Autonomous Differential Drive Robot for Warehousing Tasks

Soham Purohit, Andy Qin, Elvin Yang, Fan Yang

***Abstract*—In this project, we developed an autonomous differential drive robot with functionalities supporting pick-and-place warehouse tasks. These include robot motion control with PID controllers, simultaneous localization and mapping (SLAM) with particle filtering, A-star path planning, and frontier exploration. We also include object detection and a forklift design for identifying and picking up objects. In conclusion, our project successfully integrates key components to create an efficient robot tailored for intelligent robotic systems in warehouse automation.**

I. Introduction

In the realm of warehouse automation, the development of autonomous robots capable of executing complex tasks is a pivotal stride toward efficiency and productivity. Warehouse automation involves tackling the following three challenges: planning trajectories navigating from the robot to the target, constructing a comprehensive map for the environments localizing the robot, and actuating the robot to follow the desired trajectories. This project focuses on creating an intelligent robotic system tailored for pick-and-place operations within a warehouse setting. By seamlessly integrating key components, including robot motion control, simultaneous localization and mapping (SLAM), path planning, and exploration functionalities, our autonomous differential drive robot emerges as a sophisticated solution for warehouse automation.

Our robotic system is called the MBot (Classic), a versatile platform featuring a differential drive, magnetic wheel encoders, a 2D Lidar for scanning, and a MEMS 3-axis IMU. The onboard Jetson Nano compute module communicates with other systems via Wi-Fi. Additionally, low-level motor control is handled on the MBot Control Board, which houses a Raspberry Pi RP2040-based microcontroller for efficient operation.

This paper is organized as follows. In Section II, we introduce and detail all the techniques and functionalities our bot has. This includes motion and odometry to make the robot move as desired, SLAM for simultaneously determining the environment and the pose of the bot in this environment, path planning and exploration to be used in SLAM, and miscellaneous functionalities such as camera object detection and forklift for picking up objects. Section III deals with experiments for determining the performance characteristics of these methods. We discuss the performance and limitations of our methods in Section IV. Section V concludes the project, providing future directions for improvement.

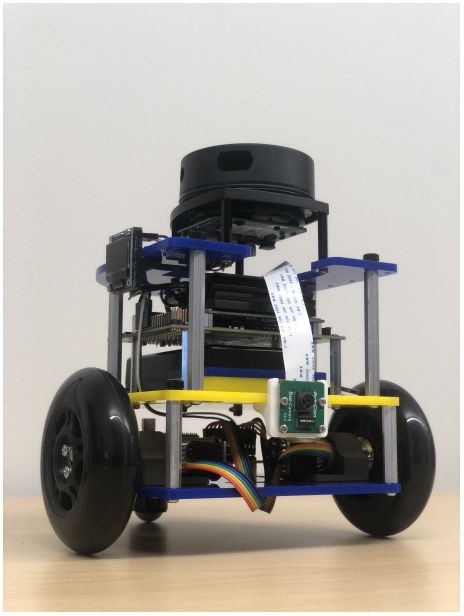


Fig. 1: MBot Classic

II. Methodology

A. Motion and Odometry

1) Motor Control: To control the robot's motion, we first consider low-level control for the wheel motors to translate instantaneous angular velocity commands into PWM signals.

Since each robot has small variations in its motors, wheels, and assembly, wheel speed calibration must be performed on each robot to determine its reference PWM values. This calibration process involves placing the robot on a flat surface and reading the wheel encoders while sending a sequence of predefined PWM commands. The parameters generated by the calibration are the parameters to a piece-wise linear function fit to the angular speed vs PWM curve of the measurements.

Due to unmodeled errors and uncertainties, an open-loop model is unlikely to work well on the real system. Instead, we implement individual PID controllers for each wheel that use angular velocity feedback from the wheels' magnetic encoders along with the specified velocity command to generate appropriate PWM signals.

To tune the PIDs, we increase the proportional gain for faster tracking of the desired speed until the system begins to becomes unstable, at which point the proportional gain is fixed to the last stable value. The differential gain dampens out oscillations that may arise in the system. A low-pass filter is applied to remove high frequency noise that would be magnified by the differential gain. Finally, the integral gain reduces steady-state errors in the system. Determining the optimal set of gain values is done through trial and error. Rise time, settling time, and overshoot are determined for each set of parameters, and modifications to the gains are made until we achieve a reasonable range of parameters. We finalized our parameters when the rise time decreased to less than 3 seconds for a typical operating wheel speed. The parameters of our PID are as follows: $K_p = 1.6, K_i = 0.8, K_d = 0.01$. The cutoff frequency we used for the low-pass filter was 4 times the main loop period.

We observed that it was difficult to tune each PID such that it could track both the typical operating velocities for the robot and the zero velocity. With the aforementioned tuned PID gains, the wheels jitter slightly after returning to a velocity of zero due to overshoot oscillation within the PID controller. While reducing the proportional gain could reduce overshoot around zero, it also increases the rise time of the controller for nonzero velocities. To mitigate this, we instead set a minimum absolute threshold that the wheel angular velocity command must exceed for the controller to issue a nonzero PWM signal. This threshold was set to 0.1 rad/s.

2) Odometry: Odometry is the process of estimating the robot's position and orientation relative to an arbitrary starting position using various sensors. In this system, odometry is calculated with proprioceptive sensors onboard the robot. As a starting point, wheel encoders are used to measure the angular velocities of each wheel, which in turn are used to calculate the distance each wheel travels and the robot's displacement for each period of the main loop.

However, wheel odometry alone is often inaccurate since various systematic (e.g. variation in component sizes and assembly) and non-systematic errors (e.g. wheel slip) accumulate to large values over time, particularly with respect to the robot's orientation. To correct the angular component of this error, we integrate gyroscope readings in addition to the wheel encoder readings. We considered using a method called Gyrodometry [1], which uses gyroscope readings if the difference between angular velocities calculated from wheel encoder readings and gyroscope readings exceeds a pre-determined threshold. However, we found that, in practice, using the filtered orientation estimate provided by the IMU SDK results in a significant and sufficient improvement in orientation accuracy compared to orientations derived from wheel encoders.

3) Motion Control: To reach unobstructed target locations in the robot's odometry frame, we represent the motion of the robot as an RTR (Rotate-Translate-Rotate) model. In this model, the robot first turns in place to align itself along the line joining its current position and the target position. This first turn minimizes angular displacement by always taking the turn with convex angle ($|\theta| \leq 180°$) to reduce the uncertainty associated with the overall motion. As a consequence, the robot may face away from the target location and must drive backward. Next, the robot drives in a straight line to reach the target position. Finally, it rotates in place again to achieve the desired orientation at the target location. Each component of the motion model is implemented with a PID controller that generates 2D twist commands to drive the robot in a manner that approaches the goal location. Odometry is provided as feedback to the PID controller to track translational and angular error relative to the goal position. As with the motor controllers, the parameters for these PID controllers are determined empirically through trial and error until satisfactory qualitative performance is observed.

Since we used separate PID controllers for the rotation and translation components of motion, they required separate tuning. The values of the parameters that we used for the controllers were

1) Translation: $K_p = 1, K_i = 0, K_d = 0.01$
2) Rotation: $K_p = 3, K_i = 0, K_d = 0.01$

In addition, the controllers for each component of the motion model consider their intermediate target reached if the error calculated from odometry falls below an absolute threshold. The thresholds are as follows:

1) Translation: $|dx| < 0.02 \wedge |dy| < 0.02$
2) Rotation: $|d\theta| < 0.05$

B. Simultaneous Localization and Mapping (SLAM)

1) Mapping: Mapping refers to the problem of generating a map of the robot's environment given a set of exact robot poses and their corresponding sensor readings. We represent the map of the surrounding environment as a bounded 2D Occupancy Grid. This representation discretizes the robot's surroundings into grid cells, each assigned with a probability of being occupied. We either increase or decrease the likelihood of each cell being occupied based on successive sensor data. The sensor data is in the form of LIDAR rays,

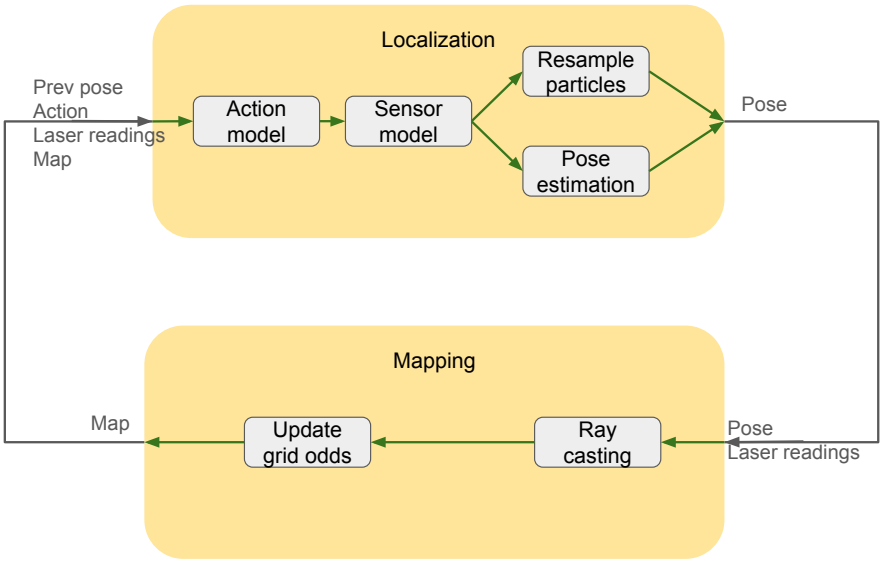


Fig. 2: The diagram outlines the general framework of a particle filter-based SLAM implementation. SLAM executes the mapping and localization algorithms iteratively. The localization algorithm takes in the previous pose estimate, robot action, laser readings, and occupancy grid map and uses a particle filter to estimate the current position. The mapping algorithm uses Bresenham's line algorithm to obtain the grid cells that the laser ray passes through. The odds of each cell in the occupancy grid map represent the probability of the cell being free or occupied and are updated accordingly.

each consisting of an origin position, a corresponding angle, and a range. For each ray, we can determine the cells through which the ray has passed and the cell from which it got reflected, which indicates the possible presence of an obstacle. When a ray passes through a cell, we decrease the likelihood of the cell being occupied, and if the ray is reflected from a point, we increase the likelihood of the corresponding cell being occupied. We use Bresenham's line algorithm for ray casting, eliminating the need for floating point numbers and significantly improving computation time.

2) Localization: The task of localization is to estimate the robot's current pose, given a map of the surrounding environment, robot actions, and sensor data. We implement Monte Carlo Localization as a particle filter. This uses an action model to create a proposal distribution of particles from a prior distribution of particles, a sensor model that recomputes the probability associated with each particle in the proposal distribution, and a resampling procedure to generate a corrected posterior distribution of particles. The posterior distribution created in each timestep becomes the prior distribution for the next timestep. Hence, the problem is broken down into the following three parts.

a) Action Model: We use the odometry model described in Section II-A3 as our action model, in which motion is divided into disjoint rotation-translation-rotation steps for the robot to move from an initial to a final pose. The equations of the odometry model are

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

Here, Δs represents the total displacement of the robot, and $\Delta\theta$ represents the net change in angle from the initial pose to the final pose. α gives the angle the robot rotates through in the first rotation. ϵ_i are errors that we sample from Gaussian distributions as follows:

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

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

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

We chose the hyperparameters $k_1 = 0.005$ and $k_2 = 0.025$ through trial and error, making sure that the uncertainty represented by the particle distribution is in the direction of motion for straight-line motion and expands when the robot turns in place.

b) Sensor Model: The sensor model calculates the likelihood of the pose estimate represented by a particle given a laser scan and a occupancy grid map. An

ideal sensor model should use Bresenham's Algorithm to perform ray casting and check every cell the ray passes through and terminates at to assign a score to that ray. However, this is computationally expensive, so we instead use the Simplified Likelihood Field Model, which checks the correspondence of a laser ray with the map only at the termination cell of the ray. The log-likelihood of each particle in the proposal distribution is reset to 0. For each particle, when the occupancy map indicates a high probability of occupancy for the termination cell of a ray, we increase the likelihood of the particle with a constant value, which works reasonably well in practice.

c) Resampling: After a reweighted proposal distribution is generated by the sensor model, a posterior distribution is created by resampling the distribution. We begin by computing the "cumulative distribution" of the particles using the partial sums of their likelihoods. We populate the posterior distribution with samples of the proposal distribution by stepping through the cumulative distribution using a fixed step size inversely proportional to the configured number of particles in the distribution. The log-like

d) Pose Estimation: We observe that the distribution of particles is nearly always unimodal and resembles a Gaussian distribution. Thus, it is reasonable to compute a weighted sum of all particles to generate a single pose estimate from the particle filter.

e) Initial Localization: A robot may periodically start operating at an arbitrary location within a mapped environment. In this situation, there is no prior information for the localization algorithm to use. A simple initialization procedure that works reasonably well in practice is to randomly sample particles in the map. As the robot starts to move, the distribution of particles begins to converge into plausible locations that the robot may be within the map. However, the particle filter may prematurely converge on a local optimum that is not consistent with the actual location of the robot. To combat this, the particle filter can be periodically reinvigorated with randomly sampled particles.

3) SLAM: With a system developed for both mapping and localization, the objective of SLAM is to simultaneously estimate the robot's pose and build a map of the robot's surroundings based on sensor information. Laser scans obtained while the robot is stationary are used to develop an initial map of the surroundings. As the robot moves, we perform the task of localization using the map developed in a previous timestamp. We update the map through successive laser scans using the pose obtained through this localization. This procedure allows us to simultaneously estimate the pose of the robot while concurrently building a map from sensor information. An overview of the SLAM algorithm is shown in Fig. 2.

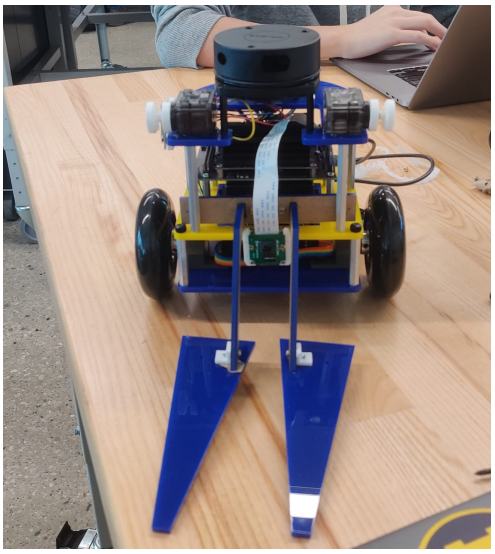


Fig. 3: Pickup Mechanism for the MBot. The forklift is raised and lowered using a pulley mechanism controlled by two servo motors.

C. Planning and Exploration

1) Path Planning: For the task of path planning, we use the popular A* algorithm. A* plans paths within occupancy grid maps by successively visiting grid cells based on a sum of two metrics. The first metric is the "real" cost of traveling from a starting position to a particular cell. The second metric is a heuristic that estimates the remaining cost of traveling from the cell to the goal position. We use the following heuristic that is admissible in 8-connected grids:

$$h(dx, dy) = |dx - dy| + \sqrt{2}\min(dx, dy)$$

Once the goal cell is reached via a neighboring cell, the best path from the initial and target cells is computed by tracing edges formed by successive cells back from the target cell to the initial cell.

2) Exploration: Our exploration algorithm is based on frontier exploration. Specifically, the frontier is defined as the boundary of the explored and unexplored regions of the map. The robot plans a path to each frontier to explore the unknown regions. In our algorithm, a cell belongs to a frontier if it is an unexplored cell with the log-odds equal to 0 and at least one of its neighboring cells is a free cell with negative log-odds. Once we find such a cell, we expand the frontier by checking whether its connected components satisfy the same condition. In practice, each frontier resembles a line separating an explored and unknown region. We compute the centroid of each frontier and sort the frontiers according to the distance of their centroid relative to the robot's position. The robot plans a path to the nearest frontier centroid

Bot no.	m_r/σ_{mr}	m_l/σ_{ml}	b_r/σ_{br}	b_l/σ_{bl}
1	0.075/4e-4	0.058/4e-4	0.047/5e-4	0.065/4e-3
2	0.063/5e-4	0.057/2e-4	0.060/6.0e-3	0.061/1.0e-3
3	0.065/4e-4	0.058/5e-4	0.063/3.4e-3	0.068/1.4e-3

TABLE I: Wheel Calibration Results for Positive Rotation

until no frontiers remain. We assume that the robot is enclosed in a bounded environment.

D. Miscellaneous

1) Camera: To accurately determine the location of a block for the robot to pick up, we mount a 5MP camera to the front of the robot to detect AprilTags [2] placed on the surface of the block. We calibrate the intrinsic matrix of the camera using a checkerboard calibration routine:

$$K = \begin{bmatrix} 1279.51 & 0.0 & 646.74 \\ 0.0 & 1284.43 & 343.29 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The location of the block in the camera frame can be computed with the intrinsic matrix and the known size of the AprilTag. This computation is done in the AprilTag library.

2) Pick-up Mechanism Design: A forklift mechanism was designed to perform the task of picking and placing the blocks. We used a pulley system attached to a frame through a string. The frame had the forklift attached. The two pulleys were made to rotate through servo motors, which led to the unraveling and raveling of a thread that lifted and dropped the mechanism. The frame was constrained to move vertically by limiting its motion by using the structures of the robot itself. Projections in the frame prevented sideways motion and the standoffs connecting the top plate to the middle plate prevented back-and-forth motion. This mechanism is displayed in Figure 3.

III. Results

A. Motion and Odometry

1) Wheel Speed Calibration: We performed the wheel calibration routine on three different robots and determined the variation. The slope and the intercepts of the two wheels for their positive rotation for all three bots are shown in Table I. These results give us a linear relationship between the PWM and wheel speed values. Note that different linear functions are used for positive rotations and negative rotations. We only show the results in the positive direction for conciseness.

We observe a nonzero standard deviation in the values because of non-systematic errors such as non-uniform slipping conditions, sensor noise in the IMU data, and voltage fluctuations in the system.

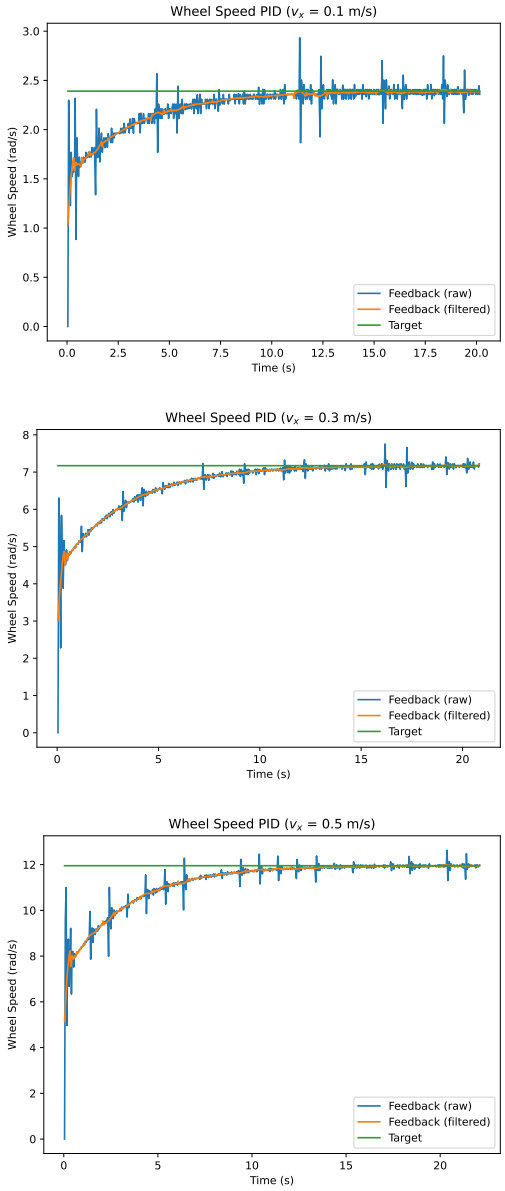


Fig. 4: Wheel speed feedback for (a) 0.1 m/s (b) 0.3 m/s (c) 0.5 m/s robot velocity. Smoothing is performed for analysis.

2) Odometry: To test our odometry, we first moved the robot in a straight line by a known distance and noted the values of the odometry outputs. We then rotated the robot by known angles in both the clockwise and counterclockwise directions and obtained the odometry data. The results are shown in Table. II.

	Ground Truth	Measured	% Error
Translation	1.225m	1.193m	2.6
Rotation (cw)	45°	40.37°	10.3
Rotation (ccw)	45°	49.61°	10.2

TABLE II: Odometry Test Results

Since for this part, we manually measure the distance and angles, relatively large errors during measurement are expected. Despite the errors, We believe we have obtained reasonable results for the odometry.

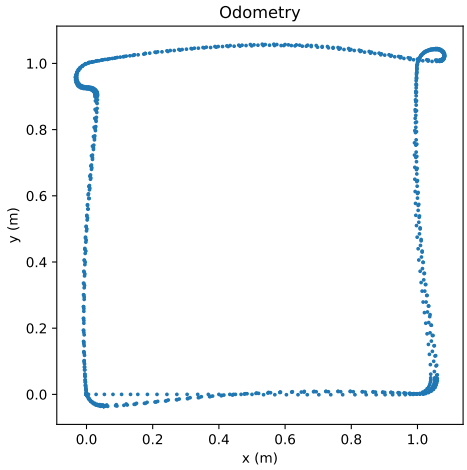
3) Wheel Speed PID Controller: We tested the performance of our PID by determining characteristics such as rise time and settling time at different reference velocities. The plots of our tracking can be found in Figure 4.

Speed (m/s)	Rise Time	Settling Time	Steady State Err.
0.1	4.27s	6.69s	0.128%
0.3	4.53s	6.87s	0.076%
0.5	4.68s	6.65s	0.045%

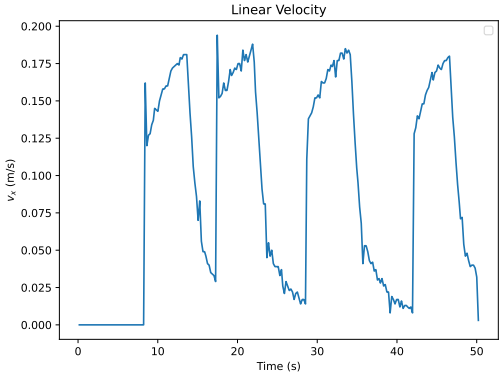
TABLE III: Wheel speed PID statistics at different robot body speeds. Since the PID appears nearly critically damped (Fig. 4), the overshoot for all cases is 0.

The characteristics of our wheel speed PID controller are reported in Table III for three common operating velocities for the robot. Fig. 4 contains the corresponding feedback curves. Note that the speed of the robot and the wheel speed are different quantities. Rise time is computed as the time taken by the feedback to achieve 90% of the target. Settling time is computed as the time taken by the feedback to settle at within 5% of the target. Finally, the steady-state error is calculated as the relative difference between the feedback in the last second of the data and the target. Since the raw feedback from the wheel encoder has periodic spikes (see Fig. 4), all statistics are computed using the smoothed feedback response from a Savitzky-Golay filter with window size 15 and polynomial order 2.

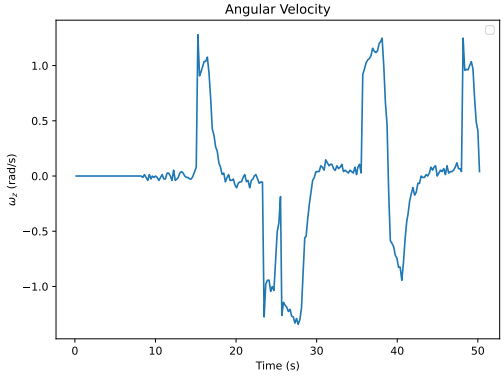
4) Motion Controller: To systematically evaluate the errors of our motion controller, we command our bot to track a known trajectory using dead reckoning only. Specifically in our experiment, we command our robot to track the trajectory of a square four times. We plot out the trajectories and the velocity profiles, which are shown in Fig. 5. The dead reckoning results demonstrate that our robot has a minor drifting away from a perfect square. However, in practice, there is observable drifting due to errors such as wheel slipping, which we should deal with using SLAM.



(a) Odometry-estimated pose of the bot for four square cycles.



(b) Translational velocity of the robot for one square cycle.



(c) Angular velocity of the robot for one square cycle.

Fig. 5: Trajectories and velocities of the robot following a square trajectory.

B. Simultaneous Localization and Mapping (SLAM)

1) Mapping: We tested our mapping algorithm on pre-recorded log files. In one such log, the ground truth

poses of a robot driving around a maze as well as the laser scan information had been stored. Using these ground truth values, we generated a map using our algorithm. The generated map is displayed in Figure 6. The generated map was found to closely resemble the actual map which had been stored in a corresponding map file.

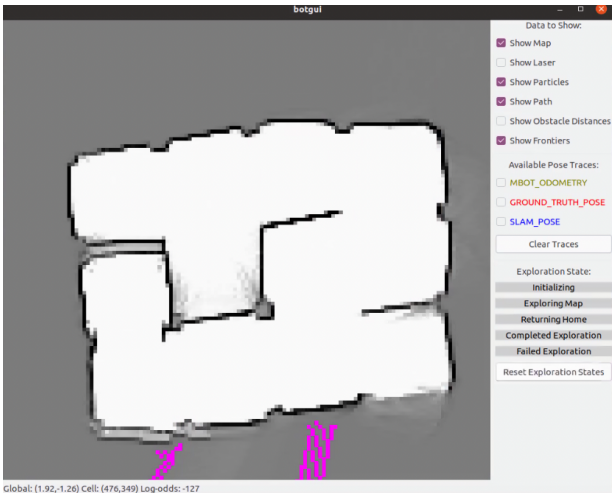


Fig. 6: Occupancy grid map generated by SLAM from the `drive_maze` log file.

2) Monte Carlo Localization:

a) Particle Filter: The number of particles is an important parameter that affects the performance of the particle filter. With a larger number of particles, the calculated probability distribution closely resembles the ground truth distribution at the cost of a larger amount of computational time. We are limited by the rate at which the SLAM algorithm is run, which is 10 Hz in our case. We wish to determine the maximum number of particles we can use to maximize our performance. The Table IV displays the average runtime of a single particle filter iteration for varying numbers of particles.

No. of particles	Average time for update
100	0.0038 s
500	0.0164 s
1000	0.0318 s
2500	0.0789 s
3000	0.0965 s

TABLE IV: Average runtime of the particle filter for varying numbers of particles.

We plotted these values and observed a linear relationship. We then extrapolated these values after drawing the best-fit line to obtain the maximum possible value of particles for 10 Hz frequency. This value was 3320 particles.

Next, we also determined the extent to which our particles were getting distributed by simulating the robot driving around a square and recording the particle distribution at different points in time. We did this to ensure that the particles gave a good resulting distribution that was neither blowing up nor collapsing at any time. The result of this experiment is shown in Figure 7. Visually, the scale of the standard deviation of the particles matches the magnitude of the actions. Specifically, when the robot is moving forward, the particles are "stretched" along the moving directions, and when the robot is turning, the particles are all around the current locations.

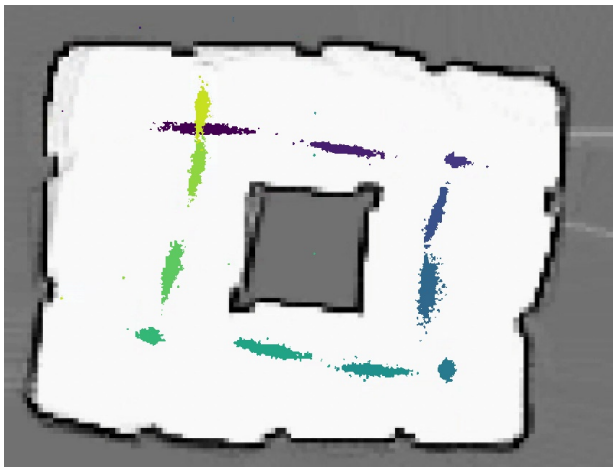


Fig. 7: Distribution of the particle filter at various times during the playback of the `drive_square` log file. In this visualization, the particles become lighter as the playback progresses. We observe that the distribution of particles elongates as the robot drives forward and condenses when the robot turns in place.

Test	Min	Mean	Max	Median	Std. Dev.
Convex grid	56	1561.5	3067	1532	1505.5
Maze grid	1301	1951.75	2511	1301	442.205
N. con. grid	702	535178	862442	862442	381096
W. con. grid	620	131547	198415	198415	92586.2

TABLE V: A* test time statistics (μs)

3) SLAM: We tested our SLAM using a pre-recorded log file with ground truth poses. We calculate the Root Mean Squared Error (RMSE) of the relative pose error as follows to obtain a translational error of 0.482 m and a rotational error of 5.62°.

C. Planning and Exploration

1) Path Planning: To test our path planning algorithm, we simulated the algorithm on several example logs, namely, an empty grid, a filled grid, a narrow constriction grid, a wide constriction grid, a convex grid, and a maze grid. These tests determined the average time required for finding the path to the target. The results were as follows, where all results are in μs.

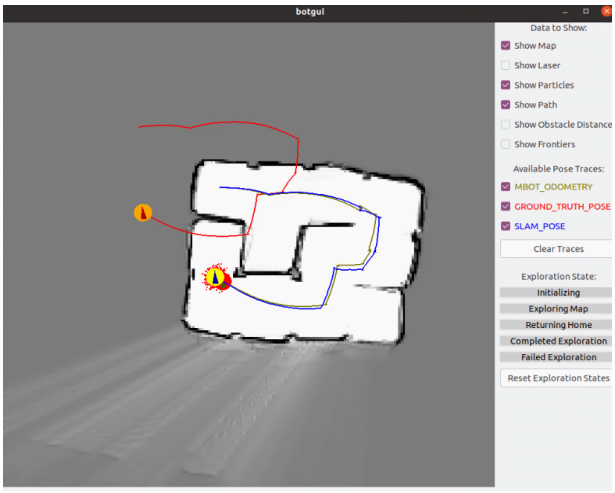


Fig. 8: SLAM, Ground Truth, and Odometry Poses for the `drive_maze_full_rays` log file. Since the `GROUND_TRUTH_POSE` trajectory is offset from the map, we calculate the relative pose error rather than the absolute trajectory error.

The test works well for the convex grid and maze grid, taking a few milliseconds. For the constriction grids, the median times are of the order of fractions of a second. This increase in runtime could be due to the presence of local minima in the environment that result in a large number of states to be explored. Possible strategies that could improve runtime are a coarser discretization resolution, non-admissible heuristics to find the goal faster, and more efficient memory management.

IV. Discussion

In this project, we successfully developed a differential drive robot that can perform SLAM in a new environment through exploration, determine optimal paths from initial to final locations, and move to those target locations through motion control. Further, it can identify and pick up objects using a forklift mechanism and place them in desired locations.

There are various limitations of both the platform itself and our implementation of an autonomy stack. With regard to the platform, we frequently encountered stability issues with the platform that could only be resolved with a reset of the control board microcontroller or a full reboot of the Jetson computer. The robot would often stutter when teleoperated via mouse or keyboard from the web interface, making tasks such as SLAM somewhat difficult. We were also unable to test path planning using the web interface or the GUI application on the Jetson as the functionality to specify a goal location did not work. We tried changes to the GUI application suggested by other classmates, but those did not work for us. With regard to our implementation, although manually tuning parameters has diminishing returns, it is possible that we can improve the accuracy of our controllers and localization algorithm by further tuning the parameters. We note that we obtained parameters qualitatively rather than quantitatively in the interest of time. The performance of our SLAM implementation may be improved by using more sound representations of probability (e.g., using floating point log-likelihoods instead of integer log-odds) and considering alternative approaches such as scan matching, which is used in the frontends of high-performance 2D Lidar SLAM solutions such as Google Cartographer and Hector SLAM. However, neither of these changes would currently be supported by the mbot tooling.

V. Conclusion

This project is a stepping stone to warehouse automation. Furthermore, the system's modular design facilitates scalability and future enhancements, opening avenues for additional functionalities and improvements. The successful integration of SLAM, exploration, path planning, motion control, and object manipulation positions our project at the forefront of robotic applications, providing a foundation for further advancements in autonomous systems and intelligent automation.

References

[1] J. Borenstein and L. Feng, "Gyrodometry: a new method for combining data from gyros and odometry in mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, 1996, pp. 423–428 vol.1.

[2] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2006. [Online]. Available: http://www.probabilistic-robotics.org/

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