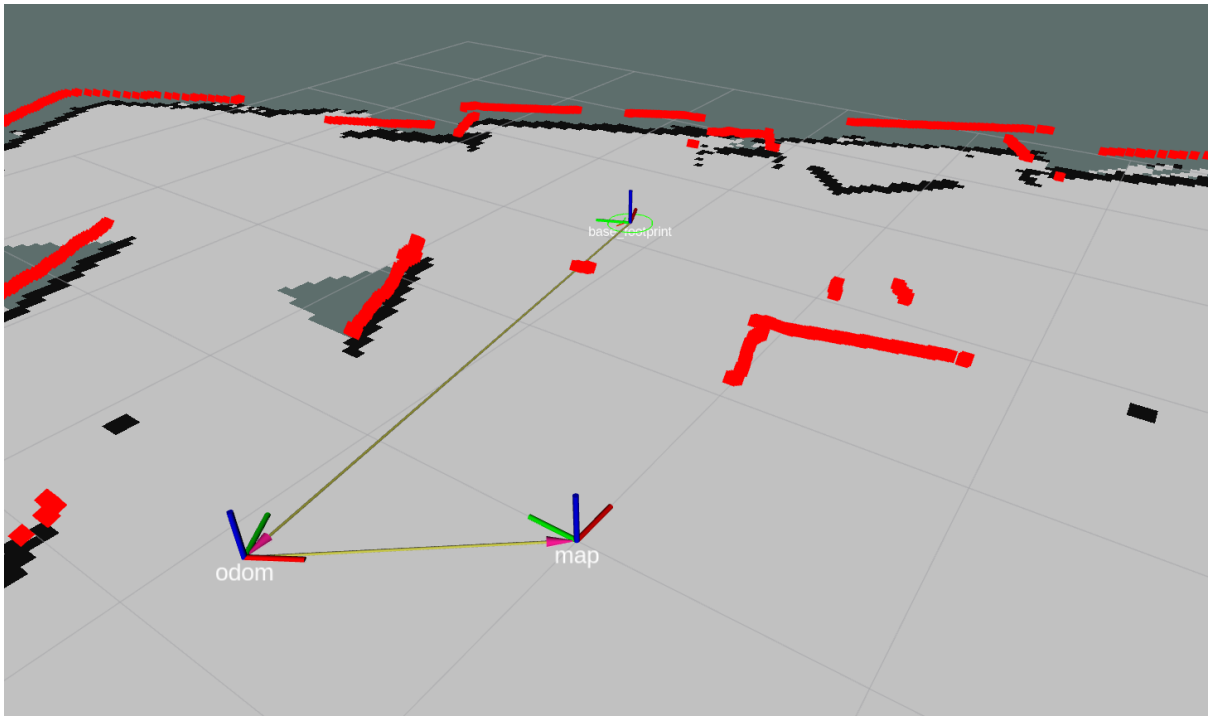

Figure 4.5: First path case (zoom).

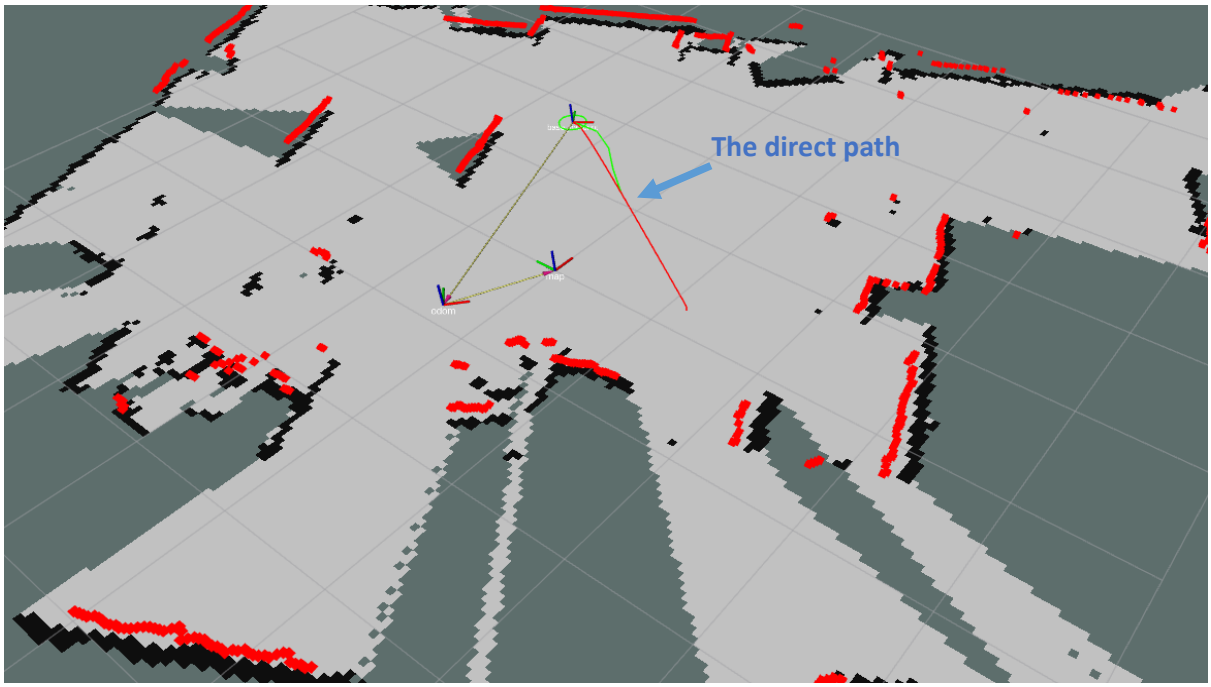

Figure 4.6: building the 2-D map and navigation.

the other did not. The PID controller effectively regulated both motors, ensuring they maintained close alignment with their respective reference points throughout the paths. This demonstrates the controller's capability to navigate complex environments while adhering to desired trajectories, regardless of the presence of obstacles(see Figure 4.8 and

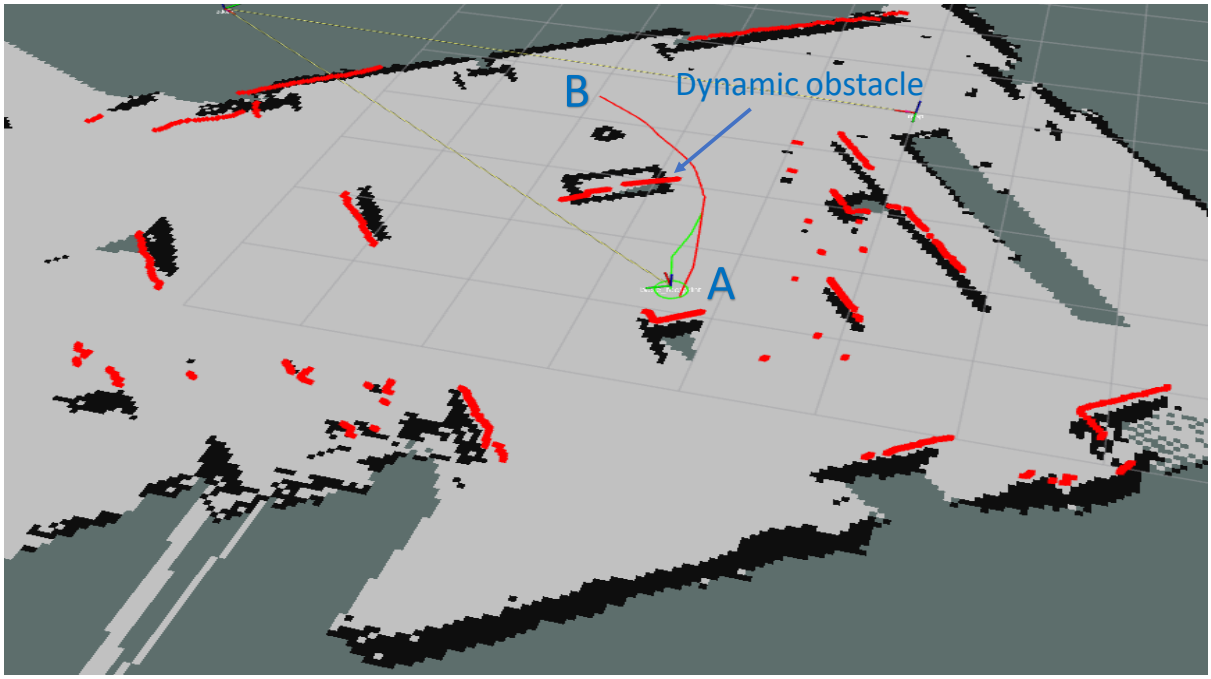

Figure 4.7: Autonomous navigation of robot from point A to B .

Figure 4.9).

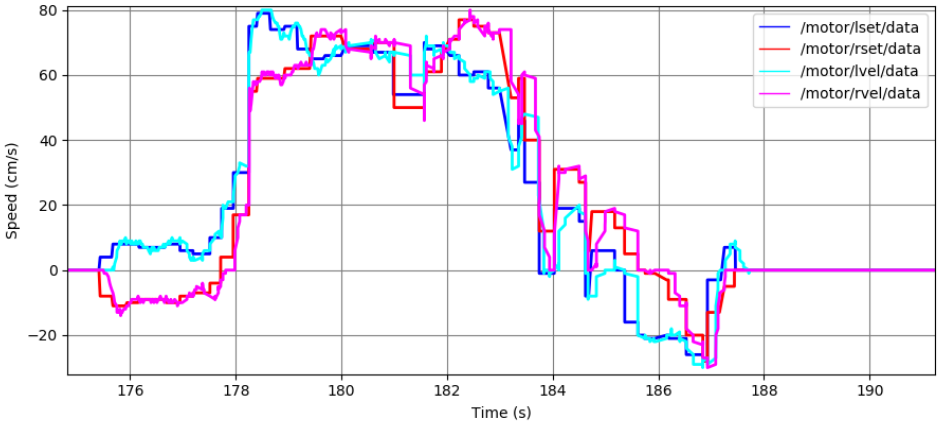

Figure 4.8: Different experimental responses of two motors using PID controller (with obstacles)

We continue this process iteratively until we generate a comprehensive map that covers the entire surveyed environment. This iterative approach ensures that every area within the studied environment is thoroughly mapped and included in the final output. By systematically repeating this process, we achieve a detailed and complete representation of the entire environment, capturing all relevant spatial information. This methodical approach guarantees that no part of the environment is overlooked, leading to a compre-

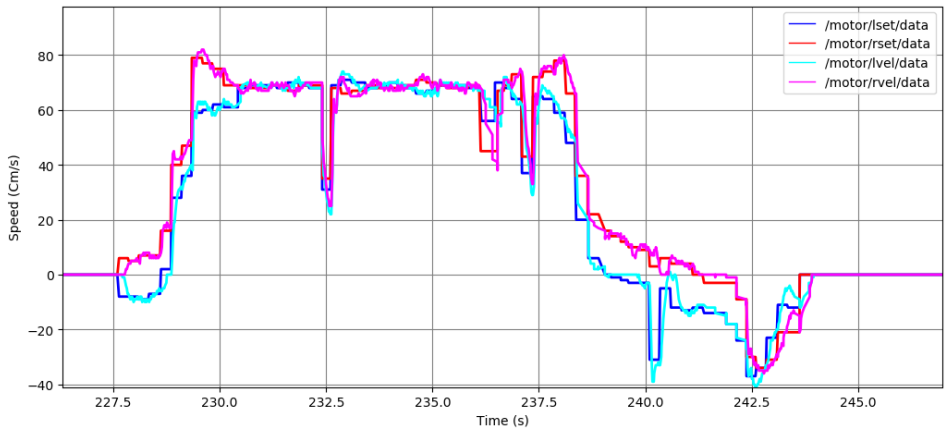

Figure 4.9: Different experimental responses of two motors using PID controller (without obstacles)

hensive and accurate final map.

4.5 Conclusion

In conclusion, this chapter has outlined our approach to implementing robot navigation focusing on map building and path planning using advanced technologies such as LIDAR and algorithms like AMCL. Through collaborative efforts with our colleagues, we have integrated PID controllers to ensure precise and efficient navigation From point A to point B. Our exploration into AI-driven mobile robot navigation underscores the importance of accuracy in path following and objective pursuit. The results presented highlight the successful integration of these technologies and algorithms, paving the way for further advancements in autonomous robotics systems.

GENERAL CONCLUSION

This master's thesis focuses on utilizing SLAM for the navigation and mapping of mobile robots.

In the first chapter, we look at autonomous robots and explore the different types they come in. We also discuss various navigation systems that help robots move on their own, noting that better systems usually make robots work better. Furthermore, we explain how the Robot Operating System (ROS) helps in developing robots. ROS gives us many tools and ways to communicate, which make it easier to create and control complex robot behaviors.

Then, in the second chapter we delve into how mobile robots Achieve simultaneous localization and mapping (SLAM) to understand their surroundings and pinpoint their own position.we examine the mathematical model for mobile robots, explore the challenges associated with SLAM, and look into LiDAR-based SLAM algorithms such as Hector SLAM, Gmapping, and pre-mapping, known for their accuracy. We also emphasize the importance of loop closure in SLAM systems and provide an overview of SLAM-based navigation systems covering localization, path planning, navigation, motion planning, and mapping.

After that, we detailed the different parts of the robot, describing their specific features and functions. Following this, we explored how communication protocols between the robot and computer could be optimized to increase the robot's independence. We also fine-tuned the robot to guarantee accuracy and effectiveness in task execution. Mov-

ing forward, we introduced and applied algorithms like Gmapping and Hector SLAM, generating maps within a virtual environment. Additionally, we discussed the use of joystick-based control for directing the robot during navigation. These efforts collectively laid the groundwork for effective navigation capabilities.

Finally,our focus was on implementing robot navigation, starting with positioning the robot and guiding it along specified paths—from point A to point B—utilizing the PID controller. Concurrently, we explored the process of map construction using LiDAR technology, examining how data is processed to create detailed two-dimensional maps. Additionally, we delved into the AMCL algorithm, studying how the robot accurately determines its location by analyzing the associated equations. Finally, we presented and discussed the results achieved through our robot navigation experiments.

BIBLIOGRAPHY

[1] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. Introduction to autonomous mobile robots. MIT press, 2011.

[2] Weihua Yang. Autonomous robots research advances. Nova Publishers, 2008.

[3] Danilo Gervasio. Design and development of an autonomous mobile robot (amr) based on a ros controller, October 17 2022.

[4] Faiyaz Ahmed, JC Mohanta, Anupam Keshari, and Pankaj Singh Yadav. Recent advances in unmanned aerial vehicles: a review. Arabian Journal for Science and Engineering, 47(7):7963–7984, 2022.

[5] Yu-Hsien Lin, Shao-Yu Chen, and Chia-Hung Tsou. Development of an image processing module for autonomous underwater vehicles through integration of visual recognition with stereoscopic image reconstruction. Journal of Marine Science and Engineering, 7(4):107, 2019.

[6] BOUSIF Fares. Conception, réalisation et implémentation d'un robot mobile dans un environnement de travail, 2020.

[7] Adebayo Segun Adewumi Azeez Ibraheem Abiodun. A review of global navigation satellite systems (gnss) and its applications. International Journal of Scientific Engineering Research, pages 3–4, 2021.

[8] Grant Maloy Smith. What is an inertial navigation system? pages 3–4, 2021.

[9] Morgan Quigley, Brian Gerkey, and William D Smart. Programming Robots with ROS: a practical introduction to the Robot Operating System. " O'Reilly Media, Inc.", 2015.

[10] Bashar Alsadik and Samer Karam. The simultaneous localization and mapping (slam)-an overview. Journal of Applied Science and Technology Trends, 2(02):147–158, 2021.

[11] Miguel Bolaños José Fallas Kevin Trejos, Laura Rincón and Leonardo Marín. 2d slam algorithms characterization, calibration, and comparison considering pose error, map accuracy as well as cpu and memory usage. pages 2–3, 2022.

[12] Muhammad Razmi Razali, Ahmad Athif Mohd Faudzi, and Abu Ubaidah Shamsudin. Visual simultaneous localization and mapping: a review. PERINTIS eJournal, 12(1), 2022.

[13] Muhammad Umar Diginsa, Noraimi Shafie, Nazir Yusuf, and Usman Suleiman Sabi'U. Issues and challenges of simultaneous localization and mapping (slam) technology in autonomous robot. International Journal of Innovative Computing, 13(2):59–63, 2023.

[14] Weifeng Chen, Guangtao Shang, Aihong Ji, Chengjun Zhou, Xiyang Wang, Chonghui Xu, Zhenxiong Li, and Kai Hu. An overview on visual slam: From tradition to semantic. Remote Sensing, 14(13):3010, 2022.

[15] K Wang. Implementation of odometry with ekf for localization of hector slam method, 2016.

[16] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In 2011 IEEE international symposium on safety, security, and rescue robotics, pages 155–160. IEEE, 2011.

[17] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. IEEE transactions on Robotics, 23(1):34–46, 2007.

[18] Saba Arshad and Gon-Woo Kim. Role of deep learning in loop closure detection for visual and lidar slam: A survey. Sensors, 21(4):1243, 2021.

[19] Udo Frese, René Wagner, and Thomas Röfer. A slam overview from a user's perspective. KI-Künstliche Intelligenz, 24:191–198, 2010.

[20] Sebastian Thrun, Michael Beetz, Maren Bennewitz, Wolfram Burgard, Armin B Cremers, Frank Dellaert, Dieter Fox, Dirk Haehnel, Chuck Rosenberg, Nicholas Roy, et al. Probabilistic algorithms and the interactive museum tour-guide robot minerva. The international journal of robotics research, 19(11):972–999, 2000.

[21] Yudai Hasegawa and Yasutaka Fujimoto. Experimental verification of path planning with slam. IEEJ Journal of Industry Applications, 5(3):253–260, 2016.

[22] Xuexi Zhang, Jiajun Lai, Dongliang Xu, Huaijun Li, and Minyue Fu. 2d lidar-based slam and path planning for indoor rescue using mobile robots. Journal of Advanced Transportation, 2020:1–14, 2020.

[23] Hai Zhu Pan and Jin Xue Zhang. Extending rrt for robot motion planning with slam. Applied Mechanics and Materials, 151:493–497, 2012.

[24] WAS Norzam, HF Hawari, and K Kamarudin. Analysis of mobile robot indoor mapping using gmapping based slam with different parameter. In IOP Conference Series: Materials Science and Engineering, volume 705, page 012037. IOP Publishing, 2019.

[25] WaveShare. Jetbot ros ai kit. https://www.waveshare.com/jetbot-ros-ai-kit.htm, n.d.

[26] S. Slamtec.Co., Ltd. Rplidar a1. https://cdnshop.adafruit.com/product-files/4010/4010$_d atasheet.pdf, 2018. Version 2.1.$

[27] Fiqri Ahmad Agung, H Herizon, Era Madona, Muhammad Rohfadli, et al. Implementation of lidar sensor for mobile robot delivery based on robot operating system. JECCOM: International Journal of Electronics Engineering and Applied Science, 1(2):67–79, 2023.

[28] InvenSense. Mpu-9250 nine-axis (gyro + accelerometer + compass) mems motiontracking device. http://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/, n.d.

[29] Open Source Robotics Foundation. Creating a ros package. http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage, n.d. Accessed: 2024-06-26.