ROB 550 Report: Autonomous Navigation for Mobile Robot

Sandilya Sai Garimella, Xun Yang, Yuzhou Chen
{garimell, xunyang, yzc}@umich.edu

***Abstract*—The Botlab project aims to develop software and hardware for an MBot, a two-wheeled differential drive robot, to facilitate autonomous mapping, SLAM-based localization, path planning, and block transport. We achieve this by implementing motor control via PID tuning and odometry, using a particle filter and mapping algorithms, and adding path planning capabilities. Additionally, we design and construct a forklift attachment to facilitate the movement of blocks. We conduct performance evaluations to assess the effectiveness of the system's components.**

I. Introduction

This lab report describes the use of a classic differential drive Mbot equipped with two parallel wheels and a rear caster. Modifications include attaching a forklift for block handling. The robot features a 2D-LIDAR for environment sensing, utilizing its rays for SLAM. Pose estimation is achieved through a 3-DOF odometry system, which incorporates motor encoders and an IMU. Development is done via remote SSH access, and data transfer occurs through LCM communication. The system incorporates a Jetson Nano board for high-level connectivity between the sensory tools and the MBot's firmware. The Pico+ Controller manages the firmware, controlling motors and reading IMU data.

The project enhances firmware for better odometry and motor control, along with autonomy features like optimal pathfinding using A* search, SLAM, and motion control. The report details the methods implemented for these enhancements, their outcomes, and the conclusions drawn from these results.

II. Methodology

This section outlines the theoretical foundations behind the control system, SLAM model, and exploration strategies.

A. Motion Controller

1) **Wheel Speed Computation:** The robot's mobility is defined by the velocity of its wheels, which are regulated via PWM signals to the motors. Initially, to manipulate the robot's movement using open-loop control, the PWM inputs to the motors are set according to a linear relationship with the desired wheel velocity:

$$\text{pwm} = a \cdot \text{speed} + b \tag{1}$$

However, we apply Proportional-Integral-Derivative (PID) gains to make it into a closed-loop control system, explained in the upcoming subsections.

2) **Odometry:** Odometry calculates changes in position using data from motion sensors. Over time, these positional estimates can accumulate significant errors. Thus, odometry typically serves as a preliminary measurement, further refined by additional sensors such as Lidar and GPS. To perform these calculations, we must first determine the movement within the robot's own frame of reference before mapping it to a global context.

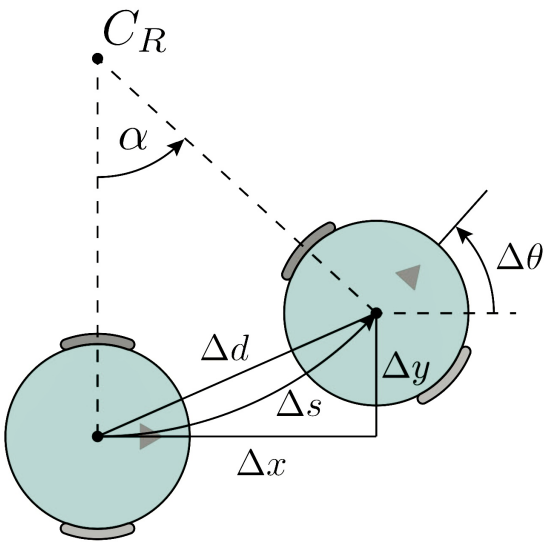

Fig. 1: Robot's configuration.

The robot's configuration (x, y, θ), belonging to the group SE(2), is illustrated in Figure 2. The parameters representing the robot's radius and wheelbase are r_w and b, respectively. We derive

the robot's odometry as follows:

$$\Delta s_L = (R - \frac{b}{2})\alpha \tag{2}$$

$$\Delta s_R = (R + \frac{b}{2})\alpha \tag{3}$$

$$\Delta\theta = \alpha = \frac{\Delta s_R - \Delta s_L}{b} \tag{4}$$

$$\Delta d = \Delta s = \frac{\Delta s_R + \Delta s_L}{2} \tag{5}$$

$$\Delta x = \Delta d \cdot \cos(\theta + \frac{\Delta\theta}{2}) \tag{6}$$

$$\Delta y = \Delta d \cdot \sin(\theta + \frac{\Delta\theta}{2}) \tag{7}$$

3) **Sensor Fusion** The sensor fusion integrates the odometry with the gyroscope-based heading estimates [1]. At low speeds or when stationary, odometry provides more accurate data compared to gyroscopic data; however, when there is a significant discrepancy between the two, gyroscopic data becomes more reliable. A straightforward sensor fusion algorithm was devised as follows, where a threshold is experimentally determined:

$$\Delta\theta_{g-o} = \Delta\theta_{\text{gyro}} - \Delta\theta_{\text{odo}} \tag{8}$$

$$\text{if } |\Delta\theta_{g-o}| > \Delta\theta_{\text{thresh}} \tag{9}$$

$$\theta_i = \theta_{i-1} + \Delta\theta_{\text{gyro}} \tag{10}$$

$$\text{else} \tag{11}$$

$$\theta_i = \theta_{i-1} + \Delta\theta_{\text{odo}} \tag{12}$$

Despite the potential for improved accuracy, this sensor fusion algorithm was ultimately not adopted due to the heavily experimental nature of the threshold and the reliance on precise calibration to ensure accurate odometry data.

4) **Closed Loop Control** To achieve precise motion control and address the physical constraints of the robot, a system of three PID controllers is employed to regulate the individual wheel speed, robot frame velocity, and pose. The settings for the first two controllers are as follows:

Controller	Kp	Ki	Kd	LPF τ
Wheel PWM	0.015	0.01	0.01	0.25
Body Frame Vel., Fwd	0.2	0.05	0.1	0.25
Body Frame Vel., Turn	0.2	0.05	0.1	0.25

TABLE I: Controller Parameters

The PID values are optimized by initially increasing Kp to allow a slight overshoot and reduce the rise time. Subsequently, Kd is raised to diminish settling time and minimize overshoot until it leads to additional oscillations in the output. Lastly, Ki is adjusted to mitigate steady-state error, being careful not to overly extend the settling time. Reference inputs and outputs are plotted to fine-tune these values.

(a) **Wheel Speed Controller:** Illustrated in Figure 2, this controller utilizes a basic PID scheme enhanced with a feedforward element for wheel speed calibration and a feedback loop for error correction. The outputs from both control strategies are combined to dictate the PWM signals sent to the motors. This hybrid approach helps reduce the reliance on large integral terms. Additionally, a low-pass filter is applied to the wheel velocity measurements to filter out noise from encoder ticks, and the control frequency is reduced to 25 Hz to accommodate rapid fluctuations in wheel speed.

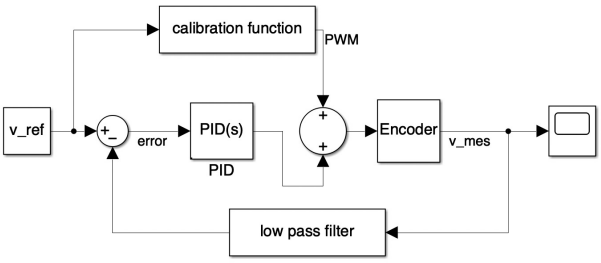

Fig. 2: Speed controller.

(b) **Control of Robot Frame Velocity:** As depicted in Figure 3, the robot frame velocity is managed using a PID controller. This controller processes the discrepancies between the reference frame velocity and the smoothed measured frame velocity to determine the necessary forward and turning velocities. These outputs from the PID controller are subsequently subjected to low-pass filtering to moderate the robot's acceleration and deceleration rates. Finally, the filtered velocities are inputted into the robot's motion model and the subordinate wheel speed controller.

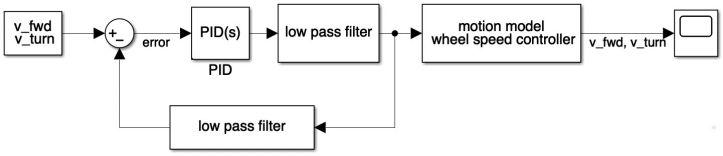

Fig. 3: Robot frame velocity controller.

(c) **Waypoint Pose Controller:** Figure 4. illustrates the RTR `SmartManeuverController` controller utilized for regulating the robot's pose. The parameters are adjusted to ensure system stability. The specific values set for the parameters

are $K_p = 0.9$, $K_i = 6.0$, $K_d = 0.0$.

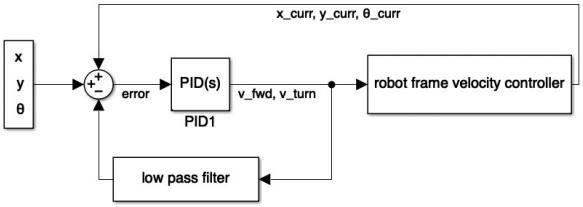

Fig. 4: Waypoint Pose Controller.

B. SLAM

1) **Mapping:** For this project, an occupancy grid map (OGM) was implemented. An OGM consists of an $M \times N$ matrix of cells, each designated as either occupied or unoccupied. The map generation uses a Bayes filter applied to odds ratios. The steps involved in the implementation include:
   a) Identifying the endpoint of each laser ray on the map grid and updating it.
   b) Rasterizing each laser ray across the map to determine which cells are visible and categorizing them as free or occupied.
   c) Converting the known position to the start cell and extending to the endpoint in the map.
   d) Calculating new log odds for each cell affected by the ray.
   e) Segmenting the ray into steps and evaluating each cell it passes through.
   f) Applying Bresenham's algorithm to update the cells along the path of the ray.
2) **Action Model:** The action model is predicated on odometry data, which includes the previous pose $(x_{t-1}, y_{t-1}, \theta_{t-1})$, the current pose (x_t, y_t, θ_t), and the pose changes $(\Delta x, \Delta y, \Delta\theta)$. Decomposing the pose change involves a 1×3 input matrix incorporating one rotation, one translation, and a second rotation as depicted in Figure 5:

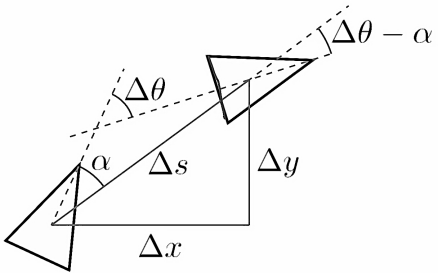

Fig. 5: Positions.

$$u = [\alpha, \Delta s, \Delta\theta - \alpha] \tag{13}$$

$$\Delta s = \sqrt{\Delta x^2 + \Delta y^2} \tag{14}$$

$$\alpha = \text{atan2}(\Delta y, \Delta x) - \theta_{t-1} \tag{15}$$

Additionally, the action error is modeled, assuming Gaussian distributions for errors in rotation1, translation, and rotation2, denoted as $\epsilon_1, \epsilon_2, \epsilon_3$ respectively:

$$\epsilon_1 \sim N(0, k_1|\alpha|) \tag{16}$$

$$\epsilon_2 \sim N(0, k_2|\Delta s|) \tag{17}$$

$$\epsilon_3 \sim N(0, k_1|\Delta\theta - \alpha|) \tag{18}$$

The updated action model incorporating these errors is then given by:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta s + \epsilon_2)\cos(\theta_{t-1} + \alpha + \epsilon_1) \\ (\Delta s + \epsilon_2)\sin(\theta_{t-1} + \alpha + \epsilon_1) \\ \Delta\theta + \epsilon_1 + \epsilon_3 \end{bmatrix} \tag{19}$$

3) **Sensor Model:** The sensor model assesses the probability of a lidar scan aligning with a hypothesized pose given a map. It simplifies the process by focusing on endpoints rather than performing extensive ray casting, which helps overcome smoothness and computational limitations found in beam models. The model uses odds ratios of the cell to calculate the scan score, and if the ray misses an obstacle, it checks the cells immediately before and after along the ray's path to integrate a fraction of the log odds into the scan score.
4) **Particle Filter:** The particle filter offers a non-parametric alternative to the Bayes filter [1]. It approximates the posterior belief with a collection of random state samples drawn from this posterior. This method allows for the representation of a broader range of distributions than those limited to Gaussian forms and can handle the non-linear transformation of random variables. A critical procedure in the particle filter is resampling, which recreates the particle set to the same size based on each particle's weight. However, frequent resampling increases variance and reduces the diversity among the particles.

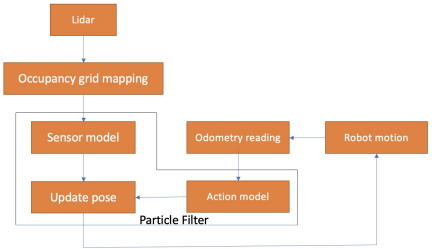

Fig. 6: Particle filter.

C. Planning and Exploration

1) **A* Algorithm:** This represents a heuristic-based method designed to identify the shortest path across a predefined road network. This algorithm leverages heuristic functions to direct its search efforts, assess each node's potential value, and choose the most advantageous node for further expansion until the goal is achieved. Notable for its clarity, efficiency, and straightforward implementation, the A* algorithm is highly regarded in computational pathfinding.
Each node's priority within the algorithm is determined by the following formula:

$$f(n) = g(n) + h(n) \tag{20}$$

where $f(n)$ is the overall priority of node n. The selection of the subsequent node for traversal always favors the node with the optimal comprehensive priority (i.e., the lowest numerical value). Here, $g(n)$ represents the travel cost from the starting point to node n, while $h(n)$ estimates the cost from node n to the endpoint, serving as the heuristic component of the A* algorithm. The heuristic function is setup as follows. In the execution of the A* algorithm, the node with the smallest value of $f(n)$ is chosen from the priority queue as the next node to traverse. The algorithm uses an open set to track nodes that are pending exploration, and a closed set to denote nodes that have been fully explored. We show our implementation of A* in the pseudocode below.

Algorithm 1 A* Path Planning in OGM

```
Input: start, goal, radius = 0.2
Output: path
if start = closed then
    return makePath(start)
open ← start
closed ← ∅
count ← 0
while open ≠ ∅ && count ¡ 50000 do
    n ← open.pop()
    count ← count + 1
    for kid in expand(n) do
        if kid = goal then
            return makePath(kid)
        dist ← obstacleDist(kid)
        if dist > radius then
            if kid ∩ closed = ∅ then
                kid.f ← g(n, kid) + h(kid)
                open ← kid
                closed ← n
    if open = ∅ then
        open ← start
        closed ← ∅
return ∅
```

2) **Map Exploration:** When a mobile robot is deployed in an unfamiliar environment with only an initial starting pose known, it is crucial for the robot to explore the map autonomously. The exploration process involves the robot selecting its next destination using frontier-based methods and calculating paths to systematically cover the entire area.
Frontiers are defined as the boundaries between known free space and areas that have not yet been explored on the occupancy grid. A cell qualifies as a frontier cell if it has log-odds of 0, indicating it is unexplored, and it is adjacent to a cell in free space. The robot employs the Breadth-First Search (BFS) algorithm to identify these frontiers, which consist of contiguous blocks of cells.
Once frontiers are identified, the robot determines the paths to these frontiers from its current location. The BFS algorithm assists in finding a cell within the centroid of a frontier to establish a feasible route. The robot then proceeds to the nearest frontier via the shortest available path. Frontiers are updated continuously as new map data is acquired by the robot. The exploration phase is deemed complete when no further frontiers are detectable in the environment.

Algorithm 2 Map Exploration Algorithm

```
Input: map, home
Output: map
frontiers ← findFrontiers()
current ← home
while frontiers ≠ ∅ do
    path ← findPath(current, frontiers)
    goal ← path[−1]
    while abs(current − goal) > 0.05 do
        motionControl(path)
        map, current ← SLAM(map, current)
    frontiers ← findFrontiers()
return map
```

3) **Map Localization with Unknown Initial Pose:** In scenarios where a robot is introduced into a known environment without a predetermined starting pose, map localization techniques are applied to ascertain the robot's location on the map. The process starts with initializing several particles uniformly distributed across open spaces on the map. The robot executes a series of rotations

in place to capture a comprehensive scan of its surroundings. It then incrementally moves forward while continuing to rotate periodically, which helps in reducing the variance among the particle positions. Localization is considered successful when the particles converge, forming a tight cluster with minimal variance.

III. Design for Block Transportation

A. Mechanism

The lifting mechanism, intended for use within a factory setting, functions analogously to a forklift and comprises several key components: the fork, transmission rack, gear, and motor holder and two main positions: maximum point in Figure 7. and minimum point in Figure 8. Upon detecting a crater, the fork is strategically positioned into apertures within the crater to secure it. Concurrently, the activation of the gears initiates a rotation that propels the transmission rack vertically relative to the ground. This vertical movement, coupled with the stabilization of the fork on the rack, facilitates the efficient lifting of the crater. This mechanism is designed to be always provide enough vision for camera excepting during lifting.

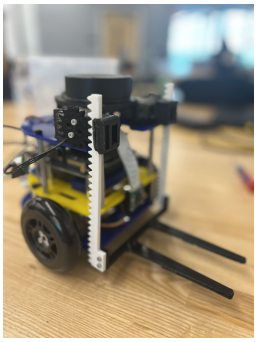

Fig. 7: max point

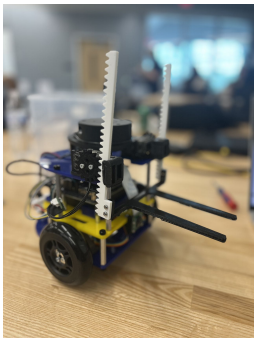

Fig. 8: min point

B. Method

The algorithm used in this system is predicated on the utilization of Apriltags. Upon the detection of an Apriltag, the mobile robot is programmed to adjust its orientation to become parallel with the crater. It then advances towards the crater until it reaches an optimal proximity. Simultaneously, the lift fork is fully inserted into the crater, which is subsequently raised to its maximum height by the interplay of the rack and gear mechanism. The robot continues to navigate through the environment, scanning for a designated drop-off area while maintaining the raised position of the crater. In the absence of a crater at the drop-off location, the mechanism lowers the crater to its minimum height to facilitate the release and complete the initial delivery. Alternatively, if the protocol involves stacking, the crater is lowered to a mid-level height before release, thereby accomplishing the stacking process.

IV. Results

A. Motion Controller and Odometry

1) **Motor Control:** Results of the Motor Calibration slope and intercept derived from six calibrations of our team's MBots is detailed in Table II.

	Slope (R)	**Punch (R)**	**Slope (L)**	**Punch (L)**
Positive				
μ	0.06483	0.07099	0.07082	0.06485
σ^2	4.16e-6	0.00020	1.45e-6	5.87e-5
Negative				
μ	0.06689	-0.06989	0.06608	-0.07010
σ^2	2.25e-5	0.00190	4.21e-6	0.00013

TABLE II: Error Between Ground-Truth and SLAM Pose (m)

To assess the modifications made to the odometry, we instruct the robot to navigate a simple square by setting v and w values corresponding to the square's layout (bypassing the Motion Controller). We then record the odometry data and display it in Figure 9. In this plot, red stars indicate designated turning points within the square, and the blue line represents the odometry's (x, y) outputs.

To further refine the robot's motor control, we use the Motion Controller to direct the robot to again complete one square circuit. Using the enhanced odometry, we then analyzed the pose data under different conditions: without PID or low-pass filters, with only low-pass filters, and with a combination of both PID and low-pass filters in the motor control system.

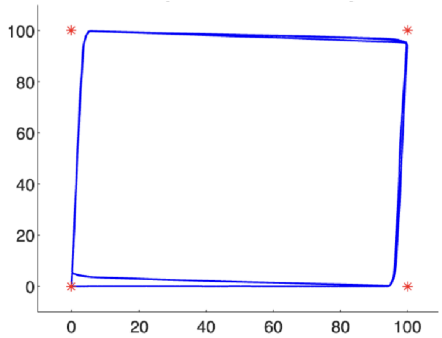

Fig. 9: (x, y) Odometry pose for 4x square path.

B. SLAM

1) **Mapping:** The map shown in Figure 10 was constructed using data from an existing log file to evaluate the precision of the mapping algorithm. In this map, obstacles or walls are distinctly marked as black pixels with a degree of rasterization, whereas areas that are free of obstacles are depicted as white pixels. Additionally, grey areas on the map denote unknown spaces that are beyond the reach of the robot's ray stride.
2) **Action Model:** In adjusting the action model, noise represents errors that are not systematic. To accurately gauge the parameters k1 and k2, which account for these errors, we conduct experiments involving both straight-line movements and rotational tests. This helps us establish appropriate values for k1 and k2. It is critical to note that if the variation is excessively narrow or broad, the localization process will be unsuccessful.
3) **Particle Filter:** We present the computational efficacy of the sensor model across different particle counts. Table 3 shows the time required to update the particle filter at various particle levels. These findings assist in determining the maximum particle capacity the filter can handle while maintaining a frequency of 10Hz.
   Additionally, we calculate the errors in the x, y, and θ values of the SLAM pose produced by the comprehensive SLAM algorithm, utilizing a log containing accurate pose data from a maze-like arena run; this log is sourced from the pre-defined `drive_maze_full_rays.log` file. The results are documented in Table IV. Figure 10 illustrates a comparison between the SLAM pose and the odometry pose as the MBot navigates a basic maze with multiple turns.

Number of Particles	Time to Update (ms)
100	11.42
300	15.76
500	18.47
1000	26.15

TABLE III: Time to update

	$e = \|x - \hat{x}\|$	$e = \|y - \hat{y}\|$	$e = \|\theta - \hat{\theta}\|$
Max(e)	0.2081	0.3115	0.3724
$\mu(e)$	0.1233	0.1692	0.0954
$\sigma(e)^2$	0.0049	0.0071	0.0023
RMSE	0.1412	0.1884	0.1068

TABLE IV: SLAM performance comparison

C. Path Planning

1) **A*:** The duration required to complete A* is indicated in Table V. We assessed the efficacy of A* through the astar_test. The performance was analyzed across various maps, and we successfully completed 5 out of the 6 tests.

Map	Mean Time (s)	Std
Convex Grid	0.000694	0.000612
Maze Grid	0.03599	0.01821
Narrow Constriction Grid	1.26004	1.78109
Wide Constriction Grid	0.98958	1.38111

TABLE V: Map times

2) **Map Exploration:** Our map exploration method effectively navigated around 80% of the maze. During its journey, the robot experienced several noteworthy scenarios:
   The exploration was occasionally considered complete before the entire maze was fully explored due to some areas being incorrectly labeled as unreachable. The robot occasionally halted when the nearest frontier was one it had already visited. The robot adeptly maneuvered between frontiers when faced with two that were equally close, demonstrating its ability to manage multiple nearby objectives.

V. Discussion

The motion control system for the robot consisted of a three-tiered architecture with PID controllers augmented by low-pass filters. The system efficiently managed the robot's linear and angular velocity, achieving rapid rise and settling times with minimal steady-state errors. The rotational velocity had relatively higher steady-state errors, but we mitigated this by reducing K_p and increasing K_i. This control scheme was pivotal in driving the robot to specified poses, compensating for discrepancies in the lower-level controls. Future improvements should focus on refining the velocity frame controller to enhance

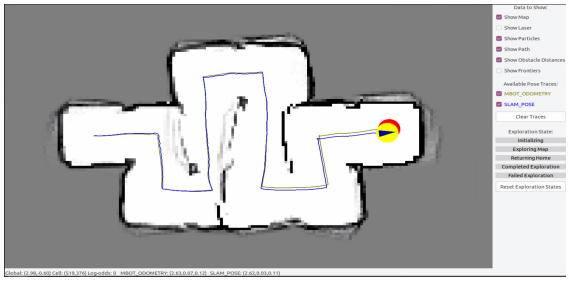

Fig. 10: Comparing SLAM and odometry trajectories

the robot's motion smoothness and precision, perhaps decoupling rotational and linear motion like the action model.

The SLAM system integrated several components: mapping, an action model, a sensor model, and a particle filter. The resultant maps accurately depicted free spaces and obstacles, illustrating the system's efficacy. However, the action model caused the particles to elongate along the motion axis, introducing uncertainties about the robot's exact location. Yet, the particles' centroid aligned well with the robot's intended trajectory. Particle filter performance was validated by combining odometry, SLAM-based poses, and the robot's actual poses, revealing a maximum error of 0.2 meters under rapid movement conditions. Enhancements in SLAM accuracy might be achieved through more sophisticated models such as beam-based or likelihood field models. Further experiments could explore optimal particle numbers for SLAM, as increased counts may introduce lag and additional positional uncertainty.

Path planning foundational to exploration was effectively implemented with the A* algorithm, successfully passing the majority of tests under various conditions swiftly. The map exploration and localization showed promising results, with several trials approaching completion. The robot was generally able to navigate and explore, although there were occasional challenges with mobility and task completion. Enhancements in obstacle grids and refined path completion criteria demonstrated potential improvements in performance. Moreover, map localization frequently produced coherent particle clusters, highlighting effective strategies in cluster classification and exploratory path planning that could enable the robot to accurately determine its position.

VI. Conclusion

In this document, we detail the implementation of various algorithms to facilitate autonomous navigation of a mobile robot. We developed an effective motion controller through calibration of wheel motors, integration of sensor data, PID control techniques, and a block lifting mechanism. The controller exhibited commendable performance, enabling the robot to navigate a maze and repeatedly drive in a square pattern with only a 7cm deviation from the starting point upon completion.

We applied a Particle Filter-based SLAM approach to concurrently map the environment and determine the robot's location. The mapping component was crafted using an inverse sensor model and Bayesian inference with odds, utilizing data from lidar scans. For localization, the action model was used to estimate the robot's pose from odometry data, supplemented by a particle filter for the sensor model. The SLAM system proved both robust and precise, with the SLAM-derived pose closely matching the actual pose, and the particles remaining converged during robot movement.

Path planning was solved using the A* algorithm, complemented by frontier detection and exploration techniques. While the system managed to navigate through the maze, its performance showed variability, with 90% of the maze being explored effectively.

Throughout this project, we gained valuable experience in designing robot controllers, constructing a SLAM system, and deploying a mobile robot for exploratory tasks in mapped environments.

References

[1] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.com/books?id=wGapQAAACAAJ