



University Mohamed Khider of Biskra

Faculty of Science and Technology

Electrical Engineering Department

Final Year Project Report in View of Obtaining

The Diploma Of

MASTER

Science and Technology

Embedded System

Theme:

LIDAR Mapping and Localization

Presented and Supported By:

Houhou Mounir

On: October ,2020

Jury:

❖ Mme: Hendaoui Mounira	MCA University of Biskra	President
❖ Mr: Rahmani Nacereddine	MAA University of Biskra	Supervisor
❖ Mme: Terghini Ouarda	MCB University of Biskra	Examiner

Academic Year: 2021 – 2020



University Mohamed Khider of Biskra

Faculty of Science and Technology

Electrical Engineering Department

Final Year Project Report in View of Obtaining

The Diploma Of

MASTER

Science and Technology

Embedded System

Theme:

LIDAR Mapping and Localization

Presented and Supported By:

Houhou Mounir

On: October ,2020

Presented by: Favorable opinion of the supervisor: Signature

Houhou Mounir RAHMANI Nacereddine .

Favorable opinion of the President of Jury: Signature

Hendaoui Mounira .

Stamp and signature

.



University Mohamed Khider of Biskra

Faculty of Science and Technology

Electrical Engineering Department

Final Year Project Report in View of Obtaining

The Diploma Of

MASTER

Science and Technology Embedded System

Theme:

LIDAR Mapping and Localization

Proposed by: RAHMANI Nacereddine

Directed by: RAHMANI Nacereddine

Abstract (English and Arabic)

The more technology gets harder the more science makes a step forward specially in robotics, to be more specific self-driving cars (autonomous vehicle), We need them to navigate throw lands and outside and inside doors application. Mapping and localization can easily be solved by autonomous robot who can create a map and gives us a localization of itself. It can be more useful in hard daily jobs like discovering caves and map old buildings; it consists of wheels and motors, circuits, sensors to detect the surroundings. Using some mapping and navigation algorithms. The goal of this project is to learn how to develop your robot and test it even before in real application and make this robot map a surface and navigate on autonomously.

كلما ازدادت صعوبة التكنولوجيا كلما استطاع العلم من اتخاذ خطوة إلى الأمام وخاصة في مجال الروبوتات، تحديدا في السيارات ذاتية القيادة (سيارة ذاتية التحكم)؛ التي نحتاج إليها للملاحة في الأراضي و المساحات الداخلية والخارجية حيث من الممكن بسهولة انشاء عملية التخطيط والتوضع بواسطة إنسان آلي ذاتي التحكم قادر على إنشاء خريطة. وقد يكون أكثر فائدة في المهام اليومية الصعبة مثل اكتشاف الكهوف ورسم خرائط للمباني القديمة؛ فهو يتألف من عجلات ومحركات ودارات كهربائية وأجهزة استشعار لاكتشاف المناطق المحيطة؛ و يمكنه استخدام خوارزميات للتخطيط والتنقل. الهدف من هذا المشروع هو تعلم كيفية تطوير الروبوت الخاص بك واختباره حتى قبل انشاءه في التطبيق الحقيقي وجعل هذا الروبوت، ينشئ خريطة سطح ما والتنقل فيها على نحو مستقل

Acknowledgement

First and supreme, I thank Allah the Almighty for this blessed work without an enough effort I will be lost thankful for protection from all diseases to be strong all the way.

I give my gratitude and appreciation to my sympathetic supervisor Mr. RAHMANI Nacereddine and for his precious time and guidance and especially the encouragement, patience and wonderful support.

I also thank specially Ouamane Fatma Zohra who helped me a lot in my work without forgetting my friends Ikhenache Wail, Houhou Ihssane who helped me to give such work supporting my back whenever I turn.

Appreciation also goes to all the administrators and teachers at the Department of electronics in the University of Biskra for their hard work and dedication and without forgetting the leader of laboratory Mr. Hamza thanks for all your support.

Finally, a heartfelt gratitude to my family and all my friends and colleagues for the encouragement and help they offered.

Dedication

I dedicate this work to my father and mother and to my brother and sister and too all professors and my friends and without forgetting any one gives help.

For all who believes in those sayings

“Give your best be good “hard work gives you good result and say good words to your classmates this will make your work be greater than just good, after being in this department in university of biskra I learned a lot of things about electronics and I miss some moments which is get to know yourself. never turn your back for someone who says to you “help me” because helping someone will shape your skills and you gain more. For me helping friends is the best thing happened in all semesters.

“Be good with family “they raise and teach you secrets of life teach you to fall and rise again. Teach you love and care so be great full for all support you get from them they laugh with you they pray for you they hope that your success through your path.

“Covid” I pray every day asking Allah to spare us and forgive us from all sins. And ask rahma for all the dead from covid to be chahid in Allah hands.

List of Content

Acknowledgement	
Dedication	
List of Figures	I
List of Tables	VIII
General Introduction	2
I. Robotics	4
I.1 Introduction Robotics	4
I.2 Characteristics and types of robots.....	4
I.2.1 Characteristics:	4
I.2.2 Types of Robots:	6
I.3 Scope And limitations Of Robots:.....	7
I.4 advantages and disadvantages:	7
I.5 Classification of robots:.....	8
I.5.1 Some robot's pictures:.....	9
I.6 Robotic Most Important Parts:	11
I.6.1 Mechanism:	11
I.6.2 Electronics Parts:.....	12
I.6.3 Software in Robots:	13
I.7 Conclusion:	14
II. Localization and Mapping on Robots:.....	16
II.1 Introduction:	16
II.2 How SLAM started:	17

II.3 The Futures of SLAM (Simultaneous Localization And Mapping):....	18
II.3.1 What is and Why we use SLAM?.....	18
II.3.2 SLAM Applications?	21
II.3.3 How SLAM Works?	23
II.4 How We Implement SLAM System in Robots:	24
II.4.1 Content And Design Of SLAM Problems:	31
II.4.2 Probabilistic SLAM:	32
II.4.3 Solutions to The SLAM Problem:	35
II.5 The Relation with Next Chapter:	37
II.6 Conclusion.....	37
III. Robotic Operation System:	39
III.1 Introduction.....	39
III.2 ROS Goals:	40
III.3 Operations System:	41
III.4 ROS Context:	41
III.5 ROS Versions(distribution):	42
III.6 Ubuntu Install of ROS kinetic	44
III.6.1 Installation:	44
III.6.2 Configure Your Ubuntu Repositories.....	44
III.6.3 Setup Your Sources List	44
III.6.4 Set Up Your Keys.....	44
III.6.5 Installation	45

III.6.6	Environment Setup	46
III.6.7	Dependencies for Building Packages	46
III.6.8	Initialize Rosdep	47
III.6.9	Build Farm Status	47
III.7	Start Your Project:	47
III.8	Why Using ROS?.....	47
III.8.1	ROS in General:.....	47
III.8.2	ROS Packages for Everything:.....	48
III.8.3	ROS Has Great Simulation Tools:	48
III.8.4	You Can Control Multiple Robots With ROS:	48
III.8.5	ROS Is Light:.....	48
III.8.6	More Compatible with Your Ideas:.....	48
III.9	Conclusion:	48
IV.	Application and Discussion.....	50
IV.1	Introduction:	50
IV.2	Component and Design of Robot:	50
IV.2.1	Objectives:.....	51
IV.2.2	Choosing the Component:.....	51
IV.3	Our Robot Electronic Circuit assembly:	59
IV.4	Getting Started:.....	59
IV.4.1	Installing Linux To Fix Our ROS Tools:	60
IV.5	Applying Some Algorithms On Linux Using The Terminal:	63

IV.6 Build Our Robots In Linux.....	69
IV.6.1 Creating Our Work Space (robot folder):	70
IV.6.2 Creating Src File Inside the Working Space And Compile The Folder: 70	
IV.6.3 Creating our robot packages:.....	72
IV.6.4 Robot Coding:	74
IV.6.5 Compiling Our Project Work Space:	82
IV.6.6 Testing Our Sensor (RPLIDAR):	87
IV.7 How Is Our Robot Attached?	90
IV.8 Joints of Our Robot:	94
IV.9 Test Our Robot (Mapping and localization):	95
IV.9.1 Mapping:.....	95
IV.9.2 Code the control file:	96
IV.9.3 The Code Of Navigation Package:.....	98
IV.9.4 Start seeing results.....	101
IV.9.5 Navigation:	106
IV.10 Measure distances in maps	109
IV.10.1 Creating our package:.....	110
IV.10.2 Results:	111
IV.11 Build Our Robot on Reality:.....	112
IV.11.1 Preparing the Raspberry Pi:.....	112
IV.11.2 Test the sensor:	113

IV.11.3	Import hector slam:	113
IV.11.4	Compile the Files:	113
IV.11.5	Run Mapping Algorithm:	113
IV.12	Conclusion:	117
General conclusion	119
Bibliographie	120

List of Figure

I. Robotics

Figure I.1 Classification of Robots by Environment and Mechanism of Interaction.	8
Figure I.2 Classification of Robots by Application Field [6].	9
Figure I.3 Industrial Robots.	9
Figure I.4 Humanoid Robots.	10
Figure I.5 Autonomes Mobile Robots.	10
Figure I.6 Learning Robot.	10
Figure I.7 Learning Robot Turtelbot.	11
Figure I.8 Robot Arm.	11
Figure I.9 Moving Robot Arm.	12
Figure I.10 Robot Electronic Parts(face).	12
Figure I.11 Robot Electrical Wiring.	13
Figure I.12 Coding Example.	13

II. Localization and mapping on robots

Figure II.1 This represent a SLAM results (mapping and localization).	17
Figure II.2 The Birth Of SLAM [10].	18
Figure II.3 represent SLAM technic to detect object using camera.	19
Figure II.4 SLAM Application (Localization).	20
Figure II.5 3D SLAM Map.	20

Figure II.6 Google's Self-Driving Car's Perception System. From IEEE Spectrum's.	21
Figure II.7 Plains Using SLAM [12].	22
Figure II.8 Plains Result Using SLAM Algorithms [12].	22
Figure II.9 3D Map Using SLAM (Mapping and Localization Technic) [12].	22
Figure II.10 Educations Environment Using Robot And Visual SLAM Technologie.....	23
Figure II.11 2D Map. SLAM Result Using Virtual Simulation In Linux.	23
Figure II.12 3D Map. SLAM Result Using Virtual Simulation in Linux.	24
Figure II.13 Robot Car with LIDAR Sensor.	25
Figure II.14 Raspberry pi 3 (Microcontroller or CPU Unite).	25
Figure II.15 Smart Vacuum Cleaner.....	26
Figure II.16 Old Vacuum Cleaner.	26
Figure II.17 Self Driving Car With LIDAR And Camera.....	27
Figure II.18 Control Unity.....	27
Figure II.19 RPLIDAR A1 Sensor.	28
Figure II.20 Overall Flowchart Of A Particle Filter-Based Algorithm.....	28
Figure II.21 The essential SLAM problem.....	32
Figure II.22 Time-update.....	33
Figure II.23 Measurement Update.....	33

III. Robotic Operation System

Figure II.24 Spring network analogy [8].	35
Figure II.25 Realization of Robot Path in The Fast -SLAM Algorithm.	36
Figure III.1 Roscore Launch in Terminal.	40
Figure III.2 ROS in Windows 10.	41
Figure III.3 ROS Topic Mechanism.	42
Figure III.4 ROS List of Distribution [24].	43

IV. Application and Discussion

Figure IV.1 Straight BO Motor.	52
Figure IV.2 Wheels.	52
Figure IV.3 Transparent chassis.	53
Figure IV.4 Tools and equipment's.	53
Figure IV.5 Raspberry pi 3 B+ Module.	54
Figure IV.6 H-bridge Driver.	55
Figure IV.7 L298 Driver.	55
Figure IV.8 Raspberry pi 3 WIFI.	55
Figure IV.9 3.7 Battery 8800 mAh.	56
Figure IV.10 RPLIDAR System Composition.	56
Figure IV.11 RPLIDAR With USB Adapter.	57
Figure IV.12 RPLIDAR A1 Working Schematic.	58
Figure IV.13 An Example About How RPLIDAR Work Inside Doors.	58

Figure IV.14 Robot Circuit.....	59
Figure IV.15 Robot Control Direction Circuit (H-bridg and Motors).	59
Figure IV.16Ubuntu 16.04 Linux System.	60
Figure IV.17Terminal Window.	61
Figure IV.18Setting the Source.	61
Figure IV.19 Preparing Keys.....	61
Figure IV.20 Updating.....	62
Figure IV.21Installing.....	62
Figure IV.22 Adding Dependencies.	63
Figure IV.23 initialize.....	63
Figure IV.24 Some Tools In ROS.	64
Figure IV.25 Source the File.....	64
Figure IV.26Creating Work Space(mybot_ws).....	65
Figure IV.27 Writing the Subscriber Node.....	66
Figure IV.28 Writing the Publisher Node.....	66
Figure IV.29 First Run the Roscore.....	67
Figure IV.30Second Rosrun Beginner_Tutorials Talker.....	68
Figure IV.31Third Rosrun Beginner_Tutorials Listener.....	68
Figure IV.32 Topic from Terminal Command.	68
Figure IV.33 Robot Final Goals.	69
Figure IV.34 Working Space Folder.	70

Figure IV.35 Src Robot Code And Configuration File.	70
Figure IV.36 Building The Project And Source The File.....	71
Figure IV.37 Mybot_Ws File After The Compiling.....	71
Figure IV.38 Create Mybot_Description Package.	72
Figure IV.39 Mybot_Description Package.	72
Figure IV.40 src file with robot packages.	73
Figure IV.41 Mybot_Discription Package folder.	73
Figure IV.42 mybot.gazebo Package folder.	74
Figure IV.43 mybot_world.launch.	75
Figure IV.44 World Launch File.	75
Figure IV.45 Gazebo Application From The Launch File	76
Figure IV.46 Colors File (materials.xacro).....	77
Figure IV.47 Gazebo Application Code Plugins(gazebo.xacro).	78
Figure IV.48 3D Design Robot Code (mybot.xacro).	79
Figure IV.49 Right Wheel Code Link(mybot.xacro).	80
Figure IV.50 Right Wheel Joint (mybot.xacro).....	81
Figure IV.51 The Difference Between The Right Wheel And Right Back Wheel(mybot.xacro).	82
Figure IV.52 how to launch the project file run_gazebo.sh.	82
Figure IV.53 The File Who Launch Our Robot Packages.	83
Figure IV.54 3D Robot Design Displayed In Gazebo Application.....	83

Figure IV.55 Rviz Launch File.	84
Figure IV.56Rviz Visualization Application.	84
Figure IV.57 Setting Map to Odom.	85
Figure IV.58 Adding Our Robot Model from This List.	85
Figure IV.59 3D Robot Model From Our Description Package In Rviz.	86
Figure IV.60 Get the Laser Scan.	87
Figure IV.61 Select The Topic of laser scan.	88
Figure IV.62 Adding Some Spheres and Cube.	89
Figure IV.63 The Readings Is Imported From Gazebo Throw The Topic To Rviz.	89
Figure IV.64 Results After Moving The Shapes.	90
Figure IV.65Open the Robot Attach Tree.	90
Figure IV.66 Frames Results.	91
Figure IV.67 Left Side from The Frames.	91
Figure IV.68The Right Side from The Frame.	92
Figure IV.69 Laser Topic.	93
Figure IV.70 Publisher and Subscriber Information Of /Mybot/Laser/Scan Topic.	93
Figure IV.71 Wheel Joint.	94
Figure IV.72 ControlFile Configuration.	96
Figure IV.73 Launch File Of The Controller.	96

Figure IV.74	Navigation Package Config File.	97
Figure IV.75	mybot_teleop.launch-----gmapping_demmo.launch Files.....	98
Figure IV.76	Map Loader File.....	99
Figure IV.77	Rviz Extra Files.....	100
Figure IV.78	Launching Gazebo With Our Robot Module With Room full with spher And A Boxes.	101
Figure IV.79	Our Robot Module Inside A Room with A Box.	102
Figure IV.80	Terminal Window with Gmapping Algorithm.	102
Figure IV.81	Rviz With Mapping Results.	103
Figure IV.82	The Robot Controller Keys.	103
Figure IV.83	After Moving through all obstacles.	104
Figure IV.84	first Map sample without obstacles.....	104
Figure IV.85	Saving The Map Ender The Name Test1_Map.	105
Figure IV.86	Rostopic List.	105
Figure IV.87	Loading The Map Of Navigation In Rviz.....	107
Figure IV.88	Camera on Our Robot.	107
Figure IV.89	Robot While Moving to The Target.....	108
Figure IV.90	lib_gazebo_ros_planar_move robot four wheels driver	109
Figure IV.91	The Robot While Taking A Turn through the obstacle	109
Figure IV.92	Compiling Our File.	110
Figure IV.93	Contours That Exist on Our first map Picture.	111

Figure IV.94 Zoom The Map.....	111
Figure IV.95 Ubuntu 16.04 Raspberry Compatible.	112
Figure IV.96 RPLIDAR And Hector Slam Repositories.	113
Figure IV.97 Launch Rplidar Package.	114
Figure IV.98 Launch the Hector Slam Package.	114
Figure IV.99 Results In Rviz Application After Launching Hector Slam (Mapping).....	115
Figure IV.100 Second Try Launching Hector Slam Algorithm Shown in Rviz.	116
Figure IV.101 Results After Solving the Problem.	116

List of tables

Table I.1 Advantages and Disadvantages Of human And Robot Technology...7	7
Table II.1 Some of SLAM Map Types Vs Algorithms	29
Table II.2 EKF Operations for Achieving SLAM.	36

General Introduction

General Introduction

Maps have old history, from ages people rely on them through drawing on rocks or animals' leathers to map the world they use it in army planning and commercial fields. Now with the advance of technology mapping get jumped from animal leathers to our screen computers with deferent designs we can tell the time and potion from the attitude and longitude. From now maps are used in every domain, even the architecture like buildings schematics from rivers maps to water channels, it's important to make our life simplr.

In our work were going to learn how to build robots that can make a map for certain environment and how they function and which tools you need, to form his shape. The important thing is how can the robot sense it's surrounding by using sensor and actuators. The main thing in this project is divided into two parts how you can use virtual world to simulate your robot in 3D application (gazebo, rviz) and try to build our real robot with sensors.

In the first chapter were going to see about robots in general from appearance and their applications characteristics, advantage of using robots, deferent type of robots and most important parts from it. In the next chapter 2(Two) we will be talking about mapping and localization of the robot and how it will be applicated and implemented it in our robot system and deferent algorithms Principe and problems. In the third chapter we ill discuses a new application with ROS (Robotic Operation System) a definition about it and how you install it and work with it in Linux environment. Maybe it will be confusing but you will be informed with every detail. In the fourth chapter you will see the simulated robot in 3D module with all sensors implemented and real practical robot with the same simulated application additional to that, were going to shape the same robot in real application with the same sensor (LIDAR). Our robot consists of raspberry pi 3 which is the robot brain who control the direction and send the sensor data for his wide application and we can install on it all ROS tools.

Now the main question is how to begin the project and how we end up with great result? the answer is in the following four chapters.

Chapter 1

Robotics

I. Robotics.

I.1 Introduction Robotics

When we think about robotics first thing we think about is automation. We all know that robot do tasks on thire own without himain interfering, except for initial programming and instruction set being provided to them by the user. Robots can be devide into two groups moving and stasionary robots , what I have seen in my childhood when we were on a visit to a milk processing factory, most close, robots can do packaging for certain materials like milk or poudrer boxes. The packaging material running through the machine, each time half a liter of milk falls into the roll and then a mechanism in the machine seals and cuts the packet [1].

I.2 Characteristics and types of robots

There are some characteristics and types of robots like the following [2]:

I.2.1 Characteristics:

- **Appearance:**

Robots have a physical body. They are held by the structure of their body and are moved by their mechanical parts. Without appearance and mechanism, robots will be just a software program and some electronics parts.

- **Brain:**

Another name of brain in robots is On-board control unit. Using this robot receive information and sends commands as output. With this control unit robot knows what to do else it'll be just a remote-controlled machine.

- **Sensors:**

The use of these sensors in robots is to gather info from the outside world and send it to Brain. Basically, these sensors have circuits in them that produces the voltage in them.

•Sensing:

In the first your robot will be able of sensing its own environment. It will do this with different way from you. Giving your robot sensors: light sensors like your eyes touch and pressure sensors like your hands chemical sensors (nose), Sonar sensors (ears) [3].

• Actuators:

The robots move and the parts with the help of these robot's move is called Actuators. Some examples of actuators are motors, pumps, and compressor etc. The brain tells these actuators when and how to respond or move [1].

• Program:

Robots only works or responds to the instructions which are provided to them in the form of a program. These programs only tell the brain when to perform which operation like when to move, produce sounds etc. These programs only tell the robot how to use sensors data to make decisions.

•Behavior:

Robots behavior is decided by the program which has been built for it. Once the robot starts making the movement, one can easily tell which kind of program is being installed inside the robot.

•Movement:

Your robot needs to be capable of moving around in environment. whether a robot with wheels or legs the robot needs to be able to move.

•Energy:

A robot needs to be powered by itself. Maybe by a solar power and battery. The way your robot will get his energy it depends on how it functions.

❖Intelligence:

A robot somehow needs to think (be smart) this is when the programming begins and be formed by programmers and receive commands on how it will be functioning.

1.2.2 **Types of Robots:**

There is many of robot's types like following [1]:

•**Articulated:**

The feature of this robot is its rotary joints and range of these are from 2 to 10 or more joints. The arm is connected to the rotary joint and each joint is known as the axis which provides a range of movements.

•**Cartesian:**

These are also known as gantry robots. These have three joints which use the Cartesian coordinate system i.e x, y, z. These robots are provided with attached wrists to provide rotatory motion.

•**Cylindrical:**

These types of robots have at least one rotatory joint and one prismatic joint which are used to connect the links. The use of rotatory joints is to rotate along the axis and prismatic joint used to provide linear motion.

•**Polar:**

These are also known as spherical robots. The arm is connected to base with a twisting joint and have a combination of 2 rotatory joint and one linear joint.

•**Scara:**

These robots are mainly used in assembly applications. Its arm is in cylindrical in design. It has two parallel joints which are used to provide compliance in one selected plane.

•**Delta:**

The structure of these robots is like spider-shaped. They are built by joint parallelograms that are connected to the common base. The parallelogram moves in a dome-shaped work area. These are mainly used in food and electrical industries

I.3 Scope And limitations Of Robots:

The people should be aware about robots benefits and if they will be helpful or not. Robots can go to planets removing them from distance and exploring the space. they can pick information from people to spy on them so robots have two parts so were going to see advantage and disadvantage of robots [4].

I.4 advantages and disadvantages:

Now were going to see Advantages and disadvantages of human and robot technology we did take medical field as an example.in small table [5]:

	Surgeons	Robots
Advantages	Task versatility Judgment experience Hand-eye coordination Dexterity at millimeter-to-centimeter scale Many sensors with seamless data fusion Quickly integrate extensive and diverse qualitative information Rudimentary haptic abilities Good judgement Easy to instruct and debrief	Repeatability Stable and untiring Resistance to ionizing radiation Use of diverse sensors in control (e.g., chemical, force, acoustic) Optimized for particular environment Spatial hand-eye transformations handled with ease: Improved dexterity Manage multiple simultaneous tasks 3-D visualization Good degrees freedom May be sterilized Good geographical accuracy
Disadvantages	Tremors Fatigue Limited dexterity Limited geometric accuracy Imprecision Variability in skill, age, state of mind Inability to process quantitative information easily ineffective at sub-millimeter scale Limited sterility Susceptible to radiation and infection Large operation room space required	Expensive Cumbersome No judgement Absence of haptic sensation Large Inability to process qualitative information Not versatile Technology still in infancy More evidence of effectiveness needed Limited to simple procedures Difficult to construct and debug

Table I.1 Advantages and Disadvantages Of human And Robot Technology.

The above table was adapted from information provided by Stoianovici (2000), Lanfranco et al (2003). and Camarillo and al (2004).

I.5 Classification of robots:

Robots can be classified based on their environment and do an action according to the next figure. (**Figure I.2**) represent the deference between mobile and fixed robots. The fixed robots are commonly used in industrial field to perform specific tasks such soldering or painting parts in car manufacturing plants and many in medical field to manipulate a surgery in hospitals according to their precision.

By contrast, mobile robots are used to move around and perform tasks in large scale. They deal with more complex situation and do it effective and fast way possible. Examples of mobile robots are robotic vacuum cleaners and self-driving cars [6].

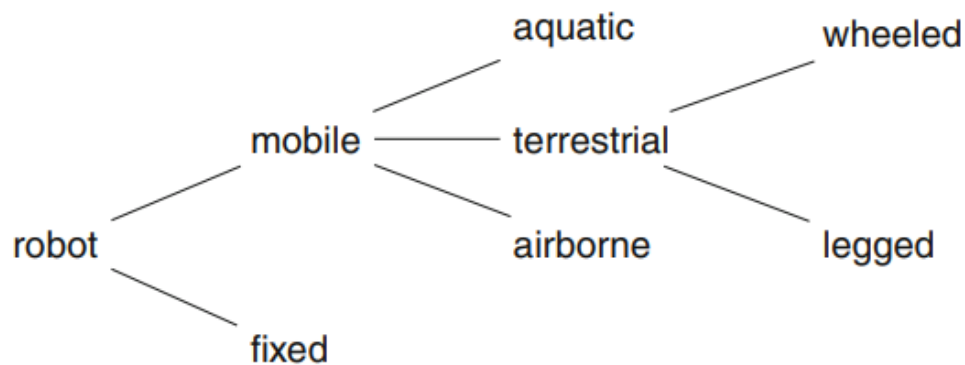


Figure I.1 Classification of Robots by Environment and Mechanism of Interaction.

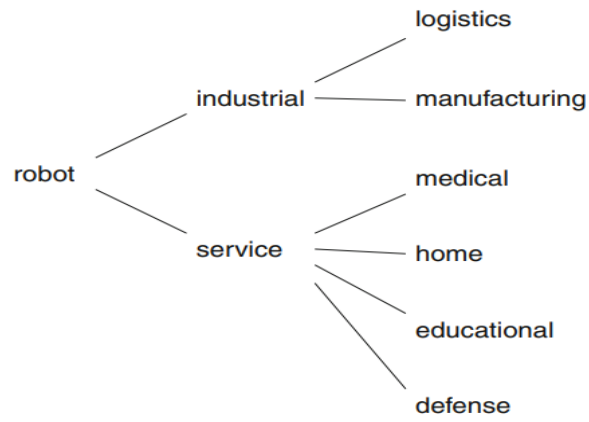


Figure I.2 Classification of Robots by Application Field [6].

I.5.1 Some robot's pictures:



Figure I.3 Industrial Robots.

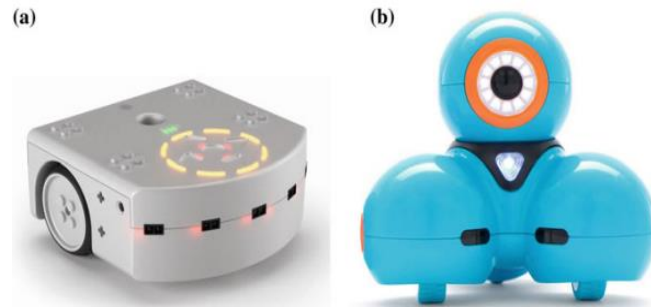


Figure I.4 Humanoid Robots.



Figure I.5 Autonomes Mobile Robots.



Figure I.6 Learning Robot.

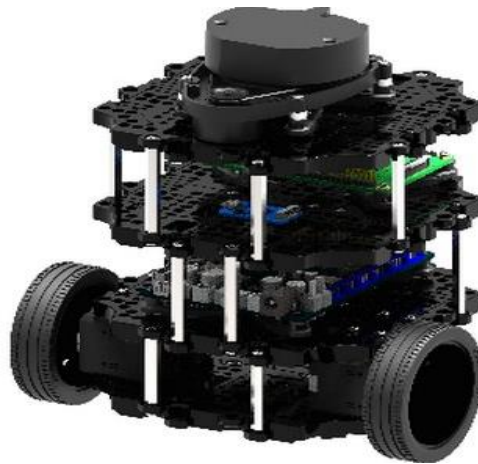


Figure I.7 Learning Robot Turtlebot.

I.6 Robotic Most Important Parts:

I.6.1 Mechanism:

A robot is a machine can do a physical work with interfering with their surroundings. physical means manipulation, locomotion, and any action will affect the environment or change the state of the robot (robot will move around). A robot has a lot of forms of mechanism parts to do certain action and can move around easily.



Figure I.8 Robot Arm.



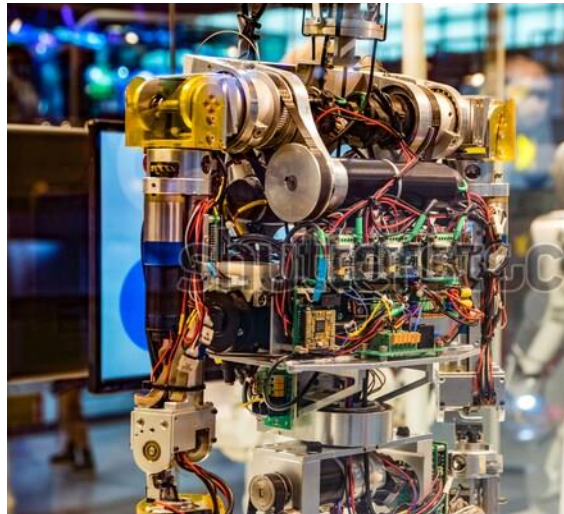
Figure I.9 Moving Robot Arm.

I.6.2 Electronics Parts:

Some robot just works with mechanism and some need electronics part like the previews Pictures **Figure (I.7,I.6)**. It represented in microcontroller or CPU unity to calculate and make sure the robot will move around by controlling motors. Or to read some sensors and show the collected data on a screen. Or give an order to the robotic arm to grab or move some stuff from the surrounding of the robot.



Figure I.10 Robot Electronic Parts(face).



www.shutterstock.com · 683304139

Figure I.11 Robot Electrical Wiring.

I.6.3 Software in Robots:

Without command robot will not do a single step without some coding (software or algorithms). The code it will be written by engineers or anyone has some level at coding. there is a lot of code language such as python, java, C++, C#. ect. the code will be uploaded into the robot electrical system (CPU). So, the CPU will execute the code so the robot can move around.

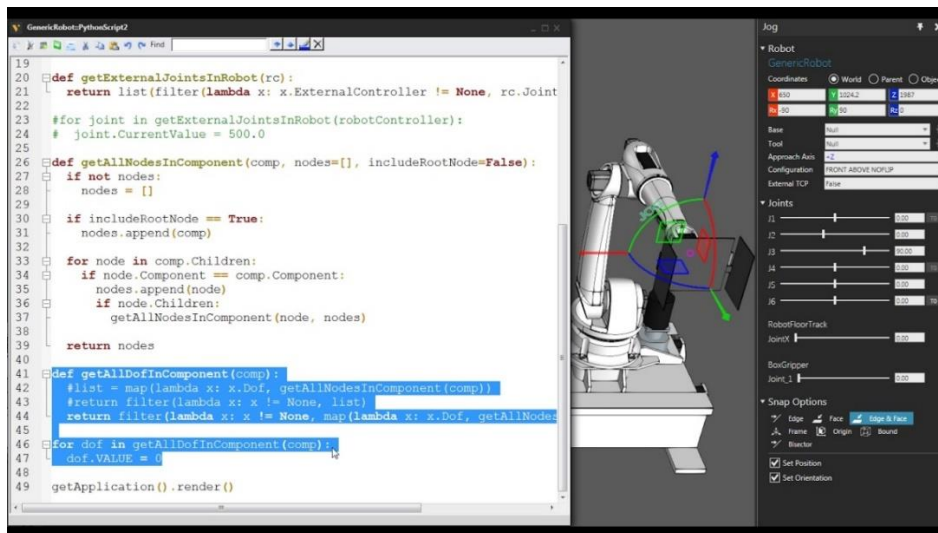


Figure I.12 Coding Example.

I.7 Conclusion:

In this chapter we did present some characteristics and types of robots depending on how they function on real life and how they sense their environment using sensors and actuator to do some duties such as industrial field or medical surgeries or helping human kind in their daily life. And we did show the advantage and disadvantage of robots and including humans in the situation to know how can robots effect our lives, some robots classified like fixed robot and mobile robots according on how they function.

In the next chapter we will be talking about Localization and mapping methods.

Chapter 2

Localization and Mapping on Robots

II. Localization and Mapping on Robots:

Chapter Guideline:

- You will learn:
 - ❖ The reason why robots used for being important to discover environment and knows its characteristics and the roll of using mapping to achieve that.
 - ❖ A title on history of mobile robots.
 - ❖ How we implement localization and mapping on robots.
- Tools needed:
 - Know the classification of the problem to make localization and mapping.
 - The relation with next chapter.

II.1 Introduction:

According to complexity and quickly becoming mathematically intractable. To being with, we have a fundamental issue with power supply, the robot needs to achieve some hard tasks autonomously during long period of time operation. This robot has to be aware with his environment all the time so it can move around without problems. Engineers can build a workspace to make robots explore it and knows its characteristics. It's a common to face unknown space or region to explore and give great results. And to have correct results Sensors be crucial for mobile robots precisely for that reason. The amount of data that the sensors can provides, it will be turned into results using deferent algorithms.

The relation between environment and the robot will be how it can recognize it characteristics so it can transform it into a map(data). So, before the robot can gives a map it has to calculate and perform some equations and algorithms to come up with some numbers, and convert it from numbers into 2D map or 3D according the how its system works. And it can give us also localization of the robot from this data.

One solution to this particular one is to estimate the map and localization of robot's positions. This called **simultaneous localization and mapping**, and it's currently known by the science community as SLAM acronym. This chapter will talk about how we can build a map assuming that localization has been solved somehow. SLAM is more complex then localization it's means that dealing with mapping and localization separately are easier.

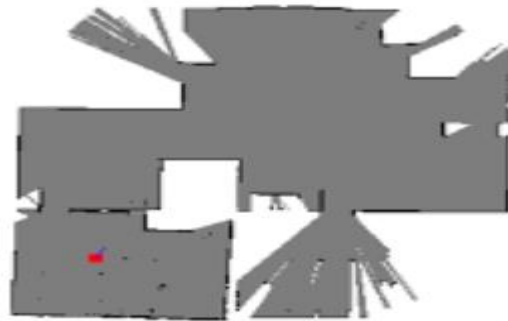


Figure II.1 This represent a SLAM results (mapping and localization).

II.2 How SLAM started:

SLAM was first shown in International Symposium on Robotics Research in 1995. after IEEE did define it in Robotics and Automation Conference in 1986. studies using statistical theory and navigation [7]. this was a time to show probabilistic methode were only just beginners to be introduced into robotics and AI am group of researchers was aiming to apply estimations theoretic method to solve mapping and localization problems. This include Peter Cheeseman, Jim Crowley, and Hugh Durrant-Whyte. Raja Chatila, Oliver Faugeras, Randal Smith and others also made useful contributions to the conversation S [8].

SLAM is a type of temporal model in which the goal is to infer a sequence of states from a noisy set of measurements [9].it can be to the calculation known for the system as a map that was given the previous states of measurements. states can be a lot of things for example Rosales and

Sclaroff (1999) used states as a 3D position of a bounding box around pedestrians for tracking their movements.

In 1998, at the Europe conference on Computer Vision Davison and others did presented a method by using camera without sensors this did led to the SLAM technology to be developed into vision-based SLAM that uses camera as a three-dimensional position detector.

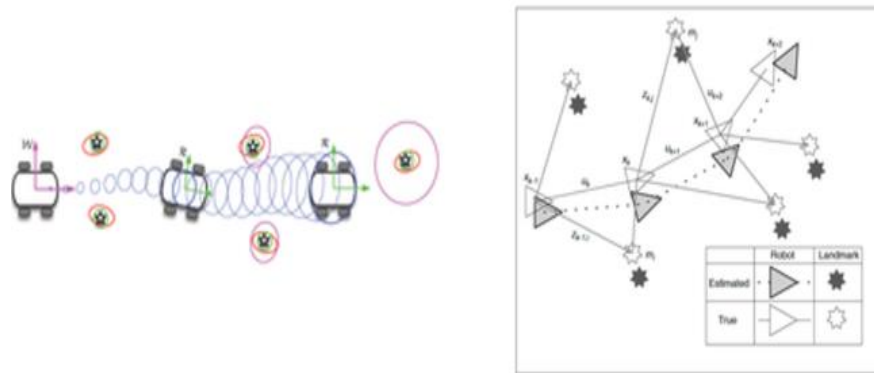


Figure II.2 The Birth Of SLAM [10].

II.3 The Futures of SLAM (Simultaneous Localization And Mapping):

There is always different method available to deal with robot localization and mapping it's only based on probability and theory and statistics. There are many of ways to get quit great efficiency, to define our environment characteristics, an available sensors and actuators are more likely needed before we put SLAM method in the picture. The more our robot have sensors the more details can get from its surrounding.

II.3.1 What is and Why we use SLAM?

Visual **SLAM** algorithms are able to simultaneously build 3D maps of the world while tracking the location and orientation of the camera (hand-held or head-mounted for AR or mounted on a robot). **SLAM** algorithms are complementary to ConvNets and Deep Learning: **SLAM** focuses on geometric problems and Deep Learning is the master of perception (recognition)

problems. If you want a robot to go towards your refrigerator without hitting a wall, use **SLAM**. If you want the robot to identify the items inside your fridge, use ConvNets [13].

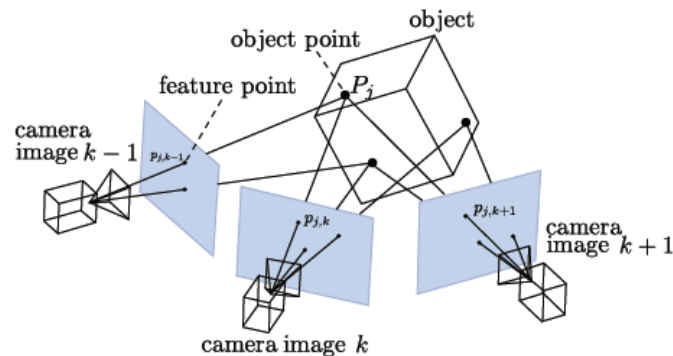


Figure II.3 represent SLAM technic to detect object using camera.

SLAM is a real-time version of Structure from Motion (SfM). Visual **SLAM** or vision-based **SLAM** is a camera-only variant of SLAM which forgoes expensive laser sensors and inertial measurement units (IMUs). Monocular SLAM uses a single camera while non-monocular **SLAM** typically uses a pre-calibrated fixed-baseline stereo camera rig. **SLAM** is prime example of a what is called a "Geometric Method" in Computer Vision. In fact, CMU's Robotics Institute splits the graduate level computer vision curriculum into a Learning-based Methods in Vision course and a separate Geometry-Based Methods in Vision course [10].

○ Localization

Many people don't know their own location or distance so they can be lost in roads the robot will calculate all the path and direction they did pass into and study all the short movements relative with the surrounding so the SLAM tech will solve this problem and detect everything and convert it into a map.

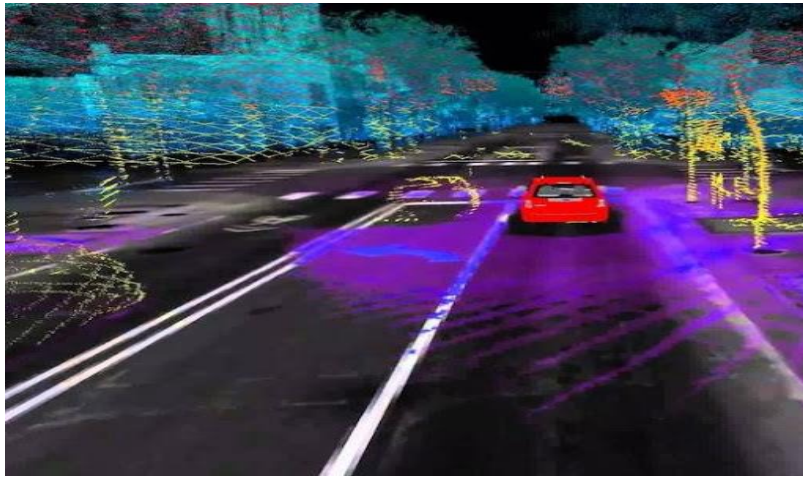


Figure II.4SLAM Application (Localization).

○Mapping

Vision-based SLAM will use camera to collect point (data) from all viewpoints through detection and matching. once all the points are assembled by the triangulation techniques a map will be generated by the program.

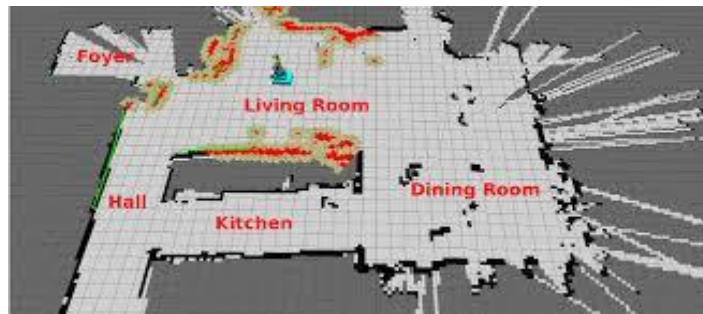


Figure II.53D SLAM Map.

II.3.2 SLAM Applications?

The most common real application of **SLAM** is self-driving cars it's from the most important application in human life that will avoid traffic and prevent work accident from happening. For many years to come it will make sense that the **SLAM** depends on how we develop many algorithms based only on webcam to be that efficiency. As a research topic, Visual **SLAM** is much friendlier to thousands of early-stage PhD students who'll first need years of in-lab experience with **SLAM** before even starting to think about expensive robotic platforms such as self-driving cars.

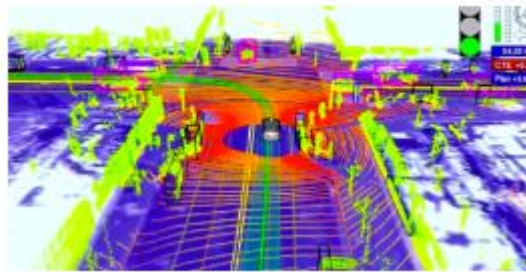


Figure II.6 Google's Self-Driving Car's Perception System. From IEEE Spectrum's.

Also SLAM technology is used into field robots. For example, rovers and landers for exploring Mars use visual **SLAM** systems to navigate autonomously. And agriculture field, as well as drones to explore valleys and forest from the top of sky's all those techs use mapping and different SLAM algorithms [11].

One of the great opportunities for SLAM systems is to replace GPS tracking to navigate in certain applications, because GPS system are not used indoors applications or in mega cities where the building is too high so that the satellite can reach. Visual SLAM systems solve each of these problems as they're not dependent on satellite information and its great measure tool with the physical world around.

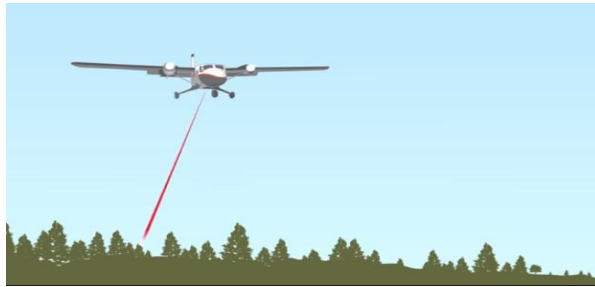


Figure II.7 Plains Using SLAM [12].

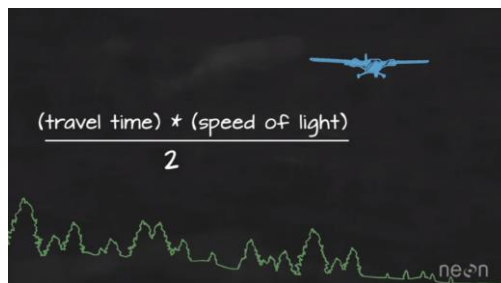


Figure II.8 Plains Result Using SLAM Algorithms [12].

In **figure 2.3** and **figure 2.3** we can see that the SLAM technology is used in plain to discover forest and to detect object from the sky from measuring latitude of plain and calculating distance from the ground and tress we get 3D map by applying difficult and complex algorithms [12] (SLAM algorithms).



Figure II.9 3D Map Using SLAM (Mapping and Localization Technic) [12].

Visual SLAM is just one of many innovative technologies under the umbrella of embedded vision. Many studies are being developed and new technics based on deep learning and machine learning so that can applicate SLAM technology in different fields. Like educations purposes.

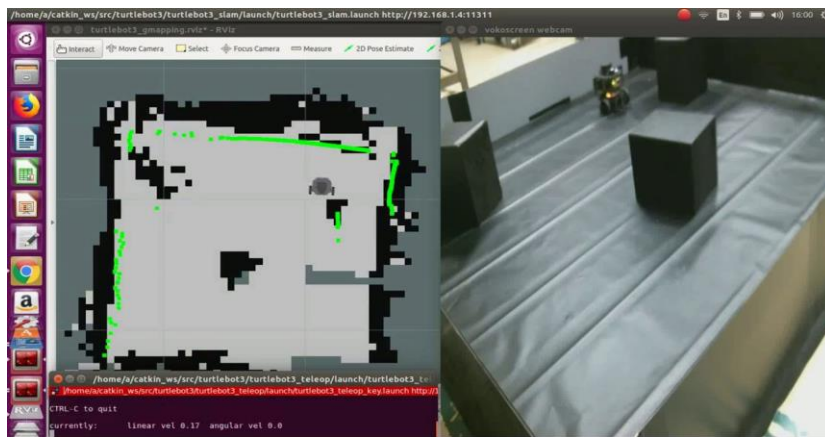


Figure II.10 Educations Environment Using Robot And Visual SLAM Technologie.

II.3.3 How SLAM Works?

Most visual SLAM systems begin by tracking set points through set of camera or deferent smart sensors to get their 3D or 2D positions. while simultaneously using this information to approximate camera pose. The goals of those systems are to map the environment based on their location for the purposes of navigation [12].

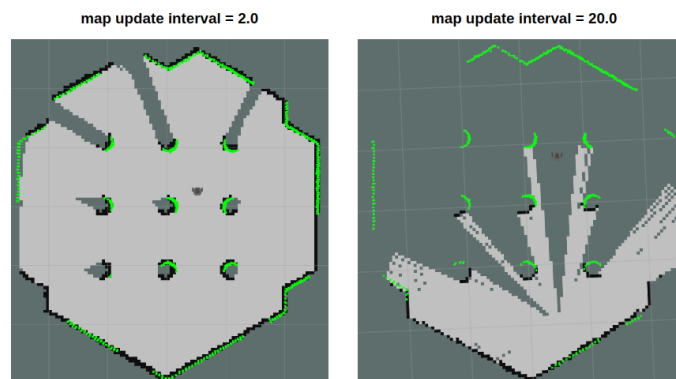


Figure II.11 2D Map. SLAM Result Using Virtual Simulation In Linux.

This is possible with a single 3D vision camera, unlike other forms of SLAM technology, as long as there more data available (points) through each frame, both the orientation of the sensor and the structure of the environment can be easy to understand by using this technology.

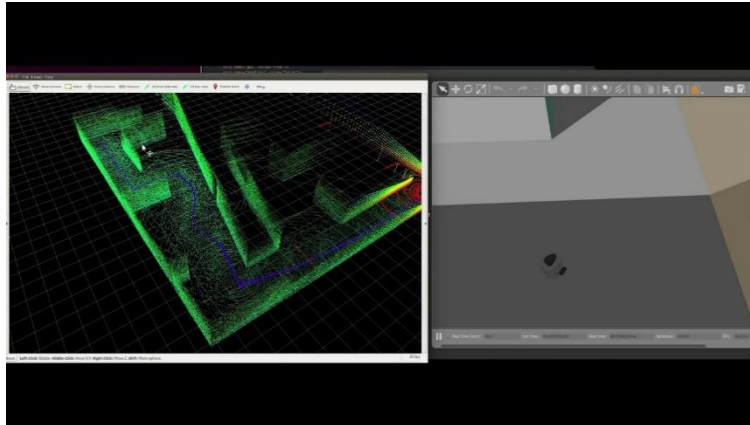


Figure II.12 3D Map. SLAM Result Using Virtual Simulation in Linux.

All systems who works with SLAM algorithms are aiming to minimize reprojections error, Visual SLAM algorithms all working in real time, so often location data and mapping data be processed in the moment, but simultaneously, to facilities faster processing before last results.

II.4 How We Implement SLAM System in Robots:

SLAM can be applicated into robots with simple ways. Like adding sensors or camera into the robot and a microcontroller to collect data and analyze it and convert it into useful information. robots needed to move around so that it can give us a map and his location or our car location for unknowns environment, it's not easy to localize itself because we need a map so the robot can estimate localization.

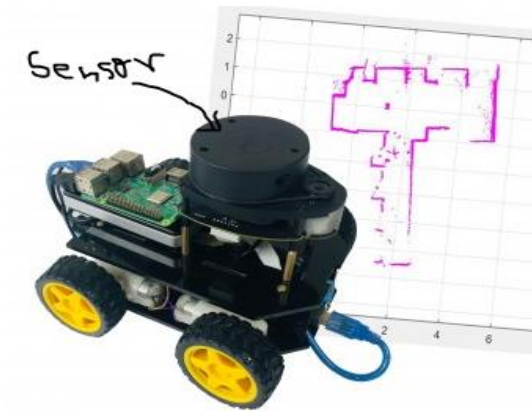


Figure II.13 Robot Car with LIDAR Sensor.

Clearly, lightweight SLAM algorithms are needed in intelligent robotic systems, mobile robots are limited because of the size and capabilities and power budget so it's simply equipped with microcontroller which has lower capability than a high CPU performance



Figure II.14 Raspberry pi 3 (Microcontroller or CPU Unite).

Robotic vacuum cleaner it's been used in indoor housekeeping applications, most of this vacuum just wander around the environment and clean up the floor along its path. After integrating SALM into the robotic system we can make our robots intelligent



Figure II.15 Smart Vacuum Cleaner.



Figure II.16 Old Vacuum Cleaner.

Also, Self-driving cars was developed to be used in autonomies driving using complex SLAM algorithms



Figure II.17 Self Driving Car With LIDAR And Camera.

We conclude how we implement SLAM in robot systems we need microcontroller and sensor and algorithms that can calculate and connect every point together to get map and localization.



Figure II.18 Control Unity.



Figure II.19 RPLiDAR A1 Sensor.

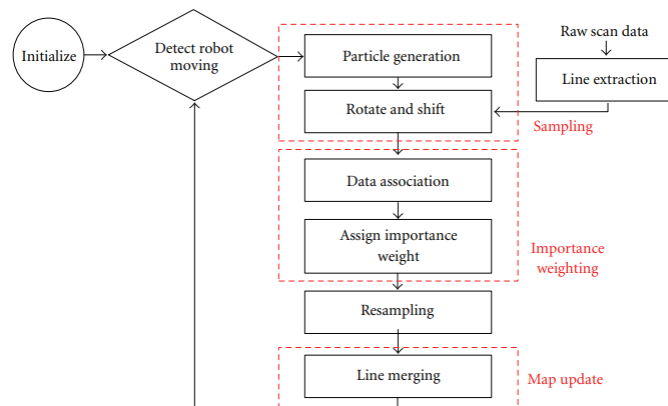


Figure II.20 Overall Flowchart Of A Particle Filter-Based Algorithm.

Usually an RBPF-based SLAM algorithm can be implemented by using the Sampling Importance Resampling (SIR) filter, which is one of the most commonly used particle filtering algorithms (hindawi, 2011)[13,14], and a map update technique. The entire procedure can be summarized in the following steps,

- (1) **Sampling**: New particles are generated from the previous particle using a motion model.
- (2) **Importance Weighting**: Each new particle is assigned an importance weight to determine the accuracy of the particle according to how well the current observation matches the map it has already built.

(3) ***Resamp***

ling: Particles with low weights are likely to be replaced by the ones with high weights.

(4) ***Map Update***: The most current map observed by the laser range finder is updated to each remaining particle after the resampling step according to its individual pose, so that each particle has a most updated map of the environment.

Not all SLAM algorithms fit any kind of observation (sensor data) and produce any map type. The following table summarizes what algorithms (of those implemented in Mobile Robot Programming Toolkit (MRPT)) fit what situation.

↓ Observations \ Maps →	Occupancy grid (Class COccupancy GridMap2D)	Point maps (Class CPointsMap)	Landmark map (class depends on SLAM method)	Graph of pose constraints
2D laser scanner (CObservation 2DRangeScan)	<ul style="list-style-type: none"> rbpf-slam icp-slam 	<ul style="list-style-type: none"> rbpf-slam icp-slam 	X	X
1D sonar range readings (CObservation Range)	<ul style="list-style-type: none"> rbpf-slam 	X	X	X
Features / Landmarks (SLAM method-dependent)	X	X	<ul style="list-style-type: none"> kf-slam srba Bundle adjustment 	X
Range-only sensors (CObservation BeaconRanges)	X	X	<ul style="list-style-type: none"> ro-slam 	X
Relative poses	X	X	X	<ul style="list-style-type: none"> graph- slam srba

Table II.1 Some of SLAM Map Types Vs Algorithms

Some of the List of MRPT apps

- Application: carmen2rawlog
- Application: carmen2simplemap
- Application: DifOdometry-Camera
- Application: DifOdometry-Datasets
- Application: features-matching

- Application: gps2rawlog
- Application: graph-slam
- Application: graphslam-engine
- Application: grid-matching
- Application: holonomic-navigator-demo
- Application: icp-slam
- Application: icp-slam-live
- Application: image2gridmap
- Application: kf-slam
- Application: kinect-3d-slam
- Application: kinect-3d-view
- Application: kinect-stereo-calib
- Application: map-partition
- Application: navlog-viewer
- Application: observations2map
- Application: pf-localization
- Application: PTG-configurator
- Application: rawlog-edit
- Application: rawlog-grabber
- Application: rbpf-slam
- Application: ReactiveNav3D-Demo
- Application: ReactiveNavigationDemo
- Application: ro-localization
- Application: robotic-arm-kinematics

- Application: SceneViewer3D
- Application: simul-landmarks
- Application: srba-slam (Relative Bundle Adjustment and Relative Graph-SLAM)
- Application: track-video-features
- Application: velodyne-view
- Application: 2d-slam-demo
- Application: camera-calib (Camera intrinsic calibration)
- Application: RawLogViewer
- Application: GridmapNavSimul (mrpt, 2020) [15]

SLAM algorithms:

- SLAM (Simultaneous Localization and Mapping) for beginners: the basics
- Bayesian range-only SLAM (RO-SLAM) with SOGs
- Derivation and Implementation of a Full 6D EKF-based Solution to Range-Bearing SLAM
- HMT-SLAM
- RBPF-SLAM algorithms (C++ library mrpt-slam)
- Sparser Relative Bundle Adjustment (SRBA)

Now we are going through some of the details about one of the SLAM algorithms which is SLAM (Simultaneous Localization and Mapping) for beginners: the basics

II.4.1 Content And Design Of SLAM Problems:

SLAM is a way that a robot can build a map and give localization information. In SLAM both the path and platform and location of all indicators are estimated on-line without the need for any previous knowledge of location.

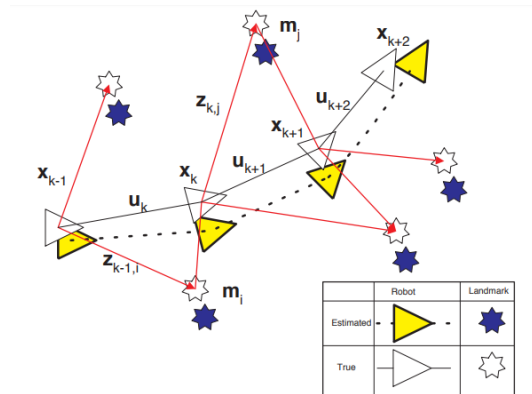


Figure II.21 The essential SLAM problem.

A simultaneous estimate of both robot and landmark locations is required. The true locations are never known or measured directly. Observations are made between true robot and landmark locations. See text for details [8].

figure II.21 At a time instant k , the following quantities are defined:

- **\mathbf{x}_k** : The state vector describing the location and orientation of the vehicle.
 - **\mathbf{u}_k** : The control vector, applied at time $k-1$ to drive the vehicle to a state \mathbf{x}_k at time k .
 - **\mathbf{m}_i** : A vector describing the location of the i the landmark whose true location is assumed time invariant.
 - **$\mathbf{z}_{k,j}$** : An observation taken from the vehicle of the location of the i the landmark at time k .
- When there are multiple landmark observations at any one time or when the specific landmark is not relevant to the discussion, the observation will be written simply as \mathbf{z}_k [8].

II.4.2 Probabilistic SLAM:

In probabilistic form, the Simultaneous Localization and Map making (SLAM) problem requires that the probability distribution [8].

$$\text{probability distribution} \longrightarrow [\mathbf{P} (\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_0: k, \mathbf{U}_0: k, \mathbf{x}_0)] \quad (1)$$

be calculated for all times k , this probability distribution explains the parts posterior density of the landmark locations and vehicle status (at time K) given the recorded observations and control inputs up to and including time k together with the initial state of the vehicle.

Starting with an estimate for the distribution $P(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1})$ at time $k-1$, the joint posterior, following a control \mathbf{u}_k and observation \mathbf{z}_k , is calculated using Bayes Theorem. This calculation needs that transition model and an observation model are define explaining the effect of control input and observation respectively.

The observation model describes the probability of making an observation \mathbf{z}_k when the location and land mark are knowing it will be described in from

$$P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}). \quad (2)$$

The **Motion model** for the robot can be explained in terms of a probability distribution on state transition in the form

$$P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \quad (3)$$

The SLAM algorithms are now added into a standard two-step recursive (sequential) prediction (time-update) correction (measurement-update)

$$\begin{aligned} & P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) \\ &= \int P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \\ & \times P(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) d\mathbf{x}_{k-1} \end{aligned}$$

Figure II.22 Time-update.

$$\begin{aligned} & P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \\ &= \frac{P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}) P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \end{aligned}$$

Figure II.23 Measurement Update.

However, the SLAM problem has more forms then this equation.

Referring again to (**figure II.21**) it can see much error between predicting and the true localization between landmarks and is in fact due to a single source errors in common of where the robot will be in the position (landmarks) observations are made. In turn, this implies that the errors in landmark location estimates are highly correlated. Practically, it's means that the location between every two landmarks m_i - m_j , will be known even if the landmark is risky to know [8].

The important thing in SLAM was realizing that the correlations with landmark be predicted increase 4 monotonically as more and more observations are made. Its means that the location is always improved to be known and never diverges, regardless of robot motion.

This convergence occurs because the robot observation can be considered as nearly independent measurements of the relative location between landmarks. Referring again to (**figure II.21**), consider the robot at location x_k observing the two landmarks m_i and m_j the relative location of landmarks is separated from coordinated frame of the vehicle and great observation from the fixed location will be independent measurements of the relative relationship between landmarks. Soon the robot when it changes a location to another will observe landmark m_j this allows the prediction of location of the robot and landmark to be updated relative to the last point (landmark) x_k . This occurs because the two landmarks are highly correlated (their relative location is well known). Also, in (**figure II.21**) at location x_{k+1} the robot will observe two new landmarks relative to the m_j immediately the new land marks will be updated to the map. Later update to these landmarks will also update landmark m_j and through this landmark m_i and so on. All of this end up with forming a network linked by relative location or correlations whose precision or value increases whenever an observation is made.

This process can be seen in (**figure II.24**) as a network of springs connecting all landmarks together, when the robot moves through this environment and takes observation if the new points (landmarks) the springs become more strong.in the limit, a grid map of landmarks orprecisely relative map of the surrounding is obtained. As the map is built, the location accuracy of the robot will be known it depend on how much the quality of the map is relative to measurement sensor. In

the theoretical limit, robot relative location accuracy becomes equal to the localization accuracy achievable with a given map.

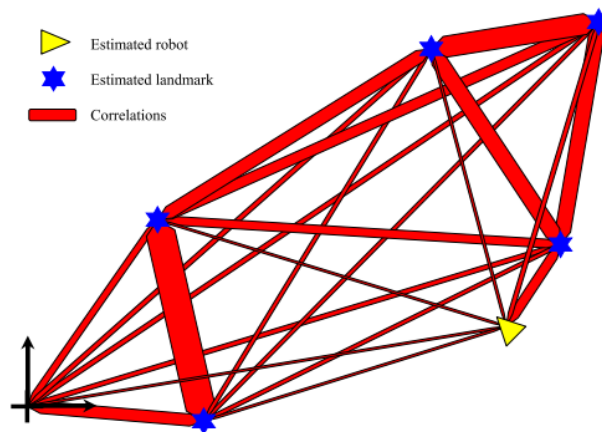


Figure II.24 Spring network analogy [8].

II.4.3 Solutions to The SLAM Problem:

Solutions to the probabilistic SLAM problem involve finding an appropriate representation for the observation model Equation 2 and motion model Equation 3 which allows efficient and consistent computation of the prior and posterior distributions in Equations 4 and 5. By far the most common representation is in the form of a state-space model with additive Gaussian noise, leading to the use of the Extended Kalman Filter (EKF) to solve the SLAM problem as described.

In robotics, EKF SLAM is a class of algorithms which deals with the extended Kalman filter (EKF) for simultaneous localization and mapping (SLAM). Normally, EKF SLAM algorithms are feature based, and use the maximum likelihood algorithm for data association. In the 1990s and 2000s, EKF SLAM had been the de facto method for SLAM, until the introduction of FastSLAM [16].

In order to achieve exploration the EKF machinery is full with extra step of landmarks initialization, where every new land marks will be added to the map Landmark initialization is performed by inverting the observation function and using it and its Jacobians to compute, from the sensor pose and the measurements, the observed landmark state and its necessary co- and cross-variances with the rest of the map. These relations are then appended to the state vector and the covariances matrix.

The following table explain the similarity and differences between EKF and EKFSLAM:

Event	SLAM	EKF
Robot moves	Robot motion	EKF prediction
Sensor detects new landmark	Landmark initialization	State augmentation
Sensor observes known landmark	Map correction	EKF correction
Mapped landmark is corrupted	Landmark deletion	State reduction

Table II.2 EKF Operations for Achieving SLAM.

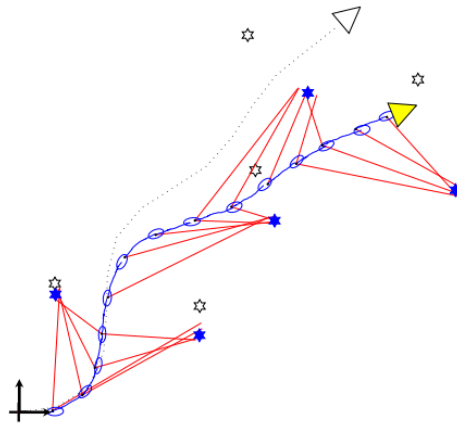


Figure II.25 Realization of Robot Path in The Fast -SLAM Algorithm.

The ellipsoids show the proposal distribution for each update stage, from which a robot did stop is a data gathering and assuming the postions is perfect the observed landmarks are updated. The quality of the map it depends on the accuracy of the trajectory. Many such trajectories provide a probabilistic model of robot location [8].

II.5 The Relation with Next Chapter:

Next chapter will be talking how we can import different SLAM algorithms under a ROS (Robot Operation System) that will operate in Linux system.

II.6 Conclusion

In this chapter we explained in general about SLAM technology (Simultaneous localization and Mapping) .to make a map from scratch and let the robot to know its localization into that environment and the SLAM work in real time and predict its own path and also connect everything together to get a 2D or 3D map. And various application field which is self-driving cars plains, drones and educational robot cars like Turtlebot. Specially how SLAM was developed in the first start by researchers and SLAM problems in first depart and some solution to that problem and it end up with fast-SLAM algorithms. And now everyone is seeking to develop faster algorithms to get great result on difficult situation like AI and deep learning.

Chapter 3

ROS

(Robotic Operation System)

III. Robotic Operation System:

III.1 Introduction

ROS (Robots Operation System) is an open-source, system for robots it provides every tool you need including hardware abstraction low-level device control implementation of commonly-used functionality, message-passing between processes and package management it also provides some tools and libraries for building writing and running some code across multiple computers. Although the research community is quite active in developing applications with ROS and extending its features, the amount of references does not translate the huge amount of work being done.

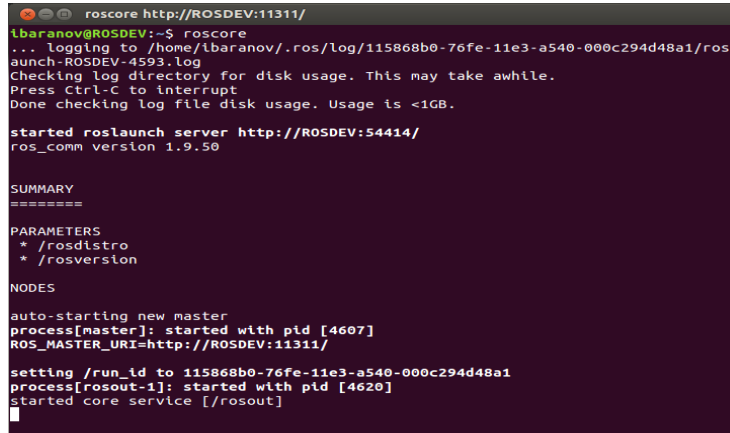
Running sets of ROS-based processes are shown in a graph architecture where processing takes place in nodes that may receive, post and veracity of sensor data, control, state, planning. ROS itself is not real-time OS but it is possible to write some real time algorithms in ROS system, the lack of support for real-time systems has been addressed in the creation of ROS 2.0.

Software in the ROS system can be separated into three groups:

- language-and platform-independent tools used for building and distributing ROS-based software.

- ROS client library implementations such as roscpp, rospy, and roslisp
- packages containing application-related code which uses one or more ROS client libraries.

III.2 ROS Goals:



```

roscore http://ROSDEV:11311/
tbaranov@ROSDEV:~$ roscore
... Logging to /home/tbaranov/.ros/log/115868b0-76fe-11e3-a540-000c294d48a1/rosl
aunch-ROSDEV-4593.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ROSDEV:54414/
ros_comm version 1.9.50

SUMMARY
=====
PARAMETERS
* /rostdistro
* /rosverston

NODES

auto-starting new master
process[rosmaster]: started with pid [4607]
ROS_MASTER_URI=http://ROSDEV:11311/

setting /run_id to 115868b0-76fe-11e3-a540-000c294d48a1
process[rosout-1]: started with pid [4620]
started core service [/rosout]

```

Figure III.1 Roscore Launch in Terminal.

The first goal of ROS is to support code preparation in robotics research and development. ROS is a distributed framework of processes (aka *Nodes*) that enables executables to be individually designed and loosely coupled at runtime. This process can be gathered into packages and be simply shared. This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools [17].

There are several other goals of the ROS framework:

ROS-agnostic libraries: the preferred development model is to write ROS-agnostic libraries with clean functional interfaces.

Language independence: ROS are easy to be putted in any modern programming language it's already prograded in **python** and **C++** and **Lisp** and we have experimental libraries in Java and Lua.

Easy testing: ROS has a builtin unit/integration test framework called rostd that makes it easy to bring up and tear down test fixtures.

Scaling: ROS is appropriate for large runtime systems and for large development processes.

III.3 Operations System:

ROS only work on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, the community has contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms. While in windows is available but it's not fully explored.

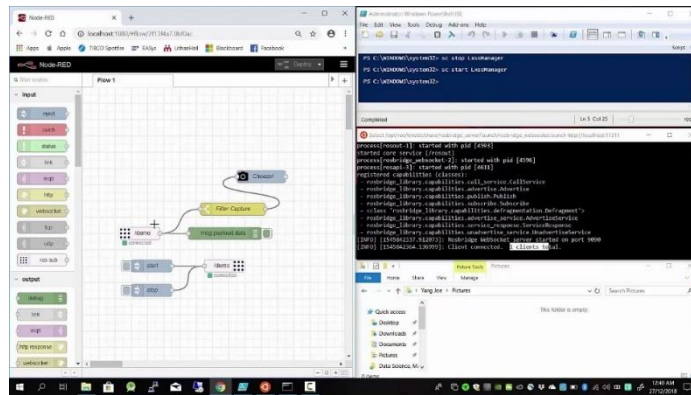


Figure III.2 ROS in Windows 10.

III.4 ROS Context:

Master: often referred to as `roscore` to launch a system core and enable all commands and tools and you can connect every **node** in the system [18].

Graph Resource: a name of resources (node, topic, service, or parameter) The naming scheme is hierarchical and has many aspects in common to UNIX file system paths.

Node: A *node* is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics[19].

Topic: A unidirectional, asynchronous, strongly typed, it's a style communication in the publish-subscribe mechanism identified by the **Graph Resource**.

Messages: tool can print out message. Also, we can say it's a description language for describing the data values that ROS node are publishing. This will make it easy to ROS tools to

generate source code for the message type in more than one language.[20] Messages can communicate with each other by publishing **messages** to topics. A message is a simple data structure, comprising typed fields.

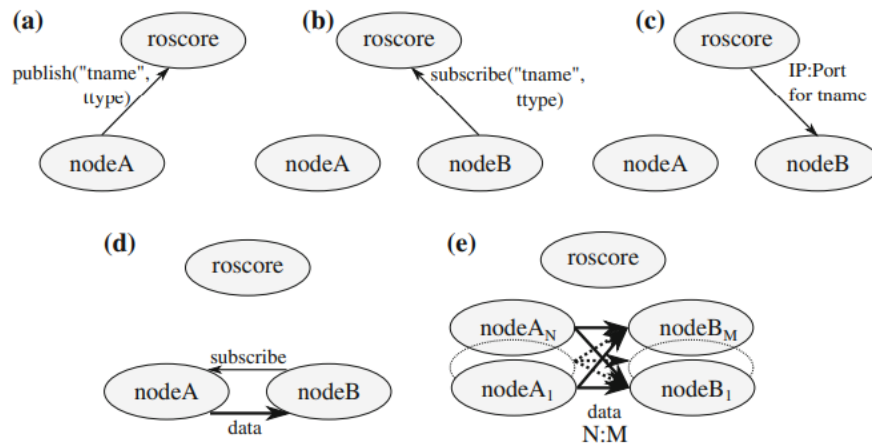


Figure III.3ROS Topic Mechanism.

Service:its defined by bunch of messages one for requesting and the other for replaying.

Services are defined using **srv** files, which are compiled into source code by a ROS client library [21].

Roslaunch: A command line tool and XML format to coherently start a set of nodes including remapping of names and setting of parameters [22]

III.5 ROS Versions(distribution):

A ROS distribution is a set of ROS packages. They are all connected to Linux. The purpose of the ROS distributions is to let developers work against a relatively stable codebase until they are ready to roll everything forward. Sometimes there is some bugs in the ROS version so they try to fix it every time by updating all packages but for "higher" level packages, the rules are less strict, and so it falls to the maintainers of a given package to avoid breaking changes [23].

Distro	Release date	Poster	Turtle, turtle in tutorial	EOL date
ROS Noetic Ninjemys	May, 2020 (planned, see Upcoming Releases)	TBA	TBA	May, 2025 (planned)
ROS Melodic Morenia (Recommended)	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kama	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			—
ROS Electric Emys	August 30, 2011			—
ROS Diamondback	March 2, 2011			—
ROS C Turtle	August 2, 2010			—
ROS Box Turtle	March 2, 2010			—

Figure III.4 ROS List of Distribution [24].

Signs:

- light yellow: future release.
- green: supported release.
- grey: unsupported release (End of Life).

III.6 Ubuntu Install of ROS kinetic

III.6.1 Installation:

ROS Kinetic **ONLY** supports Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) and Jessie (Debian 8) for debian packages. ROS kinetic only support some versions like ubuntu 15.10 Xenial Ubuntu 16.04. This installation is provided by the official community of ROS developers [25]. <http://wiki.ros.org/kinetic/Installation/Ubuntu>

III.6.2 Configure Your Ubuntu Repositories

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can [follow the Ubuntu guide](#) for instructions on doing this.

III.6.3 Setup Your Sources List

Setup your computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
```

III.6.4 Set Up Your Keys

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

If you experience issues connecting to the keyserver, you can try substituting hkp://pgp.mit.edu:80 or hkp://keyserver.ubuntu.com:80 in the previous command. Alternatively, you can use curl instead of the apt-key command, which can be helpful if you are behind a proxy server:

```
curlSL'http://keyserver.ubuntu.com/pks/lookup?op=get&search=0xC1CF6E31E6BADE8
868B172B4F42ED6FBAB17C654' | sudo apt-key add -
```

III.6.5 Installation

First, make sure your Debian package index is up-to-date:

```
sudo apt-get update
```

There are many different libraries and tools in ROS. We provided four default configurations to get you started. You can also install ROS packages individually.

In case of problems with the next step, you can use following repositories instead of the ones mentioned above [ros-shadow-fixed](#)

•**Desktop-Full Install: (Recommended)** :

ROS, [rqt](#), [rviz](#), robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception

```
sudo apt-get install ros-kinetic-desktop-full
```

•**Desktop Install:** ROS, [rqt](#), [rviz](#), and robot-generic libraries

```
sudo apt-get install ros-kinetic-desktop
```

•**ROS-Base: (Bare Bones)** ROS package, build, and communication libraries. No GUI tools.

```
sudo apt-get install ros-kinetic-ros-base
```

•**Individual Package:**

You can also install a specific ROS package (replace underscores with dashes of the package name):

```
sudo apt-get install ros-kinetic-PACKAGE
```

e.g.

```
sudo apt-get install ros-kinetic-slam-gmapping
```

To find available packages, use:

```
apt-cache search ros-kinetic
```

III.6.6 Environment Setup

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

If you have more than one ROS distribution installed, `~/.bashrc` must only source the `setup.bash` for the version you are currently using.

If you just want to change the environment of your current shell, instead of the above you can type:

```
source /opt/ros/kinetic/setup.bash
```

If you use `zsh` instead of `bash` you need to run the following commands to set up your shell:

```
echo "source /opt/ros/kinetic/setup.zsh" >> ~/.zshrc  
source ~/.zshrc
```

III.6.7 Dependencies for Building Packages

Up to now you have installed what you need to run the core ROS packages. To create and manage your own ROS workspaces, there are various tools and requirements that are distributed separately. For example, [rosinstall](#) is a frequently used command-line tool that enables you to easily download many source trees for ROS packages with one command.

To install this tool and other dependencies for building ROS packages, run:

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool  
build-essential
```


III.6.8 Initialize Rosdep

Before you can use many ROS tools, you will need to initialize rosdep. rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS. If you have not yet installed rosdep, do so as follows.

```
sudo apt install python-rosdep
```

With the following, you can initialize rosdep.

```
sudo rosdep init
```

```
rosdep update
```

III.6.9 Build Farm Status

The packages that you installed were built by the [ROS build farm](#). You can check the status of individual packages [here](#).

III.7 Start Your Project:

After the installation of the ROS kinetic and updating all packages you can work and deal with different projects and write your code even with Python OR C++ or Lisp.

III.8 Why Using ROS?

III.8.1 ROS in General:

The same reason when you build your project in the field and many application like drones and four wheels motors..., you can only use ROS to build your part in virtual world using some coding the fastest you figure out the meaning of how communications works between nodes of the program, you can set up new parts of an application very easily [26].

III.8.2 ROS Packages for Everything:

ROS this system has everything you need form driving cars automatically just you access this with a package and controlling robot with joystick that available as a package you can also import some project and modify them to suet your ideas.

III.8.3 ROS Has Great Simulation Tools:

In any project you need some tools to run your project in good conditions and those tools, such as Rviz and Gazebo. Gazebo you can simulate some robots in 3D vision. Or mapping the environment surrounding your robot.

III.8.4 You Can Control Multiple Robots With ROS:

This means you can access to different robot in the same time and each robot can communicate with each other and each robot has its own ROS system.

III.8.5 ROS Is Light:

The ROS system it can't take so much from your system drive and resources you can easily install it and start in few minutes. And you can even import the ROS into an embedded system so you can use in deferent way.

III.8.6 More Compatible with Your Ideas:

You can do more stuff in your ROS system but you don't have to create each part from your project you might focus on some development point, and integrate the rest from other sources.

III.9 Conclusion:

In this chapter we did explain about ROS (Robotic Operation System). How you upload the kinetic version into your Linux system and how you can use all tools provided by the ROS system like setting the environment source it to apply your requirement. And we showed all the version that you can install it into Linux. Most important part how the system communicate with you and how you can extract all information.

Chapter 4

Application and Discussion

IV. Application and Discussion.

IV.1 Introduction:

Maps can be so difficult things to understand and they help us in our systems to figure out our way into the exit or the target that we want to find, Humans have admit the importance and the value of maps to their lives, saying that maps can be found in the old cycles more than 5,000 years ago, now maps are so important that were using it in different fields from Google Maps into GPS to find our way, In other definition maps even became an important tools that went inside our cars systems we used daily from finding our way to home or work and event in robotics fields like autonomous cars that google companies did developed and equipped it with cameras and smart sensor to detect and efficient navigate surrounding unknown areas. This technology of mapping is more useful to be used in hard and difficult terrains to explore without spearing human soles.

Other ways GPS use maps to navigate and localize the position of the car by using satellites with triangulation method to calculate position and velocity by applying some algorithms based on Artificial Intelligent. Sometimes GPS don't work inside big buildings or mega police cities because the minimum number of satellites is 3 to localize the position so it can't connect to the GPS, Robot can access inside building so with the right sensors we can create 2D or 3D map for that facility without using GPS or something else [27].

To test this technology we have many ways, even in real application or by testing it in virtual world using some coding and simulations platforms that enable us to use some expensive sensors and multiple version robots' cars, drones, rovers, all that is shown down in this chapter.

IV.2 Component and Design of Robot:

The first thing about robots is the hardware that matters from the process unity into the power supply and for this project I did chose to work with four wheels robot can only move in flat surface, you can count on any robot can move according what you want to do, for example a robot who can climb stairs and pic up things like robotic arm so with that you can apply some high and

complex movement beside just moving around, in this chapter we only need a robot who can carry some sensor to create a simple map for the environment.

IV.2.1 Objectives:

- Installing and Learning about ROS environment.
- Applying some algorithms on Linux using the Terminal.
- build some robots in Linux.
- Built our own robot using some tools and test it.
- Built our real robot.
- Test our robot with sensors.

IV.2.2 Choosing the Component:

IV.2.2.1 Mechanical parts:

Motors, wheels, power supply, jumping wires, chassis.

○100 RPM Bo Motor – Straight:

The motors is shown down in (figure IV.1).

- Low density: lightweight, low inertia.
- Capability to absorb shock and vibration as a result of elastic compliance.
- Ability to operate with minimum or no lubrication, due to inherent lubricity.
- The relatively low coefficient of friction.
- Operating Voltage (VDC): 3~12.
- Shaft Length (mm): 8.5
- Shaft Diameter (mm): 5.5 (Double D-type)..
- No Load Current: 40-180mA.
- Rated Speed(After Reduction): 100 RPM.
- Rated Torque: 1 Kgcm.
- The motor is ideal for DIY enthusiast

○Dual Shaft Bo Motor with Big Wheel:

The Wheels is shown down in **(figure IV.2)**.

➤With upgraded tire tread for greater friction.

➤Small shaft with matching wheels gives an optimized design for your application or robot

➤Diameter: 65 mm.

➤Width: 28 mm.

➤Color: Yellow



Figure IV.1 Straight BO



Figure IV.2 Wheels.

○**Chassis:**

This this is the buddy of our robot the one who holds the motors and wheels in the follows figure:



Figure IV.3 Transparent chassis.

Tools-and-equipment:

That are the equipment your going to need them to assemble your robot together for example the multimeters for checking that all parts are connected all presented in **(figure IV.4)**.

- Soldering iron.
- Jumper wires.
- Multimeters.



Figure IV.4 Tools and equipment's.

○Process Unity:

In this project you will choose anything you want to analyze the Data coming from sensors. In this real building part, I did go with raspberry pi 3 B module in (**figure IV.5**) because we're going to use ROS (Robotic Operation System) to use some important tools from it.



Figure IV.5 Raspberry pi 3 B+ Module

➤ 1.4GHz 64-bit quad-core processor, dual-band wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and Power-over-Ethernet support (with separate PoE HAT)

➤ These modules allow a designer to leverage the Raspberry Pi hardware and software stack in their own custom systems and form factors.

➤ These modules have extra IO interfaces over and above what is available on the Raspberry Pi model A/B boards, opening up more options for the designer.

To know more about raspberry pi 3 b + visit its datasheet:

https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DAT_A_CM3plus_1p0.pdf.

○Drivers:

➤ 1298 motor driver and ultrasonic sensor hc-sr04(H-bridge) (**figure IV.6**).

It's a device that enable us to control motor direction and speed by sensing commands throw the command pins like it shown in (**figure IV.7**).

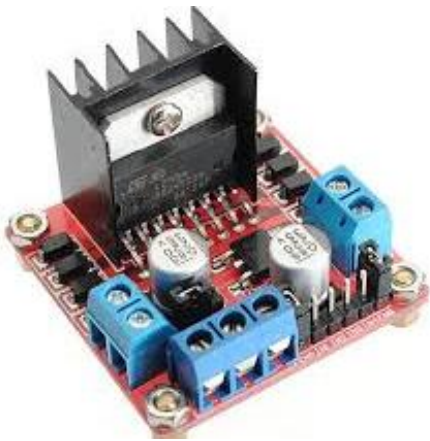


Figure IV.7 L298 Driver.

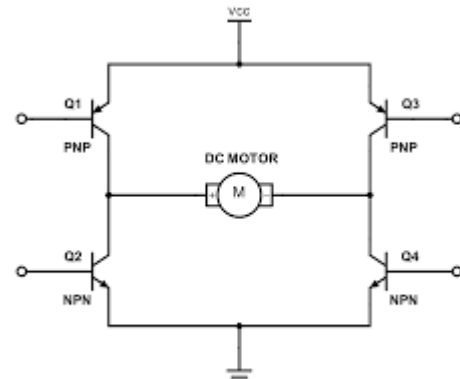


Figure IV.6 H-bridge Driver.

- **Communication:**
- we will access to the robot by using the WIFI on the raspberry pi 3 by activating the (SSH mode).

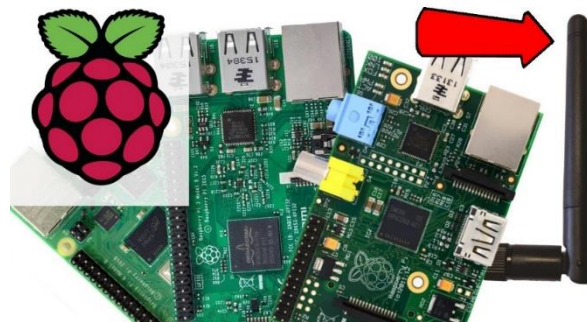


Figure IV.8 Raspberry pi 3 WIFI.

○ **Power supply:**

Using Two batteries of 3.7 v in series to have more power with 2 x 8800 mAh.



Figure IV.9 3.7 Battery 8800 mAh.

○**Sensors:**

In this project we did choose an expensive sensor (LIDAR) (**figure IV.11**).

➤RPLIDAR A1 is a low cost 360-degree 2D laser scanner (LIDAR) solution developed by SLAMTEC. The system can perform 360degree scan within 6meter range. The produced 2D point cloud data can be used in mapping, localization and object/environment modeling.

➤RPLIDAR A1's scanning frequency reached 5.5 hz when sampling 360 points each round. And it can be configured up to 10 hz maximum.

➤RPLIDAR A1 is basically a laser triangulation measurement system. It can work excellent in all kinds of indoor environment and outdoor environment without sunlight.

➤**System connection:** with USB adapter **Figure IV.11**.

➤Sensor power supply only needs from (5-10v).

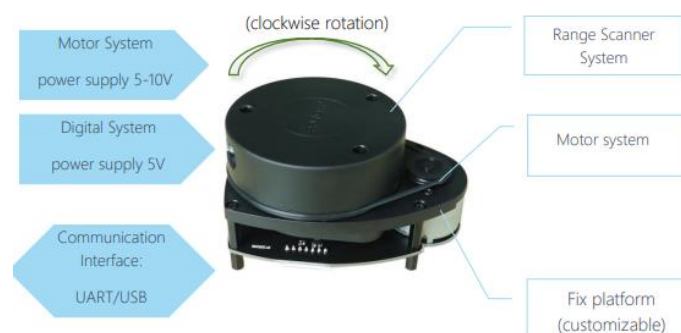


Figure IV.10 RPLIDAR System Composition.



Figure IV.11 RPLIDAR With USB Adapter.

○ **Why This Sensor:**

According to what we want to do in this project (build 2D map with our robot), the best way is with this sensor. It comes with different features specially in its own mechanism.

RPLIDAR is based on laser triangulation ranging principle and uses high-speed vision acquisition and processing hardware developed by **SLAMTEC**. The system measures distance data in more than 2000 times per second and with high resolution distance output.

RPLIDAR emits modulated infrared laser signal and the laser signal is then reflected by the object to be detected. The returning signal is sampled by vision acquisition system in RPLIDAR A1 and the DSP embedded in RPLIDAR A1 starts processing the sample data and outputs distance value and angle value between object and RPLIDAR A1 through communication interface.

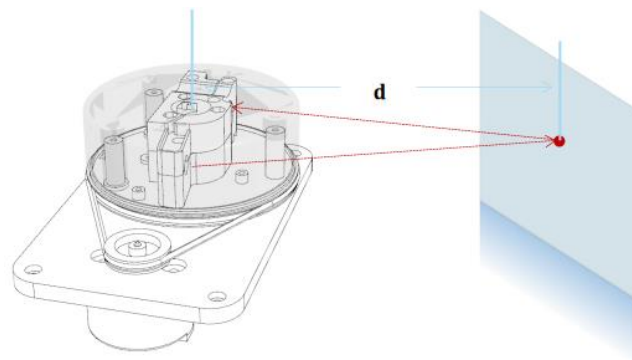


Figure IV.12 RPLIDAR A1 Working Schematic.

➤ in simple word the LIDAR gives us points each point represent a distance according into our Sensor like in (**figure IV.12**).

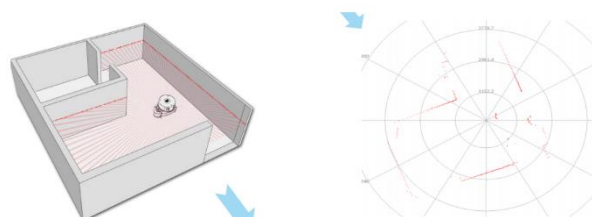


Figure IV.13 An Example About How RPLIDAR Work Inside Doors.

IV.3 Our Robot Electronic Circuit assembly:

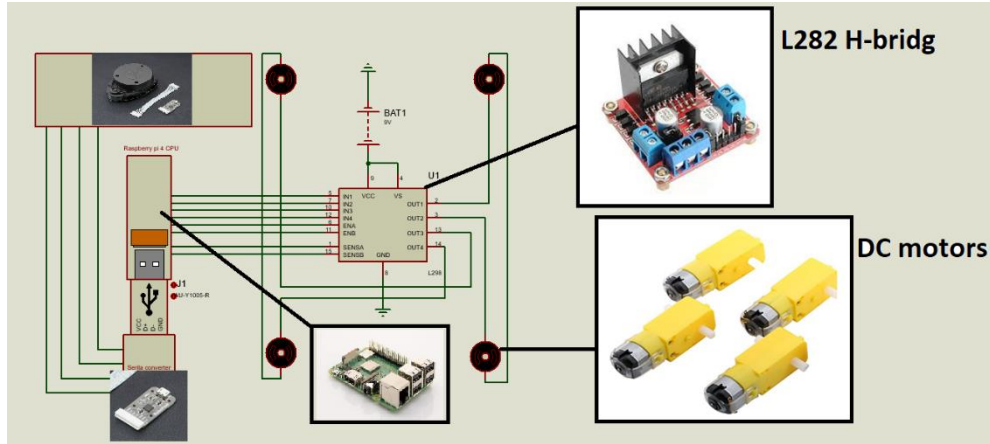


Figure IV.14 Robot Circuit.

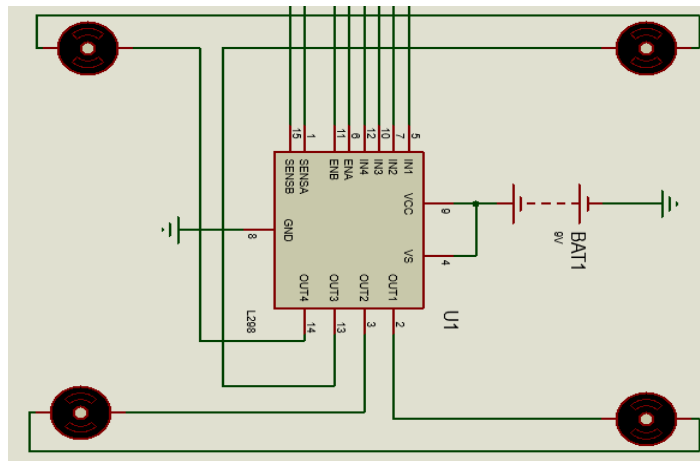


Figure IV.15 Robot Control Direction Circuit (H-bridg and Motors).

IV.4 Getting Started:

In this part were going to demonstrate the work into two divisions some work only will be in software and the other will be with the real robot.

IV.4.1 Installing Linux To Fix Our ROS Tools:

IV.4.1.1 Installing Linux:

First you need to prepare your computer with with Linux system Ubuntu 16.04 so you can install and build your project using some tools from ROS.



Figure IV.16Ubuntu 16.04 Linux System.

➤To install ubuntu 16.04 visit this site:

<https://ubuntu.com/tutorials/install-ubuntu-desktop-1604#1-overview>

IV.4.1.2 Installing and Configure ROS Environment:

To Install ROS, you need to run some commands lines in the Terminal we have deferent ROS Versions we did choose ROS kinetic distribution. To start installation, you need to follow the next steps “<http://wiki.ros.org/kinetic/Installation/Ubuntu>”.

➤Open the Terminal:

And type some configuration commands and it follow with some ROS installation command and update the system.

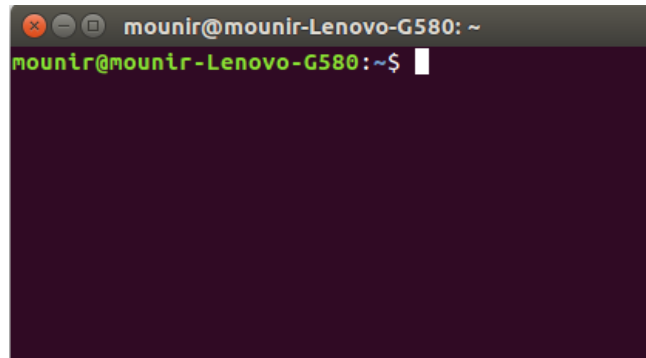


Figure IV.17 Terminal Window.

○ **First setting up the source list with this command**

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

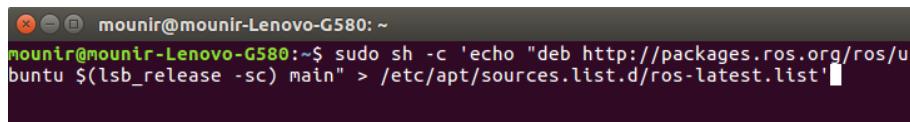


Figure IV.18 Setting the Source.

○ **setting keys**

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

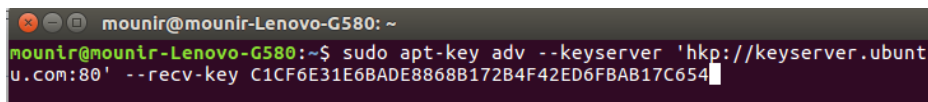


Figure IV.19 Preparing Keys.

○ **installation**

```
sudo apt-get update
```

you need the update to prepare for new installation. after completing those steps, you need to set your environment

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

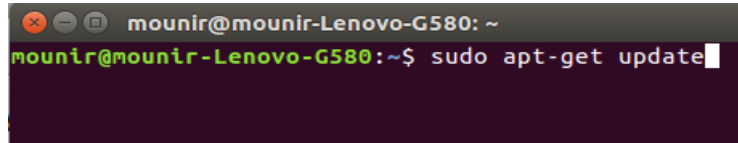
A terminal window with a dark background. The title bar shows 'mounir@mounir-Lenovo-G580: ~'. The prompt is 'mounir@mounir-Lenovo-G580:~\$' and the command 'sudo apt-get update' is being typed at the end of the line.

Figure IV.20 Updating.

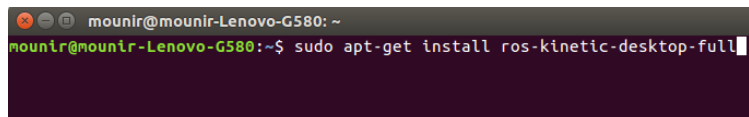
A terminal window with a dark background. The title bar shows 'mounir@mounir-Lenovo-G580: ~'. The prompt is 'mounir@mounir-Lenovo-G580:~\$' and the command 'sudo apt-get install ros-kinetic-desktop-full' is being typed at the end of the line.

Figure IV.21 Installing.

There is some extra details provided by the site web so you need to follow them for some rare cases down this page.

Some dependencies have to be installed

○ `sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential.`


```

mounir@mountir-Lenovo-G580:~
mounir@mountir-Lenovo-G580:~$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
mounir@mountir-Lenovo-G580:~$ clear
mounir@mountir-Lenovo-G580:~$ sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
[sudo] password for mounir:
mounir@mountir-Lenovo-G580:~$ sudo apt install python-rosdep
[sudo] password for mounir:
mounir@mountir-Lenovo-G580:~$ sudo rosdep init
[sudo] password for mounir:
[!] Stopped sudo rosdep init
mounir@mountir-Lenovo-G580:~$ rosdep update
reading in sources list data from /etc/ros/rosdep/sources.list.d
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/osx-homebrew.yaml
^CTraceback (most recent call last):
  File "/usr/bin/rosdep", line 4, in <module>
    rosdep_main()
  File "/usr/lib/python2.7/dist-packages/rosdep2/main.py", line 144, in rosdep_main
    exit_code = _rosdep_main(args)
  File "/usr/lib/python2.7/dist-packages/rosdep2/main.py", line 426, in _rosdep_main
    return _no_args_handler(command, parser, options, args)
  File "/usr/lib/python2.7/dist-packages/rosdep2/main.py", line 435, in _no_args_handler
    return command_handlers[command](options)
  File "/usr/lib/python2.7/dist-packages/rosdep2/main.py", line 646, in command_update
    skip_eol_distro=not options.include_eol_distro)
  File "/usr/lib/python2.7/dist-packages/rosdep2/sources_list.py", line 471, in update_sources_list
    rosdep_data = download_rosdep_data(source.url)
  File "/usr/lib/python2.7/dist-packages/rosdep2/sources_list.py", line 309, in download_rosdep_data
    data = yaml.safe_load(text)
  File "/usr/lib/python2.7/dist-packages/yaml/_init_.py", line 93, in safe_load
    return load(stream, SafeLoader)
  File "/usr/lib/python2.7/dist-packages/yaml/_init_.py", line 71, in load
    return loader.get_single_data()
  File "/usr/lib/python2.7/dist-packages/yaml/constructor.py", line 37, in get_single_data
    node = self.get_single_node()
  File "/usr/lib/python2.7/dist-packages/yaml/composer.py", line 36, in get_single_node
    document = self.compose_document()
  File "/usr/lib/python2.7/dist-packages/yaml/composer.py", line 55, in compose_document
    node = self.compose_node(None, None)
  File "/usr/lib/python2.7/dist-packages/yaml/composer.py", line 84, in compose_node
    node = self.compose_mapping_node(anchor)
  File "/usr/lib/python2.7/dist-packages/yaml/composer.py", line 133, in compose_mapping_node
    item_value = self.compose_node(node, item_key)
  File "/usr/lib/python2.7/dist-packages/yaml/composer.py", line 64, in compose_node

```

Figure IV.22 Adding Dependencies.

Initialize rosdep

```
rosdep update
```

```
[sudo] password for mounir:
mounir@mountir-Lenovo-G580:~$ sudo rosdep init
[sudo] password for mounir:
```

Figure IV.23 initialize.

Now after the ROS is install it to check you need to run this command roscore

IV.5 Applying Some Algorithms on Linux Using The Terminal:

To understand the **nodes** and **topics** in ROS (robotic operation system) we must apply some algorithms

Publisher and subscriber:

```
publish/subscribe tools
roscd
roscd /some_node
rostopic list
rostopic info /some_topic
rostopic pub /some_topic msg/messagesType "data: value"
```

Figure IV.24 Some Tools In ROS.

We preferred to put some algorithms here to give sense to our work.

After all this we must prepare the work space by creating and compiling some folders

Opening the Terminal and type the following commands:

In this line we have to source the file so we can get access to our ROS tools

```
- $ source /opt/ros/kinetic/setup.bash
```

```
mounir@mounir-Lenovo-G580: ~/mybot_ws$ source devel/setup.bash
```

after compiling source
the file

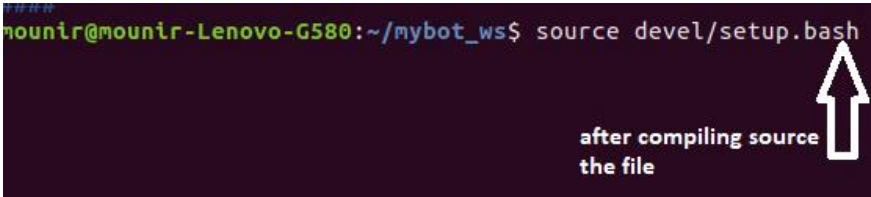


Figure IV.25 Source the File.

This following line is to create a folder inside one.

Catkin_ws and src.

Cd to access folders from command Terminals

Catkin_make is a command built our project file

```
$ mkdir -p ~/catkin_ws/src  
$ cd ~/catkin_ws/  
$ catkin_make
```

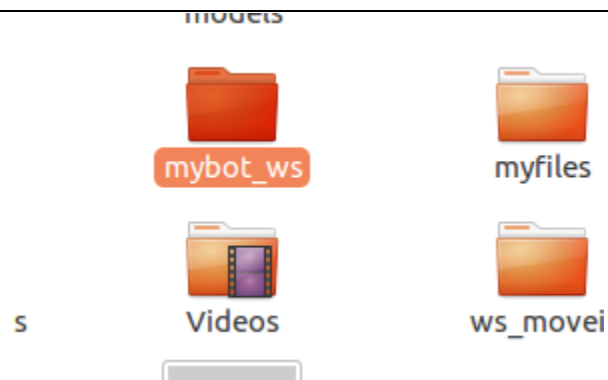


Figure IV.26Creating Work Space(mybot_ws).

Create a package with a name `beginner_tutorials` with this path:

```
Mybot_ws/src/beginner_tutorials/src/listener.cpp
```

```
Mybot_ws/src/beginner_tutorials/src/talker.cpp
```

Going back to our code:

To explain this code is like some who's righting a letter and other one who receive and read the letter.

The idea here is creating to folders each file is written with C++ language.

First file is `publisher.cpp`

Second file is `subscriber.cpp`

```

1  #include "ros/ros.h"
2  #include "std_msgs/String.h"
3  #include <sstream>
4  int main(int argc, char **argv)
5  {
6      ros::init(argc, argv, "talker");
7      ros::NodeHandle n;
8      ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
9      ros::Rate loop_rate(10);
10     int count = 0;
11     while (ros::ok())
12     {
13         std_msgs::String msg;
14         std::stringstream ss;
15         ss << "hello world " << count;
16         msg.data = ss.str();
17         ROS_INFO("%s", msg.data.c_str());
18         chatter_pub.publish(msg);
19         ros::spinOnce();
20         loop_rate.sleep();
21         ++count;
22     }
23     return 0;
24 }

```

Figure IV.27 Writing the Subscriber Node.

```

1  #include "ros/ros.h"
2  #include "std_msgs/String.h"
3  #include <sstream>
4  int main(int argc, char **argv)
5  {
6      ros::init(argc, argv, "talker");
7      ros::NodeHandle n;
8      ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
9      ros::Rate loop_rate(10);
10     int count = 0;
11     while (ros::ok())
12     {
13         std_msgs::String msg;
14         std::stringstream ss;
15         ss << "hello world " << count;
16         msg.data = ss.str();
17         ROS_INFO("%s", msg.data.c_str());
18         chatter_pub.publish(msg);
19         ros::spinOnce();
20         loop_rate.sleep();
21         ++count;
22     }
23     return 0;
24 }

```

Figure IV.28 Writing the Publisher Node.

Close and save the files.

To build and run the code open the Terminal and access to your work space which is catkin_ws

In your catkin workspace

```
$ cd ~/catkin_ws
```

```
$ catkin_make
```

To run codes

Open3 windows Terminal

```
mounir@mounir-Lenovo-G580:~/nybot_ws$ roscore
... logging to /home/mounir/.ros/log/5ef32dc8-e3e2-11ea-976e-3c970e84092d/roslaunch-mounir-Lenovo-G580-7167.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mounir-Lenovo-G580:34775/
ros_comm version 1.12.14

SUMMARY
=====
PARAMETERS
 * /roscdistro: kinetic
 * /rosverlon: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [7177]
ROS_MASTER_URI=http://mounir-Lenovo-G580:11311/

setting /run_id to 5ef32dc8-e3e2-11ea-976e-3c970e84092d
process[rosout-1]: started with pid [7190]
started core service [/rosout]
```

Figure IV.29 First Run the Roscore.

```
mounir@mounir-Lenovo-G580:~/mybot_ws$ rosrn beginner_tutorials talker
[ INFO] [1598038122.462201704]: hello world 0
[ INFO] [1598038122.562260213]: hello world 1
[ INFO] [1598038122.662239330]: hello world 2
[ INFO] [1598038122.762262809]: hello world 3
[ INFO] [1598038122.862254332]: hello world 4
[ INFO] [1598038122.962250572]: hello world 5
[ INFO] [1598038123.062268779]: hello world 6
[ INFO] [1598038123.162247755]: hello world 7
[ INFO] [1598038123.262253463]: hello world 8
```

Figure IV.30 Second Rosrun Beginner_Tutorials Talker.

```
mounir@mounir-Lenovo-G580:~/mybot_ws$ rosrn beginner_tutorials listener
[ INFO] [1598038122.762961359]: I heard: [hello world 3]
[ INFO] [1598038122.862674588]: I heard: [hello world 4]
[ INFO] [1598038122.962615725]: I heard: [hello world 5]
[ INFO] [1598038123.062761340]: I heard: [hello world 6]
[ INFO] [1598038123.162714535]: I heard: [hello world 7]
[ INFO] [1598038123.262683518]: I heard: [hello world 8]
[ INFO] [1598038123.362654776]: I heard: [hello world 9]
[ INFO] [1598038123.462705044]: I heard: [hello world 10]
[ INFO] [1598038123.562695627]: I heard: [hello world 11]
[ INFO] [1598038123.662685705]: I heard: [hello world 12]
```

Figure IV.31 Third Rosrun Beginner_Tutorials Listener.

This work of listener and talker is all to understand **nodes** and **topics** to know more we must run **rostopic list** in the Terminal and get info from our listener topic

```
Terminal File Edit View Search Terminal Tabs Help
roscore http://mounir-Lenovo-G580:1... x mounir@mounir-Lenovo-G580: ~/n
mounir@mounir-Lenovo-G580:~/mybot_ws$ rostopic list
/chatter
/rosout
/rosout_agg
mounir@mounir-Lenovo-G580:~/mybot_ws$ rostopic info /chatter
Type: std_msgs/String

Publishers:
* /talker (http://mounir-Lenovo-G580:37683/)

Subscribers:
* /listener (http://mounir-Lenovo-G580:33621/)
```

Figure IV.32 Topic from Terminal Command.

When we did check the topic /chatter we did find in his description that talker file is writing a message. And the listener is publishing to the talker to receive that message.

IV.6 Build Our Robots In Linux.

This is the main part of the work is to build our robot and simulate it in virtual application like gazebo and rviz and place our sensor in our robot and try some mapping algorithms.

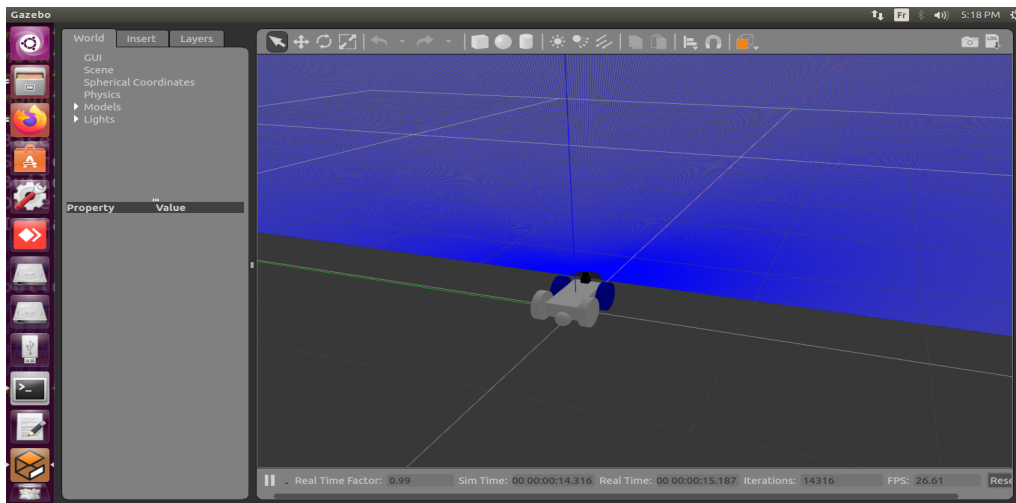


Figure IV.33 Robot Final Goals.

IV.6.1 Creating Our Work Space (robot folder):

First thing you want to do is creating a folder with your robot name (mybot_ws)

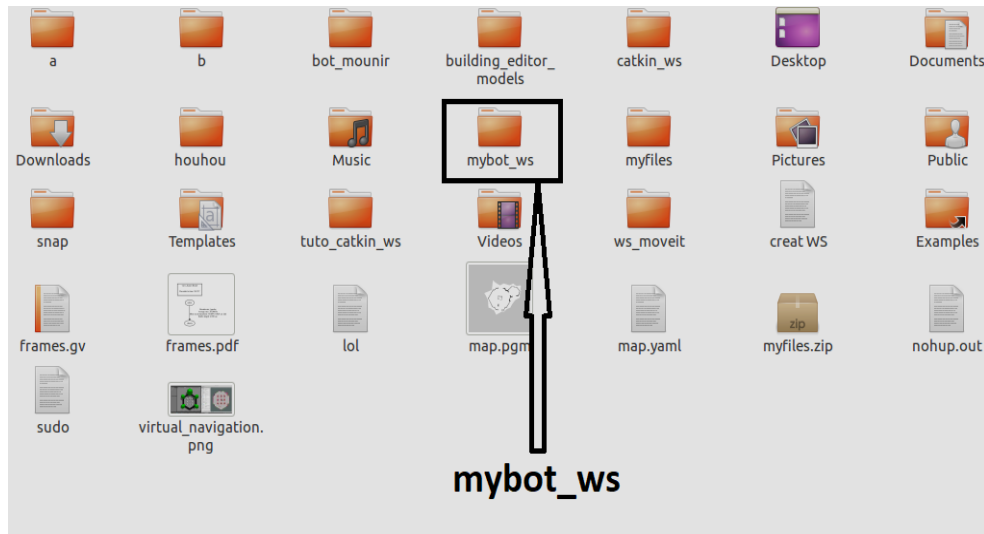


Figure IV.34 Working Space Folder.

IV.6.2 Creating Src File Inside the Working Space and Compile the Folder:

Creating a file named src to hold our robot code

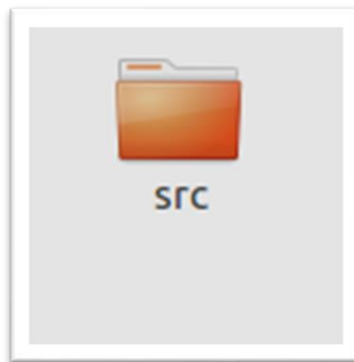


Figure IV.35 Src Robot Code And Configuration File.

Compiling this folder to make sure that all ROS packages and tools can be uploaded with two files automatically throw the Terminal.

```
/mybot_ws/src /mybot_ws/build /mybot_ws/devel
```



```

mounir@mounir-Lenovo-G580: ~/mybot_ws
mounir@mounir-Lenovo-G580:~$ cd mybot_ws/
mounir@mounir-Lenovo-G580:~/mybot_ws$ cd src
mounir@mounir-Lenovo-G580:~/mybot_ws/src$ cd
mounir@mounir-Lenovo-G580:~$ cd mybot_ws/
mounir@mounir-Lenovo-G580:~/mybot_ws$ catkin_make
Base path: /home/mounir/mybot_ws
Source space: /home/mounir/mybot_ws/src
Build space: /home/mounir/mybot_ws/build
Devel space: /home/mounir/mybot_ws/devel
Install space: /home/mounir/mybot_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/mounir/mybot_ws/build"
####
####
#### Running command: "make -j4 -l4" in "/home/mounir/mybot_ws/build"
####
mounir@mounir-Lenovo-G580:~/mybot_ws$ source devel/setup.bash

```

access the robot file

run this command catkin_make to build this folder

after compiling source the file

Figure IV.36 Building The Project And Source The File.

Source the file means saying that were going to use the ROS tools and packages only in this file that were going to create it later.

After building the file two extra files will appears build and devel.

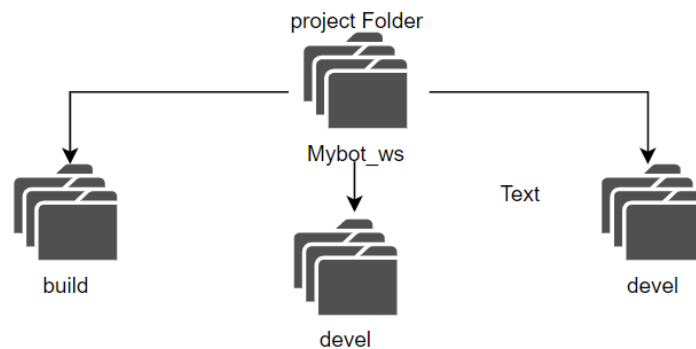


Figure IV.37 Mybot_Ws File After The Compiling.

IV.6.3 Creating our robot packages:

After compiling the project file, we need to create our packages who has the shape and design of our robot.

You can now more about packages in chapter 3 start your project.

To create a package, you need a name for this package and some parameters

Catkin_create_pkgname_of_packageparam1 param2 param 3

Packages 1:: Catkin_create_pkmybot_description

Packages 2:: Catkin_create_pkmybot_gazebo

```
mounir@mounir-Lenovo-G580: ~/mybot_ws
mounir@mounir-Lenovo-G580:~$ cd mybot_ws/
mounir@mounir-Lenovo-G580:~/mybot_ws$ cd src
mounir@mounir-Lenovo-G580:~/mybot_ws/src$ cd
mounir@mounir-Lenovo-G580:~$ cd mybot_ws/
mounir@mounir-Lenovo-G580:~/mybot_ws$ catkin_make
base path: /home/mounir/mybot_ws
source space: /home/mounir/mybot_ws/src
build space: /home/mounir/mybot_ws/build
devel space: /home/mounir/mybot_ws/devel
install space: /home/mounir/mybot_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/mounir/mybot_ws/build"
####
####
#### Running command: "make -j4 -l4" in "/home/mounir/mybot_ws/build"
####
mounir@mounir-Lenovo-G580:~/mybot_ws$ catkin_create_pkg mybot_description
```

Figure IV.38 Create Mybot_Description Package.

```
mounir@mounir-Lenovo-G580:~$ cd mybot_ws/
mounir@mounir-Lenovo-G580:~/mybot_ws$ cd src
mounir@mounir-Lenovo-G580:~/mybot_ws/src$ cd
mounir@mounir-Lenovo-G580:~$ cd mybot_ws/
mounir@mounir-Lenovo-G580:~/mybot_ws$ catkin_make
Base path: /home/mounir/mybot_ws
Source space: /home/mounir/mybot_ws/src
Build space: /home/mounir/mybot_ws/build
Devel space: /home/mounir/mybot_ws/devel
Install space: /home/mounir/mybot_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/mounir/mybot_ws/build"
####
####
#### Running command: "make -j4 -l4" in "/home/mounir/mybot_ws/build"
####
mounir@mounir-Lenovo-G580:~/mybot_ws$ catkin_create_pkg mybot_gazebo
```

Figure IV.39 Mybot_Description Package.

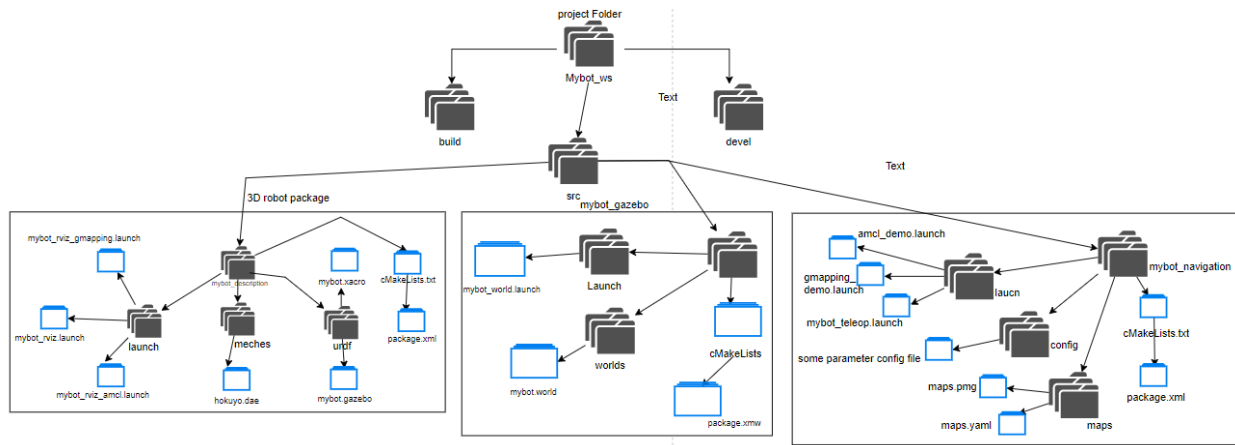


Figure IV.40 src file with robot packages.

Each package contains cMakeLists.txt and package.xml and some codes inside that were going to discuss later in this chapter. Last thing in the folder path design is packages contains.

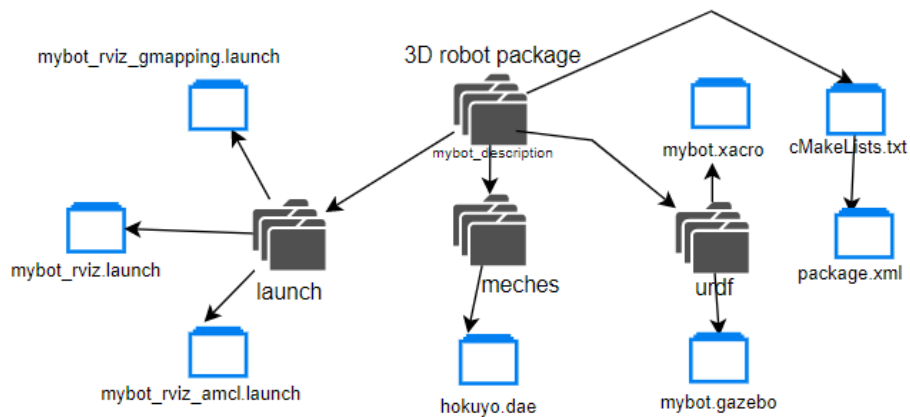


Figure IV.41 Mybot_Discription Package folder.

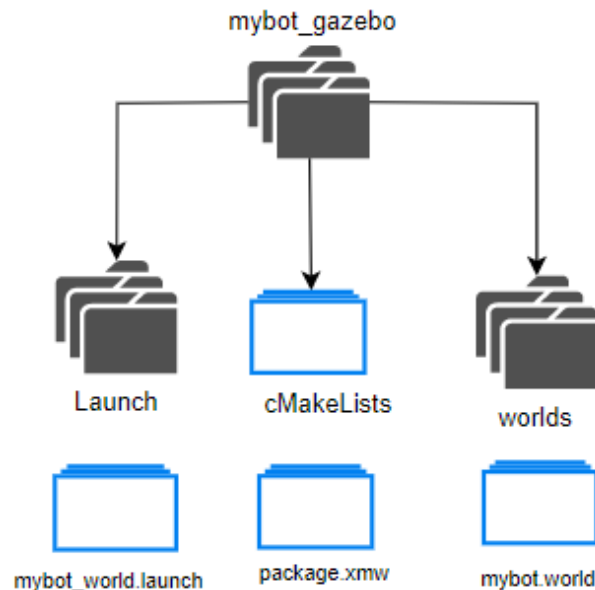


Figure IV.42 mybot.gazebo Package folder.

IV.6.4 Robot Coding:

We will learn now the Principle of using some new language rather than C++ or python using nodes and topics and how will our robot going to form

❖ Starting with Gazebo Package:

This package is responsible for the launching all nodes and call all files from the description package such as the 3D design and sensors wheels, colors. Also, this file is the main who's going to publish some information and share it with us.

○ Starting with The Launch File

(mybot_ws/src/mybot_gazebo/mybot_world.launch)

This is the launch file correspondent for launching our project

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <launch>
3 <!--/ set some parameters -->
4 <arg name="world" default="empty"/>
5 <arg name="paused" default="false"/>
6 <arg name="use_sim_time" default="true"/>
7 <arg name="gui" default="true"/>
8 <arg name="headless" default="false"/>
9 <arg name="debug" default="false"/>
10
11 <!--/ this code launch an empty world with a name and some paramerters -->
12 <include file="$(find gazebo_ros)/launch/empty_world.launch">
13 <arg name="world_name" value="$(find mybot_gazebo)/worlds/mybot.world"/>
14 <arg name="paused" value="$(arg paused)"/>
15 <arg name="use_sim_time" value="$(arg use_sim_time)"/>
16 <arg name="gui" value="$(arg gui)"/>
17 <arg name="headless" value="$(arg headless)"/>
18 <arg name="debug" value="$(arg debug)"/>
19 </include>
20
21 <!--/ this is to call the urdf file corespondent for the robot desine -->
22 <param name="robot_description" command="$(find xacro)/xacro.py '$(find mybot_description)/urdf/mybot.xacro'"/>
23
24 <!--/ this is a node named mybot_spawn to find a gazebo_ros and spawn the model with some parameters -->
25 <node name="mybot_spawn" pkg="gazebo_ros" type="spawn_model" output="screen"
26 <args="-urdf -param robot_description -model mybot" />
27 </launch>
28

```

Figure IV.43 mybot_world.launch.

- World file this file is correspondent for setting the parameter like the sun and camera position.

```

1 <?xml version="1.0" ?>
2 <sdf version="1.4">
3 <!-- We use a custom world for the rrobot so that the camera angle is launched correctly -->
4
5 <world name="default">
6 <!--include>
7 | <uri>model://willowgarage</uri>
8 </include-->
9
10 <include>
11 | <uri>model://ground_plane</uri>
12 </include>
13
14 <!-- light source -->
15 <include>
16 | <uri>model://sun</uri>
17 </include>
18
19 <!-- Focus camera on tall pendulum or setting position of the camera -->
20 <gui fullscreen='0'>
21 <camera name='user_camera'>
22 <pose>4.927360 -4.376610 3.740080 0.000000 0.275643 2.356190</pose>
23 <view_controller>orbit</view_controller>
24 </camera>
25 </gui>
26
27 </world>
28 </sdf>

```

Figure IV.44 World Launch File.

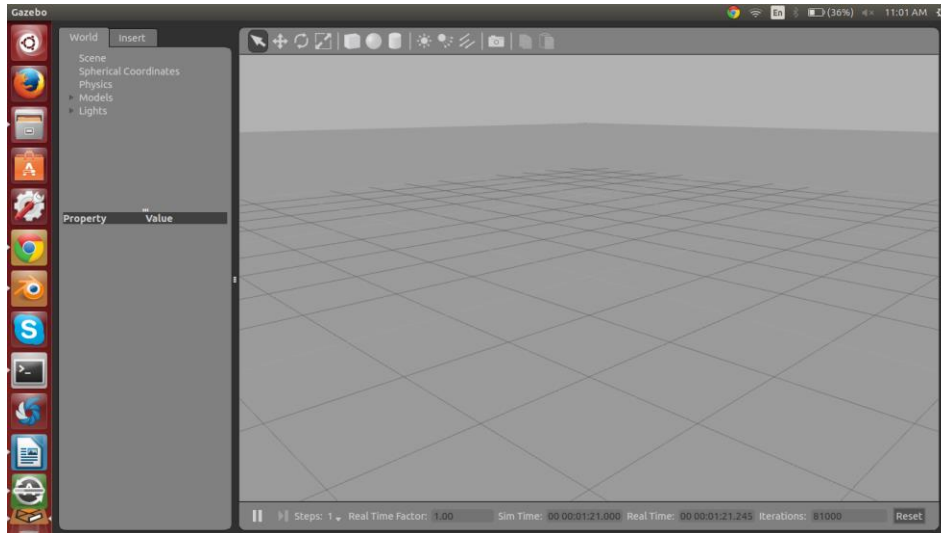


Figure IV.45 Gazebo Application From The Launch File

❖ Description Package:

This file is to set our 3D robot model using the Urdf or Xarco language we did choose to go with Xarco because it's simpler and more practical to form your robot

○ Starting by creating a folder calls URDF and creating 4 files. Each file has its own preps like colors, 3D design and plugins.

➤ Colors file code: `mybot_ws/src/ mybot_description /urdf/ materials.xacro` this file is correspondent for robot colors and calling this file from the `mybot.gazebo` like in (**figure IV.46**) line 8 from the code.

➤ Plugins robot file code `mybot_ws/src/ mybot_description /urdf/ mybot.gazebo` (**figure IV.47**)

➤ 3D design file code: `mybot_ws/src/ mybot_description /urdf/ mybot.xacro` (**figure IV.46**)

```
C:\Users\houho\Desktop\project mounir\mybot_ws\src\mybot_description\urdf> materials.xacro
1 <?xml version="1.0"?>
2 <robot>
3 <!--this colors is setted by declaring some parameters
4 like rgba and calling this file from my mybot.gazebo file -->
5 <material name="black">
6 | <color rgba="0.0 0.0 0.0 1.0"/>
7 </material>
8
9 <material name="blue">
10 | <color rgba="0.0 0.0 0.8 1.0"/>
11 </material>
12
13 <material name="green">
14 | <color rgba="0.0 0.8 0.0 1.0"/>
15 </material>
16
17 <material name="grey">
18 | <color rgba="0.2 0.2 0.2 1.0"/>
19 </material>
20
21 <material name="orange">
22 | <color rgba="{255/255} ${108/255} ${10/255} 1.0"/>
23 </material>
24
25 <material name="brown">
26 | <color rgba="{222/255} ${207/255} ${195/255} 1.0"/>
27 </material>
28
29 <material name="red">
30 | <color rgba="0.8 0.0 0.0 1.0"/>
31 </material>
32
33 <material name="white">
34 | <color rgba="1.0 1.0 1.0 1.0"/>
35 </material>
36
37 </robot>
38
```

Figure IV.46 Colors File
(materials.xacro).

```

C:\Users\houho\Desktop> project mounir > mybot_ws > src > mybot_description > urdf > mybot.gazebo
1 <?xml version="1.0"?>
2 <robot>
3
4 <!--this file is to sett the robot to be shown in our gazebo applicaiton -->
5 <gazebo>
6   <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
7     <legacyMode>>false</legacyMode>
8     <alwaysOn>>true</alwaysOn>
9     <updateRate>20</updateRate>
10    <leftJoint>left_wheel_hinge</leftJoint> <!--must call joints from the myot.xacro file -->
11    <rightJoint>right_wheel_hinge</rightJoint><!--must call joints from the myot.xacro file -->
12    <rightJoint>right_back_wheel_hinge</rightJoint><!--must call joints from the myot.xacro file -->
13    <rightJoint>left_back_wheel_hinge</rightJoint><!--must call joints from the myot.xacro file -->
14    <wheelSeparation>0.4</wheelSeparation>
15    <wheelDiameter>0.1</wheelDiameter>
16    <torque>20</torque>
17    <commandTopic>cmd_vel</commandTopic>
18    <odometryTopic>odom</odometryTopic>
19    <odometryFrame>odom</odometryFrame>
20    <robotBaseFrame>chassis</robotBaseFrame>
21  </plugin>
22 </gazebo>
23
24 <gazebo reference="chassis">
25   <material>Gazebo/blue</material><!--must call matrial from the materials.xacro file -->
26 </gazebo>
27
28 <gazebo reference="left_wheel">
29   <material>Gazebo/green</material><!--must call matrial from the materials.xacro file -->
30 </gazebo>
31
C:\Users\houho\Desktop> project mounir > mybot_ws > src > mybot_description > urdf > mybot.gazebo
41 <!-- hokuyo -->
42 <gazebo reference="hokuyo_link">
43 <sensor type="ray" name="head_hokuyo_sensor">
44 <pose>0 0 0 0 0 0</pose>
45 <visualize>true</visualize>
46 <update_rate>40</update_rate>
47 <ray>
48 <scan>
49 <horizontal>
50 <samples>720</samples>
51 <resolution>1</resolution>
52 <min_angle>-1.570796</min_angle>
53 <max_angle>1.570796</max_angle> <!--sensor plugin to make the sensor work and send laser data -->
54 </horizontal>
55 </scan>
56 <range>
57 <min>0.10</min>
58 <max>30.0</max>
59 <resolution>0.01</resolution>
60 </range>
61 <noise>
62 <type>gaussian</type>
63 <!-- Noise parameters based on published spec for Hokuyo laser
64 achieving "+-30mm" accuracy at range < 10m. A mean of 0.0m and
65 stddev of 0.01m will put 99.7% of samples within 0.03m of the true
66 reading. -->
67 <mean>0.0</mean>
68 <stddev>0.01</stddev>
69 </noise>
70 </ray>
71 <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
72 <topicName>/mybot/laser/scan</topicName> <!--sending laser data from topic /mybot/laser/scan -->
73 <frameName>hokuyo_link</frameName>
74 </plugin>
75 </sensor>
76 </gazebo>
77
78
79

```

Figure IV.47 Gazebo Application Code Plugins([gazebo.xacro](#)).


```
C: > Users > houh > Desktop > project mounir > mybot_ws > src > mybot_description > urdf > mybot.xacro
1  <?xml version='1.0'?>
2
3  <robot name="myrobot" xmlns:xacro="http://www.ros.org/wiki/xacro">
4
5
6
7  <xacro:include filename="$(find mybot_description)/urdf/mybot.gazebo" /> <!--calling the file mybot.gazebo -->
8  <xacro:include filename="$(find mybot_description)/urdf/materials.xacro" /><!--calling the file materials.gazebo -->
9  <xacro:include filename="$(find mybot_description)/urdf/macros.xacro" /><!--calling the file macros.gazebo -->
10
11
12 <link name='chassis'> <!--chassis robot of the body -->
13   <pose>0 0 0.1 0 0 0</pose>
14
15   <inertial>
16     <mass value="10.0"/>
17     <origin xyz="0.0 0 0.1" rpy=" 0 0 0"/>
18     <inertia
19       ixx="0.5" ixy="0" ixz="0"
20       iyy="1.0" iyz="0"
21       izz="0.1"
22     />
23   </inertial>
24
25   <collision name='collision'>
26     <geometry>
27       <box size=".4 .2 .1"/>
28     </geometry>
29   </collision>
30
```

Figure IV.48 3D Design Robot Code (mybot.xacro).

There is a plugin for wheels and chassis and hokuyo sensor (RPLIDAR sensor). So that it can be displayed on the gazebo you must place every joints in this file (**figure IV.45**) and even the material like colors (**figure IV.46**).

```
C:\Users\houho\Desktop\project mounir\mybot_ws\src\mybot_description\urdf\mybot.xacro
94 </link>
95
96 <link name="right_wheel">
97   <!--origin xyz="0.1 -0.13 0.1" rpy="0 1.5707 1.5707"/-->
98   <collision name="collision">
99     <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
100    <geometry>
101      <cylinder radius="0.1" length="0.05"/>
102    </geometry>
103  </collision>
104  <visual name="right_wheel_visual">
105    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
106    <geometry>
107      <cylinder radius="0.1" length="0.05"/>
108    </geometry>
109    <material name="green"/>
110  </visual>
111  <inertial>
112    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
113    <mass value="5"/>
114    <cylinder_inertia m="5" r="0.1" h="0.05"/>
115    <inertia
116      ixx="1.0" ixy="0.0" ixz="0.0"
117      iyy="1.0" iyz="0.0"
118      izz="1.0"/>
119  </inertial>
120 </link>
121
122
```

Figure IV.49 Right Wheel Code Link(mybot.xacro).

The chassis is the main robot core which is responsible for holding the wheels and the sensor. Our robot has four wheels and a laser sensor on top of it, to explain the code we're going to give an example for the right wheel.

This link code is responsible for creating a small wheel in shape of cylinder with radius 0.1 and length 0.05 you can set this parameter as you like according to your robot, you must set the position of the wheel from the next code in **figure IV.49**

```
C: > Users > houho > Desktop > project mounir > mybot_ws > src > mybot_description > urdf > mybot.xacro
183 </joint>
184
185 <joint type="continuous" name="right_wheel_hinge"> <!-- joint responsible for the position of the robot -->
186 <origin xyz="0.1 -0.15 0" rpy="0 0 0"/>
187 <child link="right_wheel"/><!-- creat a joint axe x y z and pare it with the 3D wheel model -->
188 <parent link="chassis"/> <!-- pare the axe with the chassis joint -->
189 <axis xyz="0 1 0" rpy="0 0 0"/> <!-- axe position -->
190 <limit effort="100" velocity="100"/>
191 <joint_properties damping="0.0" friction="0.0"/>
192 </joint>
193
194 <joint type="continuous" name="right_back_wheel_hinge">
195 <origin xyz="-0.15 -0.15 0" rpy="0 0 0"/>
196 <child link="right_back_wheel"/>
197 <parent link="chassis"/>
198 <axis xyz="0 1 0" rpy="0 0 0"/>
199 <limit effort="100" velocity="100"/>
200 <joint_properties damping="0.0" friction="0.0"/>
201 </joint>
202
203 <joint type="continuous" name="left_back_wheel_hinge">
204 <origin xyz="-0.15 0.15 0" rpy="0 0 0"/>
205 <child link="left_back_wheel"/>
206 <parent link="chassis"/>
```

Figure IV.50 Right Wheel Joint (mybot.xacro).

The right wheel joint is displayed also in the file gazebo.xacro so that can appears in gazebo application (figure IV.47) line 11 from the code.

Each wheel is represented by this code but you must place the right position of the wheels like the right back wheel is has just the same code as the front right wheel there is a name

and position difference like in (figure IV.49)

```

185 <joint type="continuous" name="right_wheel_hinge"> <!-- joint responsible for the position of the robot -->
186 <origin xyz="0.1 -0.15 0" rpy="0 0 0"/>
187 <child link="right_wheel"/> <!-- create a joint axe x y z and pare it with the 3D wheel model -->
188 <parent link="chassis"/> <!-- pare the axe with the chassis joint -->
189 <axis xyz="0 1 0" rpy="0 0 0"/> <!-- axe position -->
190 <limit effort="100" velocity="100"/>
191 <joint_properties damping="0.0" friction="0.0"/>
192 </joint>
193
194 <joint type="continuous" name="right_back_wheel_hinge">
195 <origin xyz="-0.15 -0.15 0" rpy="0 0 0"/>
196 <child link="right_back_wheel"/>
197 <parent link="chassis"/>
198 <axis xyz="0 1 0" rpy="0 0 0"/>
199 <limit effort="100" velocity="100"/>
200 <joint_properties damping="0.0" friction="0.0"/>
201 </joint>

```

the difference

must be the link name

Figure IV.51 The Difference Between The Right Wheel And Right Back

IV.6.5 Compiling Our Project Work Space:

You can run your project throw ROS tools by typing some commands to run specific launch file you must use rosrn command in Terminal window.

```
roslaunch package_name file_name.launch.
```

you can run this command or do something else that were going to explain here:

go to mybot_ws/ directory create a file with this name run_gazebo.sh

so we can add the following line inside that file like in (figure IV.50)

```

C: > Users > houho > Desktop > project mounir > mybot_ws > run_gazebo.sh
1  #!/bin/bash
2
3  sudo killall rosmaster  ### preventing from launching more then rosmaster or roscore
4  sudo killall gzserver
5  sudo killall gzclient
6  roslaunch mybot_gazebo mybot_world.launch  ### add the roslaunch to launch our packages |
7

```

Figure IV.52 how to launch the project file run_gazebo.sh.

To run this file run this command like in (figure IV.53)

```
./run_gazebo.sh
```

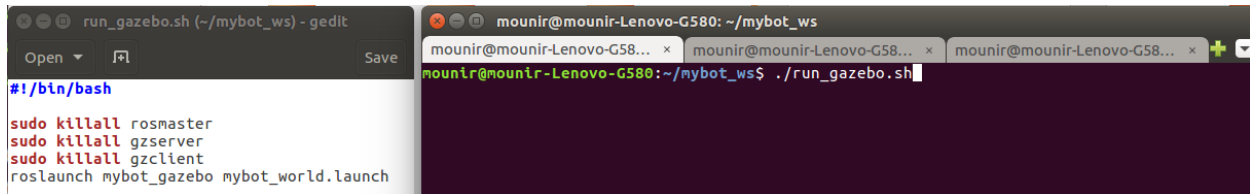


Figure IV.53 The File Who Launch Our Robot Packages.

before you run this file, you must make sure that you did source the work space with this command (source devel/setup.bash) so the ROS tools can be accessed from this file.

After you source the work space run the file like in (**figure IV.25**), soon as you tape the enter key the gazebo application should appears with the 3D robot design like in (**figure IV.54**)

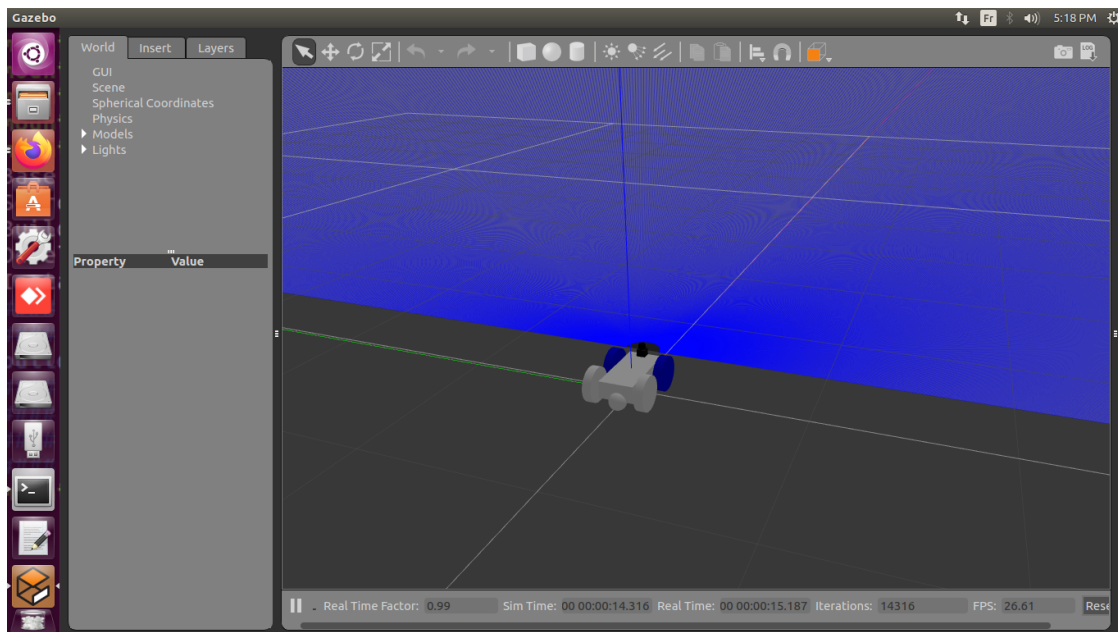


Figure IV.54 3D Robot Design Displayed In Gazebo Application.

If the robot dose not appears make sure you did write all your code correct because even if the name off the right wheel dose not match you will get an error in your code.

Next, we will launch the rviz to visual the robot from other side which is topics:

We can launch it by typing this command

```
roslaunch mybot_description mybot_rviz.launch
```

or create a sh file so that we can call the rviz launch file like in **(figure IV.55)**

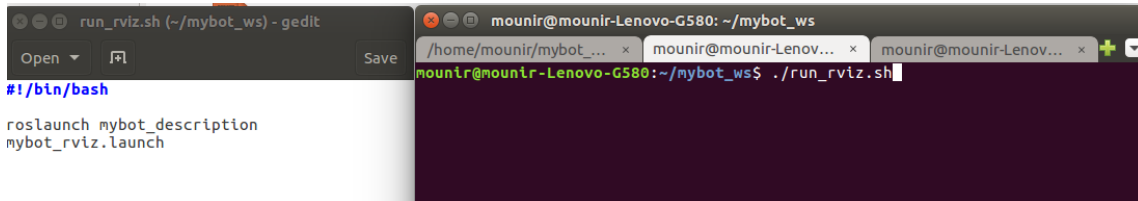


Figure IV.55 Rviz Launch File.

After launching this file rviz will appear to you with a blank world space **(figure IV.56)** so you need to set some parameters to see your 3D model robot.

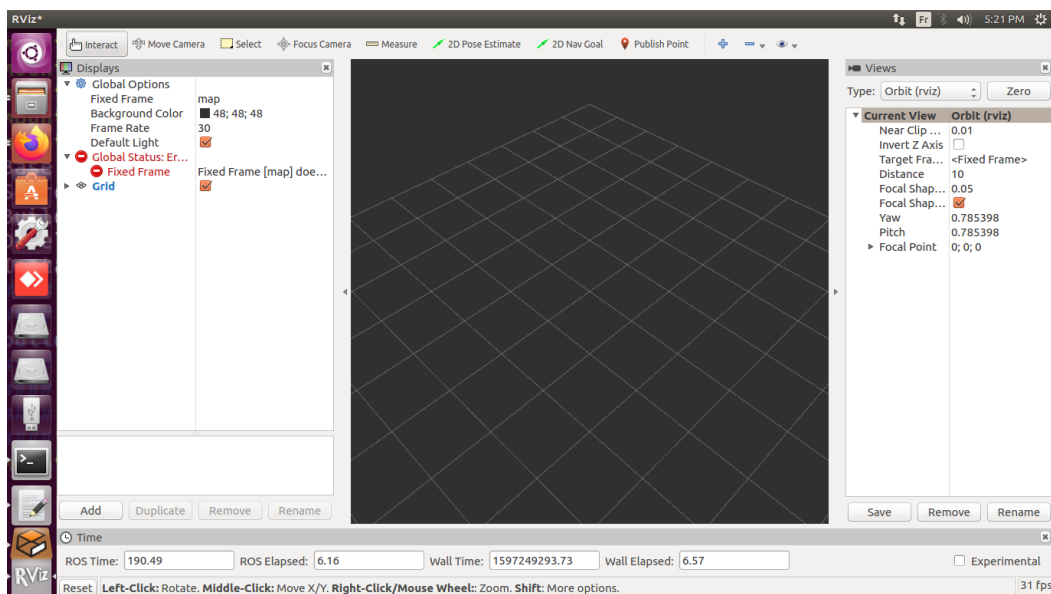


Figure IV.56 Rviz Visualization Application.

❖ **Setting up the rviz parameters:**

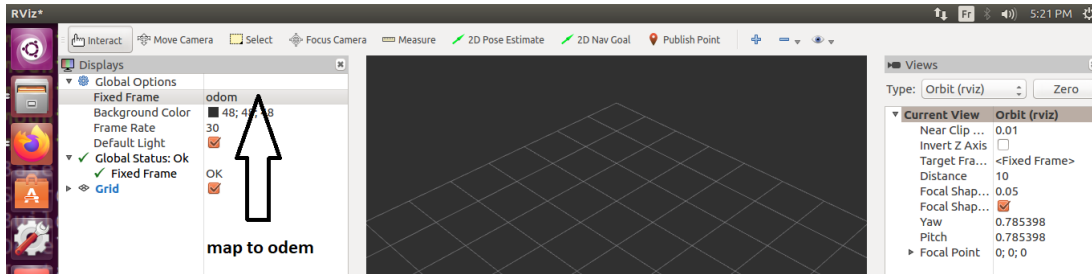


Figure IV.57 Setting Map to Odom.

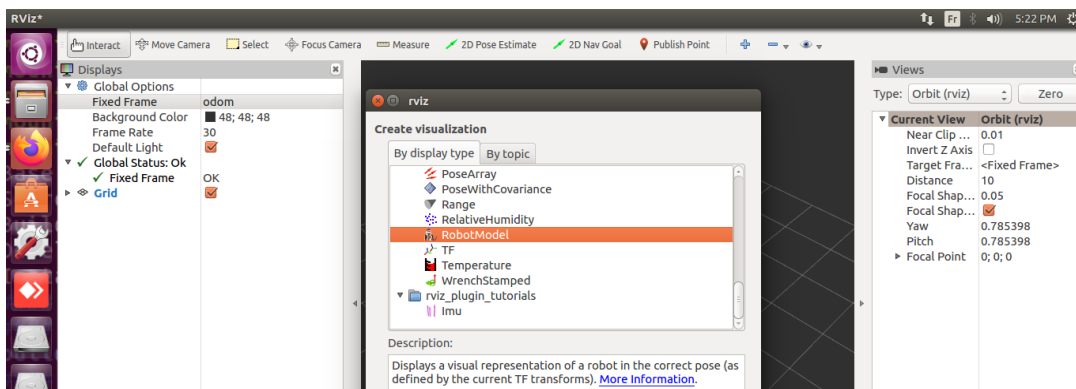


Figure IV.58 Adding Our Robot Model from This List.

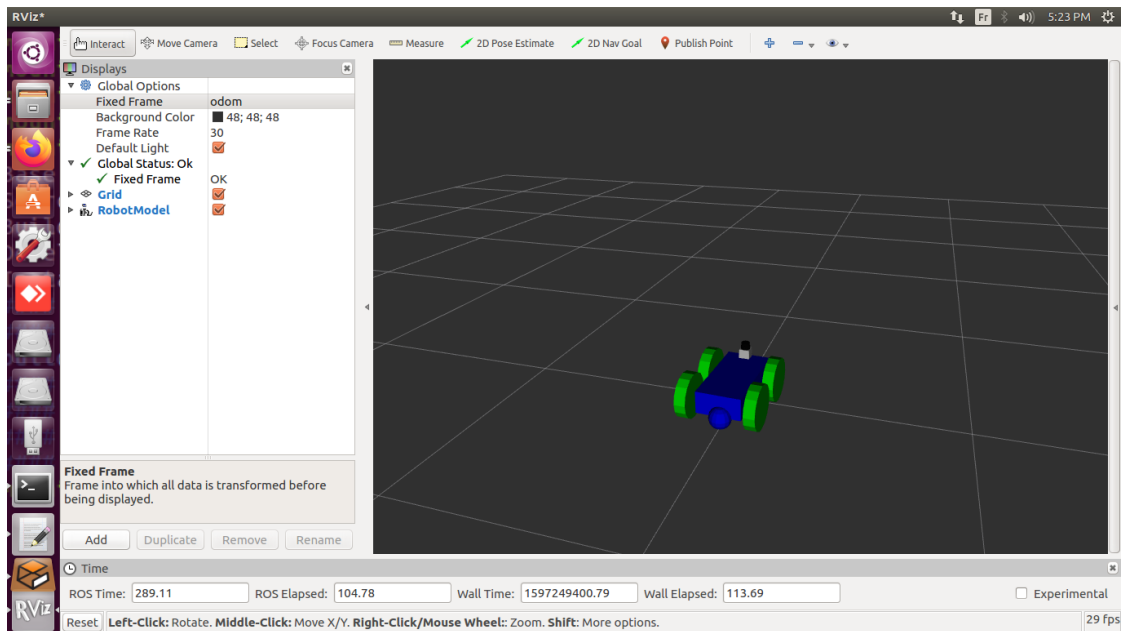


Figure IV.59 3D Robot Model From Our Description Package In Rviz.

Search the robot model by searching it in the list (**figure IV.58**).

as you add the robot model your robot will appear in the blank space (**figure IV.59**)

You can see the code in gazebo packages that we did set the colors (**figure IV.47**) in following order:

- ❖ Chassis blue.
- ❖ Wheels green.

To resume our work from building our work space to compiling our robot and see this result:

- Create work space.
- Building the work space.
- Create the packages
- Config our lunch files.
- Design our robot with xacro language.

- Source the work space file
 - Compile the project and see results.
- Problems that you're going to face:
- Some errors in xml language.
 - Errors in compiling due to bad installation of ROS kinetic.
 - Missing some libraries (must be installed).
 - Strong 3D projects need strong GPU.

IV.6.6 Testing Our Sensor (RPLIDAR):

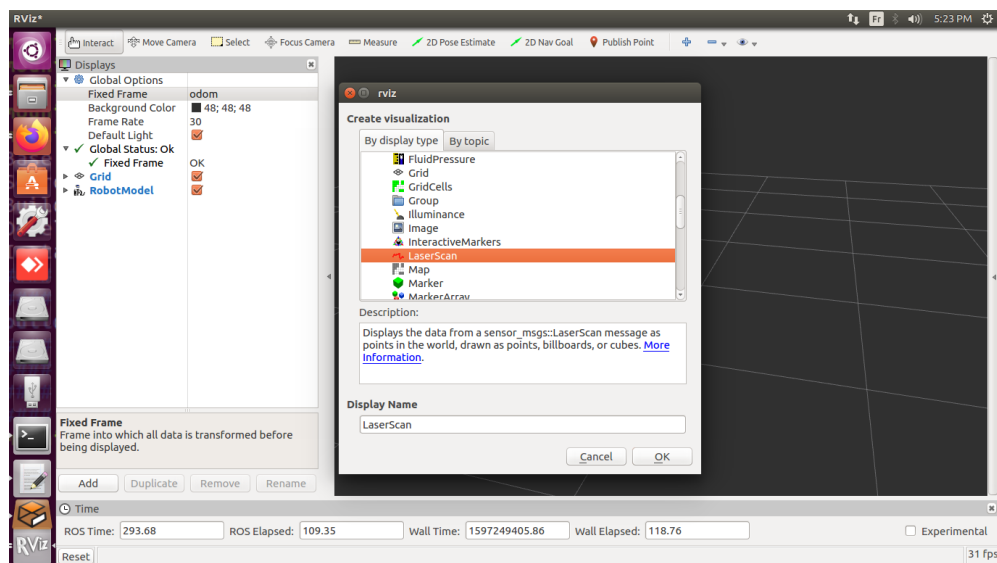


Figure IV.60 Get the Laser Scan.

When you select the laser scan you must do some modification and select the

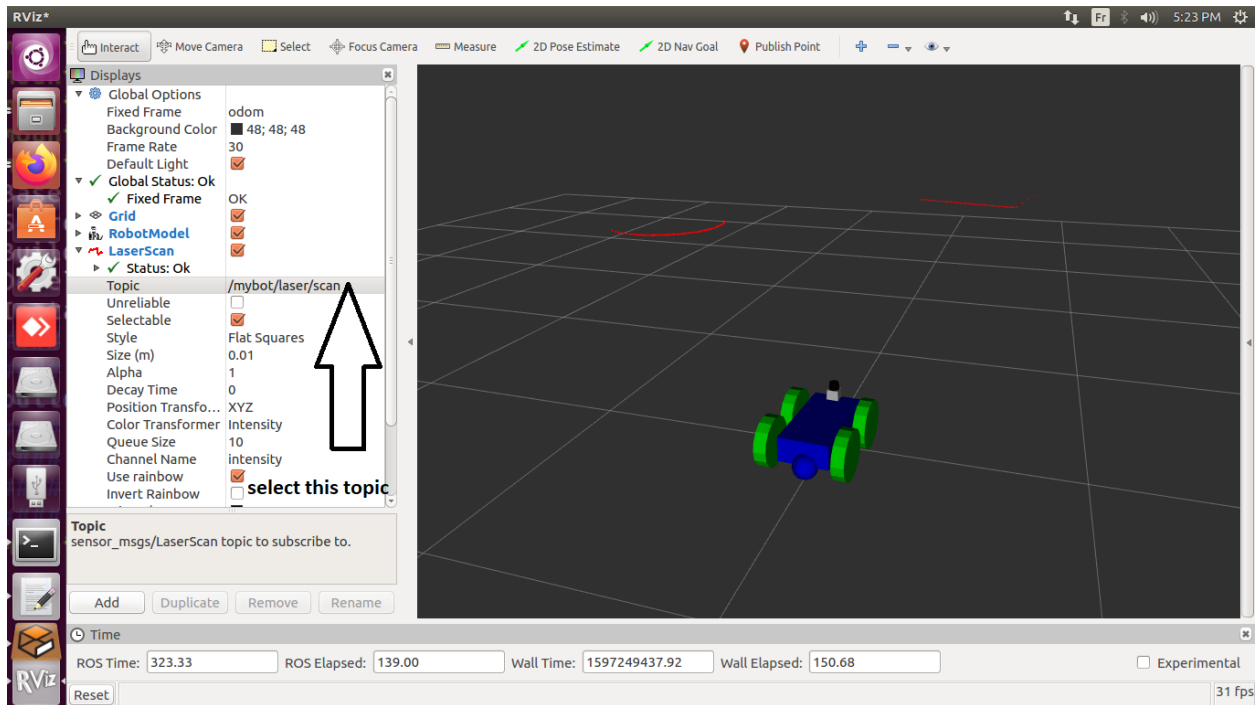


Figure IV.61 Select The Topic of laser scan.

mybot/laser/scan topic so you can get results from the laser sensor

Before you select this topic go to gazebo try to put some obstacle to test the laser

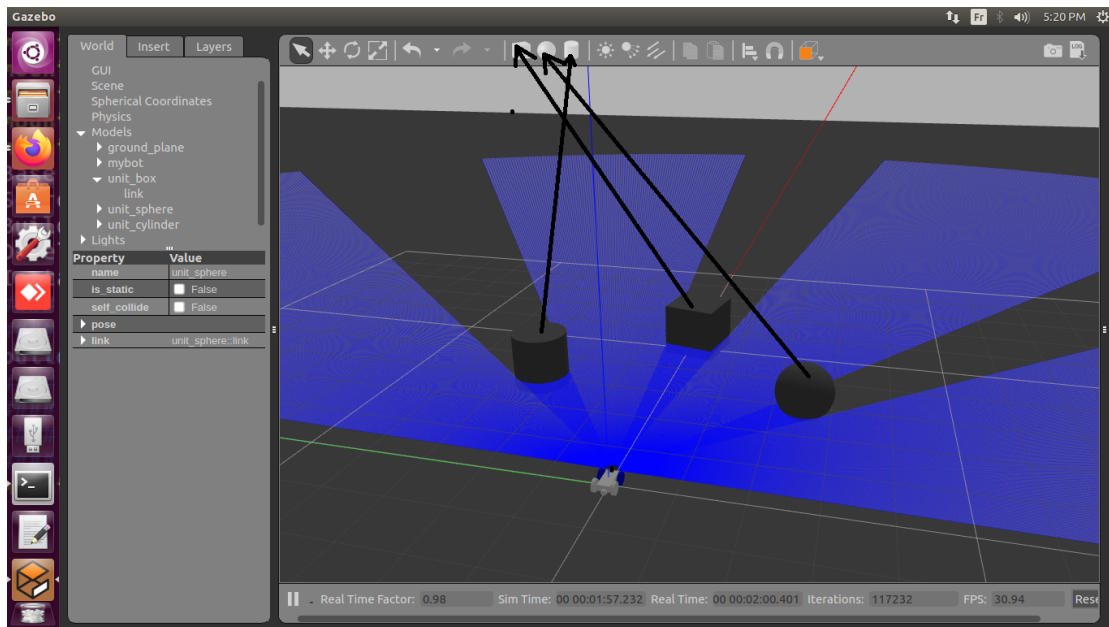


Figure IV.62 Adding Some Spheres and Cube.

Soon as you add the obstacle return to rviz and select the topic like in **(figure IV.63)** and you will receive the laser data in shape of red points like in **(figure IV.64)**.

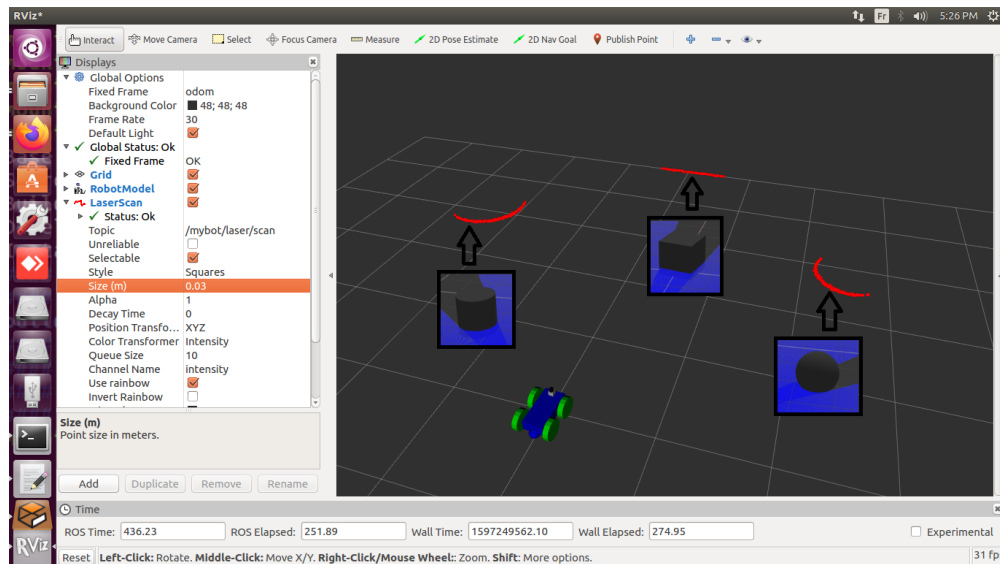


Figure IV.63 The Readings Is Imported From Gazebo Throw The Topic To Rviz.

We tried to move the shapes from their initial like in (figure IV.61) positions and the results of the laser changed.

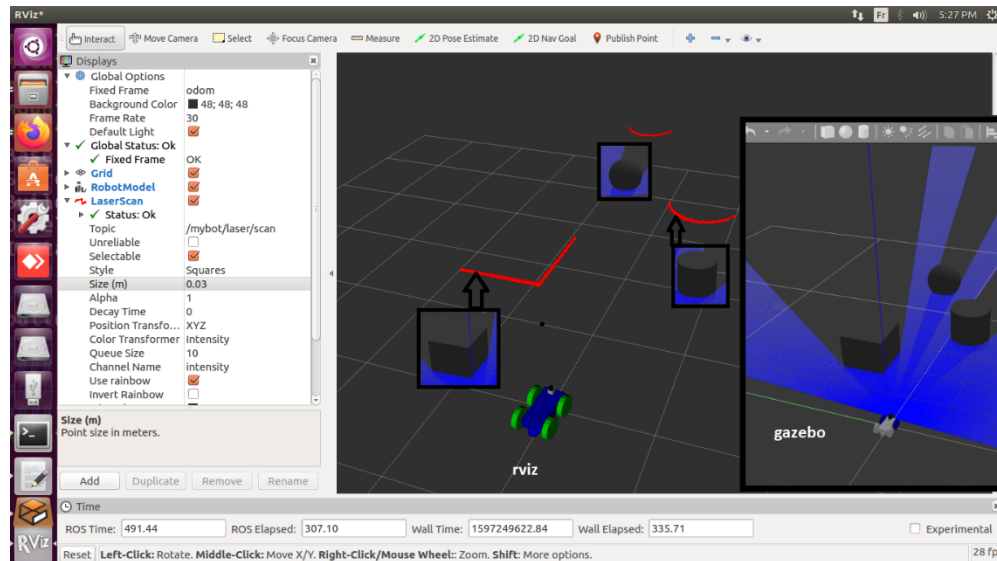


Figure IV.64 Results After Moving The Shapes.

IV.7 How Is Our Robot Attached?

To understand our code in (figure IV.47) were going to generate a tree of our joints of our robot which part is connected with.

```

mounir@mounir-Lenovo-G580: ~/mybot_ws
/home/mounir/mybot_... x /home/mounir/mybot_... x mounir@mounir-Lenov... x
mounir@mounir-Lenovo-G580:~/mybot_ws$ rqt_graph
mounir@mounir-Lenovo-G580:~/mybot_ws$ rosrn tf view_frames
listening to /tf for 5.000000 seconds
Done Listening
dot - graphviz version 2.38.0 (20140413.2041)
detected dot version 2.38
frames.pdf generated
mounir@mounir-Lenovo-G580:~/mybot_ws$ evince frames.pdf

```

run this command to register the tf topic

open the pdf with this command

Figure IV.65 Open the Robot Attach Tree.

A window will open with the pdf contain a description about how our robot is attached with every joint (**figure IV.66**).

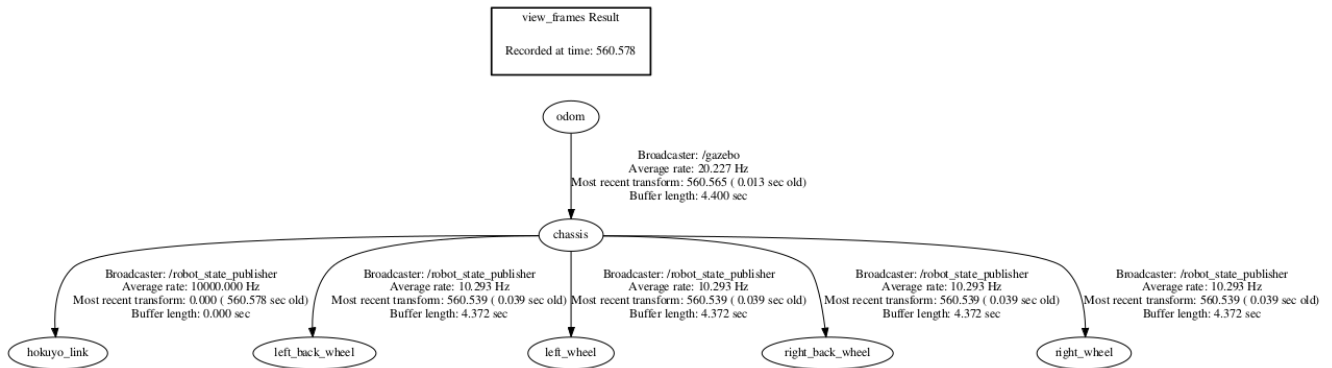


Figure IV.66 Frames Results.

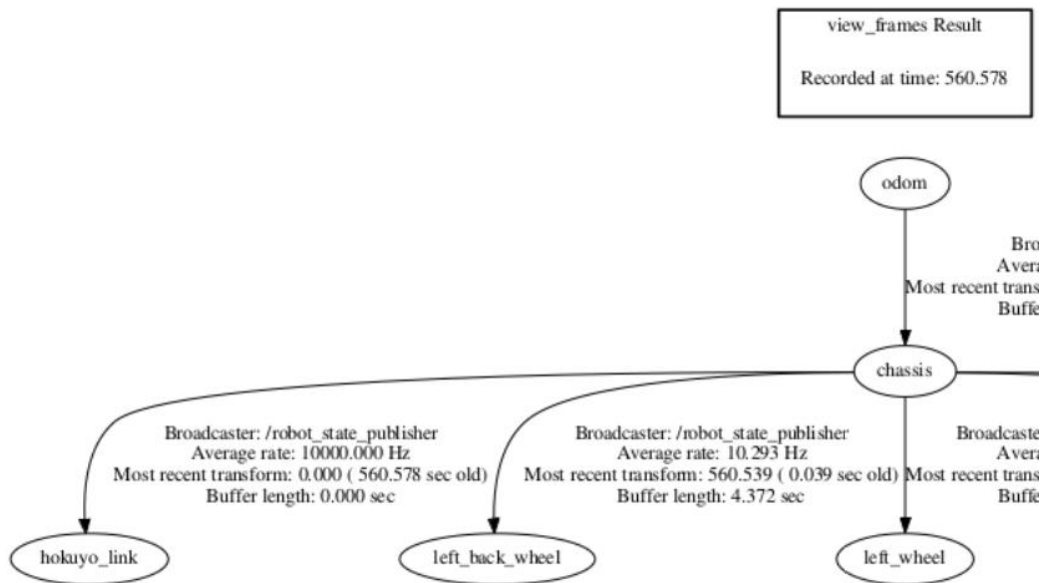


Figure IV.67 Left Side from The Frames.

This side of the frame describe where the (left_wheel, left_back_wheel and hokuyo sensor) is attached which is the chassis.

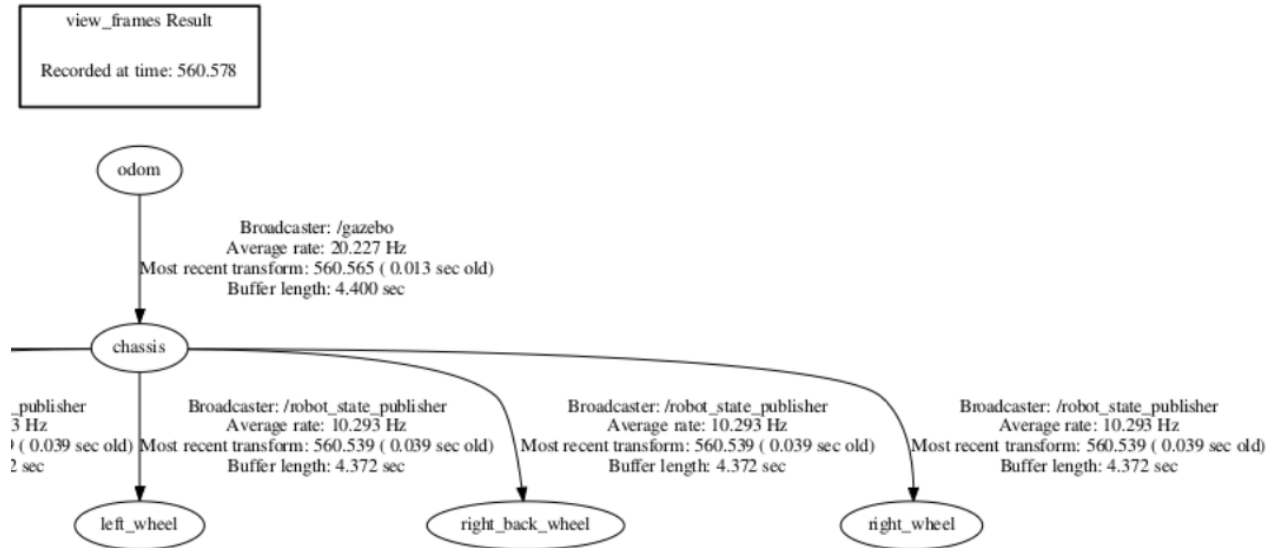
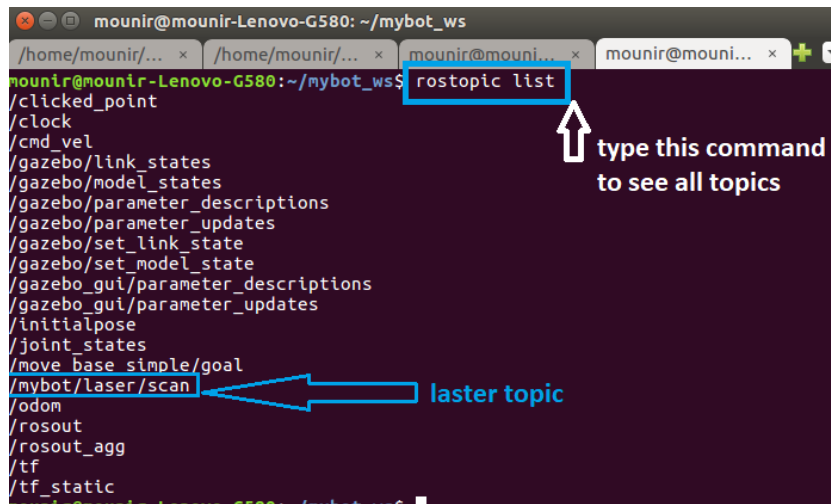


Figure IV.68The Right Side from The Frame.

This side of the frame describe where the (right_wheel, right_back_wheel) is attached which is the chassis.

Chassis: which is the main core of our robot, The chassis is broadcasting to odom which all information is attached to gazebo throw odom to transfer the direction, robot position and the laser data from gazebo and share it with rviz.

To understand this last thing which is how the information be transferred throw nodes and topics we must use the tools that exist int **figure IV.24**.



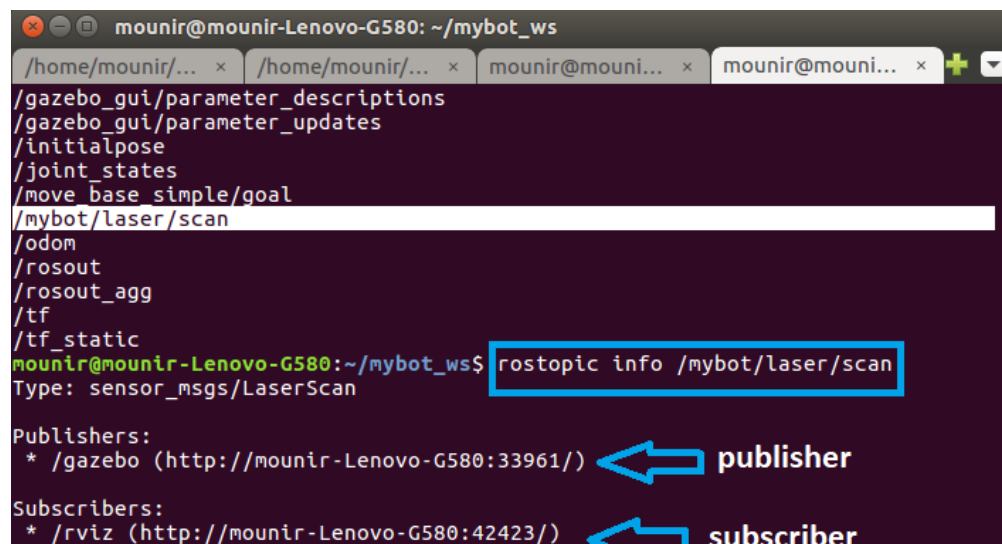
```
mounir@mounir-Lenovo-G580: ~/mybot_ws
rostopic list
/clicked_point
/clock
/cmd_vel
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/initialpose
/joint_states
/move_base_simple/goal
/mybot/laser/scan
/odom
/rosout
/rosout_agg
/tf
/tf_static
```

Figure IV.69 Laser Topic.

To check the laser topic, we need to run this command to see who's the publisher and the subscriber.

```
rostopic info topic_name
```

You must choose just one topic, like in **figure IV.69**.



```
mounir@mounir-Lenovo-G580: ~/mybot_ws
rostopic info /mybot/laser/scan
Type: sensor_msgs/LaserScan

Publishers:
* /gazebo (http://mounir-Lenovo-G580:33961/)

Subscribers:
* /rviz (http://mounir-Lenovo-G580:42423/)
```

Figure IV.70 Publisher and Subscriber Information Of /Mybot/Laser/Scan Topic.

Publisher:

Gazebo will share the information through odom through sensor_msgs/laserScan even when you open the rviz we said that you must select the odom to receive our robot state (**figure IV.55**) and to receive the laser scan we must select the laser topic from the laser scan module like in (**figure IV.60**).

Subscriber:

Rviz will subscribe to gazebo to bring all information through the node and show it in virtual world.

IV.8 Joints of Our Robot:

This following figure explains what joints are:

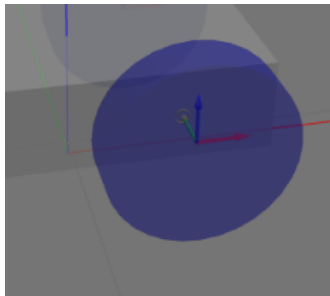


Figure IV.71 Wheel Joint.

Joints are the axes of the wheels (x y z) and we can find these parameters exist in our code to place it and attach it with chassis like in (**figure IV.50**).

IV.9 Test Our Robot (Mapping and localization):

IV.9.1 Mapping:

Now we are going to implement our mapping algorithms.

To our precedent file (work space file) we are going to add certain files to import the g-mapping algorithms like the following paths

- ❖ Mybot_ws/src/mybot_control/config/mybot_control.yaml (**figure IV.72**)
 - ❖ Mybot_ws/src/mybot_control/launch/mybot_control.launch (**figure IV.73**)
 - ❖ Mybot_ws/src/mybot_navigation/launch/amcl_demo.launch (**figure IV.76**)
 - ❖ Mybot_ws/src/mybot_navigation/launch/gmapping_demo.launch
 - ❖ Mybot_ws/src/mybot_navigation/launch/mybot_teleop.launch (**figure IV.75**)
 - ❖ Mybot_ws/src/mybot_navigation/config/base_local_planner_params.yaml
 - ❖ Mybot_ws/src/mybot_navigation/config/costmap_common_params.yaml
 - ❖ Mybot_ws/src/mybot_navigation/config/global_costmap_params.yaml
 - ❖ Mybot_ws/src/mybot_navigation/config/local_costmap_params.yaml
- The config file is displayed in (**figure IV.74**)
- ❖ Mybot_ws/src/mybot_description/launch/mybot_rviz_amcl.launch
 - ❖ Mybot_ws/src/mybot_description/launch/mybot_rviz_gmapping.launch (**figure IV.77**)

IV.9.2 Code the control file:

We must make our robot move around our 3D gazebo surface to do that we must install one of the tools which is the **teleop_twist** and we must make sure that the differential drive controller (**diff_drive_controller kinetic compatible**) is installed. So, our robot can move left, right, backward and forward.

```
C:\Users\houho\Desktop\project mounir\gmapping_navigate\mybot_ws\src\mybot_control\config> ! mybot_control.yaml
1 myrobot:
2   # Publish all joint states -----
3   joint_state_controller:
4     type: joint_state_controller/JointStateController
5     publish_rate: 50
6
7   mobile_base_controller:
8     type: "diff_drive_controller/DiffDriveController" ## this is the tools of diff drive controller compatibel ros kinetic
9     left_wheel: 'left_wheel_hinge' ##instal this controll with wheels
10    right_wheel: 'right_wheel_hinge'
11    pose_covariance_diagonal: [0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
12    twist_covariance_diagonal: [0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
13
14
```

Figure IV.72 ControlFile Configuration.

```
C:\Users\houho\Desktop\project mounir\gmapping_navigate\mybot_ws\src\mybot_control\launch> myrobot_control.launch
1 <?xml version="1.0" encoding="UTF-8"?>
2 <launch>
3
4   <!-- Load joint controller configurations from YAML file to parameter server -->
5   <roscppparam file="$(find mybot_control)/config/mybot_control.yaml" command="load"/> <!--including the config file -->
6
7   <!-- load the controllers -->
8   <!--create a node for this controller -->
9   <node name="controller_spawner"
10     pkg="controller_manager"
11     type="spawner" respawn="false"
12     output="screen" ns="/mybot"
13     args="joint_state_controller
14         mobile_base_controller"
15   />
16
17   <!-- convert joint states to TF transforms for rviz, etc -->
18   <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false" output="screen">
19     <param name="mybot_description" command="$(find xacro)/xacro.py '$(find mybot_description)/urdf/mybot.xacro'"/>
20     <remap from="/joint_states" to="/mybot/joint_states" />
21   </node>
22
23 </launch>
```

Figure IV.73 Launch File Of The Controller.

These two files are to make sure that our robot receive commands from the differential drive and enable us to control it by launching the teleop_twist.

```

! base_local_planner_params.yaml X
act mourir > gmapping_navigate > mybot_ws > src > mybot_navigation > config >
1 TrajectoryPlannerROS:
2   max_vel_x: 0.5
3   min_vel_x: 0.01
4   max_vel_theta: 1.5
5   min_in_place_vel_theta: 0.01
6
7   acc_lim_theta: 1.5
8   acc_lim_x: 0.5
9   acc_lim_y: 0.5
10
11
12   holonomic_robot: false
13

! costmap_common_params.yaml X ! mybot_control.yaml myrobot_controllaunch
C:\Users\houho\Desktop> project mourir > gmapping_navigate > mybot_ws > src > mybot_navigation > config > ! costmap_com
1 obstacle_range: 2.5
2 raytrace_range: 3.0
3 #footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
4 #robot_radius: ir_of_robot
5 robot_radius: 0.5 # distance a circular robot should be clear of the obstacle
6 inflation_radius: 3.0
7
8 observation_sources: laser_scan_sensor #point_cloud_sensor
9
10 # marking - add obstacle information to cost map
11 # clearing - clear obstacle information to cost map
12 laser_scan_sensor: {sensor_frame: hokuyo, data_type: LaserScan, topic: /mybot/laser/scan, marking
13
14 #point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, marking
15

! global_costmap_params.yaml X
C:\Users\houho\Desktop> project mourir > gmapping_navigate > mybot_
1 # static_map - True if using existing map
2
3 global_costmap:
4   global_frame: odom
5   robot_base_frame: chassis
6   update_frequency: 1.0
7   publish_frequency: 1.0
8   resolution: 0.05
9   static_map: true
10  width: 10.0
11  height: 10.0
12

! local_costmap_params.yaml X
C:\Users\houho\Desktop> project mourir > gmapping_navigate > mybot_ws > src > mybot_navigation > config > ! local_costmap_par
1 local_costmap:
2   global_frame: odom
3   robot_base_frame: chassis
4   update_frequency: 2.0
5   publish_frequency: 1.0
6   static_map: false
7   rolling_window: true
8   width: 6.0
9   height: 6.0
10  resolution: 0.05
11

```

Figure IV.74 Navigation Package Config File.

The navigation package is connected with control file. It specifies the parameters like max speed angle axes like in (**figure IV.74**).

figure IV.74 Part1

setting the speed and angles.

figure IV.74 Part2

setting the obstacle ranges.

Put seeing view (from laser scan sensor)

Config the laser scan sensor such as name, type of data and most important which topic getting the data from (**/mybot/laser/scan topic**).

figure IV.74 Part (3—4): this part of the picture is for the map configuration.

IV.9.3 The Code Of Navigation Package:

```

mybot_teleop.launch X
C: > Users > houho > Desktop > project mounir > gmaping_navigate > mybot_ws > src > mybot_navigation > launch > mybot_teleop.launch
1 <?xml version="1.0"?>
2 <launch>
3 <!-- turtlebot_teleop_key already has its own built in velocity smoother -->
4 <node pkg="turtlebot_teleop" type="turtlebot_teleop_key" name="turtlebot_teleop_keyboard" output="screen">
5   <param name="scale_linear" value="0.5" type="double"/>
6   <param name="scale_angular" value="1.5" type="double"/>
7   <remap from="turtlebot_teleop_keyboard/cmd_vel" to="cmd_vel"/>
8 </node>
9 </launch>

gmapping_demo.launch X  amcl_demo.launch
C: > Users > houho > Desktop > project mounir > gmaping_navigate > mybot_ws > src > mybot_navigation > launch > gmapping_demo.launch
1 <?xml version="1.0"?>
2 <launch>
3   <master auto="start"/>
4   <param name="/use_sim_time" value="true"/>
5   <!-- Run gmapping -->
6   <node pkg="gmapping" name="slam_gmapping" type="slam_gmapping" output="screen">
7     <param name="delta" value="0.01"/>
8     <param name="xmin" value="-20"/>
9     <param name="xmax" value="20"/>
10    <param name="ymin" value="-20"/>
11    <param name="ymax" value="20"/>
12    <remap from="scan" to="mybot/laser/scan"/>
13    <param name="base_frame" value="chassis" />
14
15    <param name="linearUpdate" value="0.5"/>
16    <param name="angularUpdate" value="0.436"/>
17    <param name="temporalUpdate" value="-1.0"/>
18    <param name="resampleThreshold" value="0.5"/>
19    <param name="particles" value="80"/>
20
21  </node>
22
23
24 </launch>

```

Figure IV.75 mybot_teleop.launch-----gmapping_demmo.launch Files.

Top file- figure IV.74:

Is correspondent for launching the teleop control file A node of this package turtlebot_teleop_keyboard publishes a topic called cmd_vel that represents velocity commands for the robot. **Like in line 7 from the code.**

gmaping file-figure IV.74:

It's a launch file who call and launch gmapping algorithm from our tools ROS repositories and set some important parameters. Like importing the data from the mybot/laser/scan topic and send all the gmapping data from a node called "slam_gmapping"

```

C:\Users\houho > Desktop > project mounir > gmapping_navigate > mybot_ws > src > mybot_navigation > launch > amcl_demo.launch
1  <?xml version="1.0"?>
2  <launch>
3    <master auto="start"/>
4
5    <!-- Map server -->
6    <arg name="map_file" default="$(find mybot_navigation)/maps/test1_map.yaml"/>
7    <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />
8
9    <!-- Place map frame at odometry frame -->
10   <node pkg="tf" type="static_transform_publisher" name="map_odom_broadcaster"
11     args="0 0 0 0 0 0 map_odom 100"/>
12
13   <!-- Localization -->
14   <node pkg="amcl" type="amcl" name="amcl" output="screen">
15     <remap from="scan" to="mybot/laser/scan"/>
16     <param name="odom_frame_id" value="odom"/>
17     <param name="odom_model_type" value="diff-corrected"/>
18     <param name="base_frame_id" value="chassis"/>
19     <param name="update_min_d" value="0.5"/>
20     <param name="update_min_a" value="1.0"/>
21   </node>
22   <!--include file="$(find amcl)/examples/amcl_omni.launch"/-->
23
24
25
26   <!-- Move base -->
27   <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
28     <rosparam file="$(find mybot_navigation)/config/costmap_common_params.yaml" command="load" ns="global_costmap" />
29     <rosparam file="$(find mybot_navigation)/config/costmap_common_params.yaml" command="load" ns="local_costmap" />
30     <rosparam file="$(find mybot_navigation)/config/local_costmap_params.yaml" command="load" />
31     <rosparam file="$(find mybot_navigation)/config/global_costmap_params.yaml" command="load" />
32     <rosparam file="$(find mybot_navigation)/config/base_local_planner_params.yaml" command="load" />
33
34     <remap from="cmd_vel" to="cmd_vel"/>
35     <remap from="odom" to="odom"/>
36     <remap from="scan" to="mybot/laser/scan"/>
37     <param name="move_base/DWAPlannerROS/yaw_goal_tolerance" value="1.0"/>
38     <param name="move_base/DWAPlannerROS/xy_goal_tolerance" value="1.0"/>
39
40
41
42   </node>
43
44
45 </launch>

```

Figure IV.76 Map Loader File.

When we launch the gmapping file to create a map after the map complete, we can save the map with the following command

```
rosrun map_server map_saver -f ~/mybot_ws/src/mybot_navigation/maps/test1_map
```

you will get two files:

```
test1_map.pgm
```

test1_map.yaml

To visual the results we must add rviz files to description package so we can see how the map will generate

```

C: > Users > houho > Desktop > project mounir > gmapping_navigate > mybot_ws > src > mybot_description > launch > mybot_rviz_amcl.launch
1  [ ]>xml version="1.0">[ ]
2  <launch>
3
4      <param name="robot_description" command="$(find xacro)/xacro.py '$(find mybot_description)/urdf/mybot.xacro'"/>
5
6      <!-- send fake joint values -->
7      <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
8          <param name="use_gui" value="False"/>
9      </node>
10
11     <!-- Combine joint values -->
12     <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
13
14     <!-- Show in Rviz -->
15     <node name="rviz" pkg="rviz" type="rviz" args="-d $(find mybot_description)/rviz/amcl.rviz"/>
16
17 </launch>
18
mybot_rviz_gmapping.launch X
C: > Users > houho > Desktop > project mounir > gmapping_navigate > mybot_ws > src > mybot_description > launch > mybot_rviz_gmapping.launch
1  [ ]>xml version="1.0">[ ]
2  <launch>
3
4      <param name="robot_description" command="$(find xacro)/xacro.py '$(find mybot_description)/urdf/mybot.xacro'"/>
5
6      <!-- send fake joint values -->
7      <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
8          <param name="use_gui" value="False"/>
9      </node>
10
11     <!-- Combine joint values -->
12     <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
13
14     <!-- Show in Rviz -->
15     <node name="rviz" pkg="rviz" type="rviz" args="-d $(find mybot_description)/rviz/mapping.rviz"/>
16     <!--node name="rviz" pkg="rviz" type="rviz" args="-d $(find mybot_description)/launch/myrobot.rviz"/-->
17 </launch>
18

```

Figure IV.77 Rviz Extra Files.

The top file of (figure IV.77) (mybot_rviz_amcl.launch) is to visual the generated map and see some navigation autonomously.

In the other file of (**figure IV.77**) (**mybot_rviz_gmapping.launch**) to visual the map while is been created by our gmapping algorithm

IV.9.4 Start seeing results

After completing the coding, catkin_make again your work space and source the file so that can be accessed to ros tools and packages

```
mounir@mounir-Lenovo-G580:~/mybot_ws$ ./run_gazebo.sh
[sudo] password for mounir:
rosmaster: no process found
gzserver: no process found
gzclient: no process found
... logging to /home/mounir/.ros/log/9f35cac6-e3e7-11ea-976e-3c970e84092d/roslaunch-mounir-Lenovo-G580-10737.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: Traditional processing is deprecated. Switch to --inorder processing!
To check for compatibility of your document, use option --check-order.
For more infos, see http://wiki.ros.org/xacro#Processing_Order
xacro.py is deprecated; please use xacro instead
started roslaunch server http://mounir-Lenovo-G580:38897/

SUMMARY
=====
PARAMETERS
* /robot_description: <?xml version="1...
* /roslistro: kinetic
* /rosversion: 1.12.14
* /use_sim_time: True

NODES
/
  gazebo (gazebo_ros/gzserver)
  gazebo_gui (gazebo_ros/gzclient)
  mybot_spawn (gazebo_ros/spawn_model)

auto-starting new master
process[master]: started with pid [10750]
```

Figure IV.78 Launching Gazebo With Our Robot Module With Room full with sphere And A Boxes.

❖ Launching gazebo:

After launching the gazebo, I did create a room with a box inside that our robot doesn't know and seen it before (**figure IV.79**)

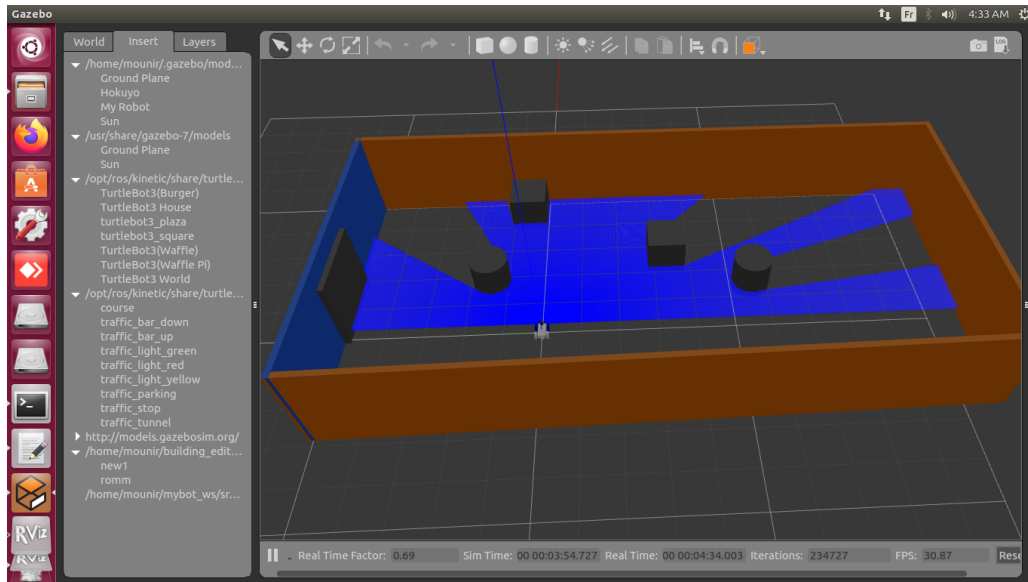


Figure IV.79 Our Robot Module Inside A Room with A Box.

```

/home/mounir/mybot_ws/src/mybot_navigation/launch/gmapping_demo.launch http://localhost:11311
mounir@mounir-Lenovo-G580:~/mybot_ws$ roslaunch mybot_navigation gmapping_demo.launch
... logging to /home/mounir/.ros/log/9f35cac6-e3e7-11ea-976e-3c970e84092d/roslaunch-mounir-Lenovo-G580-11340.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

WARNING: ignoring defunct <master /> tag
started roslaunch server http://mounir-Lenovo-G580:36825/

SUMMARY
=====
PARAMETERS
* /roscdistro: kinetic
* /rosversion: 1.12.14
* /slam_gmapping/angularUpdate: 0.436
* /slam_gmapping/base_frame: chassis
* /slam_gmapping/delta: 0.01
* /slam_gmapping/linearUpdate: 0.5
* /slam_gmapping/particles: 80
* /slam_gmapping/resampleThreshold: 0.5
* /slam_gmapping/temporalUpdate: -1.0
* /slam_gmapping/xmax: 20
* /slam_gmapping/xmin: -20
* /slam_gmapping/ymax: 20
* /slam_gmapping/ymin: -20
* /use_sim_time: True

NODES
/
  slam_gmapping (gmapping/slam_gmapping)

ROS_MASTER_URI=http://localhost:11311
process[slam_gmapping-1]: started with pid [11357]
  
```

Figure IV.80 Terminal Window with Gmapping Algorithm.

After launching this command, the gmapping will receive the data from the sensor

❖ Launching rviz to see the map:

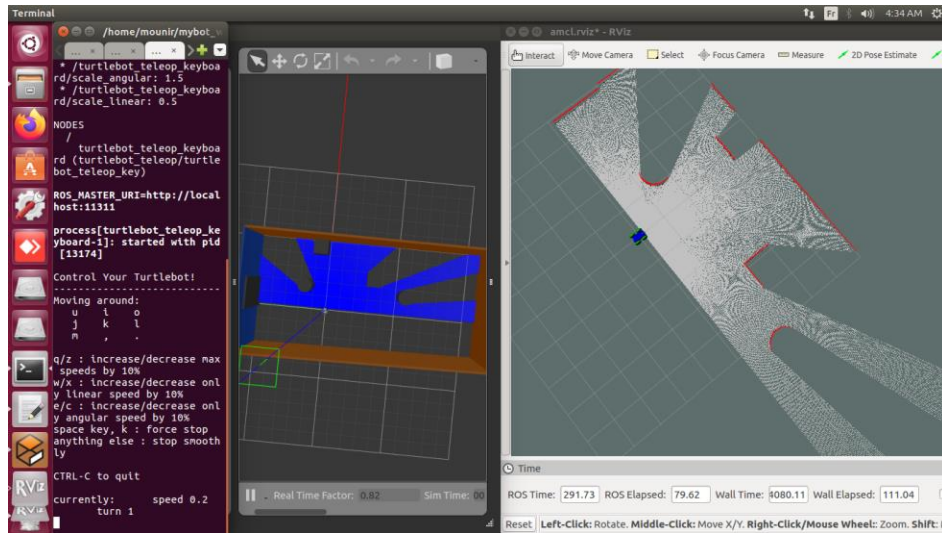


Figure IV.81 Rviz With Mapping Results.

You must add the robot model and the map to see how the map will generate when you move your robot

❖ Launch the teleop_controller:

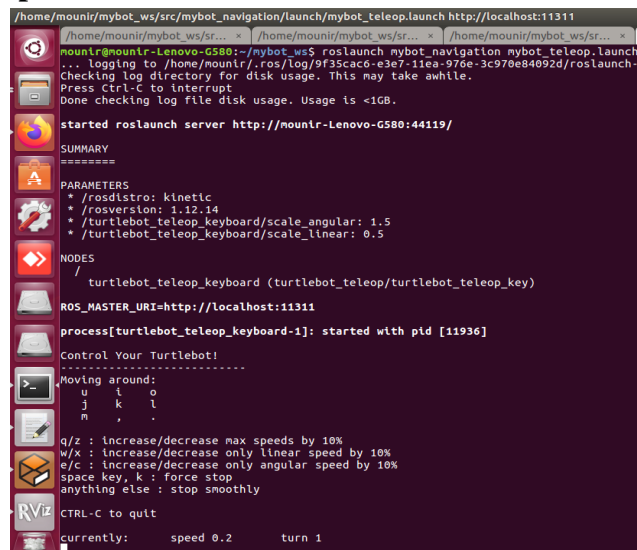


Figure IV.82 The Robot Controller Keys.

“i”: forward. “j”: backward. “l”: right. “k”: left. “k”: stop.

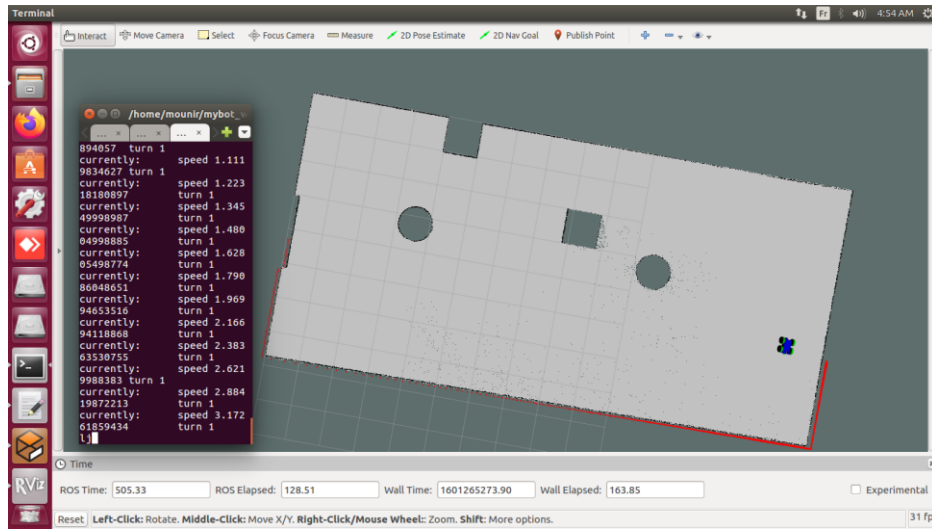


Figure IV.83 After Moving through all obstacles.

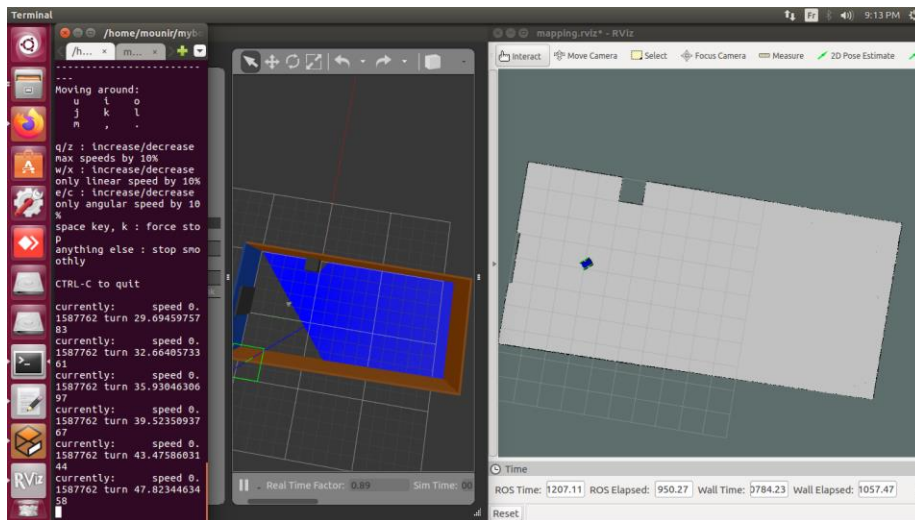


Figure IV.84 first Map sample without obstacles.

After mapping all the surrounding time to register the map into the file that exist in navigation packages with the name of maps

```

/home/mounir/mybot_ws/src... x /home/mounir/mybot_ws/src... x /home/mounir/mybot_ws/src... x /home/mounir/mybot_ws/src... x mounir@mounir-Lenovo-G5... x
mounir@mounir-Lenovo-G580:~/mybot_ws$ roslaunch map_server map_saver -f ~/mybot_ws/src/mybot_navigation/maps/test1_map
[ INFO] [1598040828.216162500]: Waiting for the map
[ INFO] [1598040828.457480034, 1247.492000000]: Received a 4000 X 4000 map @ 0.010 m/pix
[ INFO] [1598040828.457715765, 1247.492000000]: Writing map occupancy data to /home/mounir/mybot_ws/src/mybot_navigation/maps/test1_map.pgm
[ INFO] [1598040828.958617445, 1247.841000000]: Writing map occupancy data to /home/mounir/mybot_ws/src/mybot_navigation/maps/test1_map.yaml
[ INFO] [1598040828.959490829, 1247.841000000]: Done
mounir@mounir-Lenovo-G580:~/mybot_ws$

```

Figure IV.85 Saving the Map Under the Name Test1_Map.

Now the map is completed, so we can close the Gmapping Terminal window and we can take a closer look on our map by opening the.

Before you see the results and closing the terminals, we can see that the topic list shows the data from where is being taking from (slam_gmapping) (**figure IV.88**).

```

/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/initialpose
/joint_states
/map
/map_metadata
/map_updates
/move_base_simple/goal
/mybot/laser/scan
/odom
/rosout
/rosout_agg
/slam_gmapping/entropy
/tf
/tf_static
mounir@mounir-Lenovo-G580:~/mybot_ws$
mounir@mounir-Lenovo-G580:~/mybot_ws$ rostopic info /slam_gmapping/entropy
Type: std_msgs/Float64

Publishers:
 * /slam_gmapping (http://mounir-Lenovo-G580:40461/)

Subscribers: None

mounir@mounir-Lenovo-G580:~/mybot_ws$ rostopic info /mybot/laser/scan
Type: sensor_msgs/LaserScan

Publishers:
 * /gazebo (http://mounir-Lenovo-G580:42865/)

Subscribers:
 * /slam_gmapping (http://mounir-Lenovo-G580:40461/)

mounir@mounir-Lenovo-G580:~/mybot_ws$

```

Figure IV.86 Rostopic List.

We can see that slam_gmapping node is subscribing to the laser scan topic.

IV.9.5 Navigation:

In this part we are going to make sure that we add in a navigation and description file some extra files

After all we must run the gazebo file with the 3D robot module

```
./run_gazebo.sh
```

and after it make sure we run the next two files

➤ Mybot_ws/src/mybot_navigation/launch/amcl_demo.launch (**figure IV.76**)

This file is correspondent for loading the map that we did register before under the name test1_map with the next command (**figure IV.85**).

```
roslaunch mybot_navigation amcl_demo.launch
```

this two first steps the gazebo file that you did launch must be the same world that we did map so we can navigate through it.

➤ Mybot_ws/src/mybot_description/launch/mybot_rviz_amcl.launch (**figure IV.77**)

This file is correspondent for loading the rviz application so we visual the map, our robot, the path that the robot is going for.

Results:

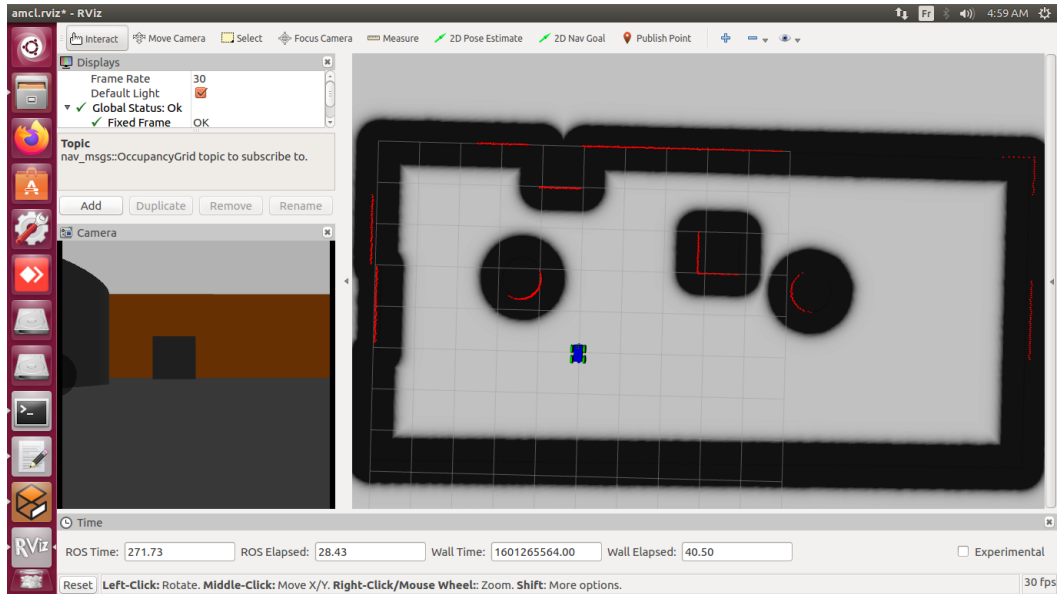


Figure IV.87 Loading The Map Of Navigation In Rviz.

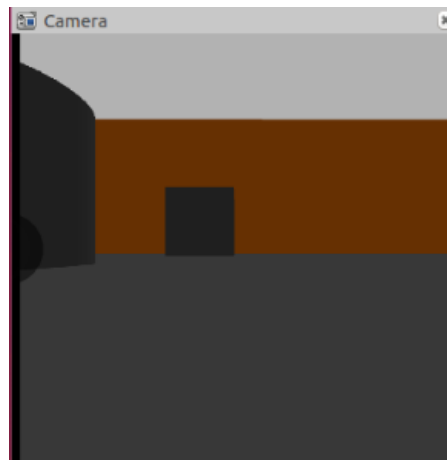


Figure IV.88 Camera on Our Robot.

We did add a camera plugin in our robot gazebo plug in file to visual our surrounding.

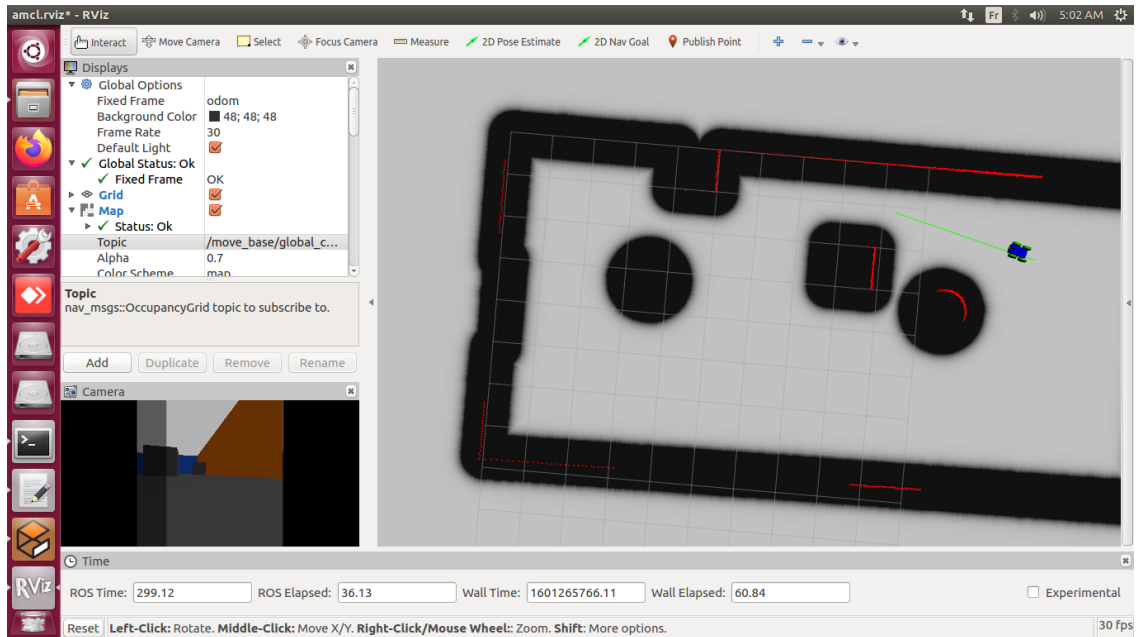


Figure IV.89 Robot While Moving to The Target.

To give the robot a target to move forward we must pick the green arrow in the rviz application and point to the direction that we want and the robot will move automatically to that target (position).

Problems:

- In this simulation we got a problem with the robot while his moving through ground which is being too slow to move. Only the reasons I found is Real Time Factor on the gazebo application it supposed to be 1 and is down to 0.71 (left bottom in **figure IV.90**) even while we were doing the mapping algorithms when we controlled the robot with teleop_twist (**figure IV.82**) the robot was too slow to turn left and right we did manage to figure out the problem will be shown in solution part.

- Having strong computer equipment get you more close to reality simulation.
- We only try to make the robot go toward and it get stuck on it.

Solution:

•To make the robot move easier we must change the mechanism control system we did install the differential drive control which supposed to our robot will only be controlled by two wheels in our case our robot has four wheels so we have to install other robot driver “lib_gazebo_ros_planar_move” to fix the slow driving movement.

```
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_planar_move.so">
    <legacyMode>false</legacyMode>
  </plugin>
</gazebo>
```

Figure IV.90 lib_gazebo_ros_planar_move robot four wheels driver

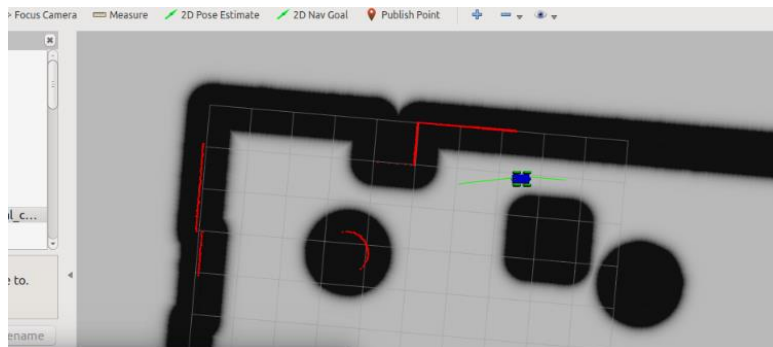


Figure IV.91 The Robot While Taking A Turn through the obstacle .

IV.10 Measure distances in maps

In this part of work were going to try to measure distances from our generated map to do that we chose to work on OpenCV language. The idea here is to try to work on image analyzing detecting contours and shapes (square, rectangles, circles) from the generated map and in the last we calculate the distances for each shape inside our generated map from our robot.

First step is installing the right version of OpenCV2 python, there is deferent type of this new language for example if you tried to compile a code who use some tools from OpenCV language it will show you an error that means you're missing the right version for the right python version from your Linux operation system.

The next step will be after preparing our environment is to create a package who's going to hold our codes parameters.

IV.10.1 Creating our package:

To createa package you need a command like following:

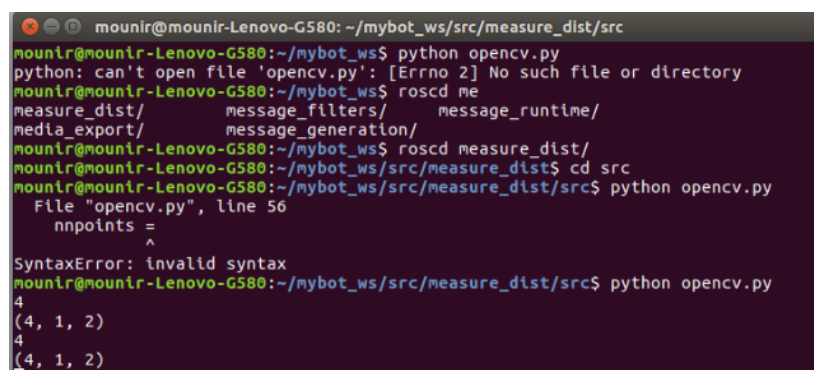
-Using the `catkin_create_pkg` command line in the Terminal to create a package under the name `measure_dis`.

-Inside the `src` file create a python file under the name `opencv.py` to hold our code.

-Make the last file executable.

-Start coding.

-Compiling the file through the Terminal typing `python opencv.py`.



```
mounir@mounir-Lenovo-G580: ~/mybot_ws/src/measure_dist/src
mounir@mounir-Lenovo-G580:~/mybot_ws$ python opencv.py
python: can't open file 'opencv.py': [Errno 2] No such file or directory
mounir@mounir-Lenovo-G580:~/mybot_ws$ roscd me
measure_dist/  message_filters/  message_runtime/
media_export/  message_generation/
mounir@mounir-Lenovo-G580:~/mybot_ws$ roscd measure_dist/
mounir@mounir-Lenovo-G580:~/mybot_ws/src/measure_dist$ cd src
mounir@mounir-Lenovo-G580:~/mybot_ws/src/measure_dist/src$ python opencv.py
File "opencv.py", line 56
    nnpoints =
              ^
SyntaxError: invalid syntax
mounir@mounir-Lenovo-G580:~/mybot_ws/src/measure_dist/src$ python opencv.py
4
(4, 1, 2)
4
(4, 1, 2)
```

Figure IV.92 Compiling Our File.

IV.10.2 Results:

➤ Detecting contours on our map

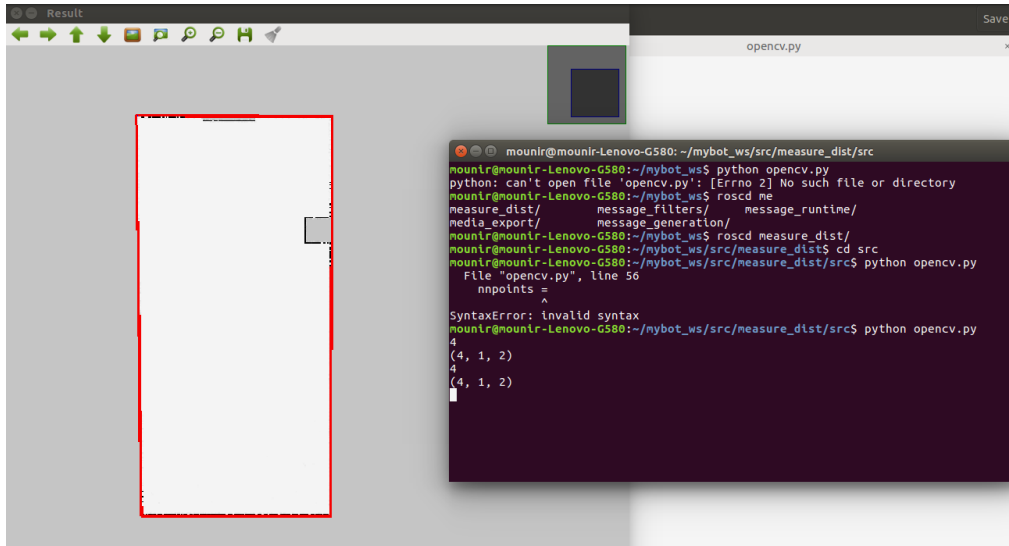


Figure IV.93 Contours That Exist on Our first map Picture.

➤ Get only big contour and zoom in

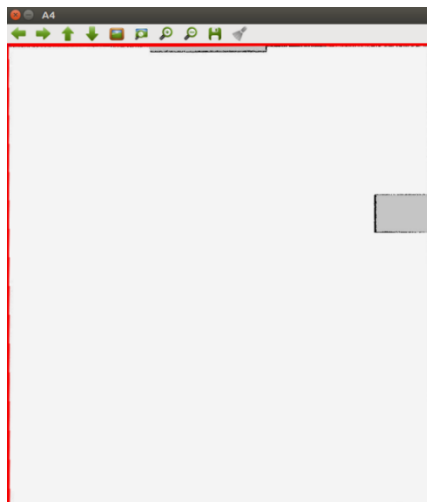


Figure IV.94 Zoom The Map.

The idea here is give the map a starting distance which is long and width from those starting points we can measure anything inside the biggest contour (our map).

IV.11 Build Our Robot on Reality:

To build our robot we need the next equipment's:

- Chassis of our robot.
- Wheels and DC motors.
- Control unity (raspberry pi 3 b + module).
- Sensor (RPLIDAR A1).
- Power supply.
- Connecting wires.
- Mouse and keyboard (to navigate throw Ubuntu).

IV.11.1 Preparing the Raspberry Pi:

To do the same work as the simulation we need to import ROS to our Raspberry pi 3 B+ module with it we can compile and execute some gmapping algorithms

Installing ubuntu version who has an integrated ROS version inside it (Raspberry pi Image Ubiquity Robotics).

In this version is exactly the same procedure from working with Terminal command to creating a work space and compile it in Linux.

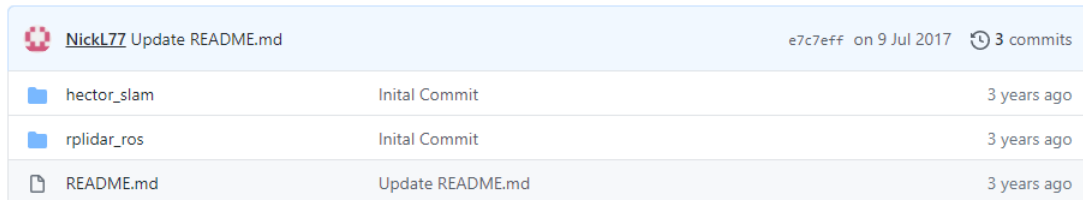
Raspberry pi Image Ubiquity Robotics= ubuntu 16.04 + ROS integrated with in.



Figure IV.95 Ubuntu 16.04 Raspberry Compatible.

IV.11.2 Test the sensor:

Get Clone RPLIDAR A1 repository from GitHub



Commit Message	Author	Date	Commits
NickL77 Update README.md	e7c7eff	on 9 Jul 2017	3 commits
hector_slam	Initial Commit	3 years ago	
rplidar_ros	Initial Commit	3 years ago	
README.md	Update README.md	3 years ago	

Figure IV.96 RPLIDAR And Hector Slam Repositories.

IV.11.3 Import hector slam:

Get Clone hector SLAM repository from GitHub which include all tools that we need do mapping and localization with our Real robot and sensor

IV.11.4 Compile the Files:

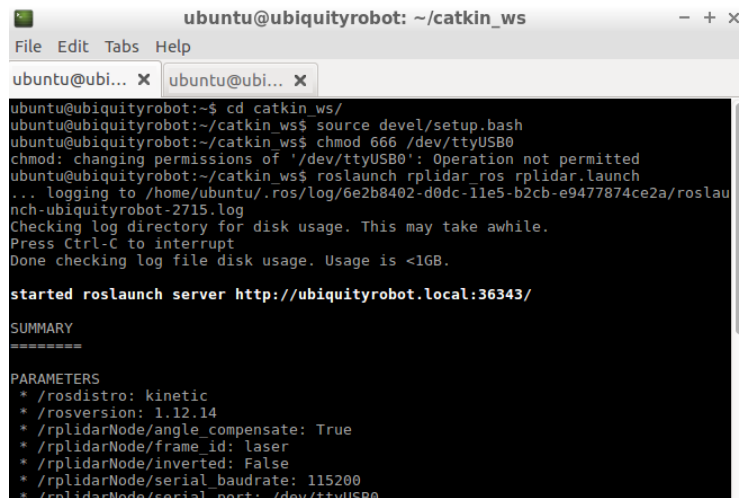
-When we did compile all files with our Raspberry pi 3 b + module it got stock because due to the lake of RAM (1GB) and with the size of our file we don't have enough RAM to compile it.

Solutions:

- Upgrade to the new Raspberry pi 4 version with 4 GB of RAM.
- Compiling done without any problem.

IV.11.5 Run Mapping Algorithm:

o Plug the RPLIDAR first with the Serial port converter with an USB cable to the Raspberry pi 4 USB port and run the following command lines.



```

ubuntu@ubiquityrobot: ~/catkin_ws
File Edit Tabs Help
ubuntu@ubi... x ubuntu@ubi... x
ubuntu@ubiquityrobot:~$ cd catkin_ws/
ubuntu@ubiquityrobot:~/catkin_ws$ source devel/setup.bash
ubuntu@ubiquityrobot:~/catkin_ws$ chmod 666 /dev/ttyUSB0
chmod: changing permissions of '/dev/ttyUSB0': Operation not permitted
ubuntu@ubiquityrobot:~/catkin_ws$ roslaunch rplidar_ros rplidar.launch
... logging to /home/ubuntu/.ros/log/6e2b8402-d0dc-11e5-b2cb-e9477874ce2a/roslau
nch-ubiquityrobot-2715.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubiquityrobot.local:36343/

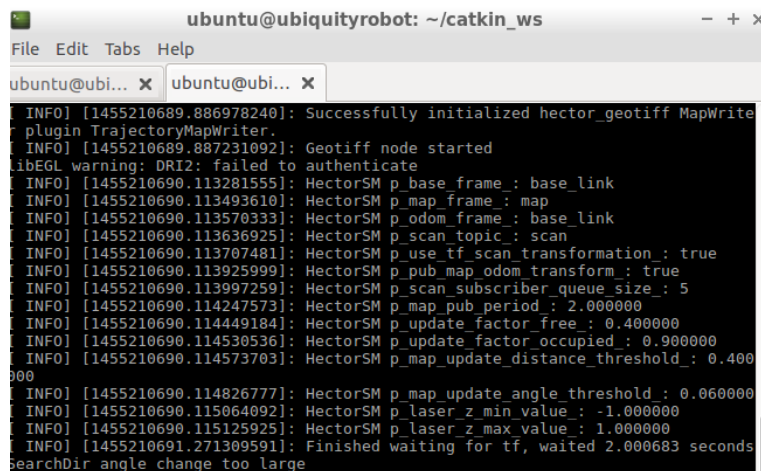
SUMMARY
=====

PARAMETERS
* /roscdistro: kinetic
* /rosversion: 1.12.14
* /rplidarNode/angle_compensate: True
* /rplidarNode/frame_id: laser
* /rplidarNode/inverted: False
* /rplidarNode/serial_baudrate: 115200
* /rplidarNode/serial_port: /dev/ttyUSB0

```

Figure IV.97 Launch Rplidar Package.

- o Run the hector slam algorithm



```

ubuntu@ubiquityrobot: ~/catkin_ws
File Edit Tabs Help
ubuntu@ubi... x ubuntu@ubi... x
INFO [1455210689.886978240]: Successfully initialized hector_geotiff MapWrite
plugin TrajectoryMapWriter.
INFO [1455210689.887231092]: Geotiff node started
libEGL warning: DRI2: failed to authenticate
INFO [1455210690.113281555]: HectorSM p_base_frame_ : base_link
INFO [1455210690.113493610]: HectorSM p_map_frame_ : map
INFO [1455210690.113570333]: HectorSM p_odom_frame_ : base_link
INFO [1455210690.113636925]: HectorSM p_scan_topic_ : scan
INFO [1455210690.113707481]: HectorSM p_use_tf_scan_transformation_ : true
INFO [1455210690.113925999]: HectorSM p_pub_map_odom_transform_ : true
INFO [1455210690.113997259]: HectorSM p_scan_subscriber_queue_size_ : 5
INFO [1455210690.114247573]: HectorSM p_map_pub_period_ : 2.000000
INFO [1455210690.114449184]: HectorSM p_update_factor_free_ : 0.400000
INFO [1455210690.114530536]: HectorSM p_update_factor_occupied_ : 0.900000
INFO [1455210690.114573703]: HectorSM p_map_update_distance_threshold_ : 0.400
000
INFO [1455210690.114826777]: HectorSM p_map_update_angle_threshold_ : 0.060000
INFO [1455210690.115064092]: HectorSM p_laser_z_min_value_ : -1.000000
INFO [1455210690.115125925]: HectorSM p_laser_z_max_value_ : 1.000000
INFO [1455210691.271309591]: Finished waiting for tf, waited 2.000683 seconds
searchDir angle change too large

```

Figure IV.98 Launch the Hector Slam Package.

After running the hector mapping package rviz will show up some parameters.

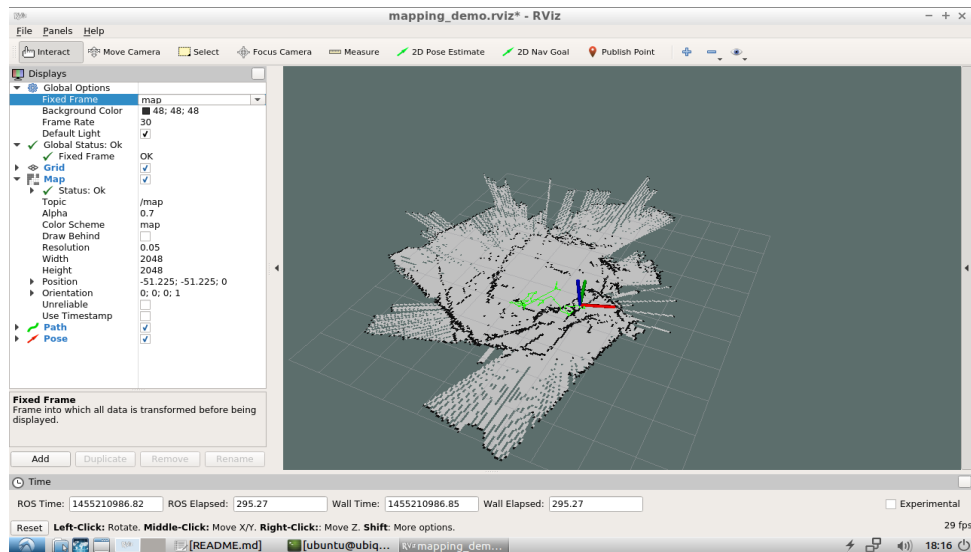


Figure IV.99 Results In Rviz Application After Launching Hecto Slam (Mapping).

The blue green red axe represents the sensor and the green line represent the path that our sensor did goes through

-The map is not well generating.

-Second problem is when you move your laser to much the map will be moved around like (Figure IV.101).

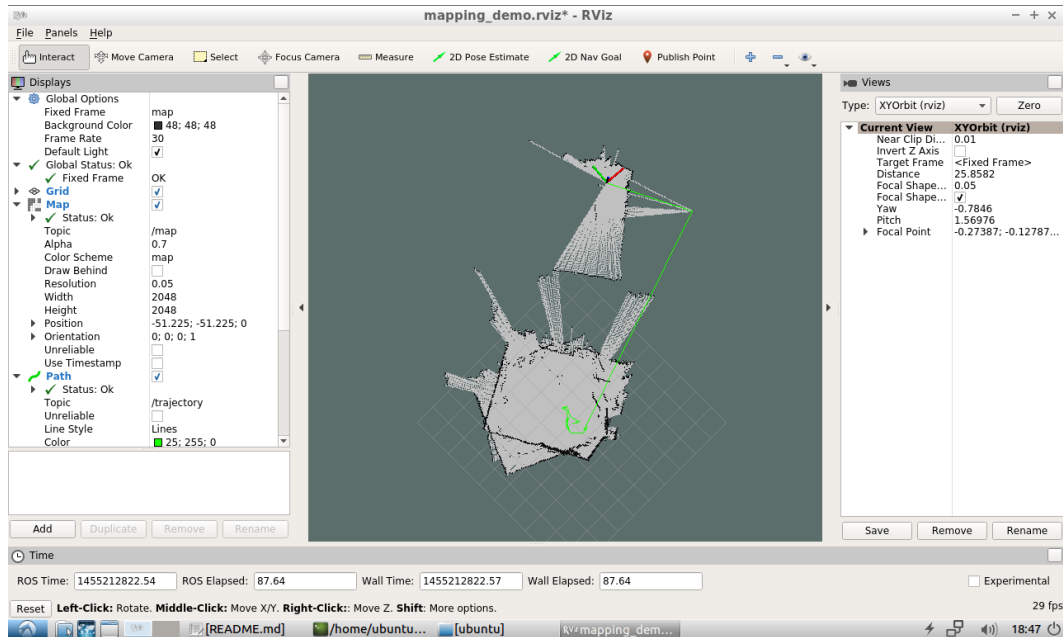


Figure IV.100 Second Try Launching Hecctor Slam Algorithm Shown in Rviz.

Solutions:

-First you need flat surface because in the code we determined that the robot space is flat.

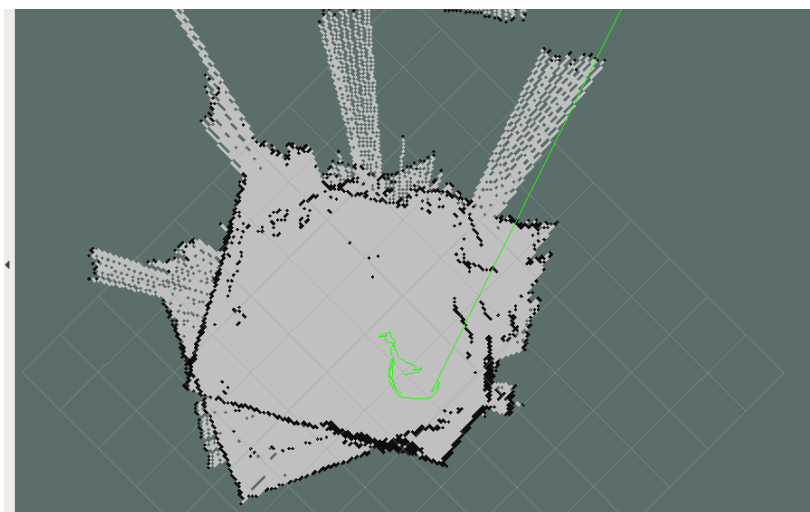


Figure IV.101 Results After Solving the Problem.

In this third try we looking at the room map is being almost generated (**figure IV.103**) but when we did move the laser the map image got ruined

IV.12 Conclusion:

After building our robot in 3D world and in reality, successfully we did generate a map with it and save it. Also managed to control all system using nodes and topic and figure out how Linux work and how much problem to install all tools like robot module, Gazebo, Rviz, and with a lot of bugs in the system specially during compiling files. Without forgetting about how much our system requirement is low to get a close real time simulation but we manage to do Mapping and navigation. Also, we come up with an idea of measure the map shapes from a starting point using OpenCV language and some python assistance but unfortunately it's not completely done.

General Conclusion

General Conclusion

After all, every day the science is jumping a step forward in electronics and programming due to their importance and matter in our style life to make it easier. This advantage is making us to give more potential and hard work to provide new way to the next generation each move consider like mine of gold. Humanins are smart to create artificial intelligent and integrate it inside robots so it can act like us and think like us and predict our steps into the future and improve it.

The objective of our research project not completely achieved, but the main goal from this project was done, which is mapping and small navigation step. Through developing a 3D robot inside 3D environment, with a short in equipment we achieved that goal. And we did manage to aim on how this project was build and pass through some hard steps like libraries installation errors and multiple bugs in Linux system and dealing with Robotic Operation System. However, all steps in the simulation and results was done in real application except building the robot, that we suppose to build the real robot with 4 DC motors and an expensive Sensor RPLIDAR A1 and the newest version of raspberry pi 4 we did manage to compile the project and got some results of just building a map.

In this work the most difficult thing is to install and prepare for your ROS system because there will be a lot of errors and bugs in the system (Linux) so when you face something you must search online to see all solution possible. To build your robot you need to learn some coding Princip, and the best advice it's trying to compile easy things like we did in the last chapter.

As a small project is really hard to study all things all at once but you can learn a lot of things from it like python, C++, OpenCV, Robotic Operation System, URDF and SDF files Nodes and Topics all of this will leads you to having great skills but it's a risky way to march on, the perspectives of this work is to know for example the deference between working with MATLAB and with real robot environment to work with tools can be close to real application and you can develop your own library and equipment with any language you want.

Bibliographie

- [1] bright hub engineering 2020, What is Robotics. What are Robots? Introduction to Robotics, viewed 04 april 2020, < <https://www.brighthubengineering.com/robotics/26216-introduction-to-robotics/> >.
- [2] Geeksforgeeks, Robotics | Introduction, viewed 04 april 2020, <<https://www.geeksforgeeks.org/robotics-introduction/>>.
- [3] Robotics, But what exactly is a robot? viewed April 2020, <<https://www.galileo.org/robotics/intro.html>>.
- [4] Online sciences 2014, Advantages and disadvantages of using robots in our life,viewed 04 april 2020,<<https://www.online-sciences.com/robotics/advantages-and-disadvantages-of-using-robots-in-our-life/>>.
- [5] Researchgate 2008-2020, Advantages and disadvantages of human and robot technology, viewed 04 april 2020, <https://www.researchgate.net/figure/Advantages-and-disadvantages-of-human-and-robot-technology_tbl1_228652684>.
- [6] M. Ben-Ari and F. Mondada.(2018). Robots and Their Applications , 1(1), page 1-2 <https://doi.org/10.1007/978-3-319-62533-1_1 >.
- [7] medium. (2019, 7 18). SLAM, Core technology of AR, What is it? Retrieved 07 17, 2020, from medium: <<https://medium.com/maxst/slam-core-technology-of-ar-what-is-it-e6c9ae4839b4>>.
- [8] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping (SLAM): part I The Essential Algorithms. Robotics & Automation Magazine, 2, 99–110, page 1. <<https://doi.org/10.1109/MRA.2006.1638022>>.
- [9] Simon J. D. Prince (2012). Computer Vision: Models, Learning and Inference. Cambridge University Press.

- [10] Computervisionblog. (2016, 1 13). The Future of Real-Time SLAM and Deep Learning vs SLAM. Retrieved 05 10, 2020, from computer vision blog: <<https://www.computervisionblog.com/2016/01/why-slam-matters-future-of-realtime.html?m=1&fbclid=IwAR3h7gXu9BKDgWIXIutYYwMAWcsUPH7epBo3SoOntRpiW3tSAp20JmxbP4g>>.
- [11] visiononline. (2009-2020). What-is-Visual-SLAM-Technology-and-What-is-it-Used-For. Consulté le 06 02, 2020, sur visiononline: <<https://www.visiononline.org/blog-article.cfm/What-is-Visual-SLAM-Technology-and-What-is-it-Used-for/99?fbclid=IwAR2gou8TNByEp2tnqTzldESmazE6Gs2MXx4SzcDtOxvim0l9BaWEuZ35uY>>.
- [12] How Does LiDAR Remote Sensing Work? Light Detection and Ranging(video file)available from:<<https://www.youtube.com/watch?v=EYbhNSUnIdU&t=105s>>.
- [13] hindawi. (2011). A Light-and-Fast SLAM Algorithm for Robots in Indoor Environments Using Line Segment Map. Retrieved 05 01, 2020, from hindawi: <<https://www.hindawi.com/journals/jr/2011/257852/>>.
- [14] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34–46, 2007.
- [15] Mrpt. (2020). List of MRPT apps. Retrieved 05 15, 2020, from mrpt: <<https://www.mrpt.org/list-of-mrpt-apps/>>.
- [16] Wikipedia contributors. (2019, August 29). EKF SLAM. In Wikipedia, The Free Encyclopedia. Retrieved 16:26, July 18, 2020, from< https://en.wikipedia.org/w/index.php?title=EKF_SLAM&oldid=913085542>.
- [17] wiki.ros.(2018,August,08).ROS Introduction, in Wiki ROS, The Free Encyclopedia Retrieved 16:26, July 18, 2020, from< <http://wiki.ros.org/ROS/Introduction>>.
- [18] wiki.ros.(2019-September-05).roscore, in Wiki ROS, The Free Encyclopedia Retrieved 16:26, July 18, 2020, from< <http://wiki.ros.org/roscore>>.

- [19] wiki.ros.(2018-December-05).nodes, in Wiki ROS, The Free Encyclopedia Retrieved 17:00, July 18, 2020, from< <http://wiki.ros.org/Nodes>>.
- [20] wiki.ros.(2019-January-13).msg, in Wiki ROS, The Free Encyclopedia Retrieved 17:15, July 18, 2020, from< <http://wiki.ros.org/msg>>.
- [21] wiki.ros.(2019-July-18).Services, in Wiki ROS, The Free Encyclopedia Retrieved 17:15, July 18, 2020, from< <http://wiki.ros.org/Services>>.
- [22] Alajlan, M., & Koubâa, A. (2016). Writing global path planners plugins in ROS: A tutorial. In Studies in Computational Intelligence (Vol. 625, Issue Volume 1),page 31-32,<https://doi.org/10.1007/978-3-319-26054-9_4>.
- [23] wiki.ros.(2020-June-11).Distrubutions , in Wiki ROS, The Free Encyclopedia Retrieved 17:15, July 18, 2020, from< <http://wiki.ros.org/Distributions>>.
- [24] wiki.ros.(2020-June-11).Distrubutions ,List of Distrubutions, in Wiki ROS, The Free Encyclopedia Retrieved 17:15, July 18, 2020, from < <http://wiki.ros.org/Distributions>>.
- [25] wiki.ros.(2020-mars-25). Ubuntu install of ROS Kinetic, in Wiki ROS, The Free Encyclopedia Retrieved 17:15, July 18, 2020, from < <http://wiki.ros.org/kinetic/Installation/Ubuntu>>.
- [26] NIRYO. (2018, 11 23). Debug Niryo One motors one by one. Consulté le 07 10, 2020, sur NIRYO: <<https://niryo.com/2018/01/8-reasons-use-ros-robotics-projects/D>>
- [27] <https://www.icsm.gov.au/education/fundamentals-mapping/history-mapping>