DE-42 (MTS) NOMAN AYAZ, MUZAMMIL ALI,

DESIGN AND DEVELOPMENT OF DELIVERY ROBOT (CONTROL SYS)

COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND
TECHNOLOGY RAWALPINDI
2024

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DE-42 MTS
PROJECT REPORT

DESIGN AND DEVELOPMENT OF DELIVERY ROBOT (CONTROL SYS)

Submitted to the Department of Mechatronics Engineering
in partial fulfillment of the requirements
for the degree of
Bachelor of Engineering
in
Mechatronics
2024

Sponsoring DS:
Dr Zohaib Riaz

[illegible]

Submitted By:
Noman Ayaz
Malik Muzammil

ACKNOWLEDGMENTS

We would like to take this opportunity to thank Almighty Allah for his countless blessings that he has bestowed upon us and gave us everything that we have today. We want to express our heartfelt gratitude to the many individuals who have supported and guided us throughout the completion of this Final Year Project. First and foremost, we extend our deepest appreciation to our project supervisor, Dr Zohaib and co-supervisor, Col. Dr. Kunwar Faraz for their unwavering support, guidance and insights. Their expertise, encouragement and constructive feedback have been instrumental in enabling us to shape this project and enhance our understanding of the subject matter. We would also like to thank Dr. Amir Hamza for his support, motivation and advice which we needed time to time to carry on moving forward. We are grateful to the officers of RDC most importantly Brig. Dr Aqib Pervez and Col. Dr. Rashid Naseer for giving us the opportunity to work on this project and provide their support in terms of guidance and mentorship. We are also grateful to the faculty and staff of the Department of Mechatronics engineering for providing a conducive learning environment and access to essential resources, without which this project would not have been possible. We would like to extend our thanks to our batchmates and friends who have been a source of helping hand and motivation. Their discussions, brainstorming sessions and support during times of stress have played a significant role in our personal and academic growth.

We are deeply indebted to our family for their unwavering support, encouragement and patience throughout this academic journey. From our upbringing to this day, they have done everything in their power to allow us to study in this institute where we could not have been able to without their support. Their belief and their sacrifices for us have been the driving force behind our accomplishments.

This project has been a tremendous learning experience for us. We are truly grateful to everyone for their involvement in our academic journey. In the end we would like to express our gratitude to all those people whose names we have not mentioned but who have, in one way or another, contributed to the successful completion of this project.

ABSTRACT

We have stepped into a new era of technology marked with automated systems where the focus is on less dependence on human systems and more dependence on machines. Many companies have evolved to incorporate fully automated system in their production facilities such as in their production plants where machines replace human labor in the assembly lines in their logistics, in packing and delivering etc. The use of automated systems allows for many benefits enhancing efficiency, reliability and cost-effectiveness in countries where human labor is expensive. In many companies there is a requirement of automated delivery systems as it allows for 24/7 operations without the need for any human hands to bridge the gap between fully and semi-automated systems. Addressing this gap, our project endeavors to provide a solution of indoor delivery of different payloads to users working in a company or an office environment. Our delivery robot can allow for efficient indoor deliveries in a diverse environment consisting of obstacles such as furniture or humans moving together, offering safety as well as reliability. Our aim is to enhance the performance of our robot to make it more adaptable to different areas where faster and efficient round the clock deliveries are a concern. Using sensor technologies such as the LIDAR and algorithms such as SLAM, our proposed system enables accurate navigation, obstacle avoidance and path planning. Its control system is designed such that it can work through a diverse environment and can keep an eye on unexpected obstacles ensuring that it manages to stop and move around them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

<u>LIST OF SYMBOLS</u>

Acronyms

SOP Standard Operating Procedures

ToF time-of-flight

IMUs inertial measurement units

UAVs unmanned aerial vehicles

EOL End of Life

ROS Robot Operating System

AMCL Adaptive Monte Carlo Localization

CHAPTER 1 - INTRODUCTION

1.1 Project Background:

Recently we have seen a lot of technological process especially in the field of robotics where this technological revolution has directly affected the way industries like manufacturing healthcare and logistics are being run. The most noticeable use that is being seen in this field are the autonomous deliveries robots. This opens the pathway to the use of the robots for the delivery of goods without any human intervention and so it has a chance of reducing the cost in the form of human labour. It can also be applied very effectively in the cities where a lot of different obstacles may be encountered by such a robot.

Along with this sensor technology has also evolved with new and latest sensors, while at the same time computational power of the embedded systems is increasing. This has helped in the development of smart robotic systems which can navigate in complex and dynamic environments with precision. In this regard the combination of high-accuracy sensors with computing systems, can be used to create a smart control system for delivery robots.

This work considers the concept, development, and implementation of a control system for a 4-wheel drive delivery robot, using RP Lidar A1M8 and Raspberry Pi 4 as its basis. The major purpose is to develop a navigation system that is reliable and effective in enabling the robot to travel around any indoor environment, being able to avoid obstacles and reach its destination safely.

1.2 Motivation:

The motivation behind this project stems mainly from our degree of mechatronics where our job is to design such systems. Also, with the growing demand for autonomous delivery solutions in the world we decided to work on such a project. With the rise of online orders and same-day delivery services, there was an increase in automated robots that can deliver around the neighborhood having the ability to track their location using GPS. These robots could navigate around the general public and could climb sidewalks and inclines moving efficiently and precisely. However, we saw that there was a need for indoor deliveries where robots cannot rely on GPS for their positioning around an area and the existing outdoor robots could not navigate effectively in indoor environments.

The integration of the systems used in this project also provided us a unique opportunity to develop a control system capable of real-time environment perception and navigation. By taking in the data provided by the Lidar and the computational capability of the Raspberry Pi, we aimed to create a control system that can accurately map its surroundings, identify obstacles and make intelligent path planning decisions in real-time.

1.3 Objectives:

The primary objective of this thesis is to design implement and evaluate a control system for a 4-wheel drive delivery robot using Lidar and Raspberry Pi 4.

Specifically, the key objectives include:

1. Integrating the RP Lidar A1M8 sensor with the Raspberry Pi 4 platform to enable real-time environment perception.
2. Developing algorithms for obstacle detection, mapping, and path planning.
3. Implementing control algorithms to enable autonomous navigation of the delivery robot in dynamic environments.
4. Conducting comprehensive experimental evaluations to assess the performance and reliability of the proposed control system.
5. Analyzing the results, identifying limitations, and proposing avenues for future research and improvement.

By achieving these objectives, this thesis aims to contribute to the advancement of autonomous robotic systems for delivery applications, with the ultimate goal of enhancing efficiency, reliability, and scalability in urban logistics operations.

CHAPTER 2 – LITERATURE REVIEW

2.1 Overview

Autonomous delivery robots represent a promising solution to the challenges of delivering items in outdoor urban environments where they rely mainly on GPS to guide them to their destination. In this section, we review the existing literature related to delivery robots, control systems, RP Lidar A1M8 and Raspberry Pi 4, providing valuable insights into the current field of robotics and identifying research that motivates the present study.

2.2 Previous Research

Numerous studies have delved into the integration of lidar sensors with robotic platforms, aiming to enhance navigation and obstacle avoidance capabilities. Design of a low-cost autonomous delivery robot was proposed by Fernando Sancho, Andrés Olivares, and Luis Payá [1]. The authors mainly focused on the development of a robot that was suited mainly for indoor environments. Many factors were in consideration during their course of work. First of all, there was the demand of autonomous delivery systems in different areas such as offices, hospitals, warehouses etc. The system designed had to be low cost so that it could be available to a larger range of people. The authors discuss the application of sensors such as laser scanners to map the environment and the use of algorithms such as SLAM that allow the robot to plan its path and navigate the environment.

The control of delivery robots utilizing RP Lidar and Raspberry Pi was mainly utilized by the company TurtleBot [2] which has made multiple designs consisting of the TurtleBot 3 and the TurtleBot 4 having multiple features but mainly consisting of the main two systems mentioned above. It is a low-cost robot kit that was created in 2010 by Melonee Wise and Tully Foote. TurtleBot is a self-navigating robot that utilizes SLAM and navigation to allow it to move autonomously. It can build a map of its surroundings and then move around in that area. It works using the Robot operating system which we will come to later.

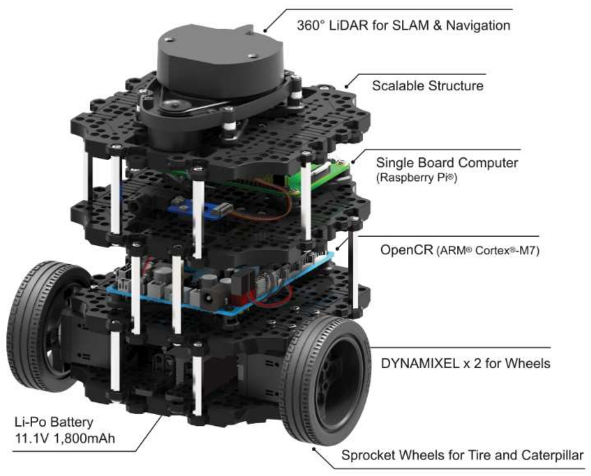

360° LiDAR for SLAM & Navigation
Scalable Structure
Single Board Computer (Raspberry Pi®)
OpenCR (ARM® Cortex®-M7)
DYNAMIXEL x 2 for Wheels
Li-Po Battery 11.1V 1,800mAh
Sprocket Wheels for Tire and Caterpillar

Figure 1 TurtleBot 3 Burger Model with components labelled [2]

2.2.1 Delivery Robots and Their Applications

Ground-based delivery robots, aerial drones and sidewalk robots are among the various types of delivery robots that have been proposed and developed for different applications. Starship Technologies [3] and Amazon [4] are some companies that have deployed ground-based delivery robots for package and grocery delivery in urban environment. These robots are equipped with sensors and navigation systems to autonomously navigate sidewalks and avoid obstacles. They are outdoor robots that rely on Global Positioning Modules that help them navigate by providing them their location and also their target destination according to the world maps.

Figure 2 A Starship Technologies food delivery robot [3]

2.2.2 Control Systems for Autonomous Robots

The design of control systems in robotic platforms has been a topic of recurring interest among researchers, who have been looking for different ways to regulate robot behaviour. Thrun and his team (2004) suggested a mobile tour-guide robot called MINERVA, [5] with a simple architecture and the navigation and interaction control algorithms implemented for autonomous behaviour. This research shows that the effective control systems are essential to robot operating in environments with other robots and humans.

Moreover, researches are doing to integrate perception systems, such as lidar sensors, cameras, and inertial measurement units (IMUs), into control systems for the purpose of real-time environment awareness and decisions. Deng et al. (2021) wrote an elaborate paper on the role of perception in enhancing robot navigation and mapping that relies on simultaneous localization and mapping (SLAM) [6].

2.2.3 Hardware Integration

Hardware and software integration in robotics play a pivotal role in realizing the functionalities of autonomous systems. Vallecorsa et al. (2018) [7] discussed the implementation of a data acquisition system for a particle physics experiment utilizing FPGA-based hardware and software development techniques. While it is not directly related to domain of delivery robots, this study underscores the importance of seamless integration between hardware and software components for the successful deployment of robotic systems.

2.3 Introduction to RP LIDAR A1M8

The RP Lidar A1M8, developed by RoboPeak [8], stands as a pivotal technology in the realm of robotics, particularly in the domain of perception and navigation. This compact 2D laser scanner offers an impressive array of features, including a wide field of view, high angular resolution, and extended range sensing capabilities. Such attributes make the RP Lidar A1M8 an invaluable tool for various applications in robotics and automation.

It comes along with its own serial port and USB interface which allows it to be connected to any computer easily.

Figure 3 The RP LIDAR A1M8 [8]

2.3.1 Working Principle

The RP Lidar A1M8 operates on the principle of time-of-flight (ToF) sensing, where laser pulses are emitted and the time taken for the reflected light to return is measured. This allows for the precise calculation of distances to objects in the surroundings, enabling accurate mapping, localization, and obstacle detection. With a scanning range of up to 12 meters and a scanning frequency of 6,000 samples per second, the RP Lidar A1M8 provides rich, real-time data essential for autonomous navigation tasks.

2.3.2 Applications

In robotics applications, the RP Lidar A1M8 finds widespread utility in tasks such as simultaneous localization and mapping (SLAM) [6], environment perception, and obstacle avoidance. Its ability to generate detailed, 360-degree maps of the surroundings facilitates robust localization of the robot within its environment, even in dynamic and cluttered settings. Moreover, the high-resolution scanning capability enables precise detection and avoidance of obstacles, ensuring safe navigation of the robot.

Beyond robotics, the RP Lidar A1M8 has also found applications in areas such as industrial automation, unmanned aerial vehicles (UAVs), and smart cities. Its compact form factor, low power consumption, and reliability make it well-suited for integration into various systems requiring accurate environmental sensing capabilities.

2.4 Raspberry Pi 4 and its capabilities in robotics

2.4.1 Introduction

Raspberry Pi 4 stands out as a versatile and cost-effective single-board computer that has garnered significant attention in the realm of robotics due to its remarkable capabilities and extensive community support. It comes in a small size along with a wide range of options that enable it to well suited to its need.

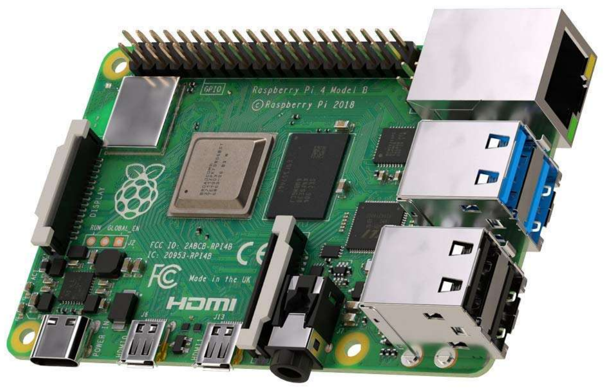


Figure 4 The Raspberry Pi 4 [9]

2.4.2 Features

The Raspberry Pi 4 is an extraordinary computing device that boasts impressive computational power, incorporating a quad-core ARM Cortex-A72 processor, which can reach up to 1. 5GHz and options for different levels of RAM including 2GB to 8GB [9]. Through these enhanced computational capabilities, Raspberry Pi 4 becomes capable of executing complex algorithms and computations that are necessary for real-time control, data processing and decision-making in the autonomous robotic systems. For the delivery robot project, Raspberry Pi 4 is used as the brain, with control algorithms being executed, interfacing with the RP Lidar A1M8 sensor and running all navigation tasks with the precision and efficiency required.

In addition, Raspberry Pi 4 provides a vast range of connectivity options such as USB ports, Ethernet, Wi-Fi and Bluetooth that make the integration with sensors, actuators and communication modules, which are essential for robotics application, easier and more efficient. Therefore, the connectivity flexibility enables the robot to be in communication with the environment, interact with the external systems, and use additional sensors for the improvement of the perception and localization.

It features a range of software and development tools like the popular Linux-based OS Raspbian, Python programming language, and a vast array of libraries and frameworks well-suited for robotics programming. This streamlines the software development and deployment processes as scientists and developers are allowed to focus on innovation and experimentation in robotics while the software-related complexities are left aside.

To sum up, the Raspberry Pi 4 is a powerful and versatile computing platform that is a perfect fit for robotics applications. It is fast enough, has plenty of connectivity options, and it is also supported by software necessary for the control of the autonomous delivery robots equipped with RP Lidar A1M8 sensor.

2.5 Software

2.5.1 Ubuntu

The Ubuntu OS [10] is a popular Linux distribution known for its user-friendly interface, stability and extensive software making it a suitable choice for robotics projects and development environments.

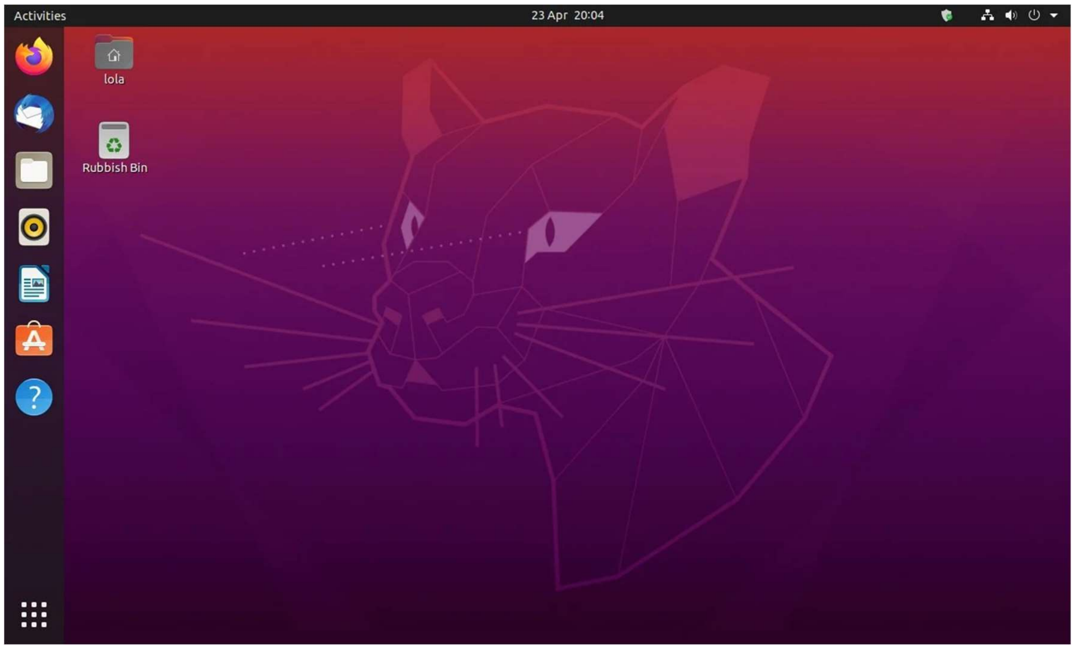


Figure 5 Ubuntu 20.04 Desktop Interface

While the Raspberry Pi 4 can run various operating systems, including Raspbian (a Debian-based distribution optimized for Raspberry Pi), Ubuntu offers an alternative choice for developers familiar with its environment. Ubuntu for Raspberry Pi [11] provides a familiar Linux environment with access to a wide range of software packages and development tools, facilitating software development and deployment for the control system of the delivery robot. It has many versions however the one we are utilizing is the ubuntu 20.04 LTS desktop which is a stable build and that can support our software and hardware.

It provides developers with access to a vast repository of software packages, libraries and development tools. This enables developers to utilize existing software solutions and frameworks for robotics applications, including ROS (Robot Operating System), Gazebo simulator and Python programming language, among others. Ubuntu's compatibility with these software tools streamlines the development process and enhances the capabilities of the delivery robot's control system.

2.5.2 Robot Operating System (ROS)

Figure 6 The ROS LOGO [12]

ROS, short for Robot Operating System, [12] is a flexible and powerful framework widely used in robotics projects for developing, simulating and deploying robotic applications. Originally developed by Willow Garage, ROS has evolved into a standard platform embraced by researchers, developers and enthusiasts worldwide. ROS facilitates the integration of hardware and software components, enabling seamless communication, data exchange and collaboration within robotic systems.

ROS has undergone significant evolution, with ROS 1 being the initial iteration and ROS 2 representing the next-generation platform. While ROS 1 has been widely adopted and utilized in numerous robotics projects, ROS 2 introduces several enhancements to address limitations and incorporate modern software engineering practices. ROS 2 offers improved real-time capabilities, better support for multi-robot systems, and enhanced security features, making it well-suited for advanced robotics applications. However, ROS 1 remains relevant and supported, particularly in existing projects and environments where migration to ROS 2 may not be immediately feasible.

ROS 1 encompasses several versions including ROS Indigo, ROS Kinetic, and ROS Melodic, each introducing enhancements and improvements to the framework. These versions have been pivotal in shaping the ROS ecosystem and have been extensively utilized in a wide range of robotics projects from research laboratories to industrial applications.

Feature	ROS 1	ROS 2
Real-time Support	Limited	Improved real-time support
Cross-platform Support	Primarily Linux	Linux, Windows, macOS
Security Features	No built-in security	SROS2 (Secure ROS 2) for authentication, encryption, access control
Build System	catkin	colcon
Performance	Limited scalability and real-time capabilities	Better performance for distributed and real-time systems
Launch System	roslaunch	Improved launch system (launch)
Multi-robot Support	Limited	Improved multi-robot support
Node Execution	All nodes typically run in separate processes	Nodes can run in separate processes or as components in a single process

Table 1 Key Differences between ROS 1 and ROS 2

2.5.2.1 ROS Architecture

In ROS (Robot Operating System), several key components facilitate communication, data exchange, and coordination within robotic systems. [13]

- ROS Nodes

ROS nodes are computational units that perform specific tasks within a robotic system. Nodes can represent sensors, actuators, control algorithms, perception modules, or any other functional component of the robot. ROS nodes communicate with each other by publishing and subscribing to topics, enabling modular and distributed architecture in robotics applications.

- ROS Topics

ROS topics are named communication channels that facilitate the exchange of messages between ROS nodes. Topics are used to transmit data, events or commands related to specific aspects of the robot's operation. For example, topics can convey sensor data e.g., camera images, lidar scans, control commands e.g., velocity commands for motors, or status updates e.g., diagnostic information. Topics follow a publisher-subscriber messaging pattern allowing multiple nodes to exchange information asynchronously.

- Publishers

  Publishers are ROS nodes responsible for sending messages on a specific topic. A publisher node generates messages containing data or commands and publishes them to the corresponding topic. For instance, a camera node may act as a publisher, continuously capturing images from the camera sensor and publishing them on the "/camera/image" topic. Publishers provide data to other nodes that have subscribed to the same topic, allowing them to access and process the information.
- Subscribers

  Subscribers are ROS nodes that receive messages from a specific topic. A subscriber node subscribes to a topic of interest and listens for messages published on that topic by one or more publishers. When a message is published on the subscribed topic, the subscriber node receives the message and processes its contents accordingly. For example, a navigation control node may subscribe to the "/cmd_vel" topic to receive velocity commands for controlling the robot's movement. Subscribers consume data provided by publishers, enabling coordination and collaboration between different components of the robotic system.
- ROS Services

  ROS services enable synchronous communication between ROS nodes by facilitating request-response interactions. Unlike topics, which allow for asynchronous communication, services provide a mechanism for nodes to request specific actions or information from other nodes and receive a response. Services are defined by a pair of messages: one for the request and one for the response. When a node requests a service, it sends a request message containing relevant data to the service server. The service server processes the request, performs the requested action, and sends a response message back to the requesting node. Services are commonly used for tasks such as querying sensor data, executing actions, or performing computations on-demand within a robotic system.
- Parameter Server

  The parameter server is a centralized key-value storage system used for storing and retrieving dynamic configuration parameters within a ROS environment. Parameters stored on the parameter server can include configuration settings, tuning parameters, sensor calibration data, or any other runtime parameters needed by ROS nodes. Nodes can access and modify parameters on the parameter server during runtime, allowing for flexible configuration and runtime adjustments. The parameter server provides a convenient mechanism for managing configuration parameters across multiple nodes within a robotic system. Additionally, ROS provides tools such as the ROS Parameter Server API and the "rosparam" command-line tool for interacting with the parameter server, enabling users to query, set, and delete parameters programmatically or from the command line.

2.5.3 ROS Noetic

ROS Noetic [14] is the latest long-term support (LTS) release of ROS 1, introduced in May 2020. Built upon Ubuntu 20.04 (Focal Fossa), ROS Noetic is designed to provide stability, reliability and compatibility for robotics projects over an extended period. As the successor to ROS Melodic, ROS Noetic inherits many of its features while introducing new capabilities and optimizations. Its EOL (End of Life) date is expected to be around May 2025 so it is still supported at the time of writing this thesis. In this project we utilized the ROS Noetic distribution which is the final ROS 1 distribution.

Some of the Key Feature of ROS Noetic are:

- Enhanced Compatibility

  ROS Noetic is compatible with Ubuntu 20.04 and Python 3, aligning with the latest software standards and ensuring long-term support for robotics development.

- Improved Performance

  Getting optimizations and updates to core packages, ROS Noetic offers improved performance and efficiency enabling faster development and deployment of robotic applications.

- Expanded Ecosystem

  ROS Noetic benefits from an extensive set of packages, libraries and tools contributed by the ROS community, providing developers with a wealth of resources for robotics development. It has a vast amount of documentation and resources available on multitudes of work done on different projects. This allows it to have an edge over ROS 2 which does not have a lot of documentation as its community is comparatively small and still in growing phase.

- Simulation and Visualization

  ROS Noetic includes all the basic simulation and visualization tools such as Gazebo and Rviz, enabling developers to simulate and visualize robotic systems in virtual environments, facilitating testing, debugging, and validation of algorithms and behaviours.
- Integration with Robotics Projects

  ROS Noetic offers several advantages. Developers can leverage ROS Noetic to implement sensor integration, perception algorithms, navigation strategies and control systems for the delivery robot. Additionally, ROS Noetic facilitates simulation-based development, enabling developers to prototype and test robotic behaviours in simulated environments before deploying them on physical hardware.

CHAPTER 3: METHODOLOGY

3.1 Working Theory

The navigation of mobile robots indoors using lidar follows the principle of sensor-based navigation and closed-loop control. The Lidar delivers the function of continuous scanning of the robot's surroundings by applying the process of sending out laser pulses and accordingly measuring the time taken for the returned light to come. [15] The ultrasonic sensor output is analysed by Raspberry Pi 4 and based on this sensor information the distance map of the robot's environment is constructed (the environment consists of obstacles, walls or any other object).

Lidar generates a 2D point cloud that is used by the Raspberry Pi 4 for identifying obstacles and determining the position with respect to its surrounding. Multiple algorithms, for example SLAM are applied to define an unknown environment and a robot's position and orientation level within this environment. Environmental data combined with the robot's current position gives control system a set of parameters, which allows it to calculate an optimal routing from the robot's current position to its target point.

By using path planning algorithms, like A* (A-star) or Dijkstra's algorithm, the shortest or safest journey can be determined, where the objects picked up by a lidar sensor avoid obstacles. The Raspberry Pi 4 sends out command for the motors to follow the route that might be comparatively curved and the Raspberry Pi 4 also sends out command to the motors. It is determined by the planned path and the intended trajectory. These instructions control the rate as well as the direction of the motors so that the robot's movement in the given path is done smoothly and with accuracy. An encoders or an additional sensor's feedback is incorporated into the robot, guaranteeing that it adheres to the planned trajectory and is maintaining the desired orientation. The control system runs in a closed-circuit manner using feedback, to continually monitor the robot's movement and to alter the motor commands in real-time in order to correct any deviations from the programmed locations.

The feedback control system performs this function by constantly comparing the robot's real movement to the one already demanded by the designer, so it keeps robust and responsive navigation principle as the robot is able to adapt to the dynamics and/or unanticipated obstacles discovered on its way during the mission. In short, a robot will use a Raspberry Pi 4 to process sensory data and perceive the environment, will plan the path and navigate the obstacles, and will finally move the motor with the Lidar. Through the adoption and utilization of these controllers and the integration of closed-loop control algorithms, the robot will be able to accomplish tasks in highly dynamic and unstructured environments in an effective, reliable and accurate way.

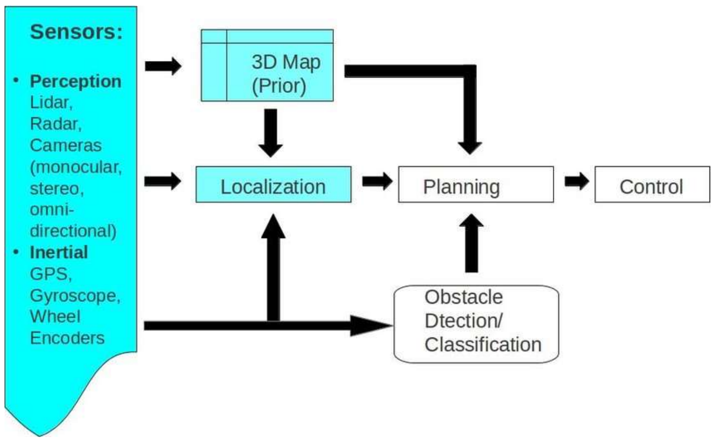

Sensors:
• Perception Lidar, Radar, Cameras (monocular, stereo, omni-directional)
• Inertial GPS, Gyroscope, Wheel Encoders
3D Map (Prior)
Localization
Planning
Control
Obstacle Dtection/ Classification

Figure 7 The basic working block diagram [16]

3.2 Overview

The RP Lidar A1M8 2D laser scanner made by RoboPeak, [8] has a wide field of view, high angular resolution and long ranging sensing ability. The sensor, created for the purpose of robotic and automation applications can provide precise distance calculation and environment mapping within 360 degrees, an attribute that makes it an excellent sensory tool for the navigation, mapping and obstacle detection operation.

The RP Lidar A1M8 and the Raspberry Pi 4 are connected together. The Raspberry Pi can be called the brain of the system while the Lidar is the main sensing element or simply the eye of the robot. In terms of its computational abilities, the Raspberry Pi 4 model 4 is not to be underrated, which can be implemented in connecting most hardware devices, sensing, data processing, and executing the control in real-time. Alongside both of these components Arduino Mega 2560 comes in which is connected serially to the Raspberry Pi 4 and its I/0 pins connect to the motor drivers for the motor control.

The RP Lidar A1M8 and the Raspberry Pi 4 are integrated to fulfill the mission of the robotic system through communication and collaboration. Thus, both components ensure that the robots work smartly and efficiently.

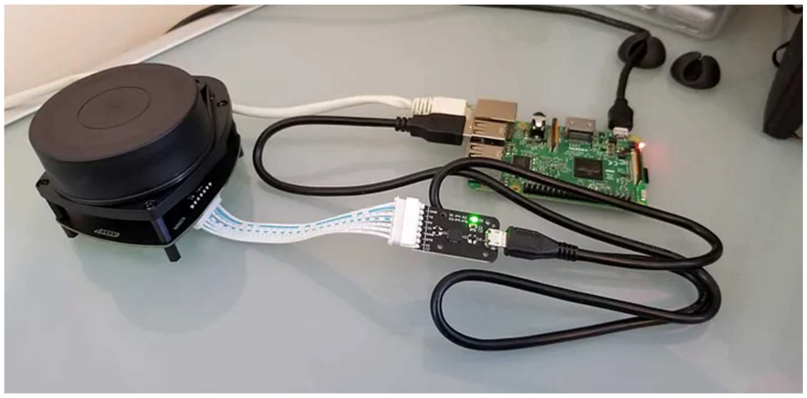


Figure 8 Connection of RP Lidar and Rpi 4 [17]

3.3 Hardware Setup

We are using the RP Lidar A1M8 that comes with a built-in serial port and USB interface. A ribbon cable and a board with a micro-USB port is included with the box. A micro-USB cable is needed to connect the board to the Raspberry Pi 4. It is powered through the same USB port and does not require external power. Arduino Mega 2560 is used for driving the motors of the robot. Arduino has I/O pins to connect it to the motor drivers and the encoders output is also connected to these pins. It has a serial communication port that connects to the Raspberry Pi 4 using a cable to allow communication between the two.

- Hardware Configuration

For higher level control we are using the Raspberry Pi 4 that runs the required software and algorithms and interprets all the data coming from the Lidar. For lower-level motor control we are using the Arduino Mega 2560 as the on-board controller.

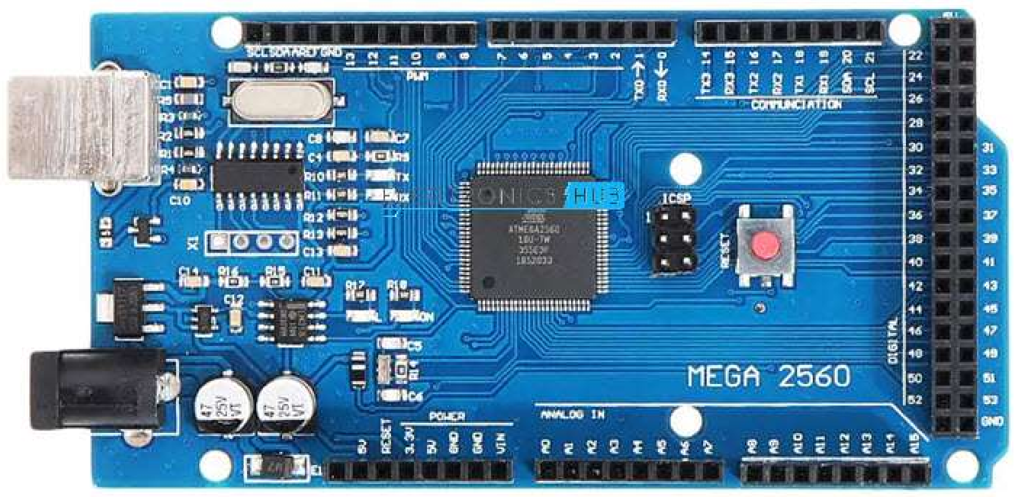


Figure 9 Arduino MEGA 2560 [17]

The Arduino Mega 2560 [17] is a microcontroller board based on the Atmega2560 processor, offering a significant increase in memory and input/output pins compared to standard Arduino boards. The Arduino Mega 2560 is well-suited for lower-level motor control in the project for a number of reasons. It has an abundance of I/O pins for ample connectivity with sensors, motor drivers and actuators. Pulse Width Modulation (PWM) is an essential part of motor control where the Mega provides multiple PWM pins allowing precise control of motor speed by varying the duty cycle. It has numerous libraries such as the ROS library that can allow it to interface with a raspberry Pi operating on ROS. It communicates with the Raspberry Pi 4 via serial communication protocols such as UART or I2C. This enables the Raspberry Pi to send high-level commands or setpoints to the Arduino for motor control, while the Arduino handles the low-level execution of control algorithms and interfacing with motor drivers. Moreover, using Arduino to drive the motors allows us to prevent any mishap or accident with the Raspberry Pi as we do not have to utilize any of its I/O pins.

- Software Configuration

The Raspberry Pi 4 running with the integrated software libraries and drivers is capable of interpreting the RP Lidar A1M8. Depending on the applications that require data collection, processing and visualization, either RPLIDAR SDK or ROS (Robot Operating System), or other open-source libraries that can be used to aid those purposes, might help the user to achieve objectives.

- Data Acquisition and Processing

  Raspberry Pi 4 gets data direct from the RP Lidar A1M8 in real time, capturing distance measurements and angle information from the laser scans. This type of data is processed by algorithms via the ROS nodes that are running and publishing this information, assuring environmental perception, obstacle detection, and the localization capabilities.

- Control and Navigation

  The Raspberry Pi 4 amplifies the given the processed data from the lidar to deploy complex control commands to the motors that power the robot allowing it to autonomously navigate, glimpse paths forward and avoid obstacles. The Raspberry Pi 4 executes control algorithms that guarantee smooth and accurate operation of the robotic system in changing environments due to its liveness and response.

3.4 Software Setup

3.4.1 Ubuntu

We are using Ubuntu 20.04 version in this project. This version was selected as it is a stable release and is supported by the ROS 1 Noetic distribution. In the latest Ubuntu 22.04 version we had installed ROS however it failed to properly install and gave errors.

To install ubuntu we had to use an SD card as it is the only storage type used in Raspberry Pi. A software provided by Raspberry Pi called the Raspberry Pi Imager is used to flash the OS in the SD card. We insert the card into a computer and run the software where we select the correct partition of the SD card. Then we select the correct OS in this case, the Ubuntu 20.04. For this version we can only install the Ubuntu Server OS which lacks the GUI of a full desktop and only runs the terminal where we can run commands.

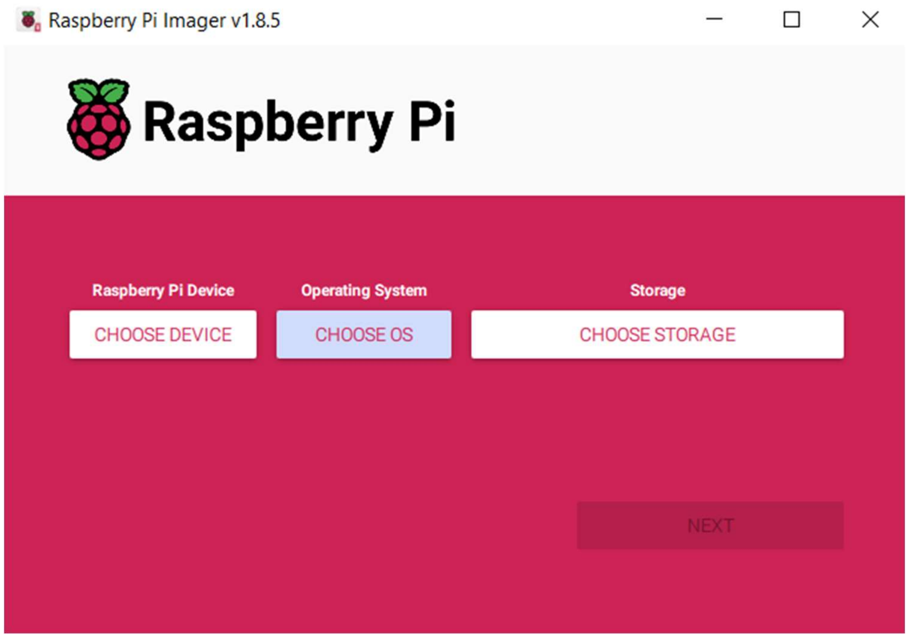
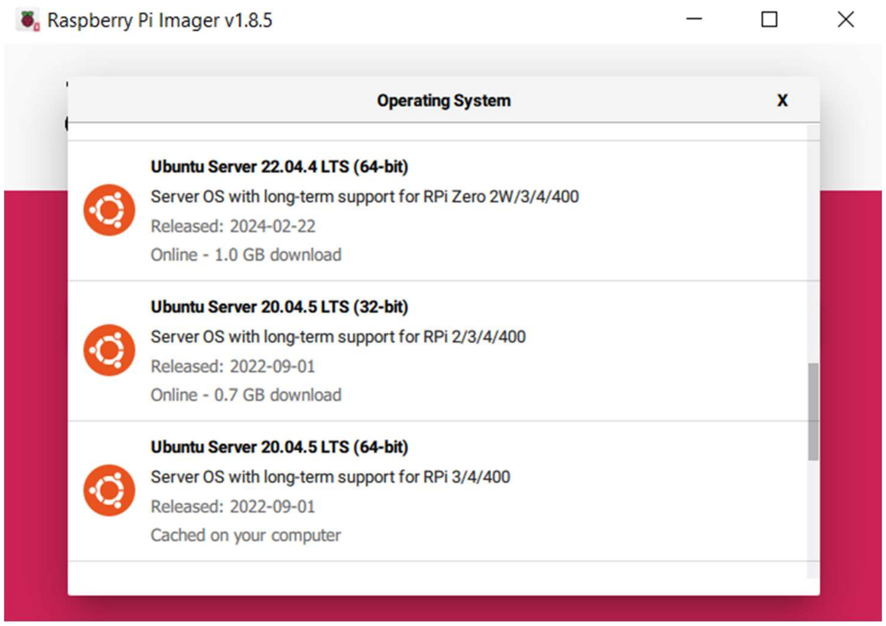


Figure 10 The Raspberry Pi Imager



Ubuntu Server 22.04.4 LTS (64-bit)
Server OS with long-term support for RPi Zero 2W/3/4/400
Released: 2024-02-22
Online - 1.0 GB download

Ubuntu Server 20.04.5 LTS (32-bit)
Server OS with long-term support for RPi 2/3/4/400
Released: 2022-09-01
Online - 0.7 GB download

Ubuntu Server 20.04.5 LTS (64-bit)
Server OS with long-term support for RPi 3/4/400
Released: 2022-09-01
Cached on your computer

Figure 11 Only Ubuntu server OS is available for 20.04 version

After the OS is installed in the SD card, we can place it in the Raspberry Pi and connect the power cable to it to turn it on as this version of the Pi does not have a power switch. After booting the Pi, we are introduced to the Ubuntu terminal where firstly we have to login using the username and password both as “ubuntu”. Then we can run the commands to connect to a Wi-Fi network and then install the Ubuntu Desktop that has the required graphical user interface.

After the desktop is installed, we can again open the terminal to install ROS. Run the command “sudo apt update” to update the Debian package. Then run “sudo apt install ros-noetic-desktop-full” to install the complete desktop plus 2D/3D simulators and 2D/3D perception packages.

3.4.1.1 Gazebo

Gazebo [18] is a powerful simulation tool which is a part of the robot operating system and is used in robotics for modelling, simulating and testing of systems in virtual environments. It is an integral component for developing and validating robotic applications and has a realistic and flexible simulation environment for experimentation and testing.

It is seamlessly integrated with ROS Noetic, allowing developers to use its capabilities within the ROS environment. Through ROS interfaces, Gazebo can communicate with ROS nodes, exchange sensor data, and execute control commands, enabling close integration between simulation and actual robotic systems.

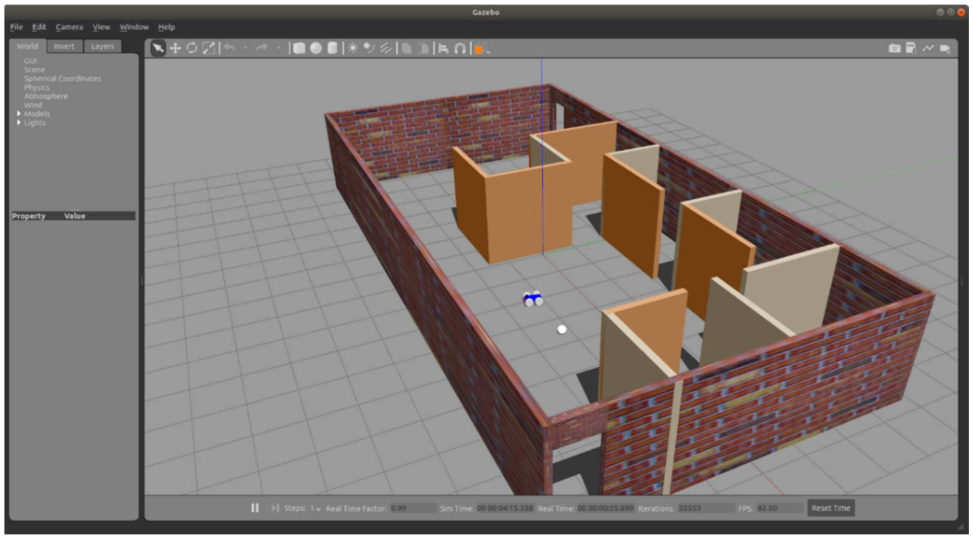


Figure 12 Gazebo simulation of a robot in an indoor environment [19]

Some Key Features of Gazebo are:

- High-Fidelity Simulation:

  Gazebo offers a detailed simulation environment with realistic physics, dynamics and sensor models that correspond to those of the real world. Via simulations developers create a variety of robotic platforms, sensors and environments that are both precise and accurate. Therefore, they can test and validate complex algorithms and behaviours thoroughly before running them on their designs.

- Customizable Environments:

  Gazebo permits the construction of custom surroundings specifically meant for distinct robotics applications. From including indoor environments that have obstacles and structures to outdoor terrains that display terrain variations, developers could produce those environments that resemble the scenarios in which real-world robot systems discover themselves.

- Sensor Simulation:

  Gazebo supports simulation of a wide range of sensors commonly used in robotics including cameras, lidars, sonars, and IMUs (Inertial Measurement Units). Developers can configure sensor parameters, generate realistic sensor data, and evaluate sensor fusion algorithms within the simulation environment.

- ROS Integration:

  Gazebo does not only support ROS 1 Noetic but also works with the nodes that are present in ROS, eliminating the need of human intervention for the bidirectional communication. Through ROS topics, services and parameters, which can be accessed and manipulated in Gazebo, data exchange of sensor data, injection of control signals, and visualization of the actual state of the robot is facilitated in real-time.

- Plugin Architecture:

  The gazebo software relies on a plugin architecture which makes it possible to add new features to the simulator and even configure different simulation behaviours. ROS

Noetic 1 comes with an extensive collection of Gazebo plugins tuned for specialists in robotics. In cases of robots, sensors and controllers, these can range up to gazebo robot models, sensor models and controller interfaces.

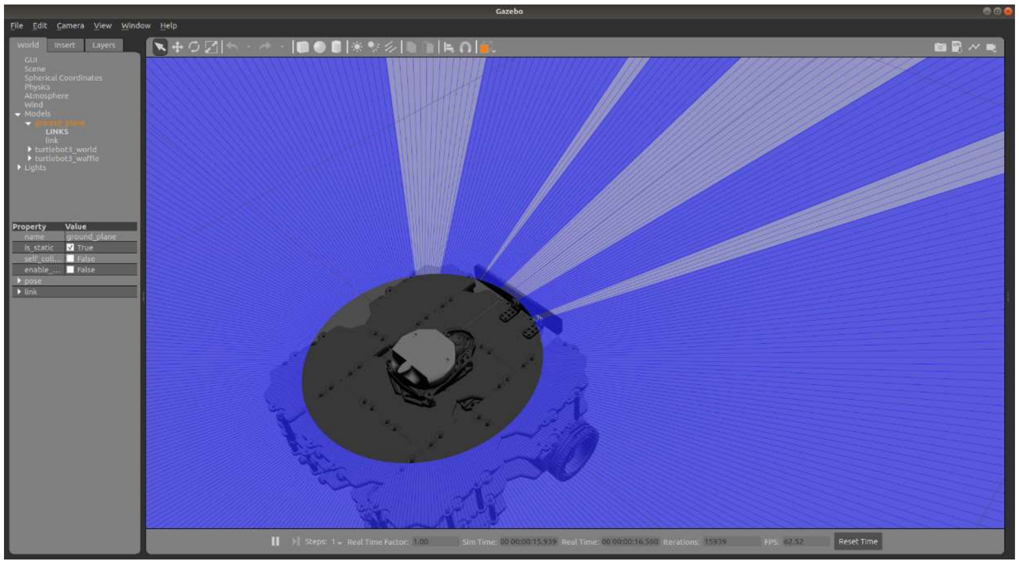
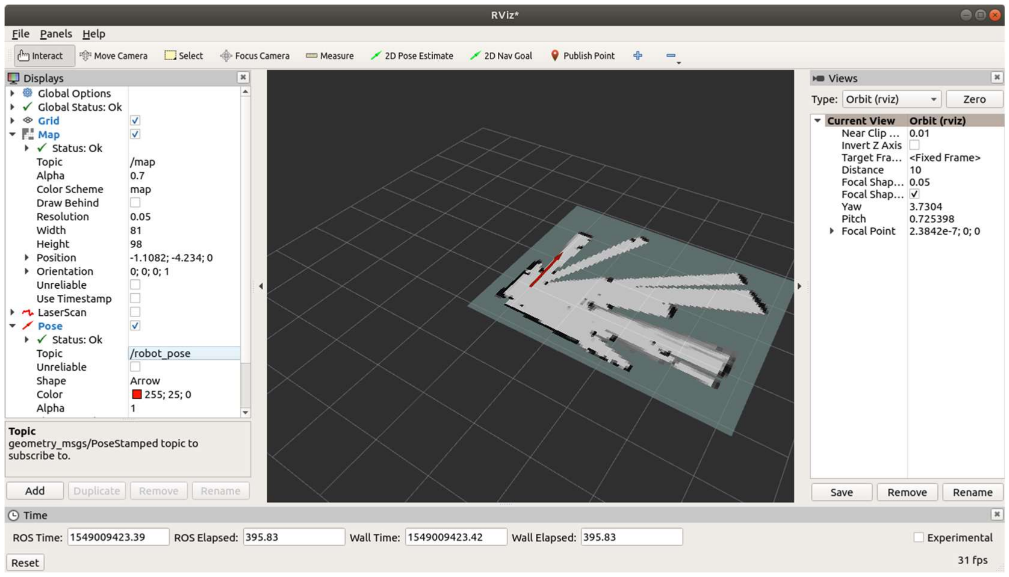


Figure 13 Autonomous bot in Gazebo showing laser scan (in blue) [20]

3.4.1.2 RViz

RViz [21] short for ROS Visualization, is a powerful 3D visualization tool which is a part of ROS, widely used in robotics projects for visualizing robot states, sensor data and environment models. RViz serves as an essential component for debugging, monitoring and visualizing robotic systems along with Gazebo providing developers with insights into the behaviour and performance of their applications.


RViz*
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Views
Time
Add Duplicate Remove Rename
Save Remove Rename
Zero
Reset
Experimental

```
Global Options
Global Status: Ok
Grid                ✓
Map                 ✓
  Status: Ok
  Topic             /map
  Alpha             0.7
  Color Scheme      map
  Draw Behind       ☐
  Resolution        0.05
  Width             81
  Height            98
  Position          -1.1082; -4.234; 0
  Orientation       0; 0; 0; 1
  Unreliable        ☐
  Use Timestamp     ☐
LaserScan           ☐
Pose                ✓
  Status: Ok
  Topic             /robot_pose
  Unreliable        ☐
  Shape             Arrow
  Color             🟥 255; 25; 0
  Alpha             1

Topic
geometry_msgs/PoseStamped topic to
subscribe to.

Type: Orbit (rviz)
Current View        Orbit (rviz)
  Near Clip ...     0.01
  Invert Z Axis     ☐
  Target Fra...     <Fixed Frame>
  Distance          10
  Focal Shap...     0.05
  Focal Shap...     ✓
  Yaw               3.7304
  Pitch             0.725398
  Focal Point       2.3842e-7; 0; 0

ROS Time: 1549009423.39  ROS Elapsed: 395.83  Wall Time: 1549009423.42  Wall Elapsed: 395.83
31 fps
```

Figure 14 RViz showing map building using SLAM [22]

Some Key Features of RViz in ROS are:

- Customizable Visualization

  RViz is versatile and it allows developers to have complete control of their visualizations. Users can select the appearance of visualizations and arrange them in a way that is tailored to their needs. Varying from robot models and data from sensors to point clouds and trajectories, RViz is adaptable for visualizing almost anything in the robotics field.

- Real-Time Visualization

  RViz is a visualization tool that enables real-time monitoring and analysis of robot states and sensor data. This allows developers to watch the program run while comparing the expected and actual results. It allows for live debugging and adaptation of algorithms ensuring the high standard of robot systems resulting in a reliable operation.

- Interactive Controls

  RViz allows the user to do the manipulations of visualizations by means of interactive controls, like pan, zoom or rotate. This interactivity prompts users to analyze different points of view across the simulated world, zoom in on important items, and acquire a good understanding about the physical relationships between robot products and their surroundings.

- Integration with TF (Transform Library)

  RViz wraps TF (Transform) package from the ROS community in such a way that you can obtain visualization of coordinate frames, transformations between coordinate frames and so on in the robotic system. This feature allows for correct spatial representation of robot's components and helps to coordinate all elements within robotic systems like sensors, actuators and others which are essential.

- Plugin Architecture

  ROS Visualization has a plugin system which allows the developers to create plugins to change the application behaviour or add a new interface or visualization. With the ROS Noetic version of RViz, you can use many plugins that are robotics applications specific, for example, sensors data visualization, robot trajectories and 3D environment models.

3.4.2 Fusion 360

Fusion 360 [23] is a powerful, cloud-based 3D CAD, CAM and CAE tool developed by Autodesk. It integrates design, engineering and manufacturing into a single platform, making it an ideal choice for developing complex mechanical systems, including robotic structures. With its comprehensive suite of tools and user-friendly interface, Fusion 360 is widely used in various fields such as product design, mechanical engineering, and industrial design. It allows us to create URDF (Unified Robot Description Format) files of our robot.

The unified robotics description format (URDF) [24] is an extensible markup language (XML) file type that includes the physical description of a robot. It is essentially a 3-D model with information around joints, motors, mass, etc. The URDF files are then run through the Robot Operating System (ROS). The data from the URDF file informs the human operator what the robot looks like and is capable of before they begin operating the robot.

Fusion 360 offers a robust set of 3D modeling tools that allow users to create detailed and accurate models of their robotic components. A fusion to URDF script [25] exists on GitHub that allows users to export the model of their robot to URDF format. We have to download the files and from command prompt we can install the script according to the instructions provided by the creator. After designing the robot using the various modeling techniques offered in this software such as solid modeling, surface modeling and mesh modeling we can run the script to generate a URDF file that can be opened in Rviz and Gazebo.

3.5 Algorithms And Other Tools

3.5.1 SLAM

Simultaneous Localization and Mapping (SLAM) [26] is a fundamental problem in robotics concerned with constructing a map of an unknown environment while simultaneously estimating the robot's position and orientation within that map. It works using sensors such as the Lidar to build a map of the robots' surroundings and also localize itself with the map. SLAM algorithms play a crucial role in enabling robots to autonomously navigate and explore unknown environments, making them essential for tasks such as robotic exploration, mapping and localization.

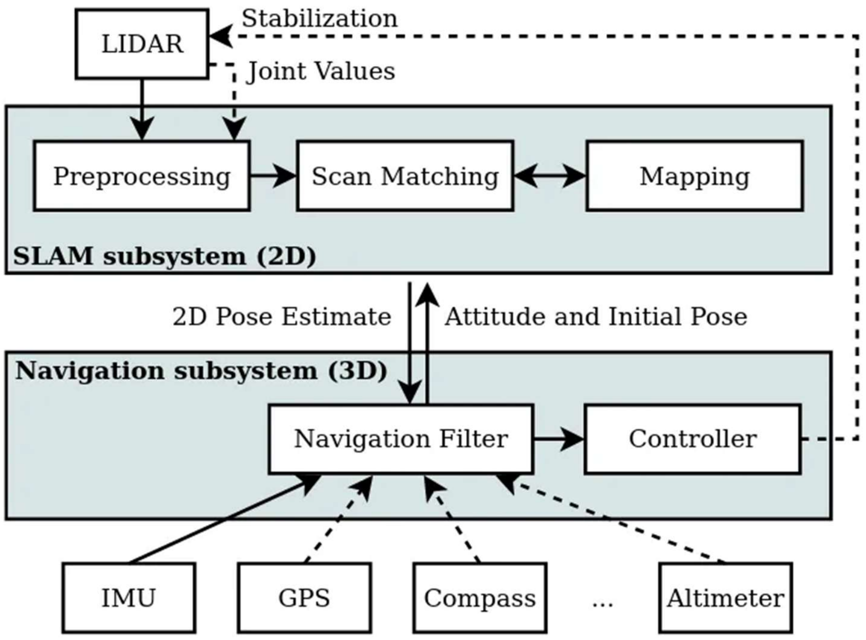


Figure 15 Overview of mapping and navigation system using SLAM [27]

There are several SLAM algorithms available, each with its own characteristics and suitability for different scenarios.

Few of some algorithms that we considered for this project are [28]:

- GMapping

  GMapping is a popular grid-based SLAM algorithm that uses the FastSLAM framework to construct a map of the environment. It represents the environment as a grid map and estimates the robot's position using a particle filter. GMapping is well-suited for scenarios with 2D lidar sensors making it a suited to this project.

- Cartographer

  Cartographer is a real-time SLAM library developed by Google. It utilizes a combination of scan matching and loop closure techniques to construct highly accurate 2D and 3D maps of indoor environments. Cartographer is well-suited for scenarios with 2D lidar sensors and offers robust localization and mapping capabilities.

- Hector SLAM

  Hector SLAM is a fast, robust and lightweight SLAM algorithm designed for mobile robots equipped with 2D lidar sensors. It only utilizes data from Lidar to successively generate a map as it moves along the environment. It uses very less computational power and puts less load on the CPU so it is particularly suitable for scenarios where computational resources are limited making it a good fit for Raspberry Pi-based systems.

- Google Cartographer (2D and 3D)

  Google Cartographer is a widely-used SLAM algorithm that is capable of producing highly accurate 2D and 3D maps using lidar and IMU sensor data. It consists of state-of-the-art optimization techniques to refine the map and robot trajectory, making it suitable for demanding mapping and localization tasks. It however requires more computational resources compared to other algorithms and may put excessive load on the Raspberry Pi system.

- ORB-SLAM2

  ORB-SLAM2 is a feature-based SLAM algorithm capable of producing accurate 3D maps using monocular, stereo, or RGB-D cameras. Although designed for camera-based SLAM, it can potentially be adapted to utilize lidar data for mapping and localization tasks. ORB-SLAM2 offers real-time performance and robustness to dynamic environments making it suitable for complex scenarios.

The GitHub community [29] has many resources related to SLAM where many developers have shared their work that can applied to implement these algorithms.

Algorithm	Key Features	Accuracy	Computational Requirements	Suitability	Examples of Use
GMapping	2D SLAM, works well in structured environments	Moderate	Moderate	Indoor environments, simple scenarios	TurtleBot, indoor robots
Hector SLAM	High update rate, no odometry needed	High	High	UAVs, high-speed robots	UAV mapping, fast-moving robots
Cartographer	2D/3D SLAM, real-time loop closure, submaps	High	High	Complex environments, large maps	Indoor and outdoor robots, UAVs
ORB-SLAM2	Monocular, stereo, RGB-D, robust loop closure	High	High	Visual SLAM applications	Drones, AR applications, robotics

Table 2 Differences between the various SLAM Algorithms

3.5.2 Path Planning

Path planning and trajectory planning [30] in the field of Robotics play a crucial role in enabling the robot to navigate from its current location to a specified goal location efficiently and safely. A robot when given a setpoint to move to has multiple paths to choose from. To determine the most optimal path involves evaluation of certain conditions. The distance and time taken is the most important criteria for allowing a robot to plan its path.

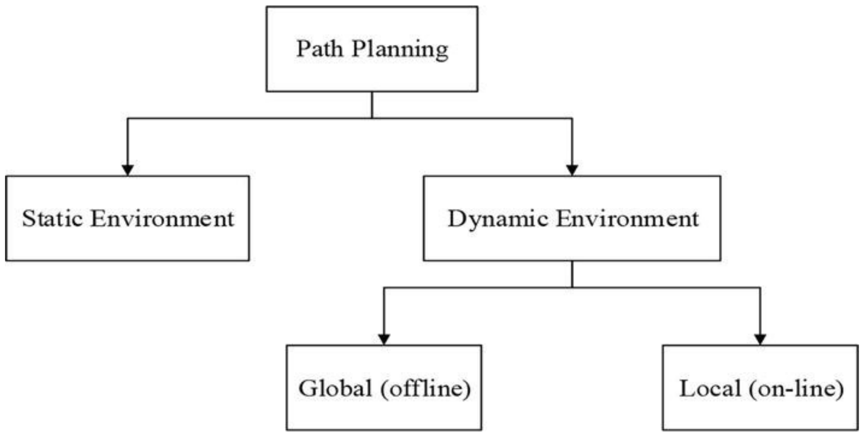


Figure 16 Mobile robot path planning classification [31]

Path planning can be classified into two classes based on the environment in which the robot moves [31]. In guiding the path-planning approach and selection of algorithms used to navigate these classes are important.

1. Static environment

   This is the class of path planning where the robot is in an environment of static obstacles such as indoor objects like furniture, walls, doors etc. The environment does not move with respect to the robot so it is simpler to plan a path and start moving.

2. Dynamic environment

   This class contains both static and moving obstacles in the environment for example moving humans inside an office. Path planning is complex as the robot has to avoid moving objects in its path and adjust its position while moving. The planned path has to change and robot has to make changes in its trajectory.

The dynamic path planning can be further subdivided into global and local path planning approaches, each serving its own purpose in the robot's navigation process.

Global Path Planning:

Global path planning algorithms aim to find an optimal path from the robot's current position to the goal location while considering the entire environment. These algorithms typically operate on a map of the environment, which may be provided in advance or constructed using sensor data and mapping techniques. Common global path planning algorithms include:

1. **Dijkstra's Algorithm**

   Dijkstra's Algorithm is a classic shortest path algorithm that finds the shortest path from a source node to all other nodes in a weighted graph. In the context of robotic path planning, Dijkstra's Algorithm can be used to find the shortest path from the robot's current position to the goal location while considering obstacles and terrain features.
2. **A* (A-star) Algorithm**

   It is a widely-used heuristic search algorithm that efficiently finds the shortest path from a start node to a goal node in a graph. A* Algorithm combines the benefits of Dijkstra's Algorithm with heuristic estimates of the remaining distance to the goal, enabling faster convergence to the optimal path. A* Algorithm is particularly well-suited for robotic path planning tasks where computational efficiency is essential.

Local Path Planning:

Local path planning algorithms focus on generating a collision-free trajectory for the robot to follow within its immediate vicinity. These algorithms operate based on local sensor data and aim to navigate the robot around obstacles while following the global path generated by the global planner. Common local path planning algorithms include:

1. **Velocity Obstacle (VO) Method:** The Velocity Obstacle Method is a reactive navigation algorithm that computes safe velocities for the robot by considering its current velocity and the velocities of nearby obstacles. By analyzing the relative motion between the robot and obstacles, the VO Method generates collision-free trajectories for the robot to follow, ensuring safe navigation in dynamic environments.
2. **Dynamic Window Approach (DWA):** The Dynamic Window Approach is a reactive local path planning algorithm that considers the robot's kinematic constraints and dynamic feasibility to generate feasible velocity commands. DWA evaluates a set of candidate trajectories within a dynamically feasible window and selects the trajectory that optimally balances path following and obstacle avoidance objectives.

3.6 Simulations

3.6.1 Fusion 360 Model

After opening Fusion 360 we can click on start a new project and start working on the design of our robot. We created a new component for the base chassis of our robot. Using the sketching and extrusion tools we designed the chassis also making sure that it has mounting points for the wheels and any other components. For the wheels we created a new component and designed one wheel using the sketch and revolve tools. After checking that the dimensions are accurate, we duplicate the wheel component for all four wheels positioning them correctly relative to the chassis as in the sketch of the robot. The design was kept relatively simple as the dimensions of the wheel base and the wheel separation were the important requirements for further simulations ROS. For the design of the Lidar, we made a base and a cylinder above it and positioned it at the top where we will mount the actual part. Then for final assembly of the robot we use joints and constraints to assemble all components. This ensures that parts move correctly in relation to each other.

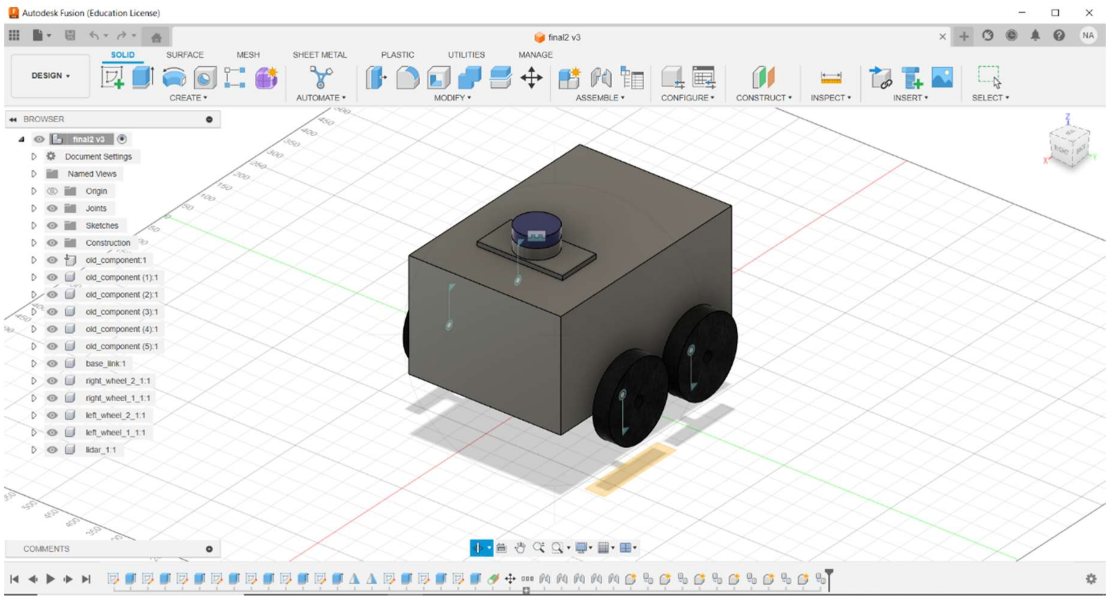


Figure 17 Model of our robot made in Fusion 360

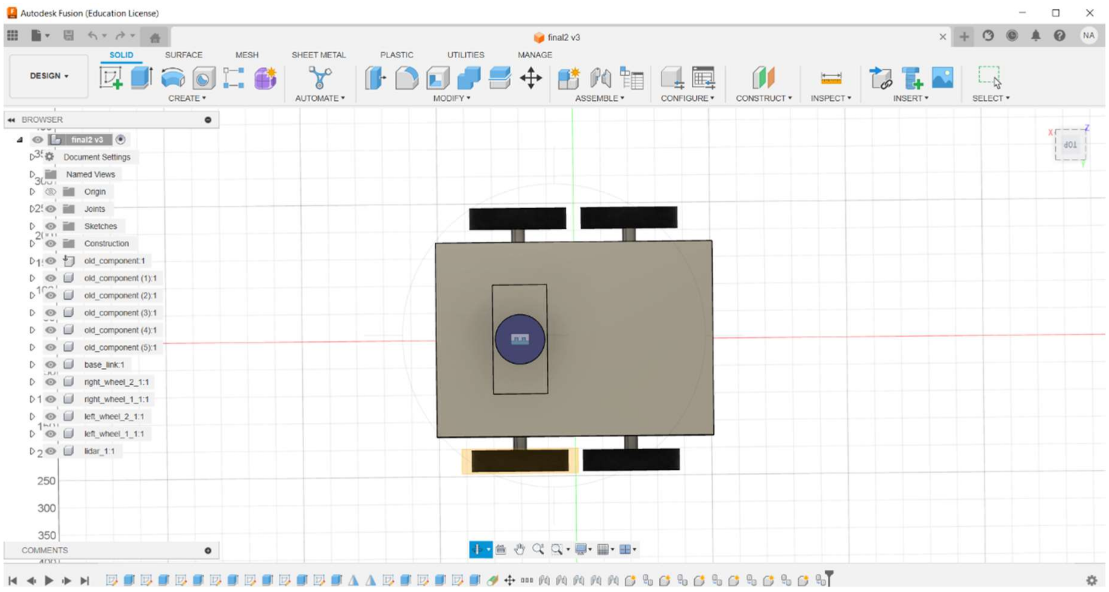
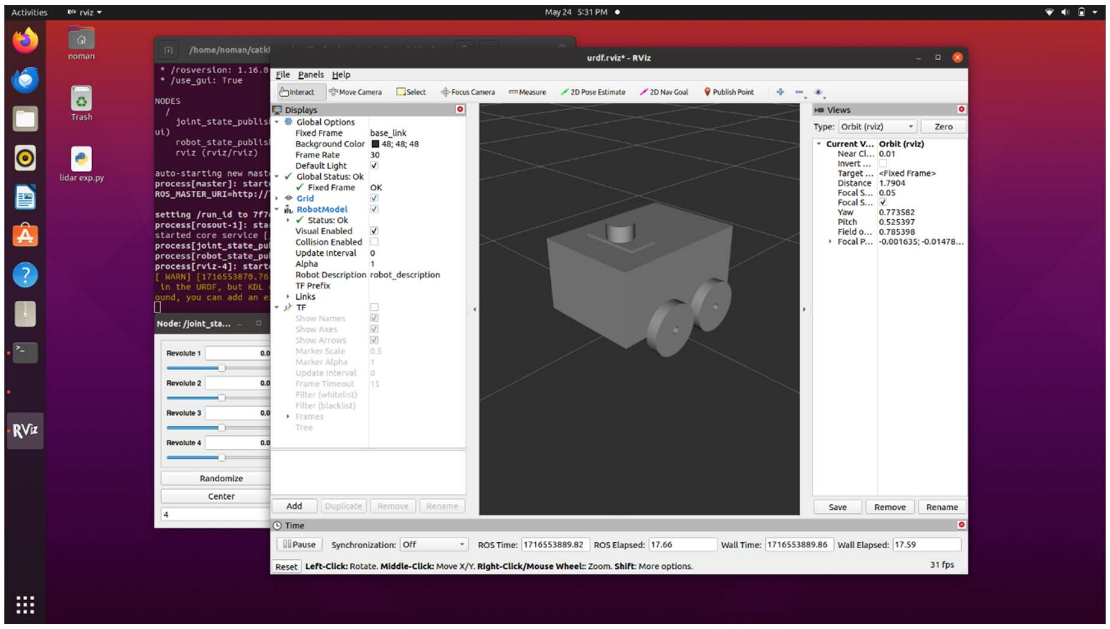


Figure 18 Top View of the Model showing the Lidar and wheels

After the design is made an important step is to name the chassis where the wheels are attached as the "base_link". Then we can export the design to URDF using the utilities section and scripts where we can run the fusion2urdf script. This creates the URDF files in a folder that can then be opened in Rviz and Gazebo.

3.6.2 RViz Model

The simulation of our robot was done in RViz by importing the URDF that we created using Fusion 360 and running the catkin make function in ROS. After this is done the model can be launched using the terminal and our model is visualized in Rviz. Here we can see if the model is correctly imported and is standing upright.



```
 * /rosversion: 1.16.0
 * /use_gui: True

NODES
  /
    joint_state_publis
ui)
    robot_state_publis
    rviz (rviz/rviz)

auto-starting new mast
process[master]: start
ROS_MASTER_URI=http://

setting /run_id to 7f7
process[rosout-1]: sta
started core service [
process[joint_state_pu
process[robot_state_pu
process[rviz-4]: start
[ WARN] [1716553870.76
 in the URDF, but KDL
ound, you can add an e
```

Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
✓ Status: Ok	
Visual Enabled	✓
Collision Enabled	☐
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
TF	☐
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	0.5
Marker Alpha	1
Update Interval	0
Frame Timeout	15
Filter (whitelist)	
Filter (blacklist)	
Frames	
Tree	

Current V...	Orbit (rviz)
Near Cl...	0.01
Invert ...	☐
Target ...	<Fixed Frame>
Distance	1.7904
Focal S...	0.05
Focal S...	✓
Yaw	0.773582
Pitch	0.525397
Field o...	0.785398
Focal P...	-0.001635; -0.01478...

Revolute 1 0.0 · Revolute 2 0.0 · Revolute 3 0.0 · Revolute 4 0.0

Synchronization: Off · ROS Time: 1716553889.82 · ROS Elapsed: 17.66 · Wall Time: 1716553889.86 · Wall Elapsed: 17.59

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options. 31 fps

Figure 19 Rviz Model exported from Fusion 360

3.6.3 Gazebo Model

After checking the correctness of our model in Rviz we can launch the Gazebo file. Gazebo allows the simulation of robot, including its physical characteristics, dynamics and sensors. By modeling the robot and controlling its movement in an environment it allows us to test the control system.

We can launch the Gazebo simulation similar to the way we launched the Rviz simulation from the terminal as our URDF files contain the Gazebo launch file also in the directory.

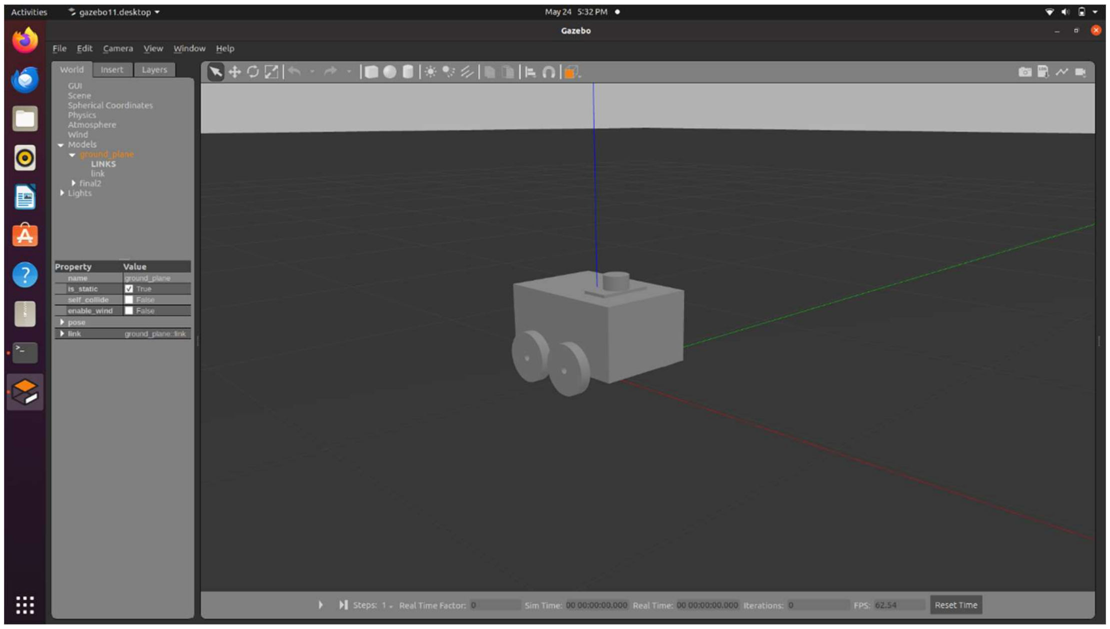
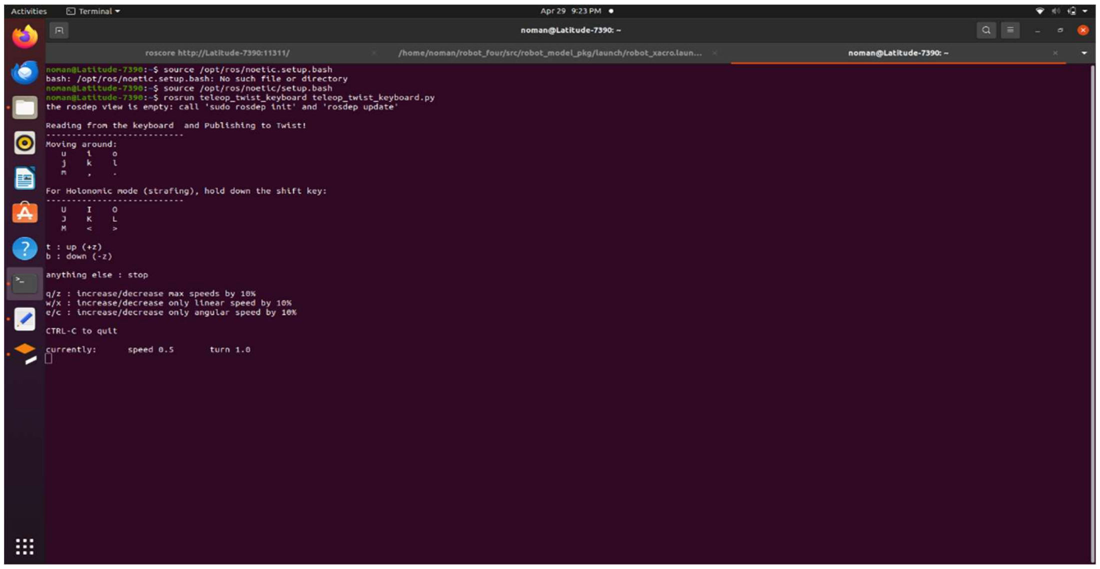


Figure 20 Our Robot Model opened in Gazebo

Next we install Teleop Twist Keyboard [32] which is a teleoperation node that converts keyboard commands to twist messages and allows us to control the movement of our robot using keyboard keys. It provides a convenient interface for controlling the movement of the robot by sending velocity commands based on user input from the keyboard. It is valuable for testing and validating the functionality of the control system.



```
noman@Latitude-7390:~$ source /opt/ros/noetic.setup.bash
bash: /opt/ros/noetic.setup.bash: No such file or directory
noman@Latitude-7390:~$ source /opt/ros/noetic/setup.bash
noman@Latitude-7390:~$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'

Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

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

CTRL-C to quit

currently:      speed 0.5       turn 1.0
```

Figure 21 The Telop Twist Keyboard node

We install teleop_twist_keyboard using the terminal and create a Teleop Launch File in the launch directory. In Gazebo plugins there is a plugin named skid steer drive controller. It is a piece of code that we have to add in the Gazebo file of our robot and edit it with the wheel links of our robot. After completing the steps, we launch Gazebo and then the teleop twist keyboard. Keeping the node open on screen and pressing the keyboard keys we can control our robot in gazebo.

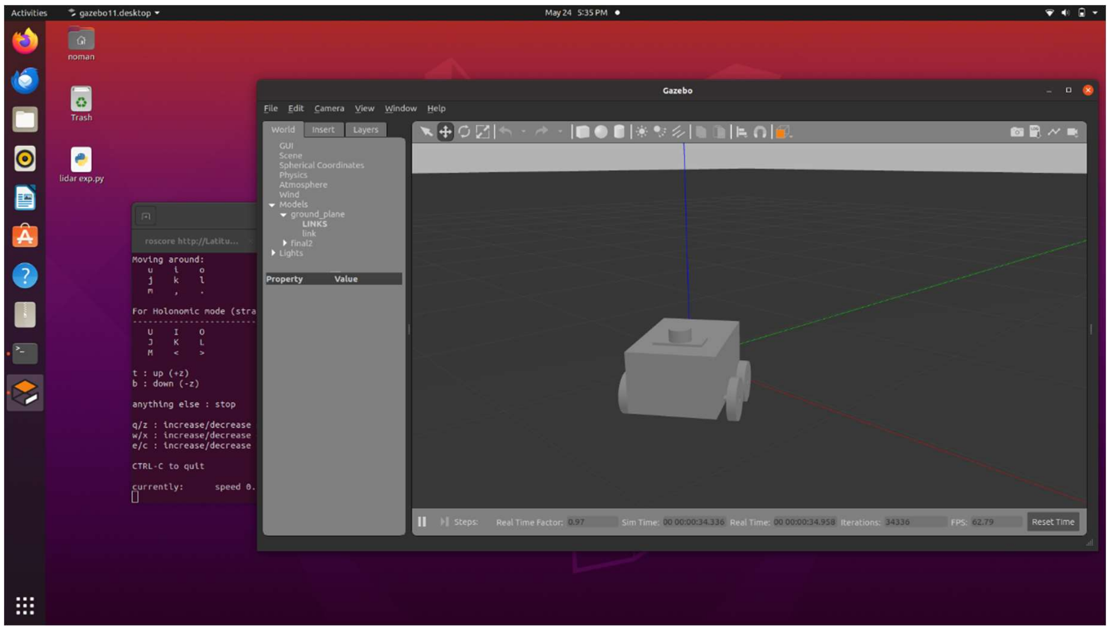

```
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (stra
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease
w/x : increase/decrease
e/c : increase/decrease

CTRL-C to quit

currently:      speed 0.
```

Figure 22 The Twist Keyboard node running alongside Gazebo

3.7 Mapping

After simulation we have to work on the live working of the robot. First of all we have to generate a map using Lidar for the environment where we are placing the robot in. For this first we install the necessary ROS packages for SLAM and also write its launch file. The launch file contains the robot state publisher node, laser scan publisher node, the gmapping node and the teleoperation node. Then after building the workspace, we can launch everything together.

Gazebo is launched where our robot is displayed along with a map of the environment. As we move the robot around using keyboard keys the map is generated and put together. Once we are satisfied with the map we can save it using the map_saver utility from the map_server package. We install Map server using "sudo apt-get install ros-noetic-map-server"

To save the Map we open a new terminal, source the workspace, and save the map using the command “rosrun map_server map_saver -f ~/my_map”. This will save the map to the created files my_map.yaml and my_map.pgm.

3.8 Path Planning

To send the robot to go to a goal position on the map we are using the ROS navigation stack. The ROS navigation stack [33] is a powerful set of software components that enables robots to autonomously navigate in an environment. It provides a framework for robots to perform tasks such as localization, mapping, path planning and obstacle avoidance. It uses the Adaptive Monte Carlo Localization (AMCL) which is a probabilistic localization system for a robot moving in 2D. It uses a particle filter to track the pose of the robot against a known map. AMCL takes in laser scans and odometry data to estimate the robot's position and orientation.

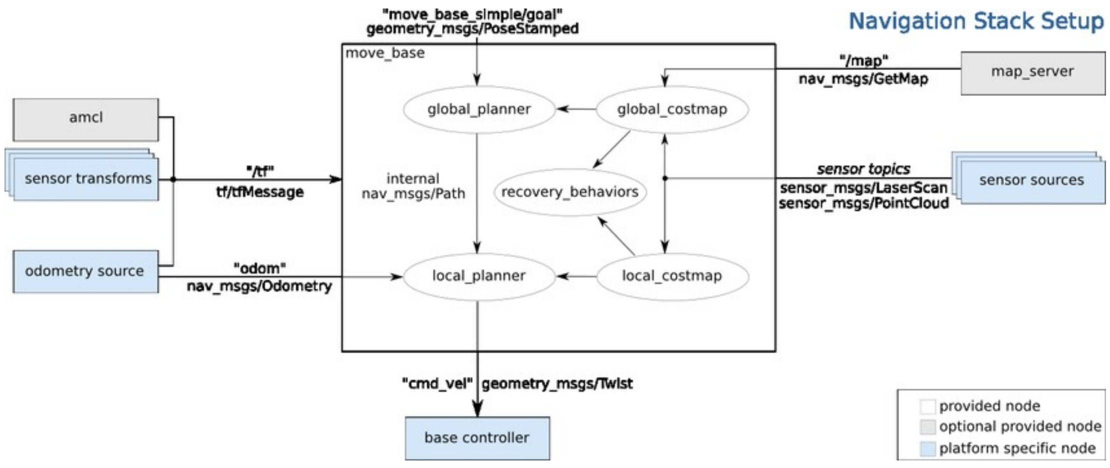


Figure 23 The Ros Navigation Stack Setup *[34]*

The ROS Navigation Stack is composed of several nodes that work together to enable a robot to navigate autonomously in an environment. These nodes handle localization, mapping, path planning, obstacle avoidance and control. Here is an explanation of the key nodes involved in the navigation stack:

- AMCL (Adaptive Monte Carlo Localization)

AMCL is responsible for localizing the robot within a known map. It uses a particle filter to estimate the robot's position and orientation (pose) based on sensor data and a pre-existing map. It takes input from the laser scans, odometry data and a static map and outputs the estimated pose of the robot in the map frame.

- map_server

It provides a map of the environment to other nodes. It loads a static map from a file and publishes it as a ROS topic.

- move_base

Central node of the navigation stack that combines global and local path planning to move the robot to a goal position. It uses global and local planners to generate and follow paths, handles recovery behaviors, and sends velocity commands to the base controller.

- Global Planner

Creates a high-level path from the robot's current position to the goal. Uses algorithms like A* or Dijkstra to find an optimal path on the global costmap.

- Local Planner

Generates a feasible path for the robot to follow in real-time. Considers the robot's kinematics and dynamic constraints, avoids obstacles detected by sensors.

- Costmaps

Global Costmap: Represents the entire environment and is used by the global planner. Static map, sensor data for static obstacles. Costmap for global planning.
Local Costmap: Represents the immediate surroundings of the robot and is used by the local planner. Sensor data for dynamic obstacles. Costmap for local planning.

- Sensor Data Integration Nodes

Laser Scan: Provides distance measurements to obstacles. Provides 3D data of the environment.
Point Cloud: Provides 3D data of the environment. Output processed point cloud data.

- Odometry

Provides the robot's movement information. Tracks the robot's position and velocity over time using wheel encoders and/or inertial measurement units (IMUs).

- Base Controller

Executes the velocity commands generated by the local planner. Converts high-level velocity commands into low-level motor commands.

To use the navigation stack first we have to install it along with its other packages by running the command in the terminal. Make sure your catkin workspace is set up and sourced. Create a launch file to start all the necessary nodes for navigation. Save this file as navigation.launch in the launch directory of your package. Create parameter files for the costmaps and move base configuration. Make them for the common costmaps, local costmaps, the global costmaps and the movebase. After configuring of all the parameters we can launch the navigation stack along with the robot in gazebo.

Then we can send goals to the robot using Rviz. Use the "2D Nav Goal" tool in RViz to set a goal position for the robot. Click on the map where you want the robot to go, and drag to set the orientation. The robot will start moving towards the goal position avoiding any obstacles in its path.

CHAPTER 4: RESULTS AND LIMITATIONS

4.1 Successful Integration and Navigation

The robot was able to localize itself accurately within the predefined map using the AMCL algorithm. The localization error was minimal, allowing the robot to maintain a reliable estimate of its position and orientation in the environment.

The integration of the global and local planners within the move_base node allowed the robot to plan and follow paths effectively. The global planner provided optimal routes from the start to the goal positions, while the local planner ensured real-time obstacle avoidance and path adjustments.

The local costmap, updated with real-time sensor data from the RP Lidar A1M8, enabled the robot to detect and avoid obstacles dynamically. The robot successfully navigated through environments with static and dynamic obstacles without collisions.

Using the 2D Nav Goal tool in RViz, the robot could autonomously navigate to specified goal positions. The robot demonstrated the ability to reach multiple goal points accurately, even in complex environments.

The teleop_twist_keyboard package allowed for manual control of the robot, providing a reliable means of testing and debugging. The robot responded promptly to teleoperation commands, showcasing the effectiveness of the communication between the Raspberry Pi 4 and the Arduino Mega 2560 for motor control.

4.2 Limitations

4.2.1 Sensor Limitations

The RP Lidar A1M8 has a limited range and resolution compared to more advanced lidar systems. This limitation affects the robot's ability to detect and avoid obstacles at greater distances, potentially reducing the effectiveness of the navigation stack in larger or more complex environments.

Relying solely on the RP Lidar A1M8 for obstacle detection and localization may not provide sufficient data in certain scenarios, such as highly dynamic environments or areas with poor lidar reflection surfaces.

4.2.2 Computational Constraints

While the Raspberry Pi 4 is a powerful single-board computer, it has computational limitations compared to more robust computing platforms. These limitations can impact the performance of the navigation stack, especially when processing large amounts of sensor data or running complex algorithms in real-time.

4.2.3 Mapping and Localization

The current implementation relies on a pre-generated static map for navigation. This approach limits the robot’s ability to adapt to dynamic changes in the environment or to navigate in completely unknown environments without prior mapping.

Also the performance of the AMCL algorithm can degrade in environments with insufficient distinct features or repetitive patterns, leading to localization inaccuracies. Additionally, sudden or extreme movements of the robot can cause a temporary loss of localization accuracy.

4.2.4 Path Planning and Control

The local planner may struggle in environments with narrow passageways or densely populated obstacles, leading to suboptimal path choices or oscillations. The tuning of planner parameters is critical to ensure smooth and reliable navigation.

The Arduino Mega 2560, while suitable for basic motor control, lacks the advanced features of dedicated motor controllers. This limitation can affect the precision and responsiveness of the robot's movements, particularly at higher speeds or with complex maneuvers.

CHAPTER 5: CONCLUSION AND FUTURE SCOPE

Our project successfully implemented a 4-wheel drive delivery robot utilizing the RP Lidar A1M8 and Raspberry Pi 4 within the ROS navigation stack. The integration of these components enabled the robot to autonomously navigate through a predefined environment, accurately localizing itself using the AMCL algorithm and effectively planning and executing paths to specified goals. The project demonstrated robust obstacle avoidance capabilities and reliable teleoperation using the teleop_twist_keyboard package, validating the effectiveness of the chosen hardware and software architecture.

Despite the successes, the project encountered several limitations, including sensor range and resolution constraints, computational limits of the Raspberry Pi 4, and reliance on a pre-generated static map for navigation. These factors occasionally impacted the robot's performance, particularly in complex or dynamic environments.

5.1 Scope of Future Work

5.1.1 Enhanced Sensor Integration

We can integrate more advanced sensors such as RGB-D cameras, higher-resolution lidar, or ultrasonic sensors to provide a richer data set for better obstacle detection and environmental awareness. Fusion of multiple sensor data can be done to improve localization accuracy and robustness in diverse environments.

Implementation of sensor fusion techniques to combine data from different sensors can also enhance the robot's ability to detect and navigate around obstacles more reliably.

5.1.2 Improved Computational Hardware

Upgrade to a more powerful computing platform, such as an NVIDIA Jetson or an Intel NUC, to handle more complex algorithms and larger data sets, thereby improving real-time processing capabilities and overall system performance.

The Arduino Mega 2560 can be replaced with more advanced motor controllers that offer better precision and responsiveness, enhancing the robot's movement control, especially in complex navigation scenarios.

5.1.3 Advanced Mapping and Localization

Implementation of more advanced SLAM algorithms such as Google Cartographer or RTAB-Map can be done to enable the robot to create and update maps in real-time, allowing it to navigate unknown or changing environments without prior mapping.

Algorithms can be developed that allow the robot to adapt to dynamic changes in the environment, ensuring consistent localization and navigation even as obstacles and environmental conditions change.

5.1.4 Enhanced Path Planning and Control

Advanced global and local path planning algorithms can be incorportated to improve the efficiency and safety of the robot's navigation in complex environments. Local planner parameters can be more fine-tuned and could incorporate predictive control methods to minimize path oscillations and ensure smoother navigation through tight spaces and around dynamic obstacles. Higher-level decision-making algorithms can be developed to enable the robot to autonomously choose optimal paths, navigate through multi-room environments, and handle complex delivery tasks with minimal human intervention.

5.1.5 Software Optimization and Features

Consider migrating to ROS2 for improved performance, better real-time capabilities and enhanced security features, which are beneficial for complex robotic applications.

Conduct extensive testing in varied environments and scenarios to ensure the robustness and reliability of the robot's navigation system, addressing edge cases and optimizing performance under different conditions.

REFERENCES

[1] A. O. a. L. P. Fernando Sancho, "Design and Development of a Low-cost Autonomous Delivery Robot for Indoor Enviroments," 2018.

[2] "TurtleBot," 2010. [Online]. Available: https://www.turtlebot.com/.

[3] "Starship Technologies," [Online]. Available: https://www.starship.xyz/.

[4] "Amazon robotic fulfillment center," [Online]. Available: https://www.waredock.com/magazine/what-is-amazon-robotic-fulfillment-center/.

[5] M. B. W. B. A. B. C. S. Thrun, "MINERVA: A Second-Generation Mobile Tour-Guide Robot," 2004.

[6] Z. Q. W. W. H. &. Z. X. Deng, "A Survey of Simultaneous Localization and Mapping with Robotic Sensors," 2021.

[7] E. L. F. M. A. M. Vallecorsa, "The FPGA-based data acquisition system for the upgrade of the CMS hadron calorimeter.," 2018.

[8] RoboPeak, "RP Lidar A1M8 Datasheet," [Online]. Available: https://www.robotshop.com/media/files/pdf/rplidar-a1m8-lidar-sensor.pdf.

[9] "Raspberry Pi 4," [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[10] "About Ubuntu," [Online]. Available: https://ubuntu.com/about.

[11] "Ubuntu for raspberry pi," [Online]. Available: https://ubuntu.com/raspberry-pi.

[12] "About ROS," [Online]. Available: https://www.ros.org/.

[13] Wikipedia, "Robot Operating System," [Online]. Available: https://en.wikipedia.org/wiki/Robot_Operating_System.

[14] "ROS Noetic Ninjemys," [Online]. Available: https://wiki.ros.org/noetic.

[15] "What is Lidar," [Online]. Available: https://www.synopsys.com/glossary/what-is-lidar.html.

[16] G. Pandey, "An Information Theoretic Framework for Camera and Lidar Sensor Data Fusion," 2014.

[17] "Arduino Mega 2560," [Online]. Available: https://store.arduino.cc/products/arduino-mega-2560-rev3.

[18] "Getting started with gazebo and ros noetic," [Online]. Available: https://automaticaddison.com/getting-started-with-gazebo-in-ros-noetic.

[19] "Build Robot using Robot Operating System (ROS 2) and Gazebo," [Online]. Available: https://bunchofcoders.github.io/basic_bocbot/.

[20] "RGBD intersects laser scan for simulation launch," [Online]. Available: https://github.com/ros-navigation/navigation2/issues/1521.

[21] "Introduction and use of Rviz," [Online]. Available: https://docs.elephantrobotics.com/docs/myarm-pi-300-en/12-ApplicationBaseROS/12.1-ROS1/12.1.4.

[22] "Sending Commands from rviz," [Online]. Available: https://ardupilot.org/dev/docs/ros-rviz.html.

[23] "Fusion 360 Overview," [Online]. Available: https://www.autodesk.com/products/fusion-360/overview.

[24] "What is URDF?," [Online]. Available: https://formant.io/urdf/.

[25] "Fusion to URDF Script," [Online]. Available: https://github.com/syuntoku14/fusion2urdf.

[26] "SLAM," [Online]. Available: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping.

[27] O. v. S. S. Kohlbrecher, "A flexible and scalable SLAM system with full 3D motion estimation," 2011.

[28] "Types of SLAM Algorithms," [Online]. Available: https://medium.com/@olesyakrindach/testing-different-slam-algorithms-with-turtlebot3-simulation-34c741db96ea.

[29] "SLAM Topics," [Online]. Available: https://github.com/topics/slam.

[30] P. B. A. L. R. V. Alessandro Gasparetto, "Path Planning and Trajectory Planning Algorithms: A General Overview," 2015.

[31] M. B. M. A. U. a. G. O. U. Zaharuddeen Haruna, "Path Planning Algorithms for Mobile Robots: A Survey," November 2023. [Online]. Available: https://www.intechopen.com/chapters/1154152.

[32] "Teleop Twist Keyboard," [Online]. Available: https://github.com/ros-teleop/teleop_twist_keyboard.

[33] "Configuration of the ROS navigation stack," [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup.

[34] K. A. M. K. Grzegorz Granosik, "USING ROBOT OPERATING SYSTEM FOR AUTONOMOUS CONTROL OF ROBOTS," 2016.

[35] H. L. S. T. C. Y. X. &. Y. J. Zhang, A Real-Time Obstacle Avoidance Algorithm for Mobile Robots Based on Improved Rapidly-Exploring Random Tree (RRT) Method., 2018.

[36] T. T. M. S. A. a. S. T. M. M. Khan, "Autonomous Delivery Robots: A Comprehensive Review," 2020.