https://doi.org/10.5281/zenodo.18541652

SLAM-Based Indoor Navigation with LiDAR and ROS2 for Autonomous Mobile Robot

D. Y. Salihu[1], A. A. Olawole[1*], A. O. Aransiola[1], A. M. Jubril[1], and A. Fisusi[1]
[1]Department of Electronic and Electrical Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria.

ABSTRACT

While robots have been able to provide insights into solutions to the inefficiency, safety, and adaptability of operational and logistic related engagements, sensor reliability, accuracy, and navigation in dynamic environments are some of the challenges that have limited their performance. This paper focuses on developing an Autonomous Mobile Robot (AMR) that can enhance logistics and operational efficiency. By utilizing the Robot Operating System (ROS) along with key components like Light Detection and Ranging (LiDAR), Raspberry Pi, and microcontrollers, the robot is developed to autonomously navigate complex environments. The performance of the developed robot is studied in terms of its ability to achieve 360 degrees of view and avoid obstacles through successive mapping and navigation in an arbitrary environment. In simulation and limited real-world trials, the system achieved reliable indoor navigation; we report configuration details and reproducibility artifacts.

KEYWORDS

Autonomous mobile robot
Light Detection and Ranging
Raspberry Pi
Robot Operating System
Simultaneous Localization and Mapping (SLAM)

1. INTRODUCTION

An autonomous mobile robot (AMR) is a robotic system designed to comprehend and navigate its environment independently. Distinct from their predecessors, autonomous guided vehicles (AGVs), which follow tracks or predefined paths and often require operator supervision, AMRs operate with greater autonomy. Historically, AMRs found application in industrial or warehouse settings, enhancing efficiency and productivity. In the modern era, robots are gradually becoming integral to our daily lives, evident in the advent of autonomous cars and autonomous drones for package delivery. The essence of an AMR lies in its ability to navigate and operate within an environment without direct human supervision. While significant progress has been made in localization, mapping, and autonomous exploration, localization remains a critical challenge due to inherent sensor limitations, dynamic environmental conditions, and computational constraints (Cadena et al., 2016). For instance, behavior-based subsystems (Sheu et al., 2010) and robots following predefined paths (Madhankumar et al., 2021) effectively manage short-distance navigation but often lack the adaptability for complex or changing environments. The insights from (Ess et al., 2010) further underscore the limitations of single-sensor-equipped robots, emphasizing the viability of integrating multiple sensors for enhanced precision.

The majority of existing work on AMR navigation often falls into two primary categories, each with distinct limitations that our research addresses. Firstly, "line-following" methods, commonly employing Infrared (IR) sensors (Hamid et al., 2019; Panda et al., 2020; Srilekha et al., 2018; Yanmida et al., 2020; Mane et al., 2015), rely heavily on predefined continuous lines or fixed paths for navigation. While effective in highly structured settings for tasks like line tracking or object detection (Hamid et al., 2019; Panda et al., 2020; Srilekha et al., 2018; Yanmida et al., 2020), this approach inherently leads to limited flexibility. If a line is interrupted, obscured by dirt, dust, or damage, the robot can lose its way or struggle to navigate accurately. Similarly, systems relying only on wheel encoders and fixed distances (Mane et al., 2015) face significant challenges in dynamically changing environments, constraining their real-world applicability. Even attempts to enhance IR-based systems with SLAM (Kasun et al., 2011) or by integrating other technologies (Shiny et al., 2017) highlight the persistent complexities in achieving robust performance across varied conditions.

Secondly, camera-base autonomous robots (Shaker et al., 2008; Nikam et al., 2017; Sahu, et al., 2019) integrate computer vision and IR sensors for target identification and obstacle detection, respectively. While capable of guiding vehicles and processing visual cues (Sahu, et al., 2019), these systems are fundamentally susceptible to environmental factors such as lighting variations and visual occlusions. They also typically demand significant computational resources for real-time image processing, often compromising consistent and precise localization, particularly in complex or low-light scenarios. Building upon the insights and limitations identified in these existing approaches, our work introduces a novel autonomous mobile robot system. Our core objective centers on building an AMR by integrating advanced technologies: LiDAR and Robot Operating System 2 (ROS2). This integration provides a fundamentally more robust and flexible navigation solution:

- Unlike camera-based systems, which can be hampered by lighting and visual complexity, our proposed method leverages LiDAR for superior localization accuracy. LiDAR offers real-time 360-degree mapping, enabling the robot to precisely perceive intricate spaces and avoid obstacles with exceptional clarity, regardless of ambient light.
- Crucially, unlike the "line-following" methods that are confined to predefined paths and offer limited flexibility, our implementation is based on Simultaneous Localization and Mapping (SLAM). This allows the robot to dynamically generate and reference maps of its surroundings, providing greater adaptability and flexibility to navigate effectively in any dynamic environment without prior, fixed routes.

This combination of LiDAR for precise environmental sensing and ROS2 for robust system integration allows our developed robot to reliably determine its position and navigate with high autonomy. ROS2 further enhances our system's capabilities through its features such as program re-usability, modularization, robust development tools, and a supportive community (Morgan et al., 2009; Galtarossa, 2018). Table 1 is a summary of the reviewed literature.

The remaining sections of this paper are organized as follows. Section 2 describes the design of the proposed autonomous mobile robot. Simulation and Experiment Results are presented and explained in Section 3. Finally, the conclusion of the paper is provided in Section 4.

2. THE PROPOSED AUTONOMOUS MOBILE ROBOT DESIGN

The proposed autonomous robot system consists of a computer system serving as a base station (BS) and a custom-developed

Corresponding Author: A. A. Olawole (alex_olawole@oauife.edu.ng)
The editor responsible for coordinating the review of this article and approving its publication was E. Obayiuwana.
P-ISSN: 1115-9782 **e-ISSN:**

Table 1: Literature Review Summary

Paper	LF	IR (LT)	IR (OD)	U/S	W/E	Web camera	Pi camera	LiDAR	Nav2	ROS2	Environnent
Shaker *et al.*, 2008				+		+					Structured
Kasun *et al.*, 2011	+		+			+					Structured
Mane *et al.*, 2015 Shiny *et al.*, 2017	+		+		+						Structured
Nikam *et al.*, 2017; Sahu *et al.*, 2019							+				Unstructured
Hamid *et al.*, 2019; Panda *et al.*, 2020; Srilekha *et al.*, 2018	+	+	+								Structured
Yanmida *et al.*, 2020	+	+		+							Structured
This paper								+	+	+	Unstructured

LF - Line following method; IR(LT): Infrared sensor based technology for line tracking; IR(OD): Infrared sensor based technology for obstacle avoidance; U/S - Ultrasonic sensor based technology for obstacle avoidance; W/E: wheel encoder for navigation; LiDAR: Light Detection and Ranging; Nav2: for navigation, path planning, and obstacle avoidance; ROS2: Robot Operating System for mapping, localization, and navigation

mobile robot system, connected via a wireless WiFi network. The Robot Operating System (ROS) enables seamless communication and data exchange between multiple processes running on the base station and the robot. A block diagram of the mobile robot system is illustrated in Figure 1. The system's design is tailored for indoor environments with flat or gently sloping surfaces, without steps or stairs,_although it can accommodate minor floor irregularities.

2.1 Autonomous Mobile Robot Design

The custom-developed robot system consists of the central high-level control unit, the motor control unit, which is composed of the motors, motor drivers, motor wheel encoder for feedback, and the medium-level motor controller, and the LiDAR unit, which is composed of the LiDAR sensors and their drivers.

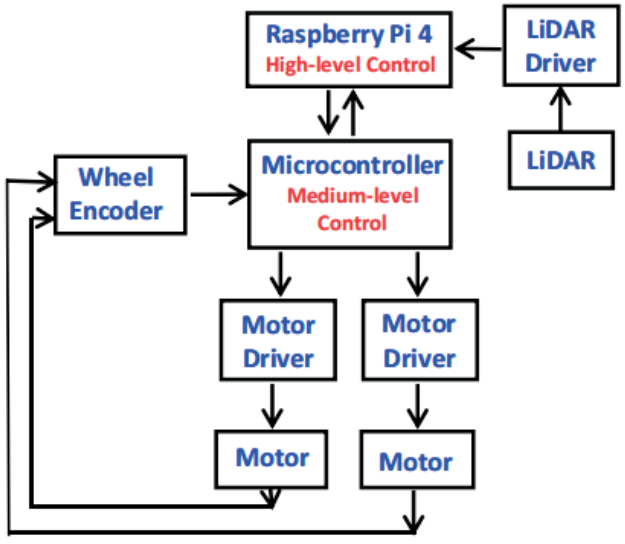


Figure 1 Block Diagram of the Autonomous Mobile Robot

Integrating the hardware (Figure 2) and software in the robot system is crucial for achieving precise movement, accurate navigation, and effective interaction with the environment.

This system combines several components, such as Arduino Nano, a Raspberry Pi, LiDAR sensors, and motor drivers, all coordinated by the software running on ROS2. The LiDAR SLAM is employed in this work for its longer range (up to 100 meters), its ability to work well even in darkness, and its high accuracy and robustness in dynamic environments (unlike the visual SLAM). It uses scan matching algorithms to align successive laser scans and estimate robot motion.

While higher resolutions capture finer details, it requires down-sampling techniques for real-time performance Together, these elements enable accurate localization and 2D mapping in the designed autonomous robot. The choice of the 2D mapping is made for its efficiency in flat, structured environments with relatively affordable 2D LiDAR, while offering low computational cost. The LiDAR SLAM configuration parameter is shown in Table 2.

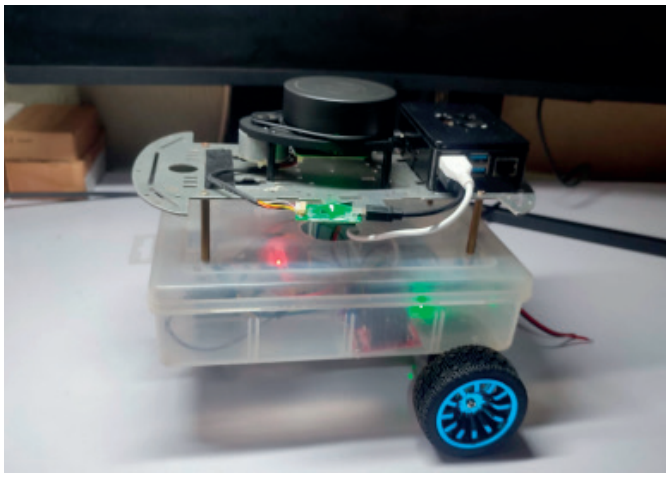


Figure 2 Hardware Setup of Robot

2.2 Motor Control System

At the heart of the motor control system is the Arduino Nano, which uses a Proportional, Integral, Derivative (PID) controller to manage the speed and position of the robot's wheels. The Arduino Nano interfaces with the motor encoders through its general-purpose input/output (GPIO) pins and uses interrupts to capture the pulses from the magnetic encoders. These pulses provide real-time feedback to the PID control loop, enabling continuous adjustments to motor outputs to maintain accurate and stable motion. The motor driver translates the Pulse Width Modulated (PWM) signals from the Arduino Nano into the required voltage and current to drive the 12V DC gear motors. The encoders on these motors provide crucial feedback, which the PID loop uses to adjust the speed and position as needed.

For motor control, the system uses the ROS2 control framework with a differential drive controller to handle linear and angular velocity commands, translating them into wheel velocities for the motors. This configuration allows the robot to perform precise movements, such as turning or navigating tight spaces. A *teleop_twist_keyboard* node is also integrated, providing manual control via keyboard inputs, which is useful for testing and calibration.

Table 2: LiDAR SLAM Configuration Parameter

Parameter	Value
Soler_plugin	CeresSolver
resolution	0.05 m/cell
max_laser_range	20.0 m
map_update_interval	5.0 s
transform_publish_period	0.02 s
use_scan_matching	true
do_loop_closing	true
loop_search_maximum_distance	3.0 m
scan_buffer_size	10 scans
minimum_travel_distance	0.5 m
loop_match_minimum_response_fine	0.45

2.2.1 PID Controller for Wheel Velocity

The control system was implemented using the ros_arduino_bridge package, which provides a serial interface between the ROS2 and an Arduino-based differential drive robot. The included PID controller was used to manage wheel velocities based on encoder feedback. The goal of the controller is to match the desired wheel velocities (sent from ROS2) with actual velocities measured via encoders. The control loop runs at 30 Hz and uses a standard discrete-time PID structure. At each iteration, the number of encoder ticks since the last update is compared to the expected number of ticks (based on the desired velocity), and the error is used to compute a new PWM value. The control output is calculated as:

$$\text{Output} = \frac{K_p \times e - K_d \, \Delta u + I_t}{K_o} \qquad (1)$$

where e is the velocity error (in encoder ticks per frame), and Δu is the change in measured velocity used as a substitute for the derivative term. The integral term, I_t, is only updated when the output is not saturated, to avoid windup. The parameters K_p, K_d, and K_i are the proportional, derivative and integral gains, respectively, and K_o is the output scaling factor.

2.2.2 Tuning

The gains were tuned manually using a basic step test, adjusting for responsiveness and stability. The process started with increasing the proportional gain until the response became fast but not overly aggressive. A small derivative term was added to smooth out the motion and reduce overshoot. No integral gain was used (i.e., $K_i = 0$), as the system already tracked velocity well enough without it, and the risk of windup outweighed the potential benefits. The values of PID parameters K_p, K_i, K_d, and K_o are 20, 0, 12, and 50, respectively. This tuning provided a stable and responsive system with low tracking error. The system behaved well across all tested speeds and surfaces. The use of encoder feedback makes the control loop robust to minor disturbances like surface friction or small obstacles. An auto-stop timer ensures that the motors are disabled if no command is received for a certain time, adding a layer of safety.

2.3 Localization and Mapping

The mapping of the environment is performed by remotely driving the robot using the SLAM algorithm (i.e., the LiDAR SLAM) from the base station. The SLAM algorithm enables the robot to navigate and map unknown indoor environments by integrating sensor data, feature extraction, and probabilistic estimation. The Raspberry Pi runs the *rplidar_node* that interfaces with a LiDAR sensor to obtain distance measurements by emitting laser pulses (in order to evaluate the time-of-flight (TOF)), detecting their reflections and, publishing the LiDAR data on the scan topic. The LiDAR data is essential for obstacle detection, mapping, and path planning. The SLAM Toolbox on the Raspberry Pi helps the robot to build a real-time map of its environment while determining its position and orientation. The combination of SLAM data and encoder feedback as shown in Figure 3 ensures robust localization and precise navigation.

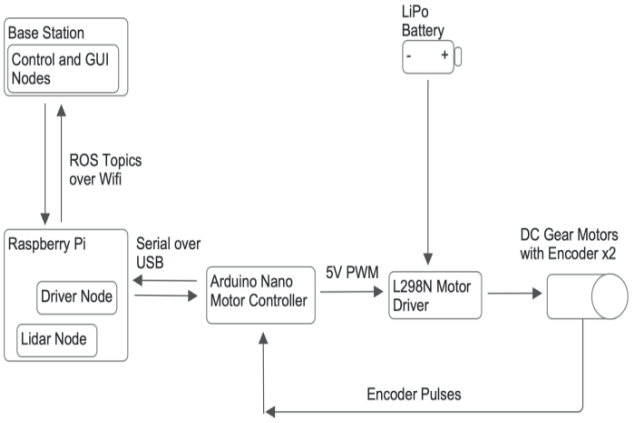


Figure 3 Hardware and Communication Layout for the Autonomous Mobile Robot

The *slam_toolbox* integrates LiDAR data with odometry from wheel encoders for localization and map building. While no explicit sensor fusion algorithms like the Extended Kalman filter (EKF) were manually implemented, the fusion and drift correction are managed through the SLAM process and Nav2 stack. The default filters in ROS2 automatically handle minimal drift correction, and loop closure helps to further correct any accumulated errors as the robot revisits previously mapped areas. This approach eliminates the need for custom sensor fusion code while ensuring accurate and stable localization during operation.

The Nav2 framework from ROS2 and SLAM provide complementary technologies. While SLAM focuses on mapping and localization, Nav2 provides a navigation and path planning framework that enables the robot to navigate complex environments effectively. The integration of SLAM with Nav2, therefore, enables the robot to navigate and map its environment simultaneously, allowing it to adapt to changing environments, avoid obstacles, and reach its goals with precision, even in unknown spaces. The robot's navigation was managed using the Nav2 stack in ROS 2, leveraging the bt_navigator for behavior tree-based planning. The global planner used was the default NavfnPlanner, and the local planner was the default Dynamic Window Approach (DWB) controller (nav2_controller) for local path following. The navigation stack used the default Nav2 costmap configuration, including resolution and size. The inflation layer, which expands obstacle boundaries to maintain safe distances during path planning, was not manually configured. As a result, the default inflation radius of 0.70 meters and a cost scaling factor of 3.0 were applied. This produced the typical gra-

dient effect around obstacles, as observed in Robot Visualization (RViz). Although costmap YAML parameters were not explicitly set, the default Nav2 configuration provided safe and effective local and global planning behavior.

2.4 ROS2 Node Architecture

The software architecture of the autonomous mobile robot is based on a modular ROS2 system distributed across two devices: an onboard Raspberry Pi (mounted on the robot) and the base station - the remote computer system - connected over a wireless network. This separation allows computationally intensive processes to run on the more powerful processor on the base station, while lightweight sensor interfacing and motor control are handled locally on the robot.

On the Raspberry Pi, two primary nodes are deployed. The first is the *rplidar_node*, which interfaces directly with the 2D LiDAR sensor and publishes laser scan data over the */scan topic*. The second is the *ros_arduino_bridge*, which includes the *arduino_node* and *diff_drive_controller* components. These nodes handle low-level motion control by subscribing to the */cmd_vel* velocity commands and publishing odometry data to the */odom topic*. Wheel encoder feedback from the Arduino is used to estimate robot displacement, which is critical for both SLAM and navigation. Figure 4 shows the software architecture of the autonomous mobile system implemented in ROS2. On the computer system, all higher-level nodes are executed. This includes the *slam_toolbox* package, which operates in either SLAM mode (for map creation) or localization mode (when using a pre-built map). It subscribes to */scan* and */odom*, and publishes both the generated map and the map to odom transform required for localization. The navigation stack (nav2) is also hosted on the computer system and is responsible for global and local path planning, obstacle avoidance, and generating velocity commands published to the */cmd_vel topic*.

During autonomous operation, navigation goals are sent to nav2 through the */navigate_to_pose* action interface using RViz, where the user interacts with the GUI to specify 2D goal poses. For manual control, the *teleop_twist_keyboard* node is also run on the computer system. It is primarily used during the SLAM phase to manually drive the robot and explore the environment for map generation and can also be used at any point for direct keyboard-based control. To prevent command conflicts, only one source is allowed to publish to */cmd_vel* at a time, either nav2 or *teleop_twist_keyboard*, depending on whether the robot is operating autonomously or being manually controlled.

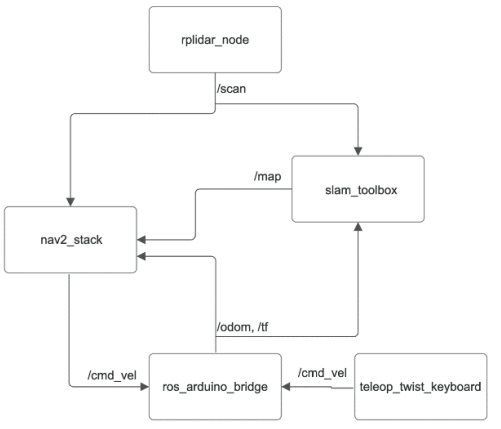


Figure 4 Software architecture of the autonomous mobile robot system implemented in ROS2, showing major nodes and topic communication.

All nodes communicate over the ROS2 data distribution service (DDS) middleware via a wireless network, enabling real-time topic exchange between the Raspberry Pi and the computer system. Although lifecycle management features are available in ROS2 (especially for nav2), this study uses a straightforward launch-based approach without explicitly invoking lifecycle transitions. Nodes are brought up using launch files or terminal commands and remain active for the duration of operation. This distributed architecture strikes a balance between efficient onboard control and centralized high-level processing, making it well-suited for real-time autonomous navigation in indoor environments. Shown in Algorithms 1 is a summary of the autonomous robot navigation system.

Algorithm 1: Autonomous Robot Navigation with SLAM

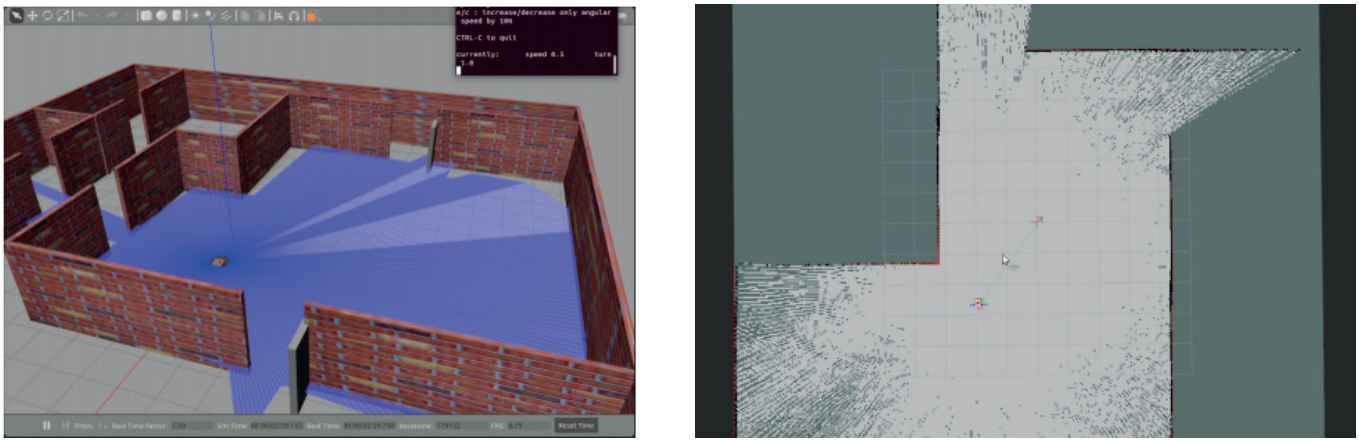
1. Begin
2. Initialize **ROS 2 environment**
3. Initialize **ROS 2 system** (Start sensor and hardware nodes)
4. Start rplidar_node to publish /scan
5. Start ros_arduino_bridge to publish **/odom,** subscribe to **/cmd_vel**
6. Wait until **/scan** and **/odom** are available
7. If map file is not available
8. **Start** slam_toolbox in SLAM mode
9. **While** mapping is active
10. Drive robot manually using *teleop_twist_keyboard*
11. **End while**
12. **Save map** to file
13. Else
14. Load map from file
15. **Start** slam_toolbox in localization mode
16. End if
17. Start nav2 stack
18. Wait until all nav2 nodes are active
19. Loop:
20. Set navigation goal pose
21. Send goal to */navigate_to_pose* action server
22. **While** goal not reached and no failure
23. nav2 computes path using **/map**, **/scan**, and **/odom**
24. nav2 publishes **/cmd_vel**
25. *ros_arduino_bridge* drives motors
26. Monitor **/odom** and **/tf** for feedback
27. **End while**
28. **If** goal is reached or path planning fails
29. Publish **zero velocity** to **/cmd_vel** (stop robot)
30. **End** if
31. **Repeat** or exit
32. End

3. SIMULATION AND EXPERIMENTAL RESULTS

This section presents our findings from the autonomous mobile robot's navigation tests conducted in both simulated and real-world environments. The robot receives goal position commands from the BS and autonomously navigates from its current position to the desired destination on the generated map.

3.1 Simulation Results

The robot was initially tested in a simulated environment using Gazebo to validate its navigation and obstacle-avoidance capabilities. The environment was designed to replicate a typical building. A model of a building was created using *Gazebo* to simulate a typical working environment of the robot (see Figure 5(a)). Figure 5(b) shows the generated map of the environment in RViz. Figure 6 shows the robot at its initial position, denoted



(a) (b)

Figure 5 Simulation Results: (a) Simulated environment (b) Generated map of the environment in Rviz

as 'A'. Its navigation path to the final position starts from a designated point within the simulated environment.

As illustrated in Figure 7, the robot can successfully navigate through the environment and reach the target location denoted as 'B' without any collision.

3.2 Real World Test Results

After the successful simulation test, the robot was tested in a real-world environment, as illustrated in Figures 8 and 9, to evaluate its performance. The mapping and navigation tests were conducted in a small university lecture room, as shown in Figure 8(a). The space was approximately 4.8 × 3 meters, with tiled flooring and several static obstacles such as wooden desks, chairs, and a cardboard box. The robot operated in this environment using LiDAR SLAM to generate an occupancy grid map (Figure 8b). Mapping was focused on the open section of the room, as the arrangement of desks and chairs made it difficult to safely navigate the robot through tighter areas. The generated map, therefore, covers only the accessible space used during navigation tests. The approximations in Table 3 describe the test setup, whereas Table 4 shows the hardware specifications.

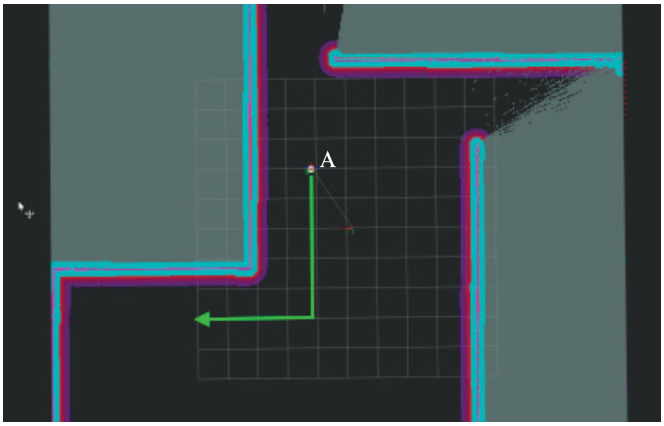


Figure 6 Simulation Results: Initial Position

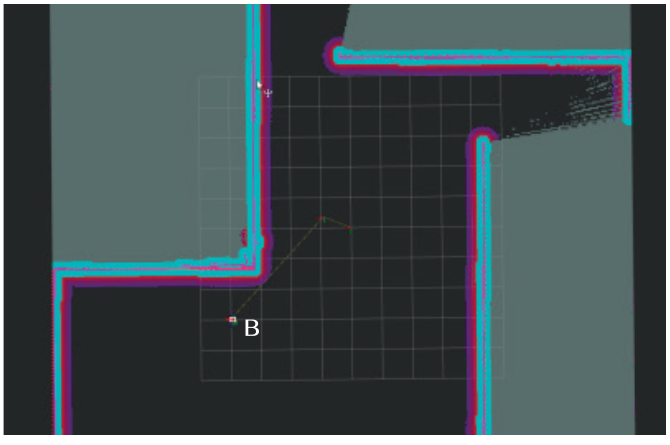


Figure 7 Simulation Results: Final Position

Table 3: Environmental Specifications

Parameter	Approximate Value
Room size	4.8 × 3 meters
Wall thickness	~0.25 meters
Robot footprint	0.3 × 0.3 meters
LIDAR range	6 meters (360° FOV)
Obstacle types	Static desks, chairs, a box

Table 4: Hardware Specifications

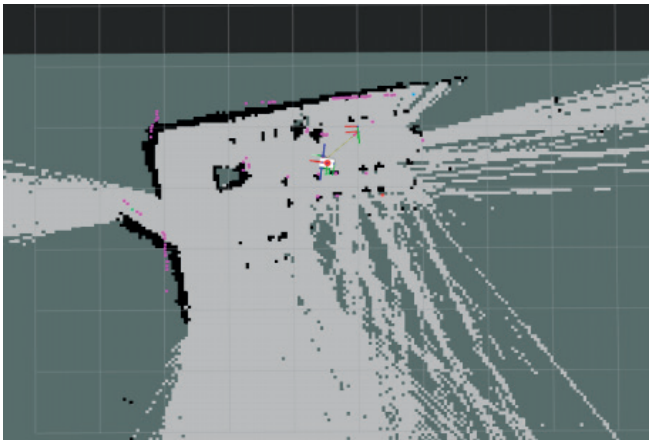
Hardware	Description
Raspberry Pi 4 - Powered via USB power bank	4GB RAM, Quad-Core ARM Cortex-A72
RP LiDAR A1-Powered via Pi USB	360° scan, 6m–12m range, 5.5Hz–10Hz
JGB37 - 520 Motor - Powered via a 12V, 1500mAh battery	12V DC, 110 RPM, 6mm shaft, Encoder: 11 pulses/rev, 7–15W
Arduino Nano powered via USB	ATmega328P, 5V logic
L298N Motor Driver - Powered from a LiPo battery	Dual H-bridge, supports 12V motors
LiPo battery (1500mAh)	
Computer System	CPU/RAM: Intel Core i5 vPro, 7th Generation, 16 GB RAM

Specifically, Figure 8(a) shows the robot in motion while Figure 8(b) illustrates the generated map of the environment. Figure 9(a) shows the robot at its starting position 'P' on the map, which corresponds to its position and orientation in the real-world environment. The robot autonomously navigates successfully to the designated endpoint denoted as 'R' through 'Q', around obstacles, demonstrating reliable navigation as shown in Figures 9(b) and 9(c).

Therefore, from the results obtained, it can be seen that the

robot demonstrates effective obstacle avoidance and reliable path-finding, and successfully navigates through the real-world environment to reach the target location. Comparing the proposed system with the IR-based method in (Ess *et. al.*, 2010, Panda *et. al.*, 2020, Srilekha *et. al.*, 2018, Mane *et. al.*, 2015, Shaker *et. al.*, 2008) which is simply dependent on a line-following and obstacle avoidance navigation system, obtained results in this paper show that the developed robot can provide a higher degree of view and navigate more complex environments using LiDAR-based Raspberry Pi for dynamic mapping and navigation in the SLAM algorithms.

(a)

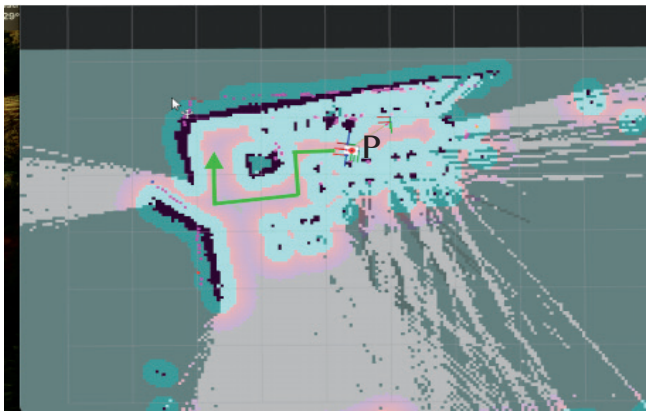


(b)

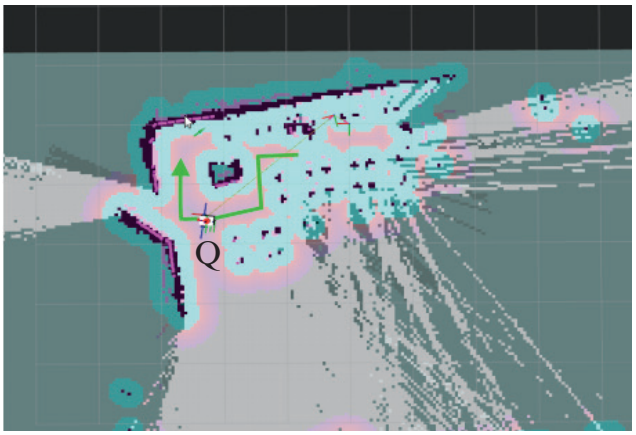
Figure 8 Real-World Test Results: (a) Robot in motion (b) Generated map of the environment

The dependency on physical lines or continuous external markers can thus be avoided, thereby providing a more suitable system without the need for physical cues. Moreover, when compared to the developed robot with the work in (Nikam *et al.*, 2017), where a camera-based SLAM was used for real-time obstacle detection via image processing, although both approaches successfully adjust navigation paths upon detecting obstacles, this paper integrates sensor data and mapping to better handle dynamic environments, providing a more robust solution than solely using image processing.

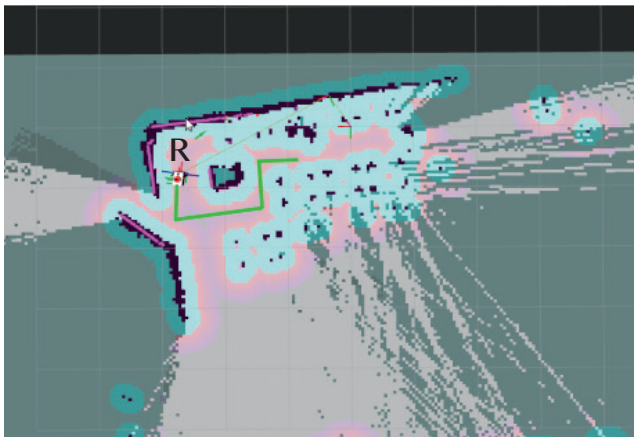
The bar chart in Figure 10 illustrates the measured angular drift under three conditions: open space, static obstacles, and a narrow corridor. Greater drift is observed in more constrained environments, indicating increased orientation error due to odometry limitations and restricted movement.



(a)



(b)



(c)

Figure 9 Test 1: RViz Showing Robot Positions in Map (a) Initial position, (b) Midway to destination, and (c) Final position

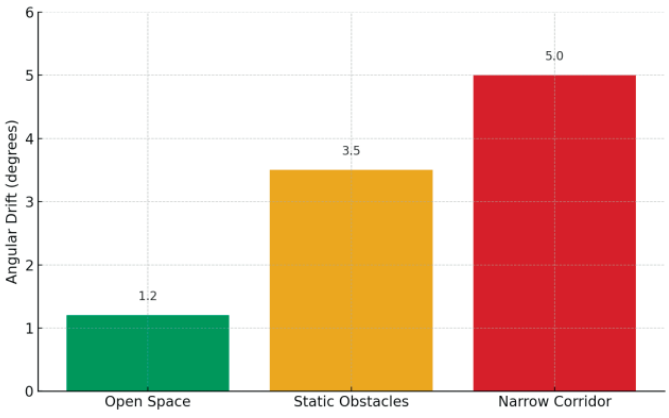


Figure 10 Angular Drift Across Navigation Scenarios

Furthermore, the estimates of the time required for the robot to reach goals at varying distances, based on a constant configured velocity of 0.1 m/s is as shown in Figure 11. As expected, the time to goal increases linearly with distance under ideal, obstacle-free conditions.

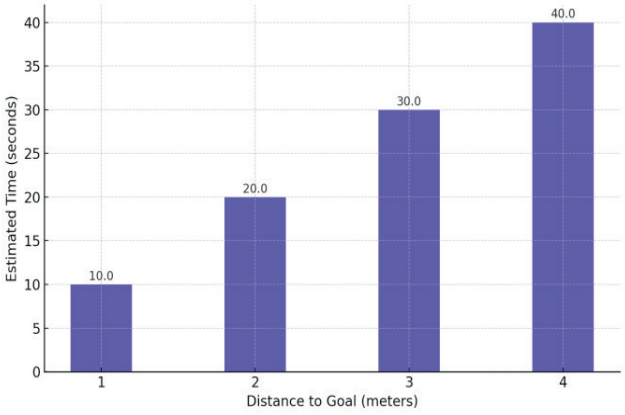


Figure 11 Estimated time to goal at different distances

The average time to goal with standard deviation is shown in Figure 12. As expected, scenarios with obstacles required longer completion times compared to obstacle-free runs. Among the obstacle-free cases, the corner navigation showed slightly faster average completion than the straight path.

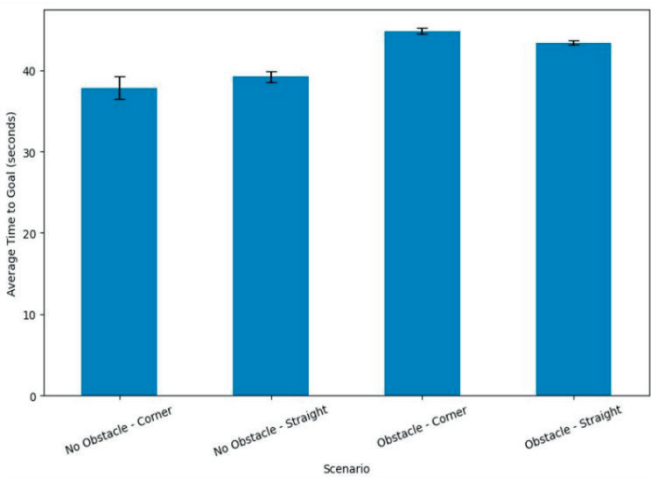


Figure 12 Average time to goal with standard deviation for all scenarios.

Individual trial results are shown in Figure 13. Most trials clustered tightly around their scenario averages, indicating consistent navigation behavior. However, occasional deviations were observed, such as a single low outlier in the *No Obstacle – Corner* case.

The statistical spread of times for each scenario is summarized in Figure 14. The box-plots highlight both variability and outliers across the scenarios. Obstacle-based runs consistently had higher median times and tighter clustering, while obstacle-free runs showed more spread due to occasional fast completions.

4. CONCLUSION

In this paper, a robot capable of autonomously navigating within a defined environment by receiving goal position commands from a base station and navigating from its current position to the desired location has been developed. The implementation has been carried out using a combination of hardware and software components, including a Raspberry Pi, a LiDAR sensor, and ROS2, for controlling and navigating the robot. The

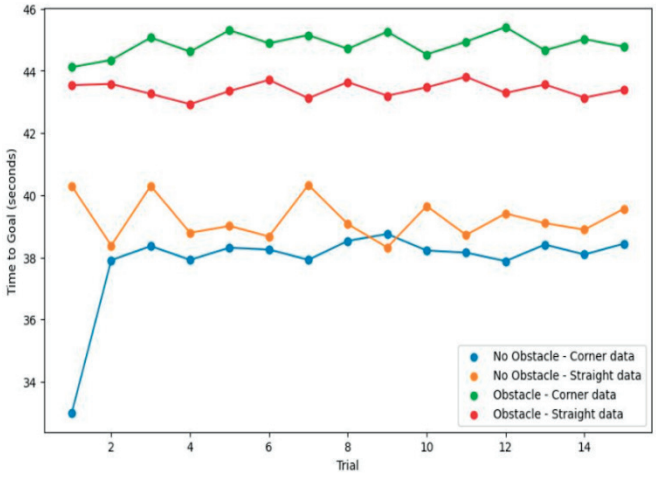


Figure 13 Time to goal across trials for each scenario.

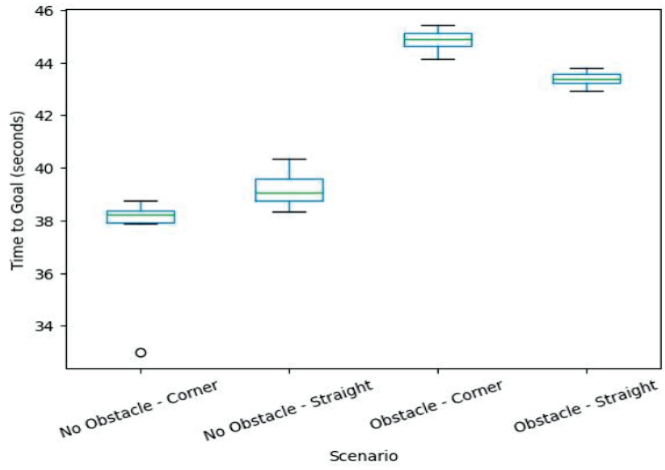


Figure 14 Distribution of Time to Goal by Scenario

robot can generate a map of its environment using LiDAR data and utilizes path-planning algorithms to navigate to a specified destination. The robot's navigation and mapping are tested in both simulated and real-world environments. The results show that the robot can successfully navigate its path in an arbitrary environment. The controller provides smooth motion and accurate speed tracking. The robot responds quickly to changes in velocity commands, and there's no noticeable oscillation or lag. Nevertheless, obstacles increase navigation time, with the *Obstacle – Corner* case being the most time-consuming. The consistency of trial outcomes suggests that the navigation system is reliable, but environmental complexity directly impacts the navigation time. CPU utilization during navigation ranges between 90–95% indicating a substantial burden on the system's resources. This autonomous navigation system has promising applications in sectors such as healthcare, restaurants, warehouses, etc. With larger motors and enhanced mechanical designs, the system can be customized for tasks such as medication delivery, patient assistance, and administrative support in environments without steps and stairs. Moreover, by automating routine tasks, AMR can help streamline hospital operations, reduce errors in medication delivery, and improve overall efficiency. Future improvements could focus on optimizing navigation algorithms to reduce computational demand while maintaining reliability.

REFERENCES

Cadena, C., Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, Present, and Future of Simultaneous Localization and

Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, 32(6), pp. 1309-1332, Dec. 2016, doi: 10.1109/TRO.2016.2624754.

Ess, A., Schindler, K., Leibe, B., and Van Gool, L. (2010). Object Detection and Tracking for Autonomous Navigation in Dynamic Environments. *The International Journal of Robotics Research*, 29(14), pp.1707–1725. doi:10.1177/0278364910365417

Galtarossa L, "Obstacle Avoidance Algorithms for Autonomous Navigation system in Unstructured Indoor areas," Torino, 2018.

Hamid, A., Hamdany, S., Albak, L. H., Al-Nima, O.R.R (2019). *Wireless Waiter Robot. Test Engineering and Management*, The Mattingley Publishing Co., Inc 81, ISSN:0193 – 4120, 2486 -2494.

Kasun K. Jinasena and Ravinda G. N. Meegama (2011). Design of a Low-Cost Autonomous Mobile Robot." *International Journal of Intelligent Systems and Applications in Robotics (IJRA)*, 2(1), pp.1–13.

Madhankumar, S., Anandraj, P., Varadarajan, A., Kumar, R. A., & Kaleeswaran, K. (2021). Design and Modelling of Autonomous Mobile Robot for Material Handling. 2021 *7th International Conference on Advanced Computing and Communication Systems (ICACCS)* doi:10.1109/icaccs51430.2021.9441831

Mane, A. A., Parihar, M. N., Jadhav, S. P., and Digey, B. B. (2015). Robotics based simultaneous localization and mapping of an unknown environment using Kalman Filtering, 5th *Nirma University International Conference on Engineering (NUiCONE)* doi:10.1109/nuicone.2015.7449602

Morgan Quigley , Brian Gerkey , Ken Conley , Josh Faust , Tully Foote , Jeremy Leibs , Eric Berger , Rob Wheeler and Andrew Ng (2009). ROS: an open-source Robot Operating System. ICRA Workshop on Open Source Software, 3(3.2)

Nikam, Amruta, Doddamani Akshata, Deshpande Divya and Manjramkar Shrinivas (2017). Raspberry Pi Based Obstacle Avoiding Robot. International Research Journal of Engineering and Technology (IRJET), 4(2), pp.917 – 919.

Panda Subodh Kumar, Ramya N., Priyanka A., Monika M. and Monika H. R. (2020). An Autonomous Robotic Trolley for Waiter Service in Restaurant with Smart Ordering System. Journal of Critical Reviews. ISSN 2394 – 5125, 7(8), 2329 – 2341.

Sahu, B. K., Kumar Sahu, B., Choudhury, J., and Nag, A. (2019). "Development of Hardware Setup of an Autonomous Robotic Vehicle Based on Computer Vision Using Raspberry Pi", Innovations in Power and Advanced Computing Technologies (i-PACT). doi:10.1109/i-pact44901.2019.8960011

Shaker, Maan M., Omran, Safaa S. and Abdulla, Ali A. (2008). Design and Implementation of an Intelligent Autonomous Robot System. *International Conference on Automation, Robotics and Control Systems (ARCS-08),* 131-136.

Sheu, J. P., Chang, C. C., Lo, K. W., and Deng, C. W. (2010). Design and implementation of a navigation system for autonomous mobile robots. *International Journal of Ad hoc and Ubiquitous Computing*, 6(3), 129, doi:10.1504/ijahuc.2010.034966

Shiny.J.S, Ashok Kumar.M, Nanthagopal.V, `Raguram.R (2017) Automation of Restaurant. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE)* 6(3), pp.1532 – 1539 DOI:10.15662/IJAREEIE.2017.0603068

Srilekha, M., Noureen Fathima, Anusha, G., D. Shravana Chary (2018) "Line Follower Alphabot Using Arduino Micro Controller". *International Journal of Engineering Research in Electrical and Electronic Engineering (IJEREEE)*, 4(2) ISSN(Online) 2395 – 2717.

Yanmida, D. Z., Alim, S. A., & Imam, A. S. (2020). Design and Implementation of an Autonomous Delivery Robot for Restaurant Services. ELEKTRIKA- *Journal of Electrical Engineering,* 19(3), 66–69. doi.org/10.11113/elektrika.v19n3.227