The Egyptian E-Learning University

College of Computers and Information Technology

Information Technology Department

Robot Serving in Hospital (Nurse)

Presented by:

Shams Yusef Ibrahim Muhammed

Amal Mahdy Ahmed

Nourhan Ahmed Mohamed Mohamed Ali

Domadios Hany Samir

Khaled Mahmoud Muhammed El-Taher

Mahmoud Abdelaziim Morsy

Mahmoud Zaker Ahmed Zaker

Supervised by:

Asst. Prof. Dr. Mahmoud Muhammed Hussein

TA. Muhammed Fekry

ACKNOWLEDGMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

We highly indebted to (Dr. Mahmoud Muhammed Hussein) for the guidance and constant supervision as well as for providing necessary information regarding the project & also for the support in completing the project.

I would like to express my gratitude to wards member of (The Egyptian E-Learning University) for the kind co-operation and encouragement which help us in completion of this project.

I would like to express my special gratitude and thanks to supervisor`s persons for giving me such attention and time.

My thanks and appreciations also go to my colleagues in developing the project and people who have willingly help me out with their abilities.

Table of Contents

Table of Contents

Abstract

In the face of increasing demands on healthcare systems, particularly within hospital settings, the integration of robotic technology presents a promising solution to enhance patient care and operational efficiency. This graduation project explores the development and implementation of a nurse robot designed to assist medical staff in a hospital environment. The nurse robot is engineered to perform a variety of tasks which reducing the workload on human nurses and improving overall healthcare delivery. However, developing an autonomous system can be more efficient when done in the Robot Operating System (ROS) framework and using a modular robot design. In ROS, path planning and navigation within a target hospital can efficiently done using digital map of the environment or that generated by SLAM algorithm; with the Adaptive Monte-Carlo (AMCL) in ROS, the position of the robot can be determined and path planning can be done. Preliminary tests show that autonomous navigation is achieved, and the robot is able to dock at a target room and achieve goals.

List of Abbreviations

ROS → The Robot Operating System

RViz → ROS 3D Robot Visualizer

RQT → Recently Qualified Teacher

SLAM → Simultaneous Localization and Mapping

MCL → Monte Carlo Localization

URDF → Unified Robotics Description Format

Xacro → XML Macro Language

AMCL → Adaptive Monte Carlo Localization

XML → Extensible Markup Language

UML → Unified Modeling Language

ERD → Entity Relationship Diagram

1

Introduction to Project

1.1 Introduction

Robot service in hospitals refers to the use of robots to perform various tasks such as monitoring patients, delivering medications, and interacting with patients. The popularity of robots in healthcare is growing due to their efficiency, accuracy, and ability to work continuously without fatigue or errors.

One example of such technology is the Robot Operating System (ROS), an open-source robotics framework that offers tools and libraries for building robotic applications. ROS is ideal for developing different types of robots that can automate hospital operations and enhance patient care. A notable application is the nurse robot, which operates using ROS to manage tasks like locating the robot, navigating to the target patient, and delivering medications.

The implementation involves various process modules or nodes in ROS. These processes are designed to:

1. Locate the robot and the target patient.
2. Navigate the robot to the patient.
3. Dock and deliver medications to patients.

Healthcare robots are revolutionizing the industry, promising a future where these machines are commonplace. The primary goals of implementing such technologies are to reduce costs and increase efficiency. Our robot excels in delivering medications swiftly and efficiently, outperforming human nurses in speed and reliability.

Benefits of the proposed model:

1. Reduces labor requirements.
2. Reduces wait time for patients.
3. Longer working hours than humans.
4. Saves money and human resources.
5. Reduces operating costs.

1.2 Motivation

- **Improved efficiency:** A nurse robot can work continuously without getting tired or needing breaks, which can enhance the speed and efficiency of healthcare services.
- **Cost-effective**: In the long run, investing in a nurse robot can be more cost-effective than hiring additional human staff, as robots do not require salaries, benefits, or overtime pay.
- **Consistency:** A nurse robot can provide consistent service every time, ensuring that patients receive the same level of care regardless of shifts or staff changes.
- **Enhanced patient experience:** The novelty of having a nurse robot can enhance the overall experience for patients, making their hospital stay more engaging and futuristic.
- **Customization:** ROS provides a flexible platform for developing customized applications for robots, allowing hospitals to tailor their nurse robots to specific needs and preferences.

Overall, using ROS for a nurse robot in a hospital has the potential to improve efficiency, reduce costs, enhance patient experience, and provide customization options.

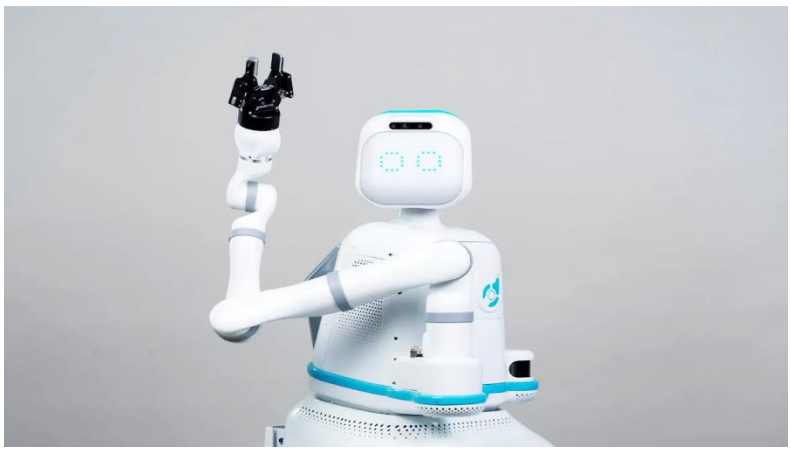


Figure 1 Nurse Robot

1.3 Problem Definition

The healthcare industry is facing a shortage of skilled nursing staff, leading to increased labor costs and decreased efficiency in patient care. To address this issue, some healthcare facilities are turning to robots to assist with nursing tasks, such as monitoring patient vital signs, administering medication, and providing basic patient care. However, there are concerns about the effectiveness and reliability of these robots in delivering quality care and meeting patient expectations. Additionally, there may be resistance from patients and staff who prefer human interaction in their healthcare experience. The challenge is to find a balance between utilizing technology to improve efficiency and maintaining a high level of patient satisfaction.

1.4 Issues

Here are some potential issues that could arise with the use of robots in nursing:

- **Technical Difficulties:** Nursing robots can malfunction or break down, causing delays in patient care and potentially compromising patient safety and well-being.
- **Limited Capabilities:** Robots may not be able to perform all tasks required in a healthcare setting, such as responding to complex patient needs, making nuanced clinical decisions, or handling emergencies.
- **Lack of Personal Touch:** Some patients may prefer human interaction and personalized care over interactions with robots, which can impact their overall satisfaction and emotional well-being.
- **Cost:** Implementing robotic systems in healthcare can be expensive, and the cost may not be feasible for smaller or underfunded facilities.
- **Job Displacement:** The use of robots in healthcare could lead to job loss for human workers, which could have negative social and economic impacts, including loss of livelihood for displaced workers.

1.5 Objectives

- **Enhance patient care experience:** The primary objective of using nurse robots is to enhance the overall patient care experience. Robots can provide timely and efficient assistance, reducing wait times for routine tasks and improving the quality of care.

- **Increase efficiency:** Nurse robots can perform repetitive tasks such as monitoring vital signs, delivering medication, and transporting supplies, freeing up human nurses to focus on more complex tasks such as patient assessments and personalized care.
- **Reduce labor costs:** By using robots for certain tasks, healthcare facilities can reduce their labor costs and allocate resources more effectively, potentially increasing profitability or enabling better allocation of funds towards patient care.
- **Improve accuracy:** Nurse robots are programmed to perform tasks with a high degree of accuracy, reducing errors in medication administration and other routine procedures, thus improving patient safety and the overall quality of care.
- **Provide consistent care:** The use of nurse robots can ensure that certain aspects of patient care are consistently performed to high standards, helping to maintain a uniform level of care across the facility.
- **Increase safety:** In the context of infectious diseases, such as during the COVID-19 pandemic, nurse robots can help reduce contact between patients and healthcare staff, increasing safety for both patients and caregivers.
- **Collect data:** Nurse robots can collect data on patient health metrics and behavior, providing valuable insights that can be used to improve care plans, identify trends, and inform medical research and healthcare strategies.

2

Background

2.1 What is ROS?

2.2 Various development tools

2.3 ROS Main Features

2.4 The technique used to Create Map by ROS

2.5 The technique used to Make Navigation by ROS

2.1 What Is ROS?

2.1.1 Definition:

- Is an open-source, meta-operating system for your robot.
- It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.
- It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.
- Ros is broken down into over 2000 packages, each package providing specialized functionality. Perhaps the number of tools connected to the framework is its greatest strength.
- Not an actual operating system, but is a framework and set of tools that provide the functionality of the operating system on a heterogeneous computer mass. Its usefulness is not limited to bots. But the majority of the tools focus on working with peripheral devices.

Figure 2 ROS

2.1.2 Software Framework for Developing Robot Software:

- It is possible to jointly develop complex programs by finely dividing them.
- With message exchanging method between nodes.
- Supports command tool, visualization tool RVIZ, GUI toolbar RQT, 3D.
- Simulator Gazebo.
- Supports modeling, sensing, recognition, navigation, and manipulation.
- Functions commonly used in robotics.
- Create Robotics Ecosystem!

2.1.3 ROS Is Meta-Operating System?

1) Operating System:

i. General Purpose Computer:

- Windows (Windows XP, 7, 8 …)
- Linux (Ubuntu, Red hat, Fedora, Mint, Gentoo …).
- MAC (OS X …) etc.

ii. Smart Phone:

- Android, IOS, Windows Phone, Symbian, Tizen etc.

2) ROS:

- ROS ➡ Robot Operating System.
- ROS is Meta-Operating System.
- The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

2.1.4 Meta-Operating System?

- It is not a traditional operating system like Windows, Linux, and Android. Rather, ROS uses the traditional operating system (Linux, Windows, OS-X, and Android).
- It uses the existing operating system's process management system, file system, user interface, program utilities (compiler, thread model, etc..)

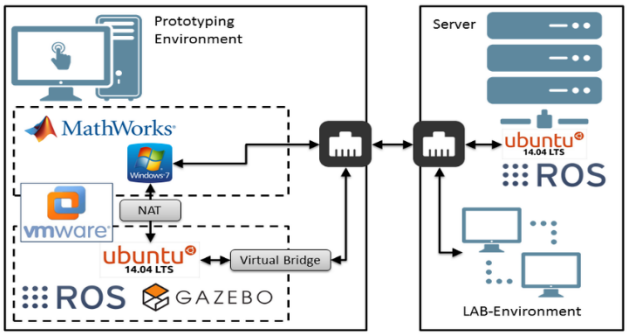


Figure 3 Meta-Operating system

- In addition, it develops, manages and provides various application programs based on the robot software framework, and has an ecosystem that distributes packages developed by users.
- In addition, it provides essential functions for developing robot application software such as data transmission / reception, scheduling and error handling among many different types of hardware in a library form.

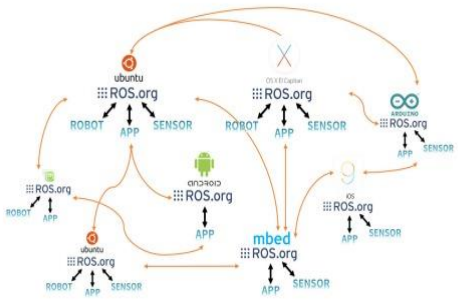


Figure 4 Ros Multi _ Communication

2.1.5 Configuring the ROS Development Environment:

- **ROS Installation:**

  Http://wiki.ros.org/noetic/Installation/Ubuntu

- **ROS Environment Setting:**

  Http://wiki.ros.org/ROS/Tutorials/installingandconfiguringrosenvironment

- **ROS Operation Test:**
  - **Roscore**
  - **Rosrun turtlesim turtlesim_node**
  - **Rosrun turtlesimturtle_Teleop_key**
  - **Rosrun rqt_graph rqt_graph**

2.2 Various Development Tools

- Provides various development tools needed for robot development.
- Improving the efficiency of robot development.

1) **Command-Line Tools:**

   Robot access only with commands provided by ROS without GUI and Use almost all ROS features.

2) **RVIZ:**

   - Provides powerful 3D visualization tool.
   - Visualizes sensor data such as laser, camera, etc.
   - Represents robot configuration and planned motion.

3) **RQT:**

   - Provides Qt-based framework for developing graphic interface.
   - Displays nodes and connection information between them (rqt_graph).
   - Floats encoder, voltage, or number that changes over time (rqt_plot).
   - Records data in message form and play back (rqt_bag).

4) **Gazebo:**

   - 3D simulator which supports physics engine, robot, sensor, environmental model, etc.
   - High compatibility with ROS.

1) 3D Visualization Tool (RVIZ):

- RVIZ is the 3D visualization tool of ROS. The main purpose is to show ROS messages in 3D, allowing us to visually verify data. For example, it can visualize the distance from the sensor of a Laser Distance Sensor (LDS) to an obstacle, the Point Cloud Data (PCD) of the 3D distance sensor such as realsense, Kinect, or Xtion, the image value obtained from a camera, and many more without having to separately develop the software.
- It also supports various visualization using user specified polygons, and Interactive Markers allow users to perform interactive movements with commands and data received from the user node.
- In addition, ROS describes robots in Unified Robot Description Format (URDF), which is expressed as a 3D model for which each model can be moved or operated according to their corresponding degree of freedom, so they can be used for simulation or control.
- The mobile robot model can be displayed, and received distance data from the Laser Distance Sensor (LDS) can be used for navigation.
- RVIZ can also display the image from the camera mounted on the robot as shown in the lower-left corner.
- In addition to this, it can receive data from various sensors such as Kinect, LDS, realsense and visualize them in 3D.

2) RVIZ Installation and Test:

- **RVIZ Installation**
  $sudo apt-get install ros-kinetic-rviz
- If you installed “ros-kinetic-desktop-full, Rviz will be installed by default
- **Run RVIZ**
  $rosrun rviz rviz **Or** $rviz

Figure 5 RVIZ

3) Gazebo (3D Simulator):

- Gazebo is 3D Simulator with Physics Engine, Robot model, Sensor, Environment model, and so on. It helps you to get data similar to the one in real environment.
- Gazebo is regarded as the best simulator among recently-introduced Open Simulator. In addition, it is selected as an official simulator for DARPA Robotics Challenge.
- It is highly compatible with ROS.

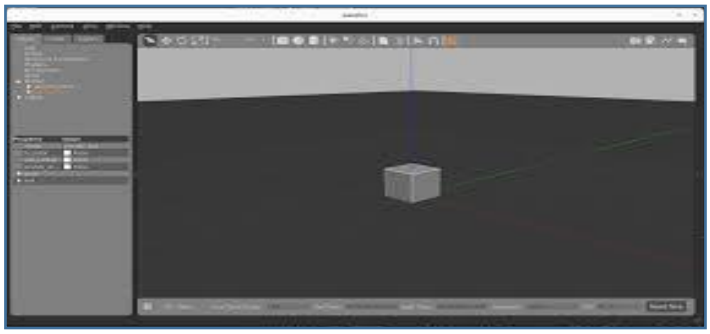


Figure 6 Gazebo

4) ROS GUI Development Tool RQT):

- Besides the 3D visualization tool RVIZ, ROS provides various GUI tools for robot development. For example, there is a graphical tool that shows the hierarchy of each node as a diagram thereby showing the status of the current node and topic, and a plot tool that schematizes a message as a 2D graph.
- Starting from the ROS Future version, more than 30 GUI development tools have been integrated as the tool called rqt6 which can be used as a comprehensive GUI tool.
- Furthermore, RVIZ has also been integrated as a plugin of RQT, making RQT an essential GUI tool for ROS.
- Not only that, but as the name suggests, rqt was developed based on Qt, which is a cross platform framework widely used for GUI programming, making it very convenient for users to freely develop and add plugins.
- In this section we will learn about the ‘rqt’ plugins ‘rqt_image_view’, ‘rqt_graph’, ‘rqt_plot’ and ‘rqt_bag’.

- **RQT Installation and Test:**
- **RQT Installation**

  $sudo apt-get install ros-kinetic-rqt ros-kinetic-rqt-common-plugins

- **RQT Run** $rqt

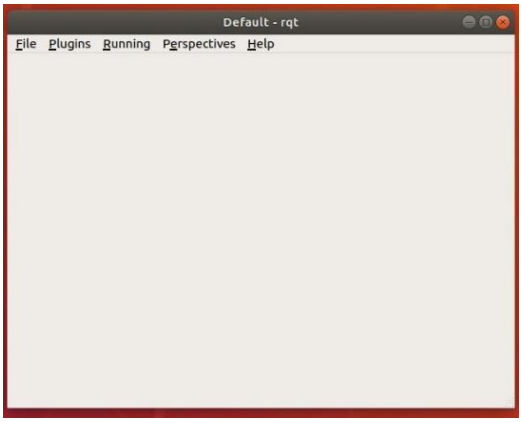


Figure 7 RQT

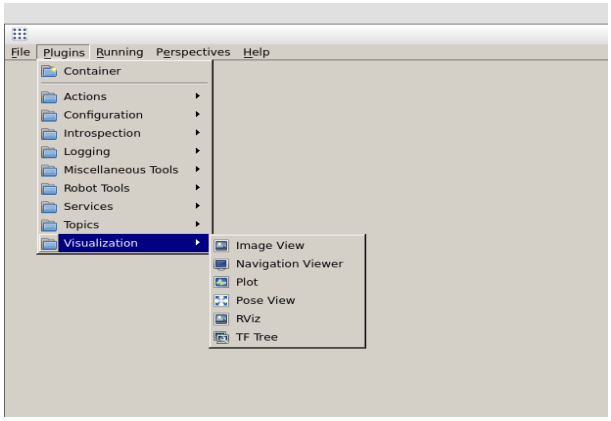


Figure 8 RQT Installation

- **When to Use RVIZ, RQT, Gazebo?**

RVIZ	Sensor and Robot Related Data Visualization becomes Very Simple.
RQT	ROS available in GUI form. Easy to create GUI Tool!
Gazebo	If you need Simulation, It is easy to work with ROS & Gazebo!!

Table 1 RVIZ RQT, Gazebo

2.3 ROS Main Features

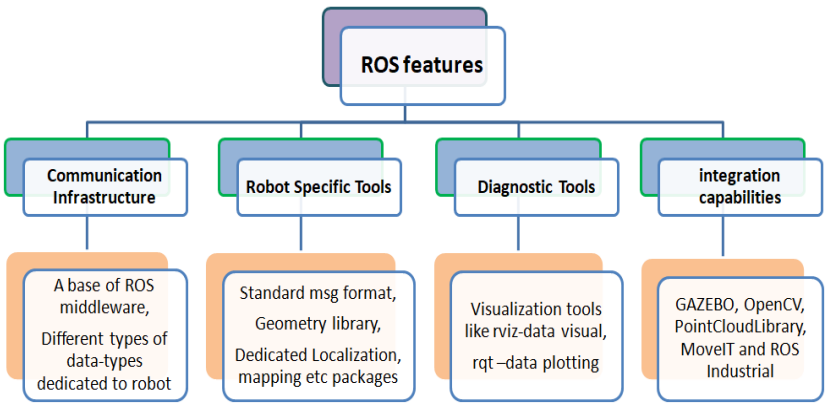


Figure 9 ROS Main Features

- **SLAM And Navigation:**

SLAM:

(Simultaneous Localization and Mapping) is a technique used in robotics to create a map of an unknown environment while simultaneously keeping track of the robot's location within that environment. It involves using sensors such as cameras, lidars, and sonars to collect data about the surroundings and then using algorithms to process that data into a map.

Navigation: In ROS (Robot Operating System) refers to the ability of a robot to move from one location to another within its environment. ROS provides various navigation packages that can be used for path planning, obstacle avoidance, and localization. These packages use SLAM techniques to create maps of the environment and then use those maps for navigation.

- **What You Need for Path Finding!**

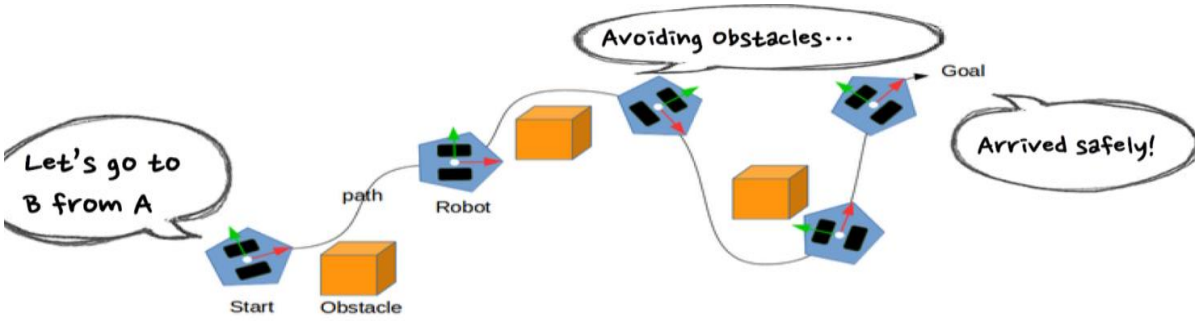


Figure 10 Path

1) **Position:** Measuring/estimating the robot's position.
2) **Sensing:** Measuring obstacles such as walls and objects.
3) **Map:** Maps with road and obstacle information.
4) **Path:** Calculate optimal path to the destination and follow the path.

1) Position:

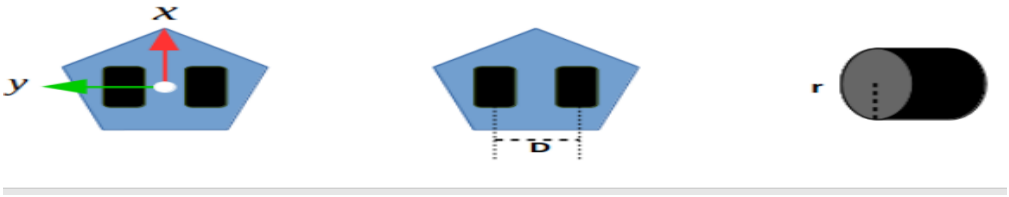


Figure 11 Position

- **Required information:**

1) Encoder value E on both wheel axes (Recalculated as gear ratio for motor shaft).
2) Distance between wheels D.
3) Wheel radius r.

2) Sensing:

Figure 12 Sensing

i. Distance Sensor:
   - Ultrasonic sensor.

ii. Depth camera:
   - Swiss Ranger, Kinect-2
   - Realsense, Kinect, Xtion, Carmine (Prime Sense), Astra.

3) Map:

- Robots need a map to find a path!
- Digital maps for infrastructure such as roads!
- Maps of hospitals, cafes, companies home.
- Maps of unknown, collapsed hazardous areas?
- SLAM :(Simultaneous Localization and Mapping).

Figure 13 Map

4) Path:

1) Navigation.
2) Localization / Pose estimation.
3) Path search and planning.
4) Dynamic Window Approach (DWA).
5) A* algorithm (A Star).
6) Potential Field.
7) Particle Filter.
8) Graph.

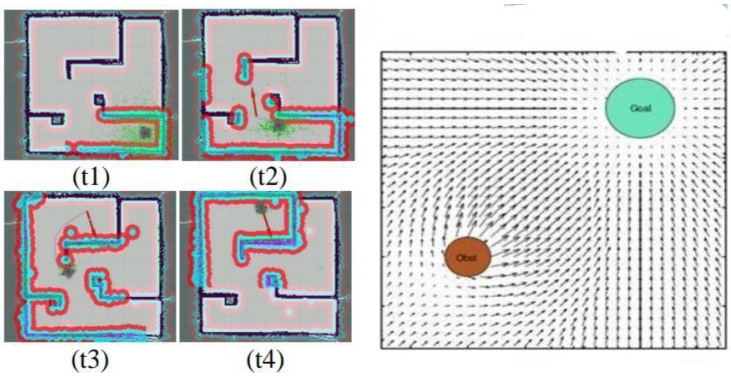


Figure 14 Path1

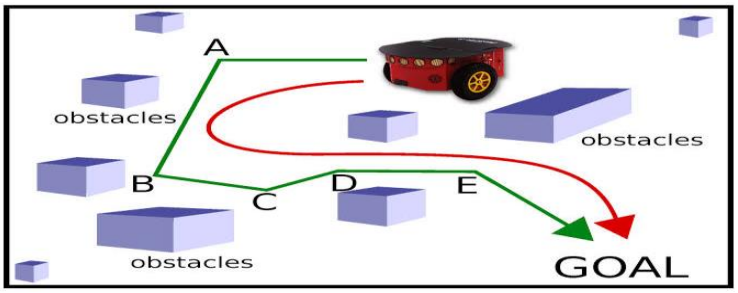


Figure 15 Path2

2.4 The Technique Used to Create Map by ROS?

Position + Sensing ⟶ Map

SLAM

Mapping in robotics involves using sensor data (such as lidar or camera data) to create a representation of the environment around the robot. This representation can be in the form of a 2D or 3D map, and can be used for various tasks such as localization and path planning.

1) The process of creating a map typically involves several steps:
2) Data collection: The robot collects sensor data as it moves through the environment.
3) Data processing: The raw sensor data is processed to remove noise and extract relevant features (such as walls or obstacles).

4) Map creation: The processed sensor data is used to create a map of the environment.
5) Map refinement: The map may be refined over time as more data is collected and processed.

ROS provides several packages for mapping, such as G-mapping and cartographer, which implement different mapping techniques such as grid-based mapping and SLAM (Simultaneous Localization and Mapping). These packages typically provide APIS for integrating with various sensors and robot platforms, making it easier for developers to implement mapping functionality in their robots.

2.5 The Technique used to Make Navigation by ROS?

Position + Sensing + Map ⟶ Path

Navigation

- Navigation by ROS (Robot Operating System) is a software framework used for building robotic systems. It provides a set of tools and libraries for developing robot applications. The navigation stack in ROS is used for autonomous navigation of mobile robots.
- The technique used in Navigation by ROS involves several components such as sensors, localization, mapping, and path planning. The sensors are used to gather information about the environment such as laser range finders, cameras, and odometry sensors. Localization is the process of determining the robot's position and orientation in the environment using sensor data. Mapping involves creating a map of the environment using sensor data. Path planning is the process of finding a path from the robot's current position to its destination.
- The navigation stack in ROS uses a probabilistic approach called Monte Carlo Localization (MCL) for localization and Simultaneous Localization and Mapping (SLAM) for mapping. MCL uses particle filters to estimate the robot's position based on sensor data. SLAM uses sensor data to create a map of the environment while simultaneously localizing the robot within that map.

- Once localization and mapping are complete, path planning algorithms such as A* or Dijkstra's algorithm are used to find an optimal path from the robot's current position to its destination.
- Overall, Navigation by ROS uses a combination of sensor data processing, probabilistic algorithms, and path planning techniques to enable autonomous navigation of mobile robot.

3

Methodology

3.3 URDF

3.4 Particle Filter and Localization

3.1 URDF

3.1.1 Definition URDF:

- The Unified Robot Description Format (URDF) is an XML specification to describe a robot.
- He main limitation at this point is that only tree Structures can be represented, ruling out all parallel robots.
- Flexible elements are not supported.
- The specification assumes the robot consists of rigid links connected by joints.

- **The Specification Covers URDF:**

1) Kinematic and dynamic description of the robot.
2) Visual representation of the robot.
3) Collision model of the robot.

- **The Description Of a Robot Consists Of:**

1) A set of link elements.
   <link>……..</link>

2) A set of joint elements connecting the links together.
   <joint>……..</joint>

Note: You can see that the root element of the URDF format.

Is a <robot>element.

<robot name="pr2">

<link> ….. </link>
<link> ..... </link>
<link> ..... </link>
<link> ..... </link>
<joint> .... </joint>
<joint> ..... </joint>
<joint> ..... </joint>

</robot>

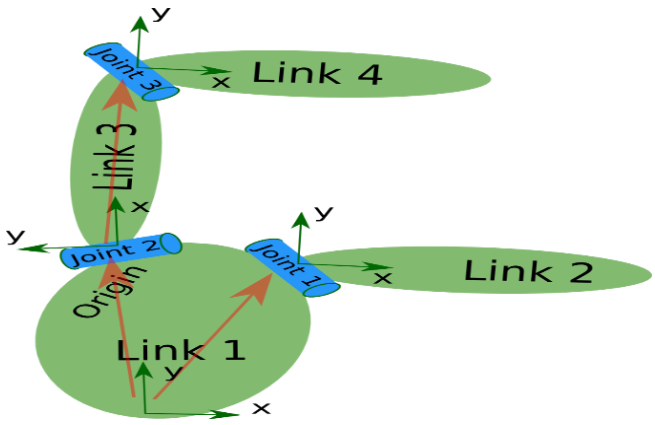


Figure 16 Description of robot

- **The URDF Syntax:**

1) URDF is based on XML, so everything is represented as a series of tags, which can be nested.

2) There are many different tags we can use, but there are three main ones we need to **know about:**
   - The robot tag and XML declaration
   - A proper XML file should have an XML declaration/prolog in the first line, and then after that will be a single tag (called the root tag), which ALL the other tags live inside of. For a URDF file, this root tag will be the robot tag, and the only thing to note here for now is that we can set the name attribute which lets us (unsurprisingly) specify the name of our robot.

3.1.2 The Link Element:

- **The link element describes a rigid body:**

1) Elements(tags):

1. Inertia.
2. Visual features.
3. And collision properties.

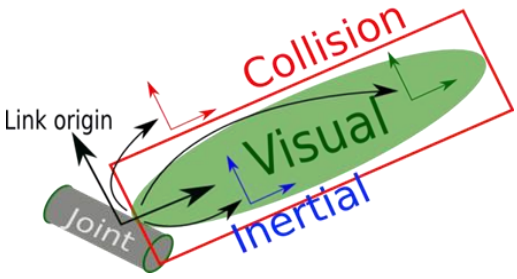


Figure 17 Link element

2) Attributes:

Name (required)The name of the link itself.

3) A link tag(Elements(tags)):

- A link tag lets us firstly specify the name of a link.
- As well as some additional characteristics (the visual, collision, and inertial properties) These additional tags are generally optional.

1. Visual:

i. This is what we see in RVIZ and Gazebo.

ii. We can specify three aspects:

- **Geometry:**

Box/ cylinder / sphere with size parameters, or a mesh.

- **Origin:()**

i. An offset for the geometry so it doesn't need to be centered around the link origin.

ii. **XYZ** (optional: defaults to zero vector) Represents the **x, y, z** offset.

iii. **RPY** (optional: defaults to identity if not specified Represents the fixed axis roll, Pitch and yaw angles in radians.

- **Material:**

i. Basically, the color. We can specify the name of a declared material, or describe the color directly.

ii. **Note:** that this will set the color in rviz but not Gazebo, more on that in the next tutorial).

iii. **Name (optional)**: Specifies a name for a part of a link's geometry. This is useful to be able to refer to specific bits of the geometry of a link.

2. Collision:

- This is used for physics collision calculations.
- We can set the **Geometry** and **Origin**
- Same options as for visual. This will often be copy-pasted from the Visual tag.
- We may want a simpler collision geometry (e.g. Box instead of a mesh) computational reasons.
- Name (optional): Specifies a name for a part of a link's geometry.

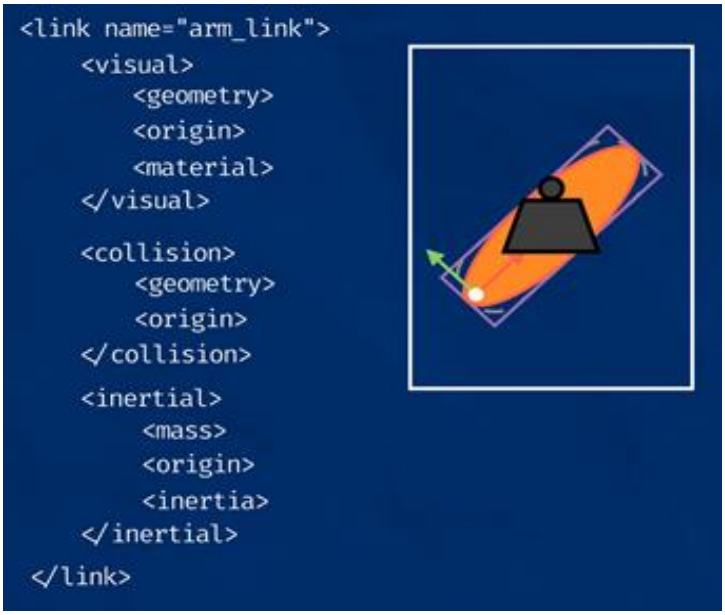


Figure 18 Collision

3. Inertial:

- This is also used for physics calculations, but determines how the link responds to forces.
- **The inertial properties are:**

1) Mass:

i. Mass of the link.

2) Origin:

i. The center of mass (center of gravity).
ii. This is the point the link could "balance" (a slightly more confusing concept in 3D).
iii. For most simple cases this will just be the center (same origin as visual/collision).

3) Inertia:

i. The rotational inertia matrix.
ii. This is probably the most confusing part of this section.
iii. It describes how the distribution of the mass will affect rotation.
iv. A list of the matrices for common shapes is available here.
v. A fairly accurate simulation can be achieved by approximating our links as prisms, ellipsoids, or cylinders, however for most purposes a very rough guess will do.
vi. Also, don't get inertial and inertia mixed up
vii. **The inertia:** matrix is just one part of the inertial properties.

- **Example of Link Element:**

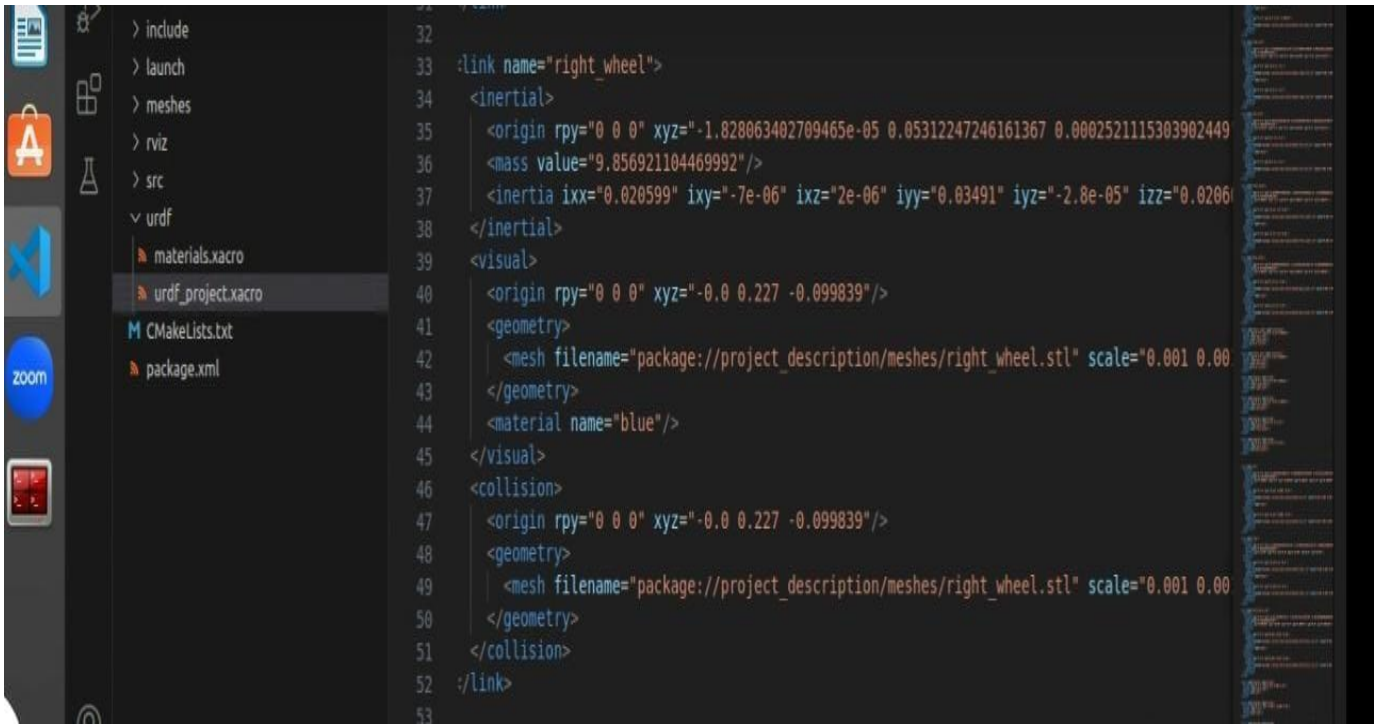


Figure 19 Example Link Element

3.1.3 Joint Element

- The joint element describes the kinematics and dynamics of the joint and also specifies the safety limits of the joint.
- Although we usually think of the robot as being made up of links, the joints are actually where all the detail is in terms of the robot's structure, as they define the link locations, and how they move relative to each other.
- This is similar to the previous tutorial on the TF system, where although we ultimately want to interact with frames, it's actually the transforms that define where the frames are, so it's important to get them right.

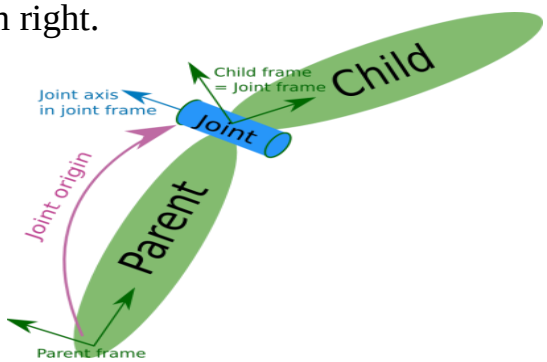


Figure 20 Joint Element

1)Attributes:

- The joint element has two attributes:
  1) Name (required):
  Specifies a unique name of the joint
  2) Type (required):
  - **Revolute:** a hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits.
  - **Continuous:** a continuous hinge joint that rotates around the axis and has no upper and lower limits.
  - **Prismatic:** a sliding joint that slides along the axis, and has a limited range specified by the upper and lower limits.
  - **Fixed**
    - This is not really a joint because it cannot move.
- All degrees of freedom are locked. This type of joint does not require the <axis>, <calibration>, <dynamics>, <limits> or <safety_controller>.
  - **Floating:** this joint allows motion for all 6 degrees of freedom.
  - **Planar:** this joint allows motion in a plane perpendicular to the axis.

Common Joint Types:

Revolute: - A rotational motion, with minimum/maximum angle Limits.
Continuous: - A rotational motion with no limit (e.g. A wheel).
Prismatic: - A linear sliding motion, with minimum/maximum Position limits.
Fixed: - The child link is rigidly connected to the parent link.
What we use for those "convenience" links.

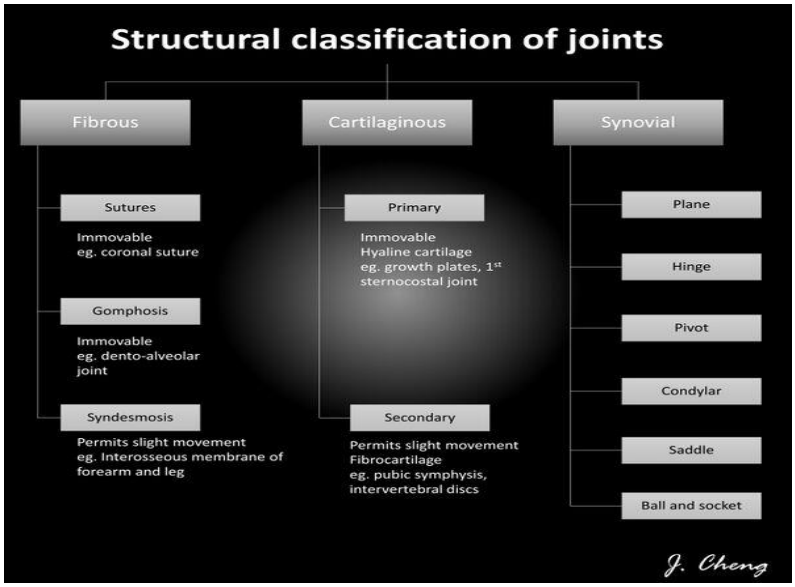


Figure 21 Common Joint Types

3) Elements(tags):

- **Fixed joint these parameter:**

1) **Parent** and **child** links:
   - Which links this joint defines a relationship between.
2) **Origin**: The relationship between the two links, before any motion is applied.

- **Non fixed joint these parameter:**

1) **Axis:** Which axis to move along or around (xyz (required)).
2) **Limits**: Physical actuation limits, which may be expected by other parts of the system.

- These can include:

i. Upper and Lower position limits: in meters/radians.
ii. Velocity limits: in m/s or rad/s.
iii. Effort limits: in N or Nm.

- # Example of Joint Element:

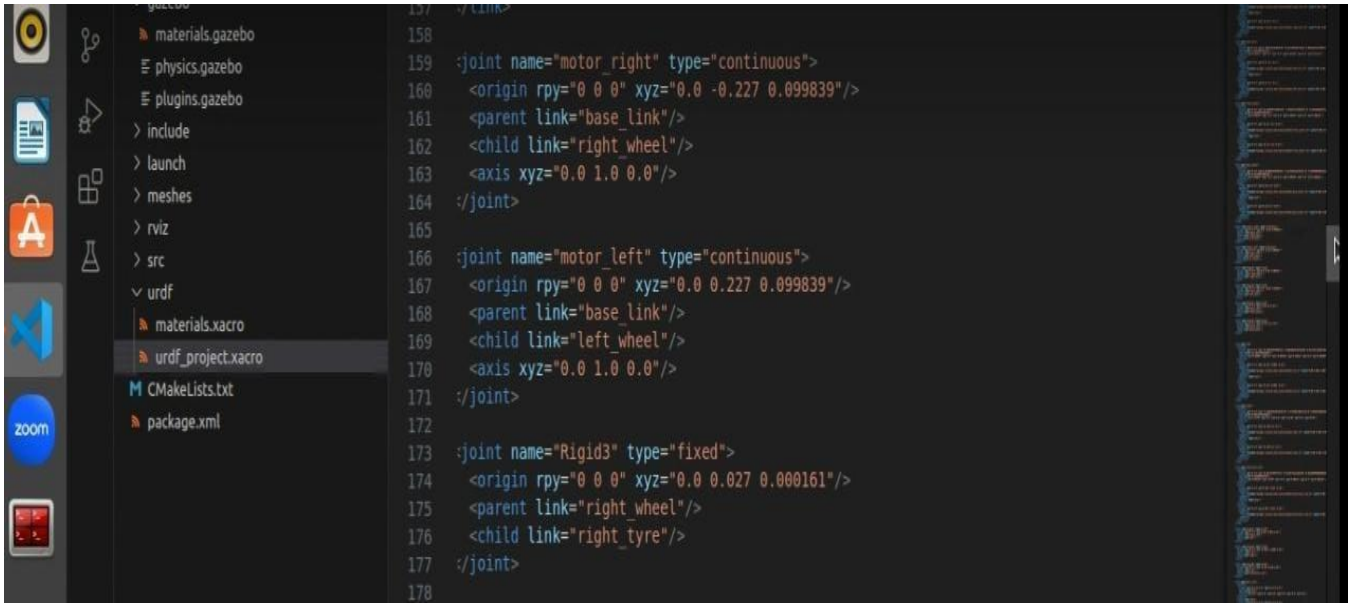


Figure 22 Example Joint Element

3.1.4 Tool Xacro:

- It's worth learning about a tool ROS provides to make them easier to write, called xacro (short for XML macro).
- Xacro lets us manipulate the URDF files in various way.
- But the two main ones we'll look at:
  1) Are splitting up code into multiple files.
  2) And avoiding duplicate code.
- Whenever we want to actually use the URDF we need to run the xacro software over the files first, which will process them into a single, complete URDF "file" (temporary in memory).

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

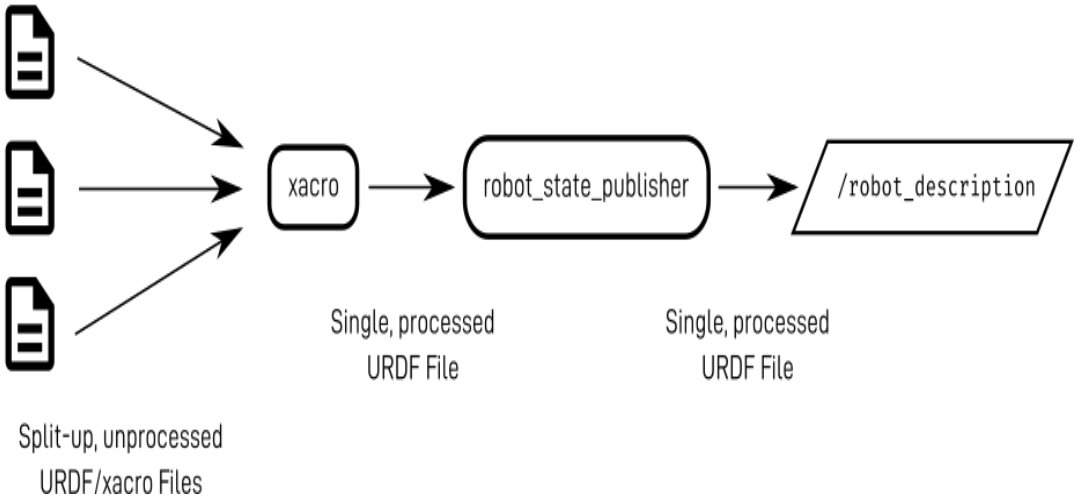


Figure 23 Tool Xacro

1) **Using xacro to split up code:**

- Being able to split up our URDF into multiple files is useful because they can become pretty large and unwieldy.
- Separating different components out makes it easier to find and change things, and also to quickly tell what has been changed when using version control software (like Git).
- How we split up a URDF will depend on the individual developer and their hardware, but one example of the different files could be:
  1) Core robot (links and joints)
  2) List of materials (colours) to use for display.
  3) Sensor links/joints and their corresponding Gazebo tags.
  4) Macros.

- For this to work, the included file has to be a bit special too. Like the main file, it will have a robot tag (with the xmlns:xacro bit), but we don't need to specify a name. Then, we can put whatever tags we want to include inside the robot tag.

  <xacro:include filename="first_file.xacro" />

- We can include as many URDF/xacro files as we like (nested or in series), and we can use includes as well as normal tags.
- Typically, the "main" URDF/xacro has the extension. Urdf.xacro, but the "included" ones can vary a little.
- Sometimes they will just use. Xacro, sometimes they will also be urdf.xacro, and sometimes they will have other extensions.

2) **Using xacro to avoid duplicate code:**

- An important principle in programming is DRY (Don't Repeat Yourself). Having the same procedure or information in multiple places is a problem for a few reasons**, including:**
  1) The more times you need to write it, the more likely you are to do one incorrectly.
  2) When you need to change a process or piece of data, it's a pain to have to do it in multiple locations.

- **Some of the ways Xacro helps us to simplify our URDF, and avoid repeating code or making mistakes are:**

  1) Properties:
  We can declare a property to be some constant and then use it later on.

  <xacro:property name="arm_radius" value="0.5" />`
  <cylinder radius="${arm_radius}" length="7" />

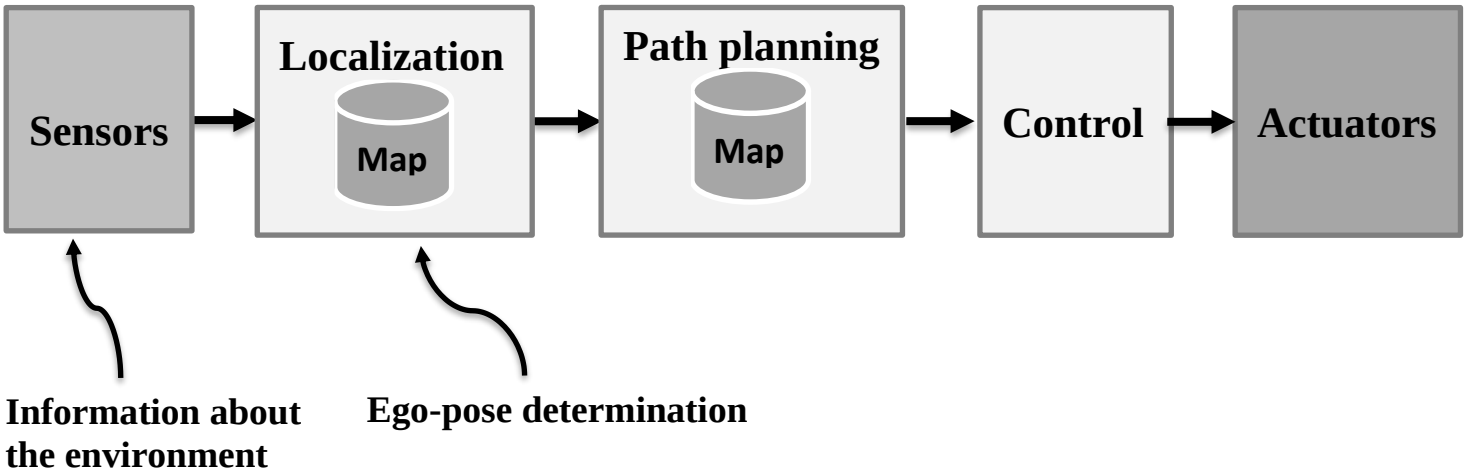
  2) Mathematical Operations:
  We can use mathematical constants and operators inside attributes, which can also Be combined with properties.

  <cylinder length="${4*arm_radius + pi}">

  3) Macros:
  We can create a macro, which is a template that can be reused later, and varies Based on parameter.

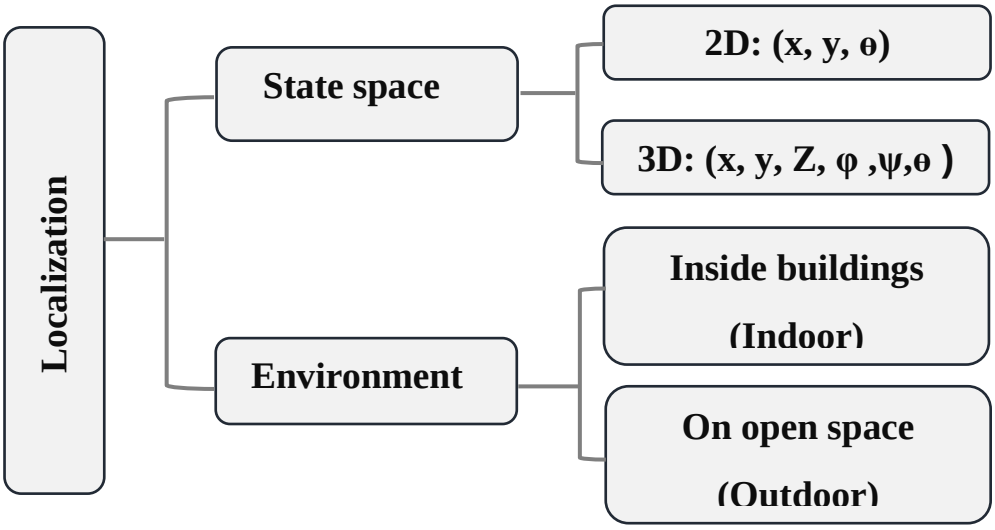
3.2 Particle Filter and Localization

3.2.1 CONTROL SCHEME OF MODERN MOBILE ROBOT



3.2.2 WHAT IS LOCALIZATION?

Localization (in robotics) stands for the process of tracking of robot pose (position and orientation) in space (some fixed reference frame).



3.2.3 WHY LOCALIZATION MATTERS?

Knowing precise pose is a prerequisite for accurate motion planning and execution.

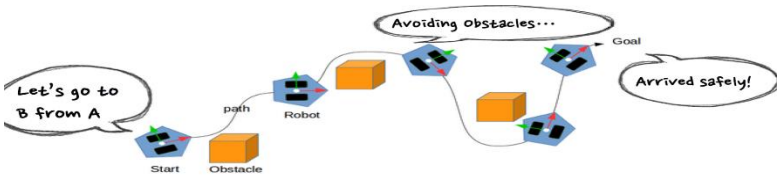


Figure 24 Localization Matters

Figure 25 Localization Matters

3.2.4 FORMAL LOCALIZATION PROBLEM DEFINITION

Given:

$\mathbf{X_1}$**: t–1** - previous states (poses)
$\mathbf{u_1}$**:t** - history of the control signals(action)
$\boldsymbol{Z_1}$**:t** - sensors measurements
Map - map

Find:

$\mathbf{X_t}$ - current robot states (pose)

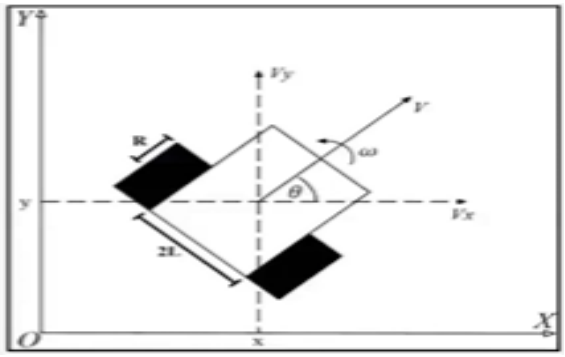


Figure 26 Localization Problem

3.2.5 MARKOV PROCESSES

Model of stochastic process which:

- After any given time, moment t does not depend on the evolution preceding given that the state of the process at this moment is fixed.
- «future» of the process does not depend on «past» given the known «present»
- Satisfies the **Markov property.**

- **Markov property** (for the discrete case)

$$p(X_n = x_n \mid X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \dots, X_0 = x_0) = p(X_n = x_n \mid X_{n-1} = x_{n-1})$$

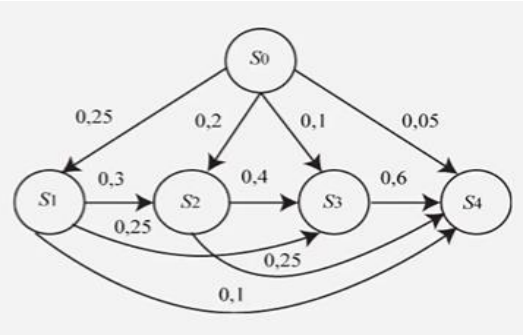


Figure 27 Markov property

  - Particle Filters make only one key assumption: The system and the measurement models are Markovian.

  - $\vec{x}_k$ **Is Markovian:**

Its conditional probability density given the past states, depends only the very last state through the transition density.

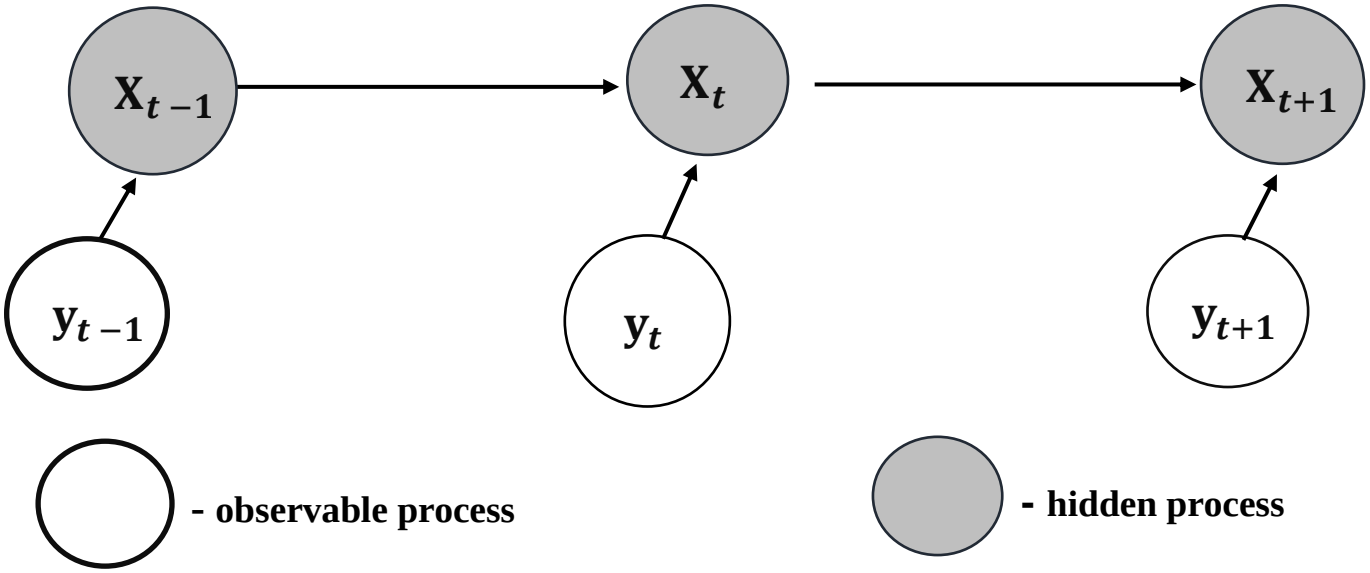
$$p(\vec{x}_k \mid \vec{x}_1, \vec{x}_2, \dots, \vec{x}_{k-1}) = p(\vec{x}_k \mid \vec{x}_{k-1})$$

  - $\vec{z}_k$ **Is Markovian:**

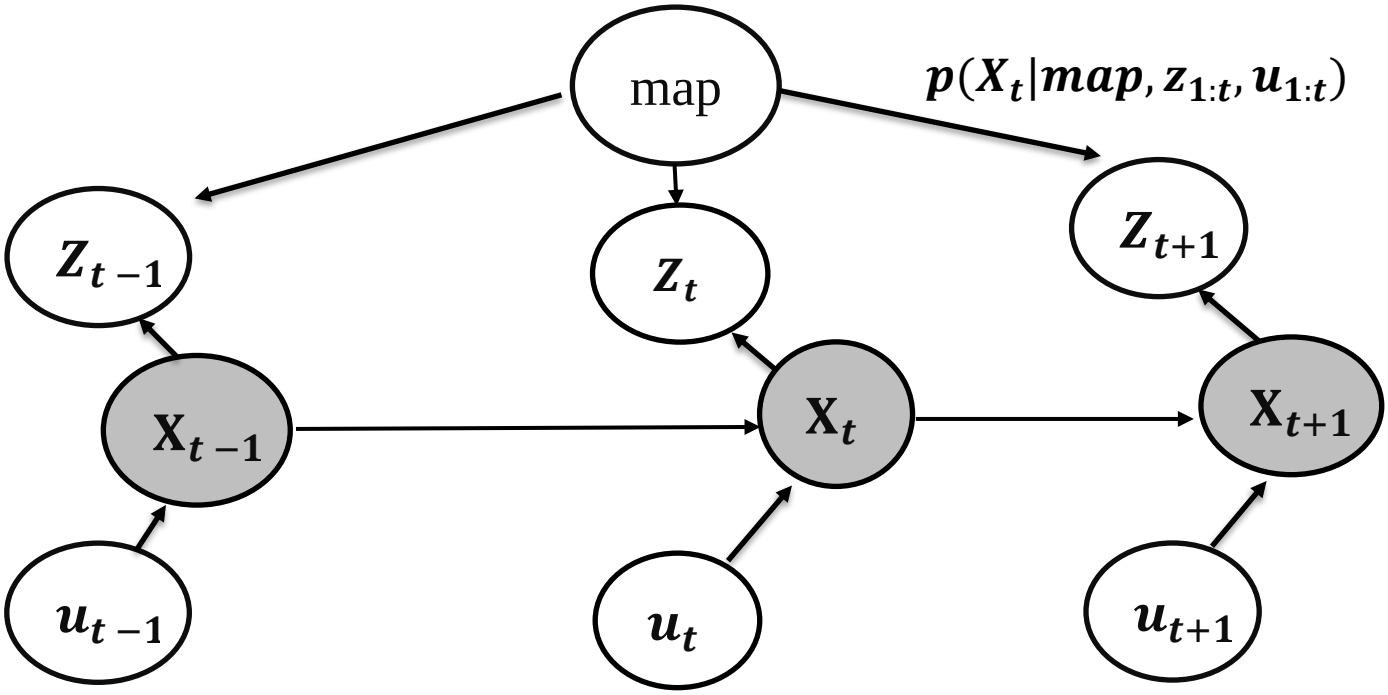
Its conditional probability density given the all states up to and including $\vec{x}_k$ And the past observation , depends only on $\vec{x}_k$ Through the conditional likelihood

$$p(\vec{z}_k \mid \vec{x}_1, \vec{x}_2, \dots, \vec{x}_k, \vec{z}_1, \vec{z}_2, \dots, \vec{z}_{k-1}) = p(\vec{z}_k \mid \vec{x}_k)$$

3.2.6 HIDDEN MARKOV PROCESSES



3.2.7 LOCALIZATION AS HIDDEN MARKOV PROCESSES



3.2.8 Localization

Entropy :

Is a measure of information

- **The Differential Entropy of a continuous random variable X is defined as**

$$\boldsymbol{h(X) = -\int_{-\infty}^{\infty} p(x)\log\, p(x)}$$

- **The Differential Entropy of a Discrete random variable X is defined as**

$$\boldsymbol{-\sum_{i} P(x_i)\log\, P(x_i)}$$

Localization :

BELIEF = PROBABILITY

SENSE = PRODUCT FOLLOWED BY NORMALZIATION

MOVE = CONVOLUTION (ADDITION)

- **PROBABILISTIC FORM OF LOCALIZATION**

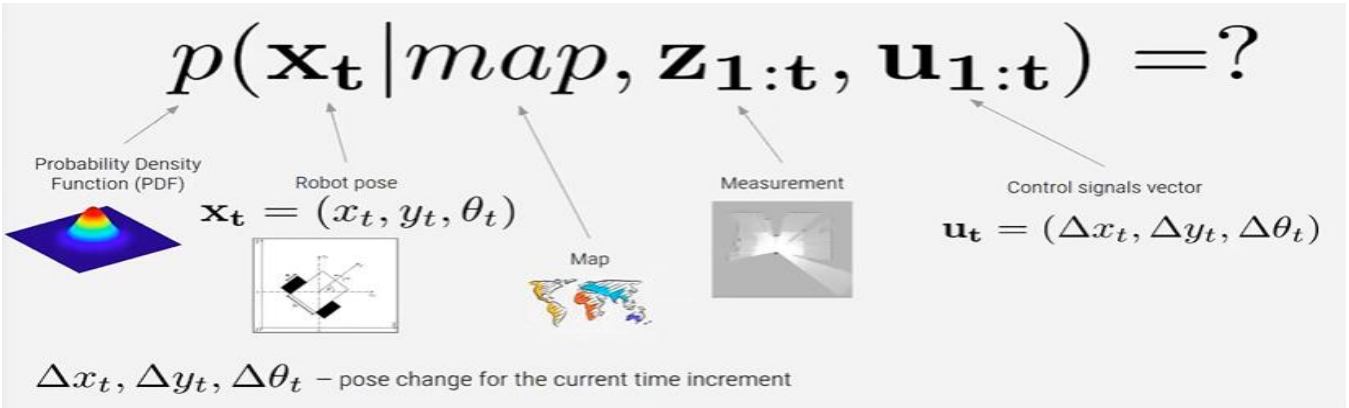


Figure 28Probabilistic of localization

3.2.9 BAYES' RULE (THEOREM)

$$p(a \mid b) = \frac{p(b \mid a)p(a)}{p(b)}$$

Posterior probability How probable, that our hypothesis a is true given the **evidence** b	**Likelihood** How probable is outcome b, given that out hypothesis a is true	**Prior probability** How probable our hypothesis a was before obtaining the evidence b	**Marginal probability** Probability of b, which does not depend on a

3.2.10 LAW OF TOTAL PROBABILITY

Discrete case

$$\sum_x P(x) = 1$$
$$P(x) = \sum_y P(x, y)$$
$$P(x) = \sum_y P(x \mid y)P(y)$$

Continuous case

$$\int P(x)dx = 1$$
$$P(x) = \int p(x, y)dy$$
$$P(x) = \int p(x \mid y)p(y)dy$$

3.2.11 RECURSIVE BAYESIAN POSE ESTIMATION

$$p(\mathbf{x_t} \mid \mathbf{map}, \mathbf{z_t}, \mathbf{u_t}) = C \cdot p(\mathbf{z_t} \mid \mathbf{x_t}, \mathbf{map}) \int_S p(\mathbf{x_t} \mid \mathbf{u_t}, \mathbf{x_{t-1}}) p(\mathbf{x_{t-1}} \mid \mathbf{map}, \mathbf{z_{t-1}}, \mathbf{u_{t-1}}) d\mathbf{x_{t-1}}$$

- **C**: normalization coefficient
- **S**: the probabilistic space of robot poses
- $p(\mathbf{z_t} \mid \mathbf{x_t}, \mathbf{map})$: observation model
- $p(\mathbf{x_t} \mid \mathbf{u_t}, \mathbf{x_{t-1}})$: motion model
- $p(\mathbf{x_{t-1}} \mid \mathbf{map}, \mathbf{z_{t-1}}, \mathbf{u_{t-1}})$: previous system state (robot pose)
- $p(\mathbf{z_t} \mid \mathbf{x_t}, \mathbf{map})$: Correction

- $p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid map, z_{t-1}, u_{t-1})$: Prediction

3.2.12 IMPLEMENTATION OF RECURSIVE BAYESIAN ESTIMATION

- Recursive Bayesian estimation is a **framework**
- There are many algorithms realizing this **framework** →
  - ❑ **Kalman filter**
  - ❑ Information filter
  - ❑ Histogram filter
  - ❑ **Particle filter**
  - ❑ …..
- Linear and nonlinear motion and observation models
- Normal and arbitrary (multimodal) error distributions
- Parametric and nonparametric algorithms

3.2.13 PARTICLE FILTER (MONTE CARLO LOCALIZATION)

- Works with any (even nonlinear) motion and observation models.
- Can describe multimodal distribution.
- Can solve global localization problem -localization without an initial approximation.
- Same dynamic framework No assumptions on system model (other than Markovian) or noise model.
- The estimation process is based on a Bayesian framework.
- Also known as:
  1) Sequential Monte Carlo
  2) Condensation Algorithm
  3) Bootstrap Filtering
  4) Survival of the Fittest
  5) Interacting Particle Approximations

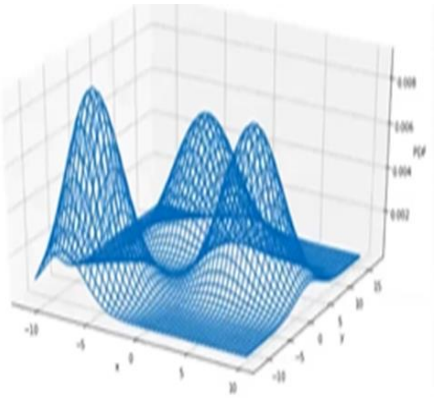


Figure 29Particle Filter

3.2.14 Importance Sampling

- More often than not we can not sample directly from p(), So we sample from another distribution q(), which must have a larger support than p().
- So take N random samples from q() instead of p():

$$\vec{x}^{(i)}, 1 \leq i \leq N \text{ drawn from } q()$$

- A pdf like q() that is used to obtan samples from p() is called importance distribution.
- The technique of using an importance distribution like q(), to obtain Samples for another distribution like p(), is called importance sampling.
    - P() : target function
    - Q() : proposal functon

- **Importance Sampling Example**

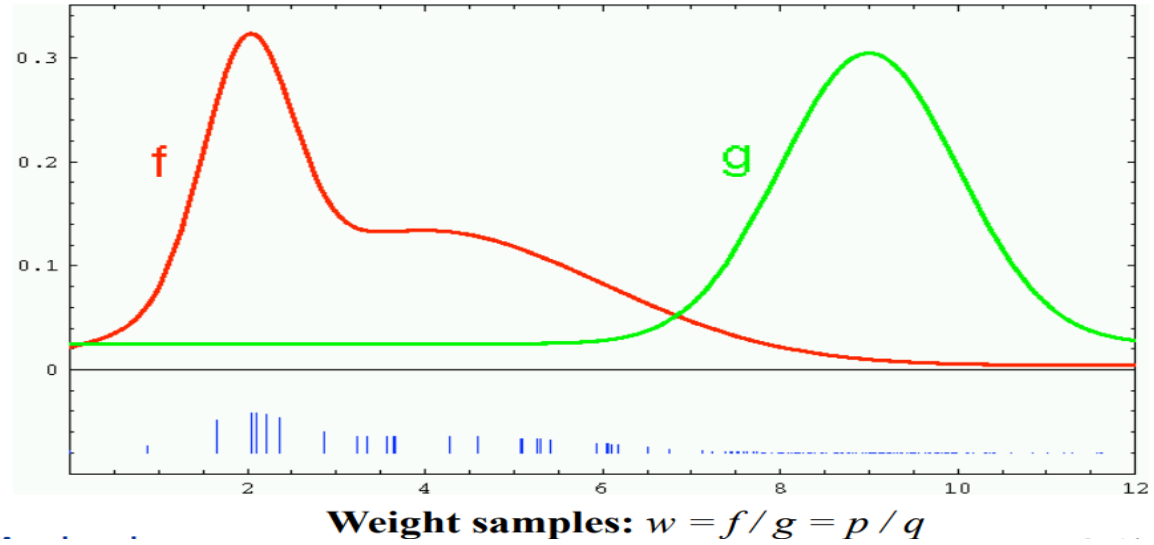


Figure 30Importance Sampling Example

3.2.15 Importance Sampling Correction

- Since q() is of course not p(), each sample must be corrected by being multiplied by an appropriate weight, so that one can obtain an unbiased estimation of the properties of p().
- So for each sample $\vec{x}^{(i)}$ There is a weight $\tilde{\omega}^{(i)}$ That handles the discrepancy between the two distributions:

$$\tilde{\omega}^{(i)} = \frac{p(\vec{x}^{(i)})}{q(\vec{x}^{(i)})}$$

We normalize all the weights so that they sum up to 1.

$$\tilde{\omega}^{(i)} = \frac{\tilde{\omega}^{(i)}}{\sum_{i=1}^{N} \tilde{\omega}^{(i)}}$$

3.2.16 PDF Estimation via Importance Sampling

- We use importance sampling to get discrete samples of the pdf $\boldsymbol{p}\left(\vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \mid \vec{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \vec{\boldsymbol{z}}_{\boldsymbol{k}}\right)$

$$\tilde{\boldsymbol{w}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \propto \frac{\boldsymbol{p}\left(\vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \mid \vec{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \vec{\boldsymbol{z}}_{\boldsymbol{k}}\right)}{\boldsymbol{q}\left(\vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \mid \vec{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \vec{\boldsymbol{z}}_{\boldsymbol{k}}\right)}$$

- But within the Bayesian + Markovian framework we can compute variables and pdf's recursively

$$\tilde{\boldsymbol{w}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \propto \tilde{\boldsymbol{w}}_{\boldsymbol{k-1}}^{(\boldsymbol{i})} \frac{\boldsymbol{p}\left(\vec{\boldsymbol{z}}_{\boldsymbol{k}} | \vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})}\right) \boldsymbol{p}\left(\vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})} | \vec{\boldsymbol{x}}_{\boldsymbol{k-1}}^{(\boldsymbol{i})}\right)}{\boldsymbol{q}\left(\vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \mid \vec{\boldsymbol{x}}_{\boldsymbol{k-1}}^{(\boldsymbol{i})}, \vec{\boldsymbol{z}}_{\boldsymbol{k}}\right)}$$

3.2.17 Weights and PDF

- The weight of each particle can be thought of as an approximation of the probability that the specific particle value will occur.

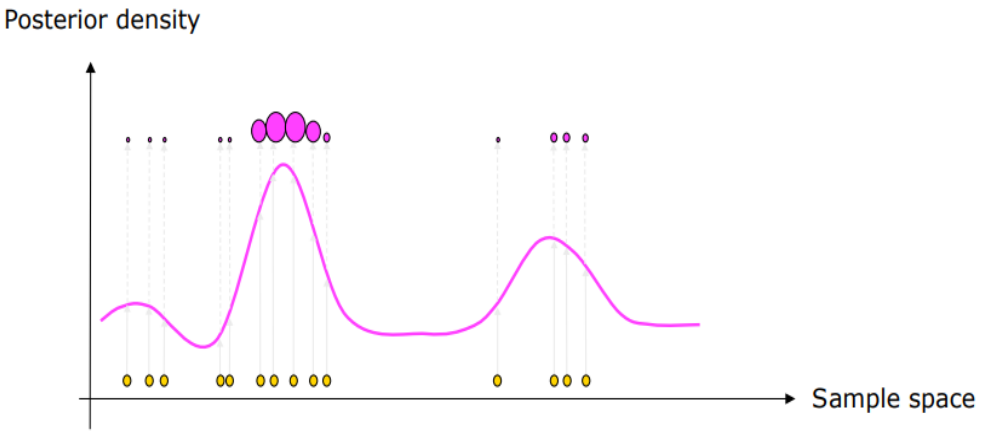


Figure 31Weights and PDF

3.2.18 Basic Particle Filter Algorithm

1) For = 1 to N
2) Draw $\vec{x}_k^{(i)}$ From $q\left(\vec{x}_k^{(i)} \mid \vec{x}_{k-1}^{(i)}, \vec{z}_k\right)$
3) Compute $\widetilde{\boldsymbol{w}}_{\boldsymbol{k}}^{(\boldsymbol{i})}$ **Using the markovian update**

$$\widetilde{\boldsymbol{w}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \propto \frac{\boldsymbol{p}\left(\vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \mid \vec{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \vec{\boldsymbol{z}}_{\boldsymbol{k}}\right)}{\boldsymbol{q}\left(\vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})} \mid \vec{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \vec{\boldsymbol{z}}_{\boldsymbol{k}}\right)}$$

4) Compute the normalized weight, $w^{(i)}$, and assign it to the corresponding particle.
5) End
6) Out of all the particles, pick $\vec{\boldsymbol{x}}_{\boldsymbol{k}}^{(\boldsymbol{i})}$With the maximum weight, or set $\vec{\boldsymbol{x}}_{\boldsymbol{k}}$ To the expected value of posterior density p().

3.2.19 Degeneracy Problem

1) When our sampling distribution closely matches the target distribution, the weights should be approximately constant. We want low variance in the weights across the different particles.
2) **Measure for degeneracy:** Effective sample size.

$$\mathbf{N} = \frac{\mathbf{N}}{\mathbf{1} + \boldsymbol{Var}(\widetilde{\boldsymbol{w}}_{\boldsymbol{k}}^{(\boldsymbol{i})})}$$

3) Small Neff indicates severe degeneracy

- Solution 1:

Use very large N: can be problematic.

- Solution 2:

Resample which means replicate particles in proportion to their weights:

- Particles with small weights are eliminated.
- Particles with large weights are replicated as many times as their weight.

- Resampling Example

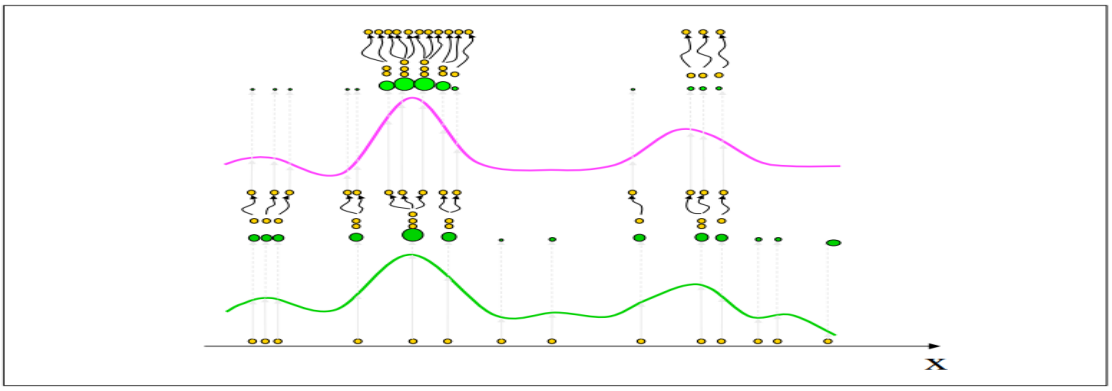


Figure 32Resampling

3.2.20 PARTICLE FILTER

- **Fun:** The particle filter allows one to Estimate the posterior distribution Of the state vector of a system Modeled by a hidden Markov process.

- $\boldsymbol{X_t}$: State of a Markov process at a time t.

- $\boldsymbol{X_t}$: Depends on the previous state X_{t-1} In accordance with the motion model

  P($\boldsymbol{X_t}$ I $\boldsymbol{U_t}$, $\boldsymbol{X_{t-1}}$).

  Where u, is the vector of control signals Applied at the moment t - 1.

Algorithm 1 Generic Monte-Carlo localization algorithm

1: procedure MCL($\mathbf{x_{t-1}}, m, \mathbf{u_t}, \mathbf{z_t}$)
2: $\{\mathbf{x_t^n}\} = \{\widetilde{\mathbf{x_t^n}}\} = \emptyset$
3: for $n = 1$ to N do
4: sample $x_t^n \sim p(\mathbf{x_t}|\mathbf{u_t}, \mathbf{x_{t-1}^n})$
5: $w_t^n = p(\mathbf{z_t}|\mathbf{x_t^n}, map)$
6: $\{\widetilde{\mathbf{x_t^n}}\} = \{\widetilde{\mathbf{x_t^n}}\} + \langle x_t^n, w_t^n \rangle$
7: end for
8: for $n = 1$ to N do
9: draw i with probability $\propto \widetilde{w_t^i}$
10: $\{\mathbf{x_t^n}\} = \{\mathbf{x_t^n}\} + \langle \widetilde{x_t^i}, \widetilde{w_t^i} \rangle$
11: end for
12: return $\{\mathbf{x_t^n}\}$
13: end procedure

Figure 33 Algorithm MCL

- X_t Is described by the measurement vector z, generated according to the observation model p(Z_t I X_t).
- Particle filter idea - approximation of an unknown posterior distribution by a set of hypotheses (X_t) and the corresponding weights (W_t).
  Where n = 1..., N, number of hypotheses / particles.

- In the localization problem, each particle corresponds to the pose of the robot: $X_t = (X_t, y_t, \theta_t)$.

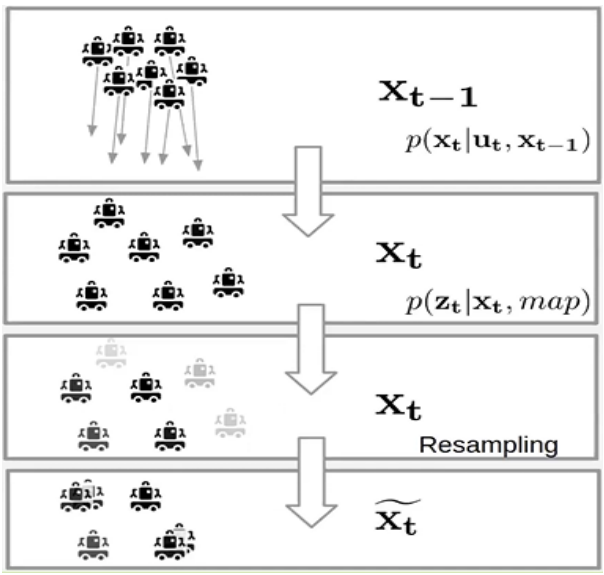


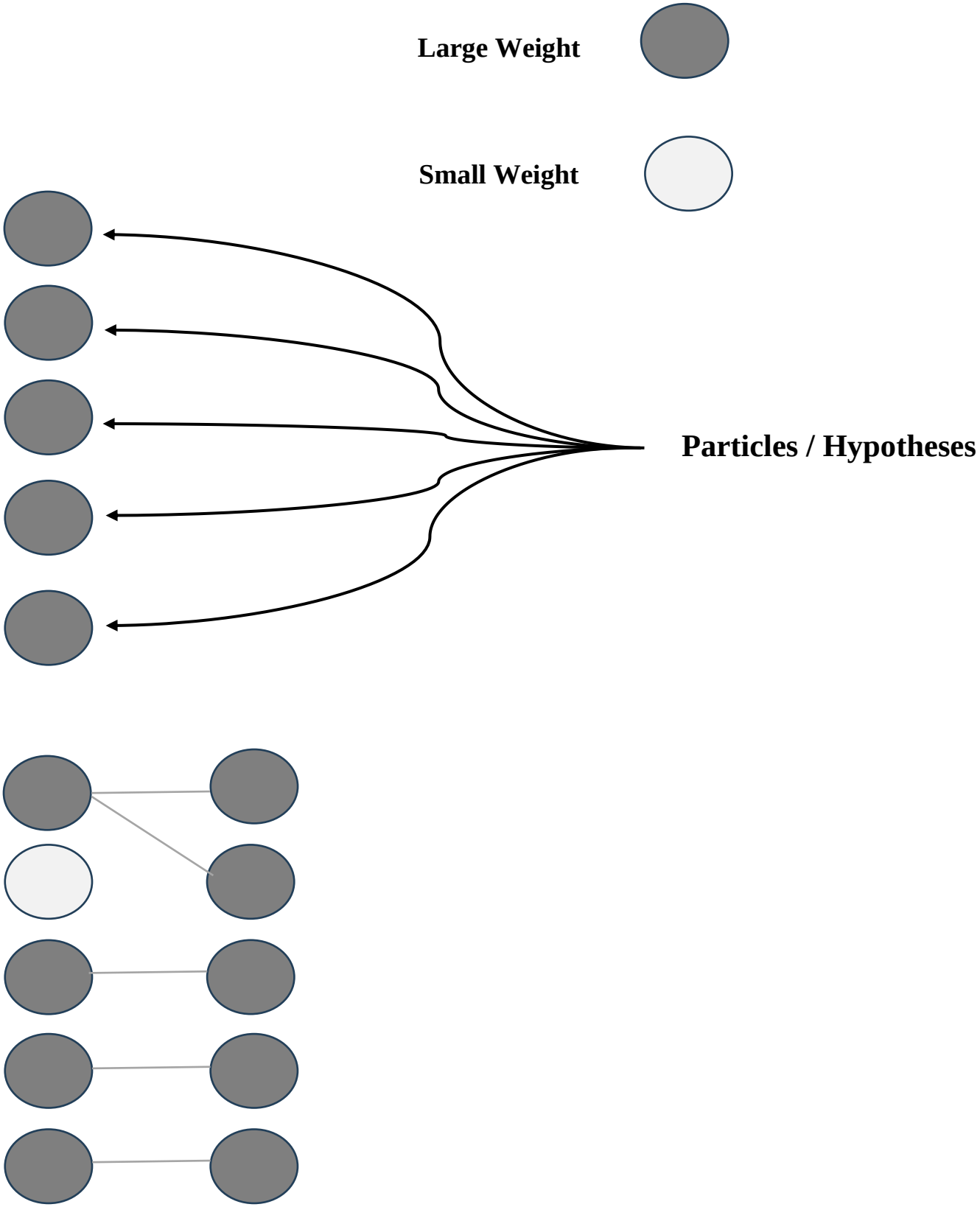
Figure 34PF 1

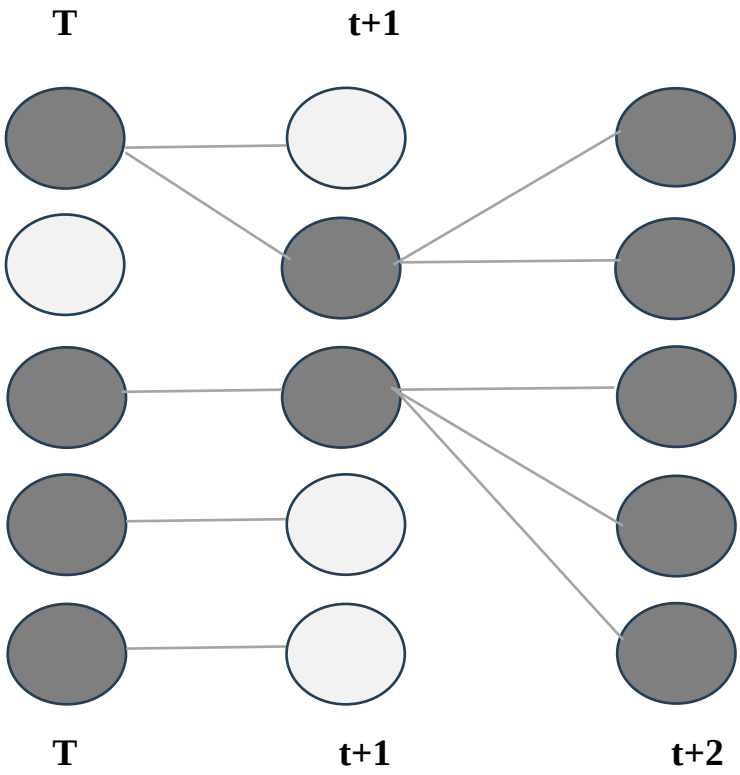
Algorithm 1 Generic Monte-Carlo localization algorithm

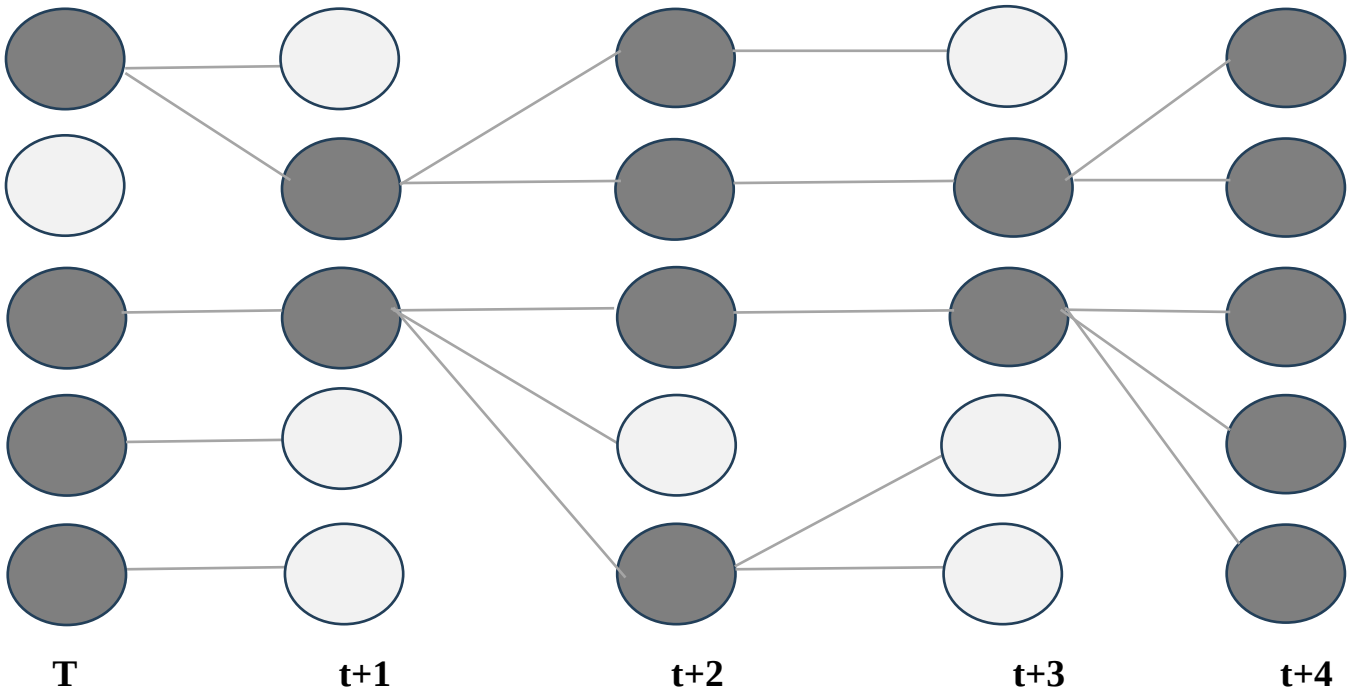
1: procedure MCL($\mathbf{x_{t-1}}, m, \mathbf{u_t}, \mathbf{z_t}$)
2: $\{\mathbf{x_t^n}\} = \{\widetilde{\mathbf{x_t^n}}\} = \emptyset$
3: for $n = 1$ to N do
4: sample $x_t^n \sim p(\mathbf{x_t}|\mathbf{u_t}, \mathbf{x_{t-1}^n})$ — Motion model
5: $w_t^n = p(\mathbf{z_t}|\mathbf{x_t^n}, map)$ — Observation model
6: $\{\widetilde{\mathbf{x_t^n}}\} = \{\widetilde{\mathbf{x_t^n}}\} + \langle x_t^n, w_t^n \rangle$ — Observation model
7: end for
8: for $n = 1$ to N do — Resampling
9: draw i with probability $\propto \widetilde{w_t^i}$ — Resampling
10: $\{\mathbf{x_t^n}\} = \{\mathbf{x_t^n}\} + \langle \widetilde{x_t^i}, \widetilde{w_t^i} \rangle$ — Resampling
11: end for — Resampling
12: return $\{\mathbf{x_t^n}\}$
13: end procedure

Figure 35PF 2

3.2.21 PARTICLE FILTER RESAMPLING




T
t+1
T
t+1
t+2


T
t+1
t+2
t+3
t+4

- **Types PARTICLE FILTER RESAMPLING**

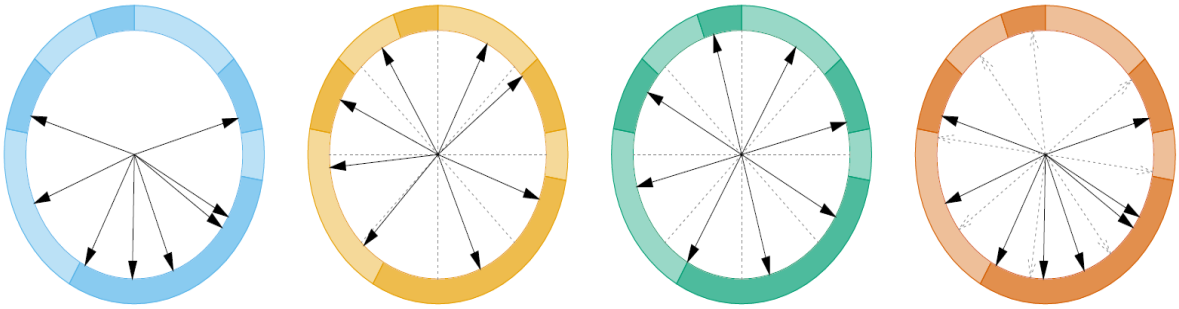


Figure 36Particle Filter Resampling

4

Analysis & Design

4.1 UML Diagram

4.2 Flowchart

4.1 UML Diagram:

We used (Use Case) methodology in our system analysis to identify, clarify, and organize our system requirements.
And to specify the roles played by the actors within the system and the relationships between and among them.

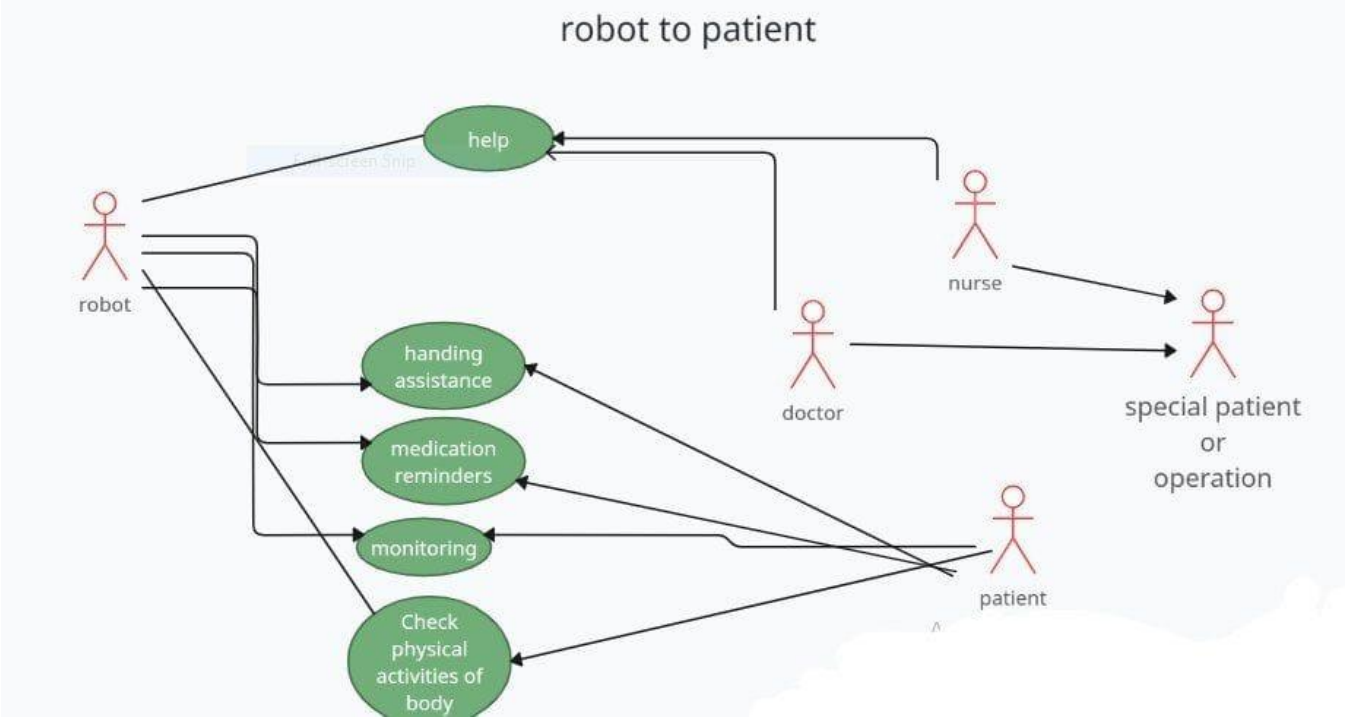


Figure 37UML Diagram

4.2 Flowchart

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.
The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analysing, designing, documenting or managing a process or program in various fields

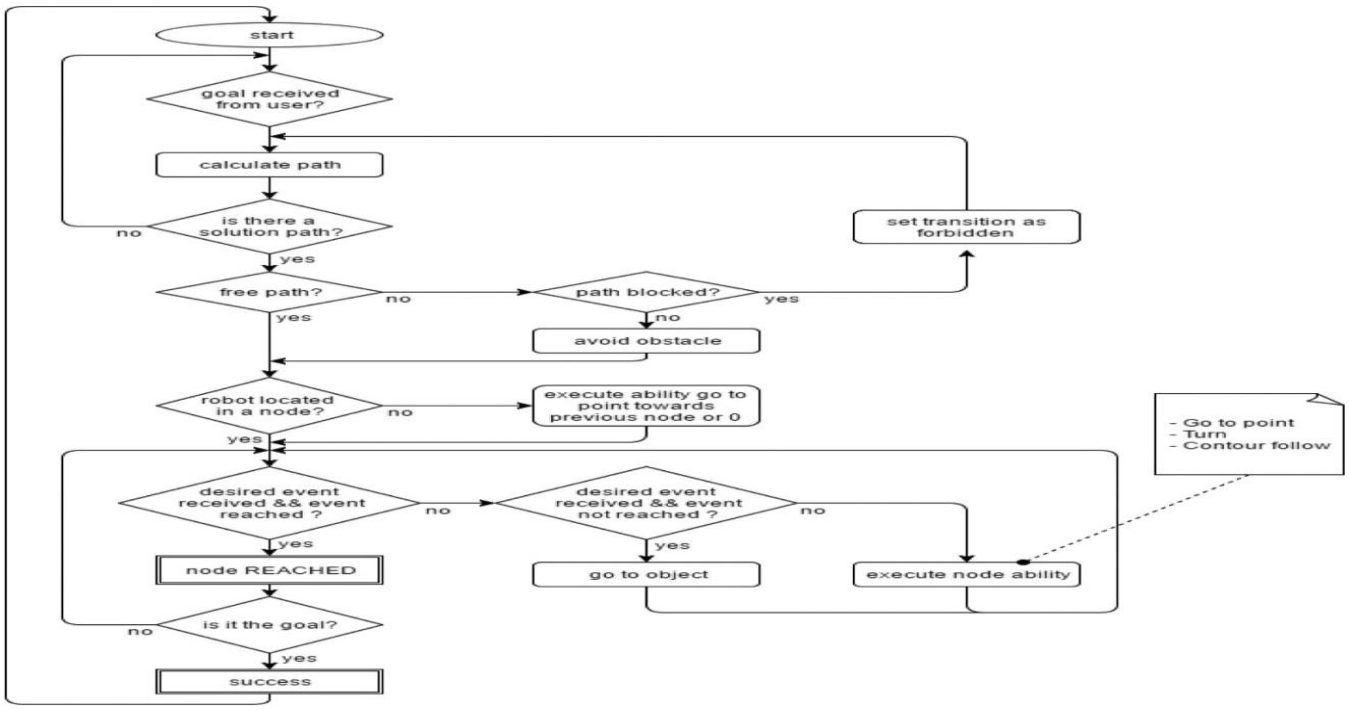


Figure 38 Flowchart 1

Flowchart Explanation:

- In Figure 38, the flow chart for navigation is shown. First, the system waits until it receives a valid target from the user. Once the goal is received, the planner calculates the path and asks the navigator to perform it. The system is continuously checking whether the path is free of obstacles before moving. If there is an obstacle in the way, the robot will check whether it is avoidable or not, meaning that the path is blocked. If it is blocked, the system will ask the planner for an alternative path, if it exists. Whereas, if the obstacle is avoidable, the robot will avoid it and continue with the execution. If the path is free or the robot has overtaken an obstacle, in order to start navigation, the system will check whether the robot is located in a known node. In case it is not, the robot will move towards one. While the robot navigates, if it detects the desired perceptive event, it marks that node as reached. If that node is the goal node, the execution will end successfully; if it is not, the system will refresh the next node to reach. If the desired event is not detected, the robot will execute the corresponding ability until it is detected. Whereas, if the robot detects the event but it is not located in the desired position, it will execute ability go to object to reach the node's position.

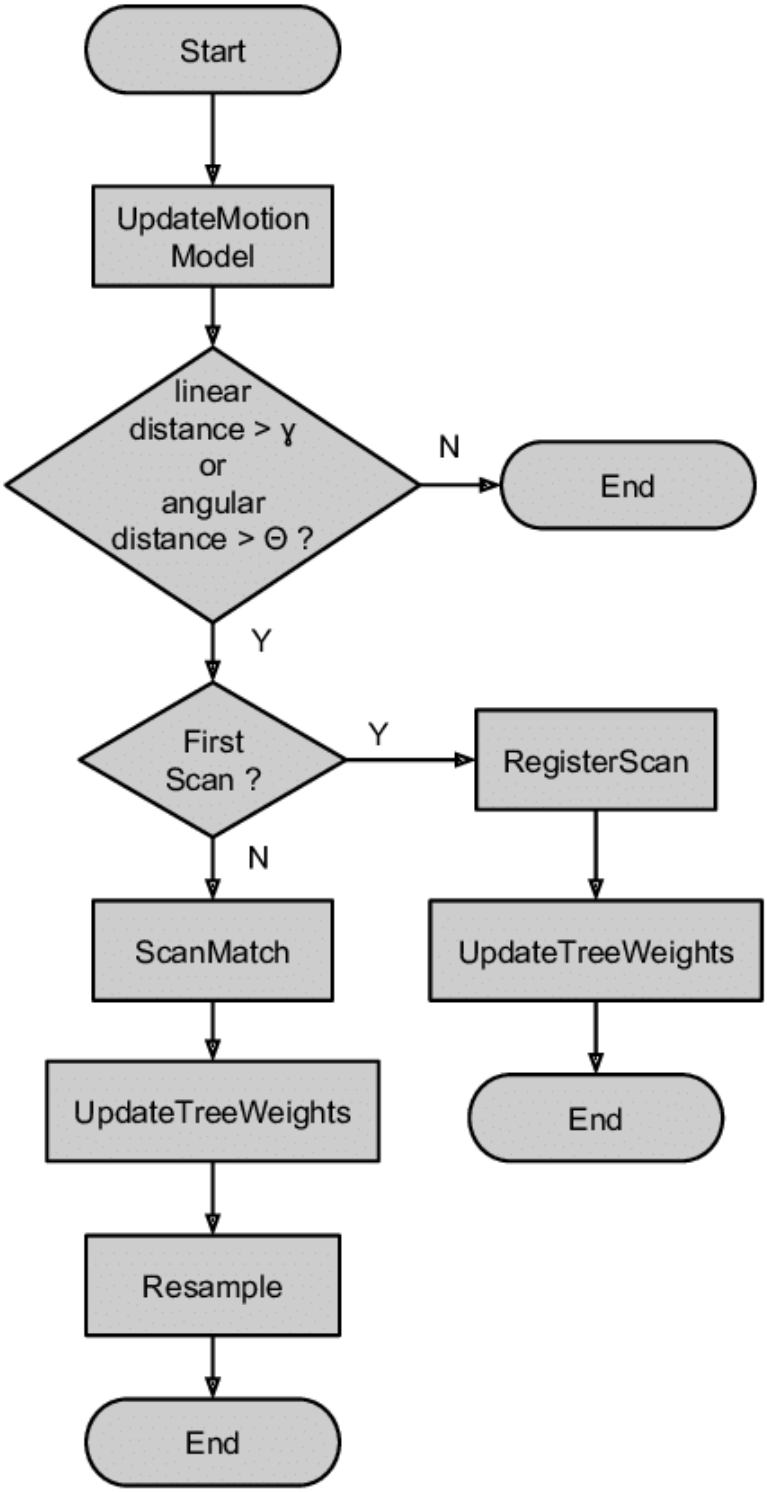


Figure 39 Flowchart of Gmapping in ROS

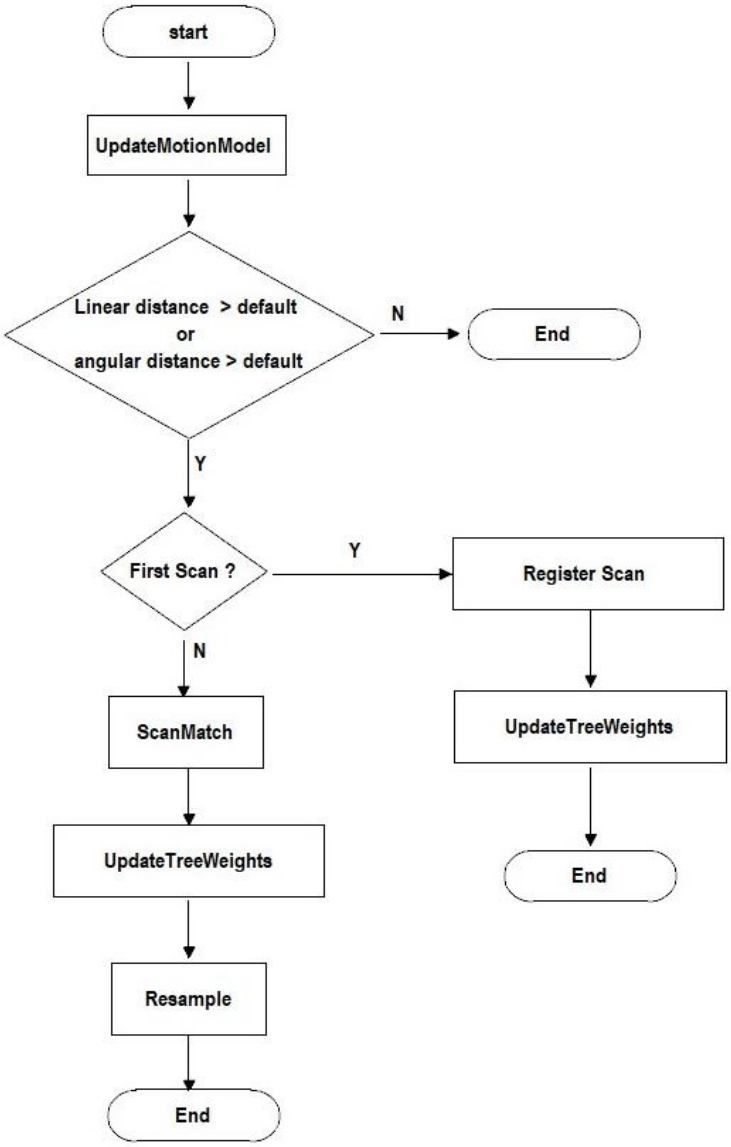


Figure 40Flowchart of the Process Scan function of the G-mapping algorithm

5

Implementation

5.1 Design Robot (URDF)

5.2 Design Environment

5.3 Mapping (G-mapping)

5.4 Localization (AMCL)

5.5 Navigation

5.1 Design Robot

5.1.1 Building Our Robot Using URDF (Xacro)

- **Our robot consists of:**
  1) Base:
     (Base link, Middle base Link ,Top base link)
  2) Bar:

     (bar1 link, bar2 link, bar3 link, bar4 link)

  3) Wheel:

     (Wheel left ,Wheel right ,Caster front ,Caster back)

  4) Camera:

     (Kinect link ,Camera optical)

  5) Ultrasonic:

     (Ultrasonic1, Ultrasonic2, Ultrasonic3)

1) Base:

The three base is designed with urdf,
the link is given a name, a visual, a collision, and an inertia.
The joint is given a name, type, parent link, and child link.

- **Base link:**

Link	Joint
<link name="base_footprint"/> <link name="base_link"> <visual> <geometry> <mesh filename="package://rsrobot_description/meshes/barista/BOTTOM_colour.dae" /> </geometry> <origin xyz="0.001 0 0.05199" rpy="0 0 0"/> </visual> <collision> <geometry> <cylinder length="0.10938" radius="0.178"/> </geometry> <origin xyz="0.0 0 0.05949" rpy="0 0 0"/> </collision> <inertial> <origin xyz="0.01 0 0"/> <mass value="2.4"/> <!-- 2.4/2.6 kg for small/big battery pack --> <inertia ixx="0.019995" ixy="0.0" ixz="0.0" iyy="0.019995" iyz="0.0" izz="0.03675" /> </inertial> </link>	<joint name="base_joint" type="fixed"> <origin xyz="0 0 0.0102" rpy="0 0 0" /> <parent link="base_footprint"/> <child link="base_link" /> </joint>

Table 2 Base link

- **Middle base Link:**

Link	Joint
<link name="middle_base_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <cylinder radius="0.17" length="0.03"/> </geometry> <material name=""> <color rgba="1.0 0.0 0.0 1.0"/> <texture filename=""/> </material> </visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <cylinder length="0.10938" radius="0.178"/> </geometry> </collision> </link>	<joint name="middle_base_joint" type="fixed"> <parent link="base_link"/> <child link="middle_base_link"/> <origin xyz="0.00 0.0 0.2" rpy="0 0 0"/> <axis xyz="0 0 0"/> </joint>

Table 3 middle base link

- **Top base link:**

Link	Joint
<link name="top_base_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <mesh filename="package://rsrobot_description/meshes/barista/TOP_colour.dae"/> </geometry> <material name=""> <color rgba="1.0 0.0 0.0 1.0"/> <texture filename=""/> </material>	<joint name="top_base_joint" type="fixed"> <parent link="base_link"/> <child link="top_base_link"/> <origin xyz="0.00 0.0 0.33" rpy="0 0 0"/> <axis xyz="0 0 0"/> </joint>

</visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <mesh filename="package://rsrobot_description/meshes/barista/TOP_colour.dae"/> </geometry> </collision> </link>	

Table 4top base link

2) Bar:

The four bare is designed with urdf,
the link is given a name, a visual, a collision, and an inertia.
The joint is given a name, type, parent link, and child link.

- **bar1 link:**

Link	Joint
<link name="bar1_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <cylinder radius="0.01" length="0.35"/> </geometry> <material name=""> <color rgba="0.0 1.0 0.0 1.0"/> <texture filename=""/> </material> </visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.0 0.0 0.0"/> </geometry> </collision> </link>	**<joint name="bar1_joint" type="fixed"> <parent link="base_link"/> <child link="bar1_link"/> <origin xyz="0.05 0.11 0.15" rpy="0 0 0"/> <axis xyz="0 0 1"/> </joint>**

Table 5bar1 link

- **bar2 link:**

Link	Joint
<link name="bar2_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <cylinder radius="0.01" length="0.35"/> </geometry> <material name=""> <color rgba="0.0 1.0 0.0 1.0"/> <texture filename=""/> </material> </visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.0 0.0 0.0"/> </geometry> </collision> </link>	<joint name="bar2_joint" type="fixed"> <parent link="base_link"/> <child link="bar2_link"/> <origin xyz="-0.09 0.11 0.15" rpy="0 0 0"/> <axis xyz="0 0 1"/> </joint>

Table 6bar2 link

- **bar3 link:**

Link	Joint
<link name="bar3_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <cylinder radius="0.01" length="0.35"/> </geometry> <material name=""> <color rgba="0.0 1.0 0.0 1.0"/> <texture filename=""/> </material> </visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.0 0.0 0.0"/> </geometry> </collision> </link>	<joint name="bar3_joint" type="fixed"> <parent link="base_link"/> <child link="bar3_link"/> <origin xyz="0.05 -0.11 0.15" rpy="0 0 0"/> <axis xyz="0 0 1"/> </joint>

Table 7bar3 link

- **bar4 link:**

Link	Joint
<link name="bar4_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <cylinder radius="0.01" length="0.35"/> </geometry> <material name=""> <color rgba="0.0 1.0 0.0 1.0"/> <texture filename=""/>	<joint name="bar4_joint" type="fixed"> <parent link="base_link"/> <child link="bar4_link"/> <origin xyz="-0.09 -0.11 0.15" rpy="0 0 0"/> <axis xyz="0 0 1"/> </joint>

```
    </material>
  </visual>
  <collision>
    <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>
    <geometry>
      <box size="0.0 0.0 0.0"/>
    </geometry>
  </collision>
</link>
```

Table 8bar4 link

3) Wheel:

The four wheel is designed with urdf.

The link is given a name, a visual, a collision, and an inertia.

The joint is given a name, type, parent link, and child link.

Wheel left:

Link	Joint
<link name="wheel_left_link"> <visual> <geometry> <mesh filename="package://rsrobot_description/meshes/wheel.dae"/> </geometry> <origin xyz="0 0 0" rpy="0 0 0"/> </visual> <collision> <geometry> <cylinder length="0.0206" radius="0.0352"/> </geometry> <origin rpy="0 0 0" xyz="0 0 0"/> </collision> <inertial> <mass value="0.01" /> <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="wheel_left_joint" type="continuous"> <parent link="base_link"/> <child link="wheel_left_link"/> <origin xyz="0.00 ${0.23/2} -0.05" rpy="${-M_PI/2} 0 0"/> <axis xyz="0 0 1"/> </joint>

Table 9 Wheel left

Wheel right:

Link	Joint
<link name="wheel_right_link"> <visual> <geometry> <mesh filename="package://rsrobot_description/meshes/wheel.dae"/> </geometry> <origin xyz="0 0 0" rpy="0 0 0"/> </visual> <collision> <geometry> <cylinder length="0.0206" radius="0.0350"/> </geometry> <origin rpy="0 0 0" xyz="0 0 0"/> </collision> <inertial> <mass value="0.01" /> <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="wheel_right_joint" type="continuous"> <parent link="base_link"/> <child link="wheel_right_link"/> <origin xyz="0.00 -${0.23/2} -0.05" rpy="${-M_PI/2} 0 0"/> <axis xyz="0 0 1"/> </joint>

Table 10wheel right

Caster front:

Link	Joint
<link name="caster_front_link"> <visual> <geometry> <sphere radius="0.04"/> </geometry> <origin rpy="0 0 0" xyz="0 0 0"/> </visual> <collision> <geometry> <sphere radius="0.04"/> </geometry> <origin rpy="0 0 0" xyz="0 0 0"/> </collision> <inertial> <mass value="0.01" /> <origin xyz="0 0 0" />	<joint name="caster_front_joint" type="fixed"> <parent link="base_link"/> <child link="caster_front_link"/> <origin xyz="0.115 0.0 -0.045" rpy="${-M_PI/2} 0 0"/> </joint>

<inertia ixx="0.001" ixy="0.0" ixz="0.0" iyy="0.001" iyz="0.0" izz="0.001" /> </inertial> </link>	

Table 11 Caster front

Caster back:

Link	Joint
<link name="caster_back_link"> <visual> <geometry> <sphere radius="0.04"/> </geometry> <origin rpy="0 0 0" xyz="0 0 0"/> </visual> <collision> <geometry> <sphere radius="0.04"/> </geometry> <origin rpy="0 0 0" xyz="0 0 0"/> </collision> <inertial> <mass value="0.01" /> <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="caster_back_joint" type="fixed"> <parent link="base_link"/> <child link="caster_back_link"/> <origin xyz="-0.1 0.0 -0.045" rpy="${-M_PI/2} 0 0"/> </joint>

Table 12caster back

4) Camera:

The two Camera is designed with urdf.

The link is given a name, a visual, a collision, and an inertia.

The joint is given a name, type, parent link, and child link.

- **Kinect link:**

Link	Joint
<link name="kinect_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.1"/> <inertia ixx="1e-6" ixy="0.0" ixz="0.0" iyy="1e-6" iyz="0.0" izz="1e-6"/> </inertial> <visual name="kinect_visual"> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <mesh filename="package://rsrobot_description/meshes/kinect.dae"/> </geometry> <material name="white"/> </visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.05 0.05 0.05"/> </geometry> </collision> </link>	<joint name="kinect_joint" type="fixed"> <origin xyz="0.13 0.0 0.13" rpy="0.0 0.0 0.0"/> <parent link="base_link"/> <child link="kinect_link"/> <axis xyz="0.0 0 1.0"/> </joint>

Table 13kinect link

- **Camera optical:**

Link	Joint
<link name="camera_link_optical"> </link>	<joint name="camera_optical_joint" type="fixed"> <origin xyz="0 0 0" rpy="-1.5707 0 -1.5707"/> <parent link="kinect_link"/> <child link="camera_link_optical"/> </joint>

Table 14camera optical

5) Ultrasonic:

The 3 ultrasonic is designed with urdf,

The link is given a name, a visual, a collision, and an inertia.

The joint is given a name, type, parent link, and child link.

Ultrasonic1:

Link	Joint
<link name="ultrasonic1_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.01 0.05 0.03"/> </geometry> <material name=""> <color rgba="1.0 0.0 0.0 1.0"/> <texture filename=""/> </material> </visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.01 0.05 0.03"/> </geometry> </collision> </link>	<joint name="ultrasonic1_joint" type="fixed"> <origin xyz="0.18 0.0 0.0" rpy="0.0 0.0 0.0"/> <parent link="base_link"/> <child link="ultrasonic1_link"/> <axis xyz="0.0 0.0 0.0"/> <limit lower="0.0" upper="0.0" effort="0.0" velocity="0.0"/> </joint>

Table 15ultrasonic1

Ultrasonic2:

Link	Joint
<link name="ultrasonic2_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.01 0.05 0.03"/> </geometry> <material name=""> <color rgba="1.0 0.0 0.0 1.0"/> <texture filename=""/> </material> </visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.01 0.05 0.03"/> </geometry> </collision> </link>	<joint name="ultrasonic2_joint" type="fixed"> <origin xyz="0.0 0.18 0.0" rpy="0.0 0.0 1.57"/> <parent link="base_link"/> <child link="ultrasonic2_link"/> <axis xyz="0.0 0.0 0.0"/> <limit lower="0.0" upper="0.0" effort="0.0" velocity="0.0"/> </joint>

Table 16ultrasonic2

Ultrasonic3:

Link	Joint
<link name="ultrasonic3_link"> <inertial> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <mass value="0.0"/> <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0" izz="0.0"/> </inertial> <visual name=""> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.01 0.05 0.03"/> </geometry> <material name=""> <color rgba="1.0 0.0 0.0 1.0"/> <texture filename=""/> </material> </visual> <collision> <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/> <geometry> <box size="0.01 0.05 0.03"/> </geometry> </collision> </link>	<joint name="ultrasonic3_joint" type="fixed"> <origin xyz="0.0 -0.18 0.0" rpy="0.0 0.0 -1.57"/> <parent link="base_link"/> <child link="ultrasonic3_link"/> <axis xyz="0.0 0.0 0.0"/> <limit lower="0.0" upper="0.0" effort="0.0" velocity="0.0"/> </joint>

Table 17ultrasonic3

5.1.2 Launching Our Robot Model In Gazebo

Run Launch File:

1) **Go to workspace:**

   $ cd catkin_ws/

2) **Launch File world.launch:**

   $ roslaunch rsrobot_gazebo world.launch

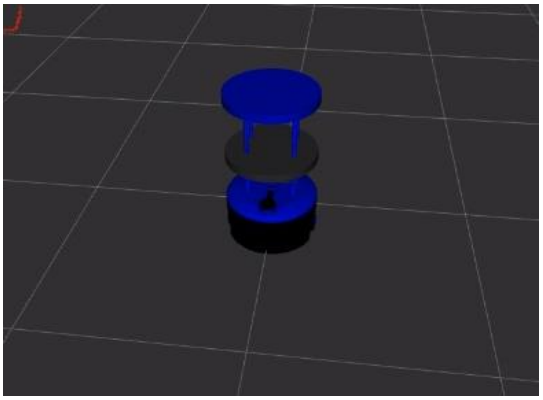


Figure 41Robot Model

5.2 Design Environment:

5.2.1 Building Custom (Model) In Gazebo

The model is chosen, whether it is a chair and a table or other different models provided by Ros, then it is pulled from the left side of Gazebo and downloaded into the Gazeb environment and building the restaurant.

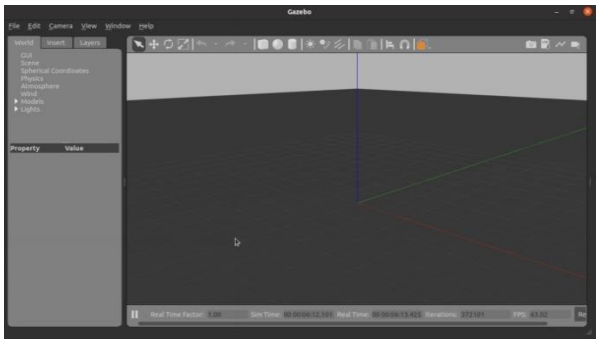


Figure 43Design Environment 1

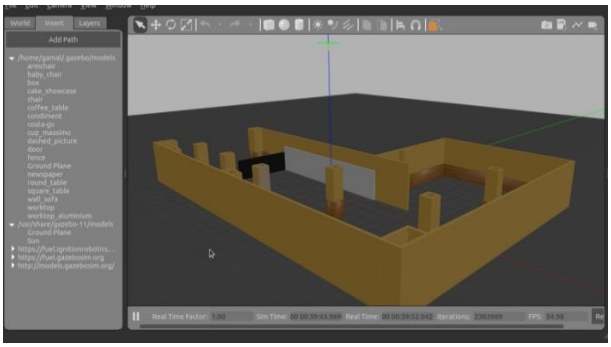


Figure 42Design Environment 2

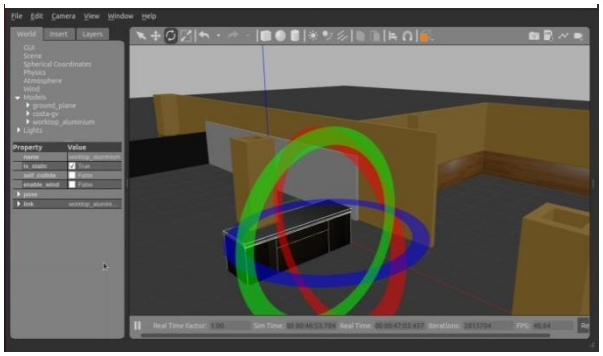


Figure 45Design Environment 3

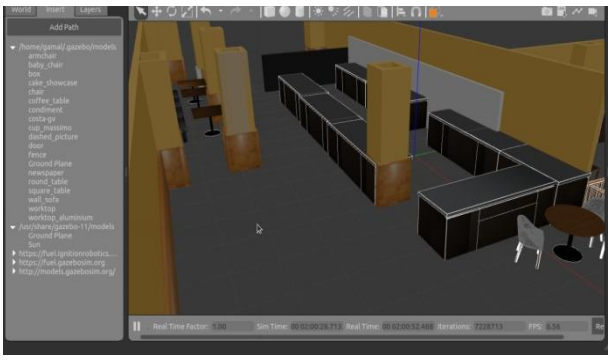


Figure 44Design Environment 4

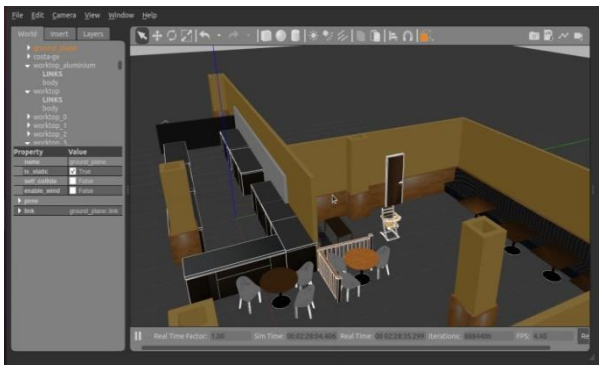


Figure 47Design Environment 5

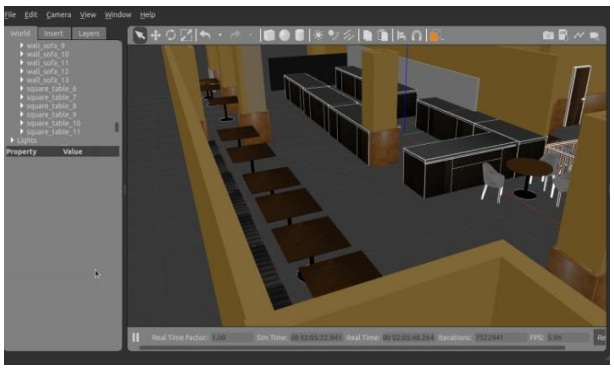


Figure 46Design Environment 6

- A link explaining the process of building the environment (the restaurant):

https://drive.google.com/file/d/1lo21SNywQ7jwrXzfrsAqKf7vhaQE02Hu/view?usp=drivesdk

5.2.2 Launching our robot in world model using gazebo

Run:-

1) Go to workspace:

$ cd catkin_ws/

2) Launch File world.launch:

$ roslaunch rsrobot_gazebo world.lau nch

Figure 49Launching Our Robot 1

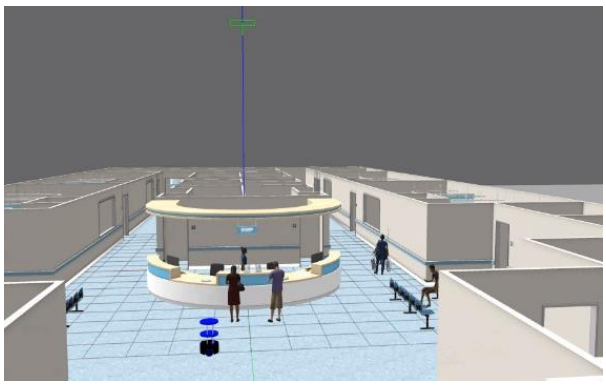


Figure 48Launching Our Robot 2

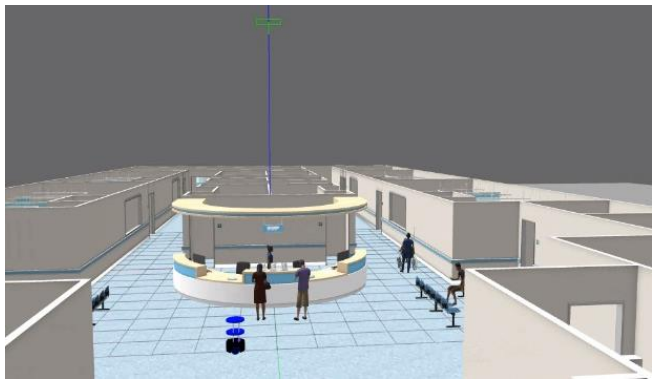


Figure 50Launching Our Robot 3

5.3 Mapping (Gmapping):

5.3.1 Definition:

- One of SLAM method published in openslam, packaged in ROS.
- Author: G. Grisetti, C. Stachniss, W. Burgard.
- Feature: Rao-Blackwellized particle filter, Decreased number of particles, grid map.
- **Completed map:**

2D Occupancy Grid Map (OGM)

- White = Free area where robot can move.
- Black = Occupied area where robot can not move.
- Gray = Unknown area.

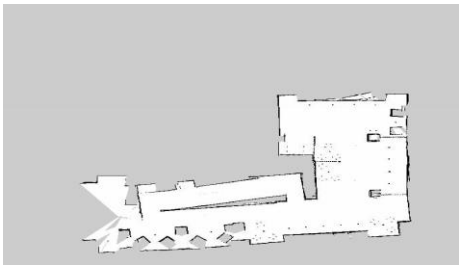


Figure 51Gmapping

5.3.2 Building a map using SLAM*:*

- He ROS Gmapping package is a wrapper of open source implementation of SLAM called Open_SLAM (https://www.openslam.org/gmapping.html).
- The package contains a node called slam_gmapping .
- Which is the implementation of SLAM which helps to create a 2D occupancy grid map from the laser scan data and the mobile robot pose.
- The basic hardware requirement for doing SLAM:-

1) Is a laser scanner which is horizontally mounted on the top of the robot.
2) And the robot odometry data.

- In this robot, we have already satisfied these requirements We can generate the 2D map of the environment using the gmapping package through the following procedure.

5.3.3 Creating a launch file for gmapping:-

1) He main task while creating a launch file for the gmapping process is to set the parameters for the slam_gmapping node.
2) The slam_gmapping node is the core node inside the ROS gmapping package.
3) The slam_gmapping node subscribes the laser data (sensor_msgs/LaserScan) and the tf data, and publishes the occupancy grid map data as output (nav_msgs/OccupancyGrid).
4) This node is highly configurable and we can fine tune the parameters to improve the mapping accuracy.
5) The parameters are mentioned at http://wiki.ros.org/gmapping .
6) Following is the rsrobot_gmapping.launch file used in this robot.
7) The launch file is placed in the rsrobot_navigation

5.3.4 the rsrobot_gmapping.launch file:-

- It consists of two nodes:-

1) node gmapping_pkg: in which parameters are defined.
2) node Rviz.

Params_gmapping:

	Parameters	Discerption
1	<param name="base_frame" value="$(arg base_frame)"/>	The frame attached to the mobile base. default: "base_link"
2	<param name="odom_frame" value="$(arg odom_frame)"/>	The frame attached to the odometry system. default: "odom"
3	<param name="map_update_interval" value="5.0"/>	How long (in seconds) between updates to the map. Lowering this number updates the occupancy grid more often, at the expense of greater computational load. default: 5.0
4	<param name="maxUrange" value="6.0"/>	The maximum usable range of the laser. A beam is cropped to this value. default: 80.0
5	<param name="maxRange" value="8.0"/>	The maximum range of the sensor. If regions with no obstacles within the range of the sensor should appear as free space in the map, set maxUrange < maximum range of the real sensor <= maxRange.
6	<param name="sigma" value="0.05"/>	The sigma used by the greedy endpoint matching. default: 0.05
7	<param name="kernelSize" value="1"/>	The kernel in which to look for a correspondence. default: 1
8	<param name="lstep" value="0.05"/>	The optimization step in translation. default: 0.05
9	<param name="astep" value="0.05"/>	The optimization step in rotation. default: 0.05
10	<param name="iterations" value="5"/>	The number of iterations of the scanmatcher. default: 5
11	<param name="lsigma" value="0.075"/>	The sigma of a beam used for likelihood computation. default: 0.075

	Parameters	Discerption
		Table 18Params_gmapping
1	<param name="xmin" value="-50.0"/>	Initial map size (in metres). default: -100.0
2	<param name="ymin" value="-50.0"/>	Initial map size (in metres). default: -100.0
3	<param name="xmax" value="50.0"/>	Initial map size (in metres). default: 100.0
4	<param name="ymax" value="50.0"/>	Initial map size (in metres). default: 100.0
5	<param name="xmin" value="-10.0"/>	Initial map size (in metres). default: -100.0
6	<param name="ymin" value="-10.0"/>	Initial map size (in metres). default: -100.0
7	<param name="xmax" value="10.0"/>	Initial map size (in metres). default: 100.0
8	<param name="ymax" value="10.0"/>	Initial map size (in metres). default: 100.0

5.3.5 Running SLAM on the differential drive robot

- **Following are the commands to start with the mapping procedure.**

1) Start the robot simulation by using rsrobot_gazebo world:

$ roslaunch rsrobot_gazebo world.launch

2) We have to convert depth image to laser scan:

$roslaunch rsrobot_navigation laser_data.launch

3) Start the gmapping launch file by using the following command:

$roslaunch rsrobot_navigation rsrobot_gmapping.launch

4) **Start the keyboard teleoperation for manually navigating the robot around the environment, the robot can map its environment only if it covers the entire area:**

$rosrun teleop_twist_keyboard teleop_twist_keyboard.py

- The current Gazebo view of the robot and the robot environment is shown next. The environment is with obstacle around the robot.

Figure 52Robot environment

- We can launch RViz and add a display type called Map and the topic name as /map.
- We can start moving the robot inside the world by using key board teleoperation, and we can see a map building according to the environment. The following image shows the completed map of the environment shown in RViz:

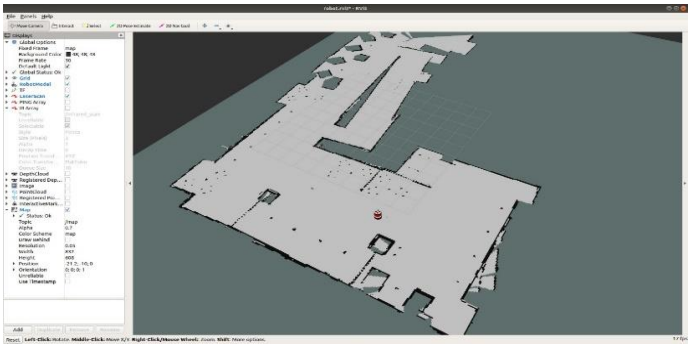


Figure 53Map in RViz

5) **We can save the built map using the following command, This command will listen to the map topic and save into the image,The map server package does this operation:-**

$ rosrun map_server map_saver -f maps

- Here maps is the name of the map file.
- The maps file is stored as two files:-

1) one is the YAML file which contains the map metadata and the image name.
2) second is the image which has the encoded data of the occupancy grid map.

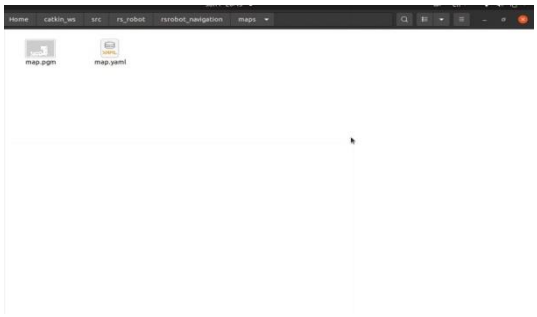


Figure 54maps file

- The saved encoded image of the map is shown next. If the robot gives accurate robot odometry data, we will get this kind of precise map similar to the environment. The accurate map improves the navigation accuracy through efficient path planning.

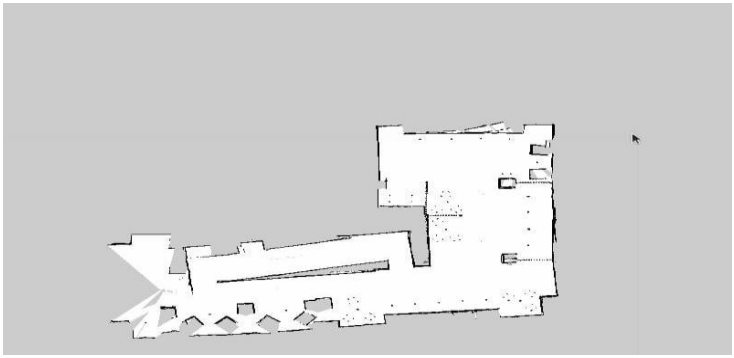


Figure 55map similar to the environment

- **The next procedure is to localize and navigate in this static map.**

5.3.6 Process of SLAM Related Nodes:

- sensor_node.
- teleop_twist.
- robot_core .
- slam_gmapping.
- map_server.

❑ Link explaining the process of drawing a map by gmapping:
https://drive.google.com/file/d/1Uao5hkHx5_1eh3DpeWx7H46gvf01rE4h/view?usp=drivesdk

5.4 Localization (AMCL):

5.4.1 Definition:

- The ROS AMCL package provide nodes for localizing the robot on a static map.
- The amcl node subscribes the laser scan data, laser scan based maps, and the tf information from the robot.
- The amcl node estimates the pose of the robot on the map and publishes its estimated position with respect to the map.
- If we create a static map from the laser scan data, the robot can autonomously navigate from any pose of the map using AMCL and the move_base nodes.
- The first step is to create a launch file for starting the amcl node.
- The amcl node is highly customizable; we can configure it with a lot of parameters.
- The list of parameters is available in the ROS package site (http://wiki.ros.org/amcl)

5.4.2 Creating an AMCL launch file :-

- He main task while creating a launch file for the amcl process is to set the parameters for the Amcl node.
- The Amcl node is the core node inside the ROS amcl package.
- This node is highly configurable and we can fine tune the parameters to improve
- accuracy.
- The parameters are mentioned at http: http://wiki.ros.org/amcl.
- The map file we created in the gmapping process is loaded here using the map_server node.
- Following is the rsrobot_amcl.launch file used in this robot.
- The launch file is placed in the rsrobot_navigation

5.4.3 the rsrobot_amcl.launch file:-

- It consists of :-

1) node amcl_pkg: in which parameters are defined.
2) Run(Load) Map Serve.

Params_AMCL:

	Paramaters	Discerption
1	<param name="odom_model_type" value="diff"/>	Which model to use, either "diff", "omni", "diff-corrected" or "omni-corrected". default: "diff"
2	<param name="odom_alpha5" value="0.1"/>	Translation-related noise parameter (only used if model is "omni"). default: 0.2
3	<param name="gui_publish_rate" value="10.0"/>	Maximum rate (Hz) at which scans and paths are published for visualization, -1.0 to disable. default: -1.0 Hz
4	<param name="laser_max_beams" value="60"/>	How many evenly-spaced beams in each scan to be used when updating the filter. default: 30
5	<param name="laser_max_range" value="12.0"/>	Maximum scan range to be considered; -1.0 will cause the laser's reported maximum range to be used. default: -1.0
6	<param name="min_particles" value="500"/>	Minimum allowed number of particles. default: 100
7	<param name="max_particles"	Maximum allowed number of particles. default:

	value="2000"/>	5000
8	<param name="kld_err" value="0.05"/>	Maximum error between the true distribution and the estimated distribution. default: 0.01
9	<param name="kld_z" value="0.99"/>	Upper standard normal quantile for (1 - p), where p is the probability that the error on the estimated distrubition will be less than kld_err. default: 0.99
10	<param name="odom_alpha1" value="0.2"/>	Specifies the expected noise in odometry's rotation estimate from the rotational component of the robot's motion. default: 0.2
11	<param name="odom_alpha2" value="0.2"/>	Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion. default: 0.2

Table 19Params_AMCL 1

	Paramaters	Discerption
1	<param name="odom_alpha3" value="0.2"/>	Specifies the expected noise in odometry's translation estimate from the translational component of the robot's motion. default: 0.2
2	<param name="odom_alpha4" value="0.2"/>	Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion. default: 0.2
3	<param name="laser_z_hit" value="0.5"/>	Mixture weight for the z_hit part of the model. default: 0.95
4	<param name="laser_z_short" value="0.05"/>	Mixture weight for the z_short part of the model. default: 0.1
5	<param name="laser_z_max" value="0.05"/>	Mixture weight for the z_max part of the model. default: 0.05
6	<param name="laser_z_rand" value="0.5"/>	Mixture weight for the z_rand part of the model. default: 0.05
7	<param name="laser_sigma_hit" value="0.2"/>	Standard deviation for Gaussian model used in z_hit part of the model. default: 0.2 meters
8	<param name="laser_lambda_short" value="0.1"/>	Exponential decay parameter for z_short part of model. default: 0.1
9	<param name="laser_model_type" value="likelihood_field"/>	Maximum distance to do obstacle inflation on map, for use in likelihood_field model. default: 2.0 meters

Table 20Params_AMCL 2

	Paramaters	Discerption
1	<param name="update_min_d" value="0.25"/>	Translational movement required before performing a filter update. default: 0.2 meters
2	<param name="update_min_a" value="0.2"/>	Rotational movement required before performing a filter update. default: π/6.0 radians
3	<param name="odom_frame_id" value="odom"/>	Which frame to use for odometry. default: "odom"
4	<param name="resample_interval" value="1"/>	Number of filter updates required before resampling. default: 2
5	<param name="transform_tolerance" value="1.0"/>	Time with which to post-date the transform that is published, to indicate that this transform is valid into the future. default: 0.1 seconds
6	<param name="recovery_alpha_slow" value="0.0"/>	Exponential decay rate for the slow average weight filter, used in deciding when to recover by adding random poses. A good value might be 0.001. default: 0.0 (disabled)
7	<param name="recovery_alpha_fast" value="0.0"/>	Exponential decay rate for the fast average weight filter, used in deciding when to recover by adding random poses. A good value might be 0.1. default: 0.0 (disabled)
8	<remap from="scan" to="$(arg scan_topic)"/>	This is an XML tag that is used in the ROS (Robot Operating System) framework to remap a topic name. The "scan" topic is being remapped to the value of the "scan_topic" argument, which can be set by the user. This allows for more flexibility in configuring ROS nodes and their communication channels.

Table 21Params_AMCL 3

- The robot will plan a path to that point and give velocity commands to the robot controller to reach that point.
- In the preceding image, we can see that we have placed a random obstacle in the robot path and that the robot has planned a path to avoid the obstacle.
- We can view the AMCL particle cloud around the robot by adding a Pose Array on RViz and the topic is /particle_cloud.

- Amcl is a probabilistic localization system for a robot moving in 2D.
- It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.

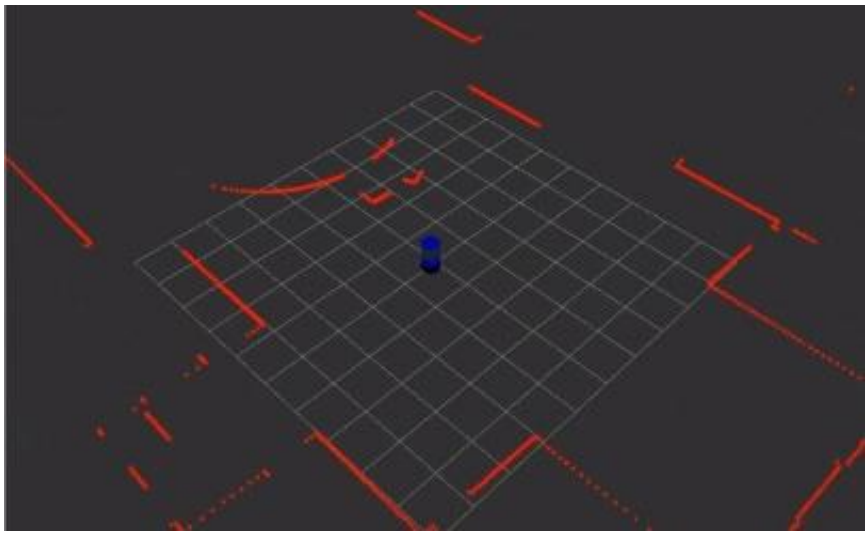


Figure 56particle_cloud

5.5 Navigation:

5.5.1 ROS Navigation stack:-

- The main aim of the ROS navigation package is to move a robot from the start position to the goal position, without making any collision with the environment.
- The ROS Navigation package comes with an implementation of several navigation related algorithms which can easily help implement autonomous navigation in the mobile robots.
- The user only needs to feed the goal position of the robot and the robot odometry data from sensors such as wheel encoders, IMU, and GPS, along with other sensor data streams such as laser scanner data or 3D point cloud from sensors like Kinect.
- The output of the Navigation package will be the velocity commands which will drive the robot to the given goal position.
- The Navigation stack contains implementation of the standard algorithms, such as SLAM, A *(star), Dijkstra, AMCL, and so on, which can directly be used in our application.

5.5.2 ROS Navigation hardware requirements: -

- The Navigation package will work better in differential drive and holonomic (total DOF of robot equals to controllable DOF of robots).

-Also, the mobile robot should be controlled by sending velocity commands in the form :

1) x: velocity.
2) y: velocity (linear velocity).
3) and theta :velocity (angular velocity).

• The robot should mount a planar laser somewhere around the robot. It is used to build the map of the environment.

• The Navigation stack will perform better for square and circular shaped mobile bases.

5.5.3 the explanations of all the blocks of the Navigation :-

- According to the Navigation setup diagram, for configuring the Navigation package for a custom robot, we must provide functional blocks which are interface to the Navigation stack.
- **All the blocks which are provided as input to the Navigational stack:**

1) **Odometry Source:**

- Odometry data of a robot gives the robot position with respect to its starting position.
- Main odometry sources are wheel encoders, MU, and 2D/3D cameras (visual odometry).
- The odom value should publish to the Navigation stack, which has a message type of nav_msgs/Odometry.
- The odom message can hold the position and the velocity of the robot.
- Odometry data is a mandatory input to the Navigational stack.

2) **Sensor Source:**

- We have to provide laser scan data or point cloud data to the Navigation stack for mapping the robot environment.
- This data, along with odometry, combines to build the global and local cost map of the robot.
- The main sensors used here are Laser Range finders or Kinect 3D sensors.
- The data should be of type sensor_msgs/LaserScan or sensor_msgs/PointCloud.

3) **Base_Controller:**

- The main function of the base controller is to convert the output of the Navigation stack, which is a twist (geometry_msgs/Twist) message, and convert it into corresponding motor velocities of the robot.

4) **Sensor Transforms/tf:**

- The robot should publish the relationship between the robot coordinate frame using ROS tf.

5) **Mapping (map_server):**

- Help to save/load the robot map.

6) **Amcl:**

- Allow localization of the robot.

Note:

- We can send a goal position to this node using a SimpleActionClient node.
- The move_base node subscribes the goal from a topic called move_base_simple/goal, which is the input of the Navigation stack, as shown in the previous diagram**.**
- When this node receives a goal pose, it links to components such as global_planner, local_planner, recovery_behavior, global_costmap, and local_costmap, generates the output which is the command velocity (geometry_msgs/Twist), and sends to the base controller for moving the robot for achieving the goal pose.

5.5.4 Working with Navigation packages:-

- By the move_base node it is clear that the move_base node takes input from sensors, joint states, tf, and odometry.
- **The move_base node:-**

  - The move_base node is from a package called move_base.
  - The main function of this package is to move a robot from its current position to a goal position with he help of other Navigation nodes.
  - The move_base node inside this package links the global-planner and the local-planner for the path planning, connecting to the rotate-recovery package if the robot is stuck in some obstacle and connecting global costmap and local costmap for getting the map.
  - The move_base node basically is an implementation of SimpleActionServer which takes a goal pose with message type (geometry_msgs/PoseStamped).

5.5.5 The packages which are linked by the move_base node:-

1) **global-planner:**

- This package provides libraries and nodes for planning the optimum path from the current position of the robot to the goal position, with respect to the robot map.
- This package has implementation of path finding algorithms such as A*, Dijkstra, and so on for finding the shortest path from the current robot position to the goal position.

2) **local-planner:**

- The main function of this package is to navigate the robot in a section of the global path planned using the global planner.
- The local planner will take the odometry and sensor reading, and send an appropriate velocity command to the robot controller for completing a segment of the global path plan.
- The base local planner package is the implementation of the trajectory rollout and dynamic window algorithms.

3) **rotate-recovery:**

- This package helps the robot to recover from a local obstacle by performing a 360 degree rotation.

4) **clear-costmap-recovery:**

- This package is also for recovering from a local obstacle by clearing the costmap by reverting the current costmap used by the Navigation stack to the static map.

5) **costmap-2D:**

- The main use of this package is to map the robot environment.
- Robot can only plan a path with respect to a map. In ROS, we create 2D or 3D occupancy grid maps, which is a representation of the environment in a grid of cells.
- Each cell has a probability value which indicates whether the cell is occupied or not.
- The costmap-2D package can build the grid map of the environment by subscribing sensor values of the laser scan or point cloud and also the odometry values.
- There are global cost maps for global navigation and local cost maps for local navigations.

5) **Map-Server:**

- Map server package allows us to save and load the map generated by the costmap-2D package.

6) AMCL:

- AMCL is a method to localize the robot in map.
- This approach uses particle filter to track the pose of the robot with respect to the map, with the help of probability theory.
- In the ROS system, AMCL can only work with maps which were built using laser scans.

7) Gmapping:

- The gmapping package is an implementation of an algorithm called Fast SLAM which takes the laser scan data and odometry to build a 2D occupancy grid map.

5.5.6 Some Important Concepts:-

1) Localizing on the map:

- The first step the robot is going to perform is localizing itself on the map.
- The AMCL package will help to localize the robot on the map.

2) Sending a goal and path planning:

- After getting the current position of the robot, we can send a goal position to the move_base node.
- The move_base node will send this goal position to a global planner which will plan a path from the current robot position to the goal position.
- This plan is with respect to the global costmap which is feeding from the map server.
- The global planner will send this path to the local planner, which executes each segment of the global plan.
- The local planner gets the odometry and the sensor value from the move_base node and finds a collision free local plan for the robot.
- The local planner is associated with the local costmap, which can monitor the obstacle(s) around the robot.

3) **Collision recovery behavior:**

- The global and local costmap are tied with the laser scan data.
- If the robot is stuck somewhere, the Navigation package will trigger the recovery behavior nodes, such as the clear costmap recovery or rotate recovery nodes.

4) **Sending the command velocity:**

- The local planner generates command velocity in the form of a twist message
- which contains linear and angular velocity (geometry_msgs/Twist), to the robot base controller.
- The robot base controller converts the twist message to the equivalent motor speed.

5.5.7 Creating a launch file for Navigation :-

- The main task while creating a launch file for the Navigation process is to set the parameters for the move_base node.
- The node we have to configure is the move_base node.
- The main parameters needed to configure are the global and local costmap parameters, the local planner, and the move_base parameters.
- The parameters list is very lengthy.
- We are representing these parameters in several YAML files.
- The parameters are mentioned at http://wiki.ros.org/navigation/Tutorials/RobotSetup.
- Each parameter is included in the param folder config inside rsrobot_navigation.

5.5.8 move_base.launch file:-

- It consists of node move_base through which we load files that contain params.
- There is inside launch files in folder rsrobot_navigation.

5.5.9 Config File:

- There is inside folder rsrobot_navigation.
- It contains a set of YAML files:-
    1) Costmap common parameters.
    2) the local costmap parameters.
    3) the global costmap parameters.
    4) the local planner parameters.
    5) the move_base parameters.

5.5.10 costmap_common_params.yaml file:-

	Parameters	Discerption
1	obstacle_range: 2.5	The "obstacle_range" parameter determines the maximum range sensor reading that will result in an obstacle being put into the costmap.
2	raytrace_range: 3.0	The "raytrace_range" parameter determines the range to which we will raytrace freespace given a sensor reading. Setting it to 3.0 meters
3	robot_radius: 0.2	In the case of specifying the footprint, the center of the robot is assumed to be at (0.0, 0.0) and both clockwise and counterclockwise specifications are supported.
4	inflation_radius: 1.55	The inflation radius should be set to the maximum distance from obstacles at which a cost should be incurred. The radius in meters to which the map inflates obstacle cost values.
5	transform_tolerance: 0.5	A tolerance on the goal point for the planner. The planner will attempt to create a plan that is as close to the specified goal as possible but no further than transform_tolerance away.
6	observation_sources:Scan	The "observation_sources" parameter defines a list of sensors that are going to be passing information to the costmap separated by spaces. Each sensor is defined in the next lines.

7	data_type: LaserScan	should be set to LaserScan or PointCloud depending on which message the topic uses.
8	topic: scan	should be set to the name of the topic that the sensor publishes data on.
9	Marking and clearing: true	parameters determine whether the sensor will be used to add obstacle information to the costmap, clear obstacle information from the costmap, or do both.

Table 22costmap_common_params.yaml file

5.5.11 Local_costmap_params.yaml file:-

	Parameters	Discerption
1	global_frame: odom	parameter defines what coordinate frame the costmap should run in, in this case, we'll choose the /map frame.
2	robot_base_frame: base_footprint	parameter defines the coordinate frame the costmap should reference for the base of the robot.
3	update_frequency: 1.0	parameter determines the frequency, in Hz, at which the costmap will run its update loop.
4	publish_frequency: 2.0	parameter determines the rate, in Hz, at which the costmap will publish visualization information.
5	static_map: false	parameter determines whether or not the costmap should initialize itself based on a map served by the map_server.
6	rolling_window: true	parameter to true means that the costmap will remain centered around the robot as the robot moves through the world.
7	resolution: 0.05	resolution (meters/cell) of the costmap
8	transform_tolerance: 0.5	A tolerance on the goal point for the planner. The planner will attempt to create a plan that is as close to the specified goal as possible but no further than transform_tolerance away.

9	cost_scaling_factor: 5	A scaling factor to apply to cost values during inflation.
10	inflation_radius: 0.55	The inflation radius should be set to the maximum distance from obstacles at which a cost should be incurred.

Table 23Local_costmap_params.yaml file

5.5.12 global_costmap_params.yaml file:-

	Parameters	Discerption
1	global_frame: map	parameter defines what coordinate frame the costmap should run in, in this case, we'll choose the /map frame.
2	robot_base_frame: base_footprint	parameter defines the coordinate frame the costmap should reference for the base of the robot.
3	update_frequency: 1.0	parameter determines the frequency, in Hz, at which the costmap will run its update loop.
4	publish_frequency: 0.5	parameter determines the rate, in Hz, at which the costmap will publish visualization information.
5	static_map: true	parameter determines whether or not the costmap should initialize itself based on a map served by the map_server.
6	transform_tolerance: 0.5	A tolerance on the goal point for the planner. The planner will attempt to create a plan that is as close to the specified goal as possible but no further than transform_tolerance away.
7	cost_scaling_factor: 10.0	A scaling factor to apply to cost values during inflation.
8	inflation_radius: 0.55	The inflation radius should be set to the maximum distance from obstacles at which a cost should be incurred.

Table 24global_costmap_params.yaml file

5.5.13 move_base_params.yaml file:-

	Parameters	Discerption
1	base_global_planner: global_planner/GlobalPlanner	The name of the plugin for the global planner to use with move_base, see pluginlib documentation for more details on plugins.
2	base_local_planner: dwa_local_planner/DWAPlannerROS	The name of the plugin for the local planner to use with move_base see pluginlib documentation for more details on plugins.
3	shutdown_costmaps: false	Determines whether or not to shutdown the costmaps of the node when move_base is in an inactive state. default: false
4	controller_frequency: 5.0	The rate in Hz at which to run the control loop and send velocity commands to the base. default: 20.0
5	controller_patience: 3.0	How long the controller will wait in seconds without receiving a valid control before space-clearing operations are performed. default: 15.0
6	planner_frequency: 0.5	The rate in Hz at which to run the global planning loop. default: 0.0
7	planner_patience: 5.0	How long the planner will wait in seconds in an attempt to find a valid plan before space-clearing operations are performed. default: 5.0
8	oscillation_timeout: 10.0	How long in seconds to allow for oscillation before executing recovery behaviors. A value of 0.0 corresponds to an infinite timeout. default: 0.0
9	oscillation_distance: 0.2	How far in meters the robot must move to be considered not to be oscillating. Moving this far resets the timer counting up to the oscillation_timeout .default: 0.5

10	**conservative_reset_dist: 0.1**	**The distance away from the robot in meters beyond which obstacles will be cleared from the costmap when attempting to clear space in the map.**
11	**cost_factor: 1.0, neutral_cost:55, lethal_cost: 253**	**Change the path of robot in different.**

Table 25move_base_params.yaml file

5.6 Dynamic Window Approach to local robot navigation on a plane:

- The dwa_local_planner package provides a controller that drives a mobile base in the plane.
- This controller serves to connect the path planner to the robot.
- Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location.
- Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map.
- This value function encodes the costs of traversing through the grid cells.
- The controller's job is to use this value function to determine dx,dy,dtheta velocities to send to the robot.
- **http://wiki.ros.org/dwa_local_planner#DWAPlannerROS.**

- **The basic idea of the Dynamic Window Approach (DWA) algorithm is as follows:**

1) Discretely sample in the robot's control space (dx,dy,dtheta).
2) For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3) Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity

to the goal, proximity to the global path, and speed Discard illegal trajectories (those that collide with obstacles).

4) Pick the highest-scoring trajectory and send the associated velocity to the mobile base.

5) Rinse and repeat.

5.6.1 base_local_planner_default_params.yaml file:-

	Paramaters	Discerption
1	max_trans_vel: 0.50	The absolute value of the maximum translational velocity for the robot in m/s. default: 0.55
2	min_trans_vel: 0.01	The absolute value of the minimum translational velocity for the robot in m/s. default: 0.1
3	max_vel_x: 0.50	The maximum x velocity for the robot in m/s. default: 0.55
4	min_vel_x: -0.025	The minimum x velocity for the robot in m/s, negative for backwards motion. default: 0.0
5	max_vel_y: 0.0	The maximum y velocity for the robot in m/s. default: 0.1
6	min_vel_y: 0.0	The minimum y velocity for the robot in m/s. default: -0.1
7	max_rot_vel: 0.30	The absolute value of the maximum rotational velocity for the robot in rad/s. default: 1.0
8	min_rot_vel: -0.30	The absolute value of the minimum rotational velocity for the robot in rad/s. default: 0.4
9	acc_lim_x: 1.25	The x acceleration limit of the robot in meters/sec^2. double, default: 2.5
10	acc_lim_y: 0.0	The y acceleration limit of the robot in meters/sec^2. default: 2.5

12	**acc_lim_theta: 5**	**The rotational acceleration limit of the robot in radians/sec^2. default: 3.2**
13	**acc_lim_trans: 1.25**	**The absolute value of the maximum translational velocity for the robot in m/s. default: 0.55**

Table 26 base_local_planner_default_params.yaml file 1

	Paramaters	Discerption
1	**prune_plan: false**	**Defines whether or not to eat up the plan as the robot moves along the path. If set to true, points will fall off the end of the plan once the robot moves 1 meter past them.**
2	**xy_goal_tolerance: 0.25**	**The tolerance in meters for the controller in the x & y distance when achieving a goal. default: 0.10**
3	**yaw_goal_tolerance: 0.1**	**The tolerance in radians for the controller in yaw/rotation when achieving its goal. default: 0.05**
4	**trans_stopped_vel: 0.1**	**The absolute value of the maximum rotational velocity for the robot in rad/s. default: 1.0**
5	**rot_stopped_vel: 0.1**	**The absolute value of the minimum rotational velocity for the robot in rad/s. default: 0.4**
6	**sim_time: 3.0**	**The amount of time to forward-simulate trajectories in seconds. default: 1.7**
7	**sim_granularity: 0.1**	**The step size, in meters, to take between points on a given trajectory. default: 0.025**
8	**angular_sim_granularity: 0.1**	**The step size, in meters, to take between points on a given trajectory. default: 0.025**
9	**path_distance_bias: 34.0**	**The weighting for how much the controller should stay close to the path it was given. default: 32.0**
10	**goal_distance_bias: 24.0**	**The weighting for how much the controller should attempt to reach its local goal, also controls speed. default: 24.0**

11	**occdist_scale: 0.05**	**The weighting for how much the controller should attempt to avoid obstacles. default: 0.01**
12	**twirling_scale: 0.0**	**The "twirling_scale" parameter is used to control how much the robot should rotate in place when it encounters an obstacle or reaches a dead end. A higher value means the robot will rotate more, while a lower value means it will rotate less.**

Table 27 base_local_planner_default_params.yaml file 2

	Paramaters	Discerption
1	**stop_time_buffer: 0.5**	**The amount of time that the robot must stop before a collision in order for a trajectory to be considered valid in seconds. default: 0.2**
2	**oscillation_reset_dist: 0.05**	**How far the robot must travel in meters before oscillation flags are reset. default: 0.05**
3	**oscillation_reset_angle: 0.2**	**refers to the angle at which the robot's movement is reset to avoid oscillation or back-and-forth movement. The value "0.2" could refer to a specific angle in radians or degrees**
4	**forward_point_distance: 0.3**	**The distance from the center point of the robot to place an additional scoring point, in meters. default: 0.325**
6	**scaling_speed: 0.25**	**The absolute value of the velocity at which to start scaling the robot's footprint, in m/s. default: 0.25**
7	**max_scaling_factor: 0.2**	**The maximum factor to scale the robot's footprint by. default: 0.2**
8	**vx_samples: 20**	**The number of samples to use when exploring the x velocity space. default: 3**
9	**vy_samples: 0**	**The number of samples to use when exploring the y velocity space. default: 10**
10	**vth_samples: 40**	**The number of samples to use when exploring the theta velocity space. default: 20**

11	**use_dwa: true**	**It is not recommended to use** **the dwa_local_planner::DWAPlanner on its own.**
12	**restore_defaults: false**	**refers to a parameter that determines whether the default values for certain settings should be restored or not. When set to "false", the navigation system will not restore the default values for any settings, and instead will use the current values that have been set by the user or program. This can be useful in situations where customized settings are preferred over the default ones.**

Table 28 base_local_planner_default_params.yaml file 3

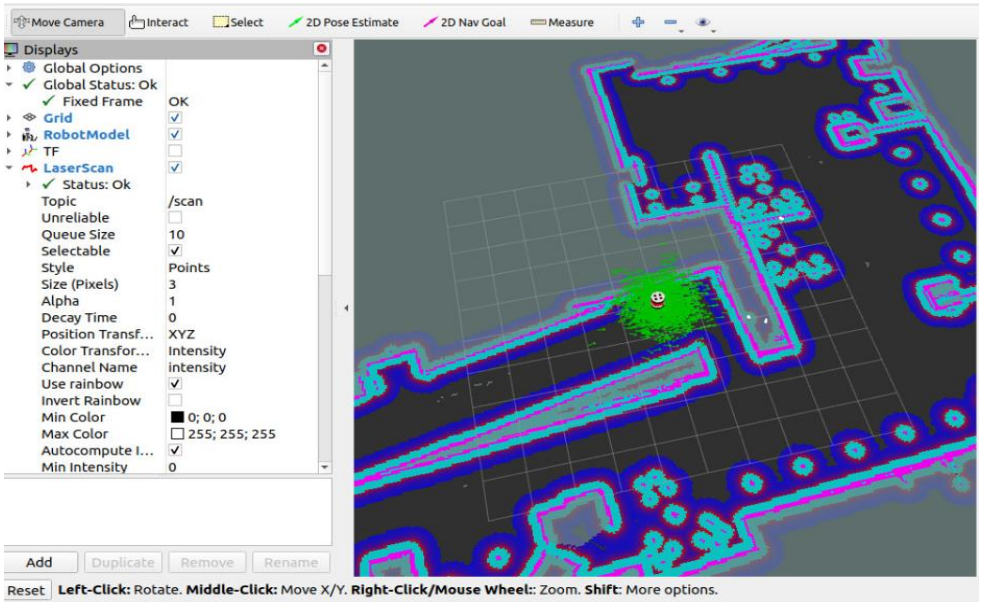


Figure 57Costmap

- **Important Notes:**

1) This package provides an implementation of the Dynamic Window Approach to local robot navigation on a plane.

2) Given a global plan to follow and a costmap, the local planner produces velocity commands to send to a mobile base.

3) This package supports any robot who's footprint can be represented as a convex polygon or circle, and exposes its configuration as ROS parameters that can be set in a launch file.

4) The parameters for this planner are also dynamically reconfigurable.

6

Conclusion & Future Work

6.1 Future work

1. **Enhanced Human-Robot Interaction (HRI)**: Future work could focus on improving the interaction between nurse robots and patients. This includes developing more advanced natural language processing (NLP) capabilities, better voice recognition, and more intuitive touch interfaces to ensure that patients feel comfortable and understood when interacting with robots.

2. **Advanced AI and Machine Learning**: Implementing more sophisticated AI and machine learning algorithms can enable nurse robots to learn from their interactions and improve their performance over time. This includes better decision-making abilities, predictive analytics for patient care, and adaptive learning to handle a wider range of tasks and scenarios.

3. **Design a Mobile App:** to Control the Robot instead of RVIZ

6.2 Conclusion

The use of ROS in nurse robots in healthcare facilities has several advantages. ROS provides a flexible and modular framework for developing robotic systems, making it easier to integrate different hardware and software components. This allows for the creation of customized robots that can perform specific nursing tasks efficiently.
In the case of nurse robots, ROS can enhance their capabilities in patient monitoring, medication delivery, and assistance with routine tasks, allowing them to support human nurses more effectively. Additionally, ROS provides tools for monitoring and controlling multiple robots simultaneously, which can help healthcare administrators manage their operations more efficiently.

Overall, the use of ROS in nurse robots has the potential to enhance patient care while also improving operational efficiency. However, it is important to note that there may be challenges associated with implementing this technology, such as high costs and the need for technical expertise.

References:

[1] “Wiki.” ros.org. http://wiki.ros.org/urdf.

[2] Josh. “Getting Ready for Ros Part 7: Describing a Robot with URDF.” Articulated Robotics, https://articulatedrobotics.xyz/ready-for-ros-7-urdf/.

[3] “Wiki.” ros.org. http://wiki.ros.org/xacro.

[4] Osrf. “URDF in Gazebo.” gazebo. https://classic.gazebosim.org/tutorials?tut=ros_urdf.

[5] Gazebo. https://gazebosim.org/.

[6] Osrf. “Gazebo Tutorials.” gazebo. https://classic.gazebosim.org/tutorials.

[7] Osrf. “Gazebo Plugins in Ros.” gazebo. https://classic.gazebosim.org/tutorials?tut=ros_gzplugins.

[8] “Wiki.” ros.org. http://wiki.ros.org/simulator_gazebo/Tutorials/ListOfMaterials.

[9] Osrf. “Color and Texture Models.” gazebo. http://classic.gazebosim.org/tutorials?tut=color_model&cat=.

[10] “Materials.” Materials - Procedural Generation for Gazebo. https://boschresearch.github.io/pcg_gazebo_pkgs/tutorials/simulation/materials/.

[11] 262588213843476. “Gazebo Color Palette with XCKD Color Names.” Gist. https://gist.github.com/naoki-mizuno/5e63a13597d5c5fe817c1600b829595e.

[12] “Simulation.” Simulation - DiffBot Differential Drive Mobile Robot. https://ros-mobile-robots.com/diffbot_gazebo/.

[13] “Wiki.” ros.org. http://wiki.ros.org/amcl.

[14] “Build Software Better, Together.” GitHub. https://github.com/topics/amcl.

[15] “Wiki.” ros.org. http://wiki.ros.org/gmapping.

[16] “Wiki.” ros.org. http://wiki.ros.org/navigation.

[17] automaticaddison, Author. “How to Set up the Ros Navigation Stack on a Robot.” Automatic Addison,https://automaticaddison.com/how-to-set-up-the-ros-navigation-stack-on-a-robot/.

[18] 262588213843476. “Gazebo Color Palette with XCKD Color Names.” Gist. https://gist.github.com/naoki-mizuno/5e63a13597d5c5fe817c1600b829595e.

[19] “Wiki.” ros.org. http://wiki.ros.org/costmap_2d.

[20] “Wiki.” ros.org. http://wiki.ros.org/base_local_planner.

[21] “Wiki.” ros.org. http://wiki.ros.org/nav_core.

[22] “Mastering Robot Operating System.” Mastering ROS for Robotics Programming. https://mastering-ros.com/.

[23] Joseph, Lentin, and Jonathan Cacace. Mastering Ros for robotics programming design, build, and simulate complex robots using robot operating system. Birmingham: Packt Publishing Limited, 2018.

Table of Figure

List of Tables