iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** V **Month of publication:** May 2025

DOI: https://doi.org/10.22214/ijraset.2025.71204

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

Design and Development of Autonomous Mobile Robot for Material Handling Application

Mr. S. Thirumalairajan[1], M. Dhyanesh[2], V. Aswaddhaman[3], M. Jaya Venkata Pudeen[4], M. Yaswanth[5]
Diploma in Mechatronics, State Board of Technical Education Government of Tamilnadu

Abstract: ***In the evolving domain of logistics and supply chain management, efficiency, accuracy, and scalability have become essential to meet growing demands. This project introduces a cutting-edge autonomous mobile robot designed for material handling and inventory management in warehouses. Utilizing advanced technologies, the robot integrates Radio Frequency Identification (RFID) for real-time product identification and tracking, RPLIDAR for navigation and mapping using Simultaneous Localization and Mapping (SLAM), and a Raspberry Pi 4 as the central processing unit. With the Robot Operating System (ROS) serving as the control framework, the solution aims to minimize human intervention and automate critical tasks, thereby addressing key inefficiencies in traditional warehouse operations.***

The robot leverages its RPLIDAR sensor to generate precise 2D maps of the environment, enabling efficient navigation and dynamic obstacle avoidance. RFID-based tagging of items ensures accurate inventory tracking and retrieval, eliminating manual errors and improving overall reliability. As the robot moves autonomously, it scans the environment, identifies specific items, and updates inventory status in real time. Tasks like stocktaking, replenishment, and material transport are seamlessly executed, facilitated by efficient data processing and decision-making via Raspberry Pi 4. This integration provides continuous, uninterrupted operations with minimal manual oversight.

By automating labour-intensive and error-prone tasks, the proposed robot enhances warehouse efficiency while significantly reducing operational costs and the risk of workplace injuries. Its scalable design allows deployment in warehouses of varying sizes and complexities, with potential for collaborative operation by multiple robots. The system's ability to operate continuously and integrate with existing warehouse systems makes it a cost-effective and forward-thinking solution for modern inventory management. This project represents a robust step toward advancing autonomous technologies to transform conventional warehousing practices and meet the growing demand for smarter supply chain solutions.

I. INTRODUCTION

The advent of Industry 4.0 has ushered in a new era of automation and efficiency in material handling processes. Autonomous Mobile Robots (AMRs) are at the forefront of this transformation, offering unparalleled flexibility, precision, and adaptability in various industries, including manufacturing, warehousing, and logistics. This project focuses on the design and development of an Autonomous Mobile Robot (AMR) aimed at optimizing material handling tasks, reducing manual labour, and enhancing operational efficiency.

The proposed AMR integrates advanced technologies such as Robot Operating System

5.3.1. (ROS 2 Humble), a modular software platform for robotic applications, and Google Cartographer, a powerful 2D mapping and localization tool. It employs an RPLidar sensor for real-time environment scanning and mapping, ensuring robust navigation in dynamic environments. Additionally, the Raspberry Pi serves as the primary computational unit, delivering high performance at an affordable cost.

This project is guided by the principles of precision engineering and smart automation, combining elements of mechanical, electrical, and software design to achieve seamless navigation, efficient material transportation, and adaptability to diverse industrial needs. The system's autonomous capabilities eliminate human dependency, streamline workflows, and reduce potential errors in material handling operations.

The primary objective is to create a cost-effective, scalable solution capable of addressing the growing demands for automation in material handling. The project also emphasizes the importance of environmental adaptability, ease of deployment, and integration with existing systems, aiming to bridge the gap between theoretical innovation and practical application in real-world settings.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

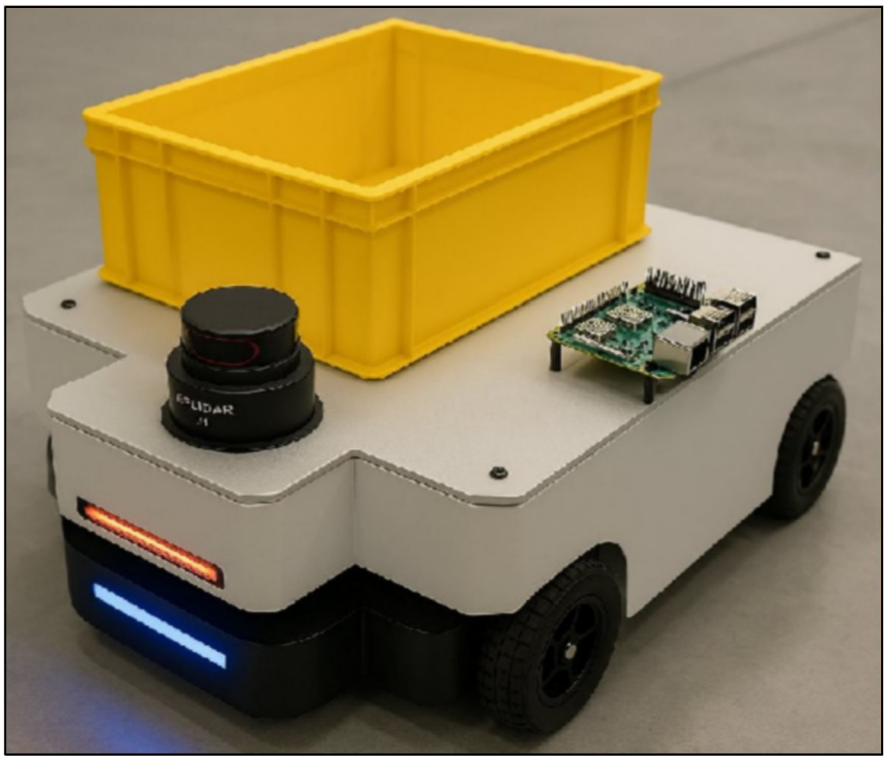


Fig 1.1

A. *Existing System*

Material handling is an integral part of industrial operations, encompassing the movement, protection, storage, and control of materials throughout manufacturing and distribution processes. The systems currently employed in industries to handle these tasks include manual labour, conveyor systems, and Automated Guided Vehicles (AGVs). While these approaches have served their purpose over the years, they are increasingly being identified as bottlenecks in the drive towards Industry 4.0 and smart automation.

1) Manual Handling Systems: Manual handling remains the most common approach for material movement in many industries, especially in small and medium-scale operations.
2) Conveyor Systems: Conveyor systems are widely used for repetitive tasks involving high-volume material flow. These systems are fixed installations that move materials along predetermined paths, offering improved efficiency for specific applications.
3) Automated Guided Vehicles (AGVs): AGVs represent a step toward automation by reducing human involvement in material handling. These vehicles typically rely on pre-defined paths marked by physical guides, such as magnetic tapes or wires.

B. *Objectives*

The primary objective of this project is to design and develop an Autonomous Mobile Robot (AMR) optimized for material handling applications in industrial settings. This entails creating a solution that integrates advanced robotics, real-time mapping, and autonomous navigation to overcome the limitations of existing systems.

The following specific objectives guide the project:

1) Develop an Autonomous Navigation System: Implement a reliable and efficient navigation algorithm using Robot Operating System 2 (ROS 2 Humble) to enable autonomous path planning and obstacle avoidance.
2) Integrate Real-Time Mapping and Localization: Employ Google Cartographer and RPLidar to achieve accurate 2D mapping and localization, enabling the robot to operate in dynamic environments without reliance on pre-defined paths.
3) Enhance Material Handling Efficiency: Design a modular system for material pickup and delivery that reduces human intervention, optimizes workflow, and minimizes handling errors.
4) Ensure Adaptability and Scalability: Create a flexible platform that can adapt to varying industrial layouts, operational requirements, and load capacities, ensuring ease of deployment and scalability.
5) Optimize Computational Resources: Leverage the Raspberry Pi as the primary control unit, ensuring cost-effectiveness while maintaining the processing capability needed for real-time operations.

6) Promote Safety and Reliability: Develop robust safety features, such as collision avoidance and emergency stop mechanisms, to ensure reliable operation in industrial settings.

7) Provide an Affordable Automation Solution: Deliver a cost-effective alternative to traditional AGVs and manual systems, making advanced automation accessible to small and medium enterprises (SMEs).

By achieving these objectives, the project aims to bridge the gap between technological innovation and practical implementation, addressing current challenges in material handling and paving the way for more intelligent and efficient industrial automation.

C. Block Diagram

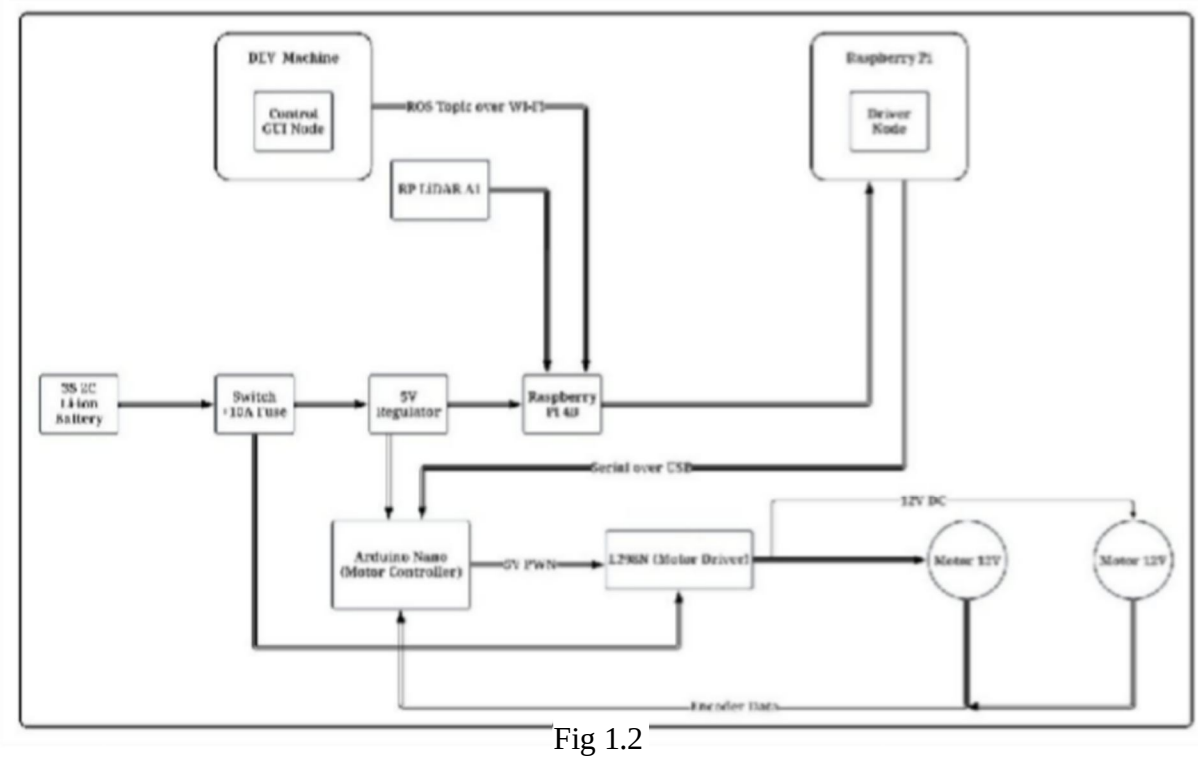


Fig 1.2

D. Working Principle

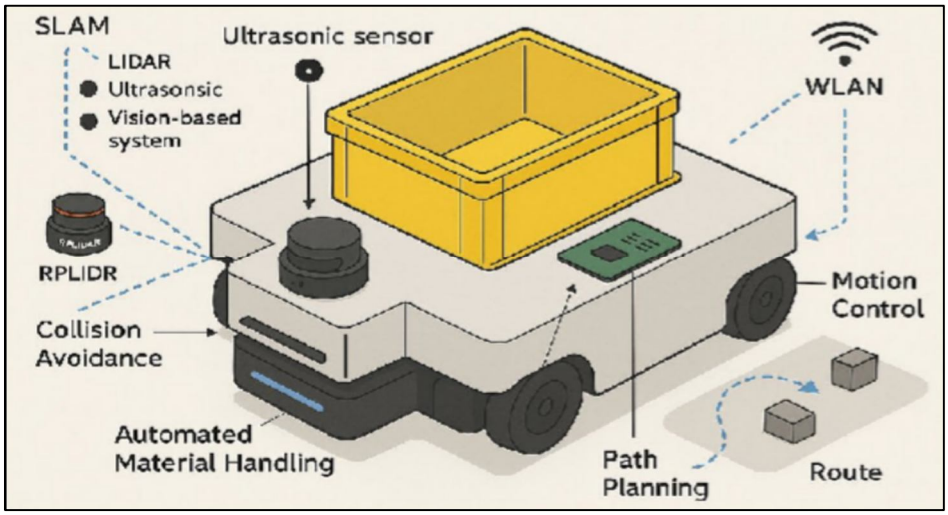
The working process begins with the SLAM technique, which enables the AMR to create and update a dynamic map of its surroundings in real-time. Using LIDAR, ultrasonic sensors, and vision-based systems, the robot continuously scans its environment to detect obstacles, navigate accurately, and adapt to changes in the workspace. These sensors provide crucial data for localization, helping the robot understand its position relative to the mapped environment. Additionally, sensor fusion technology integrates multiple sensor inputs to enhance navigation accuracy and ensure the AMR can operate effectively in complex and dynamic environments.

Once the environment is mapped, the path planning algorithm determines the most optimal route for material transportation. Unlike traditional Automated Guided Vehicles (AGVs) that follow predefined paths using magnetic strips or QR codes, the AMR dynamically plans and adjusts its path based on real-time sensor data. This adaptability allows it to operate efficiently in warehouses or factories where layouts frequently change. The AMR's decision- making system, powered by an onboard microcontroller or embedded AI processor, continuously processes sensor data and executes navigation commands to ensure smooth and efficient movement.

The motion control system regulates the movement of the AMR by controlling its motors, ensuring precise acceleration, deceleration, and turning capabilities. Integrated collision avoidance mechanisms allow the AMR to detect obstacles in its path and take corrective actions, such as stopping, rerouting, or slowing down, to prevent accidents. These mechanisms rely on real-time input from sensors and are essential for ensuring the safety of both the AMR and human workers operating in the same environment.

For material handling operations, the AMR is equipped with automated loading and unloading systems**,** which may include robotic arms, conveyors, or lift mechanisms depending on the specific application. These systems enable the robot to autonomously pick up, transport, and place materials at designated locations with high precision. Communication with a centralized control system or Warehouse Management System (WMS) is facilitated through wireless connectivity, allowing the AMR to receive task assignments, share status updates, and coordinate with other robots or equipment in a smart factory or warehouse setting.

To maintain continuous operation, the AMR also features an automated battery management system**,** which monitors battery levels and directs the robot to a charging station when necessary. Advanced AMRs incorporate AI-driven decision-making capabilities, allowing them to handle unexpected scenarios, such as rerouting due to blocked paths or prioritizing urgent tasks based on real-time demands. In conclusion, the working principle of the Autonomous Mobile Robot for material handling is based on a seamless integration of SLAM-based navigation, sensor-driven decision-making, intelligent path planning, motion control, collision avoidance, and automated material transport mechanisms. This approach significantly improves efficiency, reduces human labour, enhances workplace safety, and optimizes logistics operations in industrial and warehouse environments.



II. LITERATURE SURVEY

1) Paper 1

Title: Design and Implementation of an Autonomous Mobile Robot for Material Handling

Author: R. Shanmugasundaram & P. Rajalakshmi

Year: 2016

The development of an Autonomous Mobile Robot (AMR) for efficient material transportation in industrial and warehouse environments aims to reduce human labour, optimize logistics, and enhance workplace safety through advanced navigation and obstacle avoidance mechanisms. Traditional material handling systems, including manual labour and Automated Guided Vehicles (AGVs), often suffer from inefficiencies due to fixed path operations and limited adaptability in dynamic environments. To overcome these challenges, the proposed AMR integrates LIDAR and ultrasonic sensors for real-time collision detection and obstacle avoidance, ensuring seamless navigation even in unpredictable conditions. Unlike AGVs that rely on pre-set paths marked by magnetic strips or QR codes, the AMR utilizes Simultaneous Localization and Mapping (SLAM) techniques to dynamically map its environment and adapt its route based on real-time sensor data, making it ideal for constantly changing warehouse layouts. Sensor fusion techniques further enhance localization accuracy, combining multiple sensor inputs to create a robust perception system. The AMR's decision- making system is powered by an onboard microcontroller that processes sensor data and executes real-time navigation commands. Real-world testing in a warehouse environment demonstrated the AMR’s ability to navigate complex paths, avoid obstacles, and efficiently transport materials with minimal human intervention. The results indicated that automation using AMRs significantly reduces material handling time and operational costs, making them a viable alternative to traditional forklifts and AGVs while also improving workplace safety by preventing accidents caused by human error.

The findings suggest that AI-driven mobile robots can revolutionize logistics and warehouse automation by offering higher efficiency, flexibility, and scalability. The study concludes by emphasizing the need for further advancements in AI-driven path planning, sensor integration, and communication systems to enhance the overall performance and reliability of AMRs in large-scale industrial applications.

2) Paper 2

Title: The Role of Internet of Things (IoT) in Autonomous Mobile Robot Systems
Author: M. M. Hassan & A. V. Vasilenko
Year: 2020

IoT integration has significantly enhanced the capabilities of Autonomous Mobile Robots (AMRs) by enabling real-time data collection, advanced communication, and intelligent decision-making. Through IoT, AMRs can collect and process data from various sensors, including LIDAR, GPS, and IMUs, allowing them to navigate dynamic environments with greater accuracy using Simultaneous Localization and Mapping (SLAM). IoT enhances communication and connectivity through Machine-to-Machine (M2M) and Vehicle-to- Everything (V2X) technologies, allowing AMRs to interact seamlessly with other robots, infrastructure, and centralized control systems. This connectivity enables remote monitoring and control, where AMRs can be tracked, managed, and diagnosed via cloud-based platforms, facilitating predictive maintenance to reduce downtime and extend operational life. Furthermore, IoT-driven AMRs leverage edge computing for intelligent decision-making, allowing local data processing for quicker responses and adaptability to changing environments, such as rerouting around obstacles and optimizing navigation paths. This enhances system efficiency by improving resource allocation, reducing energy consumption, and enabling task prioritization for faster and more effective material handling. However, IoT integration poses challenges related to data security and privacy, as interconnected systems are vulnerable to cyberattacks, requiring robust encryption and secure protocols to protect sensitive information. IoT also enhances scalability and system integration, allowing the seamless addition of new AMRs without disrupting existing workflows while supporting heterogeneous systems from various manufacturers. Industry-specific applications of IoT-driven AMRs include optimizing inventory tracking and movement in logistics and facilitating automated delivery of medical supplies in healthcare environments. Future developments in IoT-enabled AMRs include the adoption of 5G for faster and more reliable communication, AI-IoT fusion for advanced decision-making, and the implementation of swarm robotics, where multiple AMRs work collaboratively to enhance large-scale industrial automation. While IoT has already transformed the efficiency and functionality of AMRs, ongoing research and innovation in sensor integration, communication protocols, and autonomous navigation are crucial to overcoming current limitations and realizing the full potential of IoT in modern industrial and logistics environments.

Conclusion

The combined analysis of both studies highlights the transformative potential of Autonomous Mobile Robots (AMRs) in industrial and warehouse environments through advanced navigation systems and IoT integration. The first study emphasizes the role of AI-driven navigation and sensor fusion in enhancing AMR capabilities, reducing human intervention, and increasing operational efficiency. It demonstrates how AMRs, using SLAM techniques and real-time decision-making systems, offer superior adaptability to dynamic and unpredictable environments compared to traditional Automated Guided Vehicles (AGVs). The second study underscores the critical impact of IoT in extending AMR functionalities by enabling real-time data collection, seamless communication, and intelligent decision-making through advanced technologies such as M2M and V2X. IoT also facilitates remote monitoring and predictive maintenance, improving system longevity and operational efficiency. Together, these studies suggest that the future of AMRs lies in the continued integration of cutting-edge AI and IoT technologies. This convergence not only enhances navigation accuracy, obstacle avoidance, and energy efficiency but also supports scalable and secure multi-robot collaboration. As industries move toward smart automation, the adoption of 5G communication, AI-IoT fusion, and swarm robotics will further elevate AMR performance, offering reliable, flexible, and cost-effective solutions for large-scale material handling and logistics operations.

III. DESIGN OF SYSTEM ELEMENTS

A. Introduction

The system design of the Autonomous Mobile Robot (AMR) focuses on building a modular and intelligent platform for efficient material handling. Key components include the RP LiDAR A1M8 for 360° mapping and obstacle detection, and a Raspberry Pi running ROS 2 for processing and navigation. An External USB Hub expands the Pi's connectivity for multiple sensors and peripherals. Motion is controlled using a 12V DC motor with encoder and L29 driver, offering precise speed and direction control via PWM. An Arduino Nano handles low-level tasks like sensor interfacing, supporting the Raspberry Pi in distributed control. Power is supplied by a 4400mAh battery, regulated by a buck converter, and maintained with a battery charger. A cooling fan ensures thermal stability, while connectors, switches, and wiring support reliable system integration. Together, these elements enable real-time, autonomous operation in dynamic environments.

B. List Of Components

The list of components are as follows:

S.No	Component Name	Quantity
1	RP LiDAR A1M8	1
2	Raspberry Pi	1
3	External USB Hub	1
4	Arduino Nano Board	1
5	12V DC motor with Encoder+L29 Driver	1
6	Battery (4400 mAh)	1
7	Battery Charger	1
8	Cooling Fan	1
9	Buck Convertor	1
10	Other(Connector, Switches, Wires & etc.,)	

Table 3.1

1) RP LiDAR A1M8

The RP LiDAR A1M8 is a high-precision laser scanner used for environmental mapping and navigation. It operates using laser triangulation and time-of-flight (ToF) principles to measure distances and generate 360-degree scans of the surrounding area. The LiDAR sensor is crucial for object detection, path planning, and obstacle avoidance, making it an essential component in robotic systems.

Key features:

- 360-degree scanning range for full environmental awareness.
- Up to 12-meter range for accurate distance measurement.
- Low power consumption, suitable for embedded applications.
- Supports real-time data acquisition, ensuring efficient navigation.

Working Principle:

LiDAR (Light Detection and Ranging) works by emitting laser pulses that reflect off surfaces and return to the sensor. By calculating the time taken for the light to return, the system determines the distance to an object. The RP LiDAR A1M8 uses a rotating mechanism to provide a full 360-degree scan of the environment. The data obtained from the LiDAR sensor is processed by the Raspberry Pi to create a detailed map of the surroundings, which can be used for path planning and autonomous movement.

Applications:

- Used in autonomous robots for obstacle detection and navigation.
- Employed in mapping and surveying applications.
- Used in drones for altitude measurements and terrain mapping.
- Integrated into self-driving vehicles for real-time road environment analysis.

Fig 3.1

2) Raspberry Pi

The Raspberry Pi is a single-board computer that serves as the main processing unit of the system. It is responsible for handling data from the LiDAR sensor, controlling the motor driver, and ensuring smooth operation of all connected components.

Key features:

- Processing LiDAR data to generate environmental maps.
- Controlling motor speed and direction based on sensor inputs.
- Managing power distribution to ensure stable operation.
- Providing wireless communication capabilities for remote control and monitoring.

Working Principle:

The Raspberry Pi runs a program that reads data from the RP LiDAR A1M8 and interprets the received distance values. Based on the processed data, the Raspberry Pi sends appropriate signals to the motor driver, adjusting speed and direction for movement. Additionally, it can communicate with a remote computer or cloud-based system for real- time monitoring and control.

Applications:

- Used in robotics for autonomous control and automation.
- Integrated into IoT systems for data collection and remote access.
- Employed in industrial automation systems for monitoring processes.
- Used in AI-based applications such as object detection and recognition.

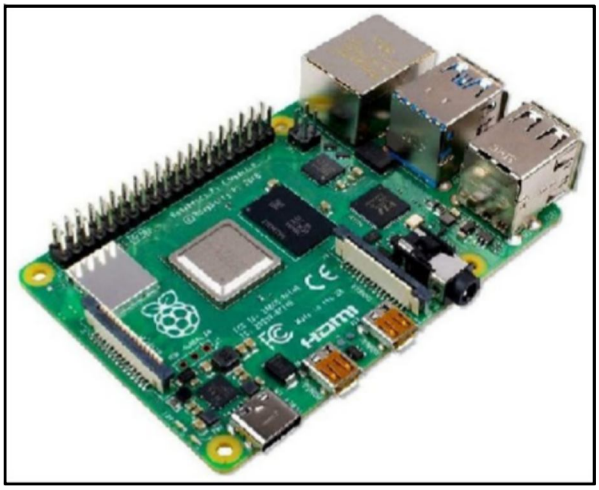


Fig 3.2

3) External USB Hub

An External USB Hub is a hardware device that expands a single USB port into multiple ports, allowing multiple USB-compatible devices to connect to a computer or controller simultaneously. It serves as a central interface for peripheral connections, enabling efficient communication and data transfer between devices. USB hubs can be either powered or unpowered, depending on the power needs of connected devices.

Key features:

- Multiple USB ports for simultaneous connection of various peripherals.
- Powered hub options available for stable operation of high-power devices.
- Plug-and-play compatibility with Raspberry Pi and other single-board computers.
- High-speed data transfer (USB 2.0/3.0 depending on the hub used).

Working Principle:

An External USB Hub expands the number of available USB ports on the Raspberry Pi. It acts as a signal repeater and distributor, allowing devices like the RP LiDAR, mouse, keyboard, or Wi-Fi dongle to be connected simultaneously. In powered variants, the hub also provides independent power to connected devices, reducing the load on the Raspberry Pi's onboard power.

Applications:

- Peripheral expansion for single-board computers.
- Data interface for multiple USB-based sensors or input/output devices.
- Lab and testing setups where multiple USB connections are required.
- Robotic systems that rely on multiple USB-based modules (e.g., LiDAR, cameras, USB-to-Serial adapters).

Fig 3.3

4) Arduino Nano Board

The Arduino Nano is a compact, breadboard-friendly microcontroller board based on the ATmega328P microcontroller. It offers the same functionality as the Arduino Uno but in a smaller form factor, making it ideal for space-constrained embedded applications. The board includes digital and analog input/output pins, supports serial communication, and can be programmed via USB using the Arduino IDE. It is widely used in robotics, automation, and sensor-based systems for performing control and processing tasks.

Key Features:

- Compact form factor, ideal for embedding in tight spaces.
- ATmega328P microcontroller, same as Arduino Uno but in a smaller layout.
- 14 digital I/O pins, 8 analog input pins, PWM support.
- USB interface for programming and serial communication.

Working Principle:

The Arduino Nano serves as a secondary controller for offloading simple sensor or actuator tasks from the Raspberry Pi. It operates using its own firmware (written in the Arduino IDE) and handles real-time control of components such as ultrasonic sensors, LEDs, or servos. Communication between the Nano and the Raspberry Pi can be established via USB, UART (Serial), or I2C.

Applications:

- Sensor interfacing where low-latency control is required.
- Distributed control in robotics systems for modular task management.
- Prototyping and embedded systems with size constraints.
- I/O extension for projects needing more digital or analog pins than a Raspberry Pi offers.

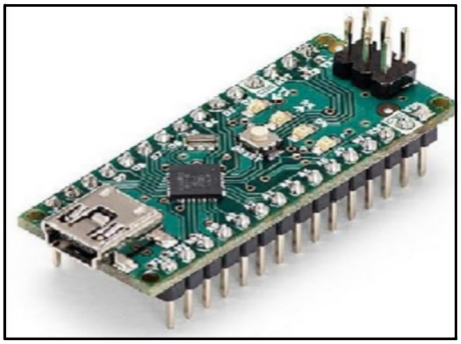


Fig 3.4

5) 12V DC Motor with Encoder + L29 Driver

To enable controlled movement, the project uses a 12V DC motor with an encoder, paired with an L29 motor driver. The encoder allows precise speed and position tracking, while the motor driver provides the necessary power and control signals to drive the motor efficiently.

Key features:

- High torque output, suitable for robotic applications.
- Encoder feedback for precise speed and position control.
- L29 driver for efficient motor control and direction switching.
- Supports PWM control, allowing variable speed adjustments.

Working Principle:

The motor encoder generates pulses based on the rotation of the motor shaft, allowing the Raspberry Pi to determine the motor's speed and position. The L29 driver acts as an interface between the Raspberry Pi and the motor, allowing for bidirectional control. The Raspberry Pi sends PWM (Pulse Width Modulation) signals to regulate the motor's speed.

Applications:

- Used in robotics for precise movement control.
- Employed in conveyor belt systems for speed regulation.
- Used in CNC machines for position accuracy.
- Integrated into electric vehicles for efficient power management.

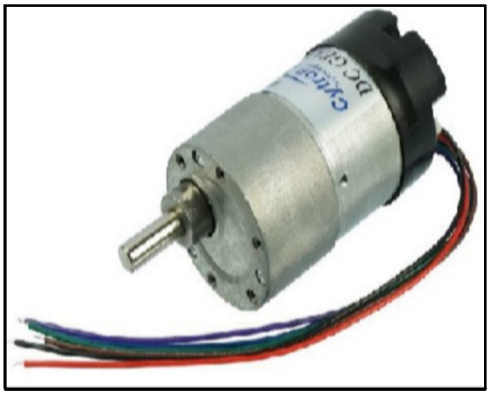


Fig 3.5

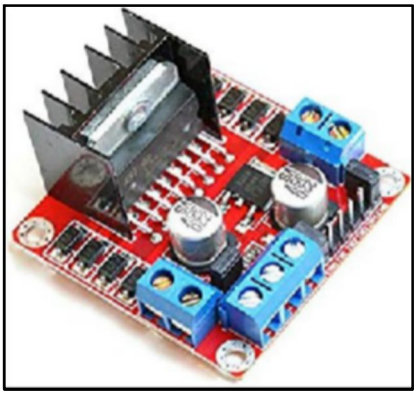


Fig 3.6

6) Battery(4400 mAh) & Battery charger

The system is powered by a 4400mAh rechargeable battery, providing the necessary energy for all components. A battery charger is used to ensure continuous power availability, preventing downtime during extended operation.

Key features:

- Long battery life, supporting extended usage.
- Rechargeable design, reducing maintenance costs.
- Stable voltage output, ensuring reliable performance.

Applications:

- Used in portable robotic systems.
- Employed in drones and UAVs for extended flight times.
- Integrated into solar-powered devices for energy storage.
- Used in backup power systems for critical applications.

Fig 3.7

7) Cooling Fan

Since prolonged operation may generate heat, a cooling fan is added to maintain optimal temperatures. Overheating can affect performance, especially for the Raspberry Pi and motor driver, making thermal management crucial for system longevity.

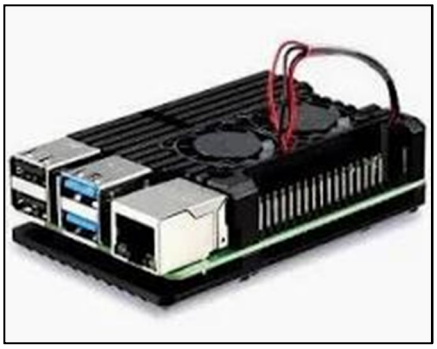


Fig 3.8

8) Buck Convertor

A buck converter is used to step down the voltage from the battery to the required levels for various components. This ensures that the Raspberry Pi, motor driver, and sensors receive stable and appropriate power.

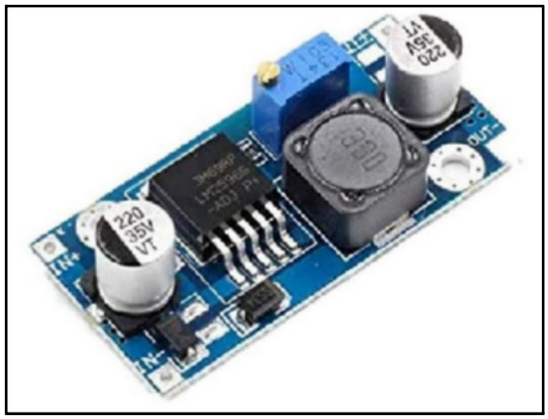


Fig 3.9

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

9) Other Components (Connectors, Switches, Wires, etc.)

Additional components like connectors, switches, and wires are essential for establishing reliable connections and enabling user control. These components ensure that power is distributed efficiently and signals are transmitted accurately between different modules.

IV. DEVELOPMENT OF ROBOT MODEL

A. *Introduction*

The robot simulation is carried out in ROS (Robot Operating System) using Gazebo, which is a powerful physics-based simulator. The robot model is described using URDF (Unified Robot Description Format), which defines its physical structure, joints, and sensors. This simulation helps test movement, obstacle avoidance, and mapping algorithms in a virtual environment before deploying them to the actual hardware.

B. *Robot Model*

This part of the code describes the robot dimensions and its physical property .The inertial values used are a type of codes are called as macro which clubs the data and can be used other files.

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
    <xacro:include filename="inertial_macros.xacro" />
    <xacro:property name="chassis_length" value="0.25" />
    <xacro:property name="chassis_width" value="0.25" />
    <xacro:property name="chassis_height" value="0.1" />
    <xacro:property name="chassis_mass" value="1.0" />
    <xacro:property name="wheel_radius" value="0.03" />
    <xacro:property name="wheel_thickness" value="0.03" />
    <xacro:property name="wheel_mass" value="0.05" />
    <xacro:property name="wheel_offset_x" value="0.200" />
    <xacro:property name="wheel_offset_y" value="0.155" />
    <xacro:property name="wheel_offset_z" value="0.01" />
    <xacro:property name="caster_wheel_radius" value="0.01" />
    <xacro:property name="caster_wheel_mass" value="0.01" />
    <xacro:property name="caster_wheel_offset_x" value="0.075" />
    <xacro:property name="caster_wheel_offset_z"
        value="${wheel_offset_z - wheel_radius + caster_wheel_radius}"></xacro:property>
```

These are used for the colours of the robot:

```
    <!-- Materials -->
    <material name="black">
                                        <color rgba="0.0 0.0 0.0 1.0" />
    </material>
    <material name="blue">
                                        <color rgba="0.0 0.0 0.8 1.0" />
    </material>
    <material name="green">
                                        <color rgba="0.0 1.0 0.0 1.0" />
    </material>
    <material name="grey">
                                        <color rgba="0.2 0.2 0.2 1.0" />
    </material>
    <material name="orange">
        <color rgba="${255/255} ${108/255} ${10/255} 1.0" />
    </material>
```


ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

This the base link, which is the first link and the base of the robot. Everything is based on the link:

```
<!-- Base link -->
<link name="base_link">
</link>
<!-- Base Footprint Link -->
<joint name="base_footprint_joint" type="fixed">
    <parent link="base_link" />
    <child link="base_footprint" />
    <origin xyz="0 0 0" rpy="0 0 0" />
</joint>
<link name="base_footprint">
</link>
```

This is the part of the main body:

```
<!-- Chassis link-->
<joint name="chassis_joint" type="fixed">
                                                    <parent link="base_link" />
    <child link="chassis" />
    <origin xyz="${-wheel_offset_x} 0 ${-wheel_offset_z}" />
</joint>

<link name="chassis">
    <visual>
        <origin xyz="${chassis_length/2} 0 ${chassis_height/2}" />
        <geometry>
            <box size="${chassis_length} ${chassis_width} ${chassis_height}" />
        </geometry>
        <material name="grey" />
    </visual>
    <collision>
        <origin xyz="${chassis_length/2} 0 ${chassis_height/2}" />
        <geometry>
            <sphere radius="${wheel_radius}" />
        </geometry>
    </collision>
    <xacro:inertial_box mass="0.5" x="${chassis_length}" y="${chassis_width}" z="${chassis_height}">
        <origin xyz="${chassis_length/2} 0 ${chassis_height/2}" rpy="0 0 0" />
    </xacro:inertial_box>
</link>
<gazebo reference="chassis">
    <material>Gazebo/Grey</material>
</gazebo>
```

This is the part of the two main wheels and one caster wheel. The main wheels have a continuous joint and caster wheel is fixed joint and it has friction set to zero to make it slide.

```
<!-- Left Wheel joint -->
<joint name="left_wheel_joint" type="continuous">
    <parent link="base_link" />
    <child link="left_wheel" />
    <origin xyz="0 ${wheel_offset_y} 0" rpy="-${pi/2} 0 0" />
    <axis xyz="0 0 1" />
```



```
</joint>
<link name="left_wheel">
    <visual>
        <geometry>
            <cylinder radius="${wheel_radius}" length="${wheel_thickness}" />
        </geometry>
        <material name="orange" />
    </visual>
    <collision>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <sphere radius="${wheel_radius}" />
        </geometry>
    </collision>
    <xacro:inertial_cylinder mass="${wheel_mass}" length="${wheel_thickness}" radius="${wheel_radius}">
        <origin xyz="0 0 0" rpy="0 0 0" />
    </xacro:inertial_cylinder>
</link>
<gazebo reference="left_wheel">
    <material>Gazebo/Orange</material>
</gazebo>

<!-- right Wheel joint -->
<joint name="right_wheel_joint" type="continuous">

    <parent link="base_link" />
    <child link="right_wheel" />
    <origin xyz="0 ${-wheel_offset_y} 0" rpy="${pi/2} 0 0" />
    <axis xyz="0 0 -1" />
</joint>
<link name="right_wheel">
    <visual>
        <geometry>
            <cylinder radius="${wheel_radius}" length="${wheel_thickness}" />
        </geometry>
        <material name="orange" />
    </visual>
    <collision>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <sphere radius="${wheel_radius}" />
        </geometry>
    </collision>
    <xacro:inertial_cylinder mass="${wheel_mass}" length="${wheel_thickness}" radius="${wheel_radius}">
        <origin xyz="0 0 0" rpy="0 0 0" />
    </xacro:inertial_cylinder>
</link>
<gazebo reference="right_wheel">
    <material>Gazebo/Orange</material>
</gazebo>
```

```
    <!-- Caster wheel -->
    <joint name="caster_wheel_joint" type="fixed">
        <parent link="chassis" />
        <child link="caster_wheel" />
        <origin xyz="${caster_wheel_offset_x} 0 ${caster_wheel_offset_z}" />
    </joint>
    <link name="caster_wheel">

        <visual>
            <geometry>
                <sphere radius="${caster_wheel_radius}" />
            </geometry>
            <material name="blue" />
        </visual>
        <collision>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <sphere radius="${caster_wheel_radius}" />
            </geometry>
        </collision>
        <xacro:inertial_sphere mass="${caster_wheel_mass}" radius="${caster_wheel_radius}">
            <origin xyz="0 0 0" rpy="0 0 0" />
        </xacro:inertial_sphere>
    </link>
    <gazebo reference="caster_wheel">
        <material>Gazebo/Blue</material>
        <mu1 value="0.001" />
        <mu2 value="0.001" />
    </gazebo>
</robot>
```

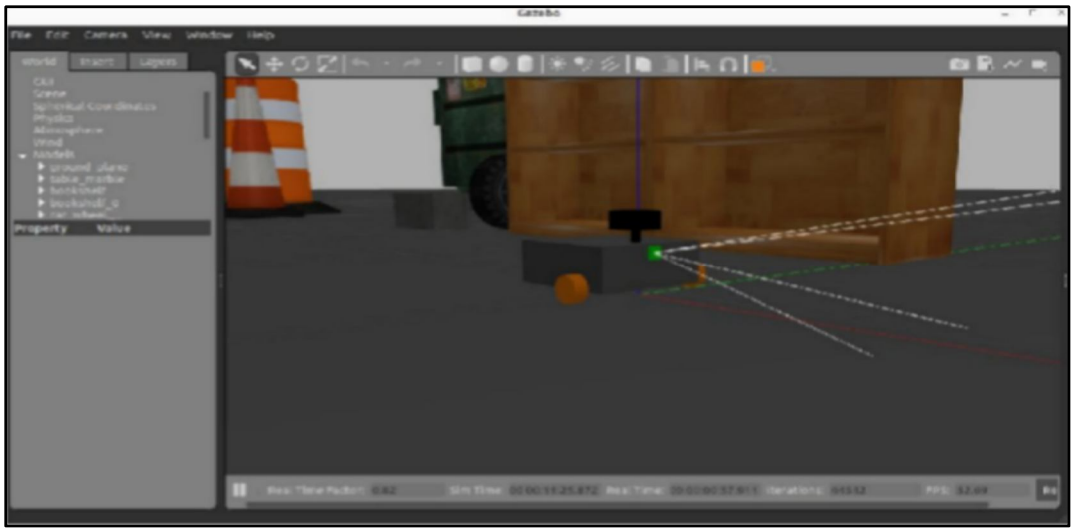


Fig 4.1

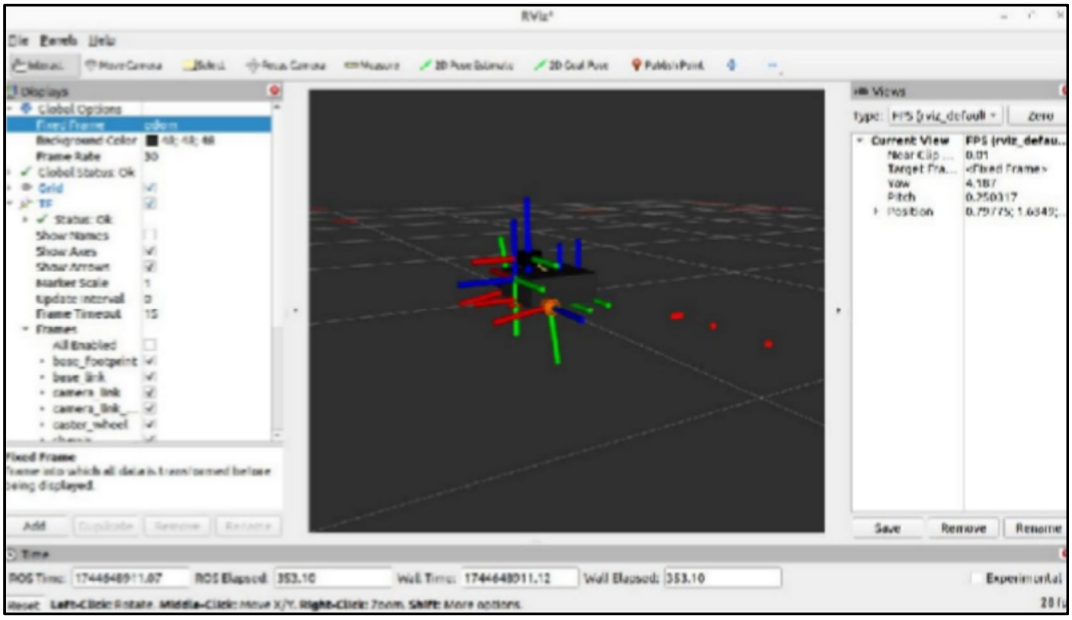


Fig 4.2

V. SYSTEM INTEGERATION

A. *Introduction*

System integration is a critical phase in the development of an autonomous robot, where different hardware and software components are combined to work as a single, cohesive system. In this project, the integration ensures seamless communication between the Raspberry Pi 4B, LiDAR sensor, motor driver, power system, and ROS 2 Humble software stack to enable real-time SLAM (Simultaneous Localization and Mapping) and autonomous navigation using Nav2**.**

B. *Hardware Integration*

Hardware integrity refers to the reliability, robustness, and proper functioning of all the physical components used in the robotic system. In this project, maintaining hardware integrity was essential to ensure consistent power supply, stable connections, and accurate data acquisition for successful SLAM and navigation.

1) *Power Supply*

The supply voltage of this robot to be appropriate for our motors. DC motors are pretty resilient, they can usually deal with a bit of over-voltage, and if we go a bit under, they'll just run slower. The motors are rated for 12V, which is sufficient for our robot. Depending on the application we may want to go for 6V for the pi and other sensors.

2) *Wiring & Connections*

All components are wired using 18 to 24 AWG wires, suitable for handling 12V 2A and 5V 2A power supplies. An XT60 connector is used between the battery and the circuit to ensure safe and reliable power delivery while preventing overload. A 10A mini blade fuse is placed near the battery to protect the electrical circuit from overcurrent by breaking the connection if the current exceeds 10 amps. Additionally, an SPDT (Single Pole Double Throw) toggle switch is connected next to the fuse to control the power flow and verify complete circuit wiring; it allows the system to power on only when the entire circuit is properly closed.

3) *Motor Connections*

The motor is given 12v supply . An Arduino nano is used as motor controller L293D motor driver is used to drive the current from the motor controller to control the motor direction with the signals by turning the motor on and off. Always check the motor data sheet for the encoder data and min and max motor data like the stall current for the appropriate power supply.

电机型号 Motor model	额定电压 Rated voltage	无负荷No-load		额定 Rated				堵转 Stall	
		转速 Speed	电流 Current	转速 Speed	力矩 Torque	电流 Current	功率 Power	力矩 Torque	电流 Current
	VDC	r/min	mA	r/min	g.cm	mA	W	g.cm	A
TFF-180SH-18140	6.0	7000	90	5900	20	350	1.21	127	1.80
TFF-180SH-10400	12.0	5000	20	4300	10	70	0.44	71	0.42

Fig 5.1

4) Power Regulator

A buck converter is used to step down 12V from the main battery to a stable 5V output. This 5V is supplied to the Raspberry Pi and other low-voltage peripherals. It ensures safe and efficient power distribution across components.

5) Raspberry Pi

Connect the 5v and GND pin to the supply of the regulator this gives the PI the overall power supply to power enough to peripherals ,but not enough to power the lidar. Therefore, we need an extra USB hub with external 5v power supply like the pi ush hub. Connect the USB hub in one of the USB ports and plug the lidar and the serial cable of the Arduino nano.

6) RP LiDAR

Connect the RP LiDAR communication to the module and the USB module. A lidar requires 5v of power supply which is inefficient to the PI's USB to provide which is why we need an external USB hub. But it can be done by wiring the lidar directly to the gpio pins.

7) Arduino Nano

The Arduino is used to provide the signal to the motor driver acting as a motor controller. The pins of the motor encoder and motor driver is wired to the Arduino. There are 2 pins A and B from motor encoder and four pins IN1,2,3,4 from the motor driver that needed to be connected. Every connection of the Arduino is listed below.

Arduino pins	Encoder pins	L293D
D10	-	IN3 RFWD
D09	-	IN2 LFWD
D06	-	IN4 RREV
D01	-	IN1 LREV
D02	Left A	-
D03	Left B	-
A05	Right A	-
A04	Right B	-

Table 5. 1

C. Software Integeration

Software integration is a vital aspect of the robotic system that ensures all ROS 2 packages, nodes, and external libraries work together harmoniously. In this project, the software integration focused on connecting the LiDAR sensor, SLAM (Simultaneous Localization and Mapping), and autonomous navigation system (Nav2) through the ROS 2 Humble middleware.

1) ROS2 Installation

ROS2 humble is ros version compatible Ubuntu Jammy (22.04) and we are installing deb packages. The pi and the developer machine use the same version of the ROS which is best for integration.

Firstly, we need to add the ROS 2 apt repository to our system so ensure that the Ubuntu
repository universe is enabled

sudo apt install software-properties-common
sudo add-apt-repository universe

Next add the ROS 2 GPG key with apt.

sudo apt update && sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg

Next add the repository to your sources list.

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros archive keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

After this we can start installing the ros2 humble packages therefore Update your apt repository caches after setting up the repositories. ROS 2 packages are built on frequently updated Ubuntu systems. It is always recommended that you ensure your system is up to date before installing new packages.

sudo apt update
sudo apt upgrade

For the developer machine install the desktop version of the humble for RViz, demos, tutorials. And for the Raspberry pi install the base version because there is no need for GUI tools only Communication libraries, message packages, command line tools.

sudo apt install ros-humble-desktop (for developer machine)
sudo apt install ros-humble-ros-base (for pi)

Finally, after installing we need to setup the environment for easy sourcing of the environment

source /opt/ros/humble/setup.bash

Fig 5.2

2) Git Hub Cloning

GitHub cloning is the process of creating a local copy of a remote repository using the git clone command. The simulation and the real robot share the same repository which helps in modifying the visualizing model to launch file to work properly. It allows developers to access, modify, and contribute to open-source or team projects stored on GitHub. This is essential for downloading ROS packages, simulation files, and codebases during development.

All launch commands launch files are in this project git hub repository. The repo link is given

https://github.com/Dhyan036/ware_bot

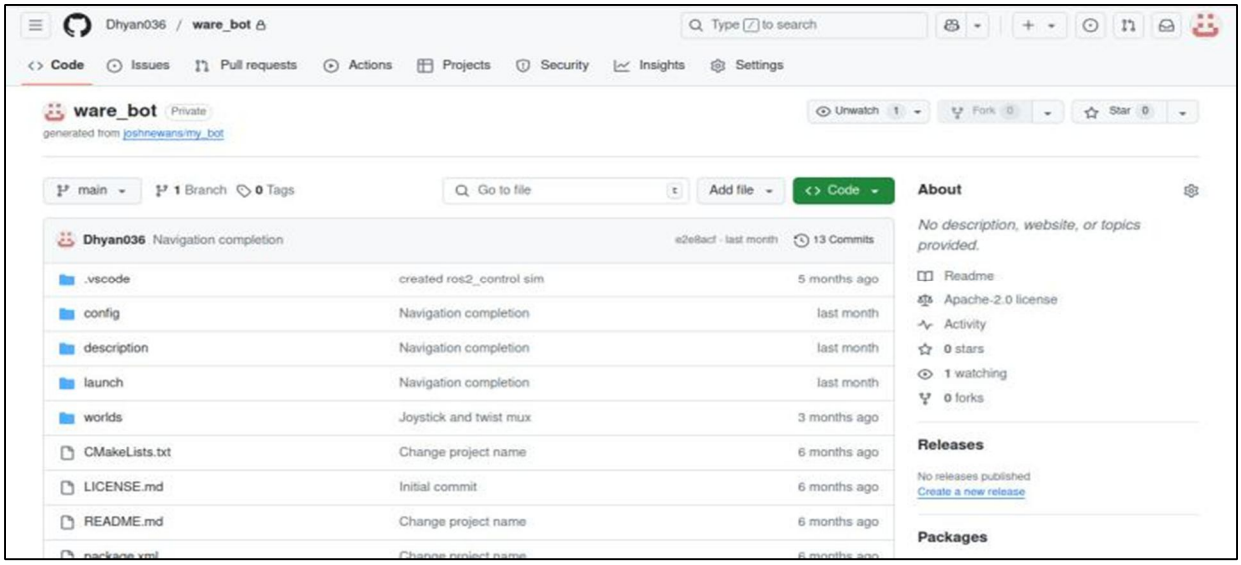


Fig 5.3

3) Linux Configuration

Ubuntu 22.04 LTS "Jammy Jellyfish" plays a crucial role in the smooth functioning of this robotics project. As the officially supported operating system for ROS 2 Humble Hawksbill, it provides a stable and long-term supported environment essential for development and deployment. It comes with pre-installed Python 3.10 and modern system libraries, which are compatible with the ROS 2 ecosystem. This makes it ideal for building and running packages like rplidar_ros and cartographer_ros, which are essential for LiDAR integration and real-time SLAM mapping. Furthermore, Ubuntu 22.04 ensures better hardware compatibility for Raspberry Pi 4B, enabling efficient use of USB devices like RPLidar and providing smooth serial communication via /dev/ttyUSB0.The Linux kernel and system utilities in Ubuntu 22.04 help manage resources effectively, making it suitable for running the robot in a headless setup. The system's compatibility with tools like systemd, udev, and SSH simplifies the setup of automated launch processes and remote control from a laptop. Package managers like apt allow easy installation of critical dependencies, while tools such as colcon and rviz2 enable building and visualizing mapping data. Altogether, Ubuntu 22.04 acts as the foundation of this SLAM-based mapping system, supporting both the real-time data flow from the LiDAR sensor and the processing of that data into navigable maps using ROS 2.

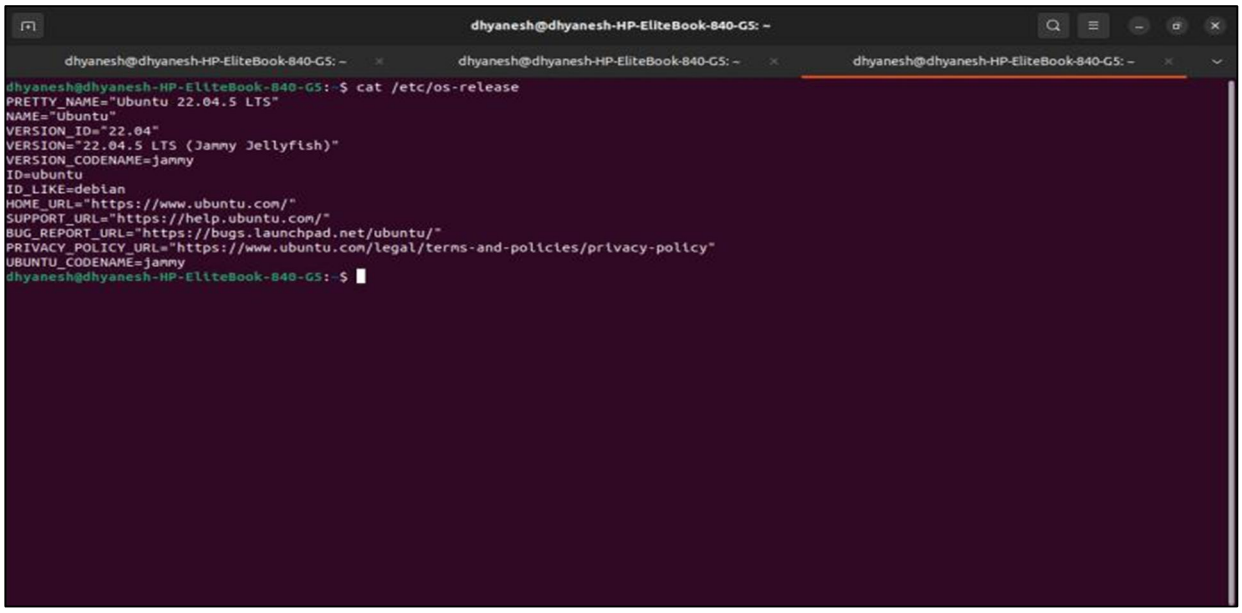


Fig 5.4

4) RViz

RViz (ROS Visualization) is a 3D visualization tool in ROS used to visualize sensor data, robot models, and the environment. It displays data like LiDAR scans, camera feeds, and robot poses in real time, helping developers debug and monitor robot behaviour effectively. It helps in providing the navigating goal commands and many mapping commands.

ros2 run rviz2 rviz2

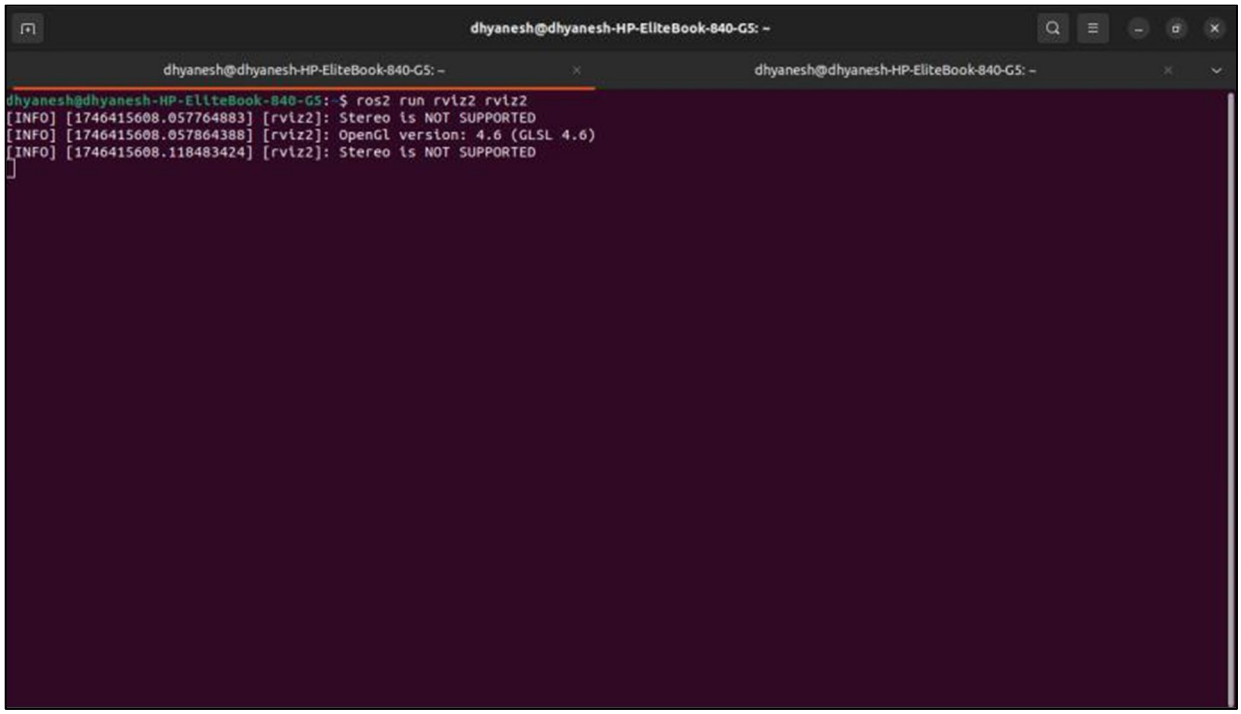


Fig 5.5

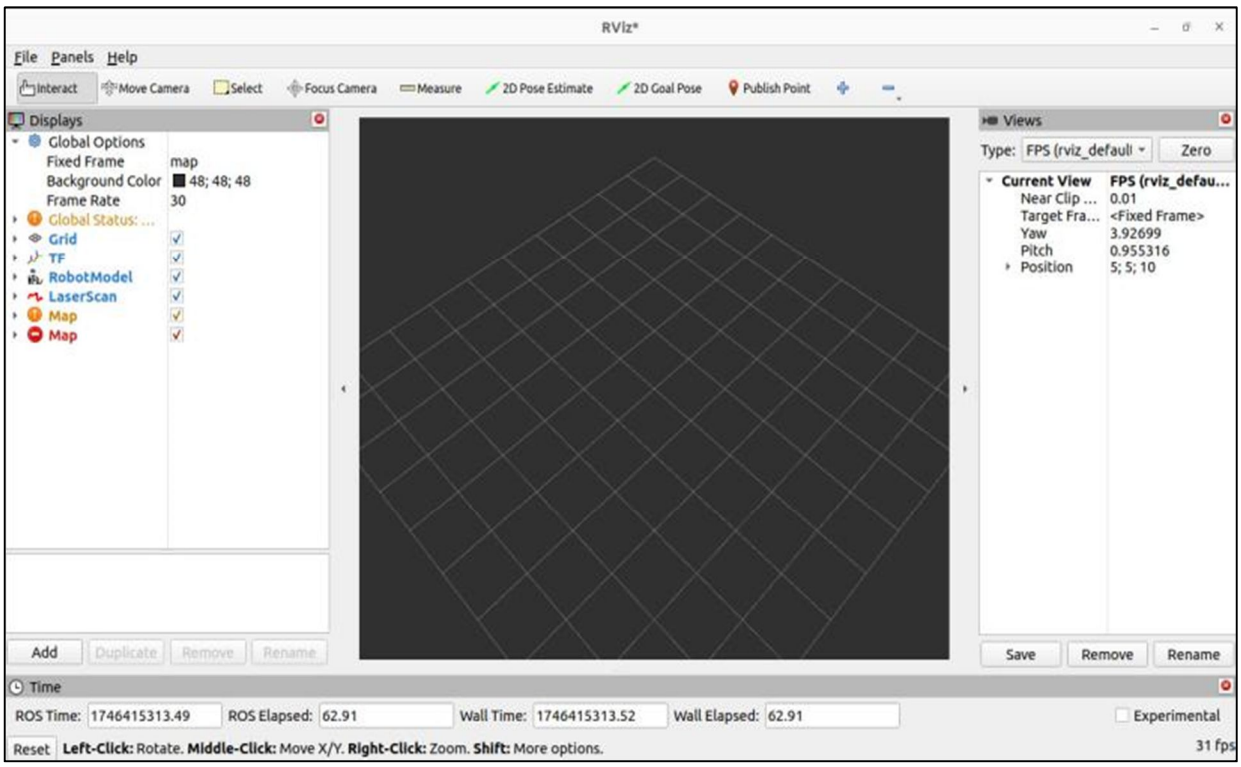


Fig 5.6

5) Gazebo

Gazebo is a powerful 3D robotics simulator used with ROS to create realistic environments for testing robots. It simulates physics, sensors, and actuators, allowing developers to evaluate robot performance and behaviour without physical hardware. This is the software to make our simulation work efficient with many parameters.

For installing the gazebo simulation software type

sudo apt install ros-foxy-gazebo-ros-pkgs

To run Gazebo, write the following command

ros2 launch gazebo_ros gazebo.launch.py

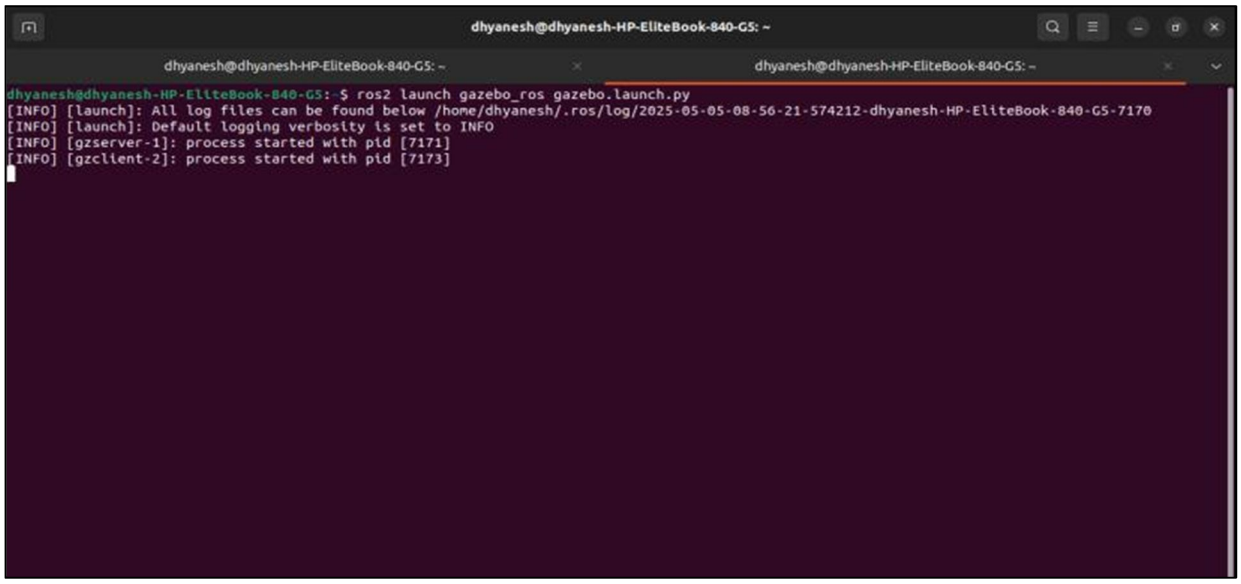


Fig 5.7

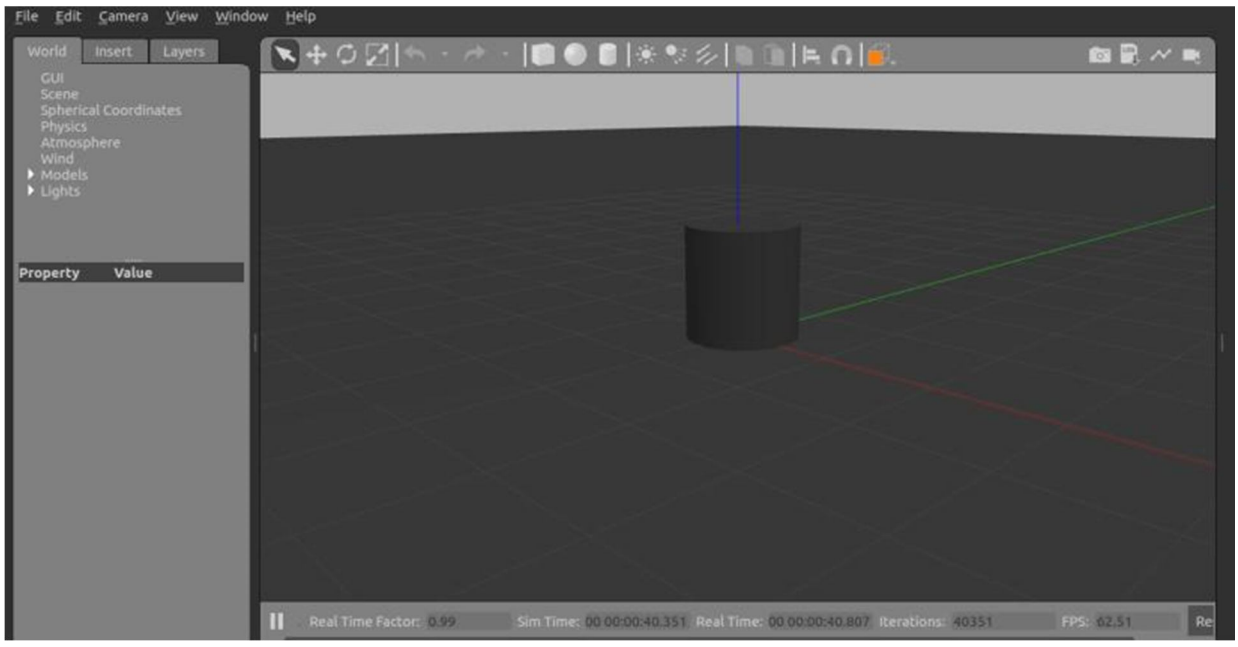


Fig 5.8

6) SLAM

SLAM is a technique used by robots to build a map of an unknown environment while simultaneously tracking their own position within it. In this project, SLAM enables the robot to navigate and map its surroundings using LiDAR data in real-time. For installing SLAM Toolbox, we need to write a sudo command which is

sudo apt install ros-humble-slam-toolbox

To launch the SLAM toolbox the command is given below but to launch in simulation we make

use_sim_time:=true
ros2 launch slam_toolbox online_async_launch.py
slam_params_file:= ./src/lidar_slam/config/mapper_params_online_async.yaml
use_sim_time:=false

7) Navigation 2

Nav2 is the navigation stack for ROS 2 that enables autonomous movement of robots in a known or unknown environment. It provides path planning, obstacle avoidance, and goal- reaching functionalities by processing sensor data and controlling the robot's motors accordingly. To install Nav2 in the work space we need a command

sudo apt install ros-<ros2-distro>-navigation2
sudo apt install ros-<ros2-distro>-nav2-bringup

We also need Turtle Bot Package because of the design of the robot which is the same differential drive control used Turtle Bot

sudo apt install ros-<ros2-distro>-turtlebot3-gazebo

To launch the NAV2 the command is given below but to launch in simulation we make

use_sim_time:=true
ros2 launch ware_bot navigation_launch.py use_sim_time:=false

8) Twist MUX

Twist Mux is a ROS package that manages multiple velocity input sources and forwards the selected one to control the robot. It ensures that different nodes (like joystick, autonomous navigation, or teleoperation) don't conflict when sending movement commands to the robot. To install Twist MUX in this machine we need

sudo apt install ros-humble-twist-mux

To launch the Twist mux the command is given below

ros2 run twist_mux twist_mux --ros-args --params-file
./src/ware_bot/config/twist_mux.yaml -r cmd_vel_out:=diff_cont/cmd_vel_unstamped

9) Joy stick

A joystick is used for teleoperation by manually controlling the robot's movement through ROS. Gamepads are great for robotics because they are already designed to easily direct something to move around, it is usually just a virtual. They typically have a few different axes and a whole lot of buttons, enough to trigger any functions we will require. To check our gamepad works in Linux, we want to install some useful tools and test using the next command.

sudo apt install joystick jstest-gtk evtest
evtest

Assign the parameters for the joystick in a params with the buttons representing different axis and even a button for turbo which increase speed of the robot. There need to be a launch file for the launching it with the params file to make it convenient. The files mention is in the git hub provided for reference .To launch the Twist mux the command is given below

ros2 launch ware_bot joystick.launch.py

10) LiDAR

The robot will be using the RPLIDAR A1 by SLAMTEC we need the driver software, to talk to the laser and publish a LaserScan topic. There are actually a few different versions of the driver and we will be using the one from the package repos. To install the driver, launch the command

sudo apt install ros-foxy-rplidar-ros

The lidar is launch in the raspberry pi terminal which is launched with a lot of arguments to work efficiently like the baud rate, port name, etc. To launch the LIDAR the command is given below

**ros2 run rplidar_ros rplidar_composition --ros-args **
-p serial_port:=/dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb- 0:1.3:1.0-port0
**/ -p frame_id:=laser_frame **

**-p angle_compensate:=true **
-p scan_mode:=Standard

11) SSH

SSH is a network protocol that allows secure remote login from one computer to another. It encrypts all data exchanged between the client and server, including passwords, commands, and output, making it ideal for managing servers remotely.

SSH Command Example:

ssh ubuntu@192.168.1.10

IP Address: 192.168.1.10

User Account: ubuntu

12) ROS2 Control

ROS 2 Control is a modular framework for managing hardware interfaces and robot controllers. It separates control logic from hardware, allowing easier simulation and real robot integration. It uses controller manager to load, start, and switch between controllers like

joint_state_broadcasterand.diff_drive_controller.

With URDF and ros2_control tags, robots can be configured for velocity, position, or effort control. This enables precise and scalable control of motors in both simulation and real-world applications. To install the ros2 control write the following code

sudo apt install ros-foxy-ros2-control ros-foxy-ros2-controllers ros-foxy-gazebo-ros2-control

The core urdf should be updated with the ros2 control urdf and a controller params file must be created. Now to use the ros2 control type ros2 control and hit tab we can see all the options. The only one we'll use right now is **list_hardware_interfaces** and sure enough, we can see our hardware interfaces.

13) Visual Studio Code

Visual Studio Code (VS Code) served as the main code editor in this project for developing and managing ROS 2 packages. It provided a user-friendly interface with powerful extensions such as Python, C++, and ROS, which enhanced code editing, auto-completion, and syntax highlighting specifically tailored to robotics development. The integrated terminal in VS Code allowed direct execution of ROS 2 commands, while Git integration simplified version control and project management. With the help of the "Remote - SSH" extension, VS Code was also used to connect and edit files directly on the Raspberry Pi running Ubuntu 22.04, eliminating the need for transferring files manually and enabling real-time development over the network.

To operate the Raspberry Pi in a headless mode, an SSH (Secure Shell) connection was configured between the Raspberry Pi and the main laptop. After enabling SSH on the Raspberry Pi and obtaining its IP address using tools like ifconfig or IP scanners, the connection was established using a terminal-based SSH client. This enabled full remote access to the robot's file system and terminal, allowing for launching nodes, monitoring LiDAR data, and controlling the mapping process. It streamlined development and debugging without needing physical peripherals connected to the Raspberry Pi.

- Navigate to the source folder of the workspace:
  cd ~/ros2_ws/src
- Clone the necessary repository from GitHub:
  git clone https://github.com/<username>/<repository>.git
- Return to the root of the workspace:
  cd ~/ros2_ws
- Install all required dependencies:
  rosdep install --from-paths src --ignore-src -r -y
- Build the entire workspace:
  colcon build --symlink-install
- Source the workspace to load the environment:
  source install/setup.bash

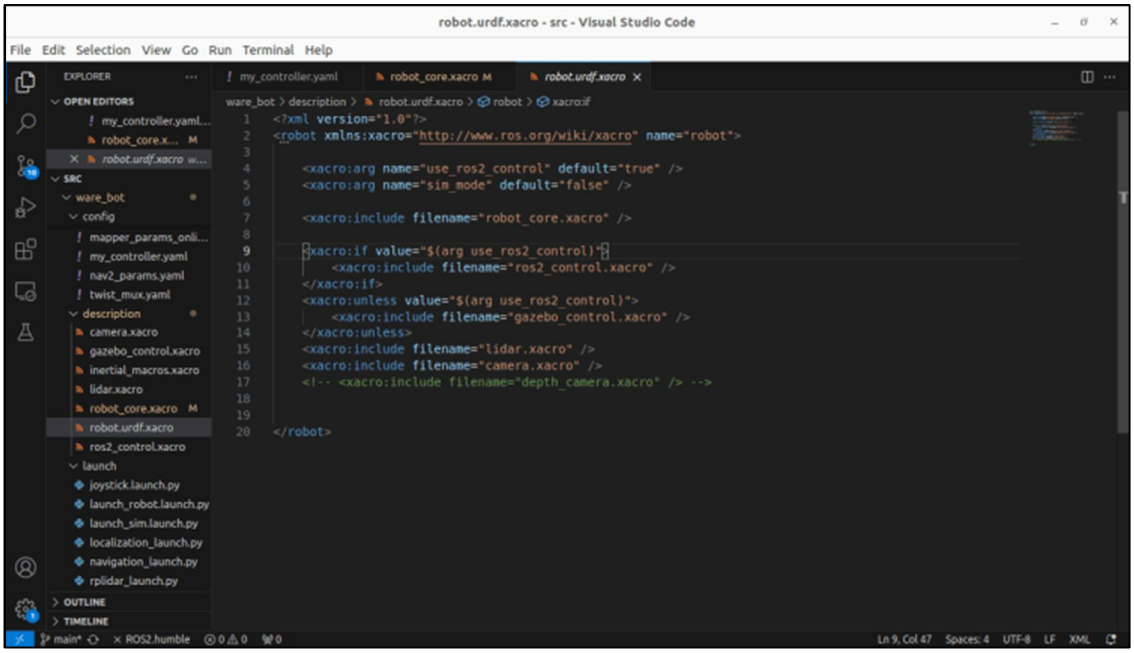


Fig 5.9

14) Simulation Launch

To begin the robot simulation, a custom launch file is used that initializes Gazebo, loads the URDF model, and launches necessary ROS 2 nodes. This launch file typically includes the robot description, world file, controller setup, and optional RViz visualization The above files are in the git hub given for this robot.in the to launch the Simulation the command is given below

ros2 launch ware_bot launch_sim.launch.py world:=./src/ware_bot/worlds/test2.world

15) Real Life Launch

To begin the real robot, a custom launch file is used itloads the URDF model, and launches necessary ROS 2 nodes. This launch file typically includes the robot description, controller setup, and optional RViz visualization. This will be mostly cloned into the raspberry pi after verifying the simulation for the safety of the hardware components. The commands and code can be viewed from the terminal of the raspberry pi using SSH. To launch the Simulation the command is given below

ros2 launch ware_bot launch_robot.launch.p

VI. TESTING OF SYSTEM

A. Introduction

Testing is a critical phase in the development of a robotic system, ensuring that both hardware and software components operate correctly and reliably. In this project, system testing involved validating sensor data accuracy, SLAM map quality, robot movement, and autonomous navigation behaviour. Comprehensive testing was essential to confirm that all subsystems worked together as expected and the robot could safely operate in real-world conditions.

B. Component Testing

Component testing involves evaluating each individual module of the robotic system separately before full system integration. This phase is crucial to isolate issues and ensure every hardware and software component functions correctly on its own.

1) LiDAR

The testing of lidar must be a crucial and necessary because its complicated source. Install all the driver for the lidar as per the instruction given on the specified lidar blogs and run the command given below.

ros2 run rplidar_ros rplidar_composition --ros-args -p serial_port:=/dev/ttyUSB0 -p frame_id:=lidar_link -p angle_compensate:=true -p scan_mode:=Standard

For the finding the serial port run the command below.

Ls *dev*/tty*

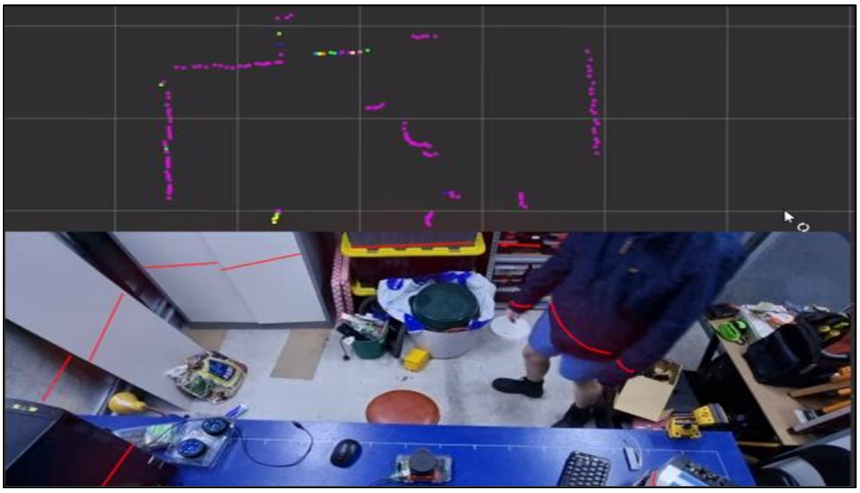


Fig 6.1

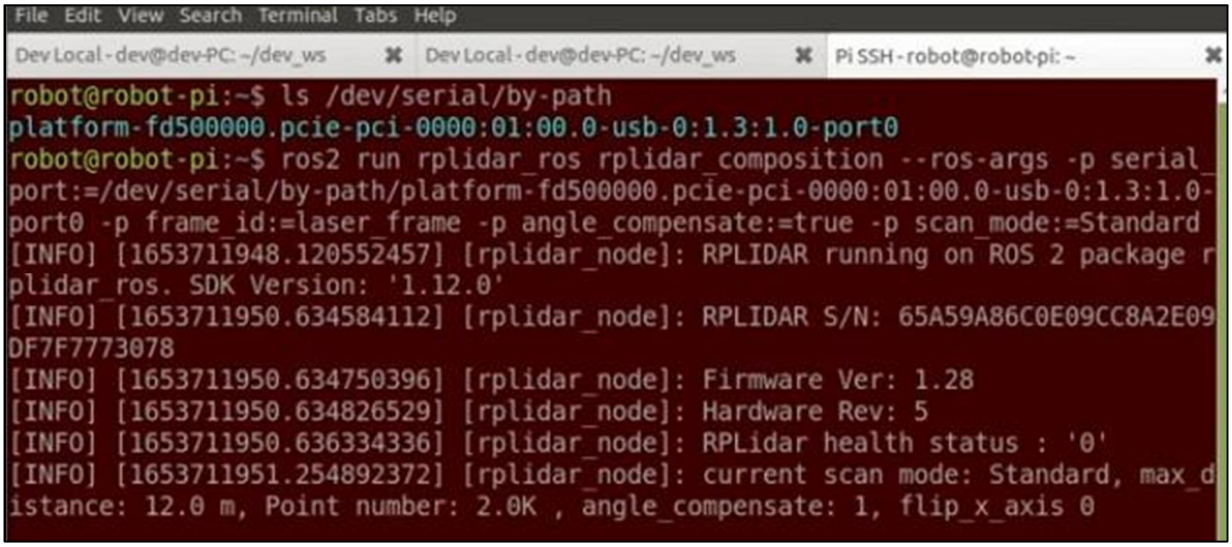


Fig 6.2

2) Motor Testing

Motor testing was carried out to verify the functionality of the robot's drive system. Each motor was independently controlled using the microcontroller and tested for direction, speed, and response time. PWM signals were varied to observe changes in speed, and the direction pins were toggled to test movement in both directions.

Proper feedback was received from the motor drivers, confirming correct wiring. The motors exhibited smooth rotation without any jittering or overheating. Testing also ensured that both forward and backward motions were stable.

This confirmed that the motor control logic and hardware setup were functioning as expected.

C. Robot Testing

1) SLAM Testing

The SLAM (Simultaneous Localization and Mapping) module was tested using SLAM toolbox and with the RP LiDAR sensor. The robot was manually moved around the environment to generate a 2D map in real-time. The generated map was accurate, and loop closure was successfully detected during mapping.

Before running the SLAM Toolbox for mapping we must ensure that the related commands are up and running. This includes Twist MUX, Joystick & Rviz.

ros2 run rviz2 Rviz2
ros2 run twist_mux twist_mux --ros-args --params-file ./src/ware_bot/config/twist_mux.yaml -r cmd_vel_out:=diff_cont/cmd_vel_unstamped
ros2 launch ware_bot joystick.launch.py

Next in the raspberry pi run the following commands like Lidar and the robot launch.

- **ros2 run rplidar_ros rplidar_composition --ros-args **
**-p serial_port:=/dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.3:1.0-port0 / -p frame_id:=laser_frame **
**-p angle_compensate:=true **
-p scan_mode:=Standard
- **ros2 launch ware_bot launch_robot.launch.py**

Run all the above command in different terminal and check whether all the commands are running properly without any error. Next run the SLAM tool box command in different terminal and make sure the use_sim_time:=false because running in real time the simulation time should be disabled for correct communication.

ros2 launch slam_toolbox online_async_launch.py
slam_params_file:= ./src/lidar_slam/config/mapper_params_online_async.yaml use_sim_time:=false

After running the slam toolbox go to Rviz and change the following steps:

- Set fixed frame to map
- Add TF, Robot model, Laser scan & Map.
- Set the robot model description topic / robot_description
- Set laser scan topic to /scan
- Set map topic to /map

Finally, after setting all the parameters in Rviz start driving the robot all over the space in a steady pace to create a clean map and make sure most of the pace is covered without any gaps.

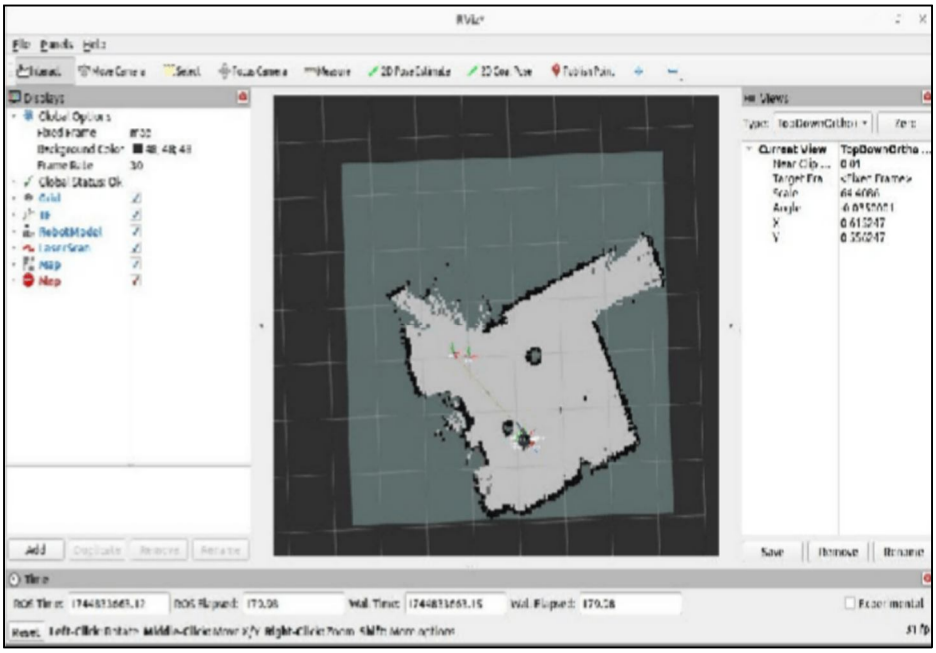


Fig 6.3

After creating a map open the panels tab and add a slam toolbox plugin panel for saving the created map. Add a name and click save map and sterilize map. This will save the map as two different file and can be used for different types of localizations and navigation.

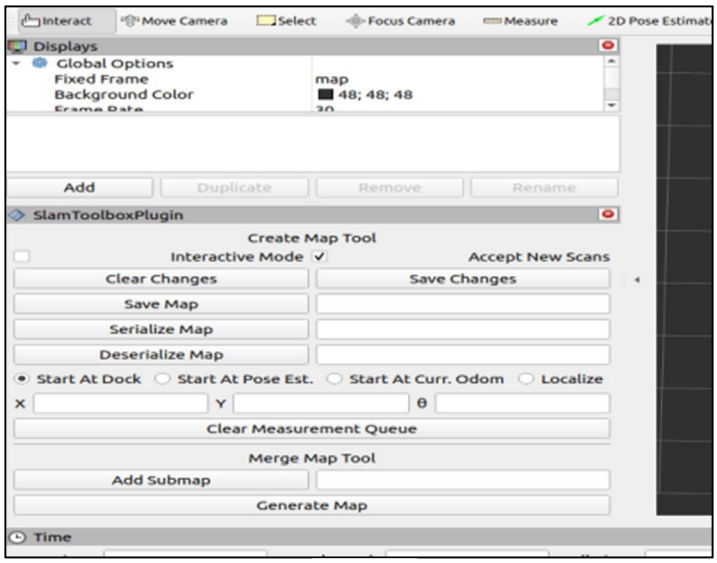


Fig 6.4

2) Navigation Testing

The Nav2 stack was tested by setting multiple navigation goals in RViz2. The robot successfully planned paths, avoided obstacles, and reached the target positions using dynamic path planning. The navigation was smooth and reliable, confirming the effectiveness of the Nav2 system. For running the navigation stack run the following command

ros2 launch ware_bot navigation_launch.py use_sim_time:=false

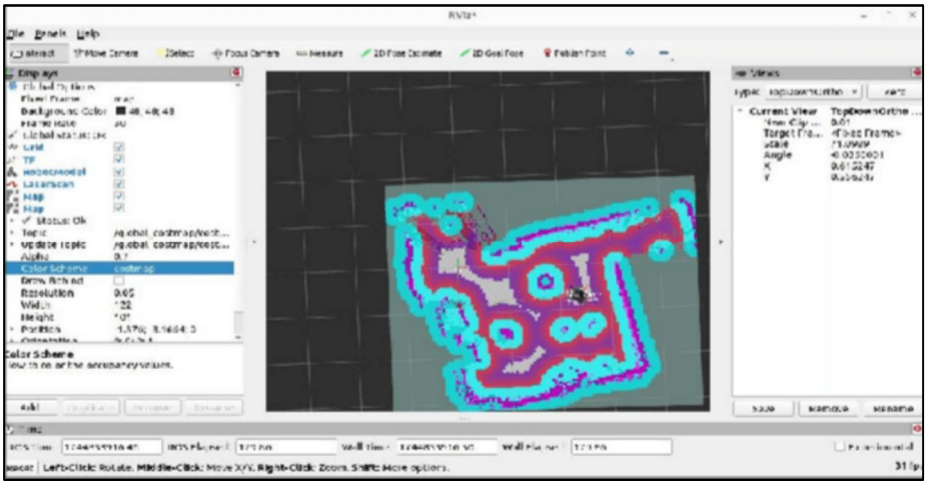


Fig 6.5

The Nav2 stack was tested by setting multiple navigation goals in RViz2. The robot successfully planned paths, avoided obstacles, and reached the target positions using dynamic path planning. The navigation was smooth and reliable, confirming the effectiveness of the Nav2 system. For running the navigation stack run the following command

ros2 launch ware_bot navigation_launch.py use_sim_time:=false

Just like SLAM we have to set a few parameters in RVIZ therefore the following steps are to be changed

- Add another map and set the topic to global cost map.
- Set the map theme to cost map.
- Add another panel navigation tool plugin
- Add navigation goal tool to the tools tab

Use the goal pose tool to give the goal location and with panel open we can give multiple goal at single usage.

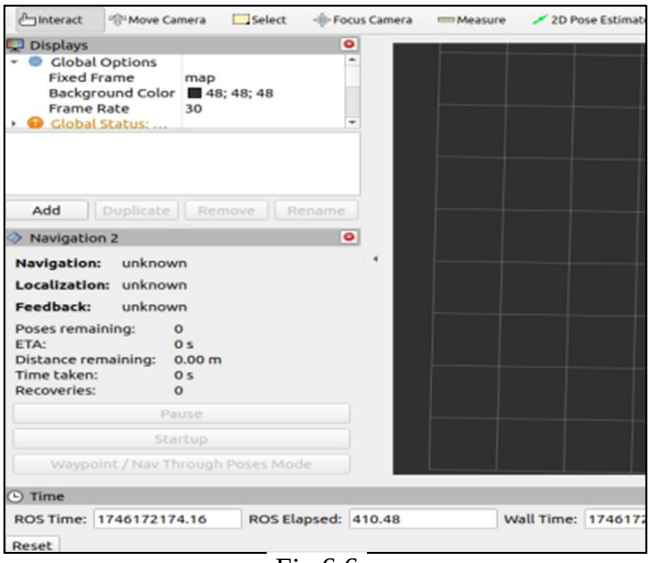


Fig 6.6

D. List Of Components & Its Costs

Table 7.1 shows the components used in this project in this project and their costs:

S.No	Component Name	Price
1	RP LiDAR A1M8	₹ 8,531.00
2	Raspberry Pi	₹ 7,670.00
3	External USB Hub	₹ 1,300.00
4	Arduino Nano	₹ 250.00
5	12V DC motor with Encoder+L29 Driver	₹ 1,110.00
6	Battery (4400 mAh)	₹ 1,300.00
7	Battery Charger	₹ 499.00
8	Cooling Fan	₹ 590.00
9	Buck Convertor	₹ 250.00
10	Other(Connector, Switches, Wires & etc.,)	₹ 250.00
	TOTAL	₹21,750.00

Table 7.1

VIII. CONCLUSION

The successful completion of the project titled “Design and Development of Autonomous Mobile Robot for Material Handling Application” has marked a significant milestone in advancing cost-effective and intelligent automation for industrial environments. This project aimed to address the limitations of traditional material handling methods—such as manual labour, conveyor systems, and Automated Guided Vehicles (AGVs)—by introducing a flexible, autonomous, and real-time responsive robotic system.

The developed AMR is built on a robust and scalable architecture that integrates key technologies such as Robot Operating System 2 (ROS 2 Humble), Google Cartographer for SLAM-based mapping, RP LiDAR A1M8 for environmental perception, and Raspberry Pi 4B as the core processing unit.

These elements collectively enabled the robot to navigate autonomously, detect and avoid obstacles, and carry out material handling tasks with minimal human intervention.

Component-level and system-level testing validated the performance of each hardware and software subsystem. The robot demonstrated precise mapping, smooth path planning, and successful navigation in simulated as well as real-world environments. Real-time data from LiDAR was processed effectively to generate accurate 2D maps, while the Nav2 stack ensured autonomous movement across dynamic paths. The implementation of Arduino Nano and motor encoders allowed effective motion control and closed-loop feedback, enhancing both accuracy and responsiveness.

This project emphasizes the practical feasibility of deploying autonomous robots in small to medium-sized industries. It highlights a tangible solution to increase productivity, reduce operational costs, and ensure safer working conditions. The learnings from this project lay the foundation for future research and development in mobile robotics, warehouse automation, and intelligent logistics systems.

A. *Advantages*

The autonomous mobile robot developed in this project brings several technical, operational, and economic advantages:

1) Autonomous Functionality: The robot operates without the need for human guidance or pre-defined tracks, using SLAM for dynamic real-time path planning and obstacle avoidance.
2) Enhanced Mapping and Localization: With the integration of RP LiDAR and ROS 2 tools, the robot can create precise environmental maps and localize itself accurately within unknown or frequently changing environments.
3) Cost-Effective Automation: Using affordable components like Raspberry Pi, Arduino Nano, and open-source ROS 2, the robot provides a low-cost solution for automation, especially suited for SMEs.
4) Modular and Scalable Design: The modular hardware and software approach allows for future upgrades or modifications based on new functional requirements, such as adding payload support or collaborative behaviour.
5) Improved Workplace Safety: Built-in safety features like obstacle detection, collision avoidance, and emergency stop mechanisms reduce the risk of workplace injuries commonly associated with manual handling.
6) Efficient Material Handling: By automating repetitive transport tasks, the robot improves workflow speed, reduces downtime, and enhances overall operational efficiency in industrial settings.
7) Minimal Infrastructure Requirements: Unlike AGVs that require tracks, QR codes, or magnetic strips, this AMR system adapts to existing floor layouts with minimal alterations.
8) Reduced Human Error: Automation minimizes handling mistakes, misplacement of materials, and delays caused by manual intervention.

B. *Future Scope*

While the current implementation showcases a fully functional autonomous mobile robot for indoor material handling, there are several avenues for enhancing the system's capabilities:

1) Future improvements can involve the use of 3D LiDAR or depth-sensing cameras for vertical environment mapping. This would allow the robot to navigate multi-level shelves, ramps, or detect hanging obstacles.
2) Incorporating AI algorithms for decision-making would enable adaptive task planning, predictive route optimization, and advanced environmental understanding. It could also support visual object detection for automated item identification and classification.
3) Developing swarm intelligence or a fleet management system would allow multiple AMRs to coordinate tasks in real time, maximizing coverage and improving overall throughput.
4) By connecting the robot to cloud platforms, remote monitoring, data logging, and predictive maintenance can be enabled. IoT integration will also allow seamless communication with Warehouse Management Systems (WMS) and Enterprise Resource Planning (ERP) software.
5) Implementing a smart docking station for battery charging will ensure uninterrupted 24/7 operation. The robot can autonomously return to the dock when the battery level drops below a threshold.
6) Additional modules such as robotic arms, conveyor-based loading systems, or lift platforms can be added for automatic loading and unloading of goods.
7) Future versions can incorporate GPS, IMU, and weather-resistant design for outdoor applications like inter-building logistics or outdoor warehousing.

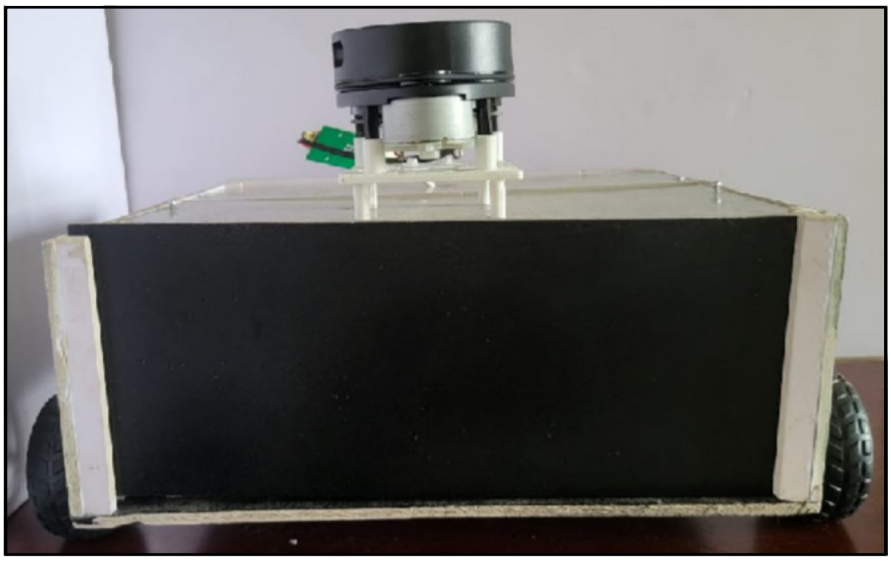
8) Leveraging 5G communication can enhance real-time data exchange and control, allowing faster and more reliable robot-to-cloud and robot-to-robot interactions.

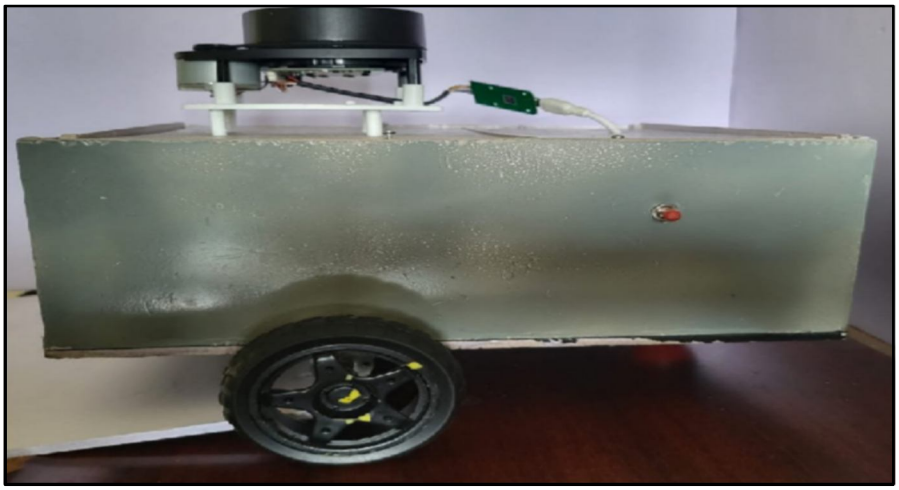
By implementing these enhancements, the AMR can evolve into a highly intelligent, scalable, and collaborative logistics solution capable of functioning in complex, multi-agent, and high- demand environments.

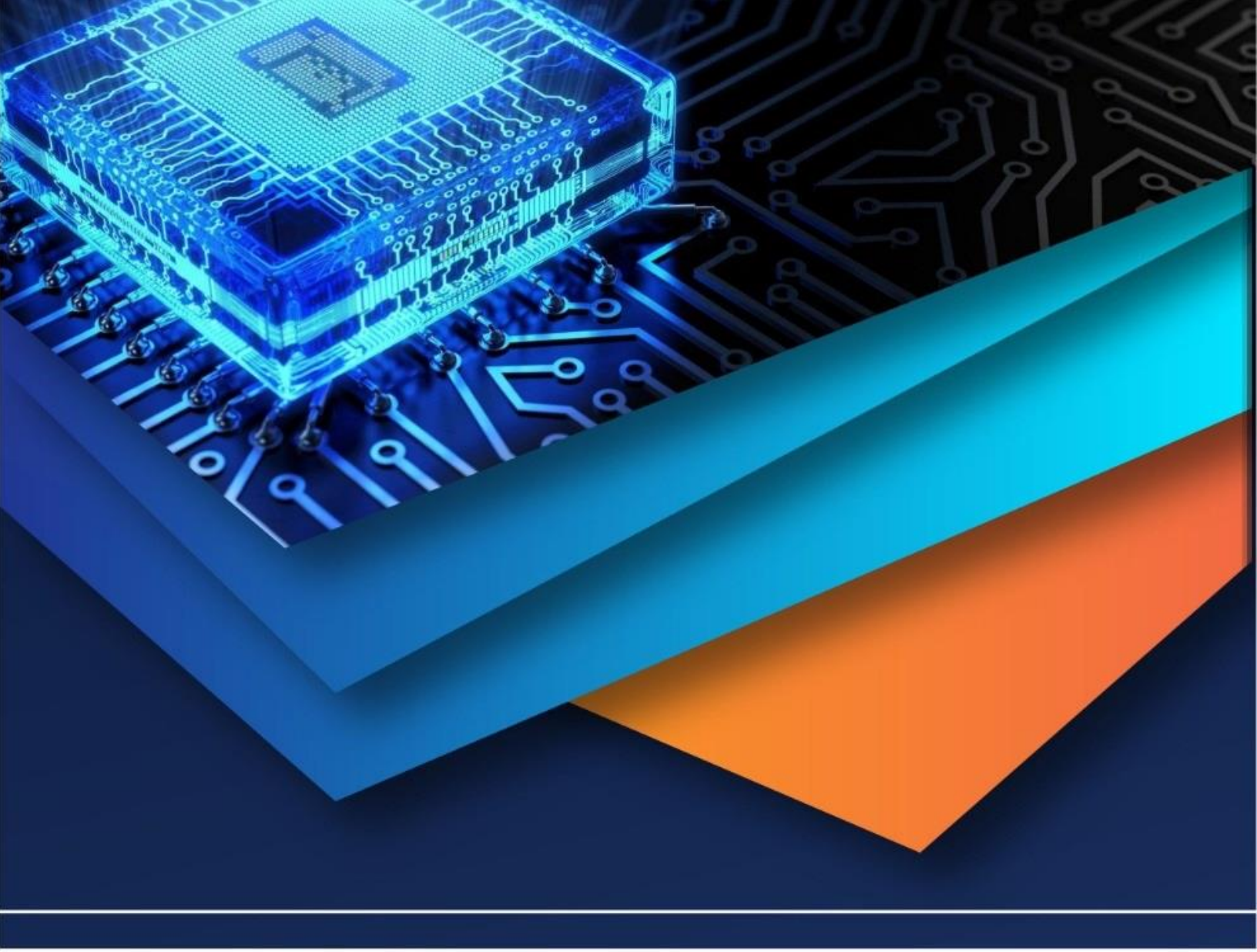
BIBLIOGRAPHY

[1] SLAMTEC, "RPLIDAR A1," 2013.
[2] "slam_toolbox — slam_toolbox 2.6.9 documentation," *Ros.org*, 2021. https://docs.ros.org/en/humble/p/slam_toolbox/
[3] "(SLAM) Navigating While Mapping — Nav2 1.0.0 Documentation," *Nav2.org*, 2023. https://docs.nav2.org/tutorials/docs/navigation2_with_slam.html
[4] "Nav2 — Nav2 1.0.0 Documentation," *docs.nav2.org*. https://docs.nav2.org/
[5] "Home," *Articulated Robotics*. https://articulatedrobotics.xyz/
[6] "ROS 2 Documentation — ROS 2 Documentation: Humble documentation," *docs.ros.org*. https://docs.ros.org/en/humble/index.html

PHOTOGRAPH OF THE PROJECT SETUP







10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY