Implementations of SLAM and A* for a Wheeled Robot

Xinyu Xu, Haowei Li, Jinjia Guo

***Abstract*—This work presents an autonomous mobile robot system implemented on the differential drive MBot platform equipped with 2D LIDAR, wheel encoders, an IMU, and a camera. The system achieves precise motion control using PID feedback and a carrot-following algorithm, robust mapping via particle filter-based SLAM with occupancy grids, and autonomous exploration through A* path planning and frontier-based strategies. Additionally, a YOLO model was trained to detect cones and blocks for task execution. Experimental results demonstrated accurate mapping and autonomous navigation, with average pose errors under 1 cm in position and 3° in orientation, highlighting its practical utility for robotics applications.**

I. Introduction

AUTONOMOUS mobile robots face significant challenges when navigating unknown environments, particularly in simultaneously constructing accurate maps and estimating their pose—a problem known as Simultaneous Localization and Mapping (SLAM). This project leverages a differential drive robot equipped with 2D LIDAR, wheel encoders, an IMU, and a camera to develop an autonomous navigation system addressing key challenges in motion control, mapping, and path planning. The system integrates PID feedback control with sensor fusion for precise motion control, a particle filter-based SLAM algorithm with occupancy grid mapping, and A* and frontier-based exploration techniques for efficient navigation.

Key challenges included enhancing odometry estimation through sensor fusion, optimizing the particle filter SLAM algorithm to meet computational constraints while maintaining accuracy, and implementing a robust path-planning algorithm capable of determining optimal, collision-free paths. Experimental trials in progressively complex environments demonstrated the system's ability to generate accurate maps and navigate autonomously. This report details the technical approaches employed and evaluates the results obtained through experimental testing [1].

II. Methodology

A. Odometry and PID Control

Accurate motion control and pose estimation are essential for autonomous navigation. This system integrates wheel encoder feedback with IMU data for robust state estimation and employs a cascaded controller for precise velocity control, supported by calibration, odometry, and motion planning.

1) Calibration: For the differential drive system, a piecewise linear model was developed to characterize the relationship between the PWM duty cycle and motor velocity. This approach allowed for accurate modeling of the motor's dynamic behavior across different operating ranges, ensuring precise control of the robot's motion:

$$u_{ff} = \begin{cases} m_p v + b_p, & v \geq 0 \\ m_n v + b_n, & v < 0 \end{cases} \tag{1}$$

where $m_{p,n}$ represent directional slopes and $b_{p,n}$ characterize static friction effects, u_{ff} is the feed forward PWM.

2) Odometry: The robot pose estimation combines wheel encoder measurements with IMU data in a dead reckoning framework. For the differential drive configuration, linear and angular velocities are computed from wheel velocities:

$$v_x = R(\omega_L + \omega_R)/2 \tag{2}$$

$$\omega_z = R(\omega_R - \omega_L)/b \tag{3}$$

where R represents wheel radius and b denotes wheelbase distance. The pose update equations follow:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + v_x \cos(\theta) dt \\ y_{t-1} + v_x \sin(\theta) dt \\ \theta_{t-1} + \omega_z dt \end{bmatrix} \tag{4}$$

The odometry data for the robot's heading θ lack precision, while the IMU data suffer from drift over time. To improve the accuracy of heading estimation, a gyrodometry fusion algorithm was developed. This algorithm combines the strengths of odometry and IMU data, leveraging the consistency of odometry for long-term stability and the responsiveness of the IMU for short-term corrections, effectively mitigating their individual weaknesses:

$$\Delta\theta_{\text{G-O}} = \Delta\theta_{\text{gyro}} - \Delta\theta_{\text{odom}} \tag{5}$$

$$\theta_t = \begin{cases} \theta_{t-1} + \Delta\theta_{\text{gyro}}, & |\Delta\theta_{\text{G-O}}| > \theta_{\text{threshold}} \\ \theta_{t-1} + \Delta\theta_{\text{odom}}, & \text{otherwise} \end{cases} \tag{6}$$

3) Motor Control: The motor control system consists of two components: feed forward and feedback. Initially, the command is provided in terms of the robot's linear velocity v and angular velocity ω. These whole-body velocity commands are then converted into the angular velocities of the individual wheels using the robot's kinematic equations:

$$\omega_R = \frac{v + \frac{b}{2}\omega}{r_w} \tag{7}$$

$$\omega_L = \frac{v - \frac{b}{2}\omega}{r_w} \tag{8}$$

Where r_w represents the radius of the wheel, the feed-forward PWM value is subsequently calculated using the calibration equation in Equation 1. These PWM signals are then independently applied to the motors driving the two wheels.

Subsequently, the actual speeds of the two wheels are determined by reading the odometry data. The actual velocities are then compared to the desired velocities, and the resulting error is calculated. A PID controller is used to compensate for this error, adjusting the command to ensure the actual velocities match the desired values.

$$v_{\text{compensate}} = K_p e_v + K_i \int e_v \, dt + K_d \frac{de_v}{dt} \tag{9}$$

$$\omega_{\text{compensate}} = K_p e_\omega + K_i \int e_\omega \, dt + K_d \frac{de_\omega}{dt} \tag{10}$$

Simultaneously, the velocity mismatch for each wheel is calculated individually. The error e , is defined as the difference between the commanded velocity and the actual wheel velocity. A PID controller is then applied to e to compute the feedback PWM, which is directly added to adjust the motor control signals.

$$u_{\text{fb}} = K_p e + K_i \int e \, dt + K_d \frac{de}{dt} \tag{11}$$

Afterward, the feedback PWM and feedforward PWM are combined to determine the final PWM input for each wheel individually. This process runs continuously in a loop as long as the robot is operational.

$$u_{\text{PWM}} = u_{\text{ff}} + u_{\text{fb}} \tag{12}$$

This firmware control architecture guarantees more precise velocity tracking while maintaining stable operation across the robot's operating range.

B. Motion Control

The high-level motion control system implements a control strategy for waypoint navigation. Carrot-following algorithm was used as motion controller because it can follow the waypoints more smooth and effective. It is designed to guide the robot toward a "carrot" point, which dynamically adjusts based on the robot's current position and a lookahead distance. The algorithm ensures smooth and stable navigation by considering both the current pose of the robot and the target destination.

The carrot-following algorithm calculates the control commands required to steer the robot towards the target pose. The key components of the algorithm include:

- **Lookahead Distance:** Determines the position of the "carrot" point along the path, ensuring the robot has a smooth trajectory.
- **Carrot Point Calculation:** Computes a point on the path (the carrot) that lies ahead of the robot, taking into account the robot's current position and orientation.
- **Velocity Commands:** Generates forward and angular velocity commands (v and ω) based on the robot's position relative to the carrot point.
- **Target Reached Check:** Evaluates whether the robot has reached its target pose, considering specified tolerances.

The carrot point is determined based on the robot's current pose (x, y, θ), the target pose (x_t, y_t, θ_t), and the lookahead distance $d_{\text{lookahead}}$. First, the displacement vector from the robot to the target is computed as:

$$dx = x_t - x, \quad dy = y_t - y. \tag{13}$$

Next, the distance to the target and the angle to the target are calculated using:

$$d_{\text{target}} = \sqrt{dx^2 + dy^2}, \quad \theta_{\text{target}} = \text{atan2}(dy, dx). \tag{14}$$

If the distance to the target exceeds the lookahead distance, the carrot point is computed as:

$$x_c = x + d_{\text{lookahead}} \cdot \frac{dx}{d_{\text{target}}}, \quad y_c = y + d_{\text{lookahead}} \cdot \frac{dy}{d_{\text{target}}}. \tag{15}$$

Otherwise, the carrot point is set directly to the target position. The control commands are derived from the distance and angle to the carrot point. The forward distance d_{fwd} and angle to the carrot point θ_c are calculated as:

$$d_{\text{fwd}} = \sqrt{(x_c - x)^2 + (y_c - y)^2}, \tag{16}$$

$$\theta_c = \text{atan2}(y_c - y, x_c - x). \tag{17}$$

The angular error α and the heading error β are computed as:

$$\alpha = \theta_c - \theta, \quad \beta = \theta_t - (\alpha + \theta). \tag{18}$$

Using these errors, the forward velocity v and angular velocity ω are determined through PID control:

$$v = \max(0.4, K_p \cdot d_{\text{fwd}}), \tag{19}$$

$$\omega = K_\alpha \cdot \alpha + K_\beta \cdot \beta. \tag{20}$$

If the angular error α exceeds 60 degrees, the forward velocity will be set to zero to avoid overshoot.

d_{fwd} represents the Euclidean distance to the target, α the angular difference, and β the target orientation error. Controller gains $[k_p, k_\alpha, k_\beta] = [4.0, 5.5, 0.0]$ were tuned for balanced speed and stability. The task is marked complete upon reaching the target pose, ensuring efficient and smooth navigation even with abrupt path changes.

C. Simultaneous Localization and Mapping (SLAM)

The SLAM framework consists of a mapping module for environment representation and a localization module that uses particle filtering with an action and sensor model to estimate the robot's position.

1) Mapping: The mapping subsystem uses an occupancy grid representation with a log-odds framework to model the environment accurately. To account for robot motion during scanning, a two-stage approach combines motion compensation and probabilistic occupancy updates.

In the first stage, raw laser data is processed through a moving laser scan framework that interpolates poses between scan endpoints and computes adjusted rays, ensuring accurate spatial representation.

Algorithm 1 Bresenham's Line Algorithm

Require: Start (x_0, y_0), End (x_1, y_1), Map
Ensure: List of traversed cells

function BRESENHAM((x_0, y_0), (x_1, y_1))
  $dx, dy \leftarrow |x_1 - x_0|, |y_1 - y_0|$, $sx, sy \leftarrow \text{sign}(x_1 - x_0), \text{sign}(y_1 - y_0)$
  $err \leftarrow dx - dy$, $(x, y) \leftarrow (x_0, y_0)$, $points \leftarrow \{(x, y)\}$
  while $(x, y) \neq (x_1, y_1)$ **do**
    $e2 \leftarrow 2 \cdot err$
    if $e2 > -dy$ **then** $err \leftarrow err - dy$, $x \leftarrow x + sx$
    if $e2 < dx$ **then** $err \leftarrow err + dx$, $y \leftarrow y + sy$
    Append (x, y) to $points$
  return $points$

The second stage applies log-odds-based occupancy grid mapping with values ranging from -128 to 127, ensuring efficient updates and numerical stability. Map updates involve two processes: endpoint scoring, which adjusts the log-odds of cells where laser rays terminate to indicate occupancy, and ray tracing, which uses Bresenham's line algorithm (Alg. 1) to mark traversed cells as free space based on a miss odds parameter.

This dual-update method ensures consistency between free and occupied space estimations while processing sensor data efficiently, enabling robust and accurate mapping of the environment.

2) Monte Carlo Localization: The localization algorithm utilizes Monte Carlo Localization within a particle filter framework to estimate and maintain multiple hypotheses about the robot's pose. This approach recursively estimates the posterior probability distribution of the robot's state by propagating and updating a set of weighted particles, guided by action and sensor models. The particle filter's capability to represent non-Gaussian and multi-modal distributions makes it highly effective for both global localization and position tracking, ensuring robust performance within the SLAM system.

a) Particle Filter: To outline the complete process of updating the particle filter, the previous set of particles (*posterior*) is first resampled using the low variance sampling algorithm (Algorithm 2), transforming them into the new *prior*. Next, the action model is applied to generate the distributed particles (*proposal*). Subsequently, the sensor model is used to compute the weight of each particle, updating the *posterior*. Finally, the robot's SLAM pose is estimated based on the updated *posterior*.

Algorithm 2 Low Variance Sampling Algorithm

Require: Number of particles N, List of particles with weights $P = \{p_1, p_2, \ldots, p_N\}$
Ensure: Resampled list of particles S

$S \leftarrow \emptyset$ ▷ Initialize an empty list for samples
Generate a random number $r \sim \mathcal{U}(0, \frac{1}{N})$
$idx \leftarrow 0$, $s \leftarrow P[idx]$.weight ▷ Initialize index and cumulative weight
for $i = 0$ to $N - 1$ **do**
  $u \leftarrow r + i \cdot \frac{1}{N}$ ▷ Calculate uniform step
  while $u > s$ and $idx < N - 1$ **do**
    $idx \leftarrow idx + 1$
    $s \leftarrow s + P[idx]$.weight ▷ Increment index and update cumulative weight
  Add $P[idx]$ to S ▷ Add selected particle to the sample list
return S

b) Action Model: The action model is responsible for estimating the motion of the robot between consecutive timesteps and sampling new particle poses based on odometry data. It uses a probabilistic motion

model that decomposes the robot's movement into three components:

- **Rotation 1 (rot$_1$)**: The initial rotation required to align the robot with the direction of motion.
- **Translation (trans)**: The linear motion in the aligned direction.
- **Rotation 2 (rot$_2$)**: The final rotation to achieve the target orientation.

This decomposition ensures that the model captures the uncertainty in each component of motion, providing a robust representation of the robot's action.

$$\delta_{\text{rot1}} = \text{atan2}(\Delta y, \Delta x) - \theta_{t-1} \tag{21}$$

$$\delta_{\text{trans}} = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{22}$$

$$\delta_{\text{rot2}} = \Delta\theta - \delta_{\text{rot1}} \tag{23}$$

where Δx, Δy, and $\Delta\theta$ represent odometry changes between timesteps. For each particle, the model samples new poses by adding noise proportional to the magnitude of motion:

$$x'_t = x_t + \delta'_{\text{trans}} \cdot \cos(\theta_t + \delta'_{\text{rot1}}) \tag{24}$$

$$y'_t = y_t + \delta'_{\text{trans}} \cdot \sin(\theta_t + \delta'_{\text{rot1}}) \tag{25}$$

$$\theta'_t = \theta_t + \delta'_{\text{rot1}} + \delta'_{\text{rot2}} \tag{26}$$

where:

$$\delta'_{\text{rot1}} \sim \mathcal{N}(\delta_{\text{rot1}}, k_1|\delta_{\text{rot1}}|) \tag{27}$$

$$\delta'_{\text{trans}} \sim \mathcal{N}(\delta_{\text{trans}}, k_2|\delta_{\text{trans}}|) \tag{28}$$

$$\delta'_{\text{rot2}} \sim \mathcal{N}(\delta_{\text{rot2}}, k_1|\delta_{\text{rot2}}|) \tag{29}$$

The uncertainty parameters used in the action model are listed in Table I. These values were chosen through empirical testing to balance accuracy and computational efficiency. The parameters k_1 and k_2 control the variance in rotational and translational components, respectively, while min_dist and min_theta define thresholds for detecting significant motion.

TABLE I: Action Model Parameters

Parameter	Value	Description
k_1	0.02	rotational uncertainty
k_2	0.005	translational uncertainty
min_dist	0.01	distance threshold
min_theta	0.01	angular threshold

The action model updates the particle poses as follows:

1) **Motion Decomposition:** Compute the motion components (rot$_1$, trans, and rot$_2$) based on the difference between current and previous odometry readings.
2) **Motion Uncertainty:** Calculate the standard deviations (σ_{rot_1}, σ_{trans}, σ_{rot_2}) using the equations above.
3) **Pose Sampling:** Sample new poses from a normal distribution for each motion component and update the particle states.

This probabilistic motion model allows the robot to account for noise and uncertainty in its odometry, ensuring robust performance in dynamic environments.

c) Sensor Model: The sensor model is a critical component of the SLAM framework, responsible for evaluating the likelihood of a particle's pose based on observed laser scan data and the occupancy grid map. The model employs a probabilistic approach to compute the alignment between laser measurements and the environment representation, incorporating ray endpoints, map features, and uncertainty.

For each laser measurement z_t, the likelihood is computed as:

$$p(z_t \mid x_t, m) = \frac{1}{\sqrt{2\pi}\sigma_{\text{hit}}} \exp\left(-\frac{d^2}{2\sigma_{\text{hit}}^2}\right) \cdot \text{offset} \tag{30}$$

where d is the Euclidean distance between the measured endpoint and its nearest occupied cell in the map, σ is the measurement uncertainty parameter, and offset is a constant to refine the equation. The Bread-First-Search(BFS) algorithm is employed to locate the nearest occupied cell to a given ray endpoint. Precomputed offsets ensure efficient traversal of neighboring cells. If the BFS search fails to locate an occupied cell, the endpoint is defaulted to a distant location to avoid incorrect scoring. The total measurement likelihood which is the weight for a particle W is the product of individual ray likelihoods:

$$W(x_t) = \prod_{k=1}^{K} p(z_t^k \mid x_t, m) \tag{31}$$

To improve computational efficiency while maintaining accuracy:

- Implement an efficient breadth-first search within a configurable radius (`search_range` = 10) to find nearest occupied cells, if it is larger than 10, it will be given a pretty large distance.
- Apply an occupancy threshold (`threshold` = 20) to determine cell occupancy.

The sensor model's probabilistic evaluation of ray alignment with the map enhances the particle filter's accuracy. It provides robust weighting for particles, even in noisy environments, enabling the SLAM system to reliably estimate the robot's pose based on laser scans and map consistency.

3) SLAM: The complete SLAM system integration follows a cyclical workflow as illustrated in the system architecture as shown in Figure 1). The posterior distribution over robot pose x_t and map m is factored according to:

$$p(x_t, m \mid z_{1:t}, u_{1:t}) = p(x_t \mid z_{1:t}, u_{1:t}) \cdot p(m \mid x_t, z_{1:t}) \tag{32}$$

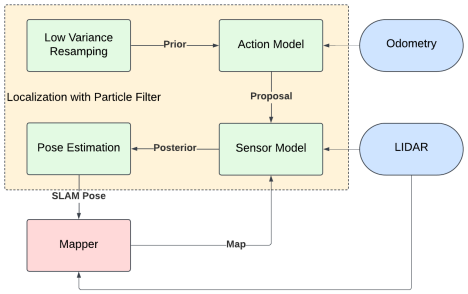


Fig. 1: SLAM System Diagram

The workflow begins with odometry data feeding into the action model, while low-variance resampling of the particle filter provides the prior distribution. The action model generates a proposal distribution of possible robot poses, which is refined by the sensor model using LIDAR scan data and the current map state to evaluate particle weights. These weighted particles form the posterior distribution, from which the current SLAM pose is estimated.

This estimated pose, combined with LIDAR measurements, updates the occupancy grid map. The updated map then informs the sensor model in the next iteration, creating a continuous refinement loop where improved localization enhances mapping accuracy, better maps improve sensor model evaluations, and refined sensor evaluations yield more accurate pose estimates.

To ensure computational efficiency, the system applies selective particle reinvigoration when distribution quality drops below 0.15 and processes laser scans with configurable stride parameters. This integrated approach enables robust, real-time performance in dynamic environments, effectively handling sensor noise and obstacles.

D. Path Planning

1) A Algorithm:* Path planning implementation utilizes an A* search algorithm enhanced with obstacle distance constraints for safe and efficient trajectory generation. The planner operates on an 8-connected grid representation where each node n maintains both path cost $g(n)$ and heuristic estimate $h(n)$:

$$f(n) = g(n) + h(n) \tag{33}$$

The path cost $g(n)$ incorporates both motion and safety considerations through a dual-component structure:

$$g(n) = g_{\text{motion}}(n) + g_{\text{safety}}(n) \tag{34}$$

where g_{motion} defines base movement costs:

$$g_{\text{motion}}(n) = \begin{cases} 1.0, & \text{for cardinal moves} \\ \sqrt{2}, & \text{for diagonal moves} \\ 10^{16}, & \text{for invalid configurations.} \end{cases} \tag{35}$$

The g_{safety} component enforces obstacle avoidance using a distance grid:

$$g_{\text{safety}}(n) = \begin{cases} (d_{\max} - d_{\text{obs}})^{\alpha}, & \text{if } d_{\text{obs}} < d_{\max} \\ 10^{16}, & \text{if } d_{\text{obs}} < r_{\text{robot}} \\ 0, & \text{otherwise.} \end{cases} \tag{36}$$

where $d_{\max}$ represents *maxDistanceWithCost* and r_{robot} is *minDistanceToObstacle*.

The heuristic function implements an admissible diagonal distance metric:

$$h(n) = \sqrt{(x_{\text{goal}} - x_n)^2 + (y_{\text{goal}} - y_n)^2}. \tag{37}$$

Node expansion explores 8 neighboring directions:

$$\text{moves} = \{(0, \pm 1), (\pm 1, 0), (\pm 1, \pm 1)\}, \tag{38}$$

with validity constraints:

- Node within grid boundaries
- $d_{\text{obs}}(n) > r_{\text{robot}}$

The implementation utilizes a priority queue for $O(\log n)$ node operations, with pruning to remove redundant waypoints where consecutive direction vectors $(dx_1, dy_1), (dx_2, dy_2)$ satisfy:

$$dx_1 \cdot dy_2 = dy_1 \cdot dx_2. \tag{39}$$

Path validation ensures:

- Start/goal pose validity
- Minimum obstacle clearance
- Configuration space reachability

This approach enables computationally efficient path planning while maintaining strict safety constraints for autonomous navigation through partially mapped environments.

E. Exploration and Relocalization

The exploration and relocalization capabilities enable the robot to perform more advanced tasks. It can autonomously explore an unknown map and accurately relocalize its pose when reintroduced to the map at a random position after leaving.

1) Exploration: A frontier-based exploration strategy that systematically maps unknown environments through efficient frontier detection and prioritized exploration was implemented. The system operates through a finite state machine architecture that manages the exploration process from initialization through completion.

To locate the nearest frontier, a BFS was performed starting from the robot's current position, identifying frontier cells where free space neighbors occupied unexplored areas. Frontiers were expanded using an 8-connected component search to ensure connectivity, and their centroids were calculated for goal selection. Among all frontiers, the closest centroid was chosen based on

Euclidean distance, and a path was planned to the nearest navigable cell using a motion planner. This iterative process enabled efficient exploration of the map.

After identifying the closest frontier, the robot navigates to the target position using a path generated by the A* algorithm. Upon reaching the target, the robot pauses for 10 seconds to ensure sufficient scanning of the area. This process is repeated for all frontiers until the entire map has been explored. Once all frontiers are covered, the robot transitions to the "RETURNING HOME" state and navigates back to the origin $(0, 0)$.

2) Relocalization: Relocalization allows the robot to determine its current pose from a random position within a known map. To implement relocalization, the initial pose of the SLAM system is reset. Initially, 10,000 particles are randomly generated across the map. If a particle is not located in free space, it is regenerated to ensure all particles are placed within valid free space regions.

The particles are then updated using low-variance sampling and the sensor model. After a short period, the particles converge to the correct position. Once convergence is achieved, the estimated current position is recorded into a text file. Subsequently, SLAM mode is reactivated, and the stored position is loaded as the initial pose for further operation. This process ensures seamless relocalization and allows the robot to resume normal SLAM functionality.

III. Results

A. Odometry and PID Control

1) Calibration: Parameter estimation was conducted through systematic trials on a concrete surface five times, yielding statistical variations as shown in Table II.

TABLE II: Motor Calibration Parameters

Parameter	Mean	Variance
Positive Slope Left	0.071	1.93e-6
Positive Slope Right	0.065	3.96e-6
Positive Intercept Left	0.065	5.89e-5
Positive Intercept Right	0.071	2.00e-4
Negative Slope Left	0.066	4.28e-6
Negative Slope Right	0.067	2.15e-5
Negative Intercept Left	-0.070	1.30e-4
Negative Intercept Right	-0.070	2.52e-4

The observed difference in variance between the slope and intercept parameters suggests that static friction introduces nonlinearities that are inherently more variable compared to the relatively predictable dynamic response of the system. Static friction effects, such as stiction, can cause abrupt changes in motor behavior, particularly at low speeds or during transitions from rest to motion. This contrasts with the dynamic response, which is governed largely by the motor's mechanical and electrical properties, leading to a more linear and stable relationship. Other potential sources of variation could include inconsistencies in the motor's internal mechanisms, variations in the surface conditions of the concrete floor, or slight differences in environmental factors, such as temperature or humidity, which may influence the system's performance.

2) Odometry: To evaluate odometry performance, the robot was subjected to two test scenarios: linear and rotational movements. In the first test, the robot was pushed one meter along a straight line in one meter. In the second test, the robot was pushed to execute a 360-degree rotation in place, the result was shown in Table III. After fusing the two data sources, the system achieves a reliable level of accuracy, making it well-suited for subsequent navigation tasks.

TABLE III: Evaluation of Odometry Performance

Parameter	Mean Error (m)
Distance difference in x direction	0.12 (m)
Angular error (before optimization)	0.8 (rad)
Angular error (after optimization)	0.02 (rad)

3) Motor Control: The final controller for the robot integrates both feedforward and feedback control mechanisms to ensure precise motion tracking and overall stability. When commanded to move along a straight line, the robot demonstrated exceptional accuracy, maintaining its trajectory with no noticeable deviation. Additionally, during transitions between high and low speeds, the robot exhibited smooth acceleration and deceleration without any oscillations or instability.

The control system in firmware parameters are summarized in Table IV.

TABLE IV: Control System Parameters

Parameter	K_p	K_i	K_d
angular velocity	0.12	0.00	0.000
left wheel	0.11	0.10	0.001
right wheel	0.12	0.09	0.001

The tuning process for the left and right wheels was performed separately, with the primary evaluation metric being the robot's ability to travel in a straight line without deviation. The process began by incrementally increasing the proportional gain (K_p) while monitoring the robot's behavior, stopping just before oscillations occurred to maintain stability. Once an appropriate K_p was established, the integral gain (K_i) was introduced to minimize steady-state error, with care taken to prevent excessive values that could cause slow responses or oscillations. Finally, a small derivative gain (K_d) was applied to dampen overshoot and enhance overall stability.

This iterative tuning approach ensured that the robot could maintain its trajectory and respond appropriately

to velocity changes. To address potential issues such as sudden starts or stops, additional features were implemented:

- **Acceleration and Deceleration Control**: To prevent abrupt changes in motion, the current PWM input was compared with the previous PWM. If the difference exceeded 0.05, it was capped at 0.05 to ensure a gradual change in speed.
- **Feedback Control Output Limiting**: The feedback control signal (u_{fb}) was constrained within a safe range to avoid overshoot and maintain stable operation.

B. Motion Control

The carrot-following algorithm ensures highly reliable navigation between waypoints. As demonstrated in Figure 2, the robot completes four loops of a square trajectory with significant overlap, indicating precise path-following performance.

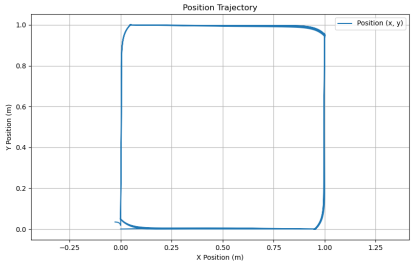


Fig. 2: Pose of Driving Square

Additionally, the algorithm facilitates smooth motion. As shown in Figure 3, the robot maintains a nearly constant speed throughout the square trajectory, demonstrating the algorithm's ability to provide stable and consistent velocity control.

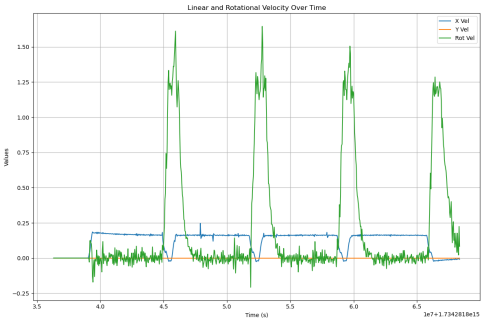


Fig. 3: Velocity of Driving Square

C. Simultaneous Localization and Mapping (SLAM)

1) Mapping: The quality of the map was evaluated using the drive maze log. As illustrated in Figure 4, the map demonstrates high accuracy, with all obstacles clearly and explicitly represented.

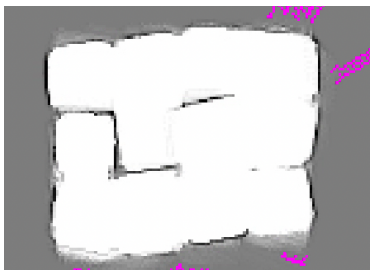


Fig. 4: Map in Drive Maze Log

2) Monte Carlo Localization: The particle filter update times are presented in Table V. Given that the lidar operates at an update rate of 7Hz, the maximum allowable update time for the particle filter corresponds to this frequency. Based on the data in the table, it can be inferred that if the lidar operates at a higher frequency, the particle filter could achieve an update rate of 10Hz with approximately 350 particles.

TABLE V: Particle Filter Update Times

Number of Particles	Update Time (Hz)
100	7
500	6
1000	3

The Figure 5 shows 300 particles along the path recorded in the drive square log. The blue line represents the SLAM pose, while the red points indicate the particle distribution at various locations along the path. The particles are concentrated around the estimated pose, reflecting the effectiveness of the particle filter in maintaining localization accuracy. The consistent particle spread throughout the trajectory demonstrates robust state estimation as the robot moves along the square path.

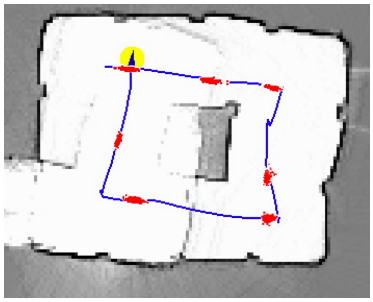


Fig. 5: Particles Along the Path in Drive Square Log

3) SLAM: In Figure 6, the blue line represents the SLAM pose, while the orange line denotes the ground truth. The close alignment between the two demonstrates the accuracy and effectiveness of the SLAM system.

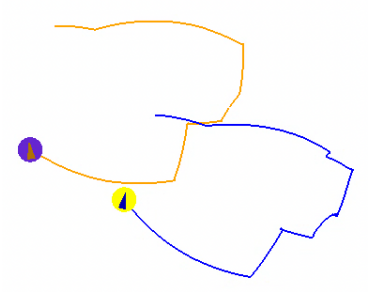


Fig. 6: Pose Compare in Drive Maze Full Rays Log

To quantitatively evaluate the performance of the SLAM pose, it was compared with the ground truth. However, due to corruption in the ground truth data, the SLAM pose from the log file was instead compared with the SLAM pose generated by the implemented particle filter. The RMS error was calculated by summing the differences in x, y, and θ at corresponding timestamps. The linear RMS error was determined to be **0.17 m** and the rotational RMS error was determined to be **0.13 rads**.

D. Path Planning

1) A Algorithm:* As illustrated in Figure 7, the blue line represents the pruned waypoints, while the purple line indicates the actual SLAM pose. It is evident that the A* algorithm successfully identifies a valid path to the target while maintaining a safe distance from obstacles. Furthermore, the robot is able to follow the generated waypoints accurately.

Fig. 7: Implementation of A* algorithm

The execution times for the A* path planning algorithm were evaluated using the `astar_test` for different grid scenarios. The results include both failed and successful planning attempts, with key statistics such as minimum, mean, maximum, median, and standard deviation. The summary of results is shown in Figure 8.

The results demonstrate that A* path planning execution times vary significantly with grid complexity. Simpler grids exhibit low update times, reflecting the algorithm's efficiency in open spaces. In contrast, complex scenarios show significantly higher execution times and standard deviations due to increased search space, heuristic limitations, and higher grid resolution. To address these issues and improve performance, strategies such as implementing advanced heuristics, dynamically adjusting grid resolution, pruning unnecessary waypoints, and integrating hybrid approaches. These enhancements can make the algorithm more efficient, especially in constrained and maze-like environments.

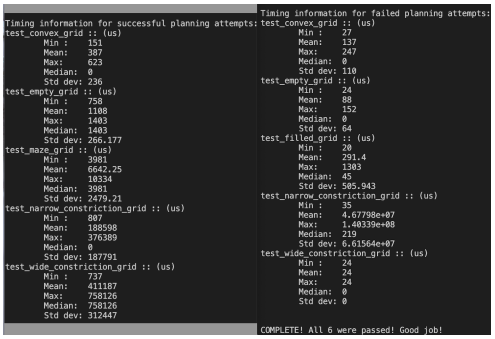
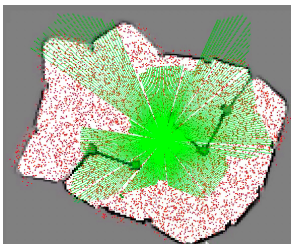


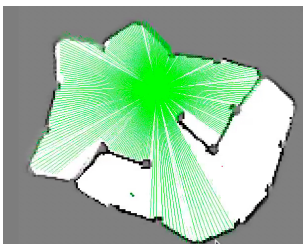
Fig. 8: Test of A* algorithm

E. Exploration and Relocalization

1) Relocalization: As shown in Figure 9a, 10,000 particles are successfully generated within the free spaces of the map. After a few seconds, the particles converge to the correct position, as illustrated in Figure 9b.



(a) Particles in Random



(b) Initial Pose

Fig. 9: Process of Relocalization

IV. Competition Results

We secured first place in our session during the final competition.

*a) **Speed Run**:* In the Speed Run task, we completed two loops along the designated route in 42 seconds. The pose error was maintained within 3 cm and 5 degrees, and the map quality was excellent. We achieved full marks in this section.

*b) **Maze Explorer**:* In the Maze Explorer task, we scored 500 points by successfully exploring the entire map with excellent map quality. Following the exploration phase, the robot was placed at a random pose within the map. It successfully relocalized itself and returned to the home position with an error of less than 3 cm and 5 degrees.

REFERENCES

[1] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.