People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering
Department of Electronics

Project Report Presented in Partial Fulfilment of

the Requirements of the Degree of

'MASTER'

In: **Electronics**

Option: **Computer Engineering**

ROS Navigation STACK on the B21r mobile robot.

Presented by:

- **DELHOUM Nabil**
- **LAKREB Samira**

Supervisors:

- **Dr.BELAIDI Hadjira**
- **Dr. KAHLOUCHE Souhila**

Registration Number: 2021/2022

Abstract

Perception is an imperative task for a mobile robot to be able to navigate autonomously inits environment. The robot can perceive its surrounding through one or more sensors; then, create a representative map of the environment and locate itself on it in real time.

The propose of this project is to build a surveillance strategy to enable a mobile robot to navigate and supervise an interior environment. It consists in the implementation of a perception techniques, localization and navigation strategy on a surveillance robot.

Hence, this project deals with ROS Navigation Stack which is powerful for mobile robots to move from place to place. We worked with B21r mobile robot of CDTA equipped with light detection and ranging (LIDAR) sensor and D435i depth camera. First, our work consists of building a map of the hall of CDTA (the Advanced Technologies Development Center) using Lidarsensor and Simultaneous Localization and Mapping techniques. Then, we localized our robot inside that map using AMCL, therefore, our B21r mobile robot is well localized. Second, we created a path so that our robot reaches its goal or target and creates a new path in case the robot faces obstacles that were not present in the old map. However, our robot was not able to correctly deal with its last task consisting in tracking the planned path; even that, its position is updated enabling it to achieve its desired destination, due to mechanical problem of robot wheel. The work is implemented using ROS melodic and Rviz tool.

Acknowledgement

First of all, we are thankful to Allah, the most gracious and the most merciful for helping us accomplish this work in such a hard time.

We would like to thank and express our great gratitude to ***Dr.BELAIDI HADJIRA,*** *our project's supervisor and Madam KAHLOUCHE SOUHILA, our project co-supervisor for their professional assistance, support, advice and guidance during our work on this project and being there whenever we had that confusion or a question.*

Finally, we would like to thank jury members and all IGEE teachers.

DEDICATION

IN THE NAME OF ALLAH THE MOST BENEFICIANT THE MERCIFULL

This work is dedicated to our loving and caring Parents who are cause of our success. And great gratitude and love for our sisters and brothers for their love and support.

This work is also dedicated to our beloved grandparents and all our closest friends for their continuous encouragement.

Thank you.

TABLE OF CONTENTS

Abstract.....i

Acknowledgement..... ii

Dedication.....iii

Table of Contents… vi

List of Figures… ..... …….. ……..xi

List of Abbreviations… .....…..xii

GENERAL INTRODUCTION.....1

CHAPTER 01: State of the art 3

I.1. Motivation.....3

I.2. Goal of the Project.....3

I.3. Background and Related Work 4

I.3.1. Definition and history of robots.....4

I.3.2. Mobile robot today.....5

I.3.3. The idea of mobile robot 5

I.3.4. Types of Robots.....6

I.3.5. Sensors of mobile robot 9

I.4. Robot Operating System.....12

I.4.1ROS Overview.....13

I.4.2. ROS nodes.....14

I.4.3. ROS messages 14

I.4.4 ROS topics 15

I.4.5 ROS services 15

I.5. The Navigation Stack—System Overview.....15

I.5.1. AMCL and Map_server 16

1.5.2.Local and Global costmaps.....16

I.5.3.Local and Global planners.....17

I.6. Conclusion 17

CHAPTER 02.....19

II.1. System Requirements 19

II.2. System Hardware 19

II.2.1. Description of The B21r mobile robot of CDTA 20

II.2.1.1. Synchronous drive mobile robot B21r 21

II.2.1.2Wheels design 22
II.2.2. RPLIDAR A2 23
II.2.2.1. System connection 24
II.2.2.2Mechanism 24
II.2.2.3.Communication interface 25
II.2.3. Intel Real sense Camera D435i 26
II.3. Overall System 28
II.4. Conclusion 29
CHAPTER 03 31
III.1. The system Network 31
III.2. ROS tools 31
III.2.1. Rviz 31
III.2.1.1. Transform (tf) 32
III.3. ROS Navigation Stack 33
III.3.1. Mapping 34
III.3.1.1. Mapping using LIDAR 37
III.3.1.2. Mapping using D435i Camera 39
III.3.1.3. Map saving 41
III.3.2. Robot-Localization 41
III.3.3. Path Planning 43
III.3.4. Obstacle avoidance 45
III.4. CONCLUSION 46
CHAPTER 04 48
VI.1. Visual-SLAM 48
VI.2. Comparing LiDAR SLAMS 49
VI.2.1. Scenario 01: Time Consuming 50
VI.2.2. Scenario 2: localization in SLAMs 53
VI.3. Localization (AMCL) 54
VI.4. Navigation stack 56
VI.4.1 Testing our navigation stack 56
VI.4.2 Random move and interrupted speed of the robot 56
VI.4.3 serial sniffer interceptty 58
VI.4.4.Robot wheels' response 59
VI.5. Conclusion 61

Conclusion 63
References and Bibliography 65

LISTE OF FIGURES

Figure1.1.Surveillance mobile robot..... 7
Figure1.2.Aerospace robot..... 8
Figure1.3.Navigational robot Drones..... 8
Figure1.4.Industrial robots..... 8
Figure1.5.Medical robots..... 8
Figure1.6.Military robots..... 8
Figure1.7.Autonomous underwater robot..... 8
Figure 1.8.Mobile robot equipped with various sensor types..... 9
Figure 1.9.IntelRealsense camera D435i..... 10
Figure 1.10. HC-SR04 Ultrasonic sensor..... 11
Figure 1.11.Odometry sensor..... 11
Figure 1.12. Inertial measurement unit (IMU)..... 11
Figure 1.13. Illustration of ROS nodes and messages..... 13
Figure 1.14. Visualisation of ROS concepts..... 14
Figure 1.15. Overview of a typical system running the Navigation Stack..... 16
Figure 2.1: The B21r mobile robot..... 20
Figure 2.2: The B21r and its onboard sensors..... 21
Figure 2.3: The platform geometry of a synchronous drive mobile robot..... 22
Figure 2.4: Decentred adjustable wheels..... 22
Figure 2.5. The RPLIDAR A2M8..... 23
Figure 2.6.The RPLIDAR A2M8 system composition..... 24
Figure 2.7. TOF ranging schematic..... 25
Figure 2.8.RPLIDAR Power Interface..... 26
Figure 2.9. Active Infrared (IR) Stereo Vision Technology..... 27
Figure 2.10. Depth Measurement (Z) versus Range (R)..... 28
Figure 2.11.Shematic diagram of mobile robot..... 29
Figure 3.1. ROS visualization tool (Rviz)..... 32
Figure 3.2. TF Frames..... 32
Figure 3.3. ROS Navigation steps..... 33
Figure 3.4. Hector Slam flowchart..... 35
Figure 3.5. Gmapping Slam flowchart..... 36
Figure 3.6. V-SLAM flowchart..... 37
Figure 3.7. LIDAR sensor running on Rviz..... 38
Figure 3.8. Hector-SLAM on rviz using lidar..... 39
Figure 3.9. D435i camera running on rviz..... 40
Figure 3.10. Starting visual slam with d435i camera..... 41
Figure 3.11. AMCL node and topics related..... 42
Figure 3.12. Particle localization..... 43

Figure 3.13. Local and global costmaps…… 44
Figure 3.14. path planning…… 45
Figure 3.15. the velocity of B21r mobile robot…… 46
Figure 4.1. Map builded using visual SLAM…… 48
Figure 4.2. map of the CDTA hallway using HECTOR SLAM…… 49
Figure 4.3. map of the CDTA hallway using Gmapping SLAM…… 50
Figure4.4-a.Hector's time stops needed during each minute to eliminate errors while mapping…… 52
Figure 4.4-b. Gmapping's time stops needed during each minute to eliminate errors whilemapping…… 52
Figure 4.5. different errors may happen when mapping…… 52
Figure 4.6. error of sudden moves…… 54
Figure 4.7-a. AMCL localization results Cloud in starting pose…… 55
Figure 4.7-b. AMCL localization results Cloud in final pose…… 55
Figure 4.8-a. random response with constant angular velocity=0.1m/s…… 57
Figure 4.8-b. random response with constant linear velocity=0.1m/s…… 57
Figure 4.9-a. Normal robot's response after the setup of interceptty to a constant angularvelocity of 0.1m/s…… 58
Figure 4.9-b.Normal robot's response after the setup of interceptty to a constant linear velocityof 0.1m/s… 58
Figure 4.10. original path vs real path and the point chosen for test…… 60
Figure 4.11. distance error of several points in real path from original path…… 61

LISTE OF TABLES

Table.4.1 number of times that we need stops while mapping for each minute……... 51
Table.4.2.: Hector slam localization…… 53
Table 4.3: Gmapping localization…… 53
Table 4.4: The error between real localization and AMCL results……... 55
Table 4.5: Manual test of the wheels' movement…… 59

LIST OF ABREVIATIONS

AMCL Adaptive Monte Carlo Localization

AMRs Autonomous Mobile Robots

AGVS Automated Guided vehicle

CDTA center for the development of advanced technologies

HW Hardware

HECTOR Heterogeneous Cooperating Team of Robots

IMU Inertial measurement unit

IR infrared sensor

LIDAR Light detection and ranging

ROS Robot operating system

SLAM Simultaneous Localisation and Mapping

SW Software

TF Transform Data

GENERAL INTRODUCTION

Nowadays, autonomous mobile robots are widely used for several purposes. Autonomous mobile robots which work without human intervention are required in Robotic fields. In particular, with the development of Lidar-based navigation technique, mobile robots could be located and navigate in “real time” in indoor and outdoor environment. Navigation technology is one of the fundamentals in field of automation and robotics.

This master project deals with localization and Navigation which are the key technologies of autonomous mobile service robots, and is essentially based on SLAM (Simultaneous Localization And Mapping) which is considered as the fundamental basis for robot localization and mapping. SLAM is constructing a map using mobile robot (mapping), then determining the position of this robot relative to this map. As it will be presented, the robot builds the map, localizes itself on the map and performs navigation. The map of the environment is a basic need for a robot to perform actions like moving room to room, picking an object from one place and taking it to another one. Our project is implemented using ROS (Robot Operating System) platform and rviz tool.

This project is realized in the “Advanced Technologies Development Center (CDTA)” in Baba Hassen, Algiers, in the robotics division, and our algorithm has been implemented on the B21R mobile robot in indoor environment.

The rest of the report is organized as follow. Chapter 1 introduces mobile robots and and their basic components; then, it introduces the Robot Operating System (ROS) and discusses its fundamentals. Chapter 2 is a description of the b21r mobile robot, Lidar sensor and RGB-D camera. Chapter 3 explains in details the implementation of ROS Navigation Stack. Chapter 4 is dedicated for discussing the obtained results. The report ends up by conclusion and future work.

CHAPTER 01

STATE OF THE ART

A mobile robot is a machine controlled by software. Mobile robot needs to estimate its location with respect to objects in its environment and to map the positions of objects that it encounters in its environment.

Mobile robots may roam around their surroundings and are not restricted to a single physical place. Using a LIDAR sensor, the mobile robot can execute navigation and localization with the help of a low-power onboard computer and a remote-control system. With intel REALSENSE DEPTH CAMERA D435i, the mobile robot can track not only the object but the depth and exactly the distance of each single object.

It is more convenient to use the open-source software ROS to realize mobile robot navigation.

I.1. Motivation

Robots are everywhere revolutioning every aspect of our life. The ROS navigation stack on the B21r mobile robot equipped with LIDAR sensor, Intel REALSENSE CAMERA D435i is a challenge that we wanted to accomplish. It also leads us to get familiar with the open-source software ROS we may be need in our professional career.

I.2. Goal of the Project

The aim of this project is to create completely a map of an indoor environment so that our mobile robot localizes itself inside it and creates a path to reach a given position then tracks this path avoiding obstacles. The robot uses RPLIDAR sensor to scan and visualize the surrounding environmentand to create a map and plan its route towards the destination in order to detect humans and avoid obstacles in the path to reach the destination.

Additionally, a list of system requirements must be developed, as well as the structure of the system's components. The system is then used to carry out its functions in accordance with the

developed system structure. In the second chapter, the components are chosen by analyzing their performance and capabilities in relation to the requirements.

Another objective is to examine how ROS can be used in the development of mobile robot, as well as to identify its potential benefits.

I.3. Background and Related Work

I.3.1. Definition and history of robots

Robots! Robots are everywhere such as robots fighting fires, making products and items, saving time and lives on Mars and in the seas, in processing plants and schools, in emergency clinics and homes... Robots now are important and have an importance on many aspects of modern life, ranging from modern manufacturing to medical services, transportation, and deep space and ocean exploration. Robots will be as ubiquitous and individual as today's Smartphones in the future [1].

The robot "concept" was established by many creative numerous historical realizations. Nonetheless, the "physical" robot had to wait until the twentieth century to see the development of its underlying technologies. The term "robot" was coined in 1920, and it derived from the Slavic word "robota," which means "subordinate labor"[1]. The first robots were created around the middle of the twentieth century. The first industrial robot, "Unimate," designed by American inventor George Devol, was put to work on a General Motors assembly line in 1961[2]. They benefited from advancements in mechanical, control, computer, and electronic technologies. New designs, as always, inspire new research and discoveries, which lead to improved solutions and thus novel concepts. Over time, this circle produced the knowledge and understanding that gave birth to the field of "robotics"[1]. With the advancement of science in all research areas and the emergence of new ones, the field of robotics grew in importance in terms of investments and research, and robotics evolved into a synthesis of various disciplines, including electrical and electronics, computer science and engineering, mechatronics and mechanical design, control and automation systems, AI and cognitive systems, and so on.

By the time robots were fully capable of providing assistance, utility, and precision to human tasks and activities under human control and supervision, the need for robotic systems to be more independent and to process the ability to complete tasks and provide services on their own had become a necessity and vital part of future robots, prompting the term "autonomous robots" to emerge. An autonomous robot is one that acts without external influence. Artificial intelligence, robotics, and information engineering are all subfields of autonomous robotics. Author and inventor David L. Heiserma proposed and demonstrated early versions[3][4][5]. Following that, the robotics systems were categorized based on where they were found. The robotics systems were then classified according to where they fell on the "autonomy Spectrum". As the system progresses from manual to fully autonomous.

I.3.2. Mobile robot today

Mobile robots are being used in increasing numbers in industrial applications. They include both Automated Guided Vehicles (AGVs) and Autonomous Mobile Robots (AMRs). The use of mobile robots is increasing as companies look for ways to deploy their existing workforces more effectively.

Mobile robots are reducing the time and resources required to transport units from storage to production or distribution stations. According to analysts at Yokosuka, a Mitsubishi Electric subsidiary, AMRs are opening up new segments and applications by providing increased autonomy and intelligence. AMRs can be equipped with manipulator arms to assist dexterous human workers in manipulating large items like chassis or parts.

Autonomous robots are becoming more and more viable for a wide range of tasks. They can react in real time to their operating environments, avoiding obstacles such as human and determining the most efficient paths to reduce transit times and increase efficiency. Vision, navigation, and artificial intelligence advancements have made them viable for new roles [6].

I.3.3. The idea of mobile robot

AMRs have the ability to understand their operating environment, allowing them to navigate around obstacles and operate safely even when human operators are present. AMRs can

use data from cameras, laser scanners and other sensors to understand their environment and make the decisions needed to move around it.

The robot can create a map of its surroundings using the principles of triangulation to detect and locate reference points by feeding the data into Simultaneous Localization and Mapping (SLAM) algorithms. These references can be used by the robot to determine its current position and provide the information needed to move around within it. SLAM allows robot operators to modify their route using a laptop or other device, with the ability to make additional changes if necessary. It also offers a flexible alternative to AGVs guided by magnetic tape.

The second option provided by SLAM is autonomous navigation, in which the robot can choose the fastest route between two points or extend its journey in response to additional information from the fleet management system (FMS). This same autonomous decision-making allows it to respond to the presence of obstacles and people, making AMRs safer to deploy alongside human workers.

I.3.4. Types of Robots

- **Surveillance mobile robots**

One of the important applications of the robots is surveillance. The surveillance is the operation of monitoring humans, locations and areas. Surveillance mobile is a system that composed by two parts.

The first part is a reactive navigation system in which a mobile robot moves avoiding obstacles in environment using sensors. Sensors are used to collect information about the robot and the environment. A robotic system requires an environment map (the mapping) to plan its route and navigate through it, and information about its position on the map (localization) in order to navigate for its current location to any other location represented in the environment map. The second part of the system uses camera for detection (like human detections or face detections...).

Surveillance robots can be found in all the normal environments (e.g: aerial, ground, surface water, underwater or space). An illustration is shown in figure1.1

Figure1.1. Surveillance mobile robot.

Surveillance robot can be found in any of the following types. Each robot has its own task and capabilities. It is not easy to define robots and not easy to categorise them. Some types of robots are described as follow.

- **Aerospace robot:** includes all types of flying robots that can operate in space such as Mars... An example is shown in figure 1.2.
- **Drones:** also called uncrewed aerial vehicle, they come in different sizes and have different levels of autonomy. An illustration is shown in figure 1.3.
- **Industrial:** consist of a manipulator arm designed to perform repetitive tasks. See figure 1.4.
- **Military & security:** used in Iraq and Afghanistan. Military robots include ground systems as the one shown in figure 1.6; while security robot include autonomous mobile systems.
- **Medical:** are robotic machines utilized in health sciences as illustrated in figure 1.5.
- **Underwater:** are robots that consist of deep-sea submersibles like Aquanaut as shown in figure 1.7....

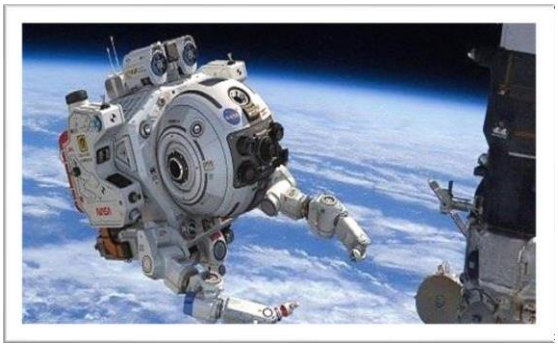

Figure1.2.Aerospace robot.

Figure1.3. Navigational robot Drones.

Figure1.4.Industrial robots.

Figure1.5. Medical robots.

Figure1.6. Military robots.

Figure1.7. Autonomous underwater robot.

The types of robots that were presented seem to be very unlike; however, their functions or their abilities given are all the same. They differ in their tasks and the complexity of robot operating system. The principal abilities in a fully autonomous robot are:

1. Perception:
    - Mapping and environment modelling
    - Localization

2. Decision:
   - Planning for paths and tasks
   - Collision avoidance
3. Actuation:
   - Path execution
   - Task accomplishment

I.3.5. Sensors of mobile robot

a) Perception sensors

Sensors are critical components of autonomous mobile robots; without them, they would be unable to gather information about the environment in which they operate and navigate.

Nowadays, a wide range of sensors are available, each with its own set of capabilities and designations, but the information they provide is frequently inconsistent or even contradictory.

An active or passive sensor can be used. Passive sensors simply receive energy from the environment and translate it into a meaningful form for the control units or systems, whereas active sensors emit energy into the environment in order to observe certain characteristics.

b) Requirement sensors

The following sensors, shown in figure 1.8, should be present in all mobile robots:

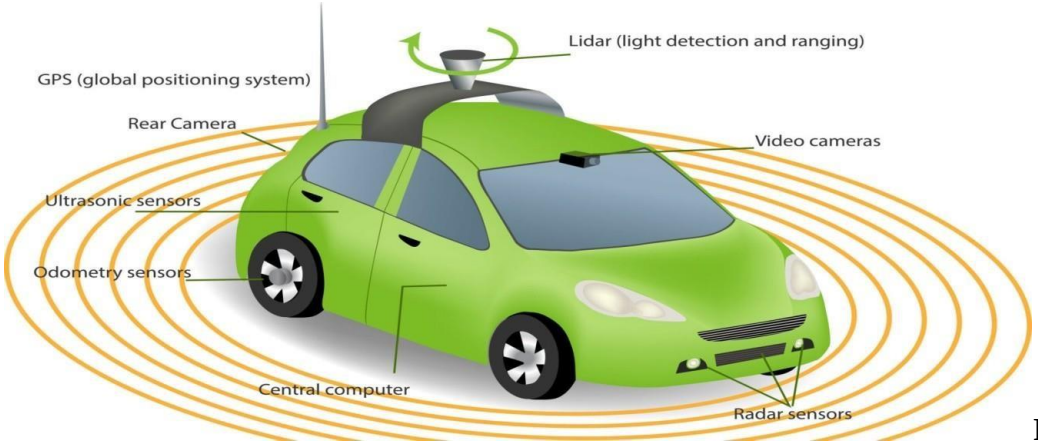

Figure 1.8. Mobile robot equipped with various sensor types [7].

- ***Global positioning system (GPS):*** Global positioning system is a simple device used with the robot for determining the location using satellite [8], complex math and general relativity. It is often used with sonar sensor; the mobile robot's position and obstacle avoidance will be determined by a combination of GPS and sonar sensor. The mobile robot should navigate according to the waypoint that is present to the GPS module.
- ***Light detection and ranging (LIDAR):*** is a remote sensing system that uses a laser signal to measure the distance to an object and then receives its reflection. It can also be used to accurately map out its surrounding. Mobile robot uses LIDAR to map the environment and detect and avoid obstacles.
- ***Camera and its working:*** the depth camera D435i is part of the intel Real sense D400 series of camera. D435i combines the robust depth sensing capabilities of the D435 with the addition of an inertial measurement unit (IMU). IMU helps to refine its depth awareness where the camera moves. D435i used to visualise the environment using SLAM and tracking allowing better point-cloud alignment [9].

Figure 1.9. IntelRealsense camera D435i.

- ***Ultrasonic sensor:*** is used to detect is an instrument used to detect objects that are far away from the robot using ultrasonic sound waves. It does not require any physical contact (see figure 1.10).

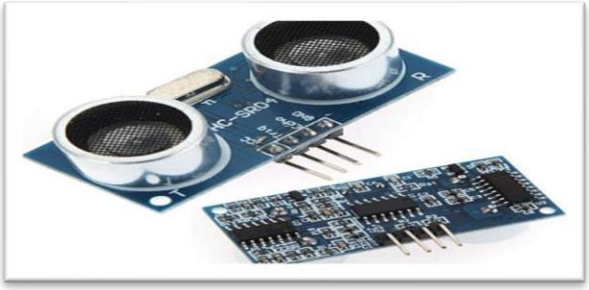

Figure 1.10. HC-SR04 Ultrasonic sensor

- ***Odometry sensor:*** are common on mobile robots, and they show how far the robot has traveled based on how much the wheels have turned**.**

Figure 1.11.Odometry sensor

- ***Inertial measurement unit (IMU):*** An IMU, as illustrated in figure 1.12, is a set of sensors. The navigation of mobile robots is aided by an inertial measurement unit. The data collected by a robot's IMU sensors is properly converted, and useful information such as position, orientation, and acceleration is calculated.

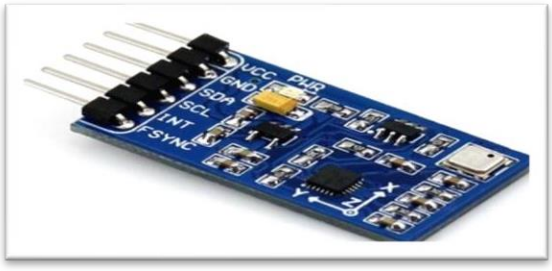

Figure 1.12. Inertial measurement unit (IMU)

I.4. Robot Operating System

Robot Operating System (ROS) is a meta-operating system, not an actual operating system. ROS is an operating system that sits on top of other operating systems and allows different processes to communicate with one another in real time. ROS provides a structured communications layer that runs on top of the operating systems of host computers [10]. The majority of ROS tools are compatible with peripheral hardware and can be used for a variety of purposes, so their use is not limited to robotics. The ROS core is made up of over two thousand packages, each with its own set of capabilities.

Hence, ROS is a collection of tools that perform the functions and services of an operating system on a single computer or across multiple computers. These services include hardware abstraction, message exchange between processes, package management, and so on. The number of available ROS tools, according to Ademovic [11], is ROS's greatest strength.

One very important characteristic of ROS is that it is completely open source. ROS was created with the goal of reusability of robotic SW in mind. Quigley [12] claims that writing SW for robots is difficult, especially as the scale and scope of robotics expands. HW can vary greatly between different types of robots, making code reuse extremely difficult. Furthermore, because robot SW is tightly coupled, extracting reusable code can be difficult, and ROS was created to address these issues.

To explain ROS, used the following five statements to summarize its philosophical goals [13]:

***ROS is peer-to-peer*:** ROS nodes are execution units that communicate directly or through publish/subscribe mechanisms.

***ROS is tools-based*:** a microkernel design, as well as a number of small tools and modules.

ROS is multi-lingual: It has support for C++, Python, Octave and LISP.

***ROS is 'thin'*:** It makes use of the Catkin build system to segment code into packages and libraries.

***ROS is free and open source*:** ROS is distributed under the terms of the BSD license, which allows the development of both non-commercial and commercial projects**.**

I.4.1. ROS Overview

Data is transferred between ROS nodes using internet protocol (IP)-based communication. ROS divides the robotic SW into ROS nodes, which can be run on a single machine or across a distributed computer cluster. ROS nodes communicate with one another via publish/subscribe channels, but they can also provide callable services to other nodes. One master node (roscore) acts as a name server in a running ROS system, allowing other nodes to find each other and form direct connections. This architecture encourages node reuse by reducing coupling between nodes. The same ROS nodes, for example, can be used without modification in both the robot and the simulator.

Nodes, messages, topics, and services are the core concepts of the ROS implementation.

The ROS Master is responsible for providing naming and registration services to the rest of the ROS system's nodes. Individual nodes can communicate with one another thanks to the Master. These nodes communicate with each other peer-to-peer once they have found each other. Nodes would be unable to locate one another, exchange messages, or invoke services without the Master.

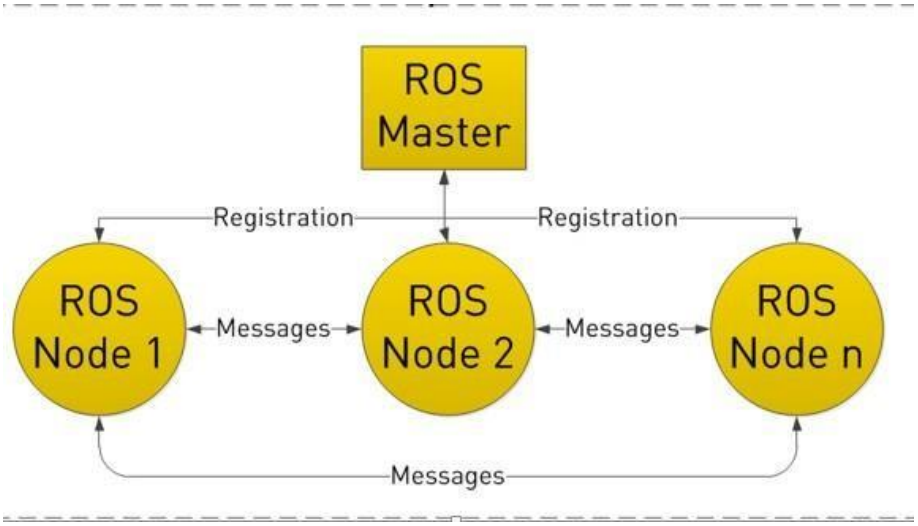

Figure 1.13. Illustration of ROS nodes and messages

The ROS concepts (nodes, messages, and topics) and their relationships are illustrated in Figure 1.14. Services, which are a type of communication between nodes, don't use the publish/subscribe mechanism and instead directly invoke the services of the other node.

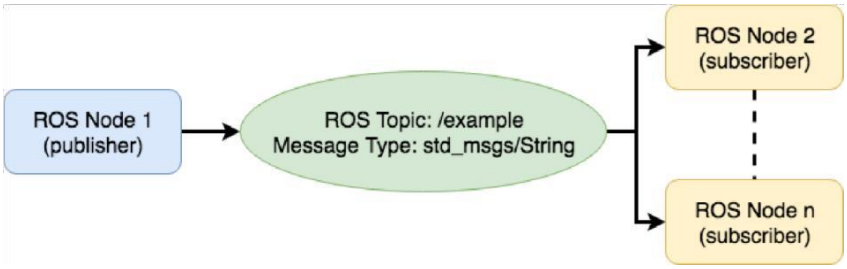

Figure 1.14. Visualisation of ROS concepts

In the subsections below, the ROS core concepts—nodes, topics, messages, and services—are explained in depth.

I.4.2. ROS nodes

In the ROS ecosystem, a node is a process that performs computation and can be thought of as a single unit of execution. Nodes can communicate with one another using a client-server architecture, in which each node is given a specific task and can act as both a client and a server at the same time. Nodes should complete their own tasks and report back to the rest of the network. This architecture has a significant advantage in terms of fault tolerance (as each node is an isolated part of the system).

I.4.3. ROS messages

Over 200 predefined messages are available in ROS, as well as the ability to create custom messages. The publish/subscribe mechanism is used to send messages between ROS nodes. The ROS message would be published to a specific ROS topic by one ROS node, while the other ROS node would subscribe to that ROS topic and receive the sent ROS message. Messages in ROS are usually described in text files in the msgs folder of the ROS folder structure. These text files adhere to certain ROS message description standards. The ROS message description format is fairly straightforward. Each ROS message is a data structure containing either primitive types (integers,

floats, or booleans) or an array of primitive types. Additionally, as a data type, ROS messages can contain other ROS messages or an array of ROS messages. ROS messages can also be exchanged in direct communication between nodes, known as ROS Services, and the messages must be stored in the srv folder in this case.

I.4.4. ROS topics

ROS nodes communicate with one another via topics: we publish to a topic to send messages, and we subscribe to a topic to receive messages.

When ROS nodes communicate using the publish/subscribe mechanism, ROS topics are used. Each ROS topic has its own name, which nodes can use to publish or subscribe to it.

I.4.5. ROS services

ROS services are used when nodes need to communicate directly with one another. The publish/subscribe mechanism is bypassed in this case, and nodes can communicate directly with one another using the defined request and reply messages. Even so, because ROS services are a form of direct communication, they improve system performance while simultaneously reducing system decoupling.

I.5. The Navigation Stack—System Overview

Navigation Stack is a set of resources that are useful so a robot is capable of planning and tracking a path while it avoids from obstacles that appear on its path. Navigation Stack needs SLAM systems to complete its task. SLAM allows a robot to locate itself and create a map of its surroundings. Some of SLAM tools are introduced in chapter 3.

The system overview of this task is shown in figure 1.15; as it can be seen, there exists three types of nodes: provided nodes, optional provided nodes and platform specific nodes[13].

- **Provided nodes:** are responsible mainly by managing the costmaps and for path planning functionalities.

- **The optional provided nodes:** amcl and map_server, since a static map is optional these nodes are also optional because they are related to static map functions.
- **The platform specific nodes:** such as sensor reading nodes and base controller nodes. They are nodes related to our robot.

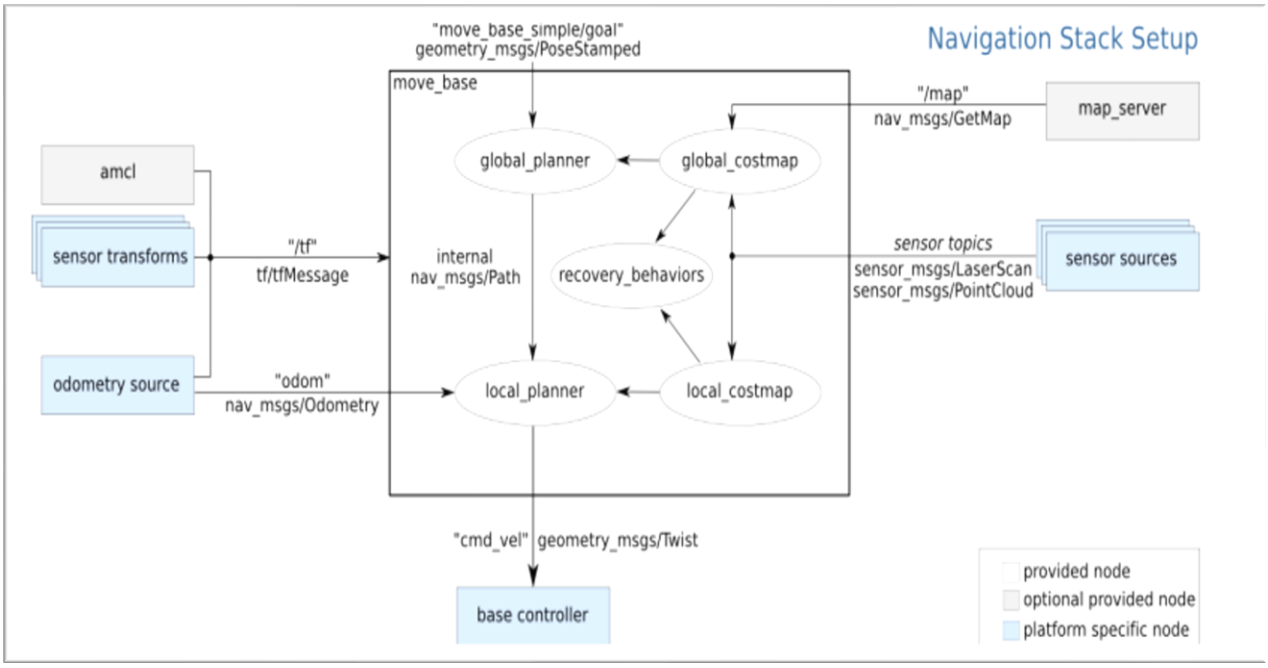

Figure 1.15. Overview of a typical system running the Navigation Stack [15].

I.5.1. AMCL and Map_server

Map_server contains two nodes. Map_server provides static map data as a ros service while map_saver saves a dynamically generated map to a file.

AMCL is a localization system that runs on a known map; it needs a static map and it will only work after a map is created. It randomly distributes particles in a known map, representing the possible robot locations based on the Monte Carlo localization approach, then uses a particle filter to determine the actual robot pose [14].

I.5.2. Local and Global costmaps

While the global costmap represents the whole environment, the local costmap is, in general, a scrolling window that moves in the global costmap in relation to the robot current position. The local and global 2D costmaps are the topics containing the information that represents the projection of the obstacles in a 2D map, as well as a security inflation radius, an area around the obstacles that guarantee that the robot will avoid any objects, no matter what is its orientation. These projections are associated to a cost, and the robot objective is to achieve the navigation goal by creating a path with the least possible cost [14].

I.5.3. Local and Global planners

The global planner takes the current position of the robot and the goal and create the path of lower cost in respect to the global costmap, then, it will send the trajectory to local planner. The local planner will execute each segment of the global planner. Local planner is as smaller part of global planner. Local planner given a path to follow (provided by global planner) will provide velocity commands in order to move the robot and starts following the path. If it finds obstacles, local planner can re-compute the path in order to avoid them [14].

I.6. Conclusion

This chapter was a thorough examination of the Robot Operating System (ROS) and mobile robot, as well as their implementation. The chapter began with an overview of mobile robot technology and its background. Following that, we talked about the basic components of a mobile robot. Then we looked at various sensors and components that are commonly used in mobile robots. Following that, we introduced the Robot Operating System (ROS) and discussed its fundamentals (nodes, messages, topics, services)

CHAPTER 02

HARDWARE SYSTEM

After the first chapter, which was about generalities of mobile robot and a detailed discussion about ROS and ROS Navigation Stack. In this chapter we will take a look at the B21r mobile robot and at the necessary hardware requirements. First, this chapter introduced the B21r mobile robot of CDTA and the light detection and ranging sensor and D435i camera that are used in our project while components will be described separately and how being connected to the B21r. The schematic diagram of our system will be shown at the end of this chapter.

II.1. System Requirements

To ensure effective implementation, the system must have the following characteristics.

- The mobile robot uses RPLIDAR to perceive its surrounding; then, create a representative map and locate itself on it in real time.
- We used RGB-D camera to visualize the surrounding environment.

II.2. System Hardware

We worked with the mobile robot B21r of the Center for the Development of Advanced Technologies (CDTA). Our system is composed of:

- Mobile robot B21r.
- A PC which is needed to handle information; it is connected to B21r mobile robot through USB-to-USB cable.
- LIDAR sensor.
- Camera.
- Micro USB cable
- USB to USB cable
- Cable matters USB C to Micro USB cable

II.2.1. Description of The B21r mobile robot of CDTA

The mobile robot B21r is the experimental platform used for the implementation of our project equipped with LIDAR sensor and d435i Camera.

The B21R mobile robot consists of three main sections (see Figure 2.1): the Base, the Enclosure and the Console. In the Base we find the mechanical drive and steering components, batteries, motors and motor control electronics. The Enclosure contains the main computer, tactile sensors, much of the power distribution system, sonar, IR sensors and communication equipment. The camera and pan-tilt unit, which are critical components of the vision system, are mounted on the Console's top.

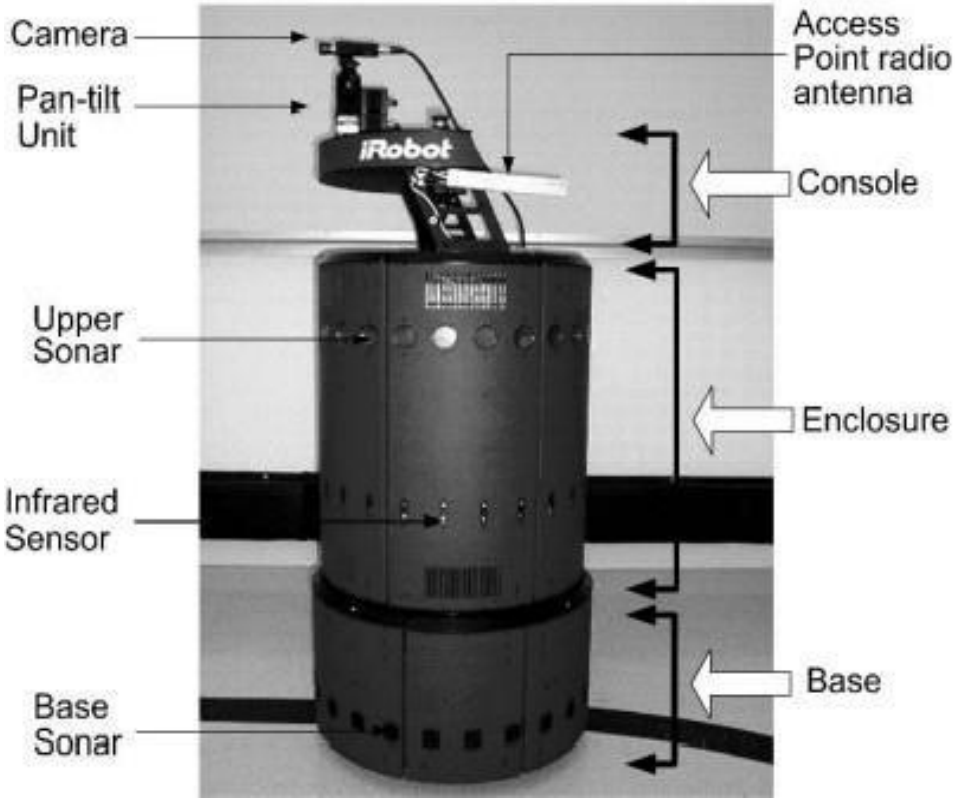

Figure 2.1: The B21r mobile robot.

The B21r is a synchronous drive mobile platform. It is made up of two ultrasonic sensor belts, an encoder, an infrared belt, a laser, tactile sensors on the sides, and a CCD camera (charge-coupled-device) (Figure 2.2).

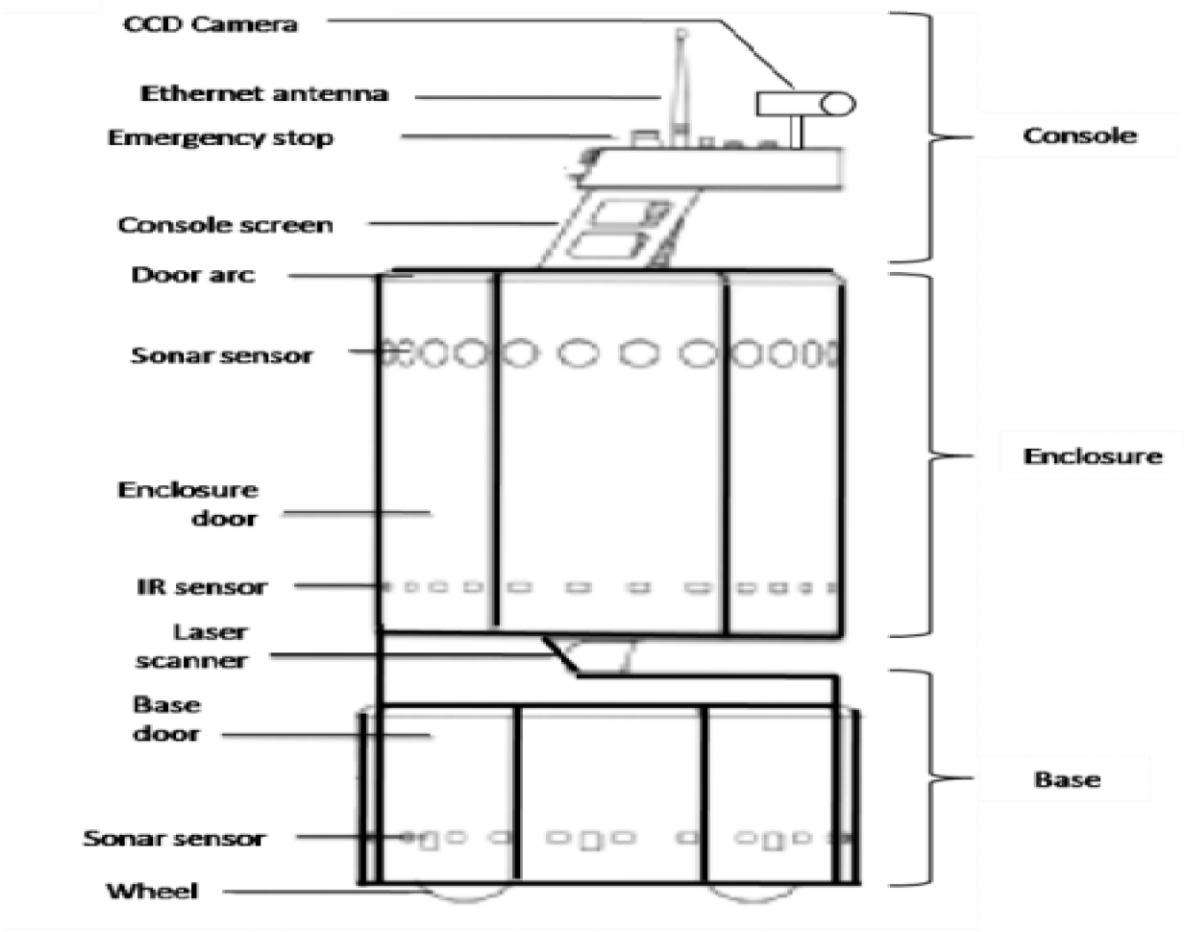

Figure 2.2: The B21r and its onboard sensors.

II.2.1.1 Synchronous drive mobile robot B21r

The synchronous drive is a technique for reducing the effect of sliding and increasing the traction strength. The configuration of the synchronous drive robot is similar to a robot with three or four coupled wheels which operate at the same time with the same velocity and the same orientation. This system is realized by two motors, one for the traction and the other for the orientation. The whole system is linked with a chain to ensure that the wheels turn in synchronous manner. Figure 2.3 shows the robot base with four coupled wheels linked by chains.

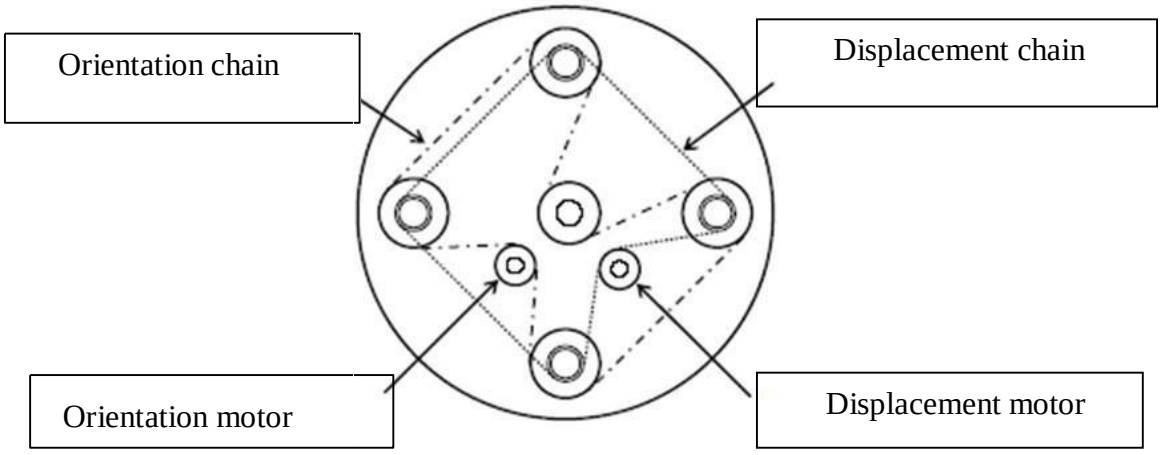

Figure 2.3: The platform geometry of a synchronous drive mobile robot.

II.2.1.2.Wheels design

B21r comes with four decentered adjustable wheels that rotate along two axes. The system rotates around the Y-axis, allowing the wheels to roll in order to achieve translation, and around the K-axis, allowing for a change in orientation (Figure2.4).

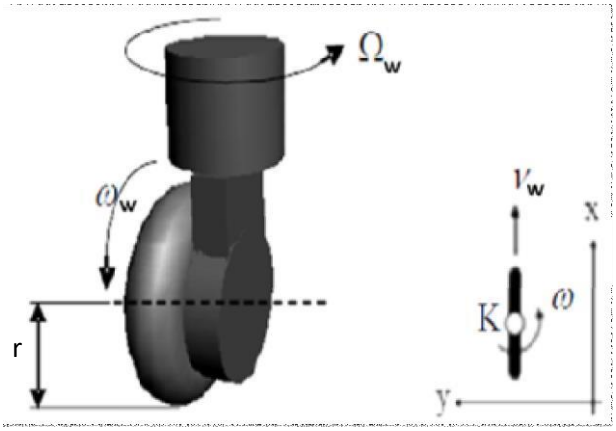

Figure 2.4: Decentered adjustable wheels.

Adjustable wheel parameters:

r = the wheel radius.

V_w = the wheel linear velocity.

ω_w = the wheel angular velocity.

Ω_w =orientation velocity.

II.2.2. RPLIDAR A2

RPLIDAR A2 is the latest generation low cost 360-degree 2D laser scanner (LIDAR). We used RPLIDAR A2M8 which is the enhanced version of 2D laser range scanner. It can perform 2D 360-degree laser scan and detect an obstacle located up to 12m from it. LiDAR was frequently employed as the primary sensor in early SLAM research because it is the best sensor for constructing a grid map for the surrounding environment. RPLIDAR A2M8 is shown in figure 2.5.

Figure 2.5. The RPLIDAR A2M8.

II.2.2.1. System connection

A2M8 is made up of a range scanner core mechanical powering components and communication and power interface (see figure 2.6). The mechanical Powering Part cause the Scanner core to rotate at high speed and scan clockwise.

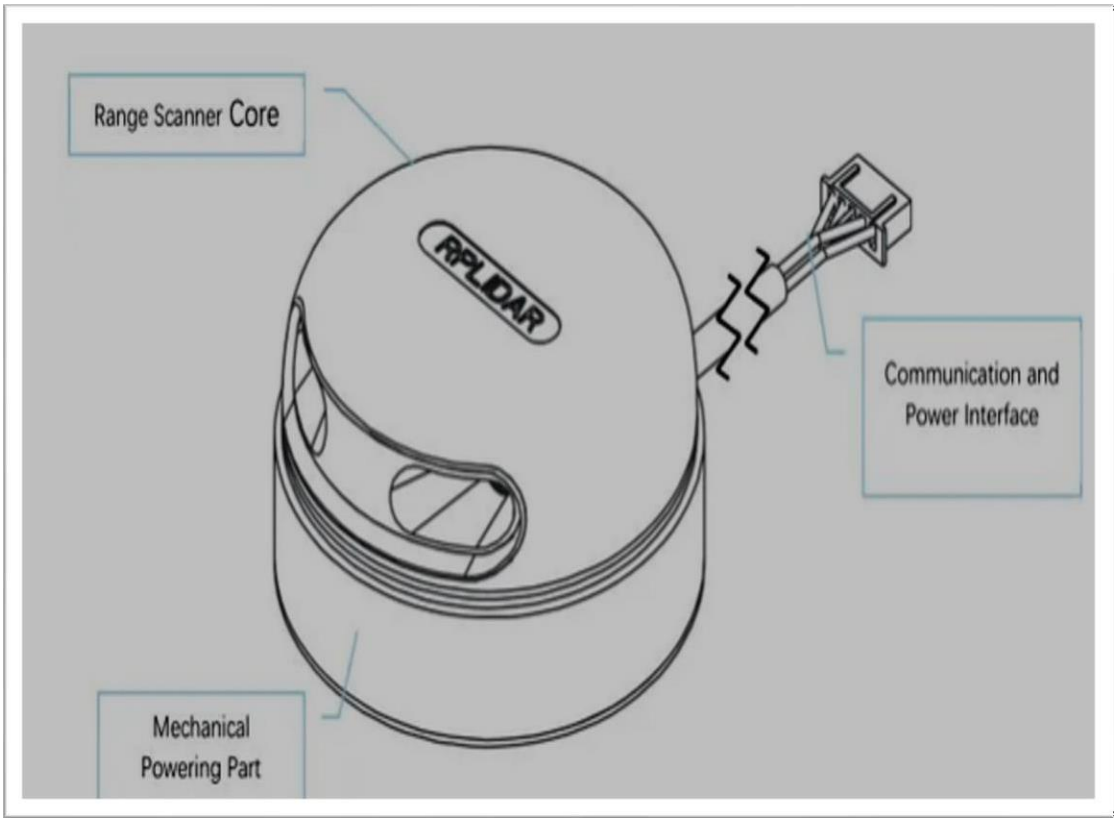

Figure 2.6. RPLIDAR A2M8 system composition.

II.2.2.2. Mechanism

The Lidar sensor on a vehicle consists of a laser transmitter and light receiver, the transmitter sends light beam that strike nearby objects present in the range of the sensor, the beam of light will be reflected back to the sensor when hitting an object as shown in figure 2.7. The

Lidar system records each beam's roundtrip data, measuring time to every object in the vehicle's vicinity and the angle of the beam relative to the sensor frame **[15].**

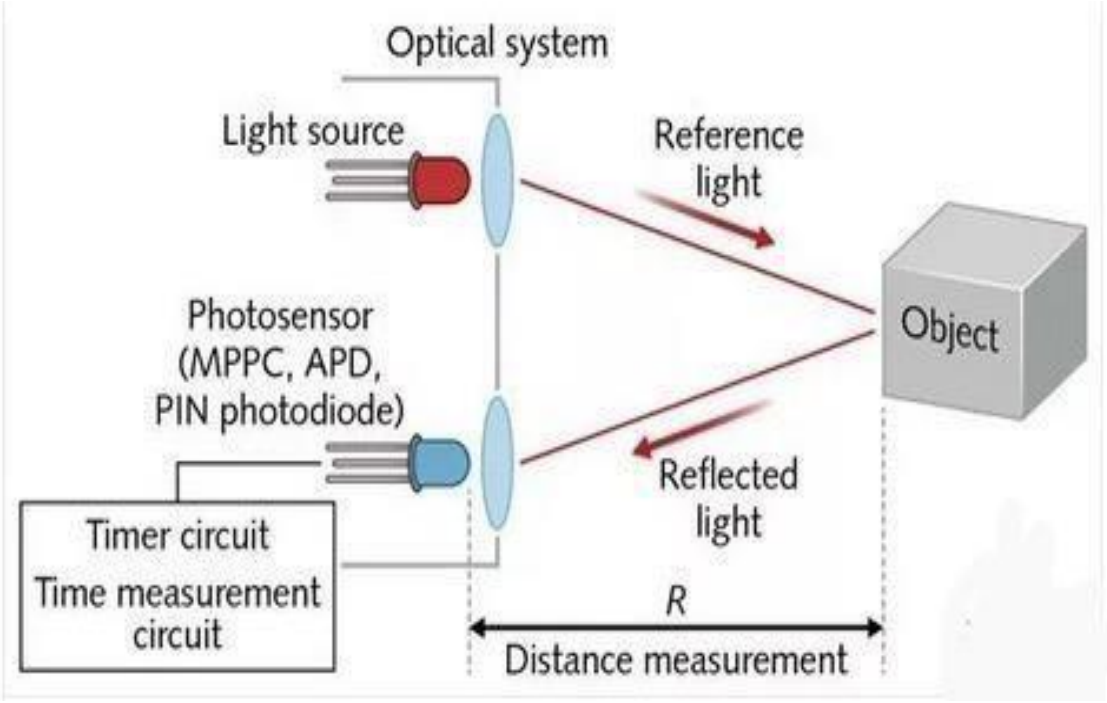

Figure 2.7. TOF ranging schematic [15].

The distance between sensor and target can be calculated with a high accuracy using the following formula: $DISTANCE = \frac{SPEED\ \&F\ LIGH \times TIME}{/}$ where "Time" is the time between emitting and receiving the signal.

II.2.2.3. Communication interface

RPLIDAR A2M8 needs 5V DC power for powering the range scanner core and the motor system. XH2.54-5P male socket shown in figure 2.8 used by the standard A2.

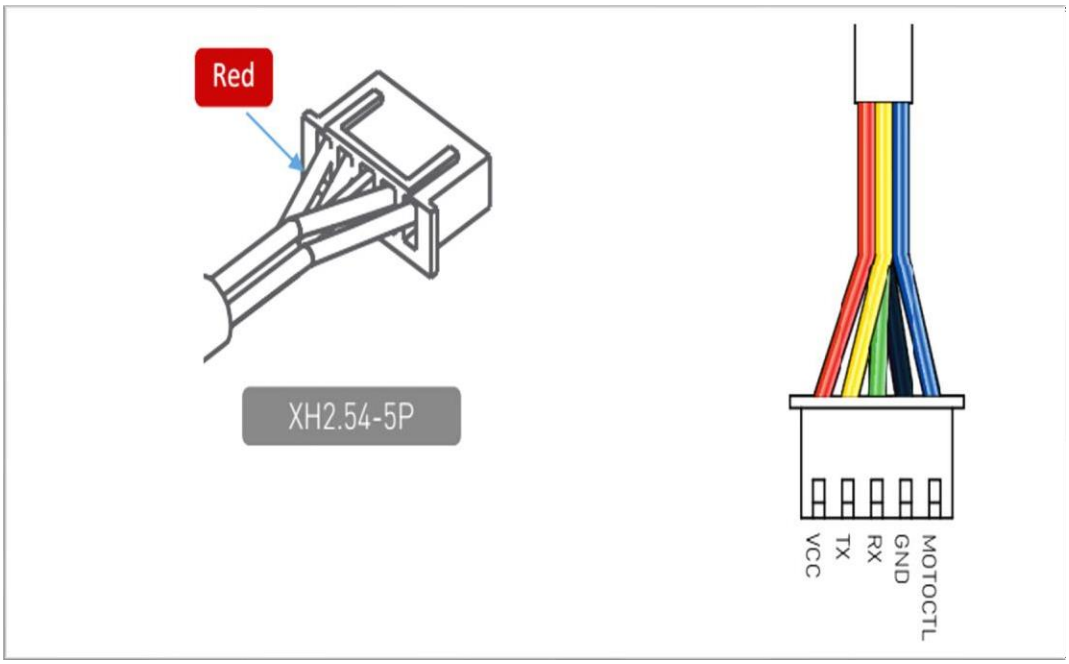

Figure 2.8. RPLIDAR Power Interface.

We connected our RPLIDAR to our laptop through Micro USB cable. Flashing green light indicates normal activity of sensor.

II.2.3. Intel Real sense Camera D435i

Our robot needs to visualize the real surroundings and get the needed information from the real environment. The needed information from the real environment to the mobile robot is delivered by D435i camera. RGB-D camera can provide both color and depth information in its view field. It is possible to achieve the tasks of mapping and localization allowing better point-cloud alignment. It is the most capable sensor for building a complete 3D scene map [16]. We connected D435i to our PC using cable matters USB C to Micro USB cable.

The Intel RealSense D400 series depth camera uses stereo vision to calculate depth. The stereo vision implementation consists of a left imager, right imager, and an optional infrared projector. The infrared projector projects a non-visible static IR pattern to improve depth accuracy

in scenes with low texture. The left and right imagers capture the scene and send imager data to the depth imaging (vision) processor, which calculates depth values for each pixel in the image by correlating points on the left image to the right image and via the shift between a point on the Left image and the Right image. The depth pixel values are processed to generate a depth frame. Subsequent depth frames create a depth video Stream [16].

The depth pixel value is a measurement from the parallel plane of the imagers and not the absolute range as illustrated in figure 2.9 and 2.10 [16].

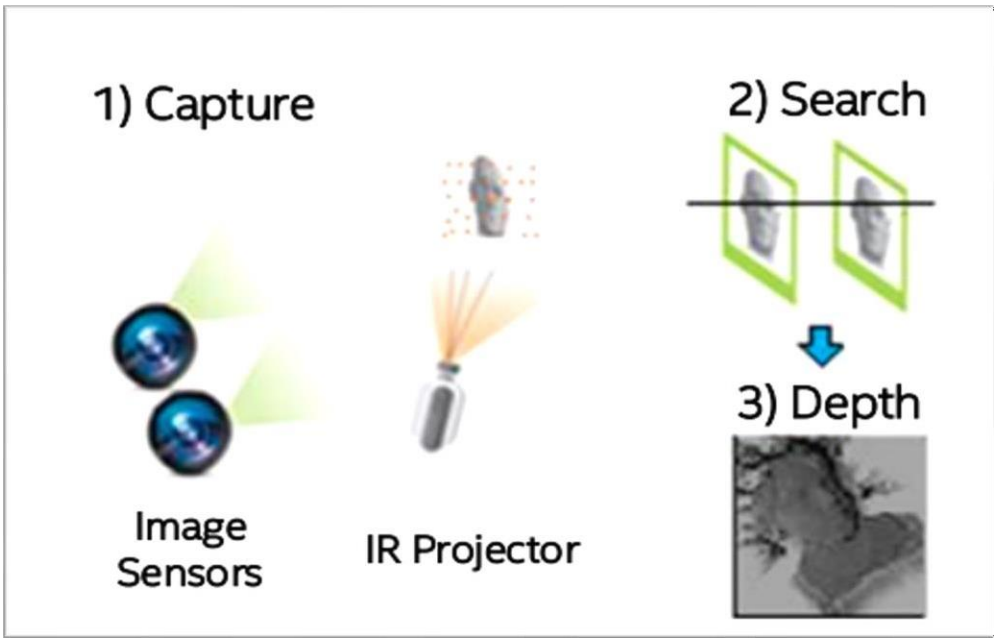

Figure 2.9. Active Infrared (IR) Stereo Vision Technology [16].

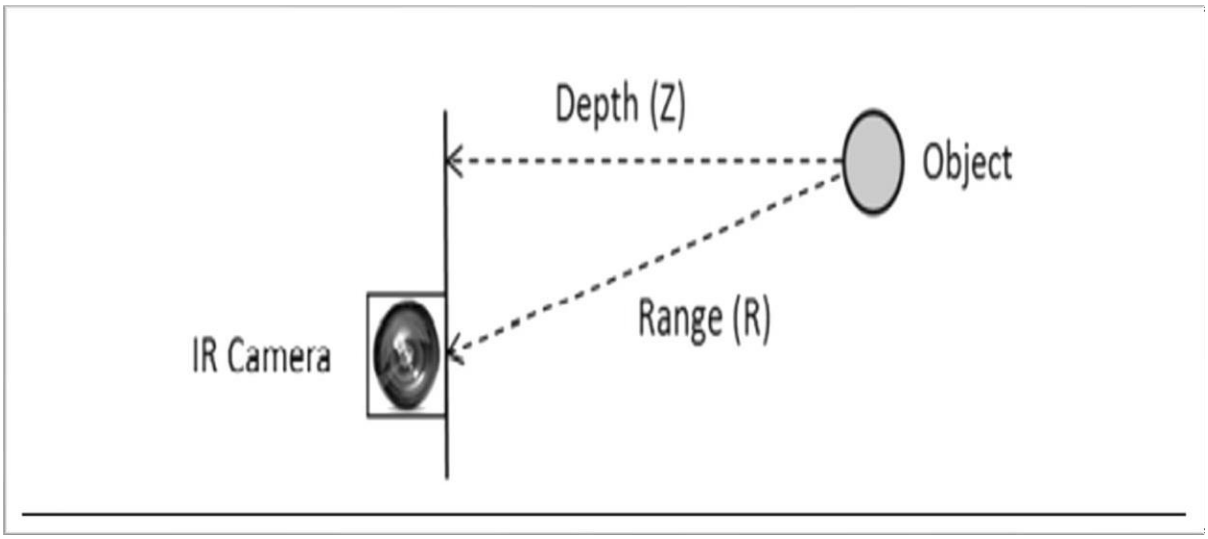

Figure 2.10. Depth Measurement (Z) versus Range (R) [16].

II.3. Overall System

After collecting and assembling each component required for our robot, we connected everything in accordance with figure 2.11, which depicts the robot's schematic diagram. Everything has been implemented in ROS framework for visualization purposes during real-time testing. Lidar is connected to our laptop using Micro USB cable and the camera is connected through USB-C to Micro USB. A laptop has been used and is connected to B21r mobile robot by USB to USB cable. It is also equipped with Ubuntu Linux OS and ROS melodic framework.

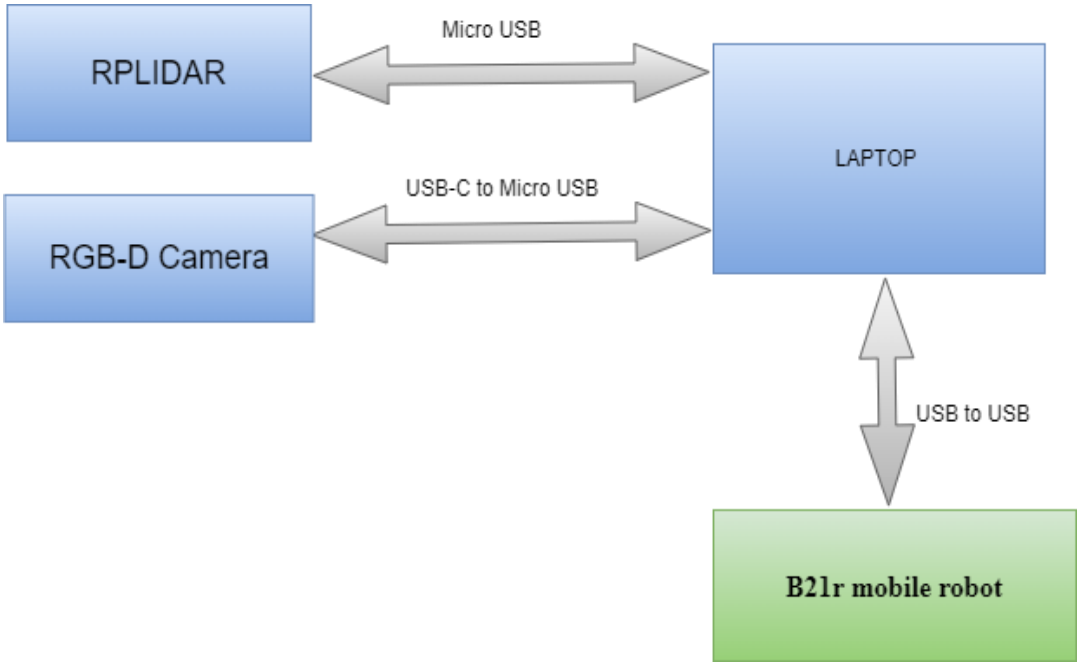

Figure 2.11. Schematic diagram of our mobile robot.

II.4. Conclusion

In this chapter, we presented B21r mobile robot that we worked with and we described in details the Lidar sensor and d435i camera and how they are connected to the b21r mobile robot. In the next chapter, we will interface the components with ROS and we will detail step by step Navigation Stack in ROS and how we built a map using either a Lidar sensor or a camera.

CHAPTER 03

SOFTWARE SYSTEM

Following the implementation of HW in the second chapter, now we move on to the implementation of SW. First, we will talk about ROS navigation stack and what we need in order to perform robot navigation and we introduced SLAM systems and some of its methods. As it will be presented, the robot builds the map, localizes itself on the map and performs navigation.

III.1. The system Network

We installed Ubuntu 18.04 double boot and ROS Melodic in our laptop. When the laptop boots for the first time we have the right to choose between windows and Ubuntu operating system. Then, we connected our PC to B21 mobile robot using USB to USB cable. The B21r robot uses RFLEX driver which provides a means of controlling and getting data from different subsystems (Motor, Sonar, IR, System...)

III.2. ROS tools

To inspect and debug messages, ROS provides a number of GUI (Graphical User Interface) and command-line tools. Rviz is one that is most usually used and the one that we used in our work.

III.2.1. Rviz

Rviz is a 3D visualizer in ROS that allows you to see 2D and 3D information from ROS topics and parameters.Rviz aids in the visualization of data such as robot models, 3D transform data (TF), point clouds, laser and image data, and a range of sensor data [17].It was used in this project to draw the robot's map and path, as well as its pose (see figure 3.1).

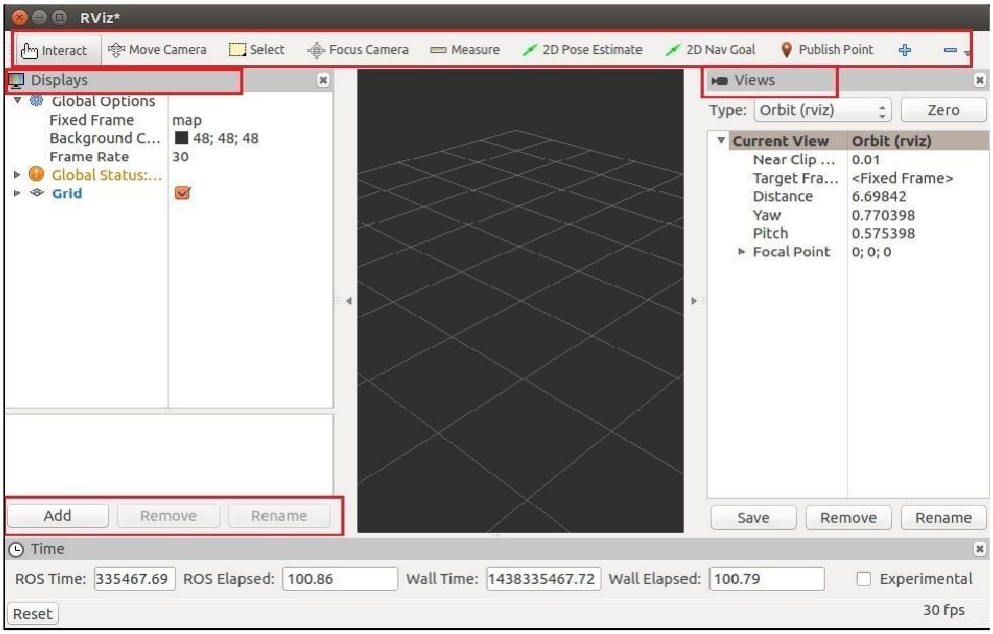

Figure 3.1. ROS visualization tool (Rviz).

III.2.1.1. Transform (tf)

Tf is a package that allows us to track several coordinate frames throughout time.The Navigation Stack requires the transformations in order to understand where the sensors are in relation to the robot's center (base link).The position of the mobile robot's frames is depicted in Figure 3.2[18].

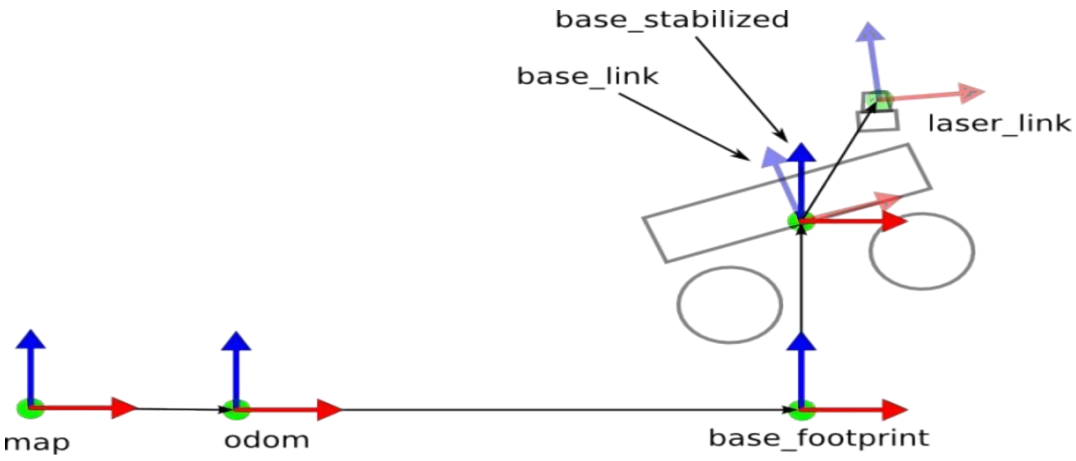

Figure 3.2. TF Frames.

- **Map:** serves as the global reference frame and the robot's position in respect to it should not change significantly over time.
- **Odom:** when new sensor data becomes available, drifts and can generate discontinuous jumps.
- **Base_link:** is attached to robot's center.
- **The base_footprint:** is a simple projection of base_link on the ground. It publishes its transform in reference to the base_link.
- **Laser_link:** is the center position of the laser sensor, and its transform is published in reference to the base_link.
- **Base_stabilized:** is the center position of the robot, it publishes its tf in reference to the base_link.

III.3. ROS Navigation Stack

In order to perform ROS Navigation, we need first a map of the environment that we want to navigate (in our case we navigate the hall of CDTA). Second, we need to localize our robot inside the map this is called localization. Then, we will need to do path planning, to calculate plans and going from one point to another in that map while avoiding obstacles. Finally, we need to avoid obstacles that are not shown in this map which is known as obstacle avoidance; these are the four key points in Navigation and will be detailed step by step as illustrated in figure 3.3.

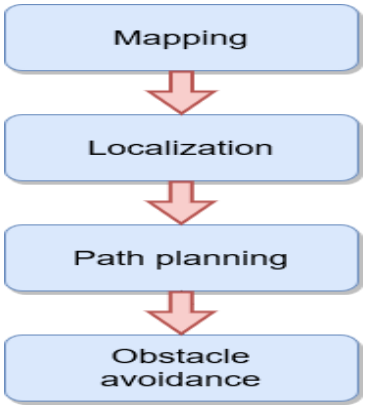

Figure 3.3. ROS Navigation steps.

III.3.1. Mapping

Mapping is the process of using the robot's sensors to create a spatial model of the environment around it. The map is then used for localization and navigation. One of the many resources needed for building a map and completing navigation stack is SLAM systems.

- **SLAM-Map Building and Navigation**

SLAM is concerned with the tasck of building a map of an unfamiliar environment by a mobile robot while at the same time navigating the environment using the map. The term SLAM is an acronym for Simultaneous Localization and Mapping. Hugh Durrant-Whyte and John J. Leonard [19] were the first to develop it.SLAM is a concept rather than a specific algorithm.SLAM consists of a number of phases, each of which can be accomplished using a variety of different algorithm.SLAM may be used in both 2D and 3D motion. There are a number of 2D SLAM algorithms that rely on a laser scanning sensor, such as:

- Hector SLAM Algorithm

Hector stands for **H**eterogeneous **C**ooperating **T**eam **o**f **R**obots. This approach has been published in 2008. Hector SLAM is an open-source method that uses a laser scan sensor (LIDAR) to create a 2D grid map of the surrounding environment. This system uses scan matching to determine the robot's location without any odometry. Figure 3.4 shows flowchart of the Hector SLAM algorithm.

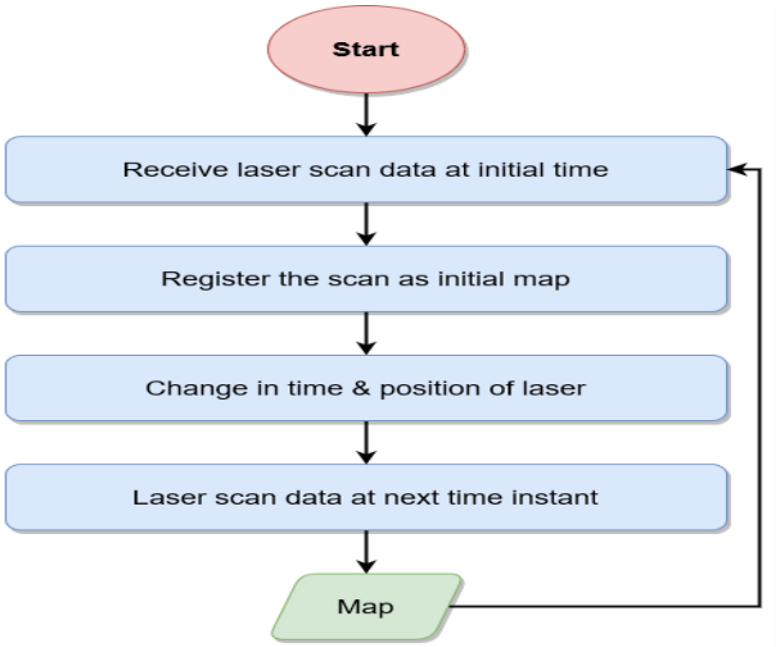

Figure 3.4. Hector Slam flowchart.

- ***- SLAM Gmapping***

Gmapping is a 2007 open source SLAM software in the ros.It is extensively used 2D lidar package.The Gmapping algorithm can be used for locating and mapping both indoors and outdoors.Not only does the gmapping method require 2D lidar data, but it also strongly relies on odometer data.The flowchart for Slam Gmapping is shown in Figure 3.5.

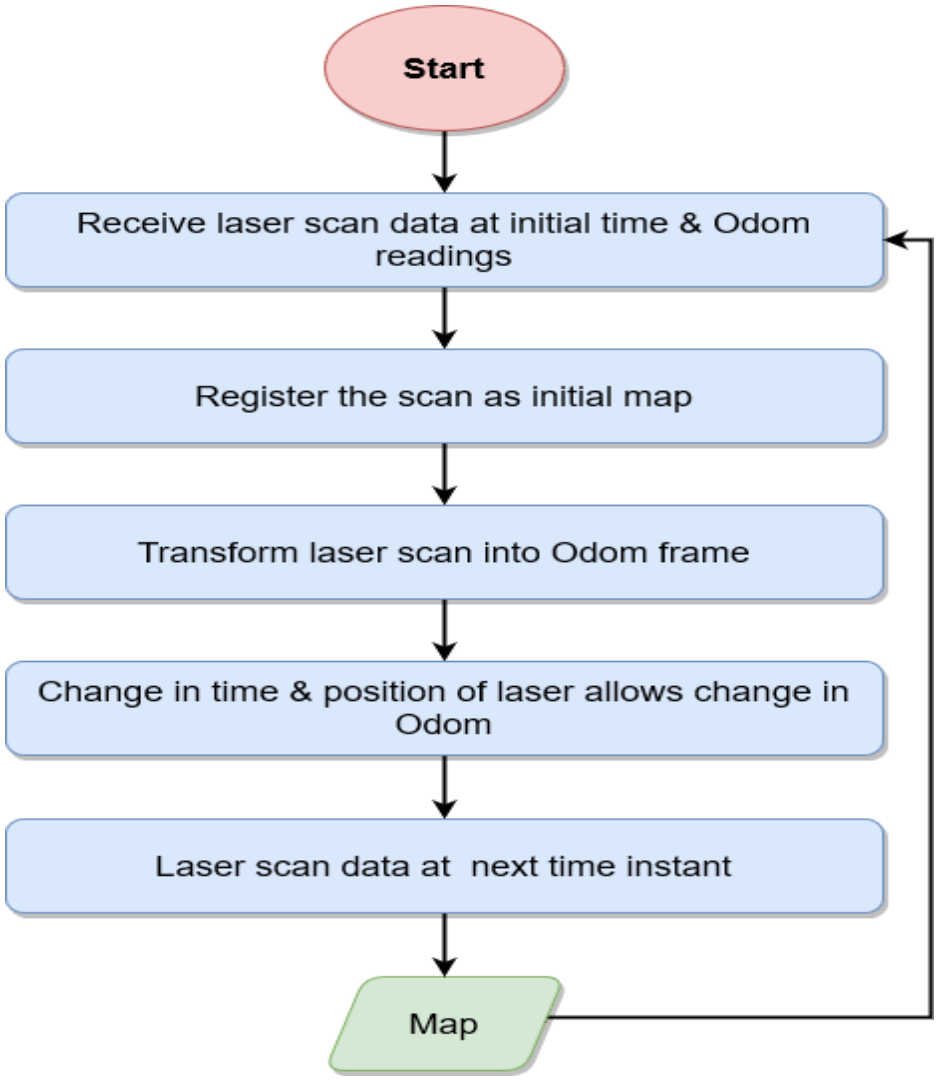

Figure 3.5. Gmapping Slam flowchart.

- ***- Visual-SLAM***

Visual SLAM is a sort of SLAM system that uses 3D vision to perform location and mapping functions. Visual-SLAM is more complex than LiDAR-SLAM because images provide too much information, while distance measurement is challenging. A popular method for Visual-SLAM is to estimate robot motion by matching extracted picture features under

different positions to generate a feature map. Figure 3.6 gives a better understanding of visual slam.

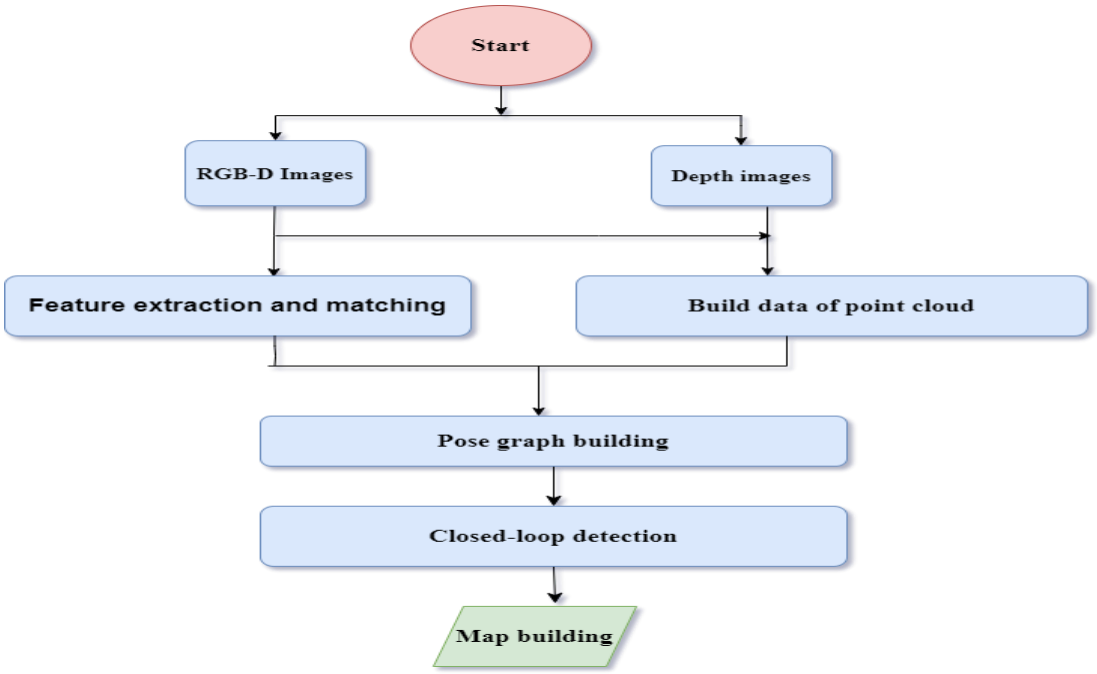

Figure 3.6. V-SLAM flowchart.

III.3.1.1. Mapping using LIDAR

To build a map using rplidar, first we need to:

- **Integrate RPLidar with ROS**

We start by opening a new terminal, we build the RPLIDAR ROS package by cloning the rplidar ros package to the workspace then building the node. Then, we run the package by running the rplidar node and viewing in the rviz. The red lines shown in figure 3.6 are the lidar data that is being published to the ros topic /scan. We changed some parameters (the parameters are found on the left of fig3.7) such as fixed frame to laser. We set the laser scan to topic /scan which is appeared by clicking on the add buttom of the rviz.

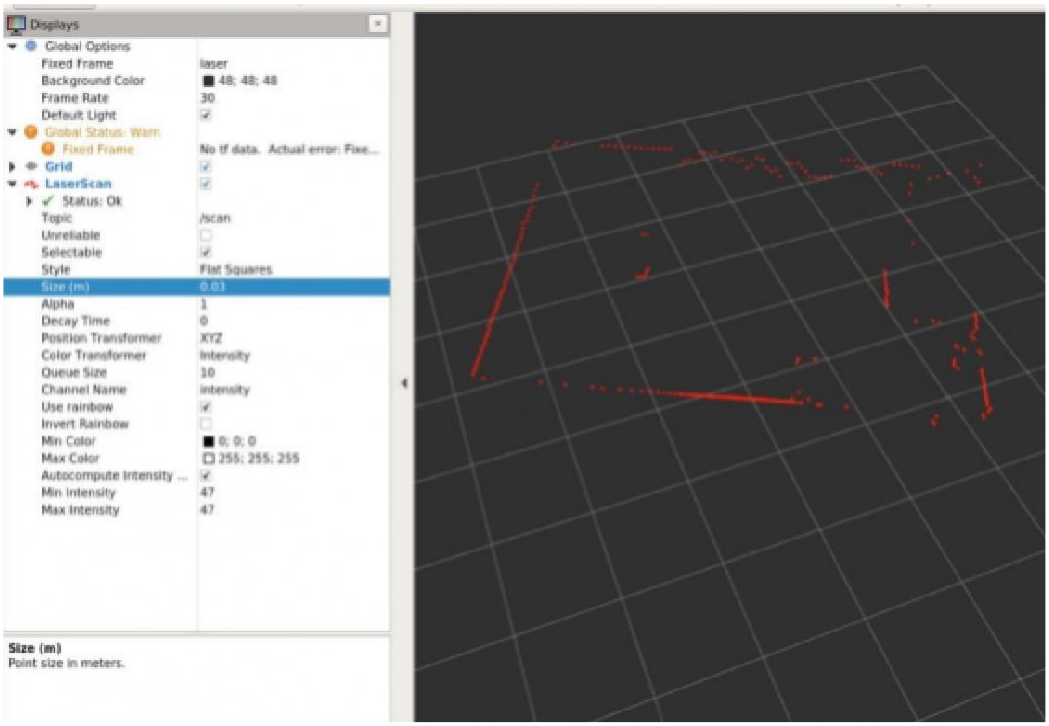

Figure 3.7. LIDAR sensor running on Rviz.

We used the two methods gmapping and hector SLAM to build a map based on RPLIDAR, in order to know which method performs better and it will be discussed in chapter 4.

- **Create a map using Hector slam OR Gmapping**

After running the lidar and confirming that is working, in new terminal we cloned the repository of Hector-SLAM or Gmapping ros packages to our source workspace folder. Then, we built, compile then and launched hector-slam or gmapping-slam nodes; this run rviz directly as shown in fig3.8.

To create a map we move the Lidar around the room but slowly to make sure that the map making is working well; the final map is presented in the following chapter.

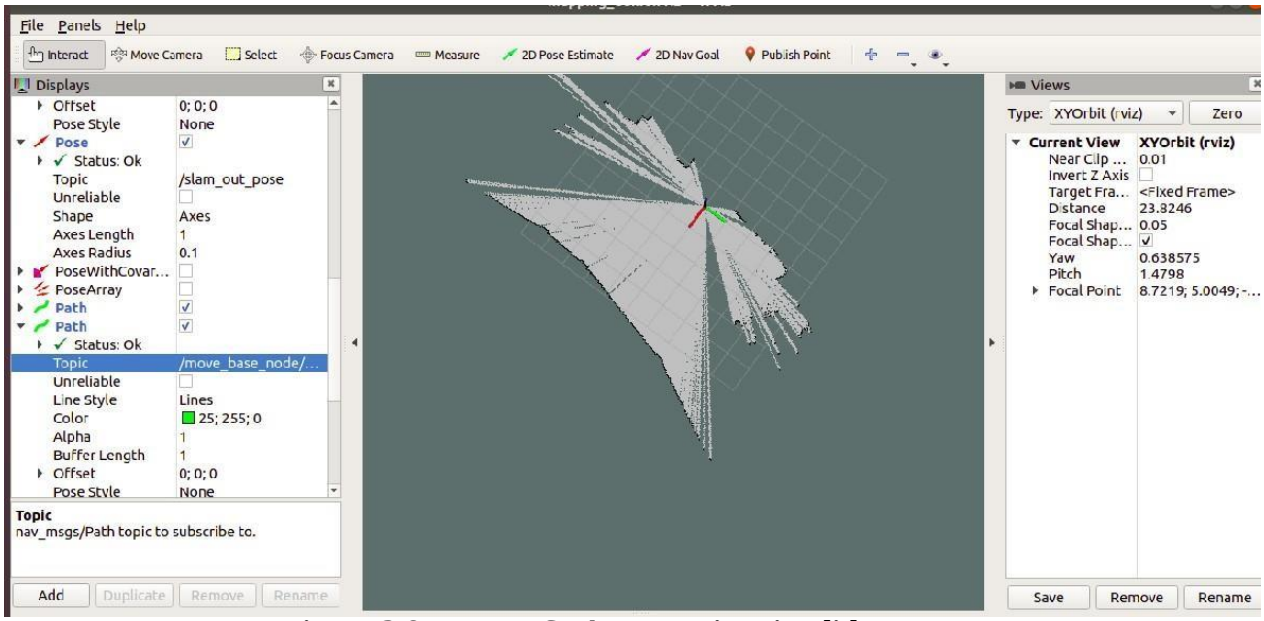

Figure 3.8. Hector-SLAM on rviz using lidar

III.3.1.2. Mapping using D435i Camera

To create a map of the surrounding using Visual SLAM based on depth camera d435i, we need to:

- **Integrate d435i camera with ROS**

We start by opening a new terminal, we cloned the Realsense2-camera ros package to the source folder of our workspace. Then, we build the node. To start the camera node we launched this command : roslaunch realsense2-camera rs-camera.launch

After that, we run rviz and change some parameters. We changed fixed frame to camera_link; we published the topics: /camera/color/image_raw, camera/depth/image_rect_raw and added a pointcloud topic /camera/depth/color/points as it is illustrated in figure 3.9.

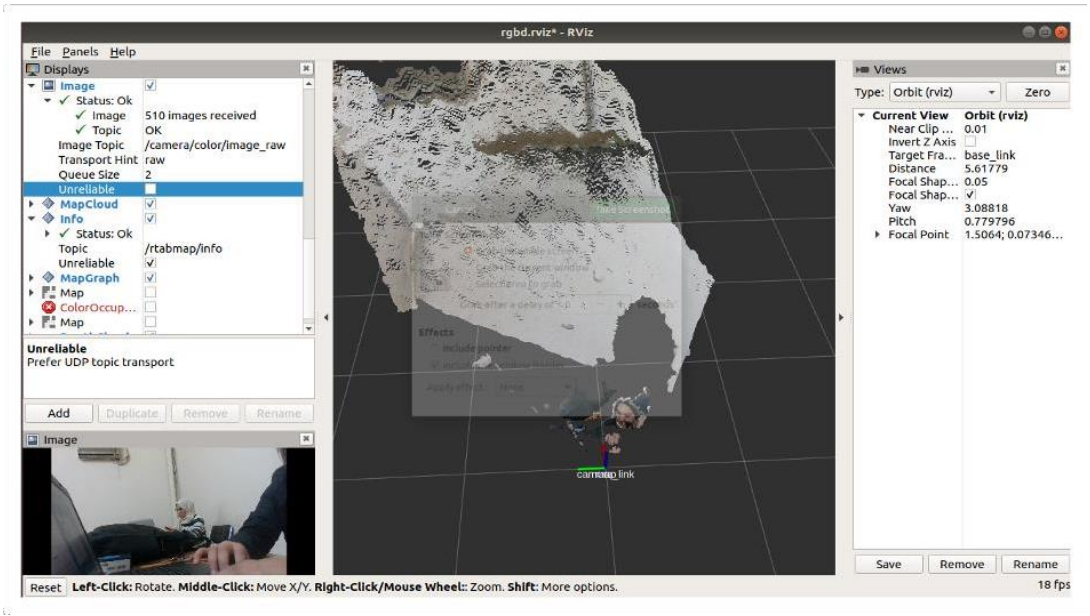

Figure 3.9. D435i camera running on rviz.

- **Create a map using Visual SLAM**

After running camera node on rviz, we installed these packages: imu_filter_madgwick, rtabmap_ros and robot_localization by typing the following commands in a new terminal:

- *sudo apt-get install ros-melodic-imu-filter-madgwick*
- *sudo apt-get install ros-melodic-rtabmap-ros*
- *sudo apt-get install ros-melodic-robot-localization*

We launched the following command and this runned rviz directly

roslaunch realsense2_camera opensource_tracking.launch

We waited a few minutes for the system to fix itself. We left out TF frames as marked only map and camera_link (see figure 3.10).

Figure 3.10. Starting visual slam with d435i camera.

III.3.1.3. Map saving

To perform localization, we need a pre-build map. For saving a map we just run the following command**:** ***rosrun map_server map_saver -f my_map***

To load the saved map we first get the ROS Master started and in a new terminal we type

rosrun map_server map_server my_map.yaml

III.3.2. Robot-Localization

The question of "Where is the robot now?" is answered by robot localisation.The technique of establishing where a mobile robot is in relation to its environment is known as robot localization.A mobile robot having sensors to track its own movement.We utilized the amcl node to execute the localization.

- **AMCL**

AMCL stand for Adaptive Monte Carlo Localization, it is a probabilistic localization system for a robot moving in two dimensions. It uses particle filter to track a robot's position against a known map. The purpose of this method is to determine the position of the robot inside the map of the surroundings.

Once the map of the desired environment is built, we start localization and for that we launch amcl node and rviz. We set laser scan display to topic /scan in order to visualize the laser data. We need to add Map display and choose the map-topic. To visualize localization, we need to add PoseArray Display and configure the topic ParticleCloud. Figure 3.11 Shows how the topics are connected to amcl node.

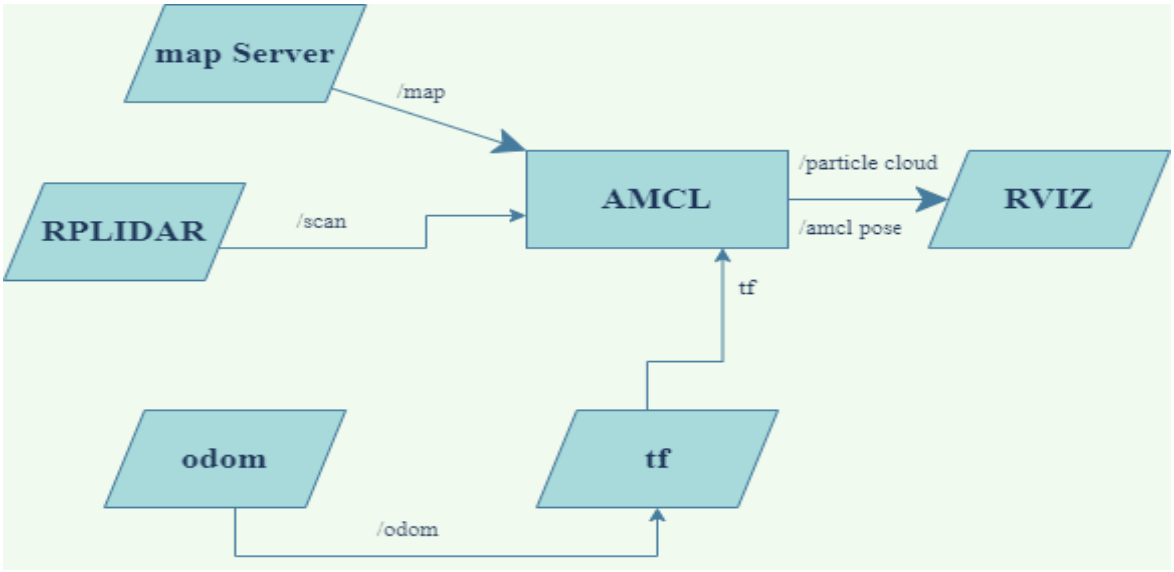

Figure 3.11. AMCL node and topics related

By adding Particle Cloud, we can visualize the characteristics arrows (red arrows) that are used in order to visualize localization (see figure 3.12). We used 2D Pose estimate tool to tell rviz where our robot is.

- **2D Pose Estimate:** allows the user to set the pose of the robot in the world to initiate the navigation stack's localization mechanism. The navigation stack is waiting for a new pose of a new topic named initial pose to appear.

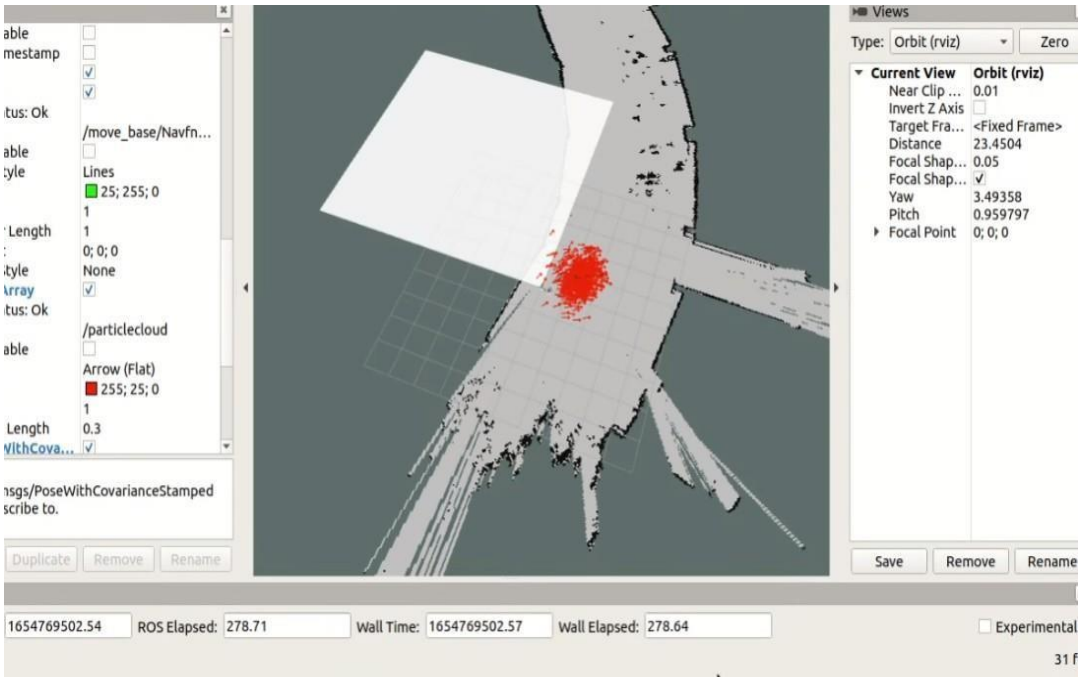

Figure 3.12. Particle localization

III.3.3. Path Planning

The path planning operation provides the answer to the question *"how should I get to where I' am going?"* Path planning can only be applied when a map of the environment is known. It is defined as finding a geometrical path from the current location of the vehicle to a target location such that it avoids obstacles. For visualizing path planning we need three elements:

- Map Display (Cost maps)
- Path Displays (plans)
- 2D Tools

We launched the command of navigation and run rviz; we added two map elements the global costmap and the local costmap as shown in Figure.3.13, which are the main elements in path planning. We added one path element and Pose Array for localization. We set a goal using 2D

Nav Goal so that the global planner calculates the path to reach that goal. The 2D Nav Goal button is used to give a goal position to the move_base node in the ROS Navigation stack through Rviz.

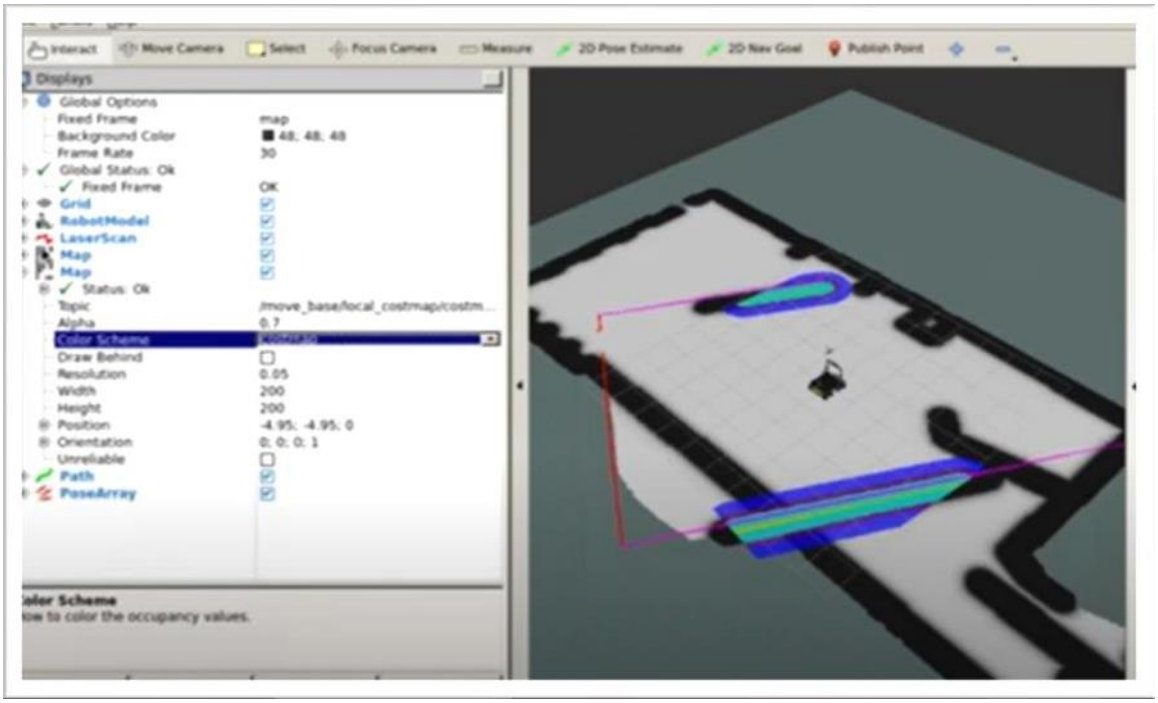

Figure 3.13. Local and global costmaps [20].

The goal position will send to the move_base node for moving the robot to that location; then move_base sends this goal to the global planner which calculates a safe path to reach that goal using global costmap. The path is clearly shown in figure 3.14.

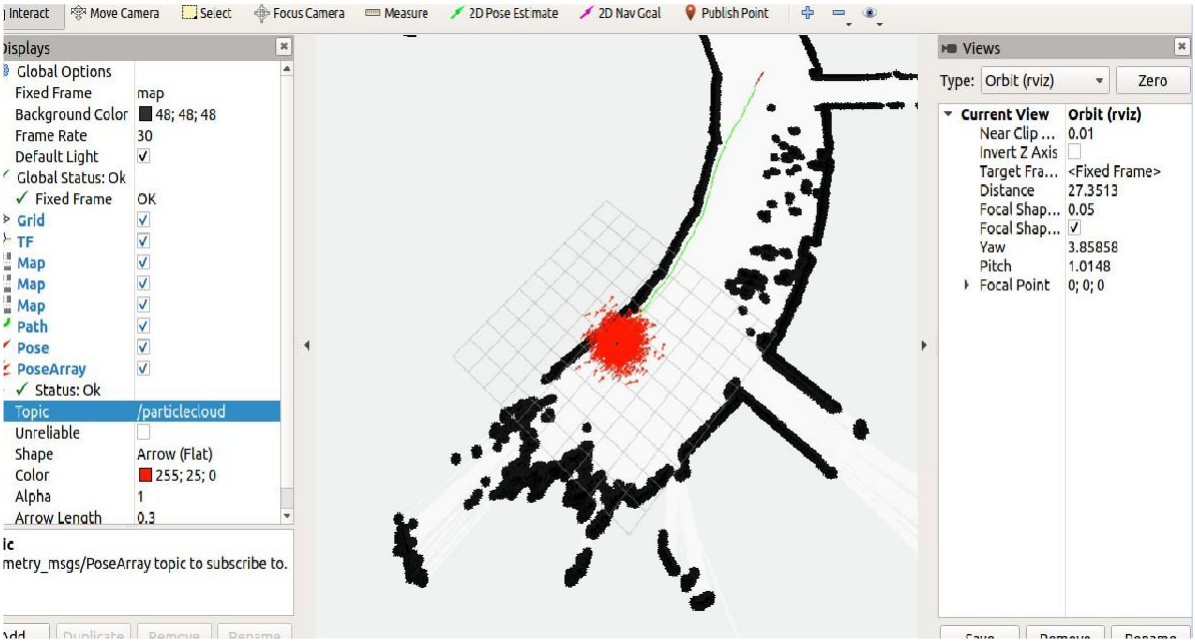

Figure 3.14. path planning

III.3.4. Obstacle avoidance

After creating the path, we will add path for the global planner with topic /move_base/NavfnROS/plan. We sent 2D Nav goal so that the global planner calculates its path. The local planner receives this path from global planner to follow but if some object comes across, local planner will recompute the path in order to avoid it. The local planner monitors the odom and laser data.

We launched rflex to drive the robot by running this command:

roslaunch rflex b21.launch

Then, B21r mobile robot moves taking the velocity published bay move base as an input.

the following command: rostopic echo cmd_vel will show the velocity of the robot as described in figure 3.15.

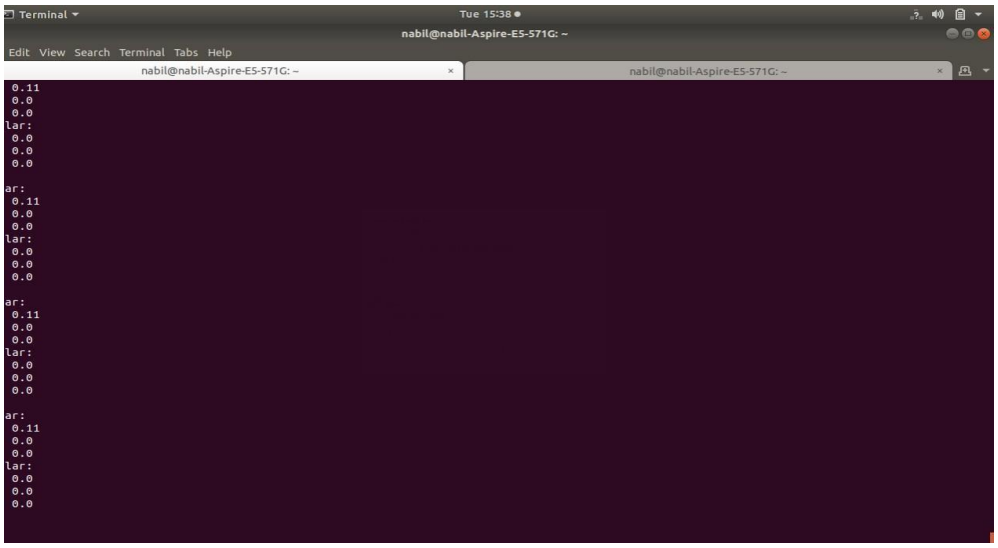

Figure 3.15. the velocity of B21r mobile robot

III.4. CONCLUSION

This chapter dealt with the process of ROS Navigation stack point by point. We have seen how to integrate our lidar sensor and d435i camera in ROS operating system. In the following chapter we will take a look at the results and discuss it.

CHAPTER04

RESULTS AND DISCUSSION

During this chapter, experimental and empirical results obtained from the implementation and the tests of the different approaches are presented. The results will be discussed and evaluated step-by-step from mapping until path tracking.

VI.1. Visual-SLAM

For mapping, two different types of SLAMs: Visual-SLAM (obtained from the camera) and LiDAR SLAM were implemented and tested as explained in the previous chapters. Starting by testing the Visual SLAM, figure 4.1 illustrated the visual map in rviz created from the camera.

Figure 4.1. Map builded using visual SLAM

As it can be noticed from figure 4.1, the map is full of details and not completely clear; moreover, it consumes more time while mapping, and sense the image in Visual-SLAM carries to much information it is more complex. Visual SLAM is currently in the early stages of development, with application scenarios and product landing to come. Laser-SLAM is now the most stable and widely used positioning and navigation system, with great precision when creating maps. For that, we preferred to use Lidar-Slam.

VI.2. Comparing LiDAR SLAMS

After the illumination of the camera, we decided to use LiDAR SLAM for mapping. However, two different SLAMs which are Hector SLAM and Gmapping SLAM can be implemented to create the map. Therefore, in this section, we try to find the main differences between the two methods and select the best one according to the obtained results.

The map built with the help of hector slam is shown in figure 4.2 and the one built using Gmapping is the shown in figure 4.3. Both of the maps present the hallway of CDTA

Figure 4.2. map of the CDTA hallway using HECTOR SLAM

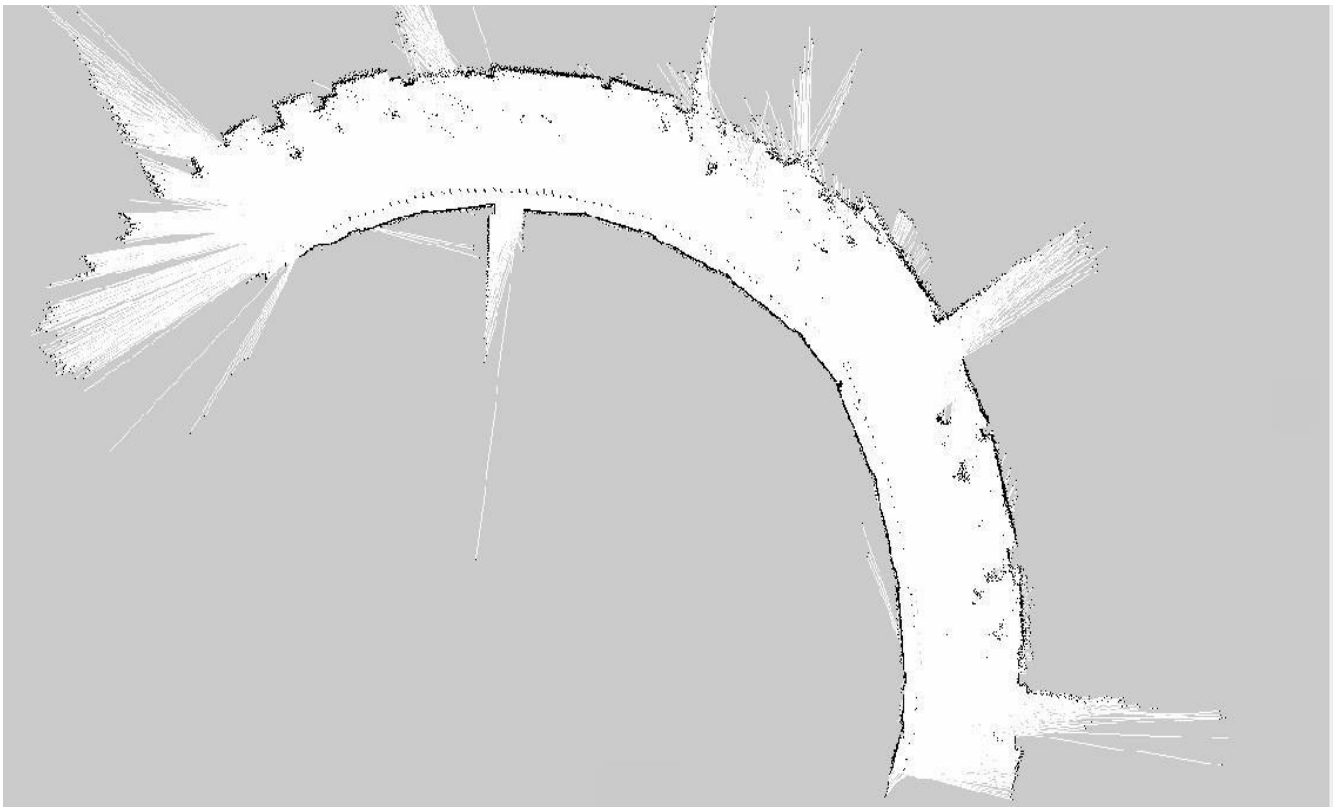

Figure 4.3. map of the CDTA hallway using Gmapping SLAM

The two maps are made using hokuyo LiDAR, which is integrated in the robot b21, by moving in CDTA hallway at the same speed. By comparing the two figures we can say that the accuracy of Hector SLAM is higher than the Gmapping SLAM.

Thus, to be more accurate, the two approaches are compared in term of time consuming and localization accuracy as detailed below.

VI.2.1. Scenario 01: Time Consuming

During the map creation, we were obliged to stop several time to arise the black point in Gmapping; on the other hand, for Hector it was faster, it doesn't require to stop so much. This scenario is illustrated in table 4.1, figure 4.4(a) and figure 4.4(b).

In order to validate this extracted conclusion about time consuming scenario, the two approaches were run to build a map for the same area (CDTA hallway). From table 4.1, it can be noted that Hector SLAM end up the creation of the map in *9min;* while, for Gmapping it took around *12min.* The extra time spend by Gmapping was devoted to the time expend by the algorithm to fill the blank area in the map and to arise the black point that appears in the map correspond to fake obstacles. The delay time is variant as shown in table.4.1 and that is related to several reasons like:

(i) The black point that appears in the map which are fake obstacles and that can be seen for hector slam in the 4th min, 5th min and the 6th min which took 23sec, 19sec, and 21sec respectively, and for Gmapping SLAM that happens in the 2nd ,3rd ,5th, 6th, 8th and 9th minutes which took 33sec, 24sec, 20sec 22sec, 27sec and 33sec respectively.

(ii) The gap in the map, which happens in the 1st, 2nd, 3rd, 8th and 9th minutes for hector and took 10sec, 11sec 12sec, 7sec and 11sec respectively, and for Gmapping appears in the 1st, 7th, 10th, 11th and the 12th minutes and took 12sec, 10sec, 22sec, 14sec and 16sec respectively.

(iii)The stop in the 7th min for hector which spends 33sec and in the 4th min for Gmapping which takes 47sec is explained by the people walks around; so, the process need to stop and rotate the robot left and right in order to arise their trace from the map. In case this step is not done, it may cause fake obstacles later.

Table.4.1 number of times that we need stops while mapping for each minute

time(min)	1	2	3	4	5	6	7	8	9	10	11	12
Hector (sec)	10	11	12	23	19	21	33	7	11	0	0	0
Gmapping (sec)	12	33	24	47	20	22	10	27	33	22	14	16

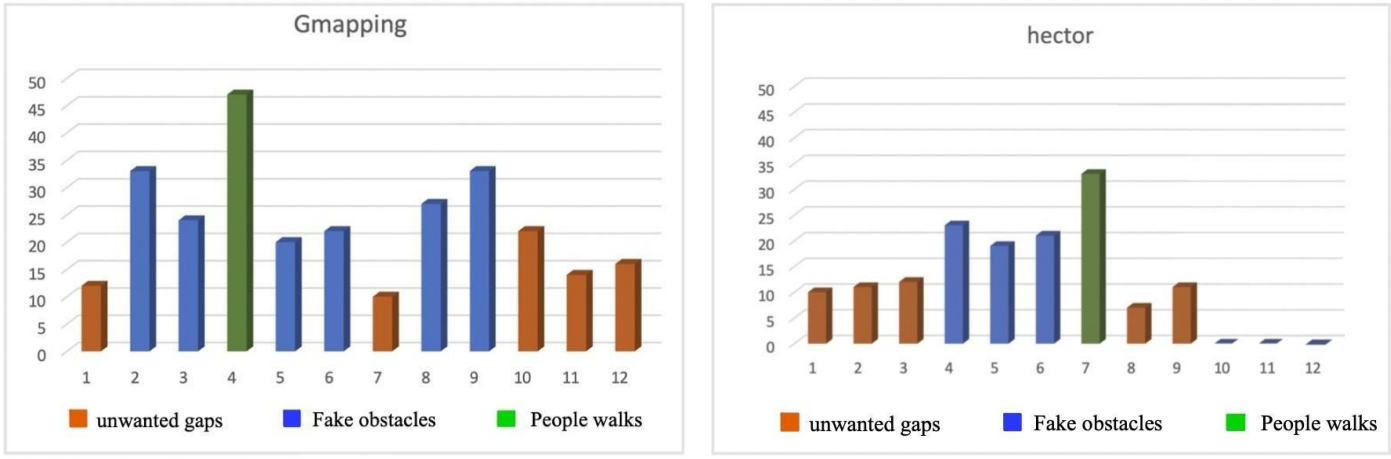

Figure 4.4. Time stops needed during each minute to eliminate errors while mapping. (a) Delay generated by Hector. (b) Delay generated by Gmapping.

Figure 4.5 shows the different problem that can appear while mapping. So, we can say that Gmapping creates more fake obstacles and took more time to arise them; thus, it is more time consuming.

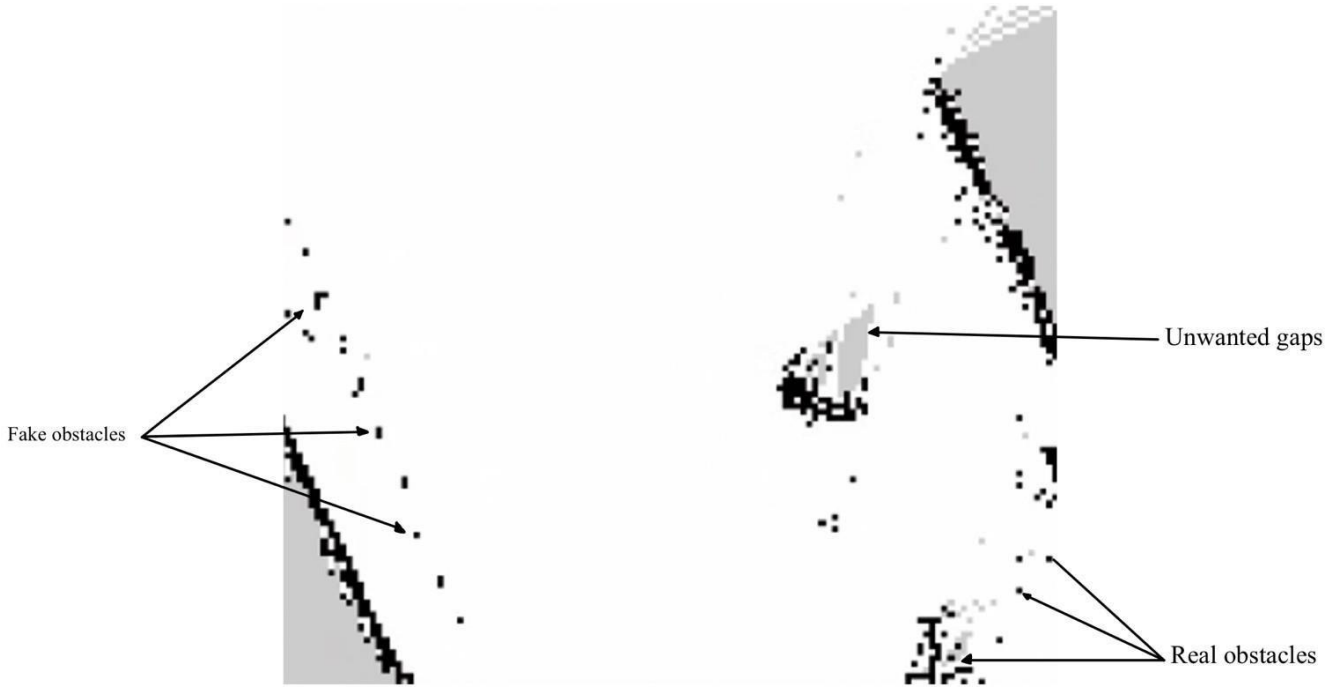

Figure 4.5. different errors may happen when mapping

VI.2.2. Scenario 2: localization in SLAMs

In this scenario, we try to know which slam is more accurate in localization while mapping to do that we specify three different points in the ground of the hallway and we mark another point in the wall and measure the distance in realty and through rviz for Hector and Gmapping SLAMs. The obtained results are shown in table 4.2 for Hector and in table 4.3 for Gmapping.

Table.4.2.-a: Hector slam localization

	Real world(m)	Rvis (m)	error(m)
p1	1.23	1.35	0.12
p2	1.77	1.90	0.13
p3	2.23	2.36	0.13

Table 4.3: Gmapping localization

	Real world(m)	Rvis (m)	error(m)
p1	1.23	1.4	0.17
p2	1.77	1.96	0.19
p3	2.23	2.46	0.23

So, as it can be noticed the error for localization in Gmapping is with an average of *0.1966 m,* higher than the hector's error which is around *0.1266 m*.

This result was not waited because GMapping combines odometry and laser scan as information sources; whereas, Hector Slam just employs laser scan. GMapping should theoretically outperform Hector Slam. Therefore, according to these tests we decided to work with Hector SLAM because is more accurate in map drawing, more accurate in localization and less time consuming (so is less energy consuming).

Other problems were also faced in both the slams such as: when the robot rotate with high speed or do an abrupt movement, a glitch (errors) appears and that affect the quality of our map and affect the estimated pose (localization). These glitches are shown in Figure.4.6.

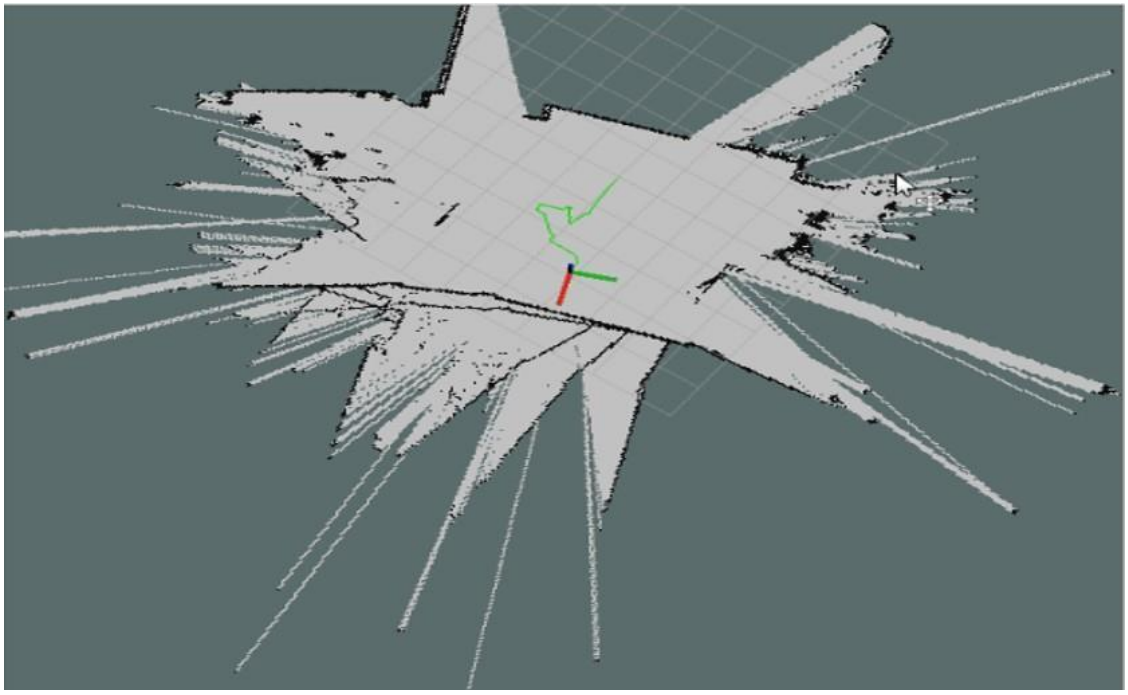

Figure 4.6. Error caused by sudden moves.

VI.3. Localization (AMCL)

Because of the previously met and mentioned problems with SLAMs, AMCL algorithm is chosen for localization. AMCL is the most famous and usable one and almost all the previous master projects were based on it. Moreover, and in order to confirm our choice, we tested the AMCL. As with the two previous scenarios, three points are selected from the map and measure the distance between them and a specific point that we chose in the wall in reality and compare it with what AMCL give us in rviz.

First, AMCL, map server and the hokuyo Lidar package were launched. It appears in the map that the robot is at the initial pose from where mapping starts. Thus, a 2D estimate pose is

returned and this period is recognized by the high number of particle clouds as shown in figure 4.7(a). After that, the robot moves and the particle clouds reduces to very small number which means that the robot is well localized as illustrated in figure 4.7(b). The test results are shown in table 4.4.

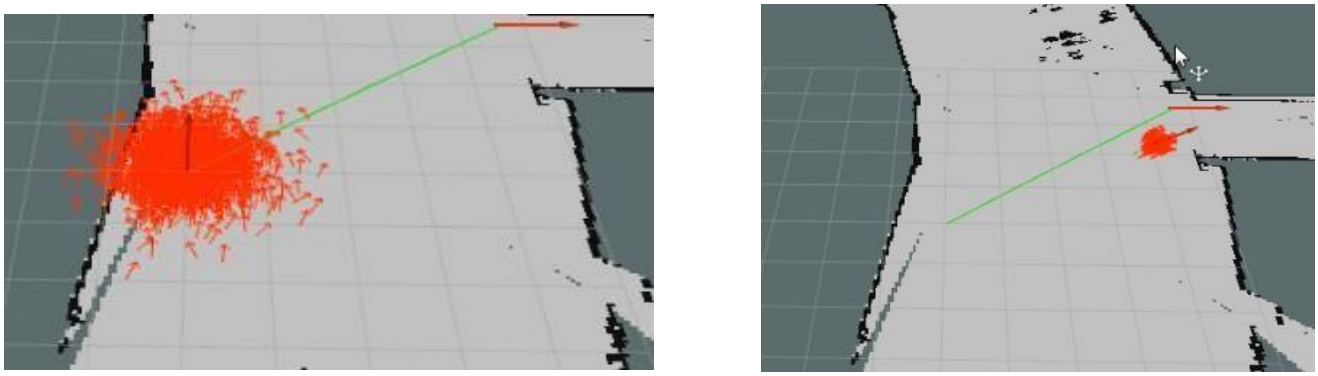

Figure 4.7. AMCL localization results. (a) Cloud in starting pose. (b) Cloud in final pose.

As we can see the error did not exceed 0.12 meters and we consider it as a good result with average precision of 94.4%. AMCL also can inform us about the direction of the robot and it looks very accurate and we can deduce that easily by comparing the direction that AMCL gives us in rviz and the direction of the robot in real word.

Table 4.4: The error between real localization and AMCL results

	Real world(m)	Rvis (m)	error(m)
p1	1.50	1.60	0.10
p2	1.89	2.01	0.12
p3	2.52	2.64	0.12

VI.4. Navigation stack

So, after we build our map and test our localization algorithm and it met our needs; now, we move on to our final step which is navigation. Navigation stands to path planning and path tracking. To deal with navigation, our configuration files: Costmap Configuration, Common Configuration, Global Configuration, Local Configuration and Base Local Planner Configuration which are in yaml file form must be created.

The move base node of the ROS Navigation Stack will use the configuration files. Behind the scenes, the move base node is in charge of arranging a collision-free path for a mobile robot from the starting pose to the goal location.

Now after creating our configuration files and creating our map and testing our localization algorithm, it is time to add all of that in our launch file to start navigating and record some results. Also for the launch file, in the beginning we were using RPLidar; so, RPLidar package was needed and used. However, RPLidar was damaged during the tests; thus, obliged us to replace it by Hokuyo Lidar. for the rflex node we launched it separately not included in the launch file.

VI.4.1Testing our navigation stack

By launching the launch file and define the robot's goal pose in rviz; the path is correctly planned and everything seems to be well configured in the launch reflex. The robot starts moving and stopping alternatively, it was a random movement with a random speed and random stops. Here where our RPLlidar was broken because the robot rotates at a high speed and the Lidar was not well fixed on the robot's chassis; so, it fell down and damaged, and we were obliged to change the Lidar by hokuyo.

VI.4.2. Random move and interrupted speed of the robot

By remarking this weird act in the movement of the robot, we were obliged to figure out the problem. To evaluate the angular and the linear velocity generated by the move base, constant

velocities are set to the robot then record its behavior corresponding to each velocity; angular and linear velocity are tested separately. The results are shown in figure 4.8.

(i) The robot's behavior responding to the angular velocity=0.1m/s is shown in figure 4.8(a). It can be clearly notified that the movement is not normal it is interrupted randomly; the robot rotates and stops at a random time.

(ii) The robot's response to a constant linear velocity of 0.1 m/s is shown in figure 4.8(b). It can be also notified that the movement is not normal it is interrupted randomly; the robot moves and stops randomly.

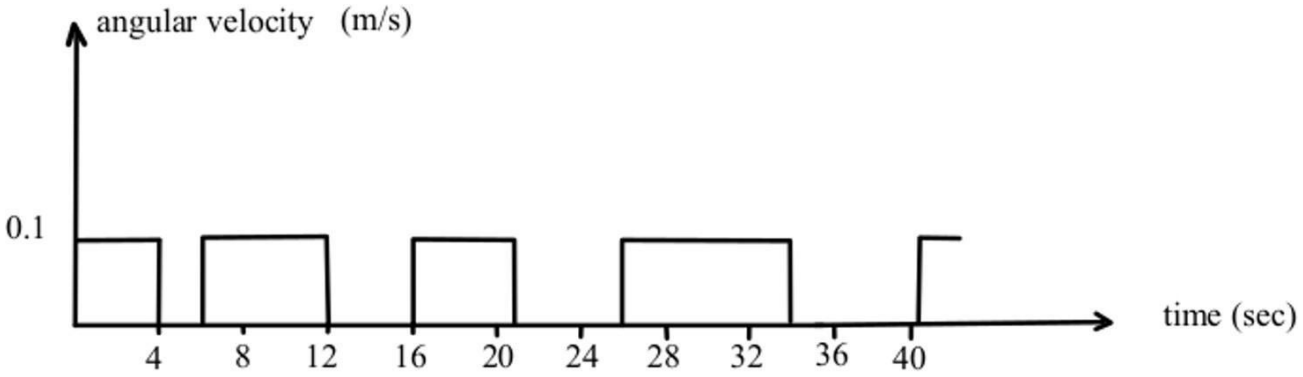

Figure.4.8-a. random response with constant angular velocity=0.1m/s

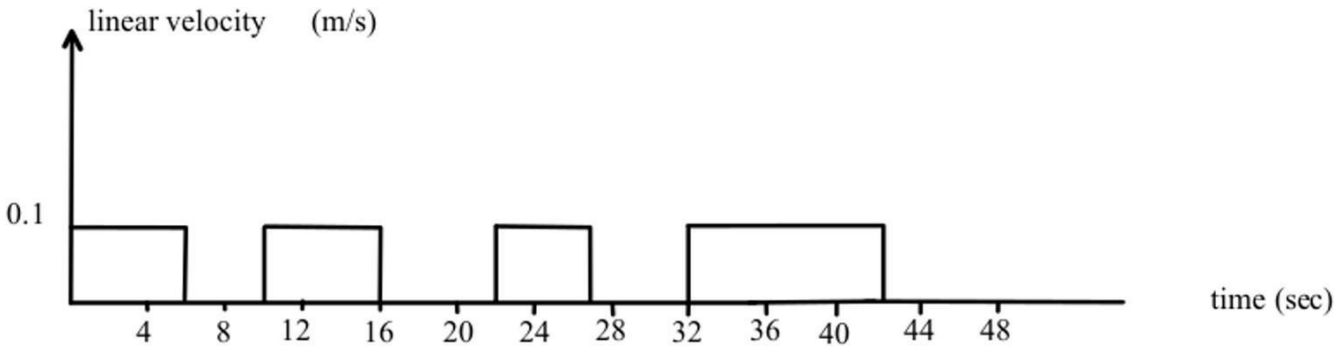

Figure.4.8-b. random response with constant linear velocity=0.1m/s

VI.4.3. serial sniffer interceptty

To solve the previously mentioned problem, we thought in Interceptty. Interceptty is a program that sits between a serial port (or another terminal device, or a program, or a socket, or something attached to a file descriptor) and a program that communicates with it. It records everything that happens between the two. It accomplishes this task by first opening the real device, generating a pseudo-tty; then, forwarding all data between the two and recording anything it sees.

It includes several settings that allow us to fine-tune the devices it utilizes; as well as, the terminal options for the actual device.

After the installation of interceptty, the velocity test was performed another time. As illustrated in figure 4.9 (a) and figure 4.9 (b), the results are satisfactory and the robot rotates and moves without abrupt stops.

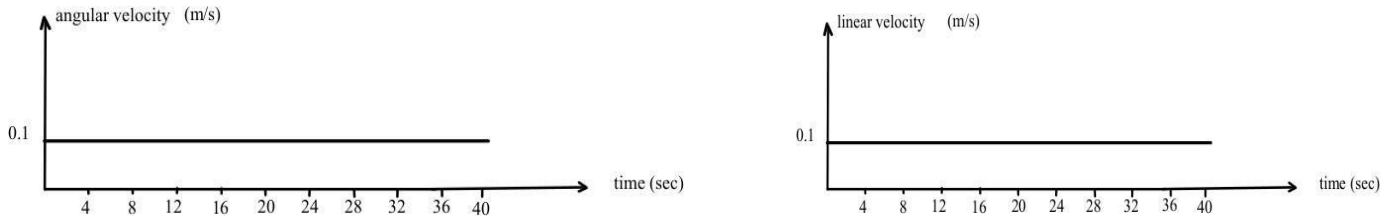

Figure 4.9. Normal robot's response after the setup of interceptty. (a) Response to a constant angular velocity of 0.1m/s. (b) Normal response to a constant linear velocity of 0.1m/s.

The problem was solved, we come back to test our navigation stack; the robot now moves with the velocity generated by move base as an input. After launching our launch files rflex and interceptty, we remark that the robot is not interrupted anymore. However, the robot is not able to track the pre-planned path correctly; it deviates left and right from the path. Nevertheless, it still able to reach the goal with an error in the interval of *0.1m* to *0.13m* measured in rviz.

VI.4.4. Robot wheels' response

this time we face a mechanical problem and that because one wheel from the four wheels of the robot was not response well in order to confirm that we test them manually and the test result are shown in table 4.5.

Table 4.5: Manual test of the wheels' movement.

	Rotation w1	**Rotation w2**	**Rotation w3**	**Rotation w4**	**Final rotation**
	W_1 +60°	W_2 +60°	W_3 +60°	W_4+60°	240°
W_1	60°	60°	0°	60°	180°
W_2	60°	60°	0°	60°	180°
W_3	43°	35°	60°	31°	169°
W_4	60°	60°	0°	60°	180°

As illustrated in table 4.5, wheel W_3 is not responding correctly to the desired rotation and the error varies from 17° to 29° for each robot's base rotation. Once one of the wheels W_1, W_2 or W_4 is rotated, the wheels and the robot's base rotate at the same angle. Whereas, only W_3 will not rotate with the same angle as the others. When, W_3 is rotated, it rotates alone (freely); so, the problem was there which generate a wrong path tracking.

This problem generated a path tracking error therefore the real path followed by the robot deviates from the preplanned path. The error's marge between the real path and the planned one is computed at discrete points. The distance from the starting pose to the target pose of the two paths is also measured (the preplanned path is *5.75m* whereas the path executed by the robot is *6.52m*). These results are summarized in figure 4.10 below, the green path present the preplanned path and the black one is the path tracked by the robot. The robot has to move for an additional distance of *0.77m* which is undesired. An error of 13.4% is generated by this problem which present unsatisfactory results.

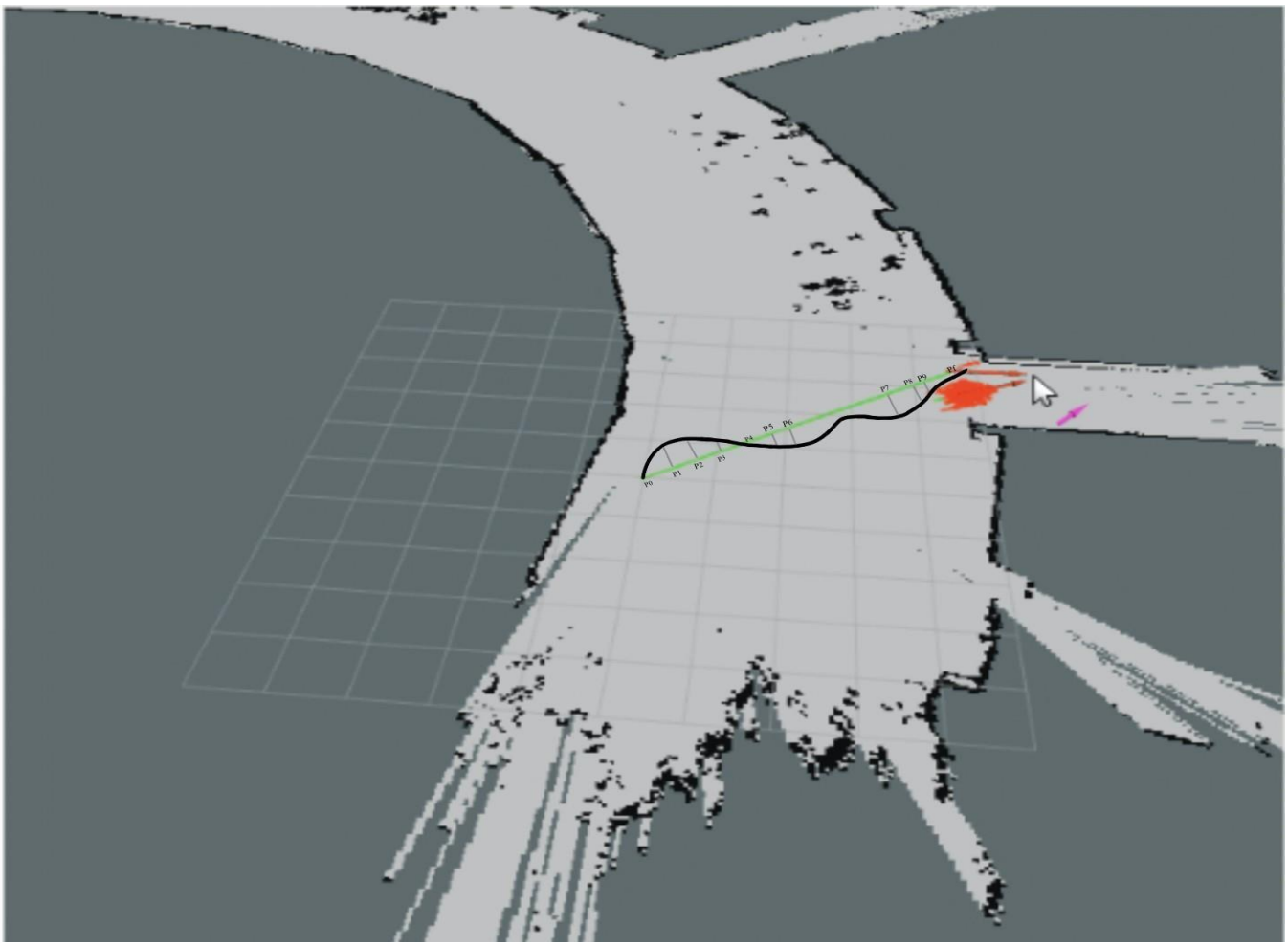

Figure 4.10. original path vs real path and the point chosen for test

The error generated from the poor response in wheel W_3 rotation is presented in figure 4.11. It is clearly appearing that the error is random but the robot still able to reach its destination with an error of around *0.12m* which can be consider as an acceptable result.

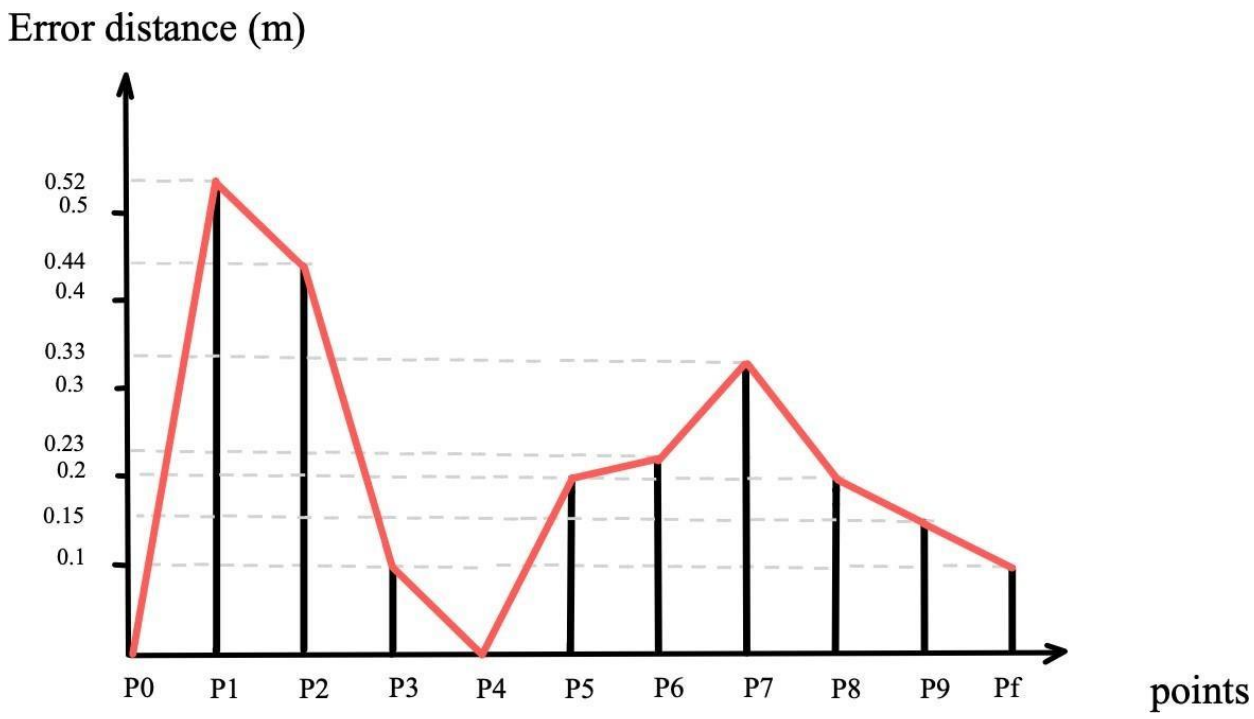

Figure 4.11. distance error of several points in real path from original path.

VI.5. Conclusion

This chapter summarized the different steps followed to implement our navigation strategy starting from map building and localization to navigation. Our work was validated by some experiment results and discussions. For map creation different slams were used, and by comparing them hector overcome Gmapping performance which was not theoretically expected. After that, AMCL was tested for localization and the results were fairly good. Therefore, move base node which contains the configuration file in yaml form was created; then, our launch file which gather all the nodes was generated too. During our work we faced some problems like the serial port was not working well which was solved by setting up interceptty sniffer. In addition to the problem with one of the robot's wheels which didn't respond correctly to the assigned rotation and causes a considerable path tracking error. Unfortunately, this last problem was not solved because it requires more time and it is out of the scope of our project. Although the robot did not follow the preplanned path correctly, it still able to reach the navigation goal with small error.

CONCLUSION

Conclusion

The goal of this work was the implementation of ROS Navigation Stack on the B21r mobile robot equipped with Lidar sensor and RGB-D camera.

Initially, we have presented a brief overview of mobile robots, ROS and ROS Navigation stack and we talked about basic components of mobile robots in general. We presented the B21r mobile robot. Afterward, we described the Lidar sensor and d435i camera and their connections to our laptop. We detailed the ROS Navigation step by step. After that, a map of the hall of CDTA was required and it was created. Then, our robot localized itself inside the map. Next, a free path was planned to enable the robot to move from one point to another in the created map while avoiding obstacles.

We have implemented the navigation stack in ros melodic. Different results were obtained. We used both RGB-D and LIDAR sensor to build a map of the hall of CDTA, but according to our experiment results we preferred to work with Lidar sensor. Then, we created a map using Hector and Gmapping slam. When we were creating our map, glishes appeared after fast rotation and sudden stops of our B21r robot. Therefore, this affect the quality of our map and this problem is generated when implementing the hector and gmapping slam methods.

By comparing Hector and Gmapping results we concluded that: mapping using Hector is more accurate and less time consuming and less fake obstacles compared with Gmapping slam. Hector slam is more accurate in localization since the average error for the robot to localize itself using gmapping was 0.1966m which was higher than hector's error (hector's average error was 0.1266 m). For that, the best choice was to implement hector slam.

After that, we used AMCL for localization, because the error did not exceed 0.12m comparing to Hector localization. Also, AMCL is very accurate by comparing the direction of the robot in real world and in Rviz. Then we started navigating our environment and we were obliged to work with hokoyo Lidar for the reason that our RPLidar is broken during our tests.

The path created looks good in Rviz but in reality our robot did not follow that path and the movement of the robot is interrupted. We tried to find the problem and we gave constant velocity to the robot and we tested the angular and the linear velocity of each wheel separately. We deduced that the problem was that the new generation of our laptop could not communicate with the B21r mobile robot, hence we fixed the problem by using a serial sniffer interceptty. After that, our robot movement was not interrupted but did not follow the path

correctly, we still having errors. We tested the four wheels of the robot, Hence the results confirmed that one wheel from the four wheels is not responding well. For that the problem was mechanical and it takes time to fix it.

For future work, we will first fix the problem of the wheel so that our robot will be able to follow the planned path correctly. In addition, realizing a new surveillance mobile robot then implementing on it all the required tools for mapping, localization and navigation basing on ROS is a good perspective because the B21r mobile robot of CDTA is very old.

References and Bibliography

[1] Bruno Siciliano, Oussama Khatib (Eds). Springer Handbook of Robotics (Springer Handbooks: Springer-Verlag Berlin Heidelberg 2008), page number: 01.

[2]JeremyNorman's,historyofinformation.com.8/14/2021.https://www.historyofinformation.com/detail.php?entryid=4071.

[3] Heiserman. David (1976). Build Your Own Working Robot. TAB Books. ISBN 0830668411.

[4] Heiserman, David (1979). How to Build Your Own Self-Programming Robot. TAB Books. ISBN 0830612416

[5] Heiserman, David (1981). Robot Intelligence with Experiments. TAB Books. ISBN 0830696857.

[6] Mobile robots are making their move. Control Engineering Europe covering control, instrumentation, and automation systems worldwide. (n.d.). Retrieved June 5, 2022, from https://www.controlengeurope.com/article/189861/Mobile-robots-are-making-their-move.aspx.

[7] *Lidar is set to drastically change the world and how we drive. here's how it works*. Alphr. (n.d.). Retrieved June 6, 2022, from https://www.alphr.com/technology/1006536/what-is-lidar-how-it-works/

[8] Rajeev Thakur, Infrared Sensors for Autonomous Vehicles, Recent Development in Optoelectronic Devices. Rutry Srivostava, IntechOpen, December 20th,2017.

[9] *Depth camera d435i*. Intel® RealSense™ Depth and Tracking Cameras. (2021, June 17). Retrieved June 5, 2022, from https://www.intelrealsense.com/depth-camera-d435i.

[10] Quigley, et al, ROS: an open-source Robot Operating system, ICRA workshop on open-source software, vol. 3, no:3,2,5,2009.

[11] Ademovic A, An introduction to robot operating system: the ultimate robot application framework, May 10,2018, Toptal.

[12] Aleksandar Zivkovic, Development of Autonomous Driving using Robot Operating System, Master thesis, Madrid, May 2018.

[13] Tellez R, how to start with self-driving cars using ROS, May 10,2018. The construct.

[14] Guimarães, R. L., de Oliveira, A. S., Fabro, J. A., Becker, T., & Brenner, V. A. (1970, January 1). *Ros Navigation: Concepts and tutorial*. SpringerLink. Retrieved June 21, 2022, from https://link.springer.com/chapter/10.1007/978-3-319-26054-9_6

[15] *SLAMTEC A2M8 Rplidar A2 low cost 360-degree Laser Range Scanner instructions*. Manuals+. (2021, September 23). Retrieved June 21, 2022, from https://manuals.plus/slamtec/a2m8-rplidar-a2-low-cost-360-degree-laser-range-scanner-manual

[16] *Intel® realsense™ depth camera D400-series Datasheet*. (n.d.). Retrieved June 21, 2022, from https://www.mouser.com/pdfdocs/Intel_D400_Series_Datasheet.pdf

[17] Joseph, L. (n.d.). Ros Robotics projects. O'Reilly Online Learning. Retrieved June 22, 2022, from https://www.oreilly.com/library/view/ros-robotics-projects/9781783554713/ch01s04.html

[18] ROS.org, How to set up hector_slam for your robot (2015), http://wiki.ros.org/hector_slam/ Tutorials/SettingUpForYourRobot. Seen 03 Jan 2015

[19] Joseph, L. (n.d.). Ros Robotics projects. O'Reilly Online Learning. Retrieved June 22, 2022, from https://www.oreilly.com/library/view/ros-robotics-projects/9781783554713/ch01s04.html

[20] Dost, A. B. (2020, December 8). Local vs global costmap. The Construct ROS Community. Retrieved June 22, 2022, from https://get-help.robotigniteacademy.com/t/local-vs-global-costmap/6752