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

SECTOR: Automatic

OPTION: Automation & Industrial Computing

SAIED Boufateh Lahcen & **HASSASA Mohamed Badredinne**

Theme

Implementation of a Perception Algorithm for a Mobile Robot Using ROS

Jury members:

BELKHEIRI Mohammed	PR	President
REGUIEGUE Mourad	MCB	Examiner
OUBBATI Brahim Khalil	MCB	Supervisor
AHMINE Yacine	MCB	Co-Supervisor

2021 / 2022

Abstract

The purpose of this master project is to implement a perception algorithm for a Pioneer mobile robot using the ROS platform. The introductory segment of this memory report talks about the generalities of ROS, mobile robots, and the SLAM algorithm. Firstly, we use the Gmapping algorithm to build a 2-D map of an unknown environment and estimate the trajectory of the robot.Secondly, we tested experimentally another SLAM technique (RGB-D) using the Kinect sensor to build a 3-D map.The simulation and experimental results show that the implementation of the perception algorithms is better in terms of building 2-D and 3-D maps and navigation.

Keywords: Robot Operating System (ROS) , Simultaneous Localization And Mapping (SLAM), Mobile Robots, Pioneer 3-AT/DX , Webots, kinect.

Résumé

L'objectif de ce projet de master est d'implémenter un algorithme de perception pour un robot mobile Pioneer utilisant la plateforme ROS. Le segment d'introduction de ce rapport sur la mémoire parle des généralités de ROS, des robots mobiles et de l'algorithme SLAM. Tout d'abord, nous utilisons l'algorithme Gmapping pour construire une carte 2-D d'un environnement inconnu et estimer la trajectoire du robot. Dans un second temps, nous avons testé expérimentalement une autre technique SLAM (RGB-D) utilisant le capteur Kinect pour construire une carte 3-D. La simulation et les résultats expérimentaux montrent que la mise en œuvre des algorithmes de perception est meilleure en termes de construction de cartes 2D et 3D et de navigation.

mots clés: Système d'exploitation du robot (ROS) , Localisation et cartographie simultanées (SLAM), Robots mobiles, Pioneer 3-AT/DX , Webots, kinect.

ACKNOWLEDGEMENTS

We would like to thank our supervisor Dr. OUBBATI Brahim Khalil and co-supervisor Dr. AHMINE Yacine for their consistent support and guidance during the running of this project, and thanks to LTSS Lab for providing helpful tools used in this project.

We would like to thank our families for all the support they have given us throughout our university journey.

We would like to thank the professors of the Department of Electronics for their great efforts and motivation over the past two years.

Contents

List of Figures

List Of Abbreviations

ROS Robot Operation System

LTS Long Time Supported

P-3DX/AT Pioneer-3DX/AT

SLAM Simultaneous Localization and Mapping

Gmapping Gridmapping

RGB-D Red Green Blue - Depth

ORB Oriented Rotated BRIEF

DSO Diract Sparse Odometry

V-LOAM Visual-Lidar Odometry and Mapping

LIDAR Light Detection and Ranging

GENERAL INTRODUCTION

Humans were successful in creating the mechanisms and tools required to aid them in their progress and make life easier with the beginning of the Industrial Revolution. Robots were one of the main inventions. They are generally used to serve or help people in several fields, from daily life to industrial applications. Robotic science has developed significantly due to the demands of people and industry recently.

The main objective of this thesis is the implementation of a perception algorithm for a mobile robot. In fact, in this work, we have used a classical SLAM algorithm to build a 2-D, 3-D map and estimate the position of the mobile robot. This memory is organized as follow:

The first chapter entitle: *'Platform on ROS, Mobile robots and the SLAM '*, we have discussed the ROS platform and advantages. Then we cross over to some concepts related to mobile robots and some main classifications. Moreover, we talk about the SLAM problem and the three based algorithms and some SLAM applications.

The second chapter entitle : *' Pioneer robot family and Slam techniques'*, We deeply understand the pioneer mobile robot family and mention some hardware specifications and mathematical models. Moreover, we have applied two main control strategies to drive the robot in an unknown environment. After that, we discuss some mainly used SLAM approaches which are based on LIDAR and VISION to solve the SLAM problems.

The third chapter entitle : *'Simulation Results '*, In this chapter, we considered the webots as a platform of the simulation in order to test and evaluate the perceptions algorithms are applied in the work.

The fourth chapter entitle : *'Experimental Results '*. Presents the experimental validation of the RGB-D SLAM technique using the Kinect sensor. We also got an

opportunity to combine our work with another team who works on the trajectory of the robot.

Chapter 1

Platform on ROS, mobile robots and SLAM

1.1 Introduction

Recently, building a map of the unknown environment and localizing the position of the robot are the major tasks of mobile robots. In this chapter, We talk about the Robot operation system (ROS), its history, and its main concepts. then, we talk about the main parts, classification, and types of mobile robots. finally, we discuss the SLAM problem and its based algorithms and some application of SLAM.

1.2 Robot Operation System ROS

1.2.1 Definition

ROS is a meta-operating system for robots that is open-source. It includes hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management, among other services needed by any operating system. It also includes tools and libraries for downloading, creating, writing, and running code on many computers, which aids in the creation of robot software.

1.2.2 The ROS history

The ROS project began in 2007 at Stanford University, founded by several PhD students Eric Berger and Keenan Wyrobek, who were working in Kenneth Salisbury's robotics lab. They saw that many of their colleagues were struggling with robot development phases when working on robot tasks, so they set out to create a baseline system that will provide a starting place for others in academia to build upon it [1].

TABLE 1.1 Timeline

2007	Robotics research startup called Willow Garage took over the project and coined the name ROS.
2009	ROS 0.4 was released, and a working ROS robot called PR2 was developed.
2010	ROS 1.0 was released, Many of its features are still in use & ROS C Turtle was released.
2011	ROS Diamondback & ROS Electric Emys was released.
2012	ROS Fuerte & ROS Groovy Galapagos was released & The Open Source Robotics Foundation (OSRF) takes over the ROS project.
2013	ROS Hydro Medusa was released.
2014	ROS Indigo Igloo was released,this was the first long-term support (LTS) release, meaning updates and support is provided for a long period of time (typically five years).
2015	ROS Jade Turtle was released.
2016	ROS Kinetic Kame was released. It is the second LTS version of ROS.
2017	ROS Lunar Loggerhead was released.
May 2018	ROS Melodic Morenia was released.
May 2020	ROS noetic Ninjemys was released.

1.2.3 ROS Concepts

ROS has three main levels of concepts: the Filesystem level, the Computation Graph level, and the Community level. These levels and concepts describe all the functions and utilization of ROS [2].

The Filesystem level:

The filesystem level is the organization of the ROS framework on a machine,The ROS file system includes **packages** , **Meta packages** , **package manifests** , **repositories**, **message types** and **services types**.

- **Packages:**

  ROS packages are the primary organizational units for ROS software, containing all source code, data files, build files, dependencies, and other items that are effectively grouped together.

- **Meta Packages:**

  Meta packages are customized packages that are used to represent a group of related packages. Meta packages are frequently used as a backwards-compatible placeholder for ros build Stacks that have been transformed.

- **Package Manifests:**

  Is a ROS package that contains an XML file. It contains all of the essential information about a ROS package, such as the package name, description, author, dependencies, and so on.

- **Repositories:**

  A ROS repository is a collection of ROS packages that share a common version control system and can be released collectively using catkin.

- **Message Types:**

  The data structures for messages sent in ROS are defined by a message type description. There are pre-existing data kinds in ROS that we may use directly

in our application, but we can also create a new ROS message if necessary. A new message is stored inside : “mypackage/msg/MyMessageType.msg”

- **Services Types:**

  For describing ROS service types, ROS provides a simplified service description language (”srv”). To enable request/response communication between nodes, this builds directly on the ROS msg format. stored in:
  ”mypackage/srv/MyServiceType.srv” , define the request and response data structures for services in ROS.

The Computation Graph level :

The Computation Graph is a peer-to-peer network of ROS processes working together to process data. The basic Computation Graph concepts of ROS are **nodes** , **Master** , **Parameter Server** , **messages** , **services** , **topics** , and **bags**, all of which provide data to the Graph in different ways.

- **Nodes:**

  Nodes are computation-based processes, such as a sensor, motor, processing or monitoring algorithm, and so on. Nodes execute actions based on information from other nodes, convey information to other nodes, as well as send and receive action requests from other nodes. A ROS node is written with the use of a ROS client library, such as `roscpp` (C++) or `rospy` (Python).
  The command to start Node is :
  `$ rosrun package-name executable-name`

- **Master:**

  The Master provides a declaration and registration service to the rest of the Computation Graph , which makes it possible for nodes to find each other and exchange data.
  the command to start the master :
  `$ roscore`

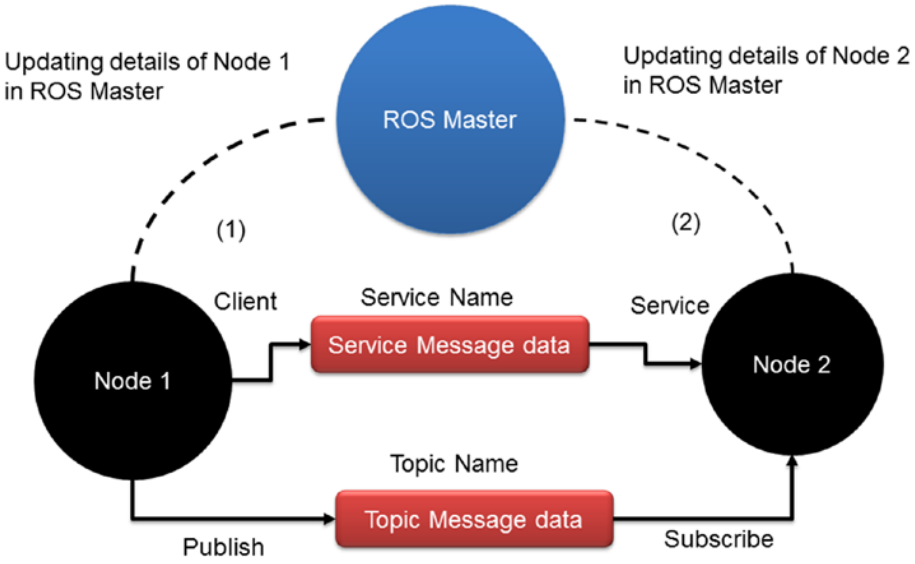

Figure 1.1: ROS Communication block diagram

- **Parameter Server:**

  Data can be kept in a central location using the Parameter Server. This server is used by nodes to store and obtain runtime information such as the distance between two fixed points in the environment, the robot's weight, and the lidar scan rate. The Parameter Server is implemented using XMLRPC and runs inside of the ROS Master.

  `$ rosparam` has many commands that can be used on parameters, as shown below:

`$ rosparam set`	set parameter
`$ rosparam get`	get parameter
`$ rosparam load`	load parameters from file
`$ rosparam dump`	dump parameters to file
`$ rosparam delete`	delete parameter
`$ rosparam list`	list parameter names

- **Messages:**

  Nodes connect with one another by sending messages to specific topics.A message is a composite data structure that combines primitive types (character strings, integers, floating points, booleans, and so on). Arrays of primitive types are

also allowed. Structures and arrays can be nested arbitrarily in messages. The message description is stored in;

"packagename/msg/myMessageType.msg".

the command for displaying information about ROS Message type :

```
$ rosmsg show
```

- **Topics:**

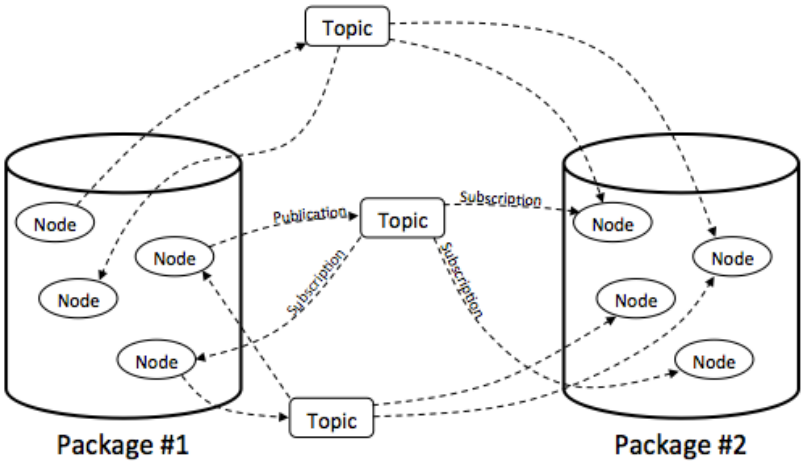

Figure 1.2: Model of how the ROS nodes publish and subscribe to topics

A topic is a data transport mechanism that allows nodes to send and receive messages amongst each other. Within their namespace, topic names must also be unique. A subject can be published by one or more nodes (**Publisher**), and one or more nodes can read data on that topic(**Subscriber**), topic is an asynchronous message bus, This notion of an asynchronous many-to-many bus is essential in a distributed system situation. sensor data, motor control commands, condition information, actuator commands, and anything else can be included in these messages. The type of data published (the messages) is always formatted in the same way when a subject is specified.

the command-line tool for interacting with ROS topics is :

```
$ rostopic list                    list the current topics
$ rostopic echo /topic_name        display Messages published to /topic_name
```

- **Services:**

A service is synchronous communication between two nodes that implements the **Request-Response** paradigm and is described by a pair of message structures. a node sends a request to another node and receives a response, for example (Request an updated map or portion of a map from a "map server"). ROS client libraries generally present this interaction to the programmer as if it were a **Remote Procedure Call (RPC)**.

The service description is stored in package_name/srv/myServiceType.srv. This file describes the data structures for requests and responses.

the command-line tool for interacting with ROS Services :

`$ rossrv` is a command-line tool for displaying information about ROS Service types:

Commands:

`$ rossrv show`	Show service description
`$ rossrv list`	List all services
`$ rossrv md5`	Display service md5sum
`$ rossrv package`	List services in a package
`$ rossrv packages`	List packages that contain services

Type `$ rossrv <command> -h` for more detailed usage.

- **Bags:**

Bags are a way of storing and retrieving ROS message data. Bags are an essential method for storing data that can be difficult to obtain but is required for creating and testing algorithms, such as sensor data.

`rosbag` is a command-line tool for displaying information about ROS Bags:

Commands:

`$rosbag record -a`	Record all the topics
`$rosbag info bag-name`	Info on the recorded bag
`$rosbag play -- pause bag-name`	Play the recorded bag, starting or paused
`$rosbag play -r #number bag-name`	Play the recorded bag at rate #number

Type `$ rosbag -h` for more detailed usage.

ROS Community Level:

ROS Community Level concepts are ROS resources made up of ROS developers and researchers who can produce and maintain packages as well as share new information about existing packages, allowing different communities to share software and knowledge. the ROS community provides the following services:

supported releases:			
Distro	Release date	Poster	EOL date
ROS Noetic Ninjemys	May 23rd,2020		May,2025(Focal EOL)
ROS Melodic Morenia	May 23rd, 2018		May,2023 (Bionic EOL)
unsupported releases (End of Life):			
Distro	Release date	Poster	EOL date
ROS Kinetic Kame	May 23rd,2016		April,2021

Table 1.2: List of Distributions.

- **Distributions:**

  ROS Distributions (table 1.2) are collections of versioned stacks that can be installed. they make it easy to install a collection of software while also maintaining consistent versions across the board.

- **Repositories:**

ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.

- **The ROS Wiki:**

  The ROS community Wiki is the main forum for documenting information about ROS. Anyone can create an account and contribute documentation, corrections, and updates, as well as write tutorials.

- **Bug Ticket System:**

  This resource can be used if we find a bug in the existing software or if we need to add a new feature.

- **Mailing Lists:**

  The ros-users mailing list serves as a key source of information about new ROS releases as well as a place to ask questions about the program.

- **ROS Answers:**

  A useful website for answering any ROS related questions, should be used as the first step in requesting help.

- **Blog:**

  The Blog is a website that hosts publications as well as regular updates, comments, and, in some cases, hyperlinks, videos, and images from one or more authors.

1.3 The Mobile Robots

1.3.1 Definition

Mobile robots are one of the most important inventions in the world of robotics since they can move around in a variety of environments and perform jobs according to their specifications (software algorithms and hardware parts).

1.3.2 The Main Parts of Mobile Robot

Every mobile robot has three primary components (computing unit, sensors, and actuators), each of which serves a distinct purpose in assisting the robot in recognizing its surroundings and making the best possible movement decisions.

The computing unit acts in response to software programs, sensors that capture and transfer data to the computing unit, which analyzes it and decides whether or not to move by sending an order to the actuators, and so on [1].

Figure 1.3 shows a general block diagram of the main parts of a robot..

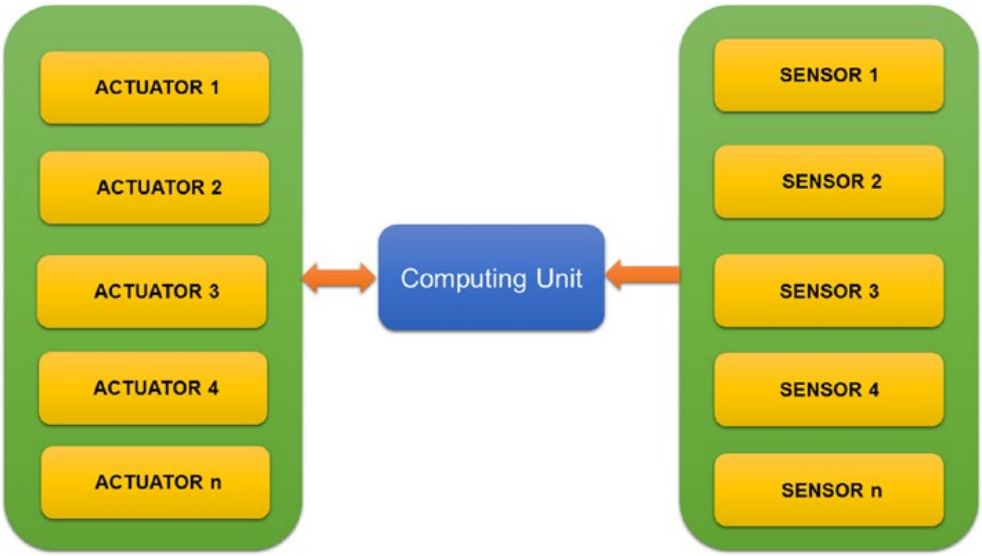

Figure 1.3: A typical robot block with actuators and sensors

1.3.3 Classification of Mobile Robots

Mobile robots are classified in two ways: by their working environment and the technology they use to move. Here are some examples of different types of environments in which mobile robots can be used :

- **Unmanned Ground Vehicles (UGVs)** usually referred to Land or home robots (figure 1.4). They are categorized as wheeled robots, tracked robots, and legged robots with two or more legs (humanoid, or resembling animals or insects). These are more complex types of robots and are autonomous humanoid as it requires many degrees of freedom and synchronization.

Figure 1.4: Unmanned Ground Vehicles (UGVs)(using Tracks and Wheels)

- **Autonomous Underwater Vehicles (AUVs)**, often known as underwater robots (figure 1.5) can navigate and travel across water on their own.Swimming Robots is another name for AUVs. Submersion is accomplished using ballast (compressed air and flooded compartments), underwater thrusters, tails, fins, or even wings.

Figure 1.5: Autonomous Underwater Vehicles (AUVs)

- **Unmanned Aerial Vehicles (UAVs)** also known as Aerial robots or drones, which fly through the air. This type of robot use propellers and wings.

Figure 1.6: Unmanned Aerial Vehicles (UAVs)

- **Polar robots** designed to navigate icy and uneven environments. same as **Unmanned Ground Vehicles(UGVs)** it can also use a Tracks or specific wheels to not slip on the ice.

Figure 1.7: Polar robots

- **Delivery and transportation** mobile robots that are designed to move materials and supplies around a work environment.

Figure 1.8: Delivery and transportation mobile robots

1.3.4 Types of Mobile Robot

There are two main types of mobile robots: Autonomous Mobile Robots (**AMR**) and Guided Mobile Robots (**GMR**).

- **Guided Mobile Robots (GMR**) or non – Autonomous Mobile Robots require some physical guidance and navigates on a pre-defined path such as magnetic tape, and bar codes, to move. AGVs are suitable for repetitive and specific tasks such as line follower robots, so all the details must be defined to AGV by the programmer since it cannot make a decision because it doesn't have any decision mechanism based on artificial intelligence.

- **Autonomous Mobile Robots (AMR**)are capable of navigating in an unpredictable environment. AMRs can sense the parameters of the environment and construct a model of the environment and locate themselves in this model to make their own decisions and then complete tasks consequently. This behavior enables AMR to build a navigation plan and optimize this plan with a special planning algorithm. An AMR does not have a predetermined navigational plan. AMR can create a map of the environment using sensor data and locate itself on the map at the same time. This is known as simultaneous localization and mapping (SLAM). SLAM enables us to create a sensitive navigation plan which can be created and improved dynamically by the programmer.

1.4 Simultaneous Localization And Mapping (SLAM)

SLAM or The Simultaneous Localization and Mapping Problem, asks if a robot can build a map of an unknown environment while also keeping track of its location on that map. The solution provided by SLAM is the most efficient and widely known technique that has been implemented in many different domains, from indoor robots to outdoor, underwater, and airborne systems, to make the robot truly autonomous to navigate and build maps and localize its current location.

1.4.1 Definition

SLAM is a process used in building maps of the environment and at the same time using those maps to compute its current location and orientation. The mainframe work of SLAM is state prediction (odometry), measurement prediction, landmark extraction, and data association that are being updated, which are vital in achieving a practical and robust SLAM implementation.

1.4.2 SLAM based algorithms

SLAM algorithm takes into consideration a variety of parameters sensors, map representation, robot dynamics, environmental dynamics, and the integration of sensor measurements and the robot's control system. the integration of all these set parameters is accomplished by using three of the most important solutions used in Slam-related algorithms, the kalmen filter specifically the extended kalmen filter EKF, and RaoBlackwellized Particle Filters (RBPF) and Graph-based.

Mapping the spatial environment using slam requires sensors to collect data that can be used in slam algorithms. The most commonly used sensor methods are raw range scan sensors and feature (landmark)-based sensors (whether extracted from scans or images). To get landmarks for an environment, there are 2 methods. The first one is landmark extraction using scans, which is a laser-based and sonar-based system on laser and sonar sensors. The second method is landmark extraction using the image, and that depends on the image that varies in parameters depending on which camera we are using to get this information. For example (stereo vision configuration multiple

camera configurations), the image-based land-marking gives information-heavy data. Still, there is a noise that must be considered.

1.4.3 SLAM application

SLAM has developed and expanded to include a set of different applications such as: innovation in indoor and outdoor navigation, underwater exploration , high-risk or difficult navigation environments, visual surveillance systems, aerial applications, and planet exploration. We can arrange them as follows:

1. **Indoor and Outdoor Navigation :**

Figure 1.9: SLAM in Outdoor and Indoor Application.

Slam can be used in indoor and outdoor applications to do certain functions like

navigation and building maps. The outdoor application represents navigation and building road maps like Google maps, and the indoor application, like a vacuum robot that uses slam to construct a map, which is usually a 2D map, uses the captured data to construct the map and figure out the current robot's location on it to locate the places that he didn't clear through until it covers the whole map.

2. **Underwater Exploration :**

   SLAM can be used in underwater applications as well. Due to the high pressure, humans can't get deeper under the water. The most widely known application can be found in the AUV's autonomous underwater vehicles that use SLAM to scan the underwater terrain and collect data to give a 3D map and give its current altitude under the water.

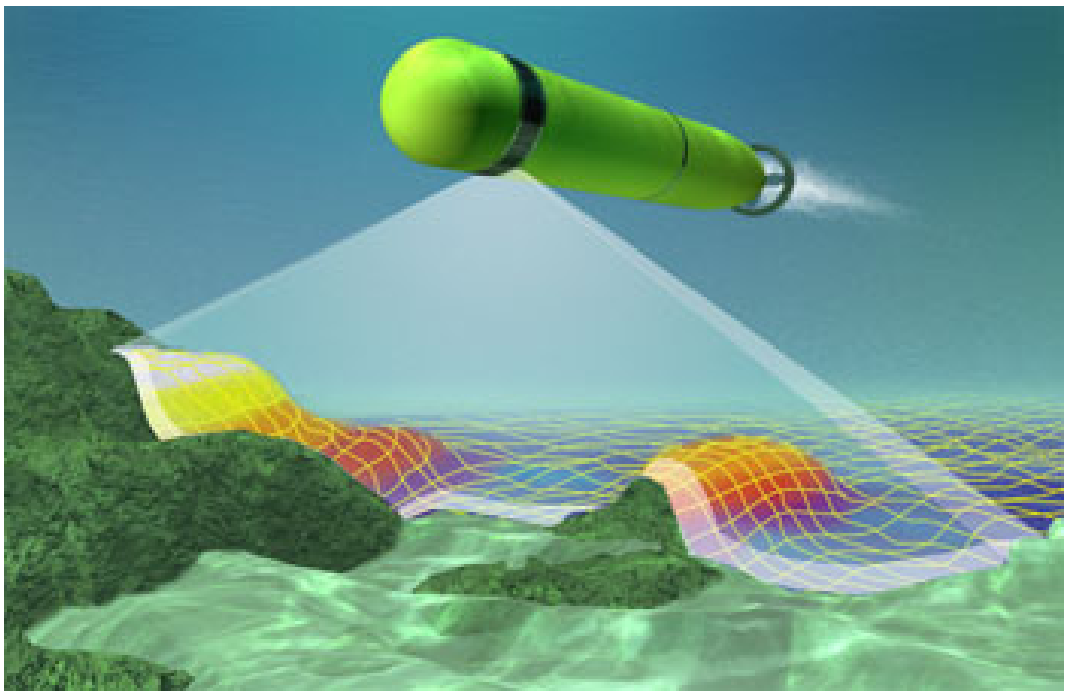

Figure 1.10: SLAM in Underwater Applications.

3. **Aerial Applications :**

   SLAM can be used in aerial applications such as 3D surveillance using a drone to construct a 3D map to determine its position and orientation on the local map,

exploring difficult environments that humans can't get into to get information about that terrain.

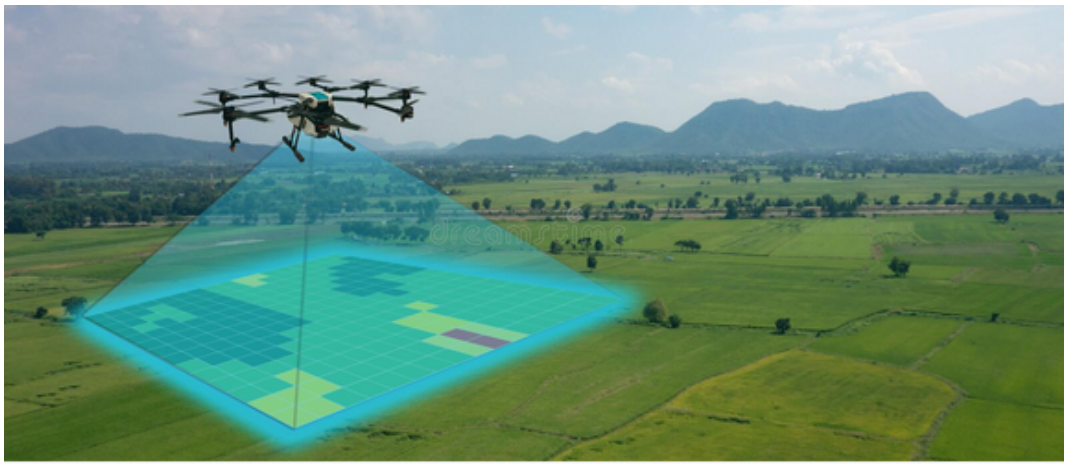

Figure 1.11: SLAM in Aerial Applications.

4. **Underground** :

SLAM also used in underground applications like caves and tunnels explorations due to the high risk environment.

Figure 1.12: SLAM in Underground Applications.

5. **Space :**

   SLAM is being used in space exploration, where it is difficult and very expensive to send humans to space because they will be exposed to many unexpected dangers.

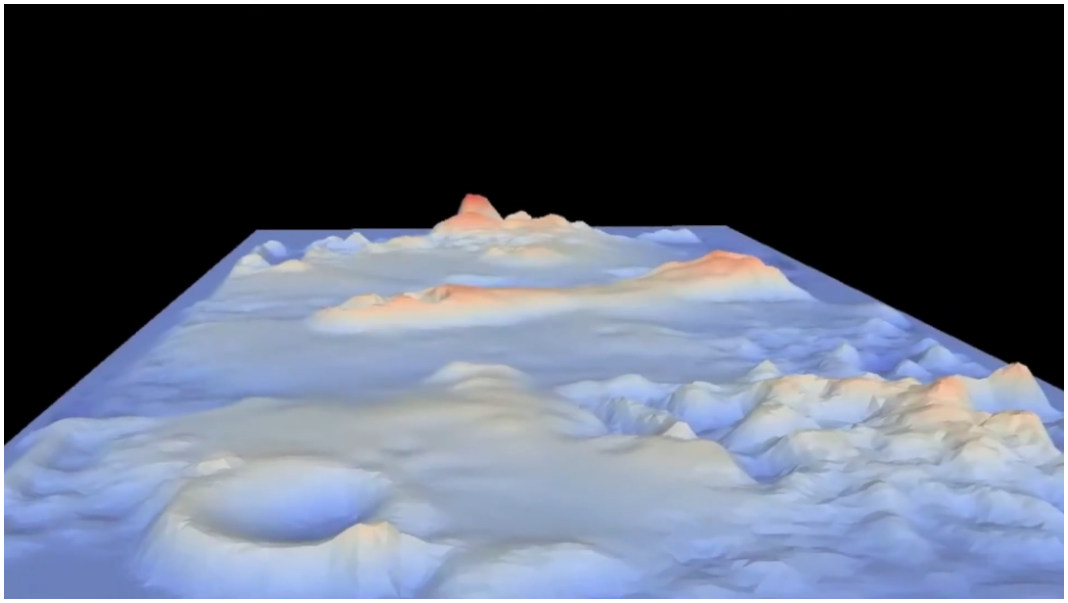

Figure 1.13: SLAM in space exploration (CSA Mars Emulation Terrain) .

1.5 Conclusion

In this chapter, we talk about robotics operating system (ROS) where we saw some basic ROS objects like nodes, messages, and topics, along with some command line tools for interacting with those objects and the important features of the system and the facilities which provides developers and researchers in the field of robotics an easy way to start. then we take a look at some generalities on mobile robots and some classification of them. also, we talk about the Simultaneous Localization And Mapping and the importance of using it with several Applications. in the following chapter , we gonna talk more about the pioneer mobile robot family with Control Strategies and SLAM approaches.

Chapter 2

Pioneer robot family and Slam techniques

2.1 Introduction

The previous chapter allowed us to introduce the ROS, Mobile robots and Slam techniques by citing different advantages of using ROS and slam approaches in the robotics field. Following this state of the art, we decided to focus our study on the mobile robots (Pioneer family) and classical Slam technique to navigate and build maps.

2.2 Pioneer family (3at and 3dx) and Mathematical Model

2.2.1 Pioneer Family

Adept Technology developed the Pioneer line of mobile robots in 1995. there are two available models Two-wheel drive and four-wheel drive . the Pioneer Family include the Pioneer 1 and Pioneer AT, Pioneer 2-DX, -DXe, -DXf, -CE, -AT, the pioneer 2-DX8/Dx8 Plus and -AT8/AT8 Plus, and the most recent Pioneer 3-DX and -AT mobile robots [3]. They are used for research purposes and applications such as monitoring, navigation, mapping, and other behaviors.

Pioneer 3-DX

The Pioneer 3-DX (figure 2.1) is a compact, weightless, two-wheel, two-motor differential drive robot with a rear balancing caster that is perfect to be used in inside laboratories or classrooms. The robot is equipped with eight front and eight rear SONAR, one 12V dc battery, wheel encoders, a micro-controller running ARCOS firmware, and the Pioneer SDK advanced mobile robotics software development kit. The datasheet file contains more details about the robot [4].

Figure 2.1: Pioneer 3-DX

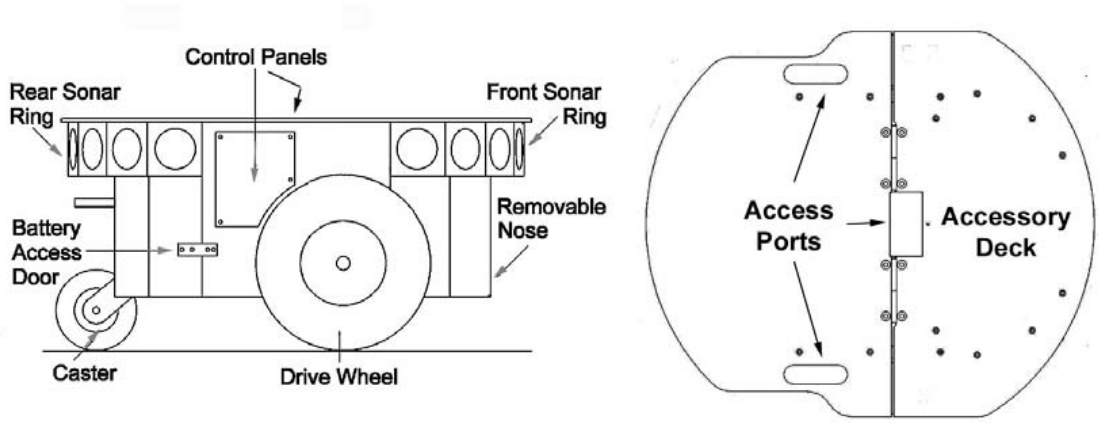

Figure 2.2: Pioneer 3-DX features

Poineer 3-AT

The Pioneer 3-AT is a little four-wheel, four-motor skid-steer robot that is perfect for all-terrain use or laboratory research. The Pioneer 3-AT is fully equipped with a single battery, eight front and rear SONAR sensors, an emergency stop button, wheel encoders, a micro-controller running ARCOS firmware, and Pioneer SDK an advanced mobile robotics software development kit. The data sheet file contains more details about the robot [5].

Figure 2.3: pioneer 3-AT

2.2.2 Mathematical Modeling

The geometry of the robot is shown in figure 2.4. When considering a moving robot model, it is assumed that the robot is perched on a level surface. where (XI, YI) is the inertial reference frame And (X, Y) is a local coordinate frame fixed on the robot at its center of mass (COM). The position of the COM is (x, y) concerning the inertial frame and θ is the orientation of the local coordinate frame with respect to the inertial frame.

"**a**" is the distance between the center of mass and the front wheels axle along X, "**b**" is the distance between the center of mass and the axle of the rear wheel along X, "**c**" is half distance between wheels along Y and "R_L, R_R" are the radii of left and right

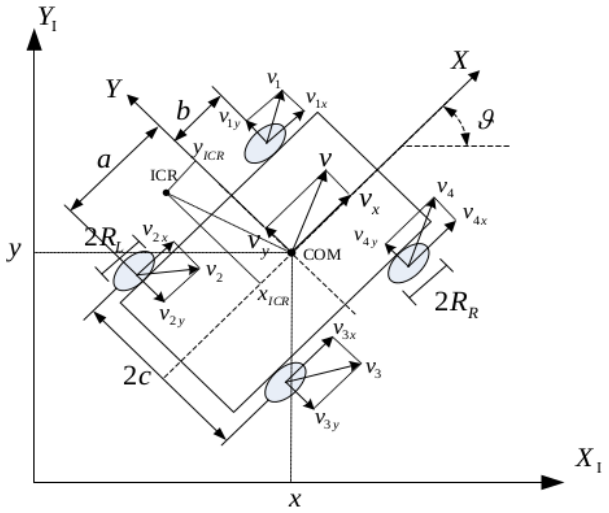

Figure 2.4: Mobile Robot Geometry.

wheels respectively. The coordinates of the instantaneous center of rotation (ICR) are (x_{ICR}, y_{ICR}).Assuming that the robot moves on a horizontal plane the linear velocity with respect to the local frame is given by :

$$\upsilon = \begin{bmatrix} \upsilon_x \\ \upsilon_y \\ 0 \end{bmatrix} \tag{2.1}$$

and its angular velocity is given by:

$$\omega = \begin{bmatrix} 0 \\ 0 \\ \omega \end{bmatrix} \tag{2.2}$$

The state vector with respect to the inertial frame is:

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.3}$$

The time derivatives of (3) denotes the robot's velocity vector and is given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \upsilon_x \\ \upsilon_y \\ \omega \end{bmatrix} \tag{2.4}$$

Supposing that longitudinal gliding between the wheels and the surface can be ignored we have the following equation :

$$\upsilon_{ix} = R_i \omega_i \tag{2.5}$$

υ_{ix} is the longitudinal component of the total velocity vector υ_i of the i-th wheel expressed with respect to the local frame and R_i is the rolling radius of that wheel.
If we take into account all wheels the following relationships between the wheels can be obtained [6].

$$\begin{aligned} \upsilon_L &= \upsilon_{1x} = \upsilon_{2x} \\ \upsilon_R &= \upsilon_{3x} = \upsilon_{4x} \\ \upsilon_F &= \upsilon_{1y} = \upsilon_{4x} \\ \upsilon_B &= \upsilon_{2y} = \upsilon_{3x} \end{aligned} \tag{2.6}$$

υ_L refers to the longitudinal coordinates of the left wheels velocities.
υ_R refers to the longitudinal coordinates of the right wheels velocities.
υ_F refers to the lateral coordinates of the front wheels velocities.
υ_B refers to the lateral coordinates of the rear wheels velocities.

In contrast to other mobile robots, the four wheel skid steering mobile robot's lateral velocities are typically nonzero because, due to its mechanical design, lateral skidding is required if the robot changes its orientation. Consequently, to finish the mathematical model , the following nonholonomic constrain in Pfaffian form is introduced:

$$\begin{bmatrix} -sin\theta & cos\theta & -x_{ICR} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = A(q)\dot{q} = 0 \tag{2.7}$$

Then we have :

$$\dot{q} = S(q)\eta \tag{2.8}$$

where :

$$S(q) = \begin{bmatrix} cos\theta & x_{ICR} sin\theta \\ sin\theta & -x_{ICR} cos\theta \\ 0 & 1 \end{bmatrix} \tag{2.9}$$

$$\eta = \begin{bmatrix} \upsilon_x \\ \omega \end{bmatrix} \tag{2.10}$$

S (q) is a full rank matrix, whose columns are in the null space of A(q) ,

$$S^T(q)A^T(q) = 0 \tag{2.11}$$

It is noted that since $\dim(\eta) = 2 < dim(q) = 3$,3 , equation(2.8) describes the kinematic of a sub-actuated robot with the nonholonomic constraint given by (2.7).

2.3 Control Strategies

To control the pioneer 3at, we use two approaches provided by ROS that allow us to move the robot **Automatically** or **Manually** using the keyboard. The 4-wheel drive skid steered robot, runs on 12V sealed lead-acid batteries, It also needs communication with a PC client, through one of the following[3]:

- Wireless radio modem
- Robot-to-laptop connector
- Robot-to-desktop tether
- Connection to an embedded computer

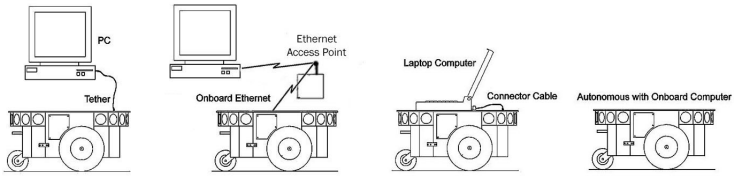

Figure 2.5: Client-server connection options

back to what we mentioned before two approaches that provided by ROS :

1. **Automatically**:

   In this mode, the robot acts autonomously and becomes responsive and intelligent. The robot moves cautiously and can avoid any static or dynamic obstacle because of the obstacle-avoidance behaviors that enable the robot to detect and

actively avoid collisions while receiving the data that comes from the SONAR sensors in the front and rear of the robot and analyzing it by the computing unit, then sending orders to the wheels to act.

2. **Manually**:

   in this strategy we are able to controlling the robot using the keyboard on PC that are connected with the robot ,we are using **Teleoperation Twist Keyboard Package** that are available on ROS packages.
   we use this command in the terminal to install the package :
   `sudo apt-get install ros-melodic-teleop-twist-keyboard`
   this package enable keyboard control for Robot, reading from the keyboard input and publishing it to `/pioneer_diff_drive_controller/cmd_vel` topic with the message type `geometry_msgs/Twist`.

   control buttons :

   $$\begin{array}{ccc} u\,\nwarrow & i\,\uparrow & o\,\nearrow \\ j\,\leftarrow & k & l\,\rightarrow \\ m\,\swarrow & ,\downarrow & .\,\searrow \end{array}$$

   q/z : increase/decrease max speeds by 10%
   w/x : increase/decrease only linear speed by 10%
   e/c : increase/decrease only angular speed by 10%
   anything else : stop

2.4 SLAM Approaches

With the advent of SLAM and the beginning of the developer's race to integrate different sensors to it such as LIDAR sensors and vision sensors. there are many different and unique approaches appeared to reach better results in map creation, navigation, and localization. we mention five widely used of SLAM approaches.

2.4.1 Gmapping based SLAM Algorithm (LIDAR)

The Gmapping method is one of the most widely used and trustworthy SLAM algorithms. Is used to build a 2D grid map and localize the position and orientation of the

robot using the Rao-Blackwellized Particle Filter (RBPF), which collects data from the LiDAR and combines it with the robot position [7].

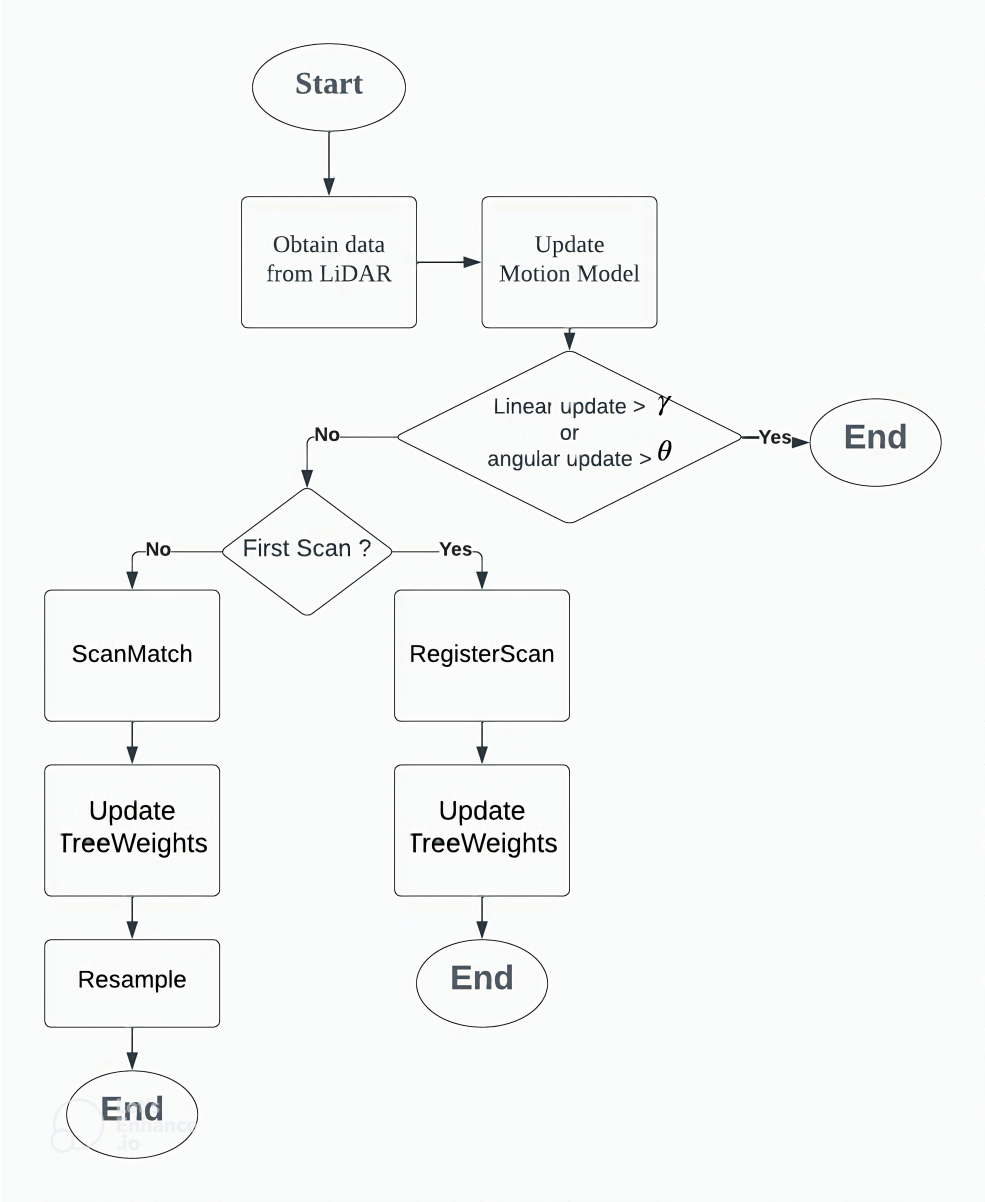

Figure 2.6: GMapping SLAM algorithm flowchart

The GMapping algorithm will begin its operation after receiving the laser data from the LiDAR sensor. Robot will simultaneously use its motion model, which is based on the odometry estimation, to continually update its pose in each processed particle.

The process will then come to a end if the linear distance or angular distance crosses the threshold γ or θ. If it is less than the threshold, it will begin to check to see if the first scan was received. If the initial scan is received, it will be immediately registered

on the map.

However, the program will perform the scan matching procedure if no data is received during the initial scan process. This method is important because it reduces mapping errors by correcting the map's pose estimation in each particle and then updating the particle tree's weight. The effective sample size N_{eff} is then calculated in order to estimate the accuracy of the current particle set, which represents the targeted posterior. The formula to calculate the N_{eff} is shown as:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (\widetilde{\omega^i})^2} \tag{2.12}$$

where $(\widetilde{\omega^i})$ represents the normalized weight of the particle i.

Resampling, the last step of the GMapping process, is carried out each time the value of N_{eff} drops below $N/2$, where N is the particle number. The samples with a higher weight will take the place of the particle with a lower weight importance value in this process. This situation reduces the risk of replacing good particles because the number of resamplings is reduced during execution and this process is only performed when necessary.

2.4.2 RGB-D SLAM (VISION)

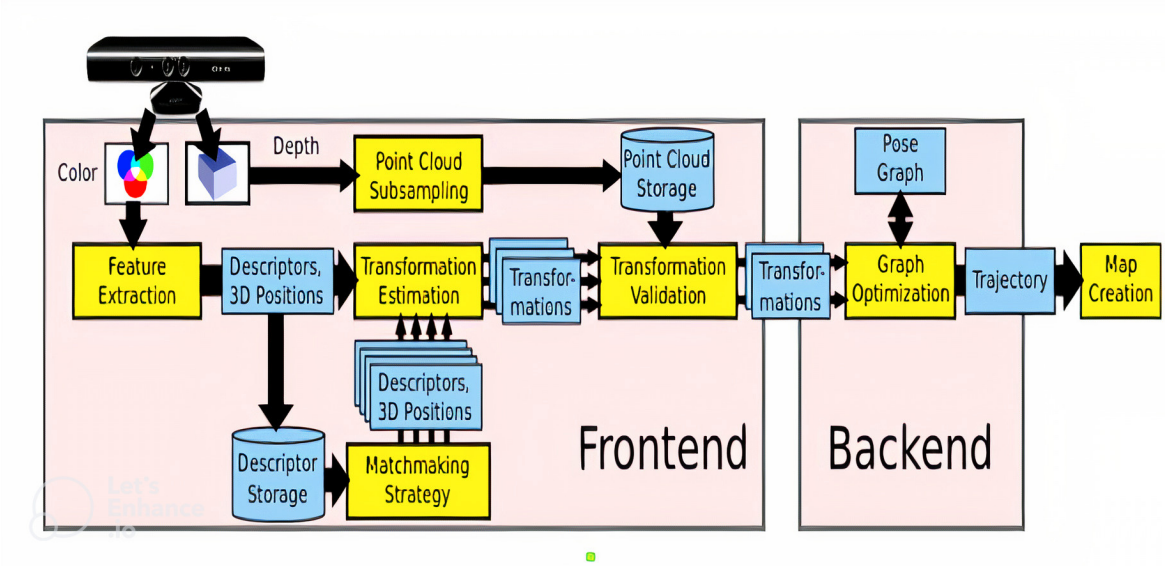

Figure 2.7: RGB-D System Overview

The SLAM RGB-D is one of the many methods that came into existence with the continued development of the slam, but this method differs from the others in that it can provide a 3D colored dense point cloud map using two types of data, namely the RGB image and the depth image. These will be computed in two main parts of the algorithm, which are a back-end and a front-end, that will give a dense colored 3D point cloud map of the scanned environment [8].

Front-end :

1. Feature Extraction :

   The data received from the RGB-D sensor will be quantified into values, in this case, RGB image and depth image, organized into data sets, and then the RGB image data sets will be utilized as an input for a feature extraction algorithm, which then outputs the extracted features. These last will be given a 3d description, which will be stored as vectors that originate from a set of points that belong to a local neighborhood of points or an entire surface.

2. Feature Matching :

   In this phase, we will use the 3D description to extract features that have been stored and use them to match the key features between different frames, which are usually a subset of the 20 previous frames that will be matched with the present 3 frames using different algorithms (orb, sift, surf). When we find matching key points, those matches aren't always 100% true, so we use an algorithm to minimize the error (RANSAC, least squared). Afterward, that data will be stored and utilized to describe the relative motion of the camera between its different viewpoints in the environment.

3. Pose Estimation :

   Using the previously matched features, we can give a rough estimation of the camera poses between the different relative poses of the camera between different frames. This forms a trajectory in the map that is being built. However, this trajectory isn't relabel in some cases, and errors can accrue in a repetitive feature

environment. Let's say like an office. We can avoid that by using the RANSAC algorithm and giving it a minimum number of features and a minimum THETA threshold for inliers. This proves to be effective and we can get a good pose estimate. This data will be then used to give a pose graph and store it.

4. Loop Closer :

   This process assures that the sensor stops storing data when it is back in the same pose. One of the used techniques is based on choosing a set of distinguished keyframes in different poses in the environment and then computing the currently received frames. If it doesn't match to any of the frames in the set, it will be added to the key frame set and those can be reduced by setting up constraints according to the environment if it has repetitive keyframes or not.

Back-end :

1. Pose Optimization:

   This part uses the previously stored and validated data from the pose graph that has been constructed. Sometimes, even with it passing the transformation estimation phase, noise can be found when we are trying to build a consistent global trajectory. We optimize the pose graph using a framework (g2o, masat, bpgo) this makes optimization of the overall global pose graph and it will be reliable to build a map.

2. Map Building :

   In this phase, the sets of points that have been gathered in local poses are referenced to their pose and associated with their depth by 2D to 3D projection. We use this cloud of 3d points that we get to reconstruct a 3d point cloud map, referencing the different clouds of points from the different poses to a global reference.

2.4.3 ORB-SLAM (VISION)

ORB-SLAM is a keyframe and feature-based Monocular SLAM. It functions in numerous situations in real-time. It can close loops and re-localize the camera from a wide range of angles. There are three parallel threads running throughout the ORB-SLAM operation **tracking**, **local mapping** and **loop closing** [9].

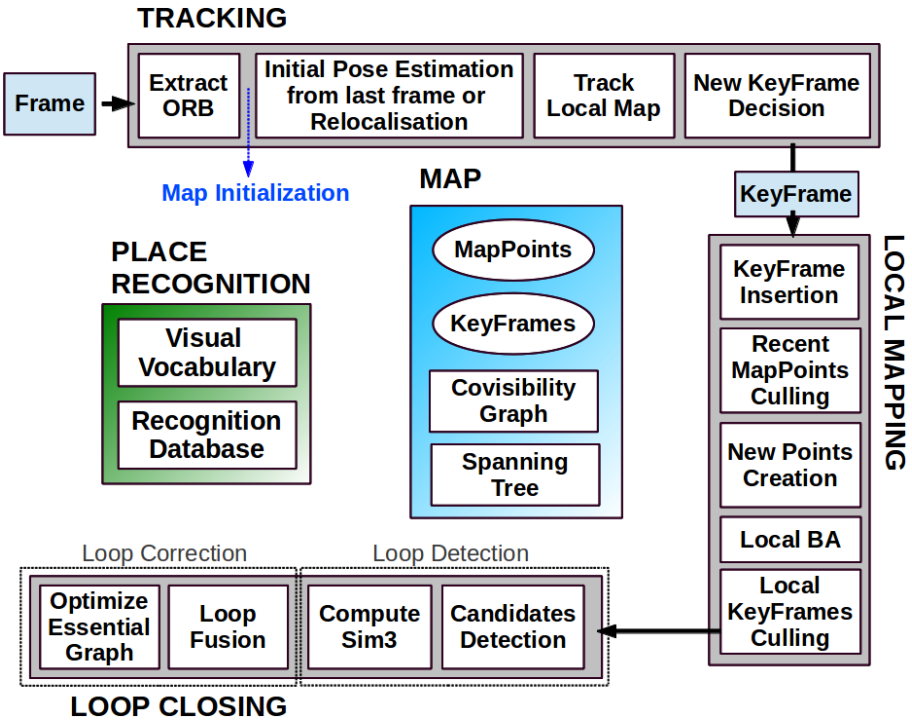

Figure 2.8: ORB-SLAM system overview.

Figure 2.8 showing all the steps performed by the tracking, local mapping and loop closing threads.

The tracking thread determines when to add a new keyframe and is in charge of localizing the camera with each frame. First, it optimizes the pose after doing an initial feature match with the previous frame. The place recognition module is used to perform global re-localization when tracking fails (for example, as a result of occlusions or abrupt motions). A local visual map is recovered utilizing the co-visibility graph of keyframes that are kept by the system after an initial estimation of the camera pose and feature matching. Following a re-projection search for matches with the local map points, the camera posture is once more optimized. The tracking thread then determines if a new keyframe should be added.

To achieve an ideal reconstruction in the area surrounding the camera posture, **the local mapping thread** processes new keyframes, updates, and bundle adjustment (BA). In order to triangulate new points, new correspondences for unmatched ORB in the new keyframe are explored in associated keyframes in the co-visibility graph. After creation, a strict point culling process is implemented to maintain only the best points, based on the data acquired during tracking. Repetitive keyframes must also be removed using the local mapping.

The loop closing thread, which looks for loops with each successive keyframe, is essential to maximizing the accuracy of SLAM algorithms. When a loop is found, a similarity transformation that describes the drift accumulated in the loop is calculated. The duplicated points are then joined when both sides of the loop have been aligned. The final step of ORB-SLAM is pose graph optimization, which performs relocating the candidate and any related keyframes to spread loop closing mistakes.

2.4.4 DSO-Direct Sparse Odometry (VISION)

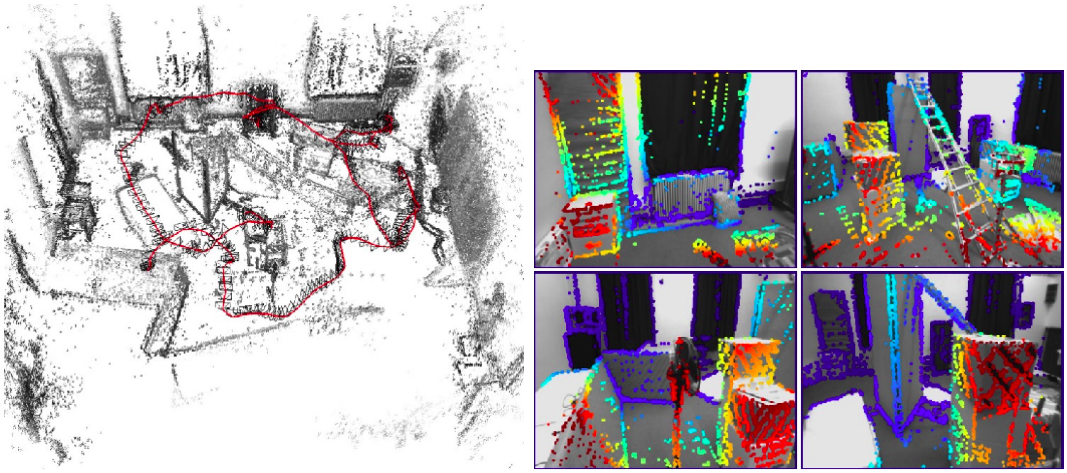

Figure 2.9: Direct sparse odometry (VISION).

DSO is a visual odometry method based on a brand-new, extremely precise formulation of direct and sparse structure and motion. combining a fully direct probabilistic model (minimizing a photometric mistake) with constant, joint optimization of all model parameters, including geometry (represented as inverse depth in a reference frame) and camera motion. It is handled in real time by equally sampling pixels

throughout the images and eliminating the smoothness. DSO sampling pixels from any areas of the image that contain intensity gradients, such as edges or smooth intensity variations on essentially featureless walls. [10].

Figure 2.9 shows 3D reconstruction and tracked trajectory for cycling around a building (monocular visual odometry only). The bottom-left inset shows a close-up of the start and end points ,as we can see DSO creates the dense map and the system is robust in terms of pose tracking, but the lack of loop close makes this map noisy. The system improved with a loop closure detection will give more precise results at the condition of careful calibration.

2.4.5 V-LOAM (VISION + LIDAR)

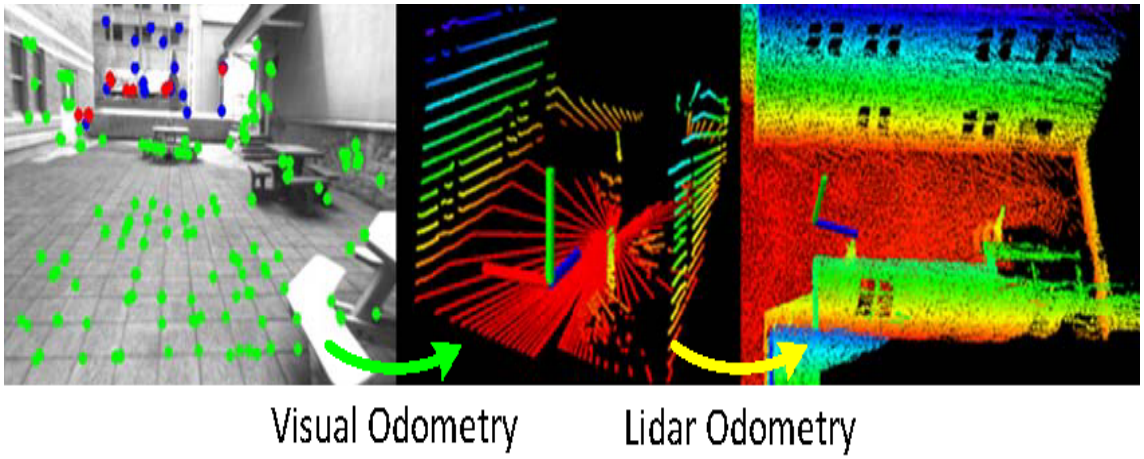

Figure 2.10: Visual-Lidar Odometry and Mapping.

Visual-lidar Odometry and Mapping is a technique that integrates the data from the visual and Lidar sensors to produce a precise and reliable framework that enables 6-DOF state estimation, mapping, and even obstacle identification.

This method closely combines the two modes to deal with both aggressive motion, such as translation and rotation, and a lack of optical texture, such as in completely white or black images. It offers remarkable accuracy in motion estimation and environment reconstruction in non-pathological circumstances.

The procedure consists of two separate steps. In the first, motion is estimated using visual odometry, which operates at a frequency higher than the frame rate of an image

(60Hz). The second method makes use of low-frequency (1 Hz) lidar odometry to eliminate distortion in the point clouds brought on by the drift of the visual odometry and smooth motion estimates. To generate maps incrementally, the distortion-free point clouds are matched and registered. As a result, the lidar odometry guarantees low-drift and robustness in undesirable illumination circumstances, while the visual odometry controls the quick motion [11].

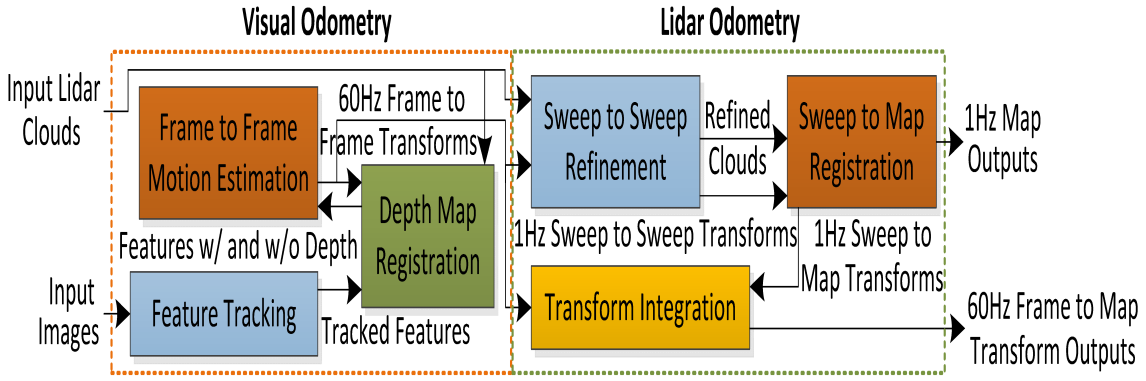

Figure 2.11: V-LOAM system overview.

The visual odometry section uses visual pictures with help from lidar clouds to estimate frame-to-frame motion of the sensor at the image frame rate. The feature tracking block extracts and matches visual features between successive photos in this section. In the depth map registration block, lidar clouds are registered on a local depthmap, and depth is connected to the visual features. The visual features are used by the frame-to-frame motion estimation block to calculate motion estimates.

The section of **Lidar odometry** is in charge of calculating the sweep-to-sweep transformation and creating a map from registered scans. The incoming scans are registered and sensor motion is estimated by the sweep-to-sweep refining module. The sweep to map registration module keeps the built-in map updated while also correcting the transform estimates. The high frequency transform is interpolated by the transform integration module using previous estimates.

2.5 Conclusion

In this chapter, we talk about the pioneer robot family where we saw two robot models the two-wheel drive pioneer 3-DX and the four-wheel drive pioneer 3-AT and the mathematical model. Then we take a look at two types of control strategies for the robot which is Automatically or Manually using the keyboard. After that, we explore a number of significant SLAM approaches, including Gmapping, RGB-D SLAM, and others, and we go through how they operate generally. in the following chapter , we gonna see the simulation results using GMapping based SLAM Algorithm with ROS.

Chapter 3

Simulation Results

3.1 Introduction

As has been mentioned previously, our objective is to navigate and build a map of the unknown environment through the mobile robot. Thereby, in order to do that, we considered the webots platform of the simulation to test and evaluate the Gmapping algorithm preferences in terms of building a 2D map and navigation.

3.2 Webots

Webots is a desktop application that simulates robots that is open source and multi-platform. It offers a complete development environment for simulating, modeling, and programming robots. It was designed for professional use and is widely used in industry, education, and research. Cyberbotics maintains webots as its main product continuously since 1998 [12].

3.3 Implementation of a Perception Algorithm using the webots

In order to test the perception algorithm using the webots and ROS we need to follow these steps presented hereafter:

3.3.1 Step1: Creating a Catkin Workspace

All of the necessary packages for our project's folder must be downloaded before we can utilize any of the crucial tools, such as mapping, navigation, and control. Create a new folder in the home directory called "project," then another inside it called "src" to download all packages in it. , the packages that we need to add are "**webots_ros package**" , "**slam_gmapping package**" and "**teleop_twist_keyboard package**".

1. Installing **webots_ros package**: to instal webots_ros Package (figure 3.1) from the source open "Terminal" and type :

   ```
   $ git clone -b melodic https://github.com/cyberbotics/webots_ros.git
   ```

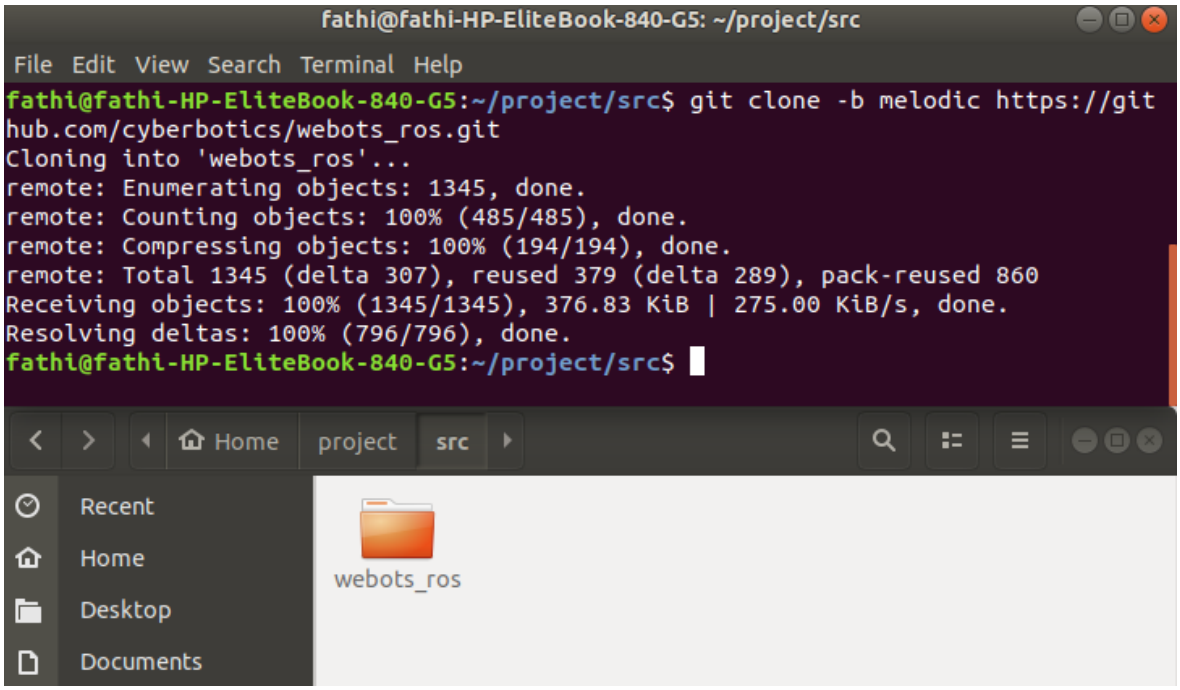

Figure 3.1: webots_ros package install.

2. Installing **slam_gmapping package**: to install slam_gmapping Package (figure 3.2). type in the same "Terminal" :

   ```
   $ git clone https://github.com/ros-perception/slam_gmapping.git
   ```

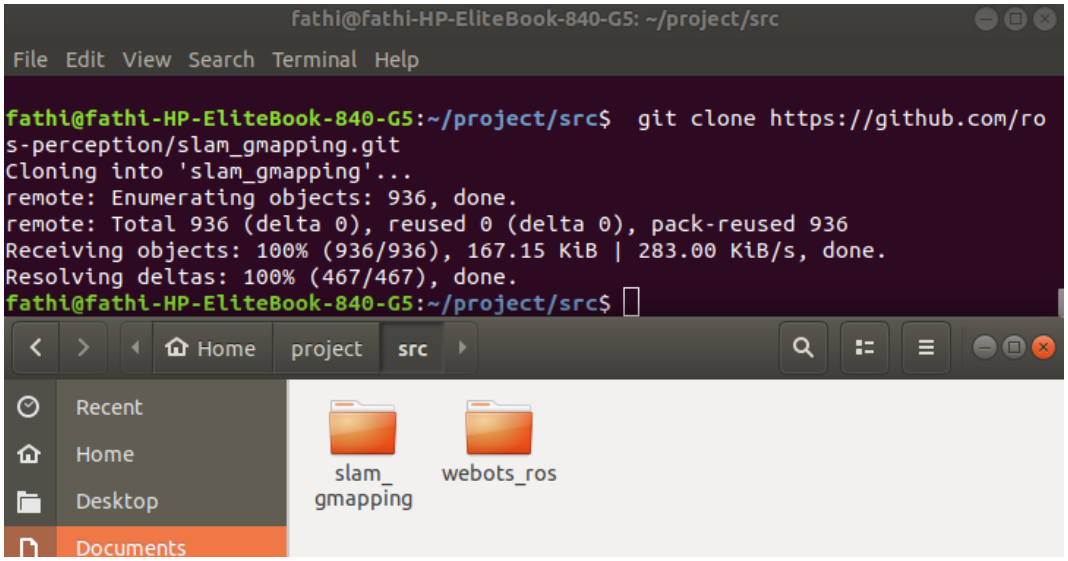

Figure 3.2: installing Slam_gmappin Package.

3. Installing **teleop_twist_keyboard package**: to install teleop_twist_keyboard package (figure3.3) type in the same Terminal :

   `$ git clone https://github.com/ros-teleop/teleop_twist_keyboard`

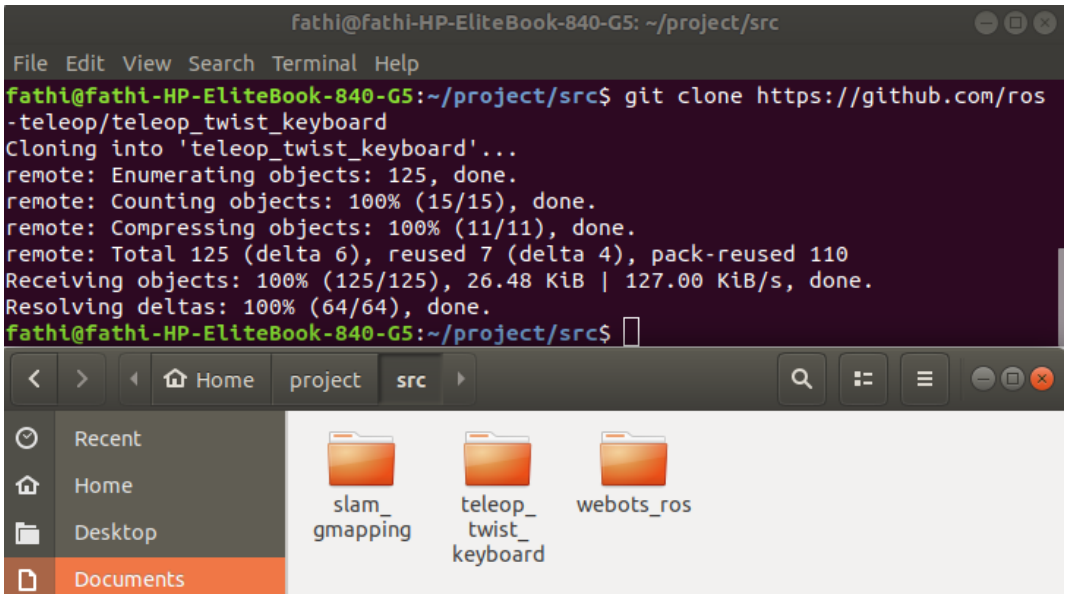

Figure 3.3: teleop_twist_keyboard package.

4. Build Packages: Catkin is an official building system in ROS. To build the previous packages, we use the command:

   `$ catkin_make`.

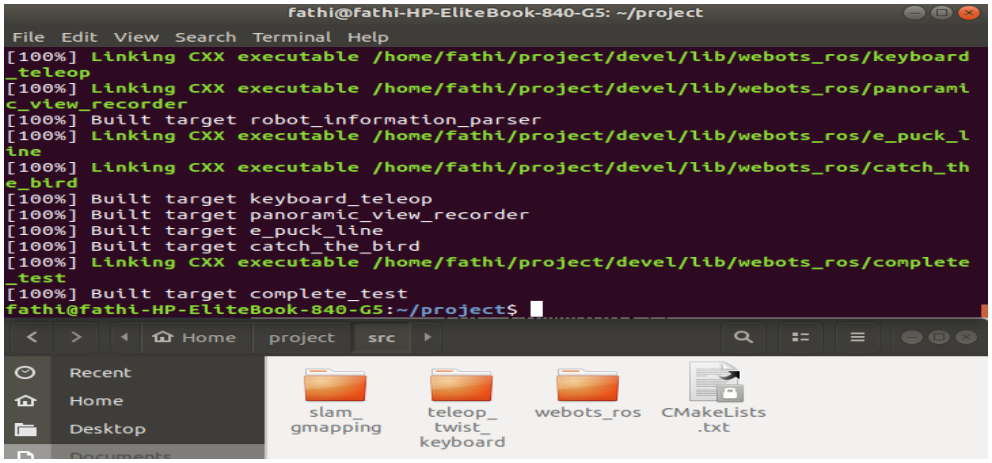

Figure 3.4: Building Packages

3.3.2 Step 2 : Starting simulation

After we finished adding those packages, we now ready to launch our project. In the same "Terminal" , we typed the following commands:

1. `$source devel/setup.bash`

   By sourcing our setup.bash file, we are adding several environment variables that ROS needs in order to work.

2. `$ export WEBOTS_HOME=/usr/local/webots`

   Defining the WEBOTS_HOME environment variable.

3. `$ cd src`

   `$ cd webots_ros`

   `$ cd scriptes`

   Changing the current working directory.

4. `$ chmod +x *.py`

   Chmod is an abbreviation for change mode, it makes the file executable.

5. `$ roslaunch webots_ros pioneer3at.launch`

   This launch file brings up a set of nodes for the package that provide some aggregate functionalities like autonomous driving and avoiding obstacles. It launches webots and setting the pioneer3at robot in it (figure 3.5) .

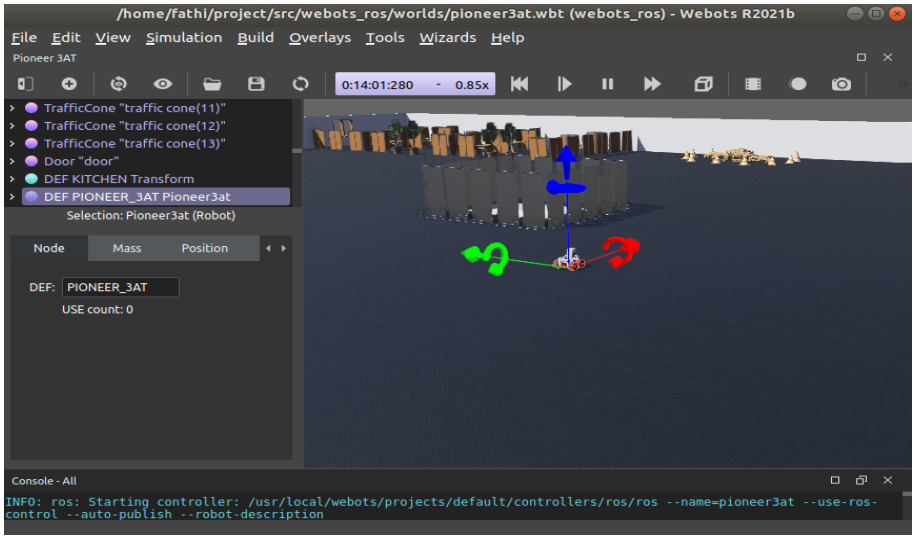

Figure 3.5: Launching Webots and setting up the P-3at mobile robot.

6. To run Gmapping SLAM algorithm, we typed in new "Terminal tap": `$ rosrun gmapping slam_gmapping scan:=/pioneer3at/Sick_LMS_291/laser_scan/layer0`

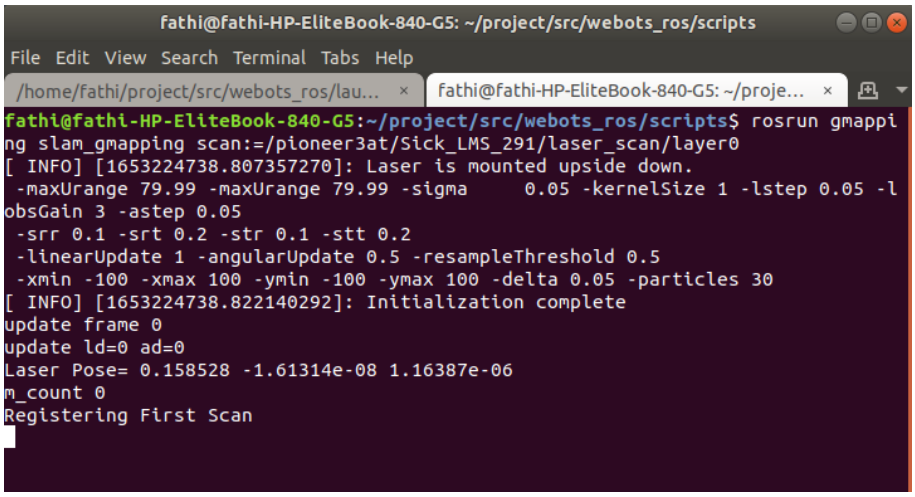

Figure 3.6: Launching Slam Gmapping Scan.

note: "/pioneer3at/Sick_LMS_291/laser_scan/layer0" is the Topic that we need to subscribed to it to get laser scans to create the map from it .

7. To control the robot via the keyboard, we open a new "Terminal Tap" and typed: `$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:= /pioneer_diff_drive_controller/cmd_vel`

We need to publish on a different topic (by default /cmd_vel) in this case: (/pioneer_diff_drive_controller/cmd_vel).

Figure 3.7: Launching Teleop Twist Keyboard.

8. "Rviz" is a 3D visualization tool for ROS. To open and set some parameters to visualize the map, we type in a new "Terminal": $ `rviz`

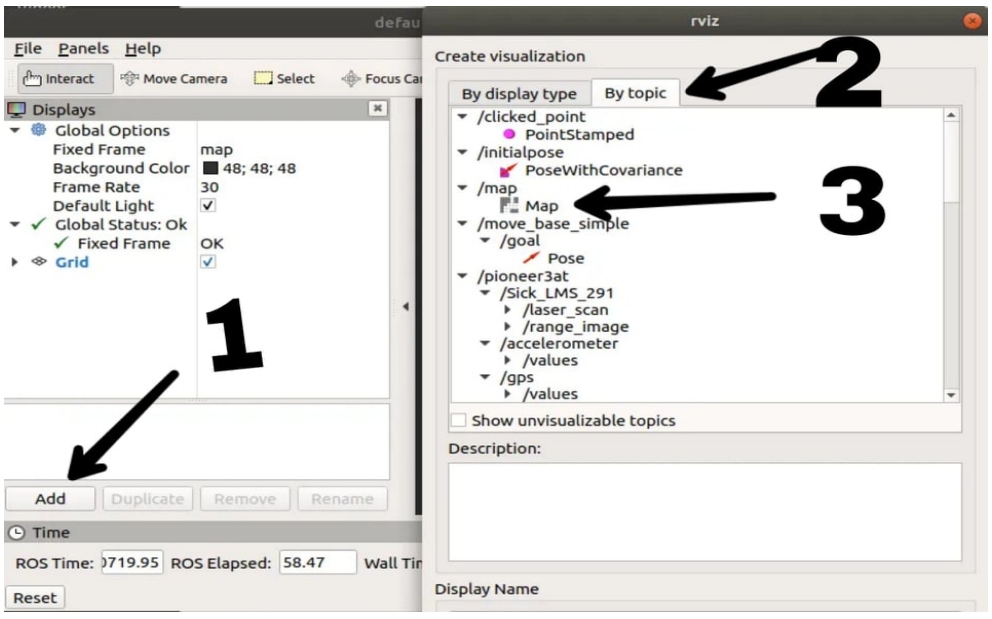

Figure 3.8: Launching Rviz and choosing Map Display

On the left is the Displays list, which will show any displays we have loaded. In step one (1) "Add", we add a new display. In step two (2), we choose "By Topic"

to see all the topics that we have. In step three (3), we choose the "Map" Topic to set the map and visualize it.

3.3.3 Results

After launching the Pioneer 3at in webots environment under two-mode Autonomously way using an intelligent algorithm to avoid an obstacle (see full code in Appendix D), and Manually way via the keyboard. Moreover, to build a 2D map, we use the Gmapping SLAM algorithm. As result, we can see that the map begins to be created in "Rviz".

Autonomously

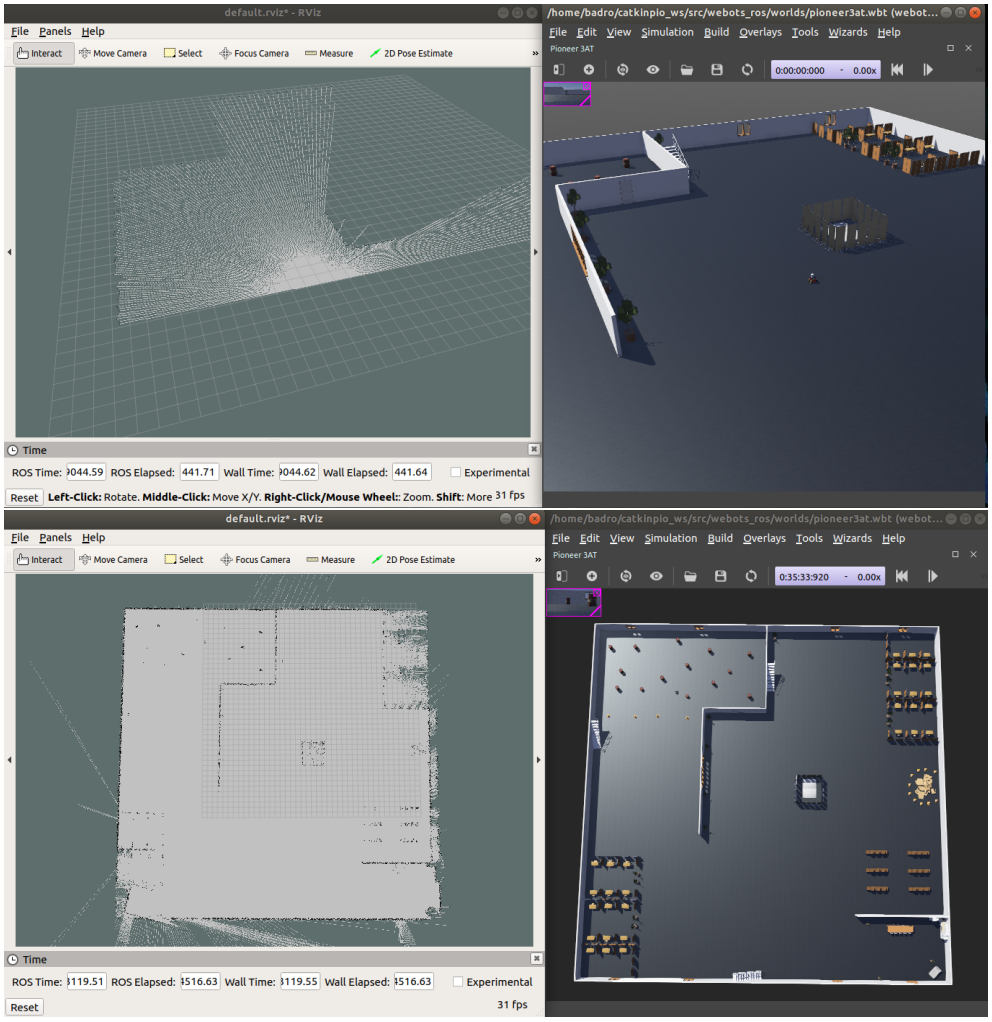

Figure 3.9: The beginning and end of building the map (Autonomously).

Figure 3.9 show the simulation results of mobile robot navigation in an unknown environment. In this autonomous case, we use an intelligent algorithm by using **the 5th command in Step 2** to make the mobile robots scan the environment, to achieve good performances in terms of building the 2-D map and tracking the position of the robots.

Manually

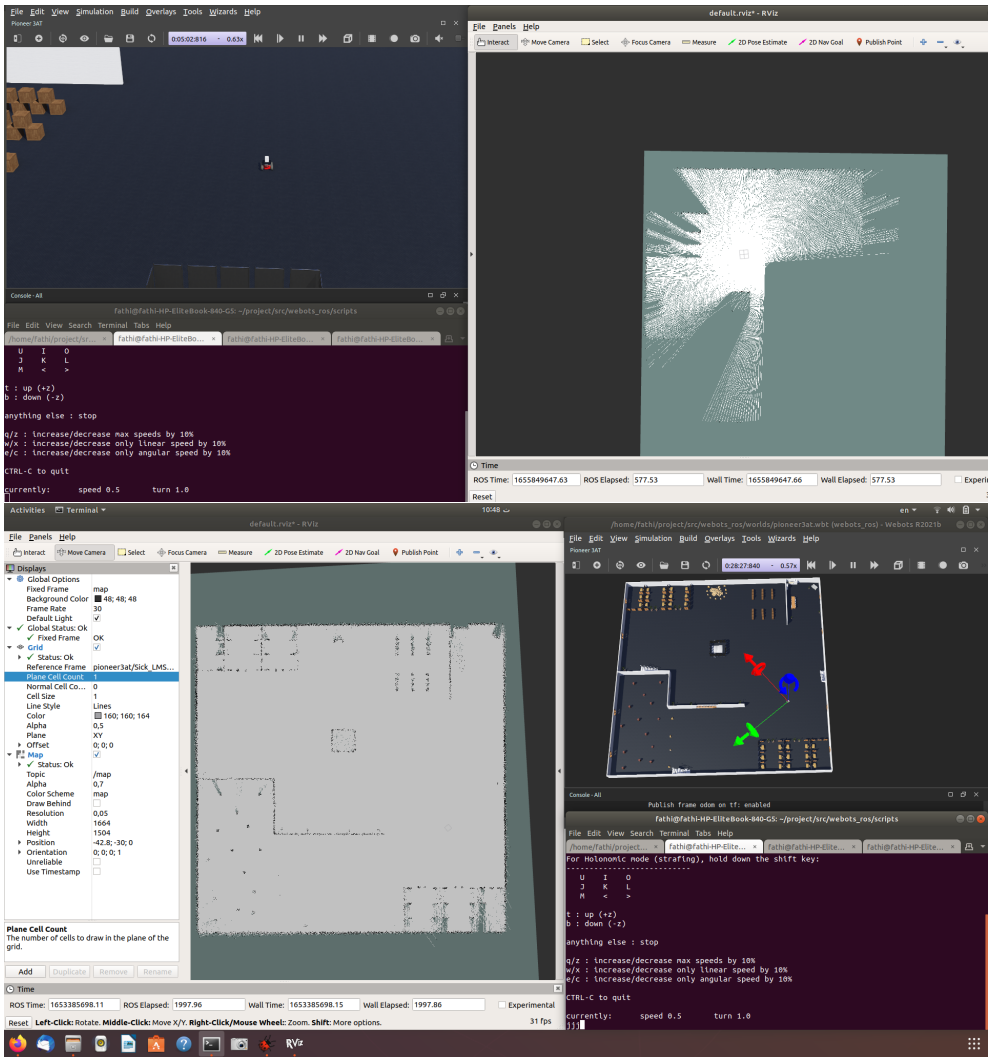

Figure 3.10: The beginning and end of building the map (Manually).

Figure 3.10 show the simulation results of mobile robot navigation in the same environment as before. In this manual case, We use the keyboard to drive the mobile robots to scan the environment, especially into the hard places, and build the 2-D map

using the Gmapping package as shown in **Step 2 / the 6th command**.

3.4 Conclusion

In this chapter, we implement all the packages needed to test the robot's capability of building 2-D maps and navigating. For this purpose, we used Webots to set the P-3AT robot in an indoor environment that consists of walls and some obstacles and set the robot in automatic mode to drove it self autonomously and manuel mode to control it by keyboard and let the robot discover and scan the environment to build the 2D map and navigate simultaneously using the Gmapping algorithm. in the next chapter, we gonna see the experimental results adapted by the kinect sensor using the RGB-D SLAM in term of building 3-D map .

Chapter 4

Experimental Results

4.1 Introduction

After we obtain the 2D map using the Gmapping algorithm in the previous chapter, In this chapter, we take advantage of having the Kinect sensor to do an experiment test with RGB-D SLAM in order to build a 3D map.

4.2 Kinect sensor (XBOX one)

Figure 4.1: XBOX ONE KINECT

The kinect is a Microsoft motion sensor that provides its user with a NUI (natural user interface). This provides the user with live control of the game without an

intermediate like the controller. The kinect recognizes individual players through face recognition, and it tracks players' movements and gestures through their skeleton model movement and voice commands through the microphone. When it was first created, it was towards gaming. Afterwards, it was introduced to robotics due to the immense capabilities and its low cost over performance.

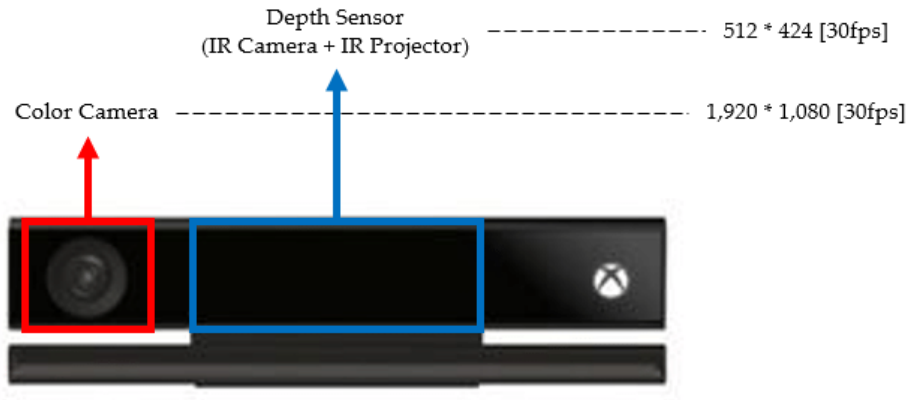

Figure 4.2: The Main Sensors of Kinect

The kinect relies on three main sensors as shown in Figure 4.2 , which are a color camera, an IR camera, and an IR projector, which are considered one unit called a depth sensor.

The kinect is made to give a 3d model, and that is by associating the frames taken by the color camera with their depth. That means that every pixel of each frame will be defined with a depth, and then we use the 2D to 3D projection to get the 3D model.

4.3 Testing RGB-D SLAM Algorithm using the Kinect motion sensor

In this section, we have two experimental tests in two different environments.

The first experiment was about tasting the Kinect sensor only in order to test the RGB-D SLAM algorithm. The second experiment was to integrate the Kinect with the P-3DX mobile robot and test the RGB-D SLAM algorithm.

4.3.1 The first experimental

we had this experiment to learn more about the Kinect sensor and how the RGB-D SLAM builds the 3D map and localizes the position and rotation of the sensor.
following these steps presented hereafter:

1. Installing ROS wrapper for kinect V-2 (Xbox one) as mentioned in (Appendix C)

2. Run rtabmap to build a map with visualization in rviz

   ```
   $ source /catkin_ws/devel/setup.bash
   $ roslaunch kinect2_bridge kinect2_bridge_custom_rviz_sd.launch
   ```

Results:

We tested the Kinect sensor in two environments:

- Environment 1 (A room) :

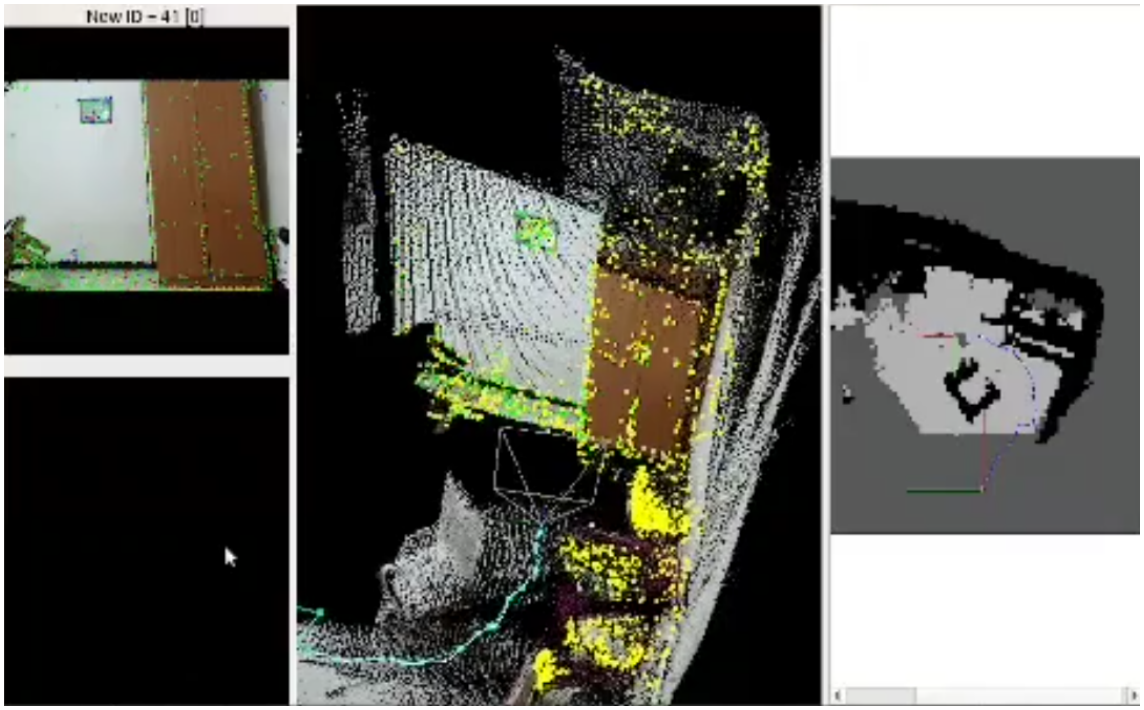

Figure 4.3: Scanning the Room with kinect.

A room scan was done by hold the device and walking around the room. When we finished scanning the room, we get the following result :

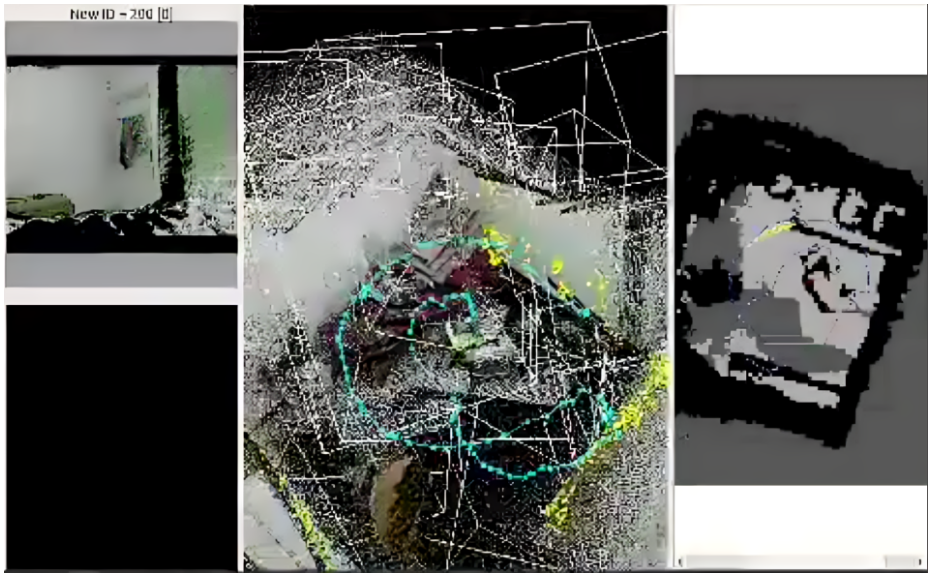

Figure 4.4: The final result of scanning the room.

- Environment 2 (LTSS LAB) :

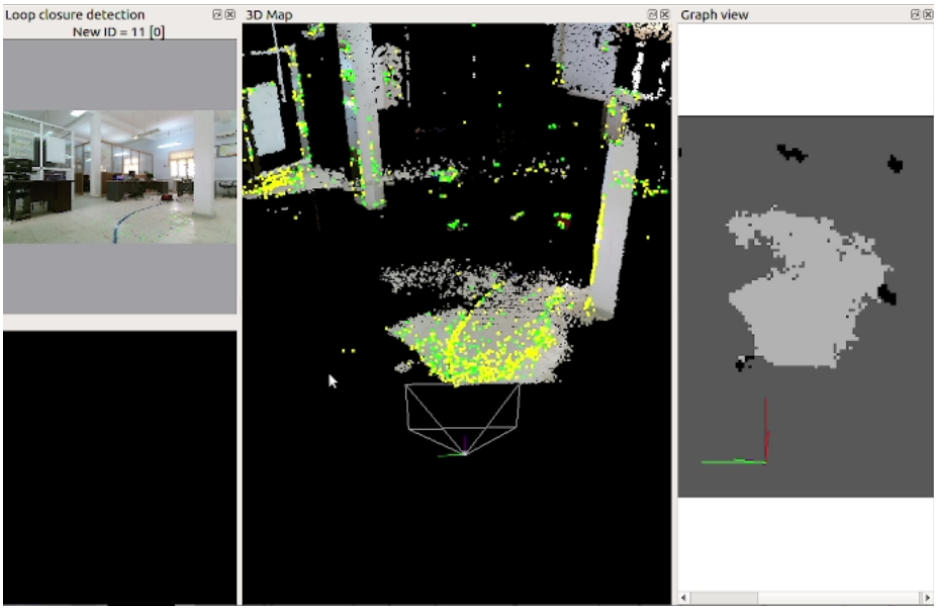

Figure 4.5: Scanning a part of LTSS Lab.

Figure 4.3 ,4.4, and 4.5 shows some simple test results. a 3d map and a 2D map were built and some special marks are generated too like the blue path of the sensor. all these results pave the way for the second experiment.

4.3.2 The second experimental

We had this experiment to gather the Kinect sensor with the P-3DX robot and scan the environment 2 (LTSS) to navigate and build the 3D and 2D maps.

To begin, we must state that this second experimental test was conducted in collaboration with another team that is working on the trajectory followed. They command the robot to follow a specific path, as shown in the blue tape.

Figure 4.6: the Kinect sensor with the P-3DX robot

following these steps presented hereafter:

Figure 4.7: The environment to scan .

1. Adding **ROSARIA** into the workspace.

   We need to download the `rosaria` package from its repository. we type in the terminal :

   ```
   $ cd /catkin_ws/src
   $ git clone https://github.com/amor-ros-pkg/rosaria.git
   ```

2. For ubuntu 18 we need to instal and build libaria

   ```
   $ sudo apt update
   $ sudo apt install libaria-dev
   $ catkin_make
   ```

   libaria-dev it is a C++ library for MobileRobots/ActivMedia robots.

3. Running the RosAria node:

   (a) change the configuration of the network interfaces

   ```
   $ sudo ifconfig enp1s0 192.168.1.1
   ```

(b) Setting the ROS_MASTER_URI environment variable to reference that `roscore` server.

```
$ export ROS_MASTER_URI=http://192.168.1.1:11311
```

(c) Set The ROS_IP environment variable to specify the IP address to use.

```
$ export ROS_IP=192.168.1.1
```

(d) Changing the permissions of the serial port device to allow all users to access it.

```
$ sudo chmod 777 -R /dev/ttyS0
$ sudo chmod 777 -R /dev/ttyUSB0
```

(e) `roscore` must always be running for any other ROS node in this ROS node network to work and communicate.

```
$ roscore
```

(f) Running the RosAria node.

```
$ rosrun rosaria RosAria
```

now we are communicate with the P-3DX mobile robot

4. Controlling the robot via the keyboard using the teleop twist keyboard package.

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=/RosAria/cmd_vel
```

We need to publish on a different topic (by default /cmd_vel) in this case: (cmd_vel:=/RosAria/cmd_vel).

5. Run rtabmap to build a map with visualization in rviz

```
$ source /catkin_ws/devel/setup.bash
$ roslaunch kinect2_bridge kinect2_bridge.launch
$ roslaunch kinect2_bridge kinect2_bridge_custom_rviz_sd.launch
```

Results:

After we gathered the Kinect sensor with P-3DX and start scanning with the launch of the robot by the other team in LTSS lab we get the following results:

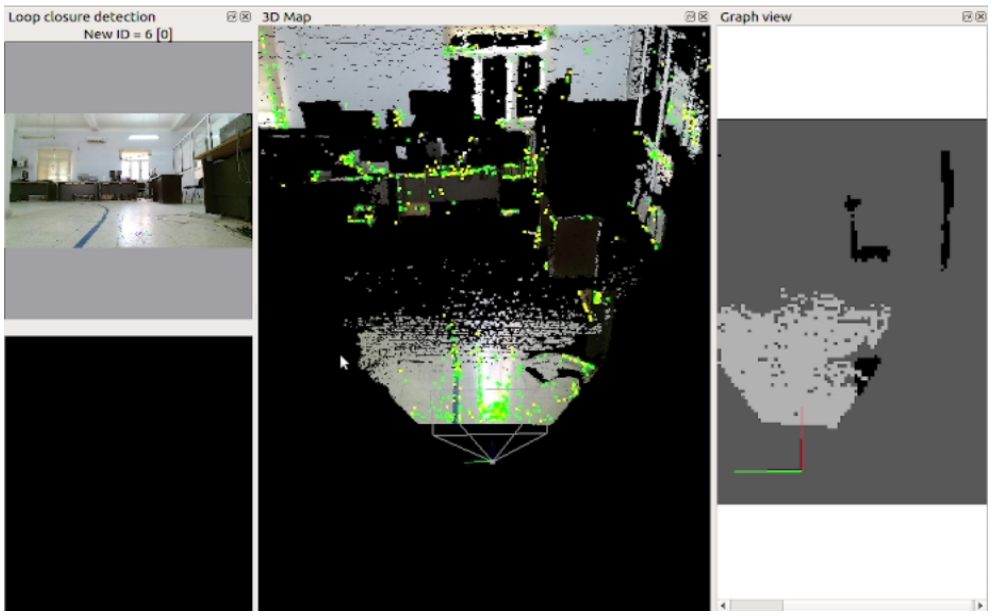

Figure 4.8: The Start of scanning.

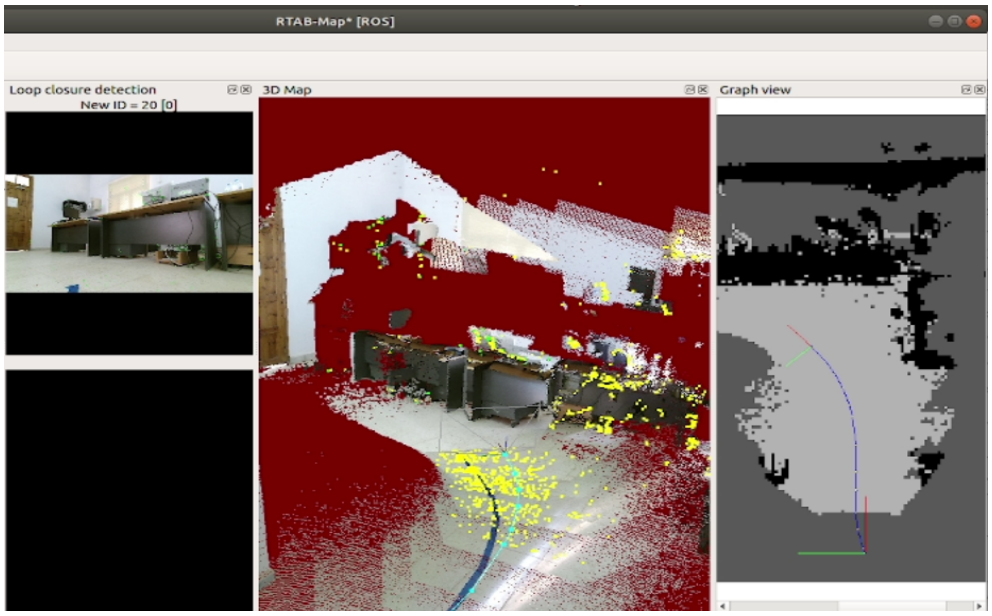

Figure 4.9: The End of scanning.

The figure 4.8 shows the beginning of the scan form the beginning of the blue tape until the end of it as shown in figure 4.9. the blue path that generates in the 2D and 3D maps is the path the robot took. On other hand, the Black places (3-D maps) are the places that have not been scanned yet.

4.4 Conclusion

In this chapter, we have used the Kinect sensor to implement another Slam technique. In fact, in the previous chapter, we used the Gmapping algorithm to build a 2-D map. So, in this part, we take advantage of the Kinect sensor which is the fusion between two sensors colour camera and a depth sensor (IR camera+ IR Projector). Therefore, the obtained experimental results demonstrate high performance in terms of 3-D map build and tracking the position.

GENERAL CONCLUSION

In this master project we discussed four chapters:

In the first chapter, we dealt with the ROS platform and discussed its main concepts, which provide helpful tools like packages, nodes, and more, as well as some command-line tools for interacting with those objects to build and develop software programs for the robot. Following that, we examine certain generalizations about mobile robots and a classification of them. Additionally, we discuss the significance of using simultaneous localization and mapping with various applications.

In the second chapter, We discussed the Pioneer Robot Family and saw two robot models. The pioneer 3-DX for two-wheel drive and the pioneer 3-AT for four-wheel drive with some hardware details and the mathematical model for four-wheel mobile robot. Then we take a look at two types of control strategies for the robot. The first one is auto mode, which allows the robot to act in an intelligent way in order to move and avoid obstacles in an unknown environment. The second mode allows us to control the robot by keyboard and drive it through the environment. Following that, we examine a number of critical SLAM approaches based on LIDAR and vision sensors, such as Gmapping, RGB-D SLAM, and some others, and we've been through how they perform construction of the 2-D and 3-D maps, tracked trajectory and estimated the position of the model with some other utilities such as feature matching and loop closure in general.

In the third chapter, we saw how to implement the "SLAM G-mapping algorithm" using ROS in our mobile robot to get the capability to navigate and build a 2D map in an unknown indoor environment in the Webots simulator. This map is built by gathering information from the LIDAR sensor and the odometer to construct the 2-D map that we see in Rviz. The performance of the Gmapping has achieved good results

in terms of building grid maps and navigation.

In the fourth chapter, we explained how simply the kinect sensor relies on three main sensors a color camera, an IR camera, and an IR projector, and we saw how to implement the RGB-D Slam using ROS and use the Kinect sensor to collect data, build a 3D map, and test it in different environments. With this, we can say the performance of the RGB-D SLAM has been achieved with good results in terms of building 3-D and 2-D maps and navigation.

As a future work, we propose to use some SLAM approaches in terms of building maps of an unknown environment and using those maps to set a specific path for the robot to follow using tracking algorithms.

Appendices

A ROS Melodic installation

In our project we used **Melodic** version of ROS. to install it , follow the next steps :

- in Ubuntu, we open a new "Terminal" window and then type the following command to make sure our Debian package index is up-to-date:
  `$ sudo apt update`

- then we use the **Desktop-Full Install** package that contains ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, and 2D/3D perception.
  in the same "Terminal" window type :
  `$ sudo apt install ros-melodic-desktop-full`

- now we need to install and initialize dependencies for building packages `rosdep` enables us to easily install system dependencies for the source we want to compile and is required to run some core components in ROS:
  install `rosdep` :
  `$ sudo apt install python-rosdep`
  initialize `rosdep` :
  `$ sudo rosdep init`
  `$ rosdep update`

B Webots installation

For our applications, we are using "Webots 2021b" .

To install webots on Ubuntu, we download the Debian package from the Cyberbotics website. After it finishes, double-click on the Debian package file to open it with the Ubuntu Software App and click on the -Install- button (figure10).

Note : you can download the Debian package of any version directly from the Cyberbotics website.

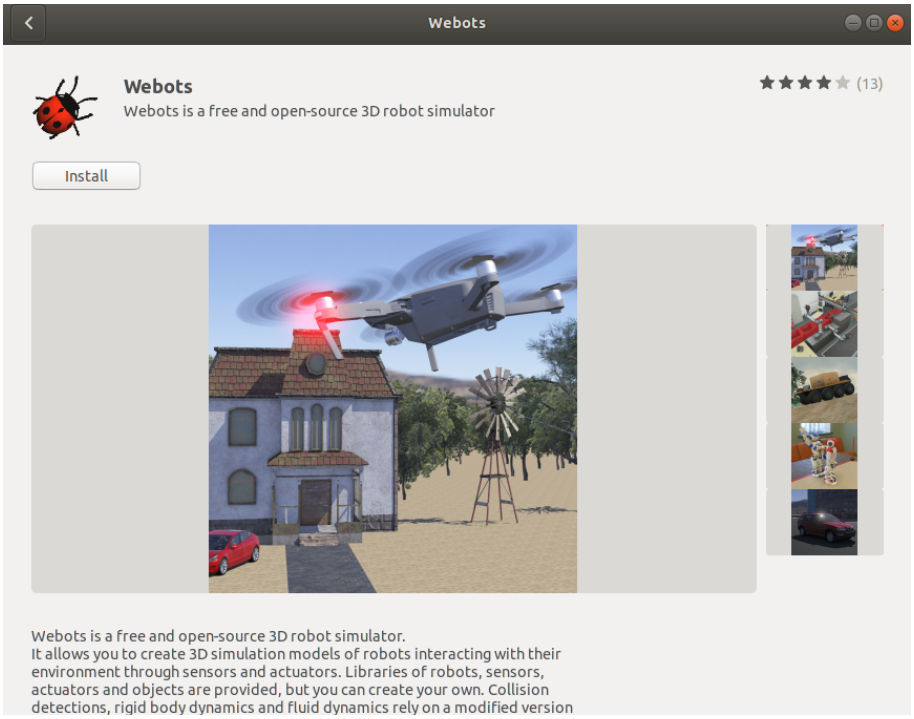

Figure 10: Installing Webots.

C Installing ROS_wrapper for kinect V-2 (Xbox one)

Firstly, we need to download the libraries by typing the following command :

```
$ git clone https://github.com/OpenKinect/libfreenect2.git
```

1. **Kinect v2 SDK Installation :**

   SDK will be installed by default in /usr/local. Run in the terminal:

   ```
   $ cd libfreenect2
   $ mkdir build
   ```

```
$ cd build
$ cmake ..
$ make
$ sudo make install
```

Note : you can get an error like "CMake Error at /usr/share/cmake-3.9/Modules/FindPackageHandleStandardArgs. cmake:137 (message):Could NOT find TurboJPEG (missing: TurboJPEG_INCLUDE_DIRS TURBOJPEG_WORKS)...), Then try:

```
$ sudo apt-get install libturbojpeg0-dev
```

After installing, please delete the build directory and rebuild following the instructions from 3. Set the udev rules for communicating with device:

```
$ sudo cp libfreenect2/platform/linux/udev/90-kinect2.rules /etc/udev/
rules.d/
```

Replug the Microsoft Kinect Xbox One. Then run in the build directory:

```
$ ./bin/Protonect
$ ./bin/Protonect gl # to test OpenGL support.
$ ./bin/Protonect cl # to test OpenCL support.
$ ./bin/Protonect cpu # to test CPU support.
```

2. **ROS installation for Kinect xbox one:**

   Now for ROS wrapper installation for kinect v2, execute:

```
$ cd  /catkin_ws/src/
$ git clone https://github.com/ArghyaChatterjee/SDK-Installation-ROS-
Wrapper-for-Kinect-v2-on-Ubuntu-18.04.git
$ cd iai_kinect2
$ rosdep install --from-paths  /catkin_ws/src/SDK-Installation-ROS-
Wrapper-for-Kinect-v2-on-Ubuntu-18.04 --ignore-src -r
```

   (Note: you can get an error like "ERROR: the following packages/stacks could not have their `rosdep` keys resolved to system dependencies:..."), just ignore it and continue.

```
$ cd  /catkin_ws
```

```
$ catkin_make -DCMAKE_BUILD_TYPE="Release"
```

After successful installation, source the environment:

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

On the same terminal, execute:

```
$ roslaunch kinect2_bridge kinect2_bridge.launch
```

On another terminal, execute:

```
$ source  /catkin_ws/devel/setup.bash
$ rosrun kinect2_viewer kinect2_viewer
```

3. **Visualizing RTABMAP and Rviz:**

   If you want to see the rtabmap, open a new terminal and make sure no other program is running. Execute:

   ```
   $ sudo apt install ros-melodic-rtabmap-ros
   $ cd  /catkin_ws
   $ source  /catkin_ws/devel/setup.bash
   $ roslaunch kinect2_bridge kinect2_bridge.launch
   ```

   In another terminal, execute:

   ```
   $ roslaunch rtabmap_ros rgbd_mapping_kinect2.launch resolution:=hd
   ```

   If you want to see the rtabmap with rviz:

   One way: For openning with default rviz, move to /opt/ros/melodic/share/rtabmap_ros/launch directory and open rgbd_mapping_kinect2.launch file. Change rviz (line number 25) to "true" and save the file. Now open a new terminal and make sure no other program is running. Execute:

   ```
   $ cd  /catkin_ws
   $ source  /catkin_ws/devel/setup.bash
   $ roslaunch kinect2_bridge kinect2_bridge_default_rviz.launch
   ```

   Second way: If you want to see the rtabmap with custom rviz , open a new terminal and make sure no other program is running. Execute:

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

```
$ roslaunch kinect2_bridge kinect2_bridge_custom_rviz_qhd.launch #to open
qhd video stream in ROS.
$ roslaunch kinect2_bridge kinect2_bridge_custom_rviz_hd.launch #to open
hd video stream in ROS.
$ roslaunch kinect2_bridge kinect2_bridge_custom_rviz_sd.launch #to open
sd video stream in ROS.
```

D P-3at obstacle avoidance with lidar code

```
/*
 * Copyright 1996-2021 Cyberbotics Ltd.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/*
 * Description:  Example of Sick LMS 291.
 *               The velocity of each wheel is set
 *               according to a Braitenberg-like algorithm which takes the
                 values returned by the Sick as input.
 */
```

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <webots/lidar.h>
#include <webots/motor.h>
#include <webots/robot.h>

#define TIME_STEP 32
#define MAX_SPEED 6.4
#define CRUISING_SPEED 5.0
#define OBSTACLE_THRESHOLD 0.1
#define DECREASE_FACTOR 0.9
#define BACK_SLOWDOWN 0.9

// gaussian function
double gaussian(double x, double mu, double sigma) {
  return (1.0 / (sigma * sqrt(2.0 * M_PI))) * exp(-((x - mu) * (x - mu))
  / (2 * sigma * sigma));
}

int main(int argc, char **argv) {
  // init webots stuff
  wb_robot_init();

  // get devices
  WbDeviceTag lms291 = wb_robot_get_device("Sick LMS 291");
  WbDeviceTag front_left_wheel = wb_robot_get_device("front left wheel");
  WbDeviceTag front_right_wheel = wb_robot_get_device("front right wheel");
  WbDeviceTag back_left_wheel = wb_robot_get_device("back left wheel");
  WbDeviceTag back_right_wheel = wb_robot_get_device("back right wheel");
```

```
// init lms291
wb_lidar_enable(lms291, TIME_STEP);
const int lms291_width = wb_lidar_get_horizontal_resolution(lms291);
const int half_width = lms291_width / 2;
const int max_range = wb_lidar_get_max_range(lms291);
const double range_threshold = max_range / 20.0;
const float *lms291_values = NULL;

// init braitenberg coefficient
double *const braitenberg_coefficients = (double *)malloc(sizeof(double) *
lms291_width);
int i, j;
for (i = 0; i < lms291_width; ++i)
  braitenberg_coefficients[i] = gaussian(i, half_width, lms291_width / 5);

// init motors
wb_motor_set_position(front_left_wheel, INFINITY);
wb_motor_set_position(front_right_wheel, INFINITY);
wb_motor_set_position(back_left_wheel, INFINITY);
wb_motor_set_position(back_right_wheel, INFINITY);

// init speed for each wheel
double back_left_speed = 0.0, back_right_speed = 0.0;
double front_left_speed = 0.0, front_right_speed = 0.0;
wb_motor_set_velocity(front_left_wheel, front_left_speed);
wb_motor_set_velocity(front_right_wheel, front_right_speed);
wb_motor_set_velocity(back_left_wheel, back_left_speed);
wb_motor_set_velocity(back_right_wheel, back_right_speed);

// init dynamic variables
```

```
double left_obstacle = 0.0, right_obstacle = 0.0;

// control loop
while (wb_robot_step(TIME_STEP) != -1) {
  // get lidar values
  lms291_values = wb_lidar_get_range_image(lms291);
  // apply the braitenberg coefficients on the resulted values of the lms291
  // near obstacle sensed on the left side
  for (i = 0; i < half_width; ++i) {
    if (lms291_values[i] < range_threshold)  // far obstacles are ignored
      left_obstacle += braitenberg_coefficients[i] * (1.0 - lms291_values[i]
      / max_range);
    // near obstacle sensed on the right side
    j = lms291_width - i - 1;
    if (lms291_values[j] < range_threshold)
      right_obstacle += braitenberg_coefficients[i] * (1.0 - lms291_values[j]
      / max_range);
  }
  // overall front obstacle
  const double obstacle = left_obstacle + right_obstacle;
  // compute the speed according to the information on
  // obstacles
  if (obstacle > OBSTACLE_THRESHOLD) {
    const double speed_factor = (1.0 - DECREASE_FACTOR * obstacle) *
    MAX_SPEED / obstacle;
    front_left_speed = speed_factor * left_obstacle;
    front_right_speed = speed_factor * right_obstacle;
    back_left_speed = BACK_SLOWDOWN * front_left_speed;
    back_right_speed = BACK_SLOWDOWN * front_right_speed;
  } else {
    back_left_speed = CRUISING_SPEED;
```

```
      back_right_speed = CRUISING_SPEED;
      front_left_speed = CRUISING_SPEED;
      front_right_speed = CRUISING_SPEED;
    }
    // set actuators
    wb_motor_set_velocity(front_left_wheel, front_left_speed);
    wb_motor_set_velocity(front_right_wheel, front_right_speed);
    wb_motor_set_velocity(back_left_wheel, back_left_speed);
    wb_motor_set_velocity(back_right_wheel, back_right_speed);

    // reset dynamic variables to zero
    left_obstacle = 0.0;
    right_obstacle = 0.0;
  }

  free(braitenberg_coefficients);
  wb_robot_cleanup();

  return 0;
}
```

Bibliography

[1] Lentin Joseph. Robot Operating System (ROS) for Absolute Beginners. Springer, 2018.

[2] ROS/Concepts wiki. http://wiki.ros.org/ROS/Concepts.

[3] ActivMedia Robotics. Pioneer 3 operations manual, 2006.

[4] Adept Mobile Robots. "pioneer 3-dx datasheet. https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf, 2011.

[5] Adept Mobile Robots. "pioneer 3-at datasheet. https://www.generationrobots.com/media/Pioneer3AT-P3AT-RevA-datasheet.pdf, 2011.

[6] Krzysztof Kozłowski and Dariusz Pazderski. Modeling and control of a 4-wheel skid-steering mobile robot. International journal of applied mathematics and computer science, 14(4):477–496, 2004.

[7] MD Tanvir Ahmed Ratul, Mohd Saiful Azimi Mahmud, Mohamad Shukri Zainal Abidin, and Razman Ayop. Design and development of gmapping based slam algorithm in virtual agricultural environment. In 2021 11th IEEE international conference on control system, computing and engineering (ICCSCE), pages 109–113. IEEE, 2021.

[8] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the rgb-d slam system. In 2012 IEEE international conference on robotics and automation, pages 1691–1696. IEEE, 2012.

[9] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. IEEE transactions on robotics, 31(5):1147–1163, 2015.

[10] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. IEEE transactions on pattern analysis and machine intelligence, 40(3):611–625, 2017.

[11] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 2174–2181. IEEE, 2015.

[12] Cyberbotics ltd. https://cyberbotics.com/.