Thiết kế và triển khai robot di động tự hành dựa trên ROS với giao diện hỗ trợ điều hướng theo lộ trình

TÓM TẮT

Robot di động tự hành (AMR) ngày càng được ứng dụng rộng rãi trong các môi trường logistics và dịch vụ trong nhà, nơi yêu cầu phải bảo đảm khả năng lập bản đồ, định vị và điều hướng tin cậy. Bài báo này trình bày thiết kế và triển khai một AMR nhỏ gọn dựa trên ROS, trang bị Raspberry Pi 4, LiDAR 2D, IMU và encoder. Nhằm tăng tính thuận tiện khi vận hành, chúng tôi phát triển một giao diện đồ họa (GUI) độc lập, thay thế thao tác trực tiếp trên Rviz, người dùng có thể điều khiển robot đến bốn điểm đích định sẵn hoặc thực thi lộ trình nhiều waypoint do người dùng cấu hình, đồng thời theo dõi trạng thái điều hướng theo thời gian thực. Hệ thống tích hợp 2D SLAM để xây dựng bản đồ, AMCL để định vị, Dijkstra cho lập kế hoạch toàn cục và Dynamic Window Approach (DWA) cho điều hướng cục bộ kết hợp tránh vật cản. Thử nghiệm trong các kịch bản indoor cho thấy odometry từ encoder có sai số quãng đường trung bình 5,8–7,0% trên đoạn 1–3 m; sai số đo khoảng cách LiDAR <3% đến 10 m; và kích thước bản đồ tái tạo lệch <1,3% so với giá trị thực tế. Trong thử nghiệm điều hướng, robot đạt mục tiêu ổn định ở cả môi trường ít và nhiều vật cản sau khi tinh chỉnh tham số, đồng thời thực thi tin cậy các nhiệm vụ waypoint liên tiếp. Kết quả khẳng định tính khả thi của việc triển khai AMR dựa trên ROS với giao diện tương tác người dùng.

Từ khoá: *ROS, AMR, SLAM, AMCL, DWA.*

Design and Implementation of a ROS-Based Autonomous Mobile Robot with a GUI-Assisted Route Navigation Interface

ABSTRACT

Autonomous mobile robots (AMRs) are increasingly deployed in indoor logistics and service environments, where reliable mapping, localization, and navigation are essential. This paper presents the design and implementation of a compact ROS-based AMR equipped with a Raspberry Pi 4, a 2D LiDAR, an IMU, and wheel encoders. To improve operational usability, we develop a standalone graphical user interface (GUI) that replaces direct RViz interaction; users can command the robot to navigate to one of four predefined destinations or execute a user-configurable multi-waypoint route while monitoring navigation status in real time. The system integrates 2D SLAM for map construction, AMCL for localization, a Dijkstra-based global planner, and the Dynamic Window Approach (DWA) for local motion generation with obstacle avoidance. Experiments in representative indoor scenarios show that encoder-based odometry yields a mean distance error of 5.8–7.0% over 1–3 m trajectories; LiDAR ranging error remains below 3% up to 10 m; and reconstructed map dimensions deviate by less than 1.3% from ground-truth measurements. In navigation trials, the robot consistently reached target goals in both obstacle-free and obstacle-rich environments after parameter tuning, and it reliably executed sequential waypoint missions. These results confirm the feasibility of deploying ROS-based AMRs with an interactive user interface for practical indoor applications.

Keywords: *ROS, AMR, SLAM, AMCL, DWA.*

1. INTRODUCTION

Nowadays, autonomous mobile robots (AMRs) are being widely deployed across numerous domains, ranging from logistics systems, goods transportation, surveillance, and quality inspection to navigation tasks in complex environments. These applications optimize operational workflows, reduce human intervention, and improve overall productivity. In robotics research, robot localization and navigation constitute key areas; algorithms such as SLAM (Simultaneous Localization and Mapping), path planning, and motion control play pivotal roles in enabling robots to operate effectively in previously unknown environments.[1-2]

The field of autonomous robotics can be traced back to Leonard's work in 1990, which introduced fundamental approaches for 2D mapping and robot localization in three-dimensional space. This line of research established the foundations for SLAM (Simultaneous Localization and Mapping), which was later advanced by Durrant-Whyte and Bailey, providing essential tools for localization and mapping in unknown environments.[3-4] These contributions have significantly influenced the development of modern autonomous robotic technologies, particularly the adoption of ROS (Robot Operating System) and SLAM algorithms for mobile robots.

Modern SLAM algorithms—especially methods leveraging LiDAR and IMU sensing—have demonstrated strong capability in accurate 2D map reconstruction and effective robot localization in three-dimensional space. Nevertheless, deploying such SLAM approaches in real-world settings can pose substantial challenges, particularly when constrained by low-cost, simplified hardware platforms. Several recent studies have addressed this issue by proposing cost-effective solutions that still maintain high performance, especially for small- and medium-scale autonomous robotic applications.[5-7]

At present, many studies in mobile robotics employ ROS to simulate SLAM algorithms or rely on off-the-shelf robot platforms such as Pioneer and Turtlebot. However, these platforms are often incompatible with the financial constraints and research infrastructure commonly encountered in Vietnam, where funding for autonomous robotics remains limited.[8] Moreover, solutions that use sensors such as LiDAR and cameras for mapping and navigation typically entail challenges related to cost and computational resources.[9]

In Vietnam, mobile robotics continues to be a focal research topic. Existing studies largely concentrate on developing control systems for mobile robots operating in known environments, often based on two-wheel differential-drive structures and fixed kinematic models. However,

such approaches may not satisfy the requirements for mobility in complex environments characterized by dense obstacles and continuously changing spatial configurations.[10] Recent research has therefore shifted toward path planning and motion control, aiming to optimize trajectories for autonomous robots in environments without detailed pre-existing maps, supported by algorithms such as A* and DWA (Dynamic Window Approach).[11-12]

With the objective of providing a low-cost solution while maintaining high performance, this paper presents a ROS-based system for navigation and mapping, together with the DWA algorithm for local trajectory planning. Both real-world experiments and simulation results are reported, demonstrating that our robot platform can transport goods to user-selected target locations marked on the map with high accuracy, thereby meeting the requirements of autonomous robotic applications. [13-14]

The remainder of this paper is organized as follows. Section 2 describes the system architecture and hardware design, followed by the differential-drive kinematic model and software implementation under ROS. Section 3 presents the adopted mapping, localization, and navigation algorithms (slam_gmapping, AMCL, global planning, and DWA-based local planning). Section 4 reports experimental protocols and quantitative results for sensor evaluation, mapping accuracy, localization tuning, and navigation performance. Section 5 concludes the paper and outlines future improvements.

2. METHODS

2.1. Design methodology

An overview of the proposed system is illustrated in Fig. 1. A user operates a host computer to monitor the mapping process and to command the robot to navigate to target locations via an embedded Raspberry Pi 4. Both the host computer and the Raspberry Pi 4 run ROS on Ubuntu 20.04 and communicate over a shared Wi-Fi network; the Raspberry Pi is configured with a static IP address. The embedded computer acquires LiDAR measurements for map construction. Meanwhile, the microcontroller collects inertial measurements from the IMU and transmits them to the embedded computer, and it also drives the actuators—two DC motors—through a motor driver module.

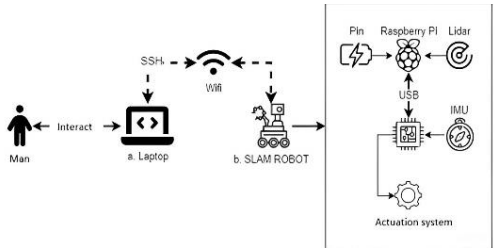

Figure 1. Overview of the proposed system

2.1.1. Hardware design

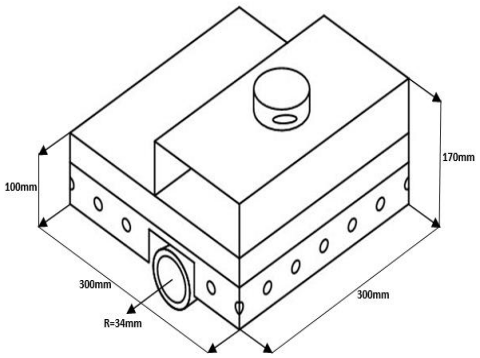

Figure 2. Dimensional drawing of the robot

The robot chassis is fabricated from aluminum to reduce mass and improve maneuverability, with overall dimensions of 300 × 300 × 170 mm. Because the platform is intended to operate indoors on flat terrain with low wheel slip, conventional wheels are adopted with radius r = 34mm (Fig 3a). The robot frame is designed to accommodate a top cover that mounts an RPLidar A1 sensor.

As shown in Fig. 3b, the robot body houses a Raspberry Pi 4 for sensor processing and control-board interfacing. A 9-DOF IMU (MPU-9250) enables motion tracking along three spatial axes, improving localization capability and motion stability. Two BTS7960 motor driver modules control two GA-25 DC geared motors equipped with dual-channel A/B encoders, allowing accurate motion execution and motor speed feedback. The system is powered by a 12-V battery pack.

(a) (b)

Figure 3. Robot prototype: (a) external view, (b) internal layout.

2.1.2. Component selection

Embedded computer (Raspberry Pi 4). The Raspberry Pi 4 is used as the main embedded

computer and communicates with the microcontroller. It features a quad-core ARM Cortex-A72 CPU at 1.5 GHz, 4 GB RAM, USB 3.0, Gigabit Ethernet, Wi-Fi 802.11ac, Bluetooth 5.0, and a 40-pin GPIO header, making it well-suited for the proposed autonomous robot platform. The embedded computer acquires motor speed information, robot yaw angle, and LiDAR measurements, processes these data using integrated ROS software packages, and transmits control commands to the microcontroller for execution.

Laser rangefinder (RPLidar A1). The RPLidar A1 measures distances to surrounding obstacles, enabling 360° environmental scanning and map generation using the slam_gmapping algorithm. The sensor supports obstacle detection up to 12 m with a sampling rate of 8000 samples/s. It is also supported by ROS (Robot Operating System) integration packages for robot applications.

IMU (9-DOF MPU-9250). The MPU-9250 integrates an accelerometer, gyroscope, and magnetometer to measure tri-axial acceleration, angular velocity, and magnetic field. Through I²C communication, it provides nine sensor readings that can be fused to estimate the robot's Euler angles (roll, pitch, yaw). This device is critical for maintaining robot stability and heading estimation, thereby supporting accurate and efficient 2D navigation.

Geared DC motor with encoder (GA-25, 12 VDC). The GA-25 is a DC geared motor equipped with an encoder to provide feedback signals. The microcontroller counts encoder pulses and adjusts motor actuation through the power driver, enabling motor speed and rotation measurement. Key specifications include: nominal voltage 12 VDC, shaft diameter 4 mm, gear ratio approximately 46.8:1, no-load speed 130 rpm, rated torque 0.9 kg·cm, and maximum torque 4.4 kg·cm. The encoder is dual-channel (A/B) with an 11-pulse disk per channel. This motor/encoder pair enables accurate estimation of the robot's translational speed and displacement, and plays a crucial role in motion control and trajectory tracking.

Motor driver (BTS7960). The BTS7960 motor driver is used to regulate motor speed and direction via PWM control signals. With a current capability up to 43 A, it supports stable velocity regulation and smooth directional changes. This driver contributes to precise and responsive motor actuation, improving the overall navigation performance of the robot.

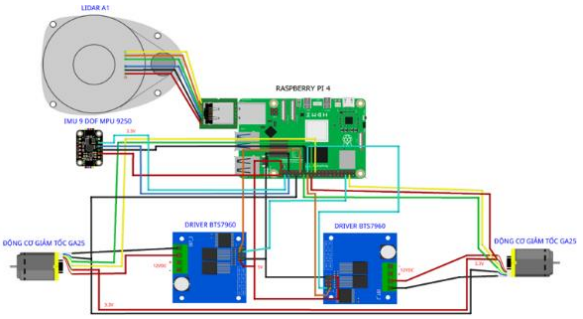

Figure 4. Hardware wiring diagram.

Figure 4 summarizes the hardware interconnections, illustrating how key modules are integrated into a complete control system.

2.2. Algorithmic description

2.2.1. Kinematic model

A differential-drive autonomous mobile robot (AMR) consists of two independently driven wheels mounted on either side of a central chassis, with passive caster wheels used for mechanical stabilization.

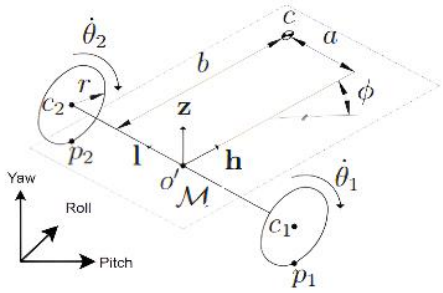

Figure 5. Kinematic model of a two-wheel differential-drive mobile robot

A typical configuration of a differential-drive wheeled mobile robot (DDWMR) is shown in Fig. 5. Let a fixed reference frame F be attached to the ground, whose axes are represented by the mutually orthogonal unit vectors x, y and z . A body (mobile) frame M is attached at the midpoint of the line segment connecting the wheel centers c_1 and c_2. The unit vectors h, l and z are associated with the moving frame M. The wheel radius is denoted by r. To derive the kinematic model of the DDWMR, the following key relationships are considered.

The angular velocities of the left and right wheels are denoted by $\dot{\theta}_1$ and $\dot{\theta}_2$, forming an independent velocity vector:

$$\dot{\theta} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \qquad (1)$$

The linear velocity V and the angular velocity $\dot{\phi}$ in the moving frame M are related to $\dot{\theta}$ via a Jacobian matrix:

$$v = \begin{bmatrix} V \\ \dot{\phi} \end{bmatrix} = K\dot{\theta},\ K = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix} \quad (2)$$

To transform velocities from the moving frame M to the fixed frame F, a rotation matrix is used:

$$R(\phi):\ R(\phi) = \begin{bmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

The twist velocity in the fixed frame $\dot{q}$ is obtained by multiplying the rotation matrix with the velocity vector v expressed in frame M:

$$\dot{q} = R(\phi)v \quad (4)$$

The Jacobian relating $\dot{q}$ and $\dot{\theta}$ is:

$$\dot{q} = J\dot{\theta},\quad J = R(\phi)K \quad (5)$$

Thus,

$$J = \begin{bmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix} \quad (6)$$

Accordingly, the robot's angular and linear velocities can be expressed in terms of wheel angular velocities:

$$\dot{\phi} = \frac{r}{d}\left(\dot{\theta}_1 - \dot{\theta}_2\right) \text{ và } \dot{\phi} = \frac{r}{2}\left(\dot{\theta}_1 + \dot{\theta}_2\right) \quad (7)$$

In this study, pure rolling motion is assumed; therefore, lateral slip angles and side-slip displacement terms δ_i and $\dot{\varepsilon}$ are neglected (set to zero).

Notation:

$\dot{\theta}_1, \dot{\theta}_2$	angular velocities of the left and right wheels
r	wheel radius
d	distance between the two drive wheels
V	robot linear velocity
K	Jacobian mapping wheel angular velocity to $[V, \dot{\phi}]^T$
$\dot{q}$	twist velocity expressed in the fixed frame F
$R(\phi)$	rotation matrix from M to F.
J	Jacobian mapping $\dot{\theta}$ to $\dot{q}$.
M	moving (body) frame
F	fixed (world) frame
l	coordinate of the midpoint between the wheels
h	wheel–ground contact location representation.
c_1, c_2	centers of the left and right wheels
p_1, p_2	left/right wheel ground contact points
x, y	unit vectors of the fixed frame F
z	unit vector along the z-axis

2.2.2. Overall algorithm description

Deploying an autonomous robot typically requires three core capabilities: mapping, localization, and navigation. In this work, ROS (Robot Operating System) is used to implement these capabilities, as it provides a mature ecosystem of tools and software packages. Specifically, we employ slam_gmapping for map construction and amcl for robot localization within a known map. For navigation, the ROS navigation stack is adopted, including move_base, global_planner, and base_local_planner. The AMCL (Adaptive Monte Carlo Localization) method uses a particle filter to estimate the robot pose on the map, maintaining localization accuracy during motion.

The ROS navigation framework is organized into two planning layers: global planning and local planning, corresponding to algorithms such as Dijkstra (global planner) and the Dynamic Window Approach (DWA) for local trajectory generation.

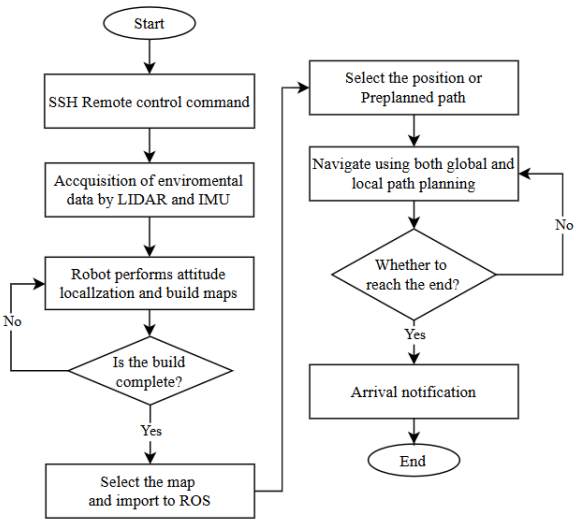

Figure 6. Flowchart of localization and navigation.

Operationally, the robot explores an initially unknown environment using LiDAR measurements (with remote access/monitoring via SSH when needed). Sensor data from the LiDAR and IMU are continuously acquired to support simultaneous localization and mapping. If the map is not yet complete, the robot continues exploring and updating the map. Once the mapping phase is finalized, the user selects start and goal poses in ROS RViz. The robot then computes an optimal path from start to goal using global and local planning modules. If the robot has not reached the goal, the navigation loop continues; otherwise, the process terminates. The procedure completes when the robot successfully navigates to the designated goal location.

In addition to the ROS-native workflow, we introduce a mission-level graphical user interface

(GUI) that abstracts away direct RViz interaction during deployment. The GUI enables non-expert users to command the robot using high-level navigation tasks: (i) selecting one of four predefined goal locations, or (ii) executing a sequential mission consisting of multiple waypoints arranged in a user-defined order. Under the hood, the GUI translates these task selections into standard ROS navigation goals, dispatches them to the navigation stack, and monitors execution states in real time (e.g., goal accepted, in-progress, succeeded, or aborted). For sequential missions, the interface issues the next waypoint only after the current one is completed (or after a configured retry/termination condition), thereby ensuring deterministic mission progression. This design simplifies operational procedures, improves usability outside laboratory settings, and provides a practical pathway for integrating ROS-based autonomy with application-oriented user interaction.

2.2.3. SLAM-based mapping algorithm

SLAM (Simultaneous Localization and Mapping) is a class of algorithms that enables a robot to construct a map of its environment while simultaneously estimating its own pose. During motion, sensors such as LiDAR, IMU, and wheel encoders acquire measurements of the surroundings, which are then transformed into a structured map representation. A reliable map is a prerequisite for subsequent localization and navigation, since the robot must be able to estimate its pose with respect to the constructed map.

The SLAM problem can be formulated as the following posterior probability:

$$P(m_t, x_t \mid o_{1:t}, u_{1:t}) \quad (8)$$

where:

- x_t denotes the robot state (pose) at time t.
- m_t is the environment map
- $o_{1:t}$ represents the sensor observations from time 1 to t.
- $u_{1:t}$ denotes the control inputs from time 1 to t.

The continuous Bayesian update for the map and pose can be expressed as:

$$P(x_t|o_{1:t}, u_{1:t}m_t) \propto P(o_t|x_t, m_t)$$

$$\int P\,(x_t \mid x_{t-1}, u_t)P(x_{t-1} \mid o_{1:t-1}, u_{1:t-1}, m_t)dx_{t-1}$$

SLAM not only yields an optimization solution, but also provides a real-time probabilistic estimate, allowing the robot to continuously update both the map and its pose in dynamic environments.

2.2.4. AMCL localization algorithm

After the environment map has been constructed, the robot must estimate its current pose within this map. This functionality is provided by Adaptive Monte Carlo Localization (AMCL), which is a particle-filter-based approach. AMCL typically consists of two main steps: (i) sampling candidate poses from the process (motion) model, and (ii) computing particle weights using the measurement model, followed by resampling to emphasize high-weight particles.

The AMCL filter proceeds as follows:

Step 1. Initialize empty particle sets:

$$\bar{X}_t = X_t = \emptyset \quad (9)$$

Step 2. For each particle, sample a predicted pose and compute its weight:

Sampling: $x_t^{[i]} \sim p\left(x_t \middle| u_t, x_t^{[i]}\right)$ (10)

Weight computation:

$$\omega_t^{[i]} = p\left(z_t \middle| x_t^{[i]}\right) \quad (11)$$

Update the temporary particle set:

$$\bar{X}_t = \bar{X}_t + \langle x_t^{[i]}, \omega_t^{[i]} \rangle \quad (12)$$

Compute the mean weight:

$$\omega_{avg} = \omega_{avg} + \frac{1}{M}\omega_i^{[m]} \quad (13)$$

Short-term weight estimate:

$$\omega_{slow} = \omega_{slow} + \alpha_{slow}\left(\omega_{avg} - \omega_{slow}\right) \quad (14)$$

Long-term weight estimate:

$$\omega_{fast} = \omega_{fast} + \alpha_{fast}\left(\omega_{avg} - \omega_{fast}\right) \quad (15)$$

Step 3. Inject random particles into the official set X_t with probability:

$$\max\left(0.0, 1.0 - \frac{\omega_{fast}}{\omega_{slow}}\right) \quad (16)$$

Step 4. Resample $x_t^{[m]}$ from the temporary set $\bar{X}_t$ according to the corresponding weights, and update the particle set:

$$X_t = X_t + \langle x_t^{[m]}, \omega_t^{[m]} \rangle \quad (17)$$

The short- and long-term update rates should satisfy:

$$0 \leq \alpha_{slow} \ll \alpha_{fast}$$

When the short-term estimate exceeds the long-term estimate, the filter is performing well, indicating that additional random particle injection is unnecessary.

2.2.5. Navigation algorithm

We employ Dijkstra's algorithm, a shortest-path method for graphs with non-negative edge weights. The environment is represented as a costmap, where each grid cell is assigned a traversal cost (e.g., obstacle cells have high cost).

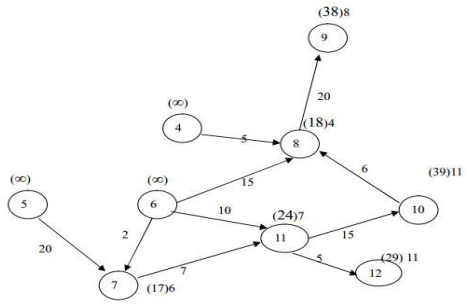

Figure 7. Illustration of Dijkstra-based path planning.

The core Dijkstra update rule is:

$$d(v) = min(d(u) + w(u,v)) \ \forall(u,v)$$

where:

$d(v)$ is the current best estimate of the minimum cost to node v, and $w(u,v)$is the traversal cost from node $u$to node v. Starting from the initial node, the algorithm iteratively expands nodes in increasing order of cost to compute the shortest path to all reachable nodes, and terminates once the goal is reached.

For global planning, the pre-built map is discretized into grid cells and Dijkstra's algorithm is applied to obtain an optimal path from the start pose to the goal pose.

For local planning, relying solely on a global planner is insufficient because unexpected obstacles may appear during execution, rendering the precomputed global path infeasible. Therefore, we adopt the DWA local planner (Dynamic Window Approach), which provides a reactive control layer coupled to the planner. DWA evaluates candidate velocity commands using a grid-based cost function that encodes traversal costs. The controller selects the command velocities $(\dot{x}, \dot{y}, \dot{\theta})$to be issued to the robot.

The key steps of DWA are summarized as follows:

Step 1: Independently sample control commands in the robot control space $(\dot{x}, \dot{y}, \dot{\theta})$

Step 2: For each sampled command, perform forward simulation from the current robot state over a short time horizon to predict the resulting motion.

Step 3: Score each simulated trajectory using a weighted objective that accounts for factors such as obstacle clearance, goal proximity, adherence to the global path, and velocity; discard invalid trajectories that lead to collisions.

Step 4: Select the highest-scoring trajectory and send the corresponding velocity and angular-rate commands to the robot.

3. EXPERIMENTS AND EVALUATION

3.1. Encoder evaluation

The robot employs two wheel encoders to independently control the left and right drive wheels. To assess encoder accuracy, we measured the robot's actual traveled distance and compared it with the distance estimated from encoder pulse counts, in conjunction with the robot kinematic model. Experiments were conducted for commanded travel distances of 1 m, 2 m, and 3 m, with three trials per distance.

Table 1. Encoder evaluation results.

To compute the theoretical encoder pulses for each travel distance, the wheel circumference is obtained by:

$$C = \pi \times D$$

Where: $C$is the wheel circumference and $D$is the wheel diameter. With $D = 68$mm, the wheel circumference is approximately $C \approx 213.6$mm.

Actual (m)	Theoretical (pulses)	Trial	Measured (pulses)	Estimated (m)	Error (%)
1	2415	1	2566	1.06334	6.33
		2	2509	1.03972	3.97
		3	2581	1.06955	6.96
		Avg	2552	1.05787	5.79
2	4831	1	5188	2.14989	7.49
		2	5089	2.10886	5.44
		3	5192	2.15154	7.58
		Avg	5156.3	2.13676	6.84
3	7247	1	7722	3.19997	6.67
		2	7751	3.21199	7.07
		3	7848	3.22135	7.38
		Avg	7773.6	3.2111	7.04

For a 1 m displacement, the required number of wheel revolutions is:

$$\text{N}= \frac{Distance}{C}$$

which yields $N \approx 4.68$revolutions. Since each revolution generates 514 pulses, the theoretical pulse count for a 1 m displacement is 2415 pulses. Similarly, the theoretical pulse counts for 2 m and 3 m are 4831 pulses and 7247 pulses, respectively.

The results indicate that for a 1 m displacement, the average error is 5.79%, reflecting relatively high encoder accuracy over short distances. For 2 m, the average error increases slightly to 6.84%; however, this level remains within a reasonable range and does not significantly affect the robot's overall performance. For 3 m, the average error is 7.04%, which is the largest among the three tested distances but is still acceptable. Overall, the results suggest that the estimation error tends to increase with travel distance.

In summary, the wheel encoders provide sufficiently accurate distance measurement for the proposed AMR platform, with errors ranging from 5.79% to 7.04% under low-disturbance indoor conditions. This accuracy is considered acceptable

for autonomous navigation tasks in the target operating environment.

3.2. IMU sensor evaluation

The IMU performance is evaluated by measuring the robot's yaw angle, which represents the robot's rotation about the vertical *z*-axis. The yaw angle is used to determine the robot heading during motion and is therefore critical for localization and navigation.

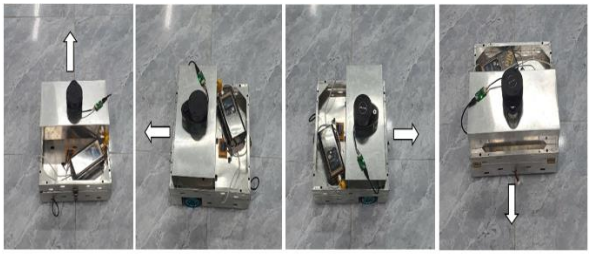

Figure 8. IMU evaluation: (a) initial yaw angle, (b) −90°, (c) 90°, and (d) 180°.

In this study, the IMU accuracy was assessed through three rotational maneuvers. The IMU outputs include raw yaw (unprocessed) and filtered yaw (after applying the proposed filter). This comparison is used to evaluate (i) the practical accuracy of the IMU under real operating conditions and (ii) the effectiveness of filtering in improving measurement accuracy. Throughout the experiments, yaw follows the right-hand convention, where counterclockwise rotations are positive and clockwise rotations are negative.

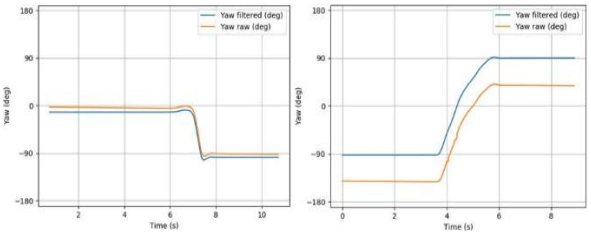

Figure 9. Raw and filtered yaw profiles for rotations 1 and 2.

The robot was initially aligned at a fixed heading of 0˚(Fig. 8a), then rotated to a target heading of −90˚(Fig. 8b). Next, it was commanded to rotate to 180˚(Fig. 8d). The corresponding raw and filtered yaw measurements are shown in Fig. 9.

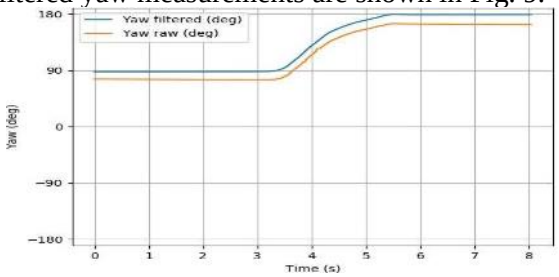

Figure 10. Raw and filtered yaw profiles for rotation 3.

Finally, the robot was rotated to a target heading of 90˚(Fig. 8c). The raw and filtered yaw signals for this third maneuver are reported in Fig. 10.

Table 2. Comparison of raw and filtered yaw measurements.

Rotation	Actual (°)	Yaw raw (°)	Yaw filtered (°)	Raw error (%)	Filtered error (%)
1	-90	-90.2	-89.5	0.22	0.56
2	180	179.2	180.1	0.44	0.06
3	90	87.5	90	2.78	0

For rotation 1, the IMU raw yaw exhibited high accuracy with an error of 0.22%, whereas filtering did not improve the estimate and resulted in an error of 0.56%. For rotation 2, the raw measurement remained accurate (0.44%), and filtering further improved the estimate, reducing the error to 0.06%. For rotation 3, the raw yaw error increased to 2.78%; however, the filtered yaw achieved 0% error, indicating that the filter effectively mitigated noise and bias in this maneuver. Overall, the IMU provides reasonably accurate raw yaw estimates, while filtering significantly reduces errors, particularly during rapid or large-angle rotations.

Overall, the IMU provides reasonably accurate raw yaw estimates, but the error may increase under rapid heading changes or large rotations. The applied filter significantly reduces the error and improves yaw accuracy under such conditions.

3.3. LiDAR sensor evaluation

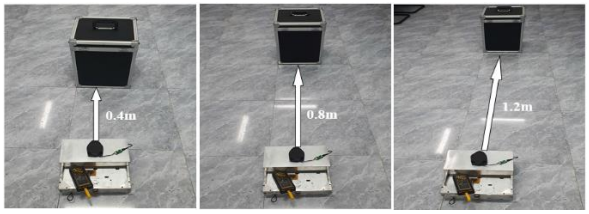

Figure 11. Experimental setup for LiDAR evaluation.

The accuracy of the RPLidar A1 laser rangefinder was evaluated as showed in Fig 11. A target obstacle was placed at a distance of exactly one floor-tile length from the LiDAR sensor (each tile is 40 cm). The sensor's measured distance was recorded and compared against the ground-truth distance. The procedure was repeated by incrementally increasing the obstacle distance by integer multiples of the tile length. For each distance, 10 measurements were collected and averaged. The results are reported in Table 3.

Table 3. LiDAR distance measurement evaluation.

Actual (m)	Mean measured (m)	Absolute error (m)	Error (%)
0.4	0.4033	0.0033	0.825
0.8	0.8086	0.0086	1.075
1.2	1.2162	0.0161	1.34
2	2.0315	0.0315	1.58
3.6	3.6639	0.0639	1.78
4.6	4.6882	0.0882	1.92
6.6	6.7531	0.1531	2.31
8.2	8.412	0.221	2.59
10	10.2835	0.2835	2.84

The LiDAR performs effectively at short ranges (below 5 m), where the measurement error remains under 2%. However, the error increases with distance, indicating reduced accuracy at longer ranges. The maximum observed error is 2.84% at 10 m. Although the manufacturer-rated maximum range is 12 m, repeated measurements suggest that the practical effective range is approximately 10–10.5 m, with an error increasing from about 0.8% to 2.8% as distance grows. These errors remain acceptable for the mapping stage in our application.

3.4. Mapping accuracy evaluation

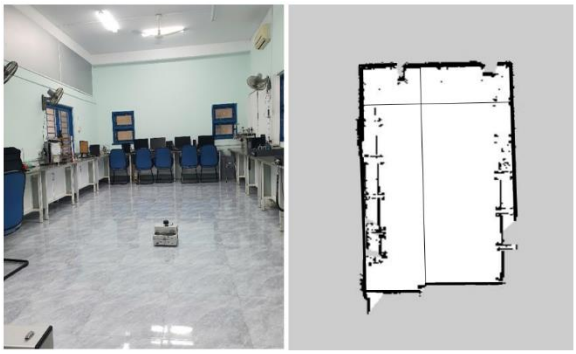

Figure 12. 2D map construction results.

We performed 2D mapping in an indoor environment. The objective is to evaluate mapping accuracy by comparing the reconstructed map dimensions with the ground-truth environment dimensions and quantifying the resulting errors. In the map, black regions represent obstacles, while white regions denote free space where the robot can traverse.

Table 4. Comparison between ground-truth and map-measured environment dimensions.

Dimension	Actual (m)	Measured on map (m)	Absolute error (m)	Error (%)
Length	10.6	10.46336	0.13664	1.29%
Width	4.08	4.04712	0.03288	0.81%

The reconstructed map closely matches the real environment. The length error is 1.29%, which is acceptable, while the width error is only 0.81%. Errors below 2% indicate that the proposed system provides reliable measurement and mapping performance. It should be noted that obstacles are detected only if they are approximately at the same height as the LiDAR sensor and lie within the sensing range; therefore, environmental and system noise can introduce additional mapping artifacts. Nonetheless, the observed errors remain within acceptable limits for the intended application.

3.5. AMCL localization performance evaluation

We first evaluated the AMCL filter under different particle set sizes to identify a configuration that provides the best trade-off between accuracy and computational efficiency.

AMCL employs a particle filter to estimate the robot pose within the environment. The number of particles is a critical factor that strongly influences both estimation accuracy and computational cost. In this experiment, three particle counts (1000, 5000, and 7000) were tested to evaluate AMCL performance.

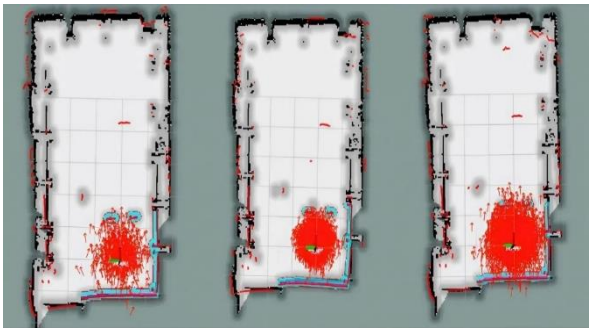

Figure 13. AMCL particle filter initialization with 1000, 5000, and 7000 particles.

Table 5. Impact of particle count on AMCL performance.

Particles	Performance (%)	Accuracy (m)	Error (%)	Computation time (s)
1000	70	1.45	10.25	0.5
5000	90	0.93	6.75	1.2
7000	95	0.75	5.31	2.3

With 1000 particles, AMCL is able to estimate robot pose, but the accuracy is limited, yielding a relatively large error (10.25%) despite low computation time. Increasing the particle count to 5000 improves pose estimation accuracy and reduces the error to 6.75%, although the accuracy remains moderate. The 7000-particle configuration achieves the best localization

accuracy (error reduced to 5.31%), at the expense of longer computation time.

Based on these results, 7000 particles is selected as the optimal configuration, providing a favorable balance between localization accuracy and overall system performance. After the robot moves for a period of time, the particle set converges and the estimated pose aligns most closely with the given map.

3.6. Navigation performance evaluation

We evaluated the robot navigation capability in two types of environments:

- Obstacle-free environment with a straight-line trajectory

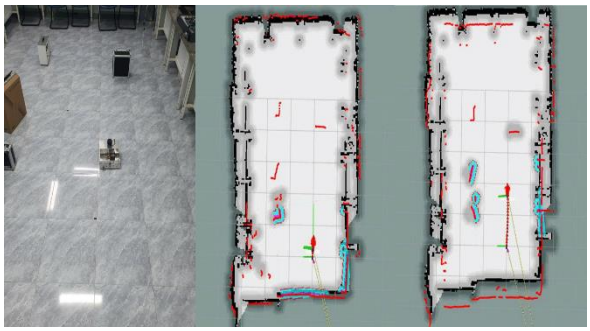

Figure 14. Robot navigation in an obstacle-free environment (straight trajectory).

- Complex environment with obstacles

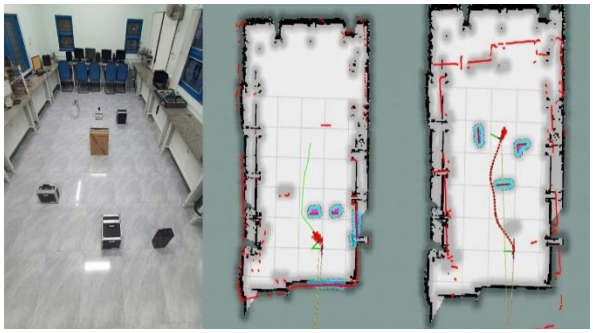

Figure 15. Robot navigation in a complex environment with obstacles.

Table bellow summarizes the navigation performance in both scenarios, Particles were fixed at 7000 in all navigation tria

Table 6. Navigation performance under obstacle-free and obstacle-rich environments

	Time to goal (s)	Travel distance (m)	Error (%)
No obstacles	4.5s	4.0m	0
Obstacles (cost factor= 1.5)	6.5s	4.2m	1.0
Obstacles (cost factor = 1.2)	5.8s	4.3m	0.7
Obstacles (cost factor = 1.0)	5.1s	4.5m	0.4

In the obstacle-free case, the robot reached the goal with a short travel time and path length, achieving accurate motion without noticeable deviation. In the obstacle-rich environment, a higher cost factor (cost factor = 1.5) caused the robot to perform overly conservative obstacle avoidance, increasing the time to reach the goal while still maintaining acceptable accuracy. When the cost factor was reduced to 1.0, the time-to-goal decreased significantly by reducing unnecessary detours, and the error dropped to 0.4%, indicating robust navigation behavior. Therefore, cost factor = 1.0 was selected as the most suitable setting for our system, achieving faster navigation while preserving reliable performance.

3.7. Navigation to predefined target locations

To enable navigation to predefined target positions, we developed a user interface with a simple and intuitive design. The operator can easily select the destination point, which streamlines autonomous operation and allows the AMR to navigate to predefined locations on the map. The interface displays several preset goal markers (Position 1, Position 2, Position 3, and Position 4) that the robot can navigate to.

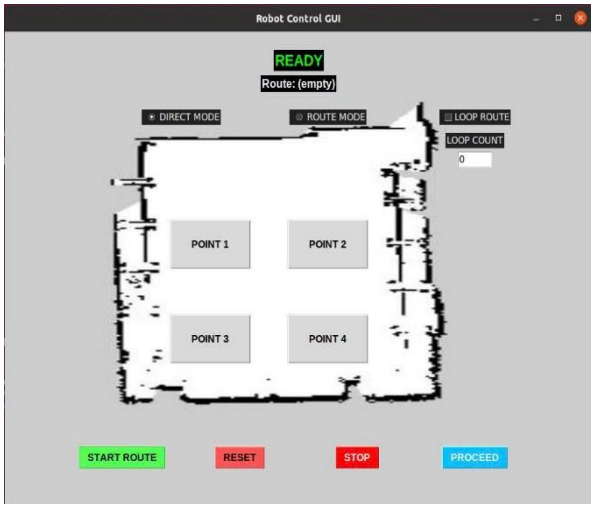

Figure 16. UI layout and labeled goals.

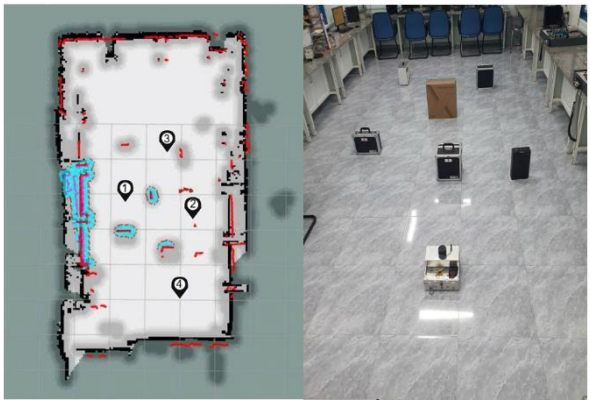

Figure 17. UI screenshot during experiment and robot trajectory overlay.

The interface provides real-time updates of the robot's current state, allowing the operator to monitor the navigation process conveniently. The preset goal points represent key task locations. During navigation, the robot must pass through narrow regions, which require accurate and agile motion control. The control system and interface support continuous observation and performance verification throughout the mission.

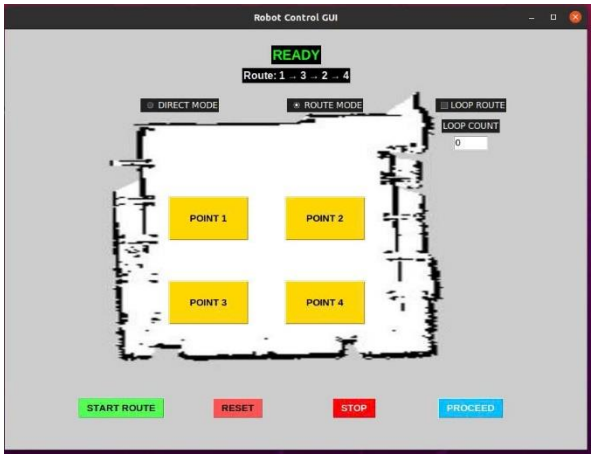

Figure 18. Robot navigation user interface.

We evaluated a multi-goal mission in which the robot started at Position 4, then navigated sequentially to Position 1, Position 3, Position 2, and finally returned to Position 4.

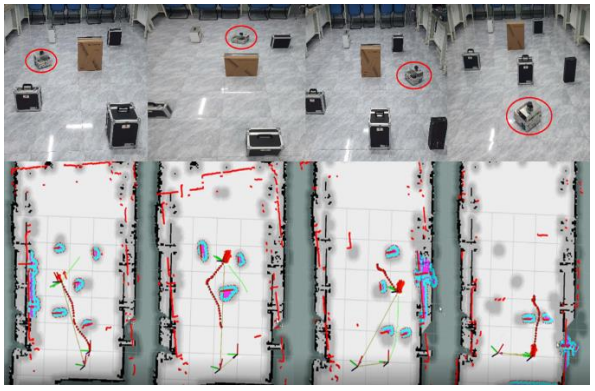

Figure 19. Robot navigation in real-world environment.

Table 7. Navigation performance for predefined goal locations.

Goal	Accuracy (%)	Time to goal (s)	Navigation outcome
Position 1	95	25	Smooth
Position 3	85	21	Wheel slip occurred in a narrow passage
Position 2	92	13	Heading deviation during goal reorientation
Position 4	90	20	Smooth

Overall, the proposed navigation system performs effectively in both obstacle-free and complex obstacle environments. The highest accuracy is obtained in open areas (up to 98%), whereas accuracy decreases to approximately 90% in narrow passages. Although the robot maintains high accuracy in most cases, deviations increase when traversing tight corners and during heading alignment toward the commanded goal. The LiDAR and IMU sensors provide reliable distance measurements and pose-related information; however, the control strategy and sensor configuration can be further improved to reduce deviation in constrained environments.

4. DISCUSSION

Previous studies on autonomous robots have mainly focused on using expensive robot platforms or complex sensing suites to achieve localization and mapping. In contrast, the proposed indoor AMR platform demonstrates that low-cost and simple hardware—including Raspberry Pi 4, LiDAR, IMU, and wheel encoders—can be effectively integrated with ROS to achieve high-performance mapping, probabilistic localization, and navigation in small- to medium-scale industrial environments, even under complex conditions. This approach not only reduces research and deployment costs, but also broadens applicability to laboratories and industries with limited budgets.

A notable contribution of this work is the intuitive user interface, which improves human–robot interaction, particularly in scenarios requiring flexible route changes or emergency handling. This is important because many prior works emphasize complex control architectures while paying less attention to usability and operator interaction.[15-16]

Our system differs clearly from related studies. For example, Zhou et al. (2022), in *"LiDAR-Based Mobile Mapping System for an Indoor*

Environment", used LiDAR to construct 2D maps for autonomous robots, but the system relied on relatively expensive hardware and sensors, required substantial computational resources, and may be difficult to deploy under budget constraints.[17] In contrast, our platform adopts a simplified hardware configuration that reduces cost while still achieving effective mapping, localization, and navigation in complex indoor environments.

Similarly, *"Autonomous Navigation System of Indoor Mobile Robots Using 2D LiDAR"* (Mathematics, 2023) presented an indoor SLAM-based navigation system using 2D LiDAR for data acquisition, mapping, and path planning.[18] The work emphasized RBPF-SLAM for robust localization in complex settings; however, it still relied on comparatively capable computing hardware and did not highlight low-cost implementation. In comparison, our design uses Raspberry Pi 4, a widely available low-cost platform that remains sufficiently capable of running ROS-based SLAM and navigation, demonstrating that accurate autonomous operation is achievable without high-end hardware.

Other studies, such as *"Research on SLAM Path Planning of ROS Robot based on LiDAR"* (2021) and (2023), have proposed strong SLAM systems using ROS, LiDAR, and IMU to optimize mapping and localization in complex environments.[19] Nevertheless, these systems generally do not emphasize user-interface optimization, focusing instead on map accuracy and navigation algorithm performance.

5. CONCLUSIONS AND FUTURE WORK

In this study, we designed and implemented an indoor autonomous mobile robot system based on ROS, integrating wheel encoders, an IMU, and a LiDAR sensor to realize key functionalities including odometry estimation, yaw/heading measurement, 2D map construction, probabilistic localization, and navigation in environments with and without obstacles. The system was evaluated through a set of real-world experiments targeting individual sensors and core tasks (mapping, localization, and navigation), thereby assessing both component-level performance and overall operational capability in indoor settings.

The main contributions of this work are as follows: (i) development of a differential-drive motion model and control mechanism based on encoder feedback, combined with a mathematical model for traveled-distance estimation; (ii) utilization of the IMU for yaw measurement and application of noise filtering to improve heading signal stability; (iii) use of LiDAR for distance sensing and 2D mapping, including verification of geometric map errors against ground-truth dimensions; (iv) implementation of AMCL localization using a particle filter and investigation of the impact of particle count (1000/5000/7000) on pose stability and computational cost; (v) navigation experiments in both obstacle-free and obstacle-rich scenarios, including parameter tuning (e.g., cost_factor, costmap parameters, and local-controller parameters) to improve trajectory quality and time-to-goal; and (vi) development of a user interface that enables selection of predefined goal locations and real-time monitoring of robot status, improving system usability and operator interaction.

Quantitatively, the encoder experiments show that the average error for 1–3 m travel distances lies in the range of 5.79% to 7.04%, indicating that encoder-based distance estimation is acceptable for indoor navigation tasks. For the IMU, raw yaw errors were as low as 0.44% in the tested rotations; after filtering, errors were reduced to approximately 0.56% down to 0%, confirming that noise filtering is necessary to improve heading reliability, especially for in-place rotations. For the LiDAR, distance-measurement errors from 0.4 m to 10 m ranged from 0.82% to 2.84%, and map dimension verification produced length and width errors of 1.29% and 0.81%, respectively, indicating that LiDAR measurements are sufficiently accurate for mapping in the experimental environment. Regarding AMCL, increasing the particle count from 1000 to 7000 improved pose convergence but increased processing time, highlighting the trade-off between localization accuracy and computational efficiency. At the system level, the robot successfully completed navigation scenarios with an overall stability of approximately 95%, and tuning the cost factor from 1.5 to 1.0 reduced overly conservative detours and improved time-to-goal.

Despite these promising results, several limitations remain. Encoder error tends to increase with distance due to wheel slip, accumulated error, and floor surface conditions. The IMU may be affected by slip and mechanical vibration, causing heading drift during rapid maneuvers if calibration and appropriate sensor fusion are not applied. LiDAR performance is constrained by surface reflectivity and scan-plane geometry; obstacles above or below the sensor height, or outside the effective sensing range, may not be fully detected. In addition, AMCL performance depends strongly on map quality, environmental features, and

parameter configuration; larger particle sets improve accuracy but increase computational load and may reduce update rates in dynamic scenarios.

For future work, localization and navigation robustness in more complex environments can be enhanced by implementing sensor fusion (e.g., fusing wheel odometry and IMU via an EKF) to mitigate slip effects and stabilize pose estimation over time. Moreover, a systematic parameter-tuning workflow for AMCL/costmap/local planner based on experimental data (including supervised optimization) can reduce reliance on manual tuning. Evaluation metrics should also be extended using standard robotics criteria such as success rate, time-to-goal, path length, number of replans, minimum obstacle clearance, and CPU/RAM utilization to provide more objective benchmarking. Algorithmically, alternative planners and local controllers can be tested for quantitative comparison, and handling of dynamic obstacles and "stuck" recovery can be added to improve robustness in real deployments. Finally, the user interface can be expanded to support multi-goal mission management, task history logging, and real-time system-health monitoring for indoor transport applications in corridors, classrooms, and warehouse-like environments.

In summary, this work demonstrates the feasibility of an indoor ROS-based autonomous robot system with an end-to-end pipeline comprising mapping, localization, navigation, and an operational user interface. Experimental results show that encoder-based odometry yields an average distance error of 5.79–7.04% over 1–3 m trajectories, while LiDAR ranging error remains below 3% up to 10 m. After AMCL parameter tuning (7000 particles), the robot achieved reliable goal-reaching performance in both obstacle-free and obstacle-rich indoor scenarios, with reduced deviations when costmap cost_factor was set to 1.0. These results validate the feasibility of a low-cost ROS-based indoor AMR and provide a reproducible baseline for further robustness improvements in narrow passages and during heading reorientation.

Acknowledgments

REFERENCES

1. H. Durrant-Whyte, T. Bailey. Simultaneous Localization and Mapping: Part I, *IEEE Robotics & Automation Magazine*, **2006**, *13*, 99-110.
2. Feder, H. J. S., Leonard, J. J., & Smith, C. M. Adaptive mobile robot navigation and mapping. *The International Journal of Robotics Research,* **1999**, *Volume* 18, 650-668.
3. J. J. Leonard and H. F. Durrant-Whyte. *Simultaneous map building and localization for an autonomous mobile robot*, in Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, Osaka, Japan, **1991**.
4. S. Thrun, W. Burgard, and D. Fox. Probabilistic Robotics, *MIT Press*, **2005**.
5. J. Zhu, J. et al., Camera, LiDAR, and IMU Based Multi-Sensor Fusion SLAM: A Survey, *IEEE Transactions on Instrumentation and Measurement,* **2024***, 73,* 1-22.
6. X. Yue, Y. Zhang, J. Chen, J. Chen, X. Zhou, and M. He. LiDAR-based SLAM for robotic mapping: state of the art and new frontiers, *Industrial Robot: the international journal of robotics research and application,* **2024***, 51,* 196-205.
7. W. Chen, W. Chi, S. Ji, H. Ye, J. Liu, Y. Jia, J. Yu and J. Cheng. A survey of autonomous robots and multi-robot navigation: Perception, planning and collaboration, *Biomimetic Intelligence and Robotics,* **2025***, 5,* 100203.
8. N. AbuJabal, M. Baziyad, R. Fareh, B. Brahmi, T. Rabie, and M. Bettayeb. A Comprehensive Study of Recent Path-Planning Techniques in Dynamic Environments for Autonomous Robots, *Sensors*, **2024**, *24*, 8089.
9. P. Chen, H. Liu and Z. Qiao. A review of research on SLAM technology based on the fusion of LiDAR and vision, *Sensors*, **2025**, *25*, 1447.
10. Y. Bai, Z. Wang, and H. Xu. Path planning for mobile robots based on A* algorithm, *Journal of Robotics and Automation*, **2017**, *34*, 169-177.
11. K. Cai, C. Wang, J. Cheng, C. W. De Silva, and M. Q.-H. Meng. Mobile Robot Path Planning in Dynamic Environments: A Survey, *arXiv*, **2021**.
12. K. Trejos, L. Rincón, M. Bolaños, J. Fallas and L. Marín. 2D SLAM algorithms characterization, calibration, and comparison considering pose error, map accuracy as well as CPU and memory usage, *Sensors*, **2022**, *22*, 6903.
13. F. G. Sayyad and D. Salunke. *Autonomous mobile robot base: cost effective solution for ROS-based navigation*, in Proceedings of the Advances in Robotics (AIR) Conference, New York, NY, USA, **2023**.
14. Z. Wang, H. Tu, S. Chan, C. Huang and Y. Zhao. Vision-based initial localization of AGV and path planning with PO-JPS algorithm, *Egyptian Informatics Journal*, **2024**, *27*, 100527.
15. W. Xu, Y. Cai, D. He, J. Lin and F. Zhang. FAST-LIO2: Fast Direct LiDAR-Inertial Odometry, *IEEE Transactions on Robotics*, **2022**, *38*, 2053-2073.
16. Y. Hu, F. Xie, J. Yang, J. Zhao, Q. Mao, F. Zhao, and X. Liu. Efficient Path Planning Algorithm Based on Laser SLAM and an Optimized Visibility Graph for Robots, *Remote Sensing*, **2024**, *16*, 2938.
17. J. Sun, J. Zhao, X. Hu, H. Gao and J. Yu. Autonomous Navigation System of Indoor Mobile Robots Using 2D LiDAR, *Mathematics*, **2023**, *11*, 1455.

18. H. Hu, L. Sun and H. Xu. Research on SLAM path planning of ROS robot based on LiDAR, *in Highlights in Science, Engineering and Technology*, **2022**, *24*, 179-181.

19. J. M. Santos, D. Portugal and R. P. Rocha. An Evaluation of 2D SLAM Techniques Available in Robot Operating System, *in IEEE Int. Symp. Safety, Security, and Rescue Robotics (SSRR)*, **2013**.