International Journal of Research and Innovation Management
ISSN 2462-1293
Vol:11 No 1 2025

Development and Implementation of a ROS-Based Differential Drive Robot for Autonomous Navigation

Ku Mohammad Yusri Bin Ku Ibrahim[1], Faizah Binti Utar @ Haji Mokhtar[2], Addzrull Hi-Fi Syam Bin Ahmad Jamil[2]
[1] Department of Electrical Engineering, Politeknik Tuanku Syed Sirajuddin.
[2] Department of Electrical Engineering, Politeknik Seberang Perai.
kmyusri@ptss.edu.my

Abstract: The development of autonomous mobile robots has gained significant attention due to their potential applications in navigation, surveillance and industrial automation. However, achieving precise localization and obstacle avoidance in dynamic environments remains a challenge, particularly for differential drive robots. This study focuses on the development of a ROS-based differential drive robot designed for autonomous navigation. The robot is equipped with a Teensy microcontroller, DC motors with encoders for motion control, an RPLIDAR A1 for mapping and obstacle detection and a GY-85 IMU for inertial sensing and orientation estimation. To address the challenges of localization and navigation, the system integrates ROS-based software packages for sensor data processing, motion control, and real-time path planning. The navigation framework employs a proportional-integral-derivative control strategy to ensure smooth and accurate movement. Additionally, simultaneous localization and mapping techniques are used to construct real-time maps of the environment while enabling autonomous path execution. Experimental evaluations demonstrate the robot's ability to navigate predefined environments efficiently while avoiding obstacles. Preliminary results indicate that the system achieves reliable localization with minimal drift and effective trajectory tracking with position errors of 5.5-12.5 cm with traveled times of 96-123 s. This study contributes to the field of autonomous robotics by presenting a robust and adaptable navigation framework for differential drive robots. The findings highlight the effectiveness of ROS-based integration for autonomous navigation and provide insights into potential improvements for enhancing real-time decision-making, mapping accuracy and system robustness in complex environments.

Keywords: *Autonomous Navigation, Differential Drive Robot, Robot Operating System (ROS), Simultaneous Localization and Mapping (SLAM), Real-Time Mapping.*

1.0 INTRODUCTION

Advancements in open-source robotics software, compact computing platforms and modern sensor technologies have significantly contributed to the progress of autonomous mobile robot development. Differential drive robots, characterized by two independently actuated wheels, have gained widespread adoption due to their mechanical simplicity, ease of control and suitability for indoor navigation, monitoring and industrial automation tasks. Despite these advantages, achieving precise localization and reliable obstacle avoidance in dynamic and unstructured environments remains a considerable challenge (Phueakthong & Varagul, 2021). To address these complexities, recent research emphasizes the integration of flexible software platforms such as Robot Operating System 2 (ROS2), dependable hardware architectures and advanced multi-sensor fusion techniques. This project aims to design and implement a ROS2-based differential drive robot equipped with wheel encoders, a GY85 inertial measurement unit (IMU) and an RPLIDAR A1 sensor for simultaneous mapping, localization

and obstacle avoidance. The overarching objective is to demonstrate a cost-effective, accurate, and reliable autonomous navigation system leveraging widely available open-source technologies.

2.0 LITERATURE REVIEW

The evolution of autonomous mobile robotics has accelerated in recent years, driven largely by advancements in modular software frameworks and integrated hardware systems. The Robot Operating System (ROS), and its enhanced iteration ROS2, have emerged as standard platforms for developing scalable and adaptable robotic systems (Macenski et al., 2022; Phueakthong & Varagul, 2021) . ROS2 offers significant improvements over its predecessor, including enhanced communication reliability, real time data exchange capabilities and decentralized system management. These enhancements are critical for enabling safe and efficient navigation in dynamic and uncertain environments (Phueakthong & Varagul, 2021).

A central technological innovation in modern robotics is sensor fusion, which refers to the combination of data from diverse sensors such as wheel encoders, IMUs, and LiDAR. This methodology enhances the robot's ability to estimate its position and perceive its environment, thereby mitigating the limitations of individual sensors. For instance, integrating wheel odometry, IMU, and visual or LiDAR data through algorithms like the Extended Kalman Filter (EKF) has been shown to improve localization accuracy, particularly in GPS denied or rapidly changing settings (Phueakthong & Varagul, 2021; Song & Zhang, 2024; Yan et al., 2022). Micro ROS, a specialized extension of ROS2, further supports efficient, real-time communication between resource-constrained microcontrollers and the main navigation architecture (Phueakthong & Varagul, 2021).

Another essential aspect is Simultaneous Localization and Mapping (SLAM), which enables a robot to construct a map of its environment while concurrently estimating its location within that map. Tools such as SLAM Toolbox, Cartographer and ORB SLAM2, all compatible with ROS2, facilitate the generation of consistent and accurate two- or three-dimensional environmental maps (Macenski & Jambrecic, 2021; Phueakthong & Varagul, 2021). Experimental research confirms that multi sensor SLAM approaches can maintain localization errors within a few centimeters, which is critical for successful operation in constrained or complex spaces (Song & Zhang, 2024; Yan et al., 2022).

Obstacle detection and avoidance have similarly been advanced through the integration of LiDAR sensors, which provide comprehensive 360-degree environmental scanning. Contemporary algorithms are capable of rapidly processing this data, enabling the detection and avoidance of obstacles in real time (Mochurad et al., 2023; Phueakthong & Varagul, 2021). ROS2's Navigation2 software exemplifies this progress by integrating both global and local planning strategies, continuously updating the robot's trajectory as environmental conditions evolve (Macenski et al., 2022).

Finally, visualization and development tools such as RViz2 are essential for monitoring system performance, facilitating debugging, and ensuring cohesive operation among the system's components (Macenski et al., 2022; Phueakthong & Varagul, 2021).

In summary, present research underscores that the integration of ROS2, advanced sensor fusion, accurate SLAM algorithms, and comprehensive visualization tools is vital for developing reliable autonomous mobile robots. These combined strategies address persistent challenges in localization and obstacle avoidance, directly supporting the objectives of this project.

3.0 METHODOLOGY

This section describes the methodology utilized in developing a ROS-based autonomous differential-drive mobile robot, detailing both hardware and software components systematically. Initially, the mechanical design is explained, highlighting a structured three-layer configuration that optimizes space usage, facilitates maintenance, and improves operational efficiency. Subsequent subsections elaborate on each hardware layer, starting from the bottom layer, which includes essential components such as the Teensy 4.1 microcontroller for real-time control, a custom-designed YHN-ROS board for streamlined sensor integration, a Cytron MDD10A motor driver for precise motor management and a 12V LiPo battery ensuring stable and reliable power distribution.

The middle layer is dedicated to the central processing unit, a Raspberry Pi 4 single-board computer responsible for managing high-level computational tasks. It runs critical operations, including Simultaneous Localization and Mapping (SLAM), autonomous navigation algorithms, and sensor fusion through ROS2. The top layer houses the RPLIDAR A1 sensor, strategically positioned to provide a comprehensive 360-degree environmental scan crucial for accurate mapping and obstacle detection.

Finally, the methodology details the integration of sensors such as the DC gear motors with encoders for accurate odometry and the GY-85 IMU for refined orientation data through sensor fusion techniques, particularly the Extended Kalman Filter (EKF). Software architecture leveraging ROS2, Micro-ROS for microcontroller integration and RViz2 for real-time visualization and monitoring is also described, offering readers a clear insight into the integrated system architecture that supports autonomous navigation.

3.1 Mechanical Design and Hardware Architecture

This robot's structural architecture consists of three layers, each of which contains specific components necessary for operation. In addition to improving space management and aesthetics, this well-organized design improves the robotic system's overall efficiency and maintainability.

The bottom layer of the robot acts as the base and contains essential hardware parts that control movement and power distribution, as seen in Figure 1. The Teensy 4.1 microcontroller, the specially made YHN-ROS board, the Cytron MDD10A motor driver and a 12V LiPo battery are all included. The YHN-ROS board aids in optimizing the wiring and connection between the encoder, IMU, and motor driver, providing a small and dependable system. The Cytron MDD10A driver is in charge of managing the motors by receiving PWM signals from the microcontroller. Separate from the computer systems, the 12V LiPo battery supplies the motors with steady and sufficient power, avoiding voltage drops while in motion.

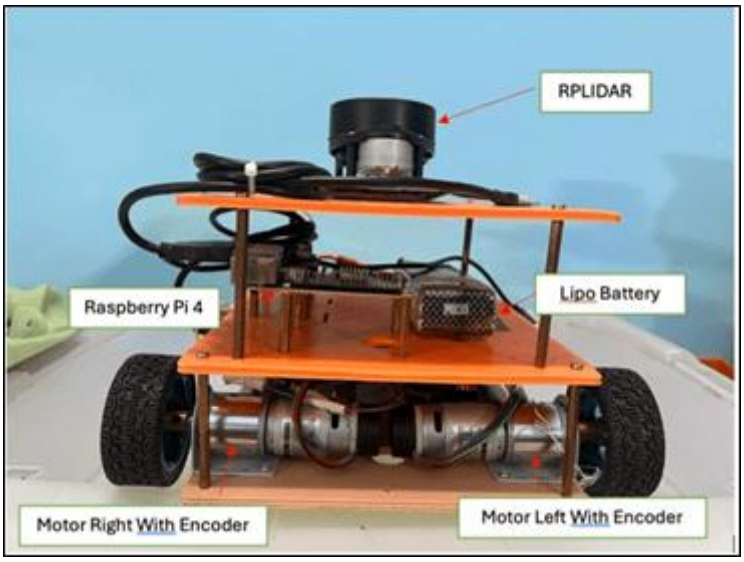


Figure 1: Design of the Robot.

Raspberry Pi 4 the robot's central processing unit is in the middle layer. Running the ROS (Robot Operating System) and managing sophisticated operations like mapping, navigation and sensor fusion are the responsibilities of this single board computer. The LiDAR sensor is positioned in the top layer, which gives it an unhindered 360-degree view for precise localization and mapping. For the sensor to operate at its best and provide accurate environmental perception, this position is essential.
Two DC Gear Motors, each with a speed of 60 RPM and a torque of 6.73 kgf-cm, are mounted at the front of the robot to facilitate movement. To provide odometry data, these motors are equipped with encoders. This arrangement guarantees balance, stable motion, and smooth turning due to two wheels with a 7 cm diameter paired with a caster ball at the back.

3.2 Electronic Devices

3.2.1 Single Board Computer (SBC)

This robot was developed using a Raspberry Pi 4 Single Board Computer (SBC) with 8GB of RAM and a 64GB Micro SD card for storage. The Raspberry Pi Foundation created the Raspberry Pi 4, a credit card-sized computer that can do many of the functions of a traditional desktop computer. It is a well-liked option for robotics and embedded system applications due to its small size, low cost and adaptability.

The Raspberry Pi 4 functions as the primary processing unit in this robotic system and is essentially the robot's “brain”. It is in charge of controlling communication with external devices like the LIDAR sensor and the Teensy 4.1 microcontroller. The Raspberry Pi 4 runs key software components of the Robot Operating System (ROS) framework and manages real-time data processing. Simultaneous Localization and Mapping (SLAM), mapping, navigation and sensor fusion algorithms comprise some of the primary features built into the Raspberry Pi 4. These elements are essential to the robot's ability to function autonomously because they allow it to sense its surroundings, locate itself on a map and make smart choices about how to navigate through space. The system also benefits from the Raspberry Pi 4's effective multitasking and high-level computation capabilities. Moreover, it makes it easy to communicate with other systems, like a PC running Ubuntu 20.04, enabling remote software updates, debugging and monitoring throughout development and deployment.

In this study, the Raspberry Pi 4 is a general strong and affordable computing platform. By bridging the gap between high-level autonomous decision-making and low-level hardware control, it is essential to the robot's ability to operate dependably and intelligently in its surroundings.

3.2.2 Teensy 4.1

This robot operates by the Teensy 4.1, a high-performance ARM Cortex-M7 microcontroller that is known for its dependability and speed in real-time applications. Teensy 4.1 is an essential component of this robotic system because it acts as a bridge between the Raspberry Pi 4-powered ROS2 system and several low-level hardware parts, including encoders, DC motors and the GY-85 IMU sensor.

Controlling the motors' speed and direction in response to velocity commands from the ROS2 topic /cmd_vel is one of the Teensy 4.1’s primary tasks, as seen in Figure 2. Micro-ROS, which allows communication between the microcontroller and the ROS2 environment, is used to transmit these commands. To ensure that the robot travels along the correct path, the microcontroller interprets the commands and modifies the PWM signals sent to the motor driver.

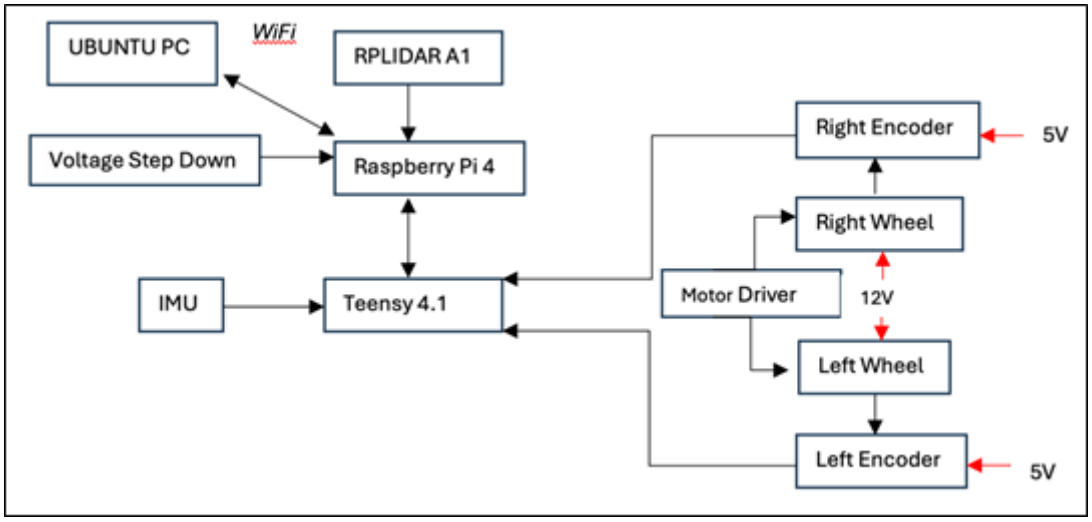


Figure 2: Hardware Architecture.

Apart from controlling the motors, the Teensy 4.1 is also in charge of gathering odometry information from the encoders that are connected to the motors. This information comprises the robot's estimated position, current velocity and distance traveled. After processing, the microcontroller sends this data to the ROS2 system, where it serves as a vital input for the navigation and SLAM algorithms. Building maps and monitoring the robot's movement in real time require precise odometry.

Additionally, the Teensy 4.1 reads information from the GY-85 IMU, which consists of a magnetometer, gyroscope and accelerometer. Pitch, roll, and yaw orientation data from these sensors are crucial for increasing the robot's directional accuracy. In SLAM processes, where accurate positioning and heading estimates have a major impact on mapping quality and navigation success, this orientation information is highly useful.

Overall, the Teensy 4.1 microcontroller connects low-level sensor and actuator hardware to the high-level ROS2 framework, forming the core of the robot's real-time control system. For autonomous navigation, real-time sensor fusion and dependable robot operation to be possible, its effective data handling and control capabilities are essential.

3.2.3 YHN-ROS Board

A specially designed development board called the YHN-ROS Board was designed to make it easier to integrate important hardware parts of an autonomous robot system. Specifically made to work with the effective Teensy 4.1 microcontroller, this board offers specific connections for key parts like the motor driver, IMU GY-85 and motor encoders. The YHN-ROS Board provides a centralized platform for hardware interfacing, which significantly simplifies wiring and lowers the possibility of human error during assembly.

Organizing the complex wiring required to connect multiple sensors and actuators is one of the most frequent problems encountered when developing autonomous robots. The process can be tedious, prone to errors and challenging to debug in the absence of a standardized interface. The Teensy microcontroller and peripherals are connected correctly thanks to the YHN-ROS Board's compact design and clearly labelled pins. In addition to increasing system dependability, this speeds up the development process's testing and prototyping stages. Using the open-source PCB design program KiCad, the YHN-ROS Board is a prime example of effective hardware design for robotics applications. Since KiCad offers flexible customization, developers can alter the board layout to suit hardware configurations or project requirements.

KiCad was used in a systematic design process to create the YHN-ROS Board. The YHN-ROS Board was developed to support the Teensy 4.1 microcontroller for robotics applications, as demonstrated in Figure 3. As illustrated by Figure 4, the design started with circuit development in KiCad's schematic editor, where necessary parts like the encoder, motor drivers, microcontroller and IMU were logically connected. Following verification, KiCad's PCB editor was used to complete the layout process, producing the board design shown in Figure 5. In order to verify component placement and enclosure compatibility, a comprehensive 3D representation of the board was finally produced, as shown in Figure 6.

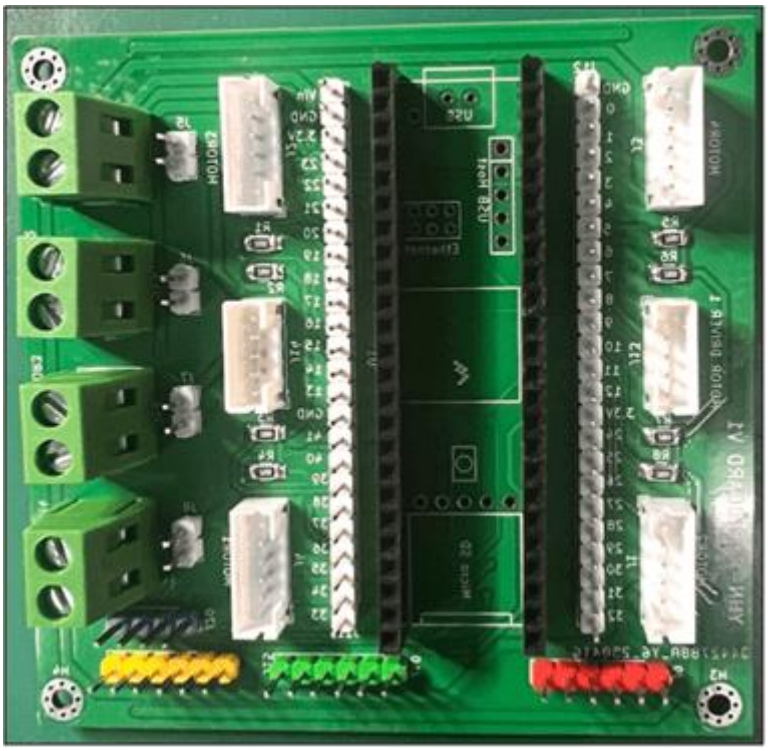


Figure 3: YHN-ROS Board.

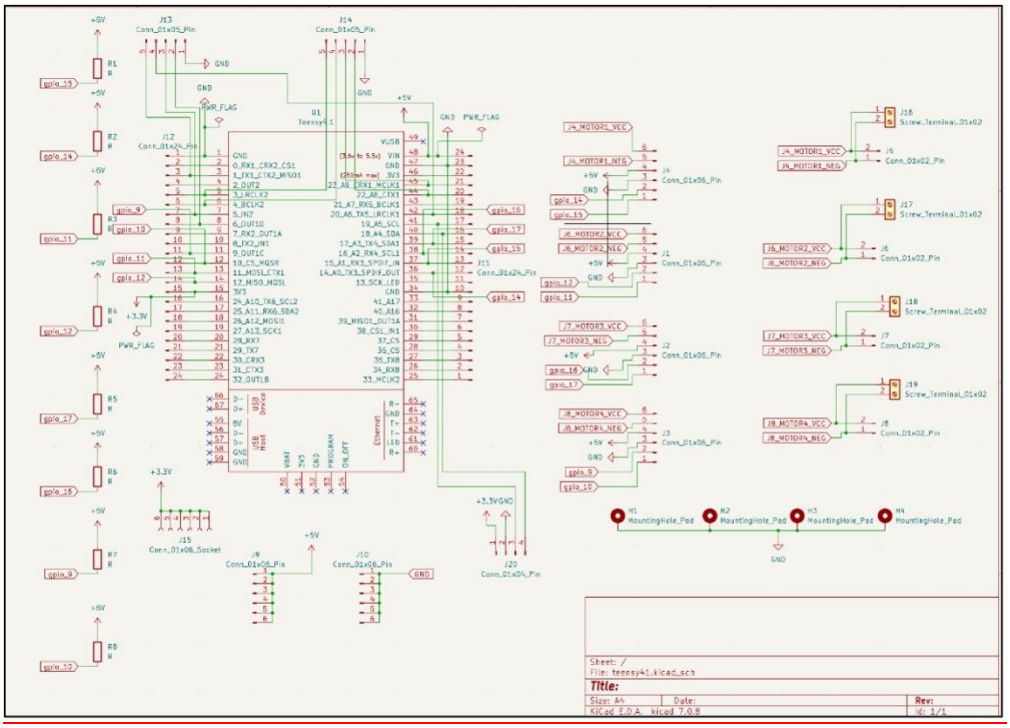


Figure 4: Circuit design of YHN-ROS Board Using KiCad.

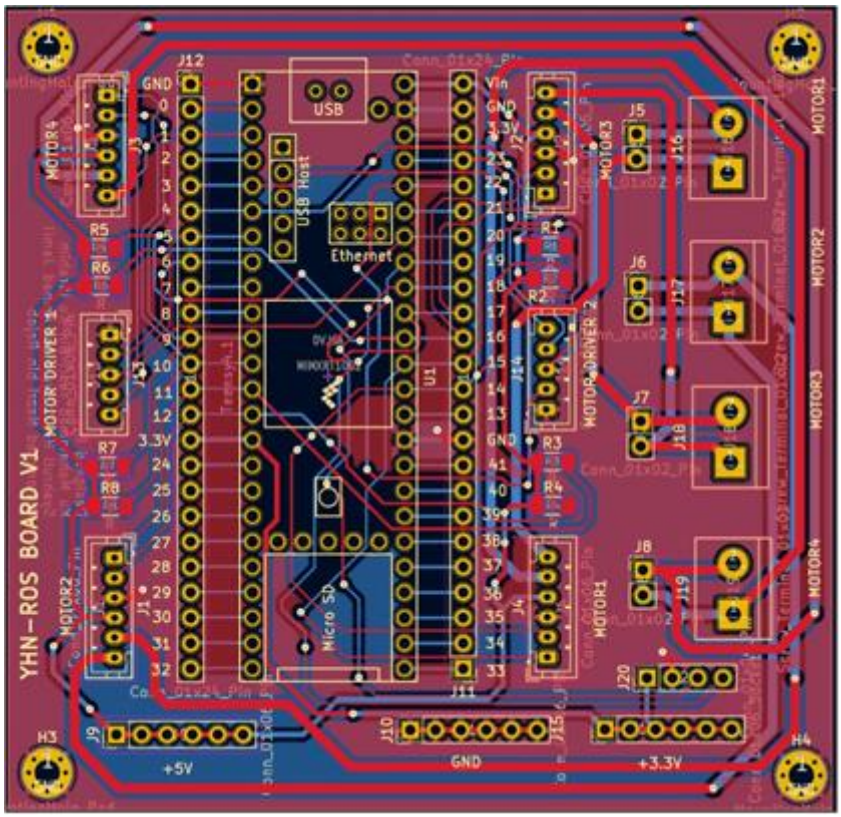


Figure 5: PCB design of YHN-ROS Board Using KiCad.

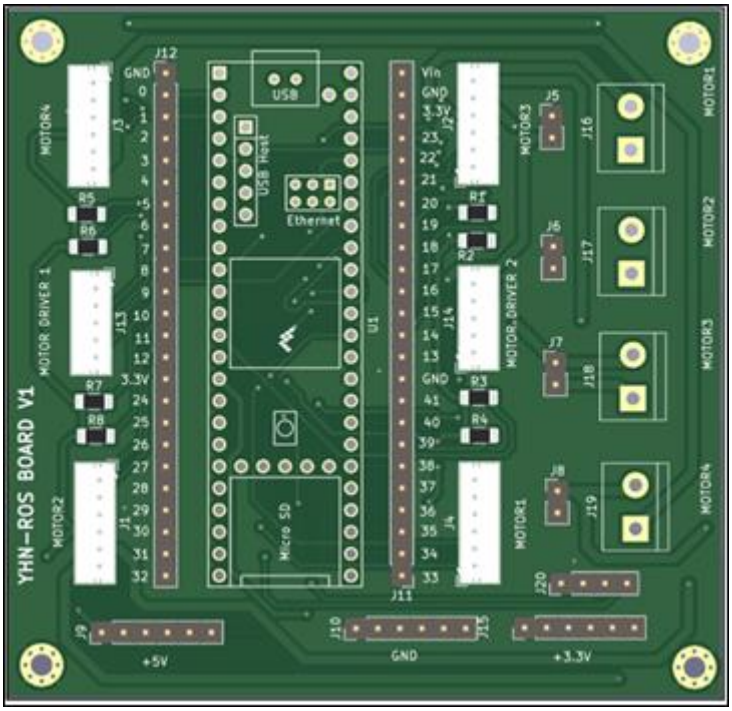


Figure 6: 3D View of YHN-ROS Board Created Using KiCad.

3.2.4 Sensor LIDAR (Light Detection and Ranging)

LIDAR data is commonly published as messages in the ROS ecosystem with the topic sensor_msgs/LaserScan. Important details like obstacle distances, scan angles and scanning frequency are contained in these messages. Several ROS packages then use this data to carry out crucial tasks like path planning, obstacle avoidance and SLAM (Mochurad et al., 2023). Integrating LIDAR data with other sensor data, such as that from an encoder or IMU, to create a more reliable and accurate navigation system is one of ROS's advantages (Firmansyah et al., 2024; Yan et al., 2022).

The RPLiDAR A1 is the LIDAR sensor utilized in this study. Since it can continuously perform 360-degree scans, this sensor is widely used in robotics. It works well in moderately sized spaces or indoor settings because of its detection range of 0.15 to 12 meters. This feature enables the robot to take a comprehensive image of its environment without having to move the sensor. In addition to being utilized for creating static maps, the RPLiDAR A1's distance data is essential for figuring out the robot's current location in relation to its surroundings. By comprehending its current location and devising a suitable path while dodging obstacles, this allows the robot to navigate on its own.

3.2.5 Inertial Measurement Unit (IMU)

The three main sensors that collectively make up the GY-85 IMU module are a three-axis magnetometer (HMC5883L), a three-axis gyroscope (ITG-3205) and a three-axis accelerometer

(ADX345). Each of these components provides crucial information for orientation and motion tracking. In order to determine the direction and speed of movement, the accelerometer measures linear acceleration in the x, y and z axes. The system can monitor rotational movements thanks to the gyroscope's ability to provide information about angular velocity along the yaw, pitch and roll axes. The robot's heading direction is determined by the magnetometer, which senses the Earth's magnetic field. This is particularly helpful for resolving orientation drift over time.

In order to maintain precise turning and heading during autonomous navigation, the IMU provides real-time data on the robot's angular changes. Nevertheless, both the encoders' and the IMU's data are frequently noisy and prone to random errors or drift. An Extended Kalman Filter (EKF) is used in a sensor fusion technique to address this. To generate a more precise and reliable estimate of the robot's pose (position and orientation), the EKF integrates information from the encoders and IMU (Moore & Stouch, n.d.; Yan et al., 2022). The ekf_node from the robot_localization package in ROS2 can be employed to accomplish this.

3.2.6 Motor System

One of the most important parts of an autonomous robot are its motors, which are directly in control, allowing it to move and navigate its surroundings. In this project, the robot's front section has two motors that are each attached to a wheel and an encoder. By independently adjusting the speed and direction of each motor, these motors' differential drive capability enables the robot to move forward, backward, and turn.

The Cytron MDD10A motor driver, which is capable of controlling high current loads and provides reliable functionality appropriate for robotics applications, powers the motors. The motors are powered by a specialized 12V LiPo battery to ensure consistent and continuous operation. At the same time, a separate 12V LiPo battery powers the Raspberry Pi 4, which acts as the central processing unit for the ROS2 system. By keeping voltage drops and electrical noise from impairing the Raspberry Pi's performance during periods of high motor activity, this separation of power sources is essential for maintaining system stability.

This system balances speed and torque for smooth and controlled mobility, with each motor rated at 60 RPM under normal load. Each motor has an encoder that measures rotational movement, providing the ROS2 system with crucial feedback. Odometry, the ongoing estimation of the robot's position and movement over time, makes use of this feedback. For more complex navigation tasks like path planning, mapping and localization, accurate odometry is crucial.

The function of the YHN-ROS board as a central interface that connects the hardware and the ROS2 ecosystem is highlighted by Figure 7, which shows how the board is connected to the motor driver, encoder and motors. The board controls motor actuation by sending PWM and direction signals to the motor driver after reading encoder signals for odometry. On the other hand, the YHN-ROS board,

which provides battery power to the motors, is connected to the Cytron MDD10A motor driver in Figure 8.

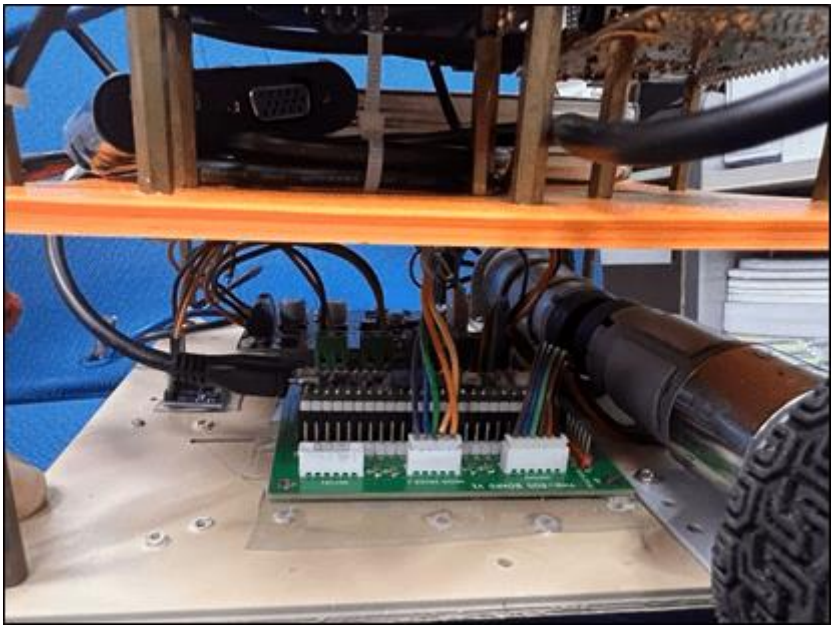


Figure 7: Connection of YHN-ROS Board to Cytron motor driver and encoder.

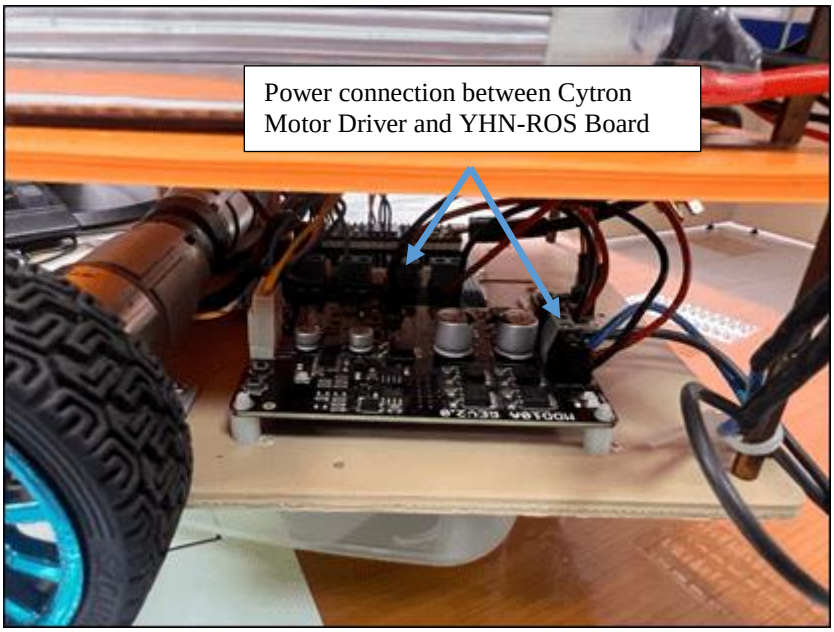


Figure 8: Power connection between Cytron Motor Driver and YHN-ROS Board.

3.2.7 Robot Operating System 2 (ROS2)

The Robot Operating System 2 (ROS2) is a next-generation framework that builds upon the original ROS1 and offers notable enhancements in scalability, communication and reliability. While

ROS1 established the groundwork for the development of robot software, ROS2 was created to address the shortcomings of its predecessor and satisfy the demands of contemporary, intricate robotic systems. DDS (Data Distribution System), which enhances data communication between nodes, is one of the main improvements in ROS2 (Phueakthong & Varagul, 2021). Quality of Service (QoS) settings in DDS give users more control over how data is transmitted and received. These parameters are crucial for controlling communication priorities like low latency and dependability, particularly in real-time applications (Ye et al., 2023).

Additionally, ROS2 provides Intra-Process Communication, which speeds up data exchange between nodes in the same process. ROS2 is independent of a central ROS master, in contrast to ROS1. This makes distributed systems more flexible and fault tolerant (Ye et al., 2023). By utilizing Node Recovery features, the system can also recover from node failures more successfully. Furthermore, ROS2 facilitates multi-threaded execution and real-time capabilities, enabling the effective parallel execution of complex tasks (Ye et al., 2023). The implementation of SROS2, which ensures secure communication between nodes, has further enhanced security. Various types of hardware, including LiDAR, cameras, IMUs, encoders and motor controllers, can be integrated with the free and open-source ROS2 framework. It contains modules such as slam toolbox for mapping, nav2_amcl for localization, ros2_control for hardware interface management and nav2_bringup for autonomous navigation.

In this study, ROS2 Humble was set up on a Raspberry Pi 4 running Ubuntu 22.04 and a PC. It demonstrated robust compatibility and consistent performance. All things considered, ROS2 offers an effective, contemporary framework for creating dependable, perceptive autonomous robotic systems.

3.2.8 Micro-ROS (ROS for microcontrollers)

Developers can delegate basic yet essential tasks, like sensor data collection, actuator control and hardware-level communication, to microcontrollers by incorporating Micro-ROS into a robotic system. This enables more potent systems, like the Raspberry Pi 4, to handle more computationally demanding tasks, like mapping and navigation. The Teensy 4.1 microcontroller running FreeRTOS, a real-time operating system designed for embedded systems, has Micro-ROS installed in the example (Phueakthong & Varagul, 2021). Together with FreeRTOS, the Micro-ROS application is uploaded, enabling the microcontroller to reliably perform real-time tasks. A serial connection to a Micro-ROS Agent running on the Raspberry Pi 4 allows communication between the microcontroller and the main robot system. By translating communications between the microcontroller and the ROS2 nodes in the main system, this agent serves as a bridge. The microcontroller thus becomes a fully functional part of the ROS2 network, able to publish and subscribe to topics in the same way as any other ROS2 node.

3.2.9 Mapping

A key process in autonomous robotics is mapping, in which a robot uses information from multiple sensors, including LiDAR, IMU, wheel encoders and RGB-D cameras, to generate a map of its environment. This map serves as a static reference that the robot uses for autonomous navigation and path planning, enabling it to identify its surroundings and make intelligent decisions about movement, obstacle avoidance and localization.

The Simultaneous Localization and Mapping (SLAM) technique is employed in this robotic system to calculate the robot's current position on the map and carry out mapping. For this purpose, slam_toolbox is a common ROS2 package that effectively integrates sensor data to create dependable and accurate maps (Macenski & Jambrecic, 2021). The robot is manually moved throughout the environment to gather data during the mapping phase. The IMU logs orientation and acceleration, the encoder tracks wheel rotation and movement and the LiDAR measures the distance to objects in the vicinity. A real-time cloud point visualization of walls and obstacles is produced by SLAM using these combined inputs.

LiDAR data is usually published to the /scan topic in the ROS2 ecosystem. In order to create a 2D representation of the environment in the visualization tool Rviz2, the SLAM package subscribes to this topic and processes the laser scan data. A dynamic "cloud point" that represents the locations of walls, furniture, and other stationary objects in relation to the robot is the result.

The final map is saved in two essential files, a .pgmfile and a .yaml file, after the environment has been thoroughly explored and mapped. While the .yaml file contains crucial metadata like the map's resolution, origin coordinates and physical dimensions, the .pgm file is a grayscale image that represents the map. The map is then used by packages like nav2 for path planning and localization during the robot's autonomous navigation phase.

3.2.10 AMCL and Navigation

An essential algorithm in ROS2 for estimating a robot's location on a known map is called AMCL (Adaptive Monte Carlo Localization). It is essential for autonomous navigation, which typically consists of three primary phases: motion control, path planning and localization.

The first step is to use AMCL to find the robot's starting position on the navigation map (Phueakthong & Varagul, 2021). By comparing LiDAR scan data and odometry readings with the known map, this algorithm uses a particle filter to estimate the robot's position. AMCL determines the robot's most likely location by evaluating how well the sensed environment matches the map. For effective path planning and successful navigation, precise localization is crucial at this point.

Path planning, which is separated into local and global planning, is the second step. The Global Planner uses algorithms like A* and Dijkstra to determine the best route from the robot's current

location to the desired position using the NavFn plugin (Phueakthong & Varagul, 2021). This planner does not account for dynamic elements, but it does account for static obstacles like walls and furniture. On the other hand, the Local Planner modifies the route in real time according to the current environment. It assists the robot in avoiding unexpected objects or moving obstacles like people. These modifications are successfully made for this robot using the DWB (Dynamic Window Approach) Local Planner.

Motion control is the last phase. During this stage, the /cmd_vel topic is used to send linear and angular velocity commands to the robot. The robot is guided along the predetermined route by these commands. The robot can navigate safely and independently in dynamic environments by coordinating these three stages: motion execution, dual-layer path planning, and localization with AMCL.

3.2.11 RViz2

RViz2 is essential for visualizing and interpreting the robot's surroundings, sensor data, and system states in any robotics project involving ROS2. RViz2 is a crucial tool for tracking and troubleshooting the robot's mapping and navigation operations in this autonomous robot project. It gives the developer a clear and interactive visualization of the robot's movements and environment by enabling real-time observation of the data collected by the robot's sensors and navigation algorithms (Macenski et al., 2022; Phueakthong & Varagul, 2021). In order to display a variety of sensor data and robotic states through visualizations, RViz2 is made to integrate seamlessly with ROS2. RViz2 is mainly utilized in this particular project to visualize the robot's SLAM (Simultaneous Localization and Mapping) process, which is crucial for autonomous navigation. Using sensors like the LiDAR, IMU, and encoders, the robot collects information about its surroundings and position as it navigates its environment. The robot's internal systems then process this data, and RViz2 shows the robot's position, the map it created, and its path.

The ability of RViz2 to display LaserScan data from the RPLiDAR sensor is one of its main characteristics in this study. By sending out laser beams and timing how long it takes for them to return, this sensor gives the robot comprehensive information about its environment. Real-time obstacles and open space surrounding the robot are depicted on a 2D map in RViz2. This aids the developer in making sure the robot is accurately assessing its surroundings and choosing the best course of action.

Furthermore, RViz2 is a great tool for visualizing the odometry data from the IMU sensor and the robot's encoders. Developers can monitor the robot's movements and confirm the precision of its localization and navigation algorithms by using RViz2 to plot the robot's position on a map. In order to help with debugging and system optimization, it can also display TF (transform) data, which illustrates the robot's orientation and position in the environment in relation to various frames of reference.

Additionally, RViz2 offers tools for visualizing robot states, including orientation, speed, and sensor readings, enabling developers to evaluate the proper operation of the robot's control systems.

Additionally, it features interactive markers that let users manually program the robot with objectives or points of interest, making it easier to test and simulate various scenarios.

In this autonomous robot study, RViz2 is an essential tool that gives developers a robust, real-time interface to track robot movements, monitor sensor data, and troubleshoot mapping and navigation algorithms. A crucial component of the project's development and testing stages, RViz2's visualizations of SLAM, odometry and LiDAR data guarantee the robot's accurate and efficient operation. Example of RViz2 mapping visualisation is depicted in Figure 9.

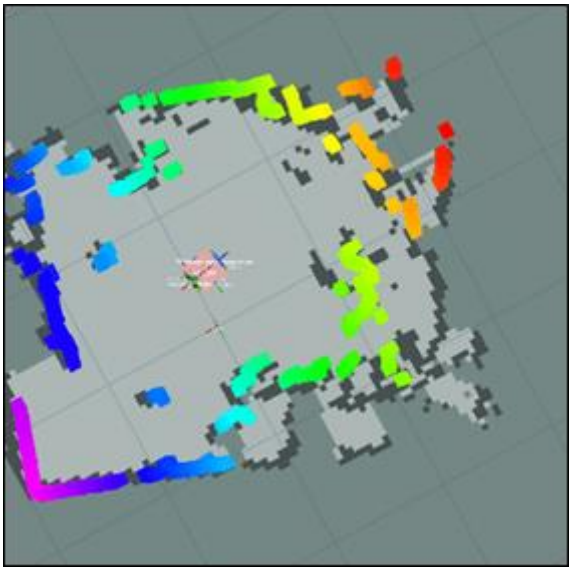


Figure 9: Visualisation of RViz2 during the mapping procedure.

4.0 RESULTS AND ANALYSIS

Two important tests, an autonomous differential drive robot was created for autonomous navigation and map creation. With the aid of sensors such as LiDAR and an IMU, the robot uses SLAM to create a map of its surroundings. The robot used odometry and sensor fusion to navigate the lab on its own, avoiding obstacles and following a path. The tests demonstrated the robot's potential for use in autonomous exploration and indoor navigation by confirming its ability to map and navigate within a controlled environment.

4.1 Navigation Map

The robot was manually teleoperated around its surroundings using a PC running Ubuntu 22.04 in order to create the navigation map. The mapping procedure was completed with the slam_toolbox package, which created a real-time occupancy grid map of the surrounding area using LiDAR and odometry data as input. The result was a navigation map with a 78 × 34 pixel resolution that accurately

and clearly depicted the layout of the surroundings. Later, the robot's autonomous navigation tasks were built upon this map as their foundation. The slam_toolbox package, which was essential to the SLAM and navigation procedures' success, is used to create the navigation map in Figure 10.

Figure 10: Navigation Map.

4.2 Navigation Test

The two components of the navigation experiment were the static map with dynamic obstacles in the environment and the static map with only static objects. While navigating, sensor data was visualized using RViz2. To illustrate obstacles, LiDAR data was shown as a point cloud, and the robot's position and orientation were indicated by IMU and encoder data published through the /odom topic. The 2D Pose Estimate tool in RViz2 was used to localize the robot before it could start navigating. An initial estimate of the robot's location and orientation on the map was given in this step. The robot's position was then continuously adjusted by the AMCL algorithm in response to sensor input. As seen in Figure 11, precise path planning and steady robot movement depend on accurate localization.

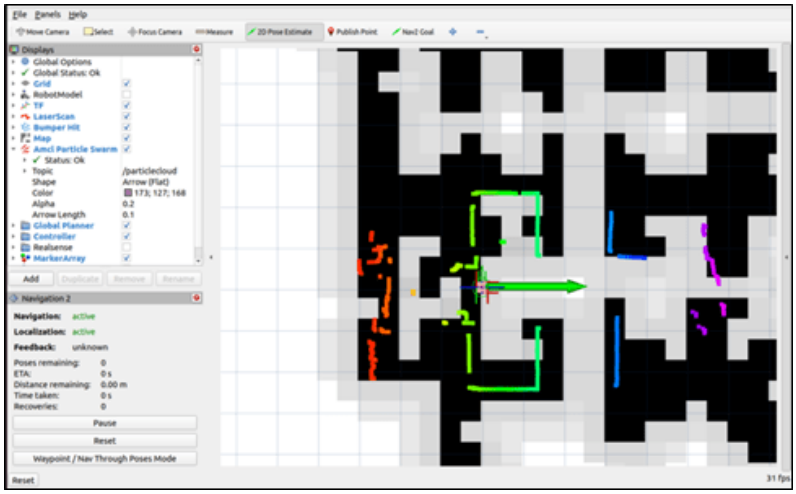
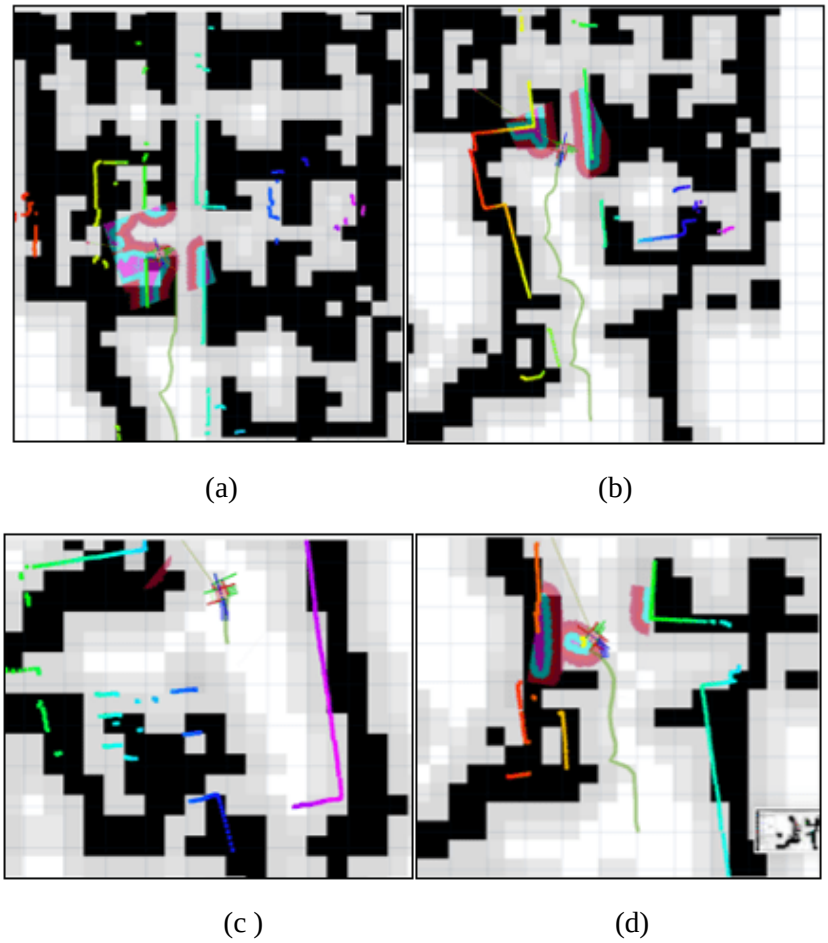


Figure 11: Robot localization process using 2D Pose Estimate feature in RViz2.

RViz2 showed both the Global Costmap and the Local Costmap after Navigation2 (Nav2) was turned on. The local map was 3 m x 3 m in size. Every navigation trial began at the same starting point and traveled to the same destination. The robot followed the global path that was created in order to reach the objective (Figure 12(b)). When a dynamic object came into the robot's path while it was moving, the Local Planner recalculated a new route to avoid it (Figure 12(d)). As configured in the controller_server, the robot's goal tolerance was set to 0.25 meters (xy_goal_tolerance) and 0.25 radians (yaw_goal_tolerance).



(a) (b)

(c) (d)

Figure 12 (a) Navigation map view with Global and Local Costmap displayed in RViz2. **(b)** Robot following the global path toward the goal location. **(c)** Robot approaching the destination point. **(d)** Local Planner generating a new path to avoid a dynamic obstacle.

Table 1

Navigation using the static map (no dynamic objects).

Experiments	Robot Location Error		Time(s)
	x(cm)	y(cm)	
1	-4.3	4.5	97
2	2.5	-2.6	101
3	-11.5	-11.8	98
4	3.7	-4.9	105
5	-2.9	5.5	96
Average	-2.5	-1.86	99.4

Table 2

Navigation with dynamic objects present.

Experiments	Robot Location Error		Time(s)
	x(cm)	y(cm)	
1	-3.6	-8.5	115
2	-5.8	-6.5	120
3	-6.4	-10.7	119
4	-2.8	-12.5	123
5	-4.9	3.6	118
Average	-7.8	-6.92	119

The robot's positional error and travel time were recorded in two separate tables:

i. Table 1: Navigation using the static map (no dynamic objects)
ii. Table 2: Navigation with dynamic objects present

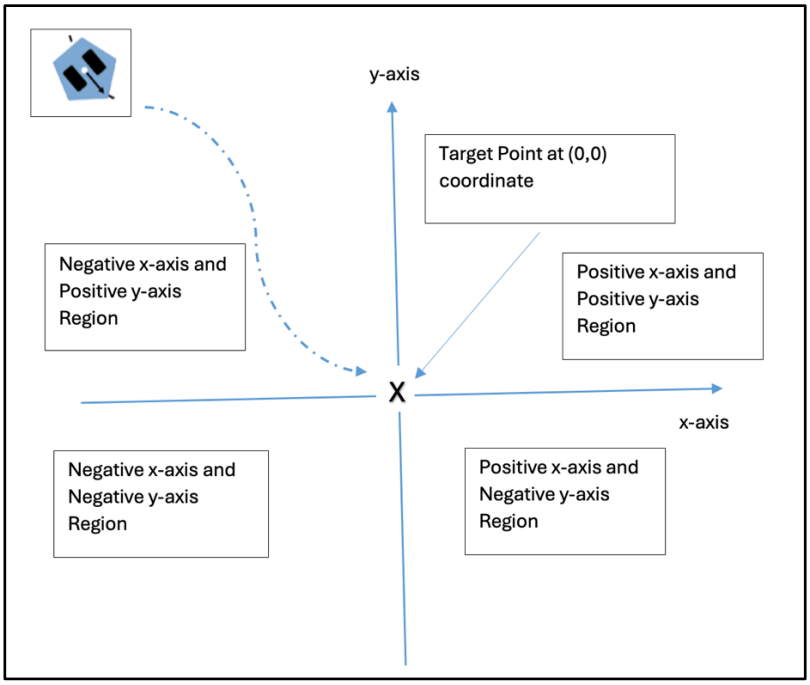


Figure 13: Reference Frame for Robot Positional Error in Cartesian Coordinates.

The coordinate system used to calculate the robot's positional error with respect to a target point at the origin (0,0) is depicted in the Figure 13. The x and y axes are used to divide the coordinate plane into four quadrants:

i. The upper-right quadrant represents the region where both x and y values are positive.
ii. The upper-left quadrant represents the region where the x values are negative and y values are positive.
iii. The lower-left quadrant is where both x and y values are negative.
iv. The lower-right quadrant contains positive x values and negative y values.

The navigation system of the robot can calculate its error in position (x and y) with respect to the objective using this coordinate-based method, which is crucial for path planning and corrective motion. Accurate average performance data was obtained by repeating each test scenario five times. The findings demonstrated that the robot successfully and accident-free arrived at its destination on its own in every trial. The robot was able to recognize dynamic objects even though they weren't on the original map and re-plan its route using the Local Costmap. The robot maintained positional accuracy even though the presence of dynamic obstacles slightly extended the travel time. The acceptable goal tolerance set up in Nav2 was maintained by the maximum recorded position error of 12.5 cm.

5.0 CONCLUSIONS

The experimental findings show that the autonomous differential drive robot used the ROS2 Navigation2 framework to successfully complete both mapping and navigation tasks. The robot used

slam_toolbox to create a usable 2D map, AMCL to precisely localize itself and autonomously navigated to its destinations while dodging both static and dynamic obstacles. The efficiency of integrating LiDAR, odometry and IMU data in a ROS2 ecosystem for autonomous indoor robotics is confirmed by this result. Future enhancements might incorporate vision-based detection, enhance obstacle prediction or optimize the local planner for congested areas.

REFERENCES

Firmansyah, R. A., Prabowo, Y. A., Suheta, T., & Utomo, A. N. D. (2024). Implementation of SLAM Gmapping and Extended Kalman Filter for Security Robot Navigation System. *Emitor: Jurnal Teknik Elektro*, 145–153. https://doi.org/10.23917/emitor.v24i2.3104

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. In *Science Robotics* (Vol. 7, Issue 66). American Association for the Advancement of Science. https://doi.org/10.1126/scirobotics.abm6074

Macenski, S., & Jambrecic, I. (2021). SLAM Toolbox: SLAM for the dynamic world. *Journal of Open Source Software*, *6*(61), 2783. https://doi.org/10.21105/joss.02783

Mochurad, L., Hladun, Y., & Tkachenko, R. (2023). An Obstacle-Finding Approach for Autonomous Mobile Robots Using 2D LiDAR Data. *Big Data and Cognitive Computing*, *7*(1). https://doi.org/10.3390/bdcc7010043

Moore, T., & Stouch, D. (n.d.). *A Generalized Extended Kalman Filter Implementation for the Robot Operating System*. http://www.cra.com/robot_localization_ias13.zip

Phueakthong, P., & Varagul, J. (2021). A Development of Mobile Robot Based on ROS2 for Navigation Application. *International Electronics Symposium 2021: Wireless Technologies and Intelligent Systems for Better Human Lives, IES 2021 - Proceedings*, 517–520. https://doi.org/10.1109/IES53407.2021.9593984

Song, M., & Zhang, Y. (2024). *Industrial Mobile Robot Navigation Based on the LiDAR-IMU-Visual Measurements*. https://doi.org/10.21203/rs.3.rs-5391723/v1

Yan, Y., Zhang, B., Zhou, J., Zhang, Y., & Liu, X. (2022). Real-Time Localization and Mapping Utilizing Multi-Sensor Fusion and Visual–IMU–Wheel Odometry for Agricultural Robots in Unstructured, Dynamic and GPS-Denied Greenhouse Environments. *Agronomy*, *12*(8). https://doi.org/10.3390/agronomy12081740

Ye, Y., Nie, Z., Liu, X., Xie, F., Li, Z., & Li, P. (2023). ROS2 Real-time Performance Optimization and Evaluation. *Chinese Journal of Mechanical Engineering (English Edition)*, *36*(1). https://doi.org/10.1186/s10033-023-00976-5