Autonomous Navigation of Mobile Robotics Using ROS

Authors

Muhammad Ans Akhtar	16-MCT-08
Muhammad Shoaib	16-MCT-49
Muhammad Uzair Sajid	16-MCT-53
Mir Usama-Ul-Haq	16-MCT-63

Project Supervisor

Dr. Ahmed Nouman

Assistant Professor

DEPARTMENT OF MECHATRONICS ENGINEERING

UNIVERSITY OF ENGINEERING & TECHNOLOGY TAXILA,

SUB-CAMPUS CHAKWAL

May 2020

Autonomous Navigation of Mobile Robotics Using ROS

Authors

Muhammad Ans Akhtar	16-MCT-08
Muhammad Shoaib	16-MCT-49
Muhammad Uzair Sajid	16-MCT-53
Mir Usama-Ul-Haq	16-MCT-63

A Project submitted in partial fulfillment of the requirements for the degree of

B.Sc. Mechatronics Engineering

Project Supervisor:

Dr. Ahmed Nouman

Assistant Professor

Chairman's Signature: _____

Supervisor's Signature: _____

DEPARTMENT OF MECHATRONICS ENGINEERING

UNIVERSITY OF ENGINEERING & TECHNOLOGY TAXILA,

SUB-CAMPUS CHAKWAL

May 2020

ABSTRACT

Since last few years, research field named mobile robotics faced some incredible improvements. In the field of mobile robotics which is interesting and keep studied well over the years, our objective is to make robots inhabitable in environments with can shared with people or not shareable. Autonomous navigation, self-localization and mapping in environment which can be shareable or non-shareable are main issues in the field of mobile robots and had been looked over with great focus over some past years but not too much we have a lot more to do to resolve these issues in an efficient way. The presented work describes the design and development of mobile robot and how it is controlled by Robot Operating System ROS based control system for autonomous navigation in predicted and unpredicted environment and includes real time environment maps and simulations and can be used in extensive application like search & rescue, social robot etc.

Keywords: Mobile Robotics, Localization, Autonomous Navigation, SL

UNDERTAKING

We certify that research work titled "*Autonomous Navigation of Mobile Robotics Using ROS*" is our own work. The material, taken from other sources is properly referred.

_____

Muhammad Ans Akhtar

(16-MCT-08)

_____

Muhammad Shoaib

(16-MCT-49)

_____

Muhammad Uzair Sajid

(16-MCT-53)

_____

Mir Usama-Ul-Haq

(16-MCT-63)

ACKNOWLEDGEMENT

*In the name of **ALLAH Almighty**, the most Beneficent, the Merciful, the Creator of world, the King of the kings, without whose mercy and grace this endeavor could not be possible. It is the grace of **ALLAH** and Love of **Hazrat Muhammad** (Peace be upon him), whose gracious favors enabled us to complete such a hard research work successfully.*

We feel a great sense of gratitude and sense of obligation to our respected supervisor, Dr. Ahmed Nouman, for his inspiring guidance and encouragement during the entire study program. He helped us at each step of this tedious task. It is impossible to do such a difficult work without the moral support of our beloved parents. Due to the prayers, hard work, encouragement and sacrifices of our parents we were able to relish our dreams.

We would like to extend our wholehearted thanks to the University of Engineering & Technology Taxila, Sub Campus Chakwal. Also, we would like to pay tribute to Project Coordinator, Dr. Muhammad Khurram Saleem and all the faculty members of Mechatronics engineering department for providing us the best guidance during this project.

Once again, we are thankful to our project supervisor, for his heartiest devotion and guidance.

DEDICATION

We dedicate this effort to our beloved parents, respected teachers who sacrificed their “today” to give us a successful and bright “tomorrow” and to all those who are suffering with COVID-19 or no more with us in this pandemic situation.

Contents

LIST OF FIGURES

LIST OF TABLES

ABBRIVIATIONS

ROS: Robot Operating System

SLAM: Simultaneous Localization and Mapping

URDF: Unified Robotic Description Format

RVIZ: ROS visualization

2D: 2 Dimensional

3D: 3 Dimensional

LIDAR: Light Detection and Ranging

AGV: Automated Guided Vehicle

SUV: Sports Utility Vehicle

PR: Personal Robot

OSRF: Open Source Robotics Foundation

API: Application Programming Interface

SLR: Single-Lens Reflex camera

MATLAB: Matrix Laboratory

XML: Xtensible Markup Language

GPS: Global Positioning System

PC: Personal Computer

Chapter: 01

INTRODUCTION

1-Chapter 1

INTRODUCTION

1.1- Introduction

Humans beings are super intelligent and have natural senses like vision and intelligence while robot lack these natural powers, human beings get information from their surroundings through these natural powers. Robot cannot judge any unknown environment unless it uses some external source that helps in sensing, this external source is different kind of sensors such as odometer, sonars, LIDARs, inertial measurement units (IMUs), global positioning system (GPS) and cameras. These sensors help the robot to sense a wide range of environment. For navigation in an indoor environment map is a basic need of a robot. Picking and placing of an object required known environment and senses. For performing such type of services, the robot should not only know about its environment but while it is moving, it should also be aware of its own location in that environment. Proper representation of the robot itself in the environment also plays a vital role to solve many issues related to autonomous navigation.

In this thesis we start with a little introduction of ROS in the next section then further we move on describing the 2D and 3D design of a robot then importing the design in ROS then making a uniformed Robot description format that is must to bring the robot in ROS environment then further describe the movement of robot in Rviz and Gazebo platform of ROS.

After that we move towards the one of major goal of the project that is map building in our project we use LIDAR sensor for this purpose and after map building we move toward the main topic of our project that is Autonomous navigation of a robot for this we use A-star algorithm which help our robot to select the optimize path rather than any path in the environment.

In this project, we are using our unique design for navigation we will discuss further about design in detail. The reason of using our own design is to meet such requirements that we need in our

industries and lack in other designs such as high load capacity, high speed of motors and a supportive design that allow further additions for manipulation purpose and for other purposes if needed.

Figure 1: Autonomous Mobile Robots

1.1.1- Auto Navigation

Autonomous Navigation means a vehicle can execute motion and plan its path autonomously without any external influence.

In some situations, remote navigation help is used in the planning step, while in other cases vehicle uses the information taken from input sensors and computes it to plan its motion by own itself. An autonomous vehicle is which not only navigates autonomously keeping it stability but also pre-plan its future movements to execute motion. AGV use navigation help when available but can also depend upon visual and auditory etc. When position information taken in the form of signals or environmental perception, machine intelligence conditions use to translate basic motivation (reason for leaving the present position) into a path and planned motion. The plan of motion can account for the communicated information of other robots to avoid any type of collisions, while taking keeping the dynamics of the vehicle's own movement.

1.1.2- ROS

Robot Operating System (ROS) is a Linux based software framework for operating robots. This framework uses the concept of packages, nodes, topics, messages and services.

A node is an executable program that takes data from the robot's sensors and passes it to other nodes. The information, which moves from node to node, called a message. Messages always travel via special ports called topics. A node, which sends messages on a topic, called a publisher and the receiving node has to subscribe the topic to receive that message, hence it called a subscriber. All related nodes combined in one package that can easily be compiled and exported to other computers. The packages are necessary to build a complete ROS-based autonomous robot control system

Background

Routing has a long history to apply autonomy science. Indeed, even before the development of robots, both the principle issues are known as "Voyaging Salesman Problem" and "Piano Mover's Problem" were around for a considerable length of time and has been tackled in different ways. At first, the directing issue was clarified for automated arms afterward for wheeled robots and flying robots. With the ascent of humanoid robots, to date, there have been a few proposed answers for directing them, every one of which has its own focal points and burdens. This broadness is at such a degree, that steering is yet an open issue. At present, look into different calculations for portable robot way arranging is an interesting issue. Versatile robots are broadly utilized in numerous risky modern fields where there might be perils for individuals, for example, aviation examine, the atomic business, and the mining business. To locate a protected way in a risky situation for the versatile robot is a necessity for the accomplishment of any portable mechanical frameworks. Along these lines, inquire about on way arranging calculations to make the robot move from the beginning point to the end point without impact

with snags is a central prerequisite for the versatile robot wellbeing in such situations. Besides, to diminish the handling time, correspondence postponement and vitality utilization, the arranged way is normally required to be ideal with the most limited length.

At the basic period of the robot business, a robot was essential involved by mechanical arms obliged by motor engines. Way making plans for the robot was normally in stationary obstruction condition. Nevertheless, with the progression of the robot advancement, robots have been used in various current fields, for instance, aeronautics explore, marine research, and mining industry, to just determine a couple. A lobster-like submerged walking robot is one of these new sorts of robots. Starting late, Australian experts have developed an unmanned submerged vehicle robot for reef contemplating. The robot is furnished with sonar and vision systems and works at the phase of the sea. Thusly, how to response quickly to the changing condition to keep up a key separation from the stationary shakes in the seabed and colossal moving fish is a fundamental issue in the arrangement and undertaking of the robot.

It's all about 2007 when some beginning pieces were becoming the part of excellent software ROS by efforts of Eric Berger, Keenan Wyrobek and students who are doing PhD at Stanford in Salisbury's robotics lab were doing personal robotics project and leading it. When working on robotics manipulation tasks there some difficulties for people due to diversity of field of robotics because a good software developer may not have required knowledge about its hardware, a person creating a latest path-planning algorithm might not have needed knowledge about the computer vision. To overcome all these difficulties, both students were ready to make a basic system from which anyone can take a start from academia to build upon. In early to do unification of system, PR1 selected as a hardware model and working on software started by keeping in mind about it. Then with the help of technology incubator named Willow Garage who was doing work on autonomous solar boat and autonomous SUV begun to work on PR2 robot as a modification

step by following PR1 and keeping ROS as a running software of it. Students were selected from different institutions to do work to make their contributions to ROS was meant to be multi-robot platform from the very beginning. 2011 was advertiser time for ROS with start of ROS Answers for ROS users. In the start of 2012, Willow Garage created the open source robotics foundation (OSRF). After these years, when Open Source Robotics Foundation took in charge of main development of ROS. After this, every year, a new version of ROS came to market. May be the main development of Open Source Robotics Foundation / Open Robotics years yet was proposal of ROS2, a change of API to ROS which is highlighting and as a result effort to make compatible real time programming, a wide range of computer environments and to use more latest technology.

Types of Environment

In motion, planning there is environment in which robot or machine must perform its operation. This environment is divided into two parts

- **Static Environment**

The environment does not contain any moving obstacle or having static obstacles known as the static environment.

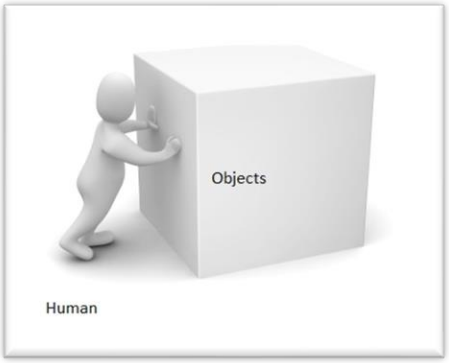


Figure 2: Static Objects and Dynamic Human Beings in Environment

- **Dynamic Environment**

The environment that contain any moving obstacle known as the dynamic environment.

1.2- Literature review

[1] **Robotics Operating System (ROS)** is an open source software for robots, which is developed by an international community. The main aim of this software is to provide a common platform for research and development of robotics. Before the development of Robotics operating system, each robot required a custom control system to be developed, manufactured, and tested before any research could begin. ROS have a built-in general algorithms and libraries, which can be easily applied to any robot configuration regardless of kinematic difference, provided that a [2] ROS compatible driver written for that robot exists. Due to this advantage of ROS, it greatly reduces the initial workload that is required for developing a new robot also with that it accelerates the research by providing a single platform to share and [3] use algorithms of each other across the globe regardless of the need of hardware for each user. There are hundreds of algorithms and libraries present in the [4] ROS platform from which we are using Simultaneous Localization and Mapping (SLAM), Navigation, and Visualization. We are also interested in [5] high accuracy path planning algorithms such as hierarchical A* search algorithm.

[6] **Simultaneous Localization and Mapping (SLAM),** the term itself describes the meaning mapping is related to map formation and localization mean position and location of robot in that map and simultaneously means both the things are happening at the same time, [7]wide range of algorithms are used for merging various sensors data to produce maps of a given environment while simultaneously determining the location of a robot within that given environment. This was serious and difficult problem under consideration because of the paradox that in order to know where the robot is, it must have an accurate map; and in order to build an accurate map, a robot must know it location that where the robot is. A problem was solved in 2001 and the solution was presented by Choset and Nagatani. Since then, [8] SLAM's ability to provide

mapping and localization at the same time has made it a technique of great and tremendous value in the field of robotics.

[9] Autonomous Navigation, Navigation is the process where a robot moves throughout the given environment for the execution of a certain task. An autonomous navigation is performed when the [10] robot moves without any externally interference (e.g., central system or person). The autonomous navigation depends on solving three main problems: localization, mapping, and path planning.

[11] **Control,** Control is the process of determining how the robot should move and sending the respective commands from the navigation system to the locomotion hardware. In this SLR, we consider control as part of the locomotion problem, since it is what drives the robot and determines its motion. [12] There are many projects related autonomous navigation using ROS but the key difference between these projects and our project is the difference of design and specifications there are certain things that we want to achieve on our end are added up in this design. Moreover, in most of the projects omni wheels are used that are much easier in operation but our design is differential drive.

1.2.1- How robot achieve auto navigation

At first robot constructs a two-dimensional (2D) geometrical representation of its given environment using the LIDAR laser scanner. It uses a combination of both the 2D geometric data and information from odometry provided by the wheel encoders to determine the location of robot where it is currently present. After gathering data, it generates a point cloud where each point corresponds to a certain location where it believes that, it could be based on gathered data. As the robot starts moving, it rules out possible locations in the environment and the number of

points in the point cloud decreases. In this way, its number of belief states rapidly converges to its true location and hence with the help of probabilistic inference robot achieves localization.

Once the localization of robot achieved successfully, then we supply the robot with goal coordinates and uses a global planner algorithm to plot an optimized path to those coordinates. The environment of the robot is represented internally as two-dimensional (2D) lossless images. The environment contains five different types of entities: known clear space, unexplored space, the inflation radius, obstacles and the robot itself. The inflation radius is a kind of boundary that radiates outward from all obstacles. The robot treats this radius as an obstacle and cannot make paths through it. The inflation radius and the robot's own radius are equal and it ensures that the robot always provide paths with enough space to clear all the obstacles. It is implemented as a buffer around all the obstacles instead of being applied to the robot itself for reasons of simplicity, and by doing this there are no effect on results. Within this paradigm, the global planner uses Dijkstra's algorithm to generate a path to the goal.

Once a global planner has generated the path, the local planner uses this data and translates this path into velocity and send commands for the robot's motors. It does this by creating a value function for a robot, sampling and simulating trajectories within the environment, scoring each trajectory based on its expected outcome, sending the value of highest-scoring trajectory as a velocity command for the robot, and repeat this process until the goal has been. The reason of local planner working this way is that it was written to be very general and simple, for the robots that may have irregular footprints, appendages, Ackerman steering geometry and other configurations which would not work using an algorithm specially designed for a robot with a simple shape, differential drive system, and no appendages.

The reason is, there is always a small amount of errors in the robot's physical movements, it is almost impossible for the robot to reach its goal coordinates accurately. This may result in the oscillating of robot around the goal as it continuously attempts to reach the exact goal location. This unacceptable behavior of robot can be circumvented by setting a range, which causes the robot to stop when it draws within a certain distance of the goal.

1.2.2- Motivation

It is our hope that our work in autonomous navigation of a mobile robot with ROS will facilitate and became a way for more advanced research in future projects and it is beneficial for the robotics community by publishing our changes. Addition to this, in implementing our project, we are able to delve deep into the inner working of the ROS navigation and trajectory planning algorithms, which has been developed by some of the leading researchers who worked on ROS and in mobile robotics from the beginning. The practical hands-on working knowledge of ROS could prove valuable in our future careers, as ROS becomes more of a standardized platform for advanced robotic applications. This project gave us working knowledge of a real-world planning system, enriching our mental toolboxes and software development skills.

1.2.3- Problem statement

“To develop a fully automated mobile robot that utilizes state of the art technology for solving Autonomous Navigation Tasks”

1.3- Application Areas

This project has many applications in our industries and other sectors like military Some of the major applications are given as follows:

- Search and rescue
- Social Robots
- Industrial robots
- Military robots

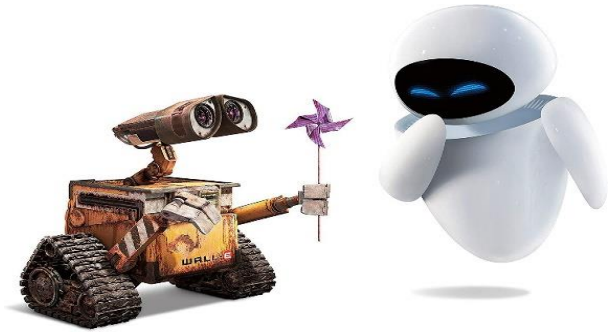


Figure 3: Social Robots

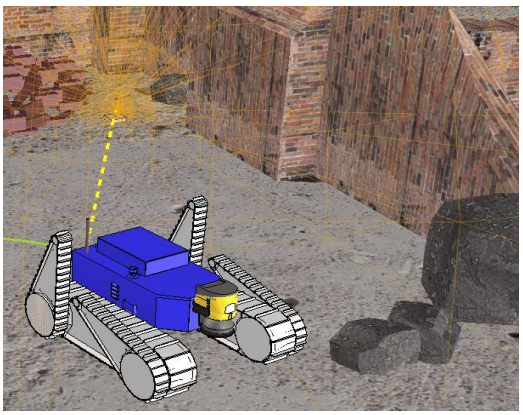


Figure 4: Search and Rescue Robot

Figure 6: Military Robots (Anti-Missile Gun)

Figure 5: Industrial Robots

1.4- deliverables

- Complete packages of a Robot in Ros
- A hardware Design with its 2D and 3D modelling
- Complete project Report
- A complete guide for a user to use this Robot

Chapter: 02

DESIGN & MANUFACTURING

2- Chapter 02

DESIGN AND MANUFACTURING

2.1- Hardware Design

Design in engineering is all about creating solutions for real-time problems, which occurs over time. This chapter describes the design of the system in hardware. The design is made based on the requirements, which are introducing this chapter. Through the design steps, some limitations could be faced to achieve the objectives.

2.1.1 Base

While designing Autonomous Guided Vehicle (AGV), the major part of the model is the base of AGV. It may control almost all of the operating characteristics of the AGV and it holds all the components being used mechanically or electronically, together. To build a successful AGV, the main step includes building an accurately designed base, which is often custom-built as per the requirement. The Baes is built/design by keeping in view the objective to be achieved in the future. Mostly, the base of the AGV is different and made as per the requirement of dimensions.

Base Material:

The first and major thing while designing a robot is base, on which the whole system is dependent, the first thing which rise a question is what would be suited material to do so the objectives were meet. In selecting material there are some factors that must be keep in mind and are mentioned below

- Material should fulfill all the product performance goals
- Material should be cost effective

- Material should have good mechanical properties

There are many materials under consideration while designing a base some most common materials are Aluminum, Acrylic, Cladding sheets and wood. We are using Aluminum because it fulfills all our product goals and have good mechanical properties and also it is cost effective for us

Some major advantages of aluminum are

- Aluminum is a very light metal with a specific weight of 2.7 g/cm^3, about a third of that of steel.
- It used to reduce dead weight and energy consumption while increasing load capacity.
- Aluminum naturally generates a protective thin oxide coating, which keeps the metal from making further contact with the environment.
- Aluminum shows increased tensile strength as temperatures drop not like steel, which rapidly becomes brittle at low temperatures.
- It does not rust in the atmosphere.

In short, keeping these things in mind for designing an Autonomous Robot Aluminum is best choice for us

2.1.2- Acrylic:

Acrylic is a plastic material, which has an outstanding strength, stiffness, and optical clarity Acrylic is a thermoplastic material that is often used in a sheet form and it can be used to build a robot.

Advantages

- Light weight and low density which gives a higher strength to weight ratio
- Clear and shiny which enhance the beauty of robot
- Available at cheap rates
- Easily available in market

Disadvantages

- Brittleness of acrylic cause sudden damage due to negligence
- Heat causes it to bend.

In short, keeping these things in mind for designing an Autonomous Robot Acrylic is best choice for us

2.1.3- Wheels

The wheel is round-shaped component transfer a vehicle's load from the axle to the ground and to provide traction on the surface over which the wheel travels. The main objective of this section is to study the wheeled of the mobile robot. However, while designing the wheel for the mobile robot the question is rise what should be the material of the wheels so the desire objectives can be met in a good manner?

Wheel Material:

As the material selection is a backbone in the process of designing any physical object. Material is a key consideration in wheel selection. In the context of product design in engineering, the

main goal of material selection is to minimize cost while meeting product performance goals. There is an abundance of material available in the market in the design context. The choice of wheels material will depend on how the load is dispersed within an Autonomous guided vehicle. For the wheel design, we consider three materials regarding the wheel design, Iron, Wood, and Teflon. We have study all these three and then choose Teflon as the best option.

Teflon

Teflon is a useful plastic material polytetrafluoroethylene (PTFE). It is one of the good class of plastics and nonreactive material and often used in containers etc. it is strong and highly resistant to wear.

Advantages

- It exhibits excellent thermal and electrical insulators and resistance to chemical reactance.
- It is long lasting than any other material and more secure.
- Lightweight, easy to manipulate, and machine.
- Its rough surface has good frictional effects that would be a benefit in the account of driving or manipulating the robot.

➢ It will help the robot to get smooth running and absorb vibrations.

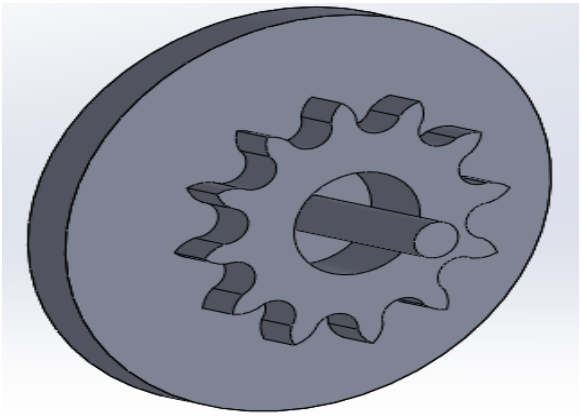


Figure 7 Wheel + Sprocket

Figure 8 Teflon Wheel

2.1.4- Raspberry pi

Raspberry pi is used as a single board computer for running all the codes for our robot. It is used to interface our software simulations with a real time hardware robot. We are using Raspberry pi 3B+ model, which has a 2 GB ram which meet out project requirements.

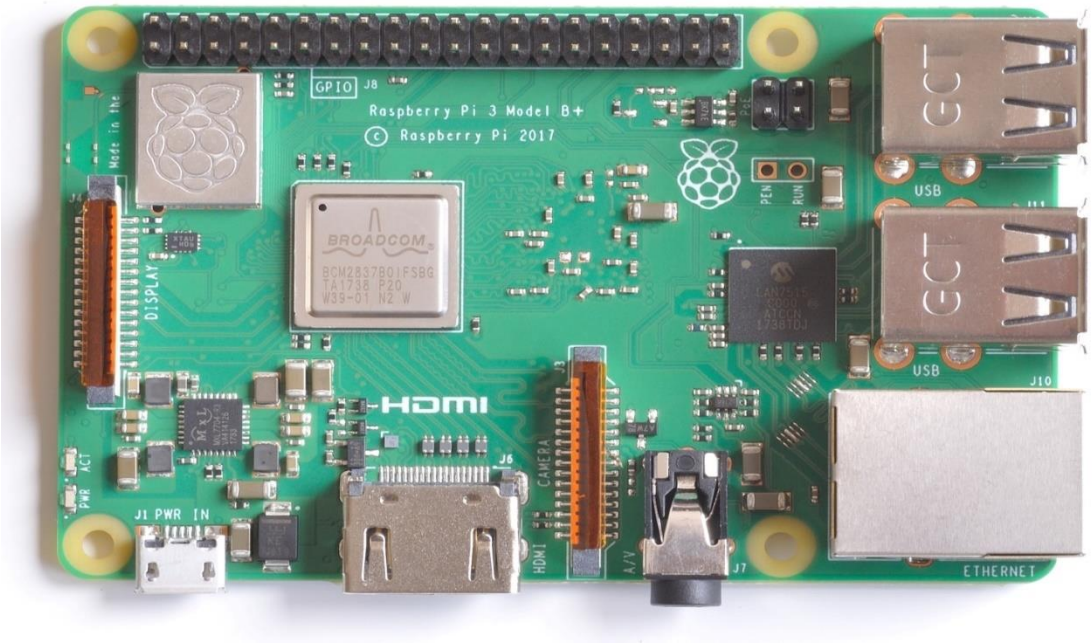


Figure 9 Raspberry pi 3B+

2.1.5- Lidar

LIDAR, which stands for Light Detection and Ranging, is a remote sensing method that uses light in the form of a pulsed laser to measure ranges (variable distances) to the Earth Lidar is a sensor used to avoid obstacles and used for creating the map of an environment.

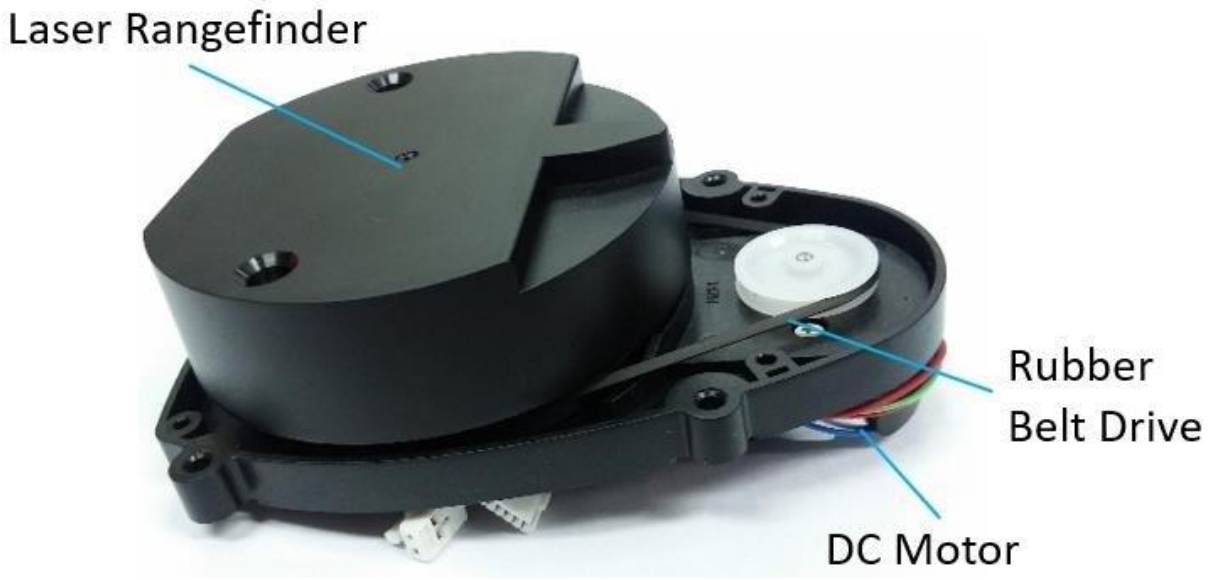


Figure 10 Lidar Sensor

2.1.6- DC encoded gear motors

DC gear motors are used as a driving unit of a robot. We are using Pololu 34:1 DC gear motors for our robot. These motors have a torque of 20Kg and 250 RPM

Figure 11 Encoded Gear Motors

2.2- Software Design

The Software part of manufacturing deals with the portions described below

2.2.1- Robot modelling in AutoCAD

As we are using our own robot instead of some already built robots like turtleBot or Roomba for autonomous navigation so we have to design it from beginning for a purpose we start with the 2D design of a robot, the software we used for modeling is AutoCAD 2016

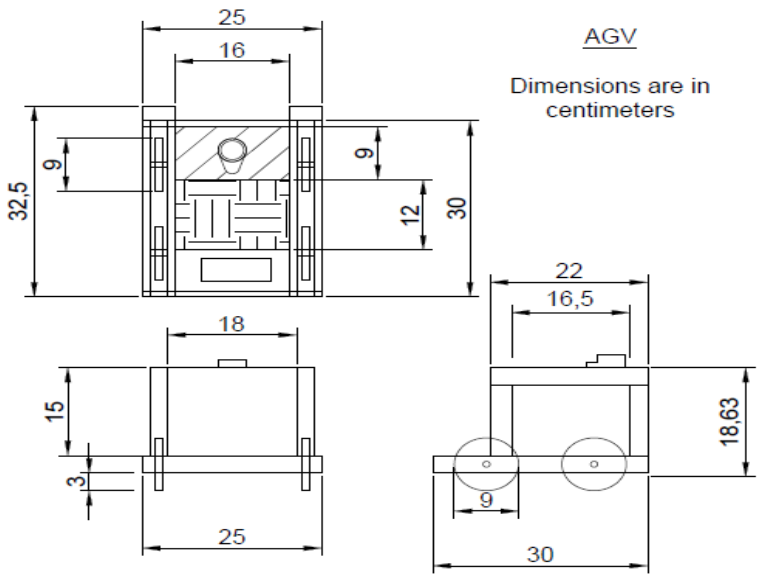


Figure 12 Robot Modeling in AutoCAD

2.2.2- Robot modelling in SolidWorks

Next step is to design 3D model of a robot. The purpose of designing 3D model is to see the original shape of a robot and then use this 3D design in our URDF for visualization in Rviz for 3D modeling of a robot we use Solidworks 2018.

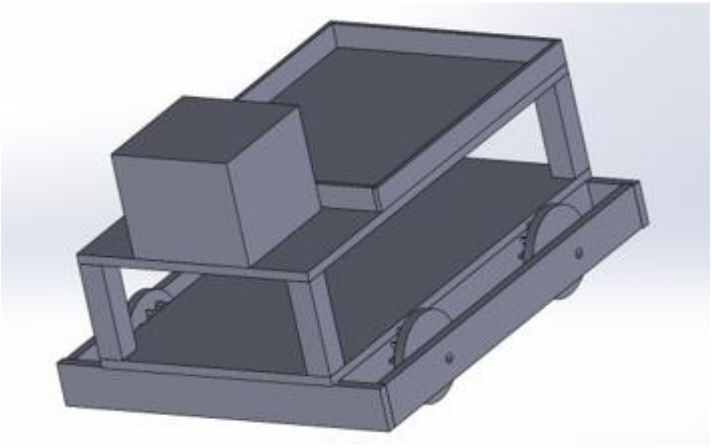


Figure 13 Robot Modeling in Solid Works

2.2.3- Robot modelling in ROS

Robot:

We already created our own autonomous robot model. To model the Robot in ROS, let's consider its essential components:

- One Base Frame
- Two rear wheels
- Two front Wheels
- Laser sensor (LIDAR)

To model the autonomous Robot in ROS we have to follow the following steps:

- URDF Formation
- Creating Macros
- Including Macros.Xacro
- Rviz for visualization

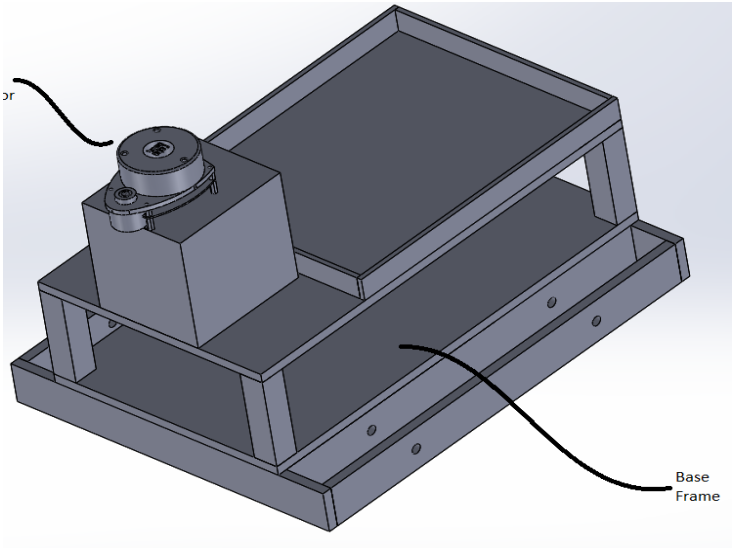


Figure 14 One Base, One Lidar Sensor

URDF:

URDF is a Unified Robot Description Format and it is written in an XML format. We model our robot through URDF in ROS. XML format is used to represent all type of robots, from a two-wheel toy robot to a walking humanoid.

You can imagine these components forming a tree: the base frame is the root, with connections to each of the wheel. In fact, URDF is only capable of representing robots whose kinematics can be described by a tree; looping structures are not allowed and fortunately are uncommon for robots.

We will translate this tree-like narrative description of the Robot into the language of URDF, which is focused primarily on **links and joints**:

- **link** is a rigid body, such as a base frame or a wheel.
- **joint** connects two links and define how the links moves with respect to each other

A joint can be several types in URDF format we can define a joint in following six ways:

Name	**Description**
continuous	A joint that can rotate indefinitely about a single axis
revolute	Like a continuous joint, but with upper and lower angle limits
prismatic	A joint that slides linearly along a single axis, with upper and lower
planar	A joint that allows translation and rotation perpendicular to a plane
floating	A joint that allows full six-dimensional translation and rotation
fixed	A special joint type that allows no motion

Table 1 Joints Types

Basic Tags of Link in URDF for Modelling:

Tags	Description
<link/>	Encloses all the properties and features of a link.
<visual/>	Encloses the visual geometry, origin and material information of link
<collisions/>	Encloses the collision geometry, creates a boundary around the robot to avoid collision.

Table 2 Tags of Link in URDF

Basic Tags of Joint in URDF for Modeling:

Tags	Description
<joint/>	Encloses all the properties and features of a joint.
<origin/>	Encloses the origin information.
<type/>	Encloses the type of joint.
<child/>	Encloses the name of children link.
<parent/>	Encloses the name of parent link

.

Base Frame:

At first, we ae going to model the frame of a robot which is the base_link of a robot. It consists of frame of robot without wheels and sensors with all the fixed joints. All the properties of this link are defined in `<link/>` tag under `<visual/>`.

```
<!-- Base Link -->
<link name="base_link">
  <visual>
    <origin rpy="0 0 1.57" xyz="${base_link_x} ${base_link_y} ${base_link_z}"/>
    <material name="grey"/>
    <geometry>
      <mesh filename="package://mcubot/meshes/base.stl" scale="1 1 1"/>
    </geometry>
  </visual>
  <collision>
    <origin rpy="0 0 1.57" xyz="${base_link_x} ${base_link_y} ${base_link_z+0.045}"/>
    <geometry>
      <box size="0.25 0.32 0.14"/>
    </geometry>
  </collision>
  <!--inertial>
    <mass value="1" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.00002" ixy="0.0" ixz="0.0"
             iyy="0.000011" iyz="0.0"
             izz="0.000025" />
    </inertial-->
</link>
```

Figure 15: Base Link URDF Script

Wheels:

Wheels consist of cylinders, the name of link for wheel is front_left_wheel_link for front left wheel, and similarly for all the links names are according to their position with base link. All the properties of this link are defined in `<link/>` tag under `<visual/>`.

```
<!--Front Left Wheel-->
<link name="front_left_wheel">
  <visual>
    <origin rpy="0 0 1.57" xyz="0 0 0"/>
    <material name="white"/>
    <geometry>
      <mesh filename="package://mcubot/meshes/v1wheel.stl" scale="1 1 1"/>
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.57 0 0"/>
    <geometry>
      <cylinder length="0.015" radius="0.046"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="0.1" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.001" ixy="0.0" ixz="0.0"
             iyy="0.001" iyz="0.0"
             izz="0.001" />
  </inertial>
</link>

<joint name="base_to_front_left_wheel" type="continuous">
      <origin rpy="0 0 0" xyz="${base_to_wheel_x} ${base_to_wheel_y} ${base_to_wheel_z}"/>
      <axis xyz="0 1 0"/>
  <parent link="base_link"/>
  <child link="front_left_wheel"/>
</joint>


<!--Front Right Wheel-->
<link name="front_right_wheel">
  <visual>
    <origin rpy="0 0 -1.57" xyz="0 0 0"/>
    <material name="white"/>
    <geometry>
      <mesh filename="package://mcubot/meshes/v1wheel.stl" scale="1 1 1"/>
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.57 0 0"/>
    <geometry>
      <cylinder length="0.015" radius="0.046"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="0.1" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.001" ixy="0.0" ixz="0.0"
             iyy="0.001" iyz="0.0"
             izz="0.001" />
  </inertial>
```

Figure 16: Wheel Links and Joints - URDF Script

Macros.Xacro:

Xacro (XML Macros) Xacro is an XML macro language. With Xacro, we can construct shorter and more readable XML files by using macros that expand to larger XML expressions.

As there are four wheels and different joint so, instead of writing the same code with very less difference again, we make the macros and then include them in model.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro" name="mcubot">
        <xacro:include filename="$(find mcubot)/urdf/properties.xacro"/>
        <xacro:include filename="$(find mcubot)/urdf/model.gazebo.xacro"/>

        <xacro:property name="base_link_x" value="0" />
        <xacro:property name="base_link_y" value="0" />
        <xacro:property name="base_link_z" value="0" />
        <xacro:property name="base_to_wheel_x" value="0.077" />
        <xacro:property name="base_to_wheel_y" value="0.0975" />
        <xacro:property name="base_to_wheel_z" value="-0.007" />
        <xacro:property name="base_to_lds_x" value="0.12" />
        <xacro:property name="base_to_lds_y" value="-0.12" />
        <xacro:property name="base_to_lds_z" value="0.12" />
```

Figure 17 XACRO Script in UR

Macro for Wheels:

Wheels consist of cylinders. The name of macro is defined as `link_wheel`. All the properties of this link are defined in `<link/>` tag that is sub-tag of

`<macro/>` tag that are:

`<visual/>`

Macro for Joints:

The name of macro for joint is defined as `joint_wheel`. All the properties of joint are defined in `<joint/>` tag that is sub-tag of `<macro/>` tag that are:

`<origin/>`

`<child/>`

```
<parent/>

<axis/>
```

Including and Using Macros:

Now we are going to include the macros created for wheels and joints. We can include Macros.Xacro by adding following line in the model.xacro (the file in which URDF code for base frame and wheel was written)

<xacro:include filename="$(find model.xacro)/urdf/macros.xacro"/>

For front_left_wheel, joint_wheel has been called by name “front_left_wheel_joint”. For front_right_wheel, joint_wheel has been called by name “front_right_wheel_joint”. For back_left_wheel, joint_wheel has been called by name “back_left_wheel_joint”. For back_right_wheel, joint_wheel has been called by name “back_right_wheel_joint”.

Joint type is defined as continuous. The parent link has been defined in macro that is base_link therefore here we don’t have to define it again.

The children link is as we want is described all the wheels that are connected with base_link that is parent link.

Laser Sensor:

We need to add a link on the front top of the base frame for LIDAR sensor, which helps in laser scan and for creating the map and avoiding the obstacles. This link consists of a box. All the properties of this link are defined in `<link/>` tag.

Material Properties:

We are using material tag for defining the material with a combination of primary colors that are red green and blue and an alpha value. Instead of writing it repeatedly it is better to make a Xacro file of materials that is materials. Xacro and then include it.

```
<?xml version="1.0" ?>

<robot name="xacro_properties" xmlns:xacro="http://ros.org/wiki/xacro">

  <!-- Init colour -->
  <material name="black">
      <color rgba="0.0 0.0 0.0 1.0"/>
  </material>

  <material name="dark">
    <color rgba="0.3 0.3 0.3 1.0"/>
  </material>
  <material name="light_black">
    <color rgba="0.4 0.4 0.4 1.0"/>
  </material>

  <material name="blue">
    <color rgba="0.0 0.0 0.8 1.0"/>
  </material>

  <material name="green">
    <color rgba="0.0 0.8 0.0 1.0"/>
  </material>

  <material name="grey">
    <color rgba="0.5 0.5 0.5 1.0"/>
  </material>

  <material name="orange">
    <color rgba="${255/255} ${108/255} ${10/255} 1.0"/>
  </material>

  <material name="brown">
    <color rgba="${222/255} ${207/255} ${195/255} 1.0"/>
  </material>

  <material name="red">
    <color rgba="0.8 0.0 0.0 1.0"/>
  </material>

  <material name="white">
    <color rgba="1.0 1.0 1.0 1.0"/>
  </material>

</robot>
```

Figure 18: Material Properties of Robot in ROS World

Rviz Launch:

Now as we have added all links and joints in model.xacro, we need a launch file that add all the required node and launch them together to visualize and to apply all provided properties perfectly. These nodes include joint_state_publisher and robot_state_publisher.

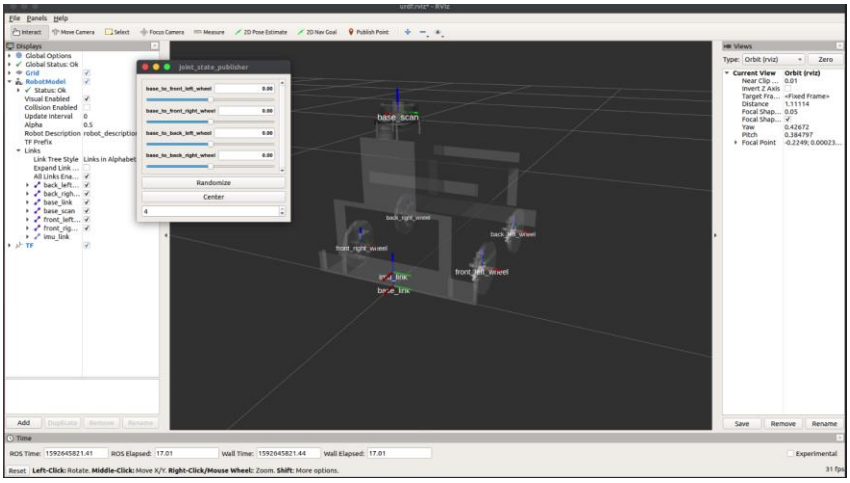


Figure 19: Robot in RVIZ

Model:

The complete model in Rviz is shown below.

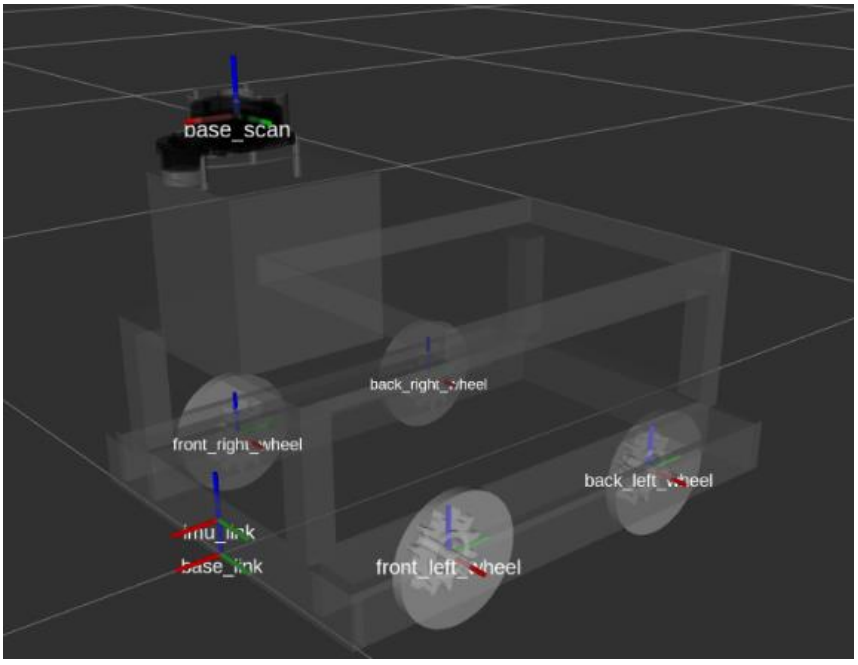


Figure 20: Mobile Robot in RVIZ

2.2.4- Algorithms Used

Dijkstra’s algorithm

Dijkstra's calculation is a calculation for finding the briefest ways between hubs in a diagram, which may speak to, for instance, street systems. It was brought about by PC researcher Edsger W. Dijkstra in 1956 and distributed three years after the fact

Pseudo code of Dijkstra:

In the going with Algorithm, the code u vertex in Q with min dist, filters for the vertex u in the vertex set Q that has the least dist regard. length (u,v) reestablishes the length of the edge joining the two neighbor center points u and v. The variable alt on line 18 is the length of the path from the root center point to the neighbor center point v if it by one way or another happened to encounter u. If along these lines is shorter than the current most concise path recorded for v, that present way is

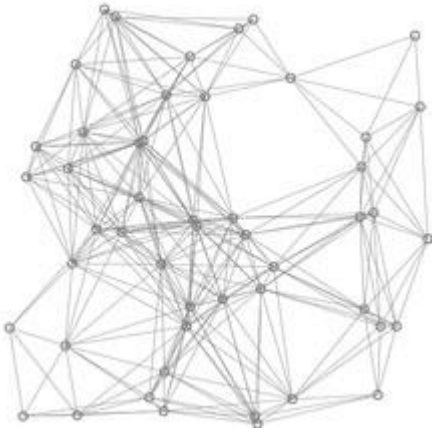


Figure 21 Dijkstra

displaced with this alt way. The prev bunch is populated with a pointer to the "accompanying bounce" center on the source diagram to get the shortest course to the source.

A demo of Dijkstra's computation reliant on Euclidean division. Red lines are the shortest way covering, i.e., interfacing u and prev[u]. Blue lines show where loosening up happens, i.e.,

interfacing v with a center point u in Q, which gives a shorter route from the source to v.

```
 1 function Dijkstra (Graph, source):
 2
 3   create vertex set Q
 4
 5   for each vertex v in Graph:           // Initialization
 6     dist[v] ← INFINITY                  // Unknown distance from source to v
 7     prev[v] ← UNDEFINED                 // Previous node in optimal path from source
 8     add v to Q                          // All nodes initially in Q (unvisited nodes)
 9
10   dist[source] ← 0                      // Distance from source to source
11
12   while Q is not empty:
13     u ← vertex in Q with min dist[u]    // Node with the least distance
14                                         // will be selected first
15     remove u from Q
16
17     for each neighbor v of u:           // where v is still in Q.
18       alt ← dist[u] + length (u, v)
19       if alt < dist[v]:                 // A shorter path to v has been found
20         dist[v] ← alt
21         prev[v] ← u
22
23   return dist[], prev[]
```

On the off chance that we are just inspired by a most brief way between vertices source and target, we can end the pursuit after line 15 if u = target. Presently we can peruse the briefest way from source to focus by turn around cycle:

```
1  S ← empty sequence
2  u ← target
3  if prev[u] is defined or u = source:     // Do something only if the vertex is reachable
4    while u is defined:                    // Construct the shortest path with a stack S
5      insert u at the beginning of S       // Push the vertex onto the stack
6      u ← prev[u]                          // Traverse from target to source
```

Graphical Representation:

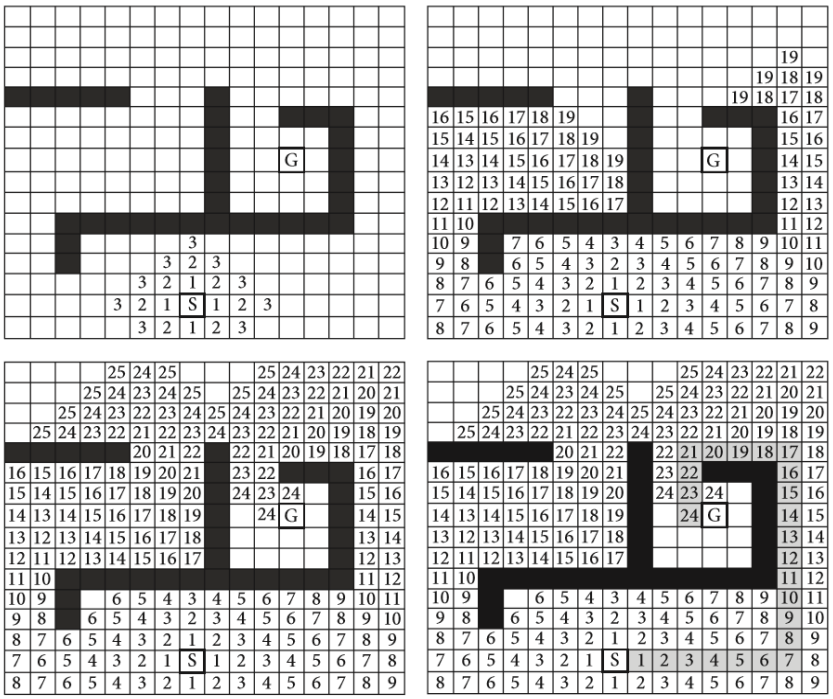


Figure 22 Dijkstra Method to find optimal path

A Star Algorithm:

A* is an educated pursuit Algorithm, or a best-first hunt, implying that it is defined as far as weighted diagrams: beginning from a particular beginning hub of a chart, it intends to discover a way to the given objective hub having the littlest cost. It does this by keeping up a tree of ways beginning toward the begin hub and expanding those ways one edge at any given moment until its end model is fulfilled. Pseudocode

The following code describes the A* algorithm:

```
1 function Dijkstra (Graph, source):
function reconstruct path (came From, current)
   total path: = {current}
   while current in cameFrom.Keys:
      current: = cameFrom[current]
      total_path.append(current)
   return total_path

function A_Star (start, goal)
   // The set of nodes already evaluated
   closedSet: = {}

   // The set of currently discovered nodes that are not evaluated yet.
   // Initially, only the start node is known.
   openSet: = {start}

   // For each node, which node it can most efficiently be reached from.
   // If a node can be reached from many nodes, cameFrom will eventually contain the
   // most efficient previous step.
   cameFrom: = an empty map

   // For each node, the cost of getting from the start node to that node.
   gScore: = map with default value of Infinity

   // The cost of going from start to start is zero.
   gScore[start]: = 0

   // For each node, the total cost of getting from the start node to the goal
   // by passing by that node. That value is partly known, partly heuristic.
   fScore: = map with default value of Infinity

   // For the first node, that value is completely heuristic.
   fScore[start]: = heuristic_cost_estimate(start, goal)

   while openSet is not empty
```

```
current: = the node in openSet having the lowest fScore[] value
if current = goal
   return reconstruct_path (cameFrom, current)

openSet.Remove(current)
closedSet.Add(current)

for each neighbor of current
   if neighbor in closedSet
      continue                    // Ignore the neighbor which is already evaluated.

   // The distance from start to a neighbor
   tentative_gScore: = gScore[current] + dist_between(current, neighbor)

   if neighbor not in openSet// Discover a new node
      openSet.Add(neighbor)
   else if tentative_gScore >= gScore[neighbor]
      continue;

   // This path is the best until now. Record it!
   cameFrom[neighbor]: = current
   gScore[neighbor]: = tentative_gScore
   fScore[neighbor]: = gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)
```

Example:

An example of an A* algorithm in action where nodes are cities connected with roads and h(x) is the straight-line distance to target point:

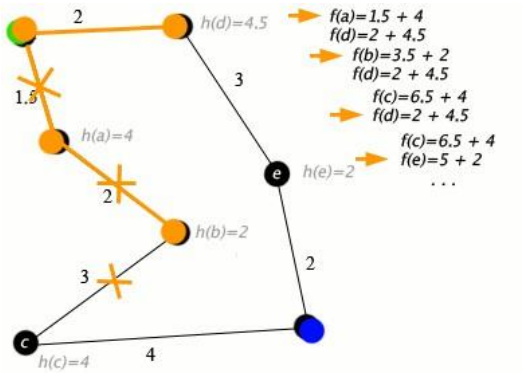


Figure 23: Example

Key: green: start; blue: goal; orange: visited

Animation

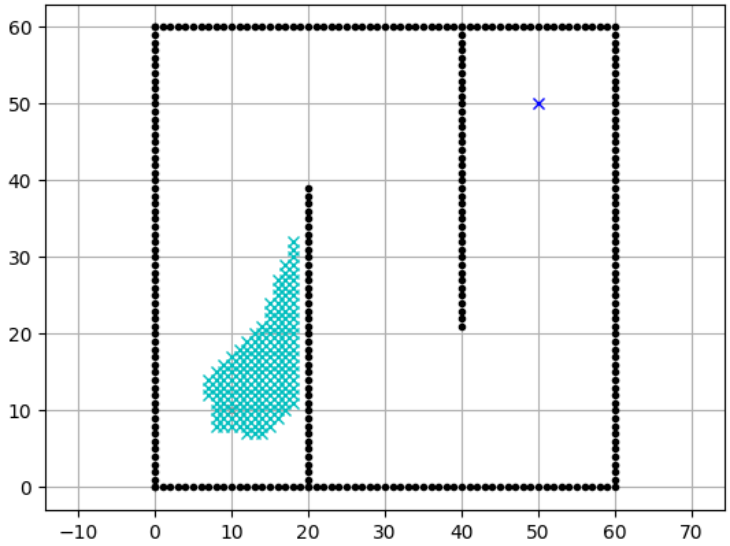


Figure 24 Before

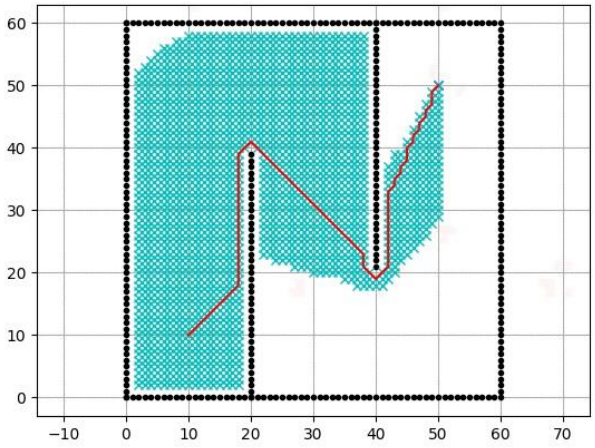


Figure 25: After

Chapter: 03

EVALUATION OF PROTOTYPE

3- Chapter 3

EVALUATION OF PROTOTYPE

3.1- Simulation of robot in ROS

Our URDF model of the Robot captures the kinematics and visual appearance of the robot, but it doesn't say anything about the physical characteristics that are needed to simulate it. To simulate a Robot in Gazebo, we need to add two new tags to every link in the model:

<collision/>

Similar to visual, this tag defines the size and shape of the robot's body, for the purpose of determining how it will interact with other objects. The collision geometry can be identical to the visual geometry, but it's often different; e.g., you may use a complex mesh for a good visual appearance, but a set of simple shapes (boxes, cylinders, etc.) for efficient collision detection.

<inertial/>

This tag defines the mass and moment of inertia of the link, which are needed to move it according to Newton's laws.

All these tags have been added in all links to simulate it.

Inertia:

We are using an inertial tag in our modeling to define the inertial properties with appropriate parameters. We are direct using this in our mode,xacro but you can also make a separate Xacro file for inertia to make it more simple and to avoid using it again and again.

robot.xacro | inertia.xacro | gazebo

```
<?xml version="1.0" encoding="UTF-8"?>
<robot>
  <macro name="cylinder_inertia" params="m r h">
      <inertia  ixx="${m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0"
                iyy="${m*(3*r*r+h*h)/12}" iyz = "0"
                izz="${m*r*r/2}" />
  </macro>

  <macro name="box_inertia" params="m x y z">
        <inertia  ixx="${m*(y*y+z*z)/12}" ixy = "0" ixz = "0"
                  iyy="${m*(x*x+z*z)/12}" iyz = "0"
                  izz="${m*(x*x+z*z)/12}" />
  </macro>

  <macro name="sphere_inertia" params="m r">
        <inertia  ixx="${2*m*r*r/5}" ixy = "0" ixz = "0"
                  iyy="${2*m*r*r/5}" iyz = "0"
                  izz="${2*m*r*r/5}" />
  </macro>
</robot>
```

Figure 26 Inertial Properties of Robot in ROS World

Now we're ready to Simulate our Robot model in Gazebo. There are a few different ways to do this. Because we want to use some ROS tools with our simulated robot (as opposed to working solely within Gazebo), we'll follow this pattern, using roslaunch to automate things:

1. Load the robot's URDF model into the parameter server.
2. Launch Gazebo (e.g., with an empty world).
3. Use a ROS service call to spawn an instance of the robot in Gazebo, reading the URDF data from the parameter server.

This process might seem a little round about, but it's actually a very flexible way of doing things. For a start, it gets the URDF model onto the parameter server, where it can be accessed by other nodes. By convention, the URDF model is stored in the parameter server under the name /robot_description (you can use another name for this parameter, but then you'd have

to change the default settings for many tools). Once it's on the parameter server, the URDF model can be used by tools like Rviz, which needs the model to visualize the robot, or a path planner, which needs the model to know the robot's shape and size.

3.2- Gazebo Launch

A launch file is required to launch all required nodes at the same time. These nodes include:

- World/Environment
- Robot Model
- Differential Drive Node (amcl)
- Move_base node

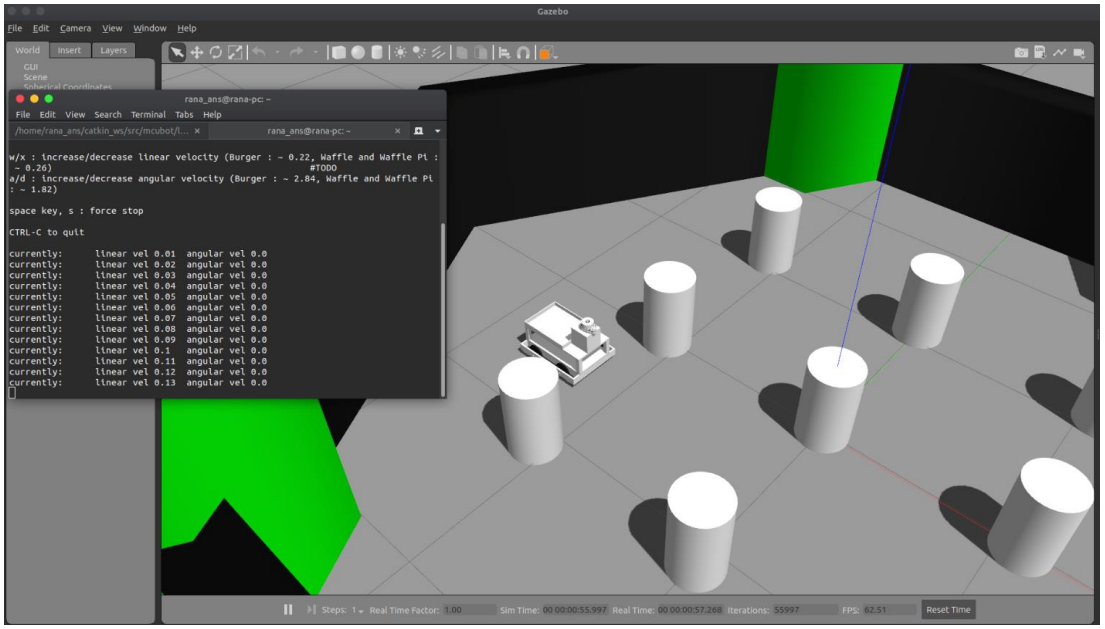


Figure 27: Gazebo World Launch

3.2.1- Simulation of Robot

The simulation of robot in gazebo is shown below.

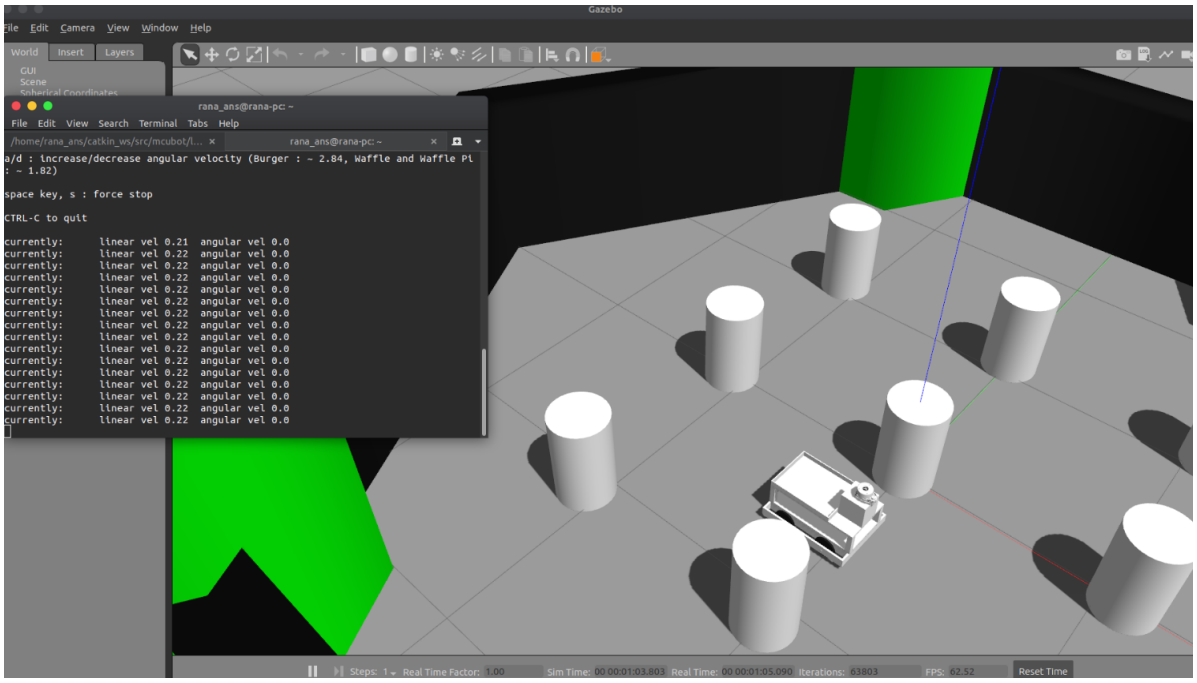


Figure 28: Simulation in Gazebo

3.3- Laser Scanner Configuration

We are building a physical robot this is where we would buy the sensor. Because here we're working in simulation, we can just edit some URDF. We need to add the link that will represent the sensor, plus a joint to attach it to the robot and we have done it. As with other parts of the robot, we must provide reasonable mass and inertia values for the laser; without that information, we cannot incorporate the laser into a physics- based simulation like Gazebo.

```
<!-- Gazebo plugin for Lidar Sensor -->
<gazebo reference="base_scan">
  <material>Gazebo/FlatBlack</material>
  <sensor type="ray" name="lds_lfcd_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>$(arg laser_visual)</visualize>
    <update_rate>5</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>360</samples>
          <resolution>1</resolution>
          <min_angle>0.0</min_angle>
          <max_angle>6.28319</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.120</min>
        <max>3.5</max>
        <resolution>0.015</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_lds_lfcd_controller" filename="libgazebo_ros_laser.so">
      <topicName>scan</topicName>
      <frameName>base_scan</frameName>
    </plugin>
  </sensor>
</gazebo>

<!-- Gazebo plugin for ROS Control -->
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/</robotNamespace>
  </plugin>
</gazebo>
```

Figure 29 LIDAR Sensor Plugin in URDF

3.4- 3D- Sensor Lidar

The Robot does not have a real laser scanner. It does, however, have a LIDAR (3D Sensor) depth camera, and the Robot software (ROS) stack extracts the middle few rows of the LIDAR depth image, does a bit of filtering, and then publishes the data as sensor_msgs/LaserScan messages on the scan topic. This means that from the standpoint of the high-level software, the data shows up exactly like "real" laser scans on more expensive robots. The only difference is that the field of view is just narrower, and the maximum detectable range is quite a bit shorter than with typical laser scanners.

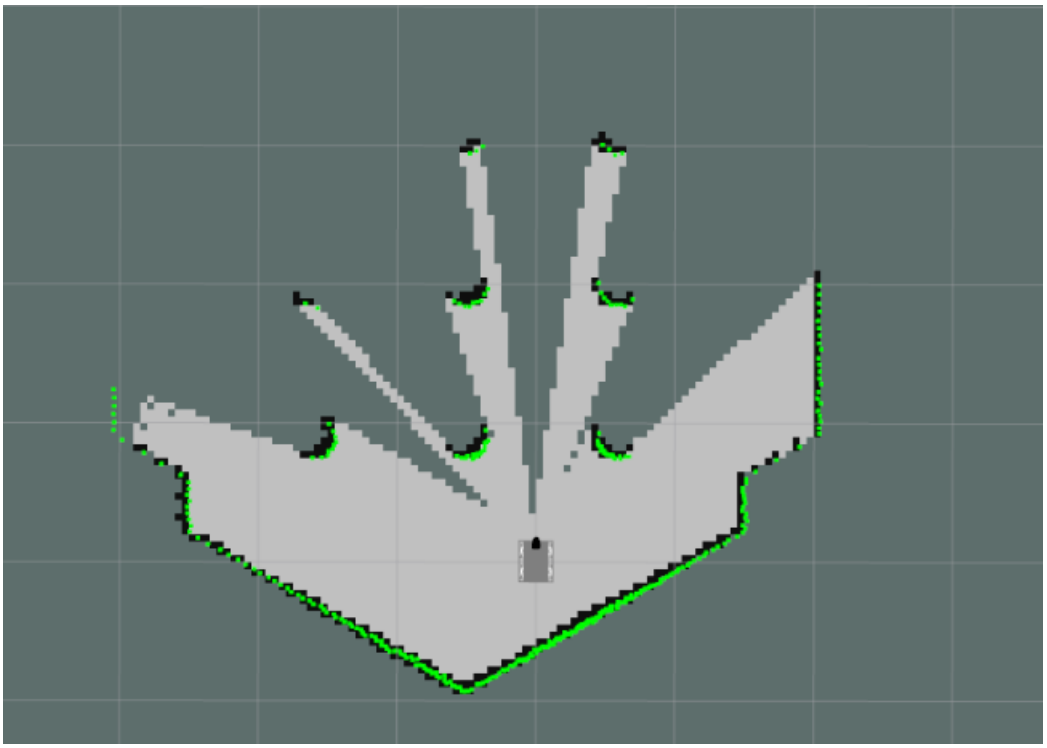


Figure 30: Laser Sensor Mapping in ROS

3.5- Simulation

The simulation of robot with an obstacle in front of it is shown blow. The detection of this obstacle is also visualized in the above fig

3.6- Navigation

The ROS Navigation Stack is intended for 2D maps, square or round robots with a holonomic drive, and a planar laser scanner, all of which our Robot has. It utilizes odometry, sensor information, and an objective posture to give safe speed directions. The hub "move_base" is the place all the enchantment occurs in the ROS Navigation Stack. Utilizations a worldwide and nearby organizer to achieve the route objective. Oversees correspondence inside the route stack. Sensor data is assembled (sensor sources hub), at that point put into viewpoint (sensor changes hub), at that point joined with a gauge of the robot's position dependent on its beginning position (odometry source hub). This data is distributed so that "move_base" can figure the direction and pass on speed directions (through the base controller hub).

3.6.1-Global planner

This package uses an implementation of A* for a fast, interpolated global planner for navigation via a discretized map. There are other implementations like a traditional Dijkstra planner that are available for configuration during launch, but the A* is undoubtedly faster.

While this provides a high-level plan through a map, the question remains of how the robot's base frame must be controlled considering local obstacles such as corners of walls and stand-alone objects. The move base package provides such a feature. Once the global planner searches and publishes a plan, this package will then take that plan and link the global/local planners with an actual controller. Below is a graphic showing how the move_base node subscribes and publishes to various topics for a differential wheel robot.

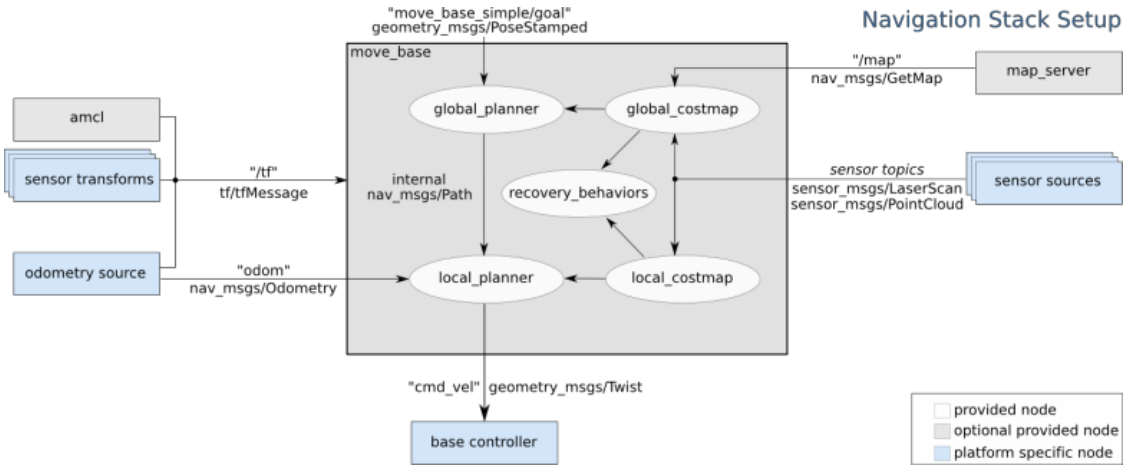


Figure 31 Navigation Stack

3.6.2- Dijkstra Vs A*

Dijkstra is novel because it begins exploring from a starting node and continues until it reaches the ending node. It then identifies the list of nodes that connect the end node to the

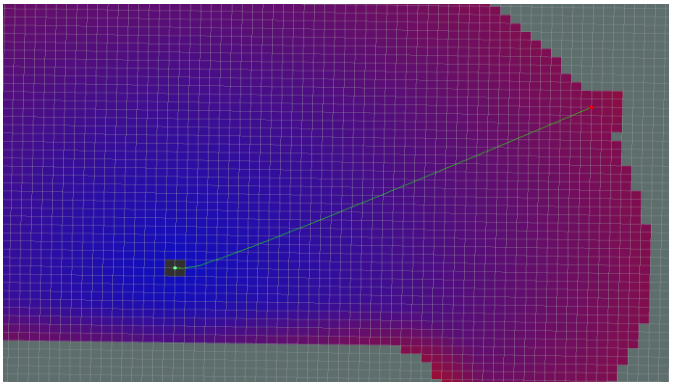
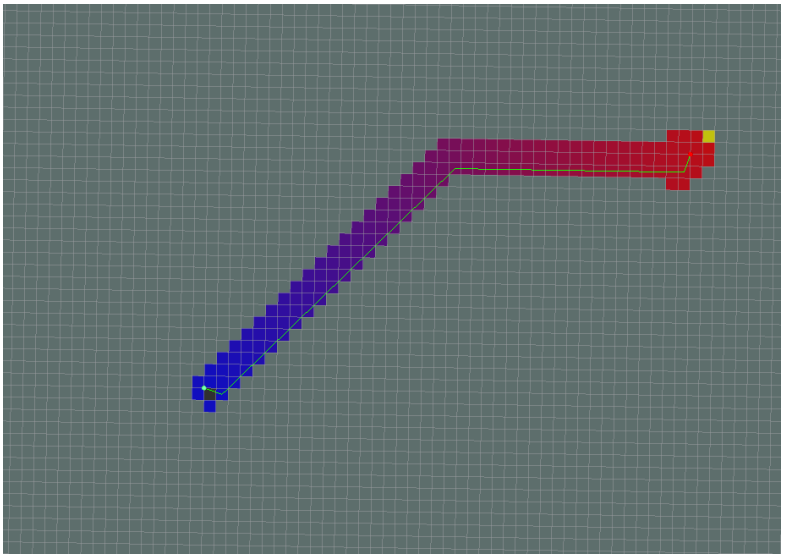


Figure 32 Dijkstra's Algorithm attempting to go from a start point to a final point

start as one single path. This means that Dijkstra works by exploring nodes in order based by their distance from the ***starting node.*** To put it another way, this means that it will equally explore nodes that are farther from the goal as well as nodes closest to the goal as it expands outwards from the starting node. As one can imagine, that can become computationally expensive as our graph representation of our world grows.

See how it has to expand out almost the entire distance of the path in every direction before it finds the goal.

A* tries to improve on Dijkstra's Algorithm by focusing only on exploring nodes that bring



us closer to our goal. This greatly improves our speed of finding the shortest and most direct path.

Figure 33 A Algorithm attempting to go from a start point to a final point*

The same path-planning problem as shown above but look at how quickly the goal is found. With the use of some forcing heuristic, we ensure that we only expand nodes that drive us towards the goal node.

The way A* does this is via the introduction of a heuristic function that guides the node expansion towards the goal. Our heuristic functions will be used to create a correlation from every node in the graph to a non-negative cost value. Heuristic functions (represented as the function H below) must satisfy two basic criteria:

- H(goal) = 0
- For any two adjacent nodes x and y:
- $H(x) <= H(y) + d(x, y)$
- d(x, y) = weight/length of edge from x to y

The idea that our expense is zero at the objective hub helps address the issue portrayed

over that when we investigate from the beginning hub, we have no genuine course. So, in the event that we have some additional esteem that can evaluate if an investigated hub will convey us closer to the objective, we improve our pursuit time. Following these two criteria, we can guarantee that each hub will be not exactly or equivalent to the length of the briefest way from hub and to the objective. Two of the most popular examples of heuristics are the Euclidean Distance, and Manhattan Distance from any node, n, in the graph to the goal, g, in the graph. We can, therefore, say, given any node, n, at (x_n, y_n) and a goal, g, at (x_g, y_g):

1. **Euclidian Distance**

$$H(x_n, y_n) = \sqrt{((x_n - x_g)^2 + (y_n - y_g)^2)}$$

2. **Manhattan Distance**

$$H(x_n, y_n) = |(x_n - x_g)| + |(y_n - y_g)|$$

3.6.3- Simulation of Navigation

Right now, we are ready to go. Use the ***2D pose estimate*** button on the top of your screen to localize the robot., use the ***2D Nav goal*** to give the robot a target position to move to. Then, watch the Robot as it generates a path and tries to follow it.

Simulation:

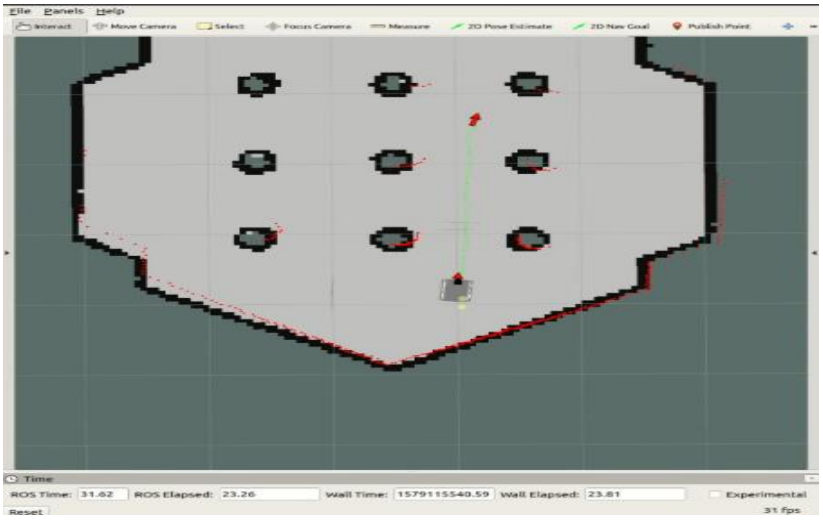


Figure 34 Simulation of Navigation

3.7- Autonomous Navigation

We are going to give our robot the ability to autonomously navigate with a known map. To add routing to a robot, we need to launch three new nodes:

- map_server, to give the static map against which the robot will locate and plan.
- amcl, to locate the robot against the static map.
- move_base, to handle global plan and local control for the robot.

To run map_server, we need a static map. Therefore, we are creating a map of world that we created earlier.

3.8- Building a map

In order to navigate in any environment, we need a few key elements. For a robot, this is no different. What we need is:

- A map of the environment
- The current location on the map (localization)
- A route to take (path planning)
- Obstacle avoidance

A map is mandatory for navigation. There are two ways to get a map:

- Use a pre-existing map
- Build by the robot itself (mapping process)

The mapping process is called **SLAM** *(Simultaneous Localization and Mapping)*. Here we are using a specific type of SLAM called **G-mapping**.

Using the Laser Scanner, the Robot can get information that are the obstacles in the

environment. The slam-gmapping node then processes this information. After that, it published to map topic.

By using the **map_server**, **map_saver** and **slam_gmappig** node the map created of the world is shown below.

Now we will load map and get our robot to localize itself.

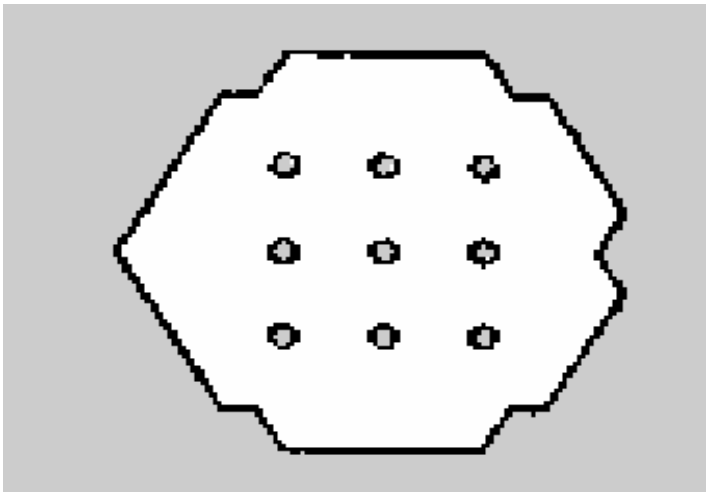


Figure 35 Maps of World

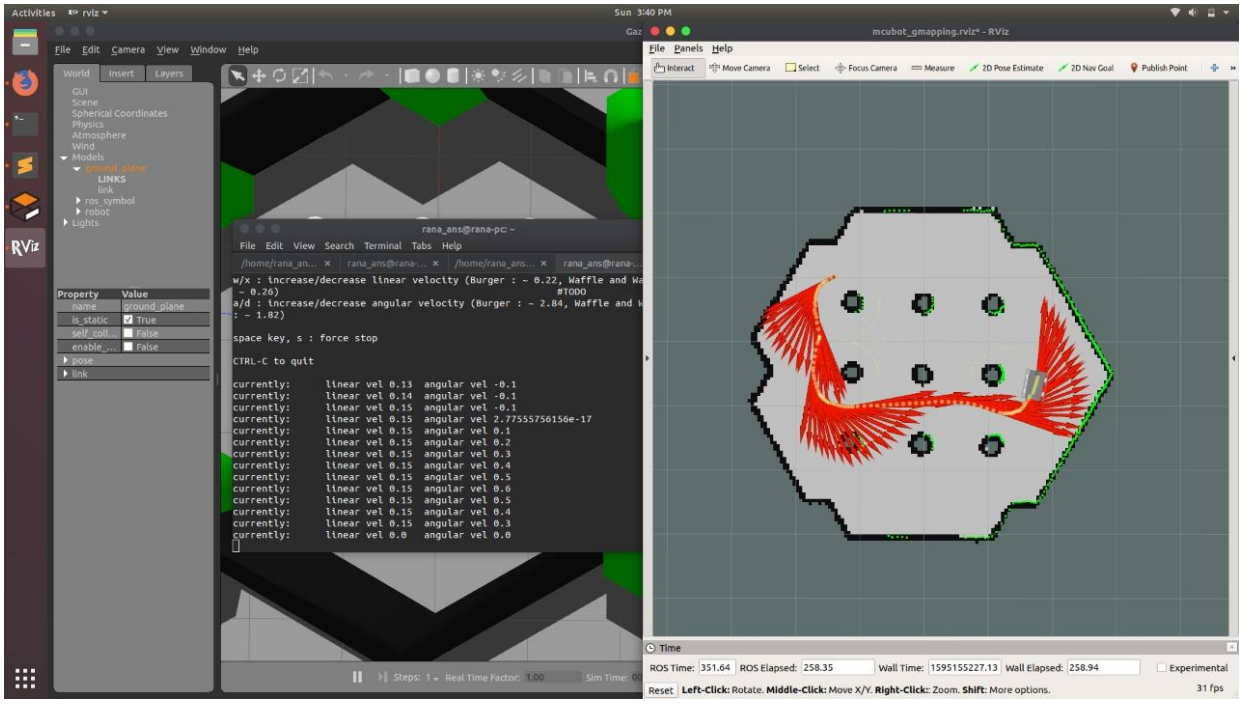


Figure 36 Mapping process of the 3-D World

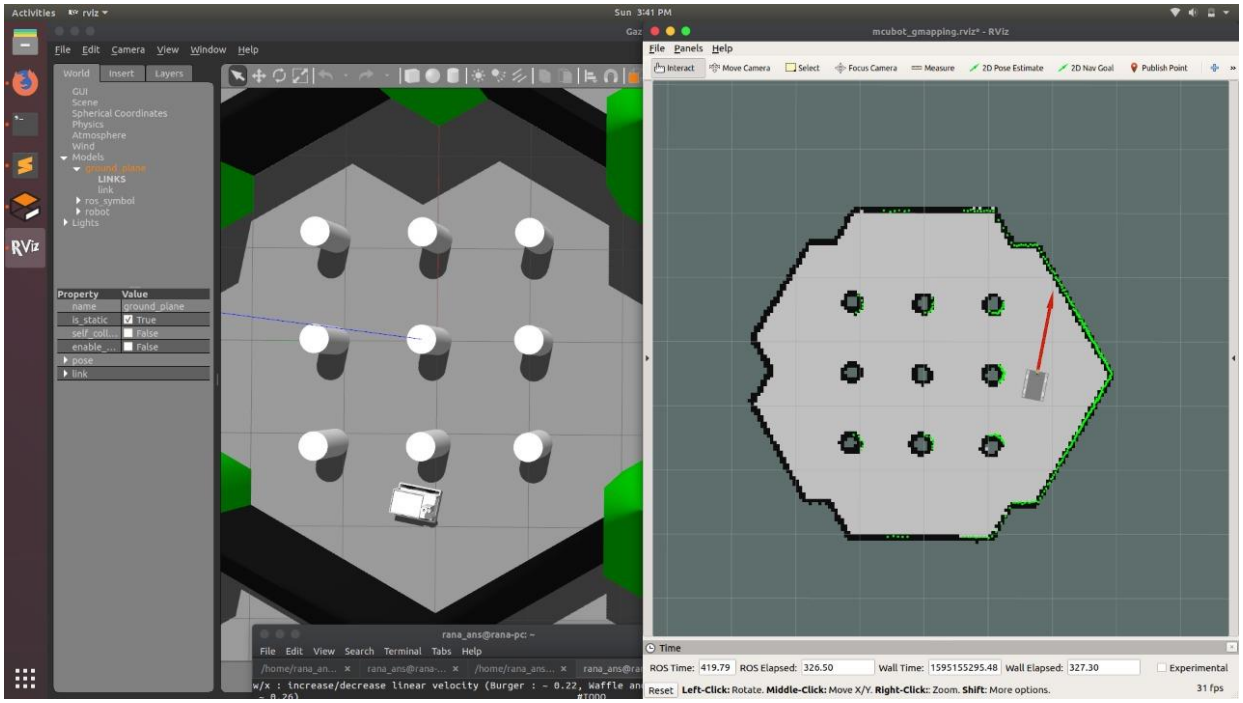


Figure 37 Mapping process of the 3-D World

3.9- Localization

Localization methods are grouped into two categories: Global and Local.

3.9.1- Global

- Give a location with respect to the world
- Often inaccurate compared to local methods
- For example: GPS or Wi-Fi hotspots

3.9.2- Local

- Give a location with respect to local sensor feedback
- Can be highly accurate compared to global methods
- For example, laser scanner and cameras

In general, Local Methods of localization are more accurate. Sometimes global localization methods cannot give accurate enough results to localize a robot in a map. Local sensor readings can give accurate positioning with respect to obstacles in the map, but there may be multiple places on the map that look the same. Location can be narrowed down by estimating a range of positions a robot is likely to be by tracking how much the robot moves and by taking many measurements.

The amcl represents Adaptive Monte Carlo Localization. This is a probabilistic confinement framework for a robot moving in 2D, which utilizes a molecule channel to follow the posture of a robot against a known guide. The known guide for this situation is the one we just made.

The round black circle is the Robot. It is surrounded by green arrows, which are probabilistic estimates of its actual position. Then you see the edges from the square as in our original map. One of its edges is purple with (and a thin white line form the laser scan is on it). This is one of 'measurements' the robot has taken to localize itself. So now, we have to move the robot and let it take more measurements to improve localization.

Now moving the robot around using keyboard, which uses **teleop node** and observe the green arrows around it. Since we get more certainty about where the robot is, the arrows should come close and closer indicating more and more accurate probabilistic model.

The robot model will be running as follows in real time world also in the software. The following photos show the proper process:

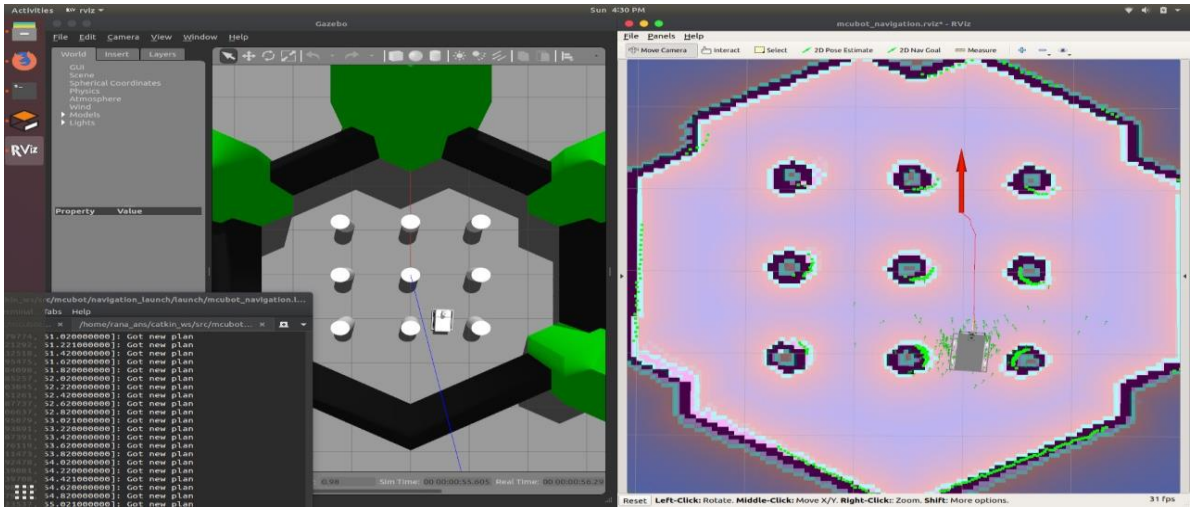


Figure 38 Goal Defined

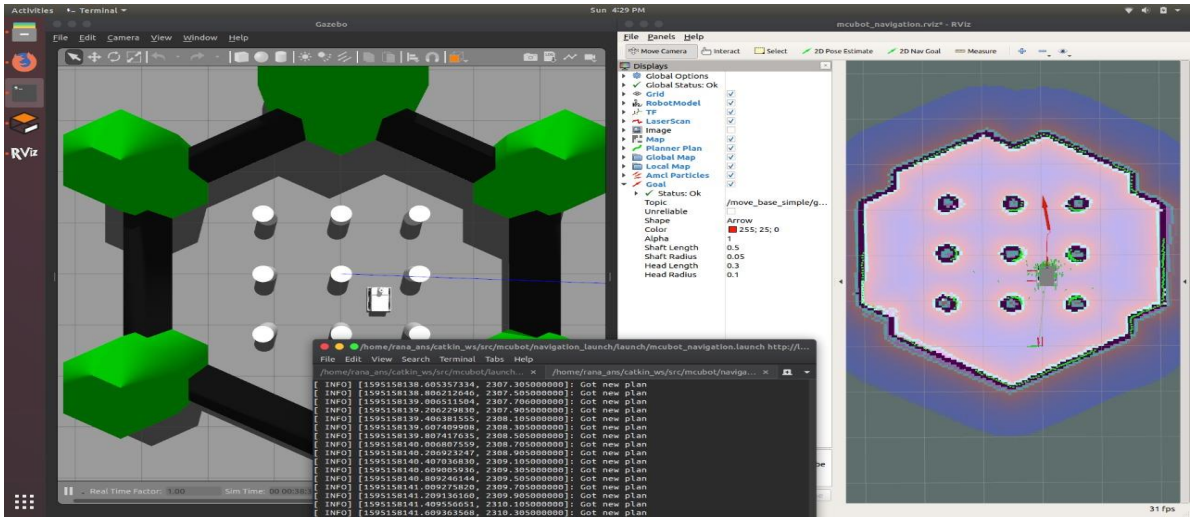


Figure 39 Going towards goal

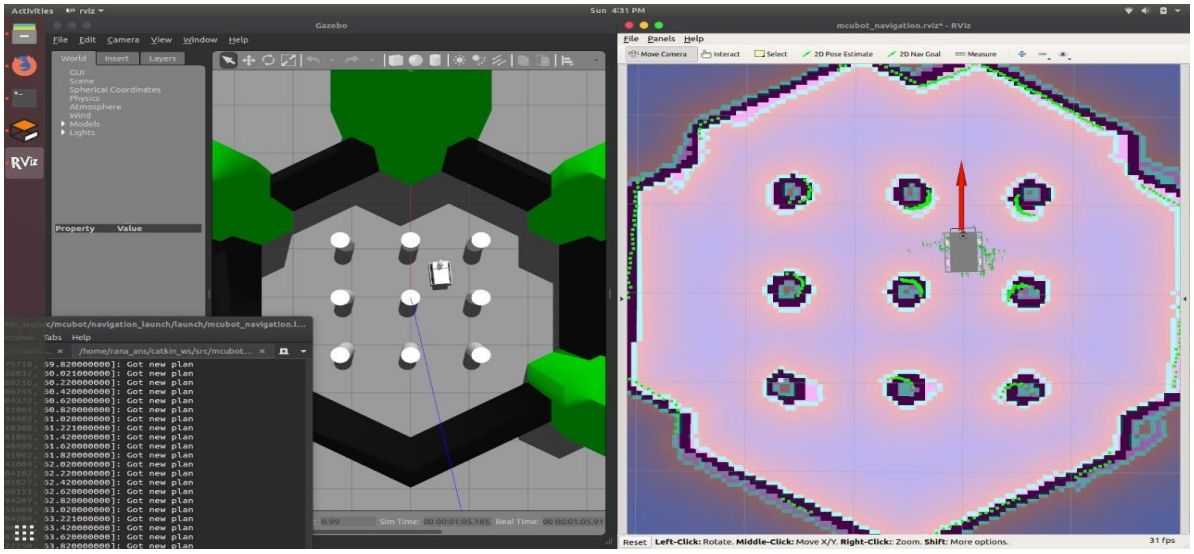


Figure 40 Goal Reached

Chapter: 04

GENERAL DISCUSSION

4- Chapter 4

GENERAL DISCUSSION

4.1- Social Impact

This project has very great impact socially on the lives of people locally and nationally. In many industries of Pakistan, there is no AGV working in warehouses or other areas. There are labours working in warehouses handling material transportation using fork lifters, which require great care to handle them and to avoid any accident. Instead of all this, there is a good probability of accident happen. This project will be ultimate solution to get rid of this any possibility of accident. AGVs' working in warehouses to handle the transportation will eliminate any source of accident and safety issue as they are exclusively doing all this work without any external assistance. It will increase the production capacity, as they will be more efficient and time saver. They will transport all the material without any mistake in less time and there will be more room for more material in the warehouse results in more revenue as there will be no labour, no accident, safer and healthier working environment in warehouses. It can affect positively on the economy of country and revenue of industry. In this way, it will be reducing any harm and provide safe and healthier environment to all the labours working in industry.

4.2- Recommendations

I. One of main recommendation is cost, this project is little bit costly should be cheaper and some items should be replaced and used in such a way that complete project should be more economical.

II. Omni wheels should be used as a replacement of Teflon wheels, which were made by us as they were less expensive as compared to them but Omni wheels enhance the movement performance in intelligent robots.

III. Hardware design of robot should be developed and constructed in such a way it should have a better machining and finish in design of robot.

IV. It is also recommended that the robot design and its circuitry should be more compact and made in such a way that it consists less volume and space.

V. Surface of weight box should be less slippery and so that there will be more friction between weight box surface and weight material so holding strength of surface would be more.

VI. It is recommended that more digital battery who has built in charging system instead of Li-Po battery so that whenever battery strength is low so it can be charged without moving the battery. Voltmeter should be used so that we can see voltages at any time on the moving robot.

These recommendations are given and we did not apply them on real time project owing to some budget problem as we had really confined and tight budget.

Chapter: 05

CONCLUSIONS

5- Chapter 5

CONCLUSIONS

5.1- Conclusions

Autonomous Robot has been demonstrated keeping in view that it meets with every one of the prerequisites to do movement arranging with ROS. The knowledge of the independent robot is viewed as dependent on the utilization of a computational Intelligence calculation utilized for the enhancement of the way arranging. It has been demonstrated keeping in view the ongoing compels. The world utilized for the motion planning is static and meets with the real time physical parameters that are given by Gazebo. We have done mapping manually and using SLAM. We have executed the navigation on our Robot using A* Algorithms and provide a way to other students that are using ROS to make their own robot and implement all the algorithms on it and achieve autonomous navigation on the Robot of desire specifications.

Chapter: 06

REFRENCES

6- Chapter 6

REFERENCES

6.1- References

[1] Mohan Kumar N; Shreekanth T; Sandeep B. "Autonomous Robot Based on Robot Operating System (ROS) for Mapping and Navigation." Mohan Kumar N et al. / International Journal of Computer Science & Engineering Technology (IJCSET) July 7, 2015.

[2] Dr. Nilufer Onder; Dereck Wonnacott; Matias Karhumaa; James Walker, J. "AUTONOMOUS NAVIGATION PLANNING WITH ROS." A Research Journal of MICHIGAN TECHNOLOGICAL UNIVERSITY 2012.

[3] Pranav Reddy Kambam1, Rahul Brungi, Prof. Gopichand G, "Artificial Intelligence in Robot Path Planning," IOSR Journal of Computer Engineering, pp. 115-119,2010

[4] Sador T. Brassaiab, Barna Iantovicsb, "Artificial Intelligence in the path planning optimization of mobile agent navigation," Procedia Economics and Finance, pp 243- 250,2012

[5] Ismail Altaharwa Alaa F. Sheta Mohammed Alweshah, "A Mobile Robot Path Planning Using Genetic Algorithm in Static Environment," Journal of Computer Science, 2008

[6] Chih-Lyang Hwang, Member, IEEE, and Li-Jui Chang, "Internet-Based Smart-Space Navigation of a Car-Like Wheeled Robot Using Fuzzy-Neural Adaptive Control," IEEE Transactions on Fuzzy Systems, pp: 1271 – 1284,2008 [5] Canberk Suat Gurel, Rajendra Mayavan Rajendran Sathyam, Akash Guha, "ROSbased Path Planning for Turtlebot Robot using Rapidly Exploring Random Trees (RRT*)," IEEE T IND ELECTRON, May 2018

[7] Akshay Kumar Guruji, Himansh Agarwal, Deep Parsediya, "Time-efficient A* Algorithm for Robot Path Planning," 3rd International Conference on Innovations in Automation and Mechatronics Engineering, 2016

[8] Pablo Marin-Plaza, Ahmed Hussein, David Martin, and Arturo de la Escalera, "Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles," Journal of Advanced Transportation, Vol. 2018

[6] http://wiki.ros.org/

[9] http://gazebosim.org/

[10] http://learn.turtlebot.com

[11] http://planning.cs.uiuc.edu/node4.html

[12] https://arxiv.org/ftp/arxiv/papers/1401/1401.0889.pd

Chapter: 07

APPENDIX

7- Chapter 7

APPENDIX

7.1- Appendix A

Dijkstra's Algorithm

```
1  function Dijkstra (Graph, source):
2
3    create vertex set Q
4
5    for each vertex v in Graph:           // Initialization
6      dist[v] ← INFINITY                  // Unknown distance from source to v
7      prev[v] ← UNDEFINED                 // Previous node in optimal path from source
8      add v to Q                          // All nodes initially in Q (unvisited nodes)
9
```

```
10   dist[source] ← 0                      // Distance from source to source
11
12   while Q is not empty:
13     u ← vertex in Q with min dist[u]    // Node with the least distance
14                                         // will be selected first
15     remove u from Q
16
17     for each neighbor v of u:           // where v is still in Q.
18       alt ← dist[u] + length (u, v)
19       if alt < dist[v]:                 // A shorter path to v has been found
20         dist[v] ← alt
21         prev[v] ← u
22
23   return dist[], prev[]
```

```
 1  S ← empty sequence
4  u ← target
5  if prev[u] is defined or u = source:                  //Do something only if the vertex is reachable
4      while u is defined:                               // Construct the shortest path with a stack S
5          insert u at the beginning of S                // Push the vertex onto the stack
6          u ← prev[u]                                   // Traverse from target to source
```

Graphical Representation:

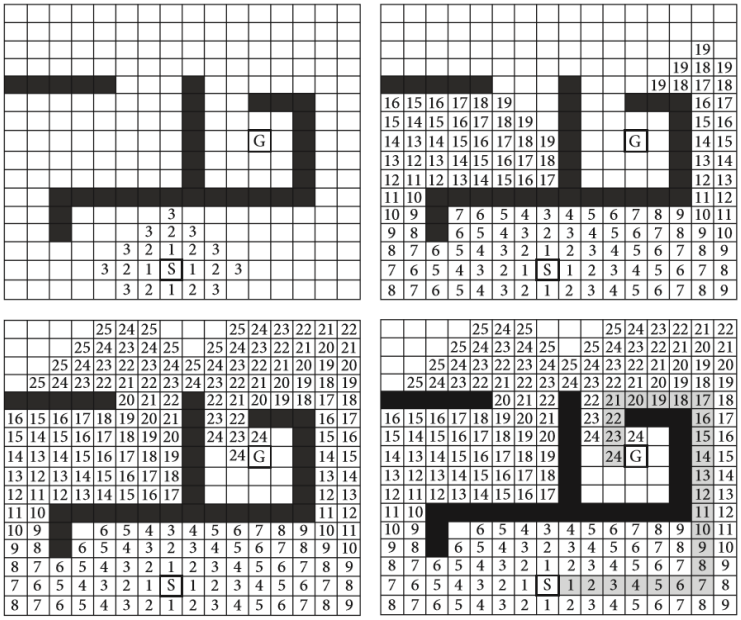


Figure 41 Dijkstra Method to find optimal path

A Star Algorithm:

The following code describes the A* algorithm:

```
1 function Dijkstra (Graph, source):
function reconstruct path (came From, current)
   total path: = {current}
   while current in cameFrom.Keys:
      current: = cameFrom[current]
      total_path.append(current)
   return total_path

function A_Star (start, goal)
   // The set of nodes already evaluated
   closedSet: = {}

   // The set of currently discovered nodes that are not evaluated yet.
   // Initially, only the start node is known.
   openSet: = {start}

   // For each node, which node it can most efficiently be reached from.
   // If a node can be reached from many nodes, cameFrom will eventually contain the
   // most efficient previous step.
   cameFrom: = an empty map

   // For each node, the cost of getting from the start node to that node.
   gScore: = map with default value of Infinity

   // The cost of going from start to start is zero.
   gScore[start]: = 0

   // For each node, the total cost of getting from the start node to the goal
   // by passing by that node. That value is partly known, partly heuristic.
   fScore: = map with default value of Infinity

   // For the first node, that value is completely heuristic.
   fScore[start]: = heuristic_cost_estimate(start, goal)

   while openSet is not empty
```

```
current: = the node in openSet having the lowest fScore[] value
if current = goal
   return reconstruct_path (cameFrom, current)

openSet.Remove(current)
closedSet.Add(current)

for each neighbor of current
   if neighbor in closedSet
      continue                    // Ignore the neighbor which is already evaluated.

   // The distance from start to a neighbor
   tentative_gScore: = gScore[current] + dist_between(current, neighbor)

   if neighbor not in openSet// Discover a new node
      openSet.Add(neighbor)
   else if tentative_gScore >= gScore[neighbor]
      continue;

   // This path is the best until now. Record it!
   cameFrom[neighbor]: = current
   gScore[neighbor]: = tentative_gScore
   fScore[neighbor]: = gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)
```

Example:

An example of an A* algorithm in action where nodes are cities connected with roads and h(x) is the straight-line distance to target point:

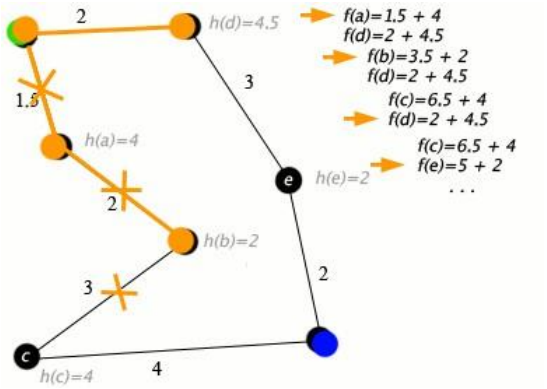


Figure 34: Example

Key: green: start; blue: goal; orange: visited

Animation

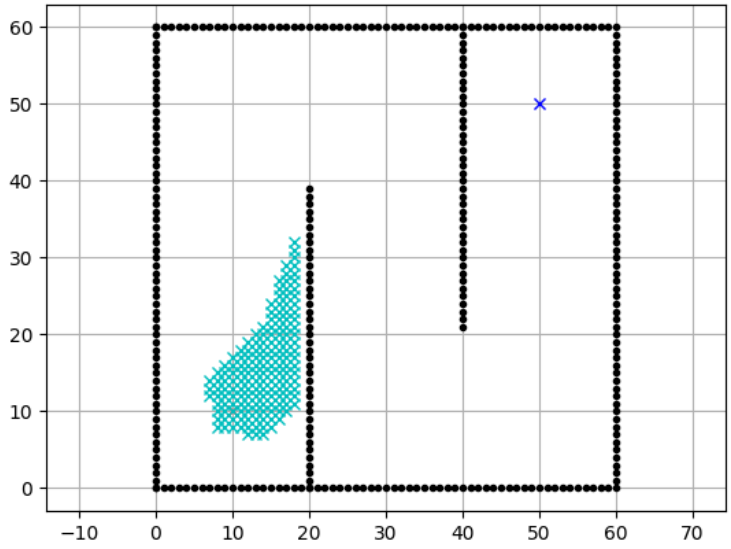


Figure 35: Before

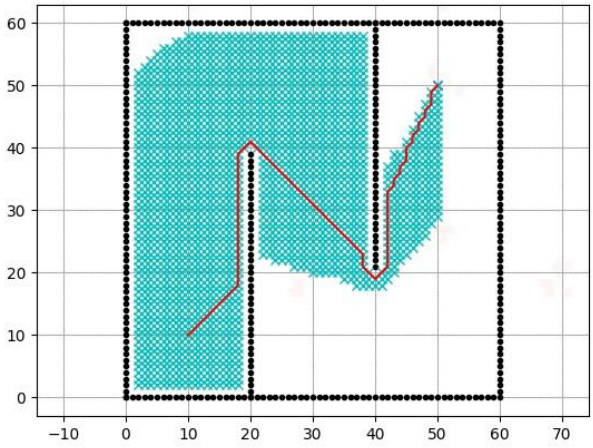


Figure 36: After