Section 2 Team 4 ROB 550 BotLab Report

Dhyey Manish Rajani, Ziqi Han, Xingchen Zhou

***Abstract*—In Botlab, we present a comprehensive analysis of the enhancement of the MBot, an autonomous mobile robot, within the BotLab framework. The project focuses on improving the robot's capabilities in movement control, planning, exploration, and autonomous localization. Key advancements include the integration of sensor feedback with closed-loop controllers for precise movement control and the implementation of a real-time LIDAR-based Simultaneous Localization and Mapping (SLAM) system. The report details methodologies, including motion and odometry improvements, SLAM implementation, and path planning and exploration strategies [1]. Through iterative testing and real-world application, the project demonstrates significant progress in autonomous robotics, highlighting practical solutions to challenges and showcasing the MBot's improved control, navigation, and adaptability in diverse environments.**

***Keywords*—** MBot, close-loop control, path planning, exploration, LIDAR, Simultaneous Localization and Mapping

I. Introduction

MBOT Lab project delves into the development and enhancement of autonomous mobile robotics, focusing on a robot named Mbot. The primary objectives include refining Mbot's capabilities in obstacle detection, movement control, planning, exploration, and autonomous localization to improve its adaptability and efficiency in diverse environments. The approach integrates advanced sensor feedback and closed-loop controllers for precise movement control. Additionally, a real-time LIDAR-based Simultaneous Localization and Mapping (SLAM) system is implemented [2]. This report provides insights into the challenges and practical solutions within autonomous robotics, showcasing the progress in control, navigation, and autonomy of the Mbot within the BotLab framework.

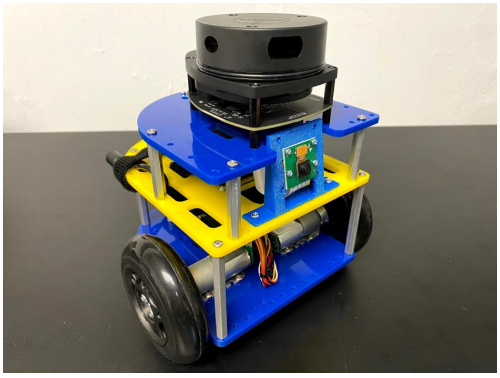


Fig. 1: The Mbot

This report meticulously details the construction of a robot capable of mapping, perception and planning. Section II entails Odometry, Motion Control, SLAM, Planning & Exploration; and section III covers the results of all the sub-sections entailed in section II; section IV discusses the challenges and improvement; and finally, section V concludes this comprehensive exploration of autonomous robotics systems.

II. Methodology

Botlab enhances Mbot autonomy through carefully configured hardware, including a high-resolution LiDAR sensor. A multilayer software architecture supports real-time processing. The first layer controls basic movements and data acquisition, the second layer utilizes LiDAR-based SLAM for mapping, and the third layer handles advanced functions like path planning. Iterative testing in controlled environments ensures reliability before real-world deployment.

A. Motion and Odometry

The Mbot, depicted in Figure 1, is a robotic model resembling a unicycle, and as such, its condition can be denoted by a positional coordinate (x, y) and an orientation angle θ . Odometry serves as a crucial mechanism for maintaining awareness of the Mbot's internal state and pose. To achieve this, the Mbot is outfitted with motor encoders affixed to both the left and right drive motors, along with an Inertial Measurement Unit (IMU) and Light Detection and Ranging (LIDAR) sensors. The utilization of encoders and IMU facilitates the implementation of odometry functionality to precisely monitor and comprehend the Mbot's movements and spatial orientation.

1) Wheel Speed Calibration: For the implementation of the open-loop control, we map Pulse Width Modulation (PWM) signals into wheel velocities [3]. This conversion is represented mathematically as,

$$velocity_{\text{left,right}} = func_{\text{left,right}}(pwm_{\text{left,right}}) \tag{1}$$

here velocity(), pwm() and func() represent wheel velocity, PWM signal command, and mapping function, respectively.

We collected the experimental data by precisely measuring the actual Wheel Base Radius and Wheel Radius using vernier calipers and then executed the wheel speed calibration program. This program utilized the method of least squares to fit a linear relationship between the motor PWM and linear velocity for both left and right motors. The calibration routine was run 20 times on the ROB 550's instructional laboratory's marble floor, during which we recorded the mean and variance for Positive Slope, Negative Slope, Positive Intercept, and Negative Intercept for both wheels. Equations derived from curve fitting are used to establish a mapping between the desired/required wheel speeds and the corresponding PWM control signals providing a reliable basis for the controller design.

2) Odometry: Odometry is a method to estimate the position and movement of a vehicle by analyzing data from its motion sensors, typically wheel encoders. It involves tracking changes in position over time to determine the vehicle's trajectory. Due to non-systematic errors like bumps or wheel slippage odometry readings depict large deviations from the actual

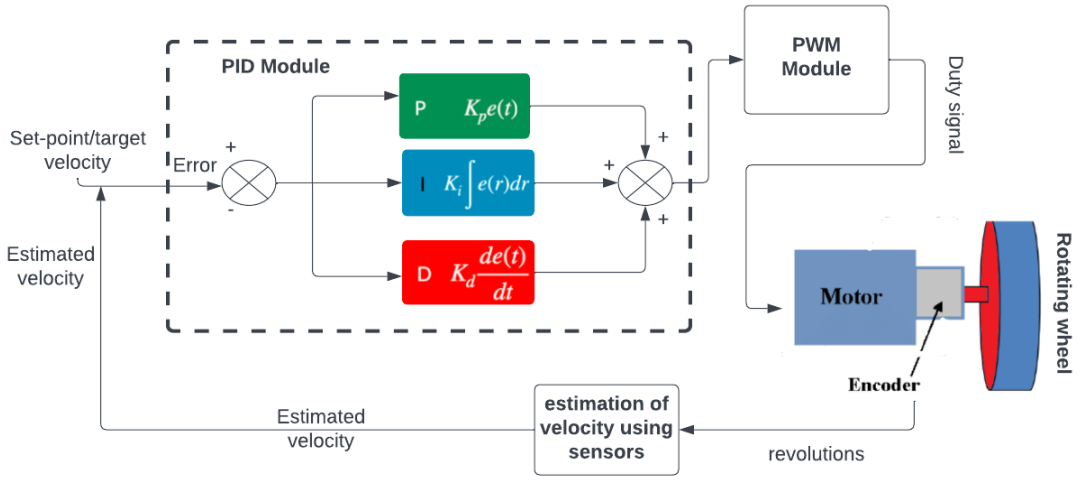


Fig. 2: Wheel Speed Controller

trajectory leading false positional and heading estimates. Drifts due to temperature-dependent bias affect the IMU or MEMS (Micro-Electro-Mechanical Systems) [4] gyroscope readings, and to mitigate this bias, regular calibration becomes imperative. Resultantly, we combine the odometry and gyroscope measurements using Gyrodometry [5], (Algorithm 1).

Here, hyper-parameter $\Delta\theta_{\text{thres}}$ represents the degree of confidence in gyro readings with respect to the odometry.The innovative idea behind Gyrodometry involves integrating odometry with a gyro-based heading estimate. By discerning when the wheels are stationary, the system can ignore gyro data, focusing instead on odometry. Conversely, if the odometry readings significantly deviate from gyro, it signals potential disturbances like bumps, prompting a shift in trust towards the gyro. Gyrodometry hinges on the hypothesis that the disparity between the odometry and gyro curves is transient.

3) Wheel Speed PID Controller: A PID (Proportional-Integral-Derivative) controller is a control loop feedback mechanism requiring continuously modulated control. It is a type of control system that uses feedback to continuously adjust [6] the output of a system to match a desired setpoint. A PID control loop (Figure 2) is employed to compute the proportional (P), integral (I), and derivative (D) terms based on the error between the desired velocities (setpoint) and actual velocities. An open-loop formulation-based feed-forward term is also incorporated.

Algorithm 1 Gyrodometry

Require: $\Delta\theta_{\text{gyro}}, \Delta\theta_{\text{odom}}$
Ensure: $\Delta\theta$
for each step in calculating odometry **do**
  Compute $\theta_{G-O} = \Delta\theta_{\text{gyro}} - \Delta\theta_{\text{odom}}$
  if $|\theta_{G-O}| > \Delta\theta_{\text{thres}}$ **then**
    $\Delta\theta = \Delta\theta_{\text{prev}} + \Delta\theta_{\text{gyro}}$
  else
    $\Delta\theta = \Delta\theta_{\text{prev}} + \Delta\theta_{\text{odom}}$
  end if
end for

During the PID controller tuning process, we followed specific principles to optimize its parameters. If the system response was sluggish, we increased the proportional gain (k_p). Persistent deviations were addressed by incrementing the integral gain (k_i), while excessive oscillations in the system prompted an increase in the derivative gain (k_d). This iterative tuning approach aimed to attain a steady-state value within a defined permissible error band around the setpoint. Additionally, as part of our efforts to enhance the PID framework, we introduced a modification involving the integration of low-pass filters. These filters effectively reduced noise in encoder readings and closed-loop controller outputs, contributing to a more robust and stable control system, particularly in real-world conditions where undesired fluctuations could impact performance.

4) Motion Controller: The motion controller presented is a critical component for orchestrating the movements of Mbot. Motion controllers, depicted in *diff_motion_controller.cpp* for various tasks involve the use of the non-holonomic robot model (Figure 3) and its control equations as shown below.

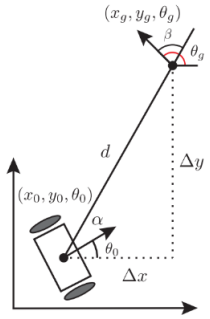


Fig. 3: Wheel Speed Controller [7]

The kinematic equations corresponding to Figure 3 involve [7]:

$$\Delta x = x_g - x \tag{2}$$
$$\Delta y = y_g - y \tag{3}$$
$$d = \sqrt{\Delta x^2 + \Delta y^2} \tag{4}$$
$$a = \text{atan2}(\Delta y, \Delta x) - \theta \tag{5}$$
$$\beta = \theta_g - \theta \tag{6}$$

Resultantly, the control law [8] for the system can be formulated as following:

$$v(forward) = K_d \cdot d \tag{7}$$

$$\omega(rotational) = K_\alpha \cdot \alpha + K_\beta \cdot \beta \tag{8}$$

The motion controller provides a higher level of abstraction to robot control by receiving a sequence of waypoints as input and producing velocity set-points. These set-points enable the robot to effectively traverse the specified path. In order to streamline the navigation process and prevent unnecessary halts at every turn, intermediate waypoints are given more tolerance compared to the ones at the final destination.

Our motion control system has been enhanced with an adjustment to the PID settings of the Smart Controller, and an acceleration limit has been added. These improvements ensure that the controller can deliver precise expected speeds to the chassis for smoother and more accurate motion control, which was further confirmed driving around a 1 meter skewed square circuit.

- PID Control Adjustment: We have optimized the PID control algorithm within the Smart Controller to improve the accuracy and responsiveness of the robot's speed and direction control.
- Acceleration Limit: To prevent the robot from accelerating too quickly, which could lead to instability or damage, we have implemented an acceleration limit. As shown in the code, if the forward velocity (fwd_vel) exceeds the last speed (last_speed) by more than 0.008 meters/second, the system automatically adjusts the speed to limit the increase in acceleration.

Dead reckoning served as a rudimentary yet effective means of localization, relying on the combination of linear and rotational velocities measured. This approach involves the continuous updating of the robot's position based on the cumulative effect of its observed movements over time. The process can be discerned more expressively in the results.

B. Simultaneous Localization and Mapping (SLAM)

Our project focuses on developing and improving complex simultaneous localization and mapping (SLAM) systems for autonomous robotics. The SLAM system includes occupancy-grid-based mapping and Monte Carlo localization.

1) Mapping: The Mapping component uses the LIDAR sensor to update an occupancy grid map of the environment. Each laser ray is processed individually as the LIDAR scans the surroundings. The Mapping component updates the map cells based on these rays, altering the log-odds values for each cell. The odds for cells hit by the laser are increased, indicating a higher likelihood of occupancy. Conversely, cells that the laser beam passes through without a hit have their odds decreased, indicating a lower probability of occupancy. This dynamic updating process creates a detailed and constantly evolving map of the environment. To ensure effective ray casting on the occupancy grid, we employed Bresenham's line algorithm.

2) Monte Carlo Localization: Monte Carlo Localization (MCL) is a localization algorithm based on particle filters.[9] To implement MCL, three key components are required: an action model to predict the robot's pose, a sensor model to calculate the likelihood of a pose given a sensor measurement, and various functions for particle filtering [10], including drawing samples, normalizing particle weights, and finding the weighted mean pose of the particles.

a) Action Model: The MCL of our Botlab starts with the Action Model, which models the robot's motion. This module calculates the Mbot's movement using odometry data that includes translation and rotational information. To account for inaccuracies in real-world robot movements, the model uses a probabilistic approach that incorporates noise into the motion calculations to realistically reflect these uncertainties. A random number generator is used in this process to simulate the effects of uncertainties on the robot's motion. The k1 and k2 are error parameters that were adjusted through trial and error during testing to correspond to the error distribution of robot rotation and translation. Throughout SLAM operation, the goal of tuning was to guarantee that the correct pose of the robot was consistently included in the pose distribution represented by the particle population. The model is illustrated by Algorithm 2.

Algorithm 2 Action Model (Update Algorithm)

Require: $x_{t-1}, \hat{x}_t, \hat{x}_{t-1}$

Ensure: x_t

$\Delta x \leftarrow \hat{x}_t - \hat{x}_{t-1}$

$\Delta y \leftarrow \hat{y}_t - \hat{y}_{t-1}$

$\Delta\theta \leftarrow \hat{\theta}_t - \hat{\theta}_{t-1}$ ▷ wrap to π for all angles

$\delta_{rot1} \leftarrow \text{atan2}(\Delta y, \Delta x) - \hat{\theta}_{t-1}$

$\delta_{trans} \leftarrow \sqrt{\Delta x^2 + \Delta y^2}$

$\delta_{rot2} \leftarrow \Delta\theta - \delta_{rot1}$

$\hat{\delta}_{rot1} \leftarrow \text{gaussian}(\delta_{rot1}, k_1\delta_{rot1})$

$\hat{\delta}_{trans} \leftarrow \text{gaussian}(\delta_{trans}, k_2\delta_{trans})$

$\hat{\delta}_{rot2} \leftarrow \text{gaussian}(\delta_{rot2}, k_1\delta_{rot2})$

$x_t \leftarrow x_{t-1} + \hat{\delta}_{trans}\cos(\theta + \hat{\delta}_{rot1})$

$y_t \leftarrow y_{t-1} + \hat{\delta}_{trans}\sin(\theta + \hat{\delta}_{rot1})$

$\theta_t \leftarrow \theta_{t-1} + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

b) Sensor Model: The Sensor Model computes the probability of sensor readings based on the state of each particle. It evaluates the alignment between the particle's state and the observed environmental data using the occupancy grid and current LIDAR scan data. This evaluation includes comparing each LIDAR ray against the map and assessing the probability that the particle is accurately positioned.

c) Particle Filter: The Particle Filter[11] is integral for localization within the SLAM system. It employs a particle filter algorithm, where a set of particles representing potential robot states is maintained and updated continually. Each particle's state is adjusted according to the Mbot's movements (informed by the Action Model) and sensor readings (guided by the Sensor Model). The Particle Filter also incorporates a resampling process, where particles are selected based on their weights, reflecting the likelihood of each particle's accuracy in representing the actual state of the robot. This component allows for initializing particles at specific poses or distributing them randomly across the map, providing flexibility in various operational contexts.

3) Combined Implementation: The SLAM system in Mbot framework integrates sensor inputs and motion data to construct a comprehensive map of the environment and accurately localize the robot within it. The use of a particle filter for localization effectively addresses the uncertainties in robot motion and sensor measurements. The system's modular design, which separates action modeling, mapping, and sensor processing, provides the flexibility and adaptability required for

efficient operation in various environments. A block diagram of how the SLAM system components interact is shown in Figure 4.

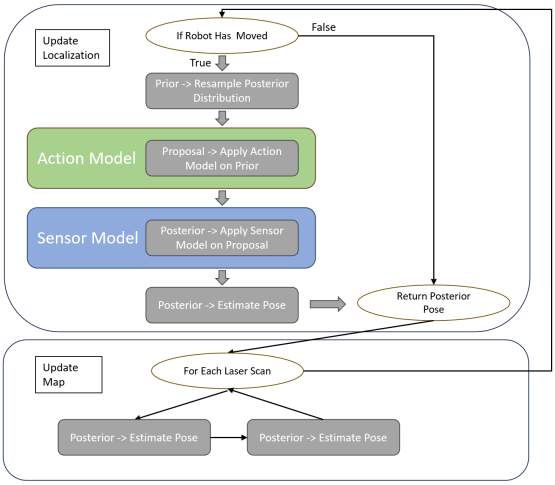


Fig. 4: Component Interactions Within SLAM

C. Planning and Exploration

Planning and Exploration phase serves as an elevated systemic abstraction, where the complex interplay of SLAM and motion controller assumes a central role. Utilizing the meticulously maintained map generated by SLAM, the system identifies unexplored regions within the robot's surroundings. Subsequently, the motion controller initiates a dynamic response, coordinating the execution of paths strategically devised to explore these unknown regions in the robot's environment.

1) Path Planning: The SLAM-generated occupancy map forms the foundational framework for path planning. Cells that have been observed and confirmed to be unoccupied, particularly those situated at a significant distance from cells identified as occupied, are deemed suitable for traversal. Conversely, cells falling outside this criteria are classified as non-traversable. The path planning process is executed through the utilization of the A* algorithm [12], (Algorithm 3 [13] in Appendix). In order to regulate the branching factor during the A* search, connectivity is established amidst cells that share either an edge/corner.

A customized version of Octile distance [14] is employed as the heuristic here, given by Equation 9.

$$h_{\text{cost}} = (d_x + d_y) + (\sqrt{2} - 2) \cdot \min(d_x, d_y) \tag{9}$$

Our A* implementation restricts exploration to movements along cardinal and ordinal headings. The heuristic employed has been designed meticulously to guarantee that an underestimation of costs is consistently maintained, ensuring admissibility. Optimality of output and execution time are thoroughly evaluated by testing on varied test scenarios.

2) Exploration: We employ a frontier-based exploration algorithm[15] to enable Mbot's autonomous exploration function. We employ a connected components search on the occupancy grid map to identify these frontiers. The search initiates from the robot's current location and utilizes Breadth-First Search (BFS) to discover all connected cells in free space, thereby avoiding unreachable frontiers. For each unvisited frontier cell, we methodically expand the frontier until all connected cells are identified. The process grid scans for more frontiers.

Upon identifying these frontiers, we calculate the centroid of each and then rank them based on their proximity to the robot. The nearest frontier is chosen as the target, followed by path planning to reach it.

This frontier-based exploration algorithm enables the robot to autonomously detect and navigate to new frontier areas, achieving comprehensive environmental exploration. The process takes into account the robot's location, the position of frontiers, and the accessibility of the areas surrounding these frontiers, thus ensuring the effectiveness and efficiency of exploration.

3) Localization with Unknown Initial Pose: We employ the Adaptive Particle Convergence Localization (APCL)[16] method to establish the initial localization of the MBot on a known map. Initially, a set of particles is randomly distributed across the map's open areas. As the robot senses its environment, each particle is assigned a weight based on the congruence of its hypothesis with the sensor data. Particles that closely align with the sensor readings are assigned higher weights, indicating a greater likelihood of representing robot's actual position.

With the robot's movement, a corresponding motion update and particle resampling are conducted for each particle. The algorithm ultimately converges, pinpointing the robot's initial position. This method ensures robust initial localization, effectively addressing the inherent uncertainties and variabilities of real-world environments, even in scenarios where the starting position is unknown or requires estimation.

III. Results

A. Motion and Odometry

1) Wheel Speed Calibration: Based on the mean and variance values for each parameter, we observe that the variances are within the range of 1e-7 to 1e-5. These variances are quite small, indicating that the calibration results are consistent and there is little variation between individual measurements.

TABLE I: Mean and Variance of Wheel Parameters

Wheel	Parameter	Mean	Variance
Left	Positive Slope	0.0611	3.4×10^{-7}
Left	Positive Intercept	0.0764	8.7×10^{-6}
Left	Negative Slope	0.0544	3.5×10^{-7}
Left	Negative Intercept	-0.0900	1.1×10^{-5}
Right	Positive Slope	0.0500	1.2×10^{-7}
Right	Positive Intercept	0.0636	1.3×10^{-5}
Right	Negative Slope	0.0563	9.1×10^{-8}
Right	Negative Intercept	-0.0480	5.4×10^{-6}

The primary potential sources of these variations could include: Tolerances in the mechatronic hardware components. Battery charge level and voltage fluctuations. Tire-ground friction difference. Sensor precision and consistency. Mechanical wear and looseness.

2) Odometry: Leveraging the IMU algorithm, we established a threshold of 0.1 to refine the computation of rotational angles within the odometry. We directed the Mbot to circumnavigate a square with 1-meter sides four times, returning to the starting point. The positional deviation between the concluding location and the initial point was measured to assess the performance of the odometry model. This experiment was

replicated twenty times to calculate the mean and variance along the x-axis, y-axis, and rotational angle.

TABLE II: Position Error Analysis

Parameter (Unit)	Improvement	Mean	Variance
Δx (cm)	With	3.4625	4.9623×10^{-2}
Δy (cm)	With	4.2600	9.0110×10^{-3}
$\Delta\theta$ (°)	With	12.0000	8.7000
Δx (cm)	Without	124.5000	161.2500
Δy (cm)	Without	19.7000	9.7000
$\Delta\theta$ (°)	Without	87.2000	32.8100

Prior to odometry optimization, the Mbot was unable to execute precise 90-degree turns, deviating from the intended path after just one circuit and ultimately failing to complete the four loops and return to the origin, with a substantial discrepancy between the actual and anticipated ending positions. Post-odometry optimization, the Mbot substantially improved at this task, with positional errors confined to within one-tenth of the vehicle's length. The significant reduction in variance post-optimization indicates a considerable increase in the model's robustness.

3) Wheel Speed PID Controller: We designed and implemented low-pass and PID filters for the wheel speed control of Mbot. Figure 5 illustrates the step response of Mbot's speed controller to a command input. It is evident from the figure that

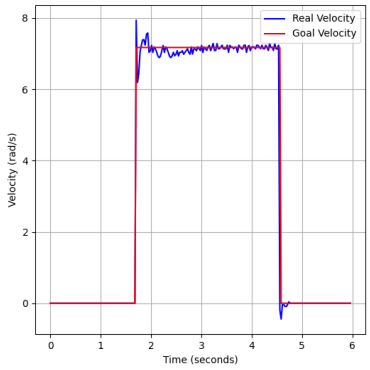


Fig. 5: Step response of wheel speed PID controller

the system is capable of swiftly accelerating to approximate the target velocity subsequent to a step change. The PID controller exhibits minimal overshoot. Upon reaching the target velocity, the actual velocity demonstrates commendable stability, with only minor fluctuations, thus maintaining proximity to the target velocity. Considering the rapid response, diminutive overshoot, and stability, we infer that the parameters are tuned to appropriate values. In tuning the PID parameters, we adhered to the following principles: an increase in P gain is required if the system response is sluggish. An increase in I gain should be applied to address persistent deviations. An increase in D gain is necessary if the system exhibits excessive oscillations. The final parameters of the Speed PID Controller are presented in TableIII.

4) Motion Controller: A constraint on maximum acceleration was instituted at $0.008\,\mathrm{m/s^2}$, effectively mitigating incidents of wheel slippage, stalling, and overturning. Figure 6

TABLE III: Parameters of Speed PID Controller

P Gain	I Gain	D Gain	Time Constant (s)	Frequency (Hz)
1.10	0.75	0.01	0.04	50

illustrates the robot's dead reckoning pose as it circumnavigates a square for 4 times. The commencement of the trajectory is indicated by a green arrow, the terminus in a red arrow. Notably, the robot's estimated pose shows minimal drift, with the starting and ending positions nearly superimposed. Experi-

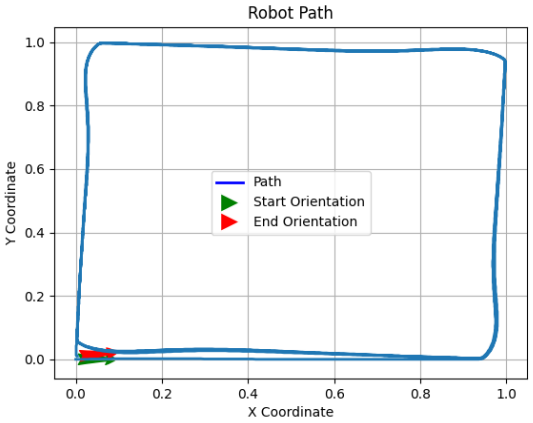


Fig. 6: Dead-reckoning odometry as Mbot drives around a square for 4 times

mental results in real environment, as detailed in Table II 'With Odometry Improvement', reveal that the final position error is confined within a tenth of Mbot's size relative to the projected error. These outcomes affirm the motion controller's capability to adhere to intricate trajectories punctuated by multiple waypoints. Figure 7 exhibits Mbot's linear and rotational velocities as it drives one loop around the square.

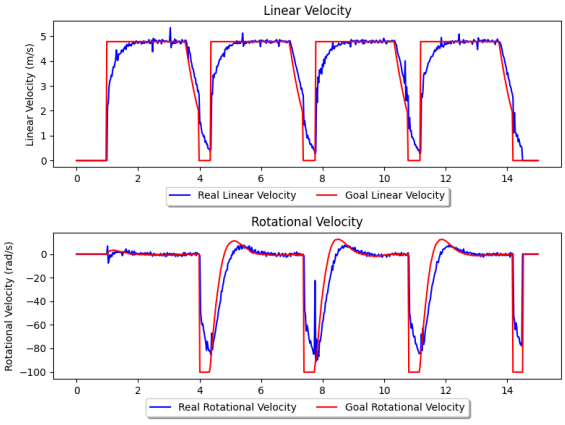


Fig. 7: Linear and rotational velocities in one loop around the square

B. Simultaneous Localization and Mapping (SLAM)

A reliable and effective algorithm was developed through a methodical and staged process that involved building an occupancy grid, integrating Monte Carlo Localization, and completing a full SLAM system. The system's precision in mapping, localization, and navigation in complex environments is demonstrated by various metrics and visual aids. This section presents the results of the mapping, action model, sensor model, and particle filter, as well as the overall SLAM system components. The findings are supported by data analysis, graphical representations, and comparison analyses against ground-truth standards.

1) Mapping: Our initial challenge was to construct an occupancy grid map using robot poses from log files. The grid utilized was highly complex, with cells no larger than 10 cm and log-odds values ranging from -127 to 127, providing a detailed representation of spatial probability. The program facilitated the mapping task using the ground-truth postures from the given log file. The generated map shown in Figure 8a reproduces map.

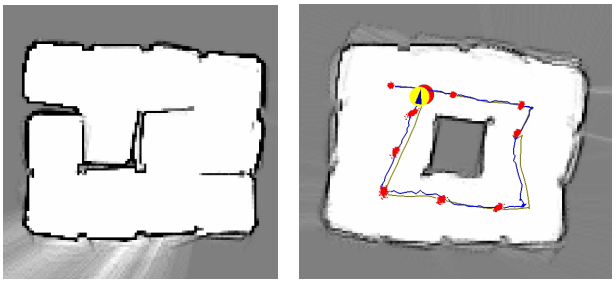
2) Monte Carlo Localization: For MCL, we implemented the action, sensor models, and various particle filtering functions. The action model included odometry and laser scans, utilizing wheel encoders and additional sensor inputs such as the IMU or computer vision techniques. The sensor model computed the pose likelihood by comparing laser scan data to an occupancy grid map.

a) Action Model: The equations of the action model include parameters for uncertainty, as shown in the table IV below. These values were determined through experimental tuning and simulation trials, with the goal of achieving a balance between accuracy and computational efficiency.

TABLE IV: Uncertainty Parameters for Particle Filter

Parameter	Description	Value
k1	Uncertainty for Rotation	0.005
k2	Uncertainty for Translation	0.005

b) Sensor Model and Particle Filter: The time required to update the particle filter for different particle counts was recorded as shown in the table V. An estimation was made for the maximum no. of particles sustainable at 10Hz, which is about 2250. Fig. 8b is the drive square task with 300 particles, which shows our implementation of the Monte Carlo localization model.



(a) (b)

Fig. 8: (a) Map from drive maze.log. (b) 300 particles at regular intervals along the path.

TABLE V: Number of Particles versus Time to Update

Numbers	Time (s)
100	11.14
300	17.48
500	26.17
1000	45.74
2250 (estimated)	100.00

3) Combined Implementation: The accuracy of the SLAM system was evaluated by comparing the calculated poses with the ground-truth data, the poses are shown in figure 9. The trajectories estimated from SLAM w.r.t to the ground-truth pose is collected by MoCAP system, and the color reflects the absolute pose error(APE) evaluated by evo [17]. To evaluate the accuracy of the system, various statistical metrics were computed, as shown in table VI.

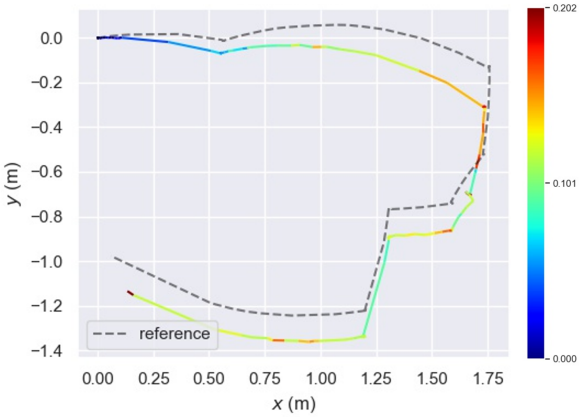


Fig. 9: SLAM Pose w.r.t. Ground Truth with Error Map

TABLE VI: Accuracy

Metric	Value
RMSE	0.120 547
SSE	2.731 930
STD	0.059 096

C. Planning and Exploration

1) Path Planning: We first use the SLAM method to construct the map of the environment, and then use the A* algorithm for navigation. As shown in Figure10, the green line represents the path planned by the A* algorithm, and the blue line represents the actual route traveled by the MBot. It is evident that the robot's actual path closely follows the expected path, achieving good path planning and navigation capabilities.

We use six maps with different typical features to further test the A* algorithm, and the results are shown in TableVII. For narrow maps, our A* algorithm searches very slowly. We optimize this issue by modifying the heuristic function of the A* algorithm. We use a weighted heuristic method, preferring routes that seem to lead directly to the goal, even if they are longer. Although this sacrifices the optimality of the path,

TABLE VII: Path planning execution times

Test Cases	Timing (us)			
	Min	Max	Mean	STD
Empty Grid	67	119475	17579.5	75756
Filled Grid	18	49	37	10.257
Narrow Grid	102	362164	115559	143948
Wide Grid	1292	35731	13302	15873
Convex Grid	476	1967	1221.5	745.5
Maze Grid	3759	6370	4909	1066.8

making the route to the destination longer, it achieves faster search times.

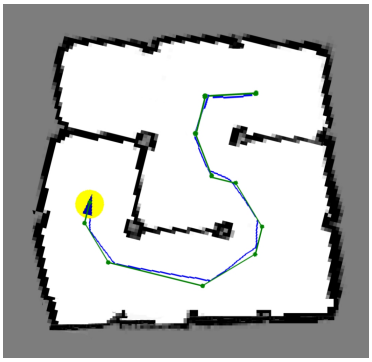


Fig. 10: Planned path (green) and Actual path (blue) driven by Mbot

2) Exploration: Building upon the foundation of A* path planning and the SLAM module, we implemented an automatic exploration feature using a frontier search algorithm. The robot begins its journey from the starting point, thoroughly explores the environment, and eventually returns to a location near the starting point.

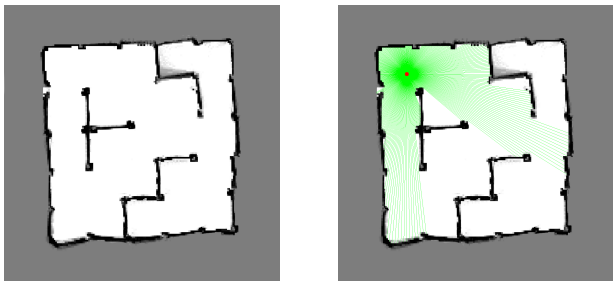
The resulting map, as illustrated, closely resembles the actual environment and is quite comprehensive. However, our algorithm encounters challenges in complex mazes, where frontier search and path planning in narrow terrains may require substantial computational resources and time. This inefficiency in the algorithm can lead to the robot becoming stuck in the middle of the map, unable to complete the entire exploration.

3) Initial Localization: we validate Mbot's capability to self-localize within a pre-mapped environment from an unknown initial position. As shown in Figure 11b, Mbot is able to accurately identify and converge upon its true pose in the exploration process. Crucially, before the localization converged, the lower motion controller relies on command paths provided by the exploration agent. Once the localization converges, control is handed over to the motion planner server for precise path navigation. The outcomes of this section not only exhibit the robot's high adaptability to the environment but also affirm the effectiveness of our localization methodology in addressing the challenges of unknown starting positions.

IV. Discussion

A. Odometry and Motion Controller

We deduce that the intrinsic noise of the sensors, the randomness of the environment our Mbot was exposed to, and



(a) (b)

Fig. 11: (a) Mbot's exploration in a Maze. (b) Localization with unknown initial pose.

the overall unevenness of the floor were all roughly dispersed. We were able to minimize these non-systematic mistakes through the calibration technique, which allowed our robot to accomplish all of its objectives with minimal interference from the motors.

B. SLAM

The maps generated were found to be less accurate than expected. This inaccuracy can be attributed to various factors such as sensor noise, inadequate sensor resolution, or inefficiencies in the mapping algorithm. It could be improved by refining the mapping algorithm to better handle sensor noise and interpret sensor data more accurately. The SLAM system exhibited larger-than-expected localization and mapping errors. These errors could be due to compounded inaccuracies in both the action and sensor models, as well as potential shortcomings in the particle filter algorithm. The possible solution includes developing more robust Action Model that better accounts for the robot and environmental factors and enhancing the Sensor Model to more accurately interpret environmental data.

C. Exploration and Planning

In enhancing path planning, it is noteworthy to address certain improvements. Particularly, we employed an A* algorithm to explore paths in the (x, y) space, prioritizing minimization of length over reduced turning. A potential refinement could involve adapting the A* implementation to explore paths within a 3D (x, y, θ) space rather than a 2D (x, y) space. Moreover, the runtime of A* was observed to be sub-optimal, indicating the need for potential enhancements through code profiling and function complexity optimization. Exploring more sophisticated path planning methods like RRT could be considered as an alternative.

V. Conclusion

Throughout the MBot Lab, we have implemented robot autonomy at different levels of abstraction by developing motion, SLAM, and exploration capabilities. Through a variety of tests, we were able to demonstrate these autonomous capabilities, leading to the ability of the robot to identify its position from unknown pose and undertake exploration tasks autonomously.

REFERENCES

[1] P. Gaskell, J. Pavlasek, T. Gao, A. Narula, S. Lewis, and O. C. Jenkins, "Mbot: A modular ecosystem for scalable robotics education," *arXiv preprint arXiv:2312.00962*, 2023.

[2] L. Huang, "Review on lidar-based slam techniques," in *2021 International Conference on Signal Processing and Machine Learning (CONF-SPML)*. IEEE, 2021, pp. 163–168.

[3] J. Meng, A. Liu, Y. Yang, Z. Wu, and Q. Xu, "Two-wheeled robot platform based on pid control," in *2018 5th International Conference on Information Science and Control Engineering (ICISCE)*. IEEE, 2018, pp. 1011–1014.

[4] K. Maenaka, "Current mems technology and mems sensors - focusing on inertial sensors-," in *2008 9th International Conference on Solid-State and Integrated-Circuit Technology*, 2008, pp. 2371–2374.

[5] 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. IEEE, 1996, pp. 423–428.

[6] D.-E. Ameddah and T. Benmokrane, "A comparison of a classical pid and sliding mode: traction control for fast wheeled mobile robot," *International Journal of Automation and Control*, vol. 4, no. 1, pp. 65–83, 2010.

[7] ROB550, "Rob550f23odometrypathfollowing," 2023, https://drive.google.com/file/d/14zxoiSBidyZR0S413N2yn5-63M5wuJx8/view?usp=drive_link.

[8] D. Kiss and G. Tevesz, "The rtr path planner for differential drive robots," , vol. 2, no. 2 (4), pp. 16–22, 2020.

[9] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, 1999, pp. 1322–1328 vol.2.

[10] A. A. Housein and G. Xingyu, "Simultaneous localization and mapping using differential drive mobile robot under ros," in *Journal of physics: conference series*, vol. 1820, no. 1. IOP Publishing, 2021, p. 012015.

[11] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEE Proceedings F (Radar and Signal Processing)*, vol. 140, no. 2, pp. 107–113, 1993.

[12] S. J. Russell and P. Norvig, *Artificial intelligence a modern approach*. London, 2010.

[13] ROB550, "Rob550f23pathplanning," 2023, https://drive.google.com/file/d/16akBYVFjg96TmF69D3-EXv8N97kSmOhWy/view?usp=drive_link.

[14] Y. Björnsson and K. Halldórsson, "Improved heuristics for optimal pathfinding on game maps," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 2, no. 1, 2006, pp. 9–14.

[15] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Proceedings of the second international conference on Autonomous agents*. ACM, 1998, pp. 47–53.

[16] A. Garcia, F. Martín, J. M. Guerrero, F. J. Rodríguez, and V. Matellán, "Portable multi-hypothesis monte carlo localization for mobile robots," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1933–1939.

[17] M. Grupp, "evo: Python package for the evaluation of odometry and slam," 2017, https://github.com/MichaelGrupp/evo.

APPENDIX

A. Path Planning

Algorithm 3 A* Algorithm

Require: start, goal(n), h(n), expand(n)
Ensure: path

if goal(start) = true **then return** makePath(start)
end if
open ← start
closed ← null
while open is not equal to null **do**
  sort(open)
  n ← open.pop()
  kids ← expand(n)
  for all kid **in** kids **do**
    kid.f ← $(n.g + 1)$ + h(kid)
    if goal(kid) = true **then** makePath(kid)
    end if
    if kid ∩ closed = null **then** open ← kid
    end if
  end for
  closed ← n
end while
return null

B. Motion Controller

Leveraging the LCM library for seamless inter-process communication, the controller encompasses distinct maneuver controllers tailored for various motion scenarios. These controllers include:

- *StraightManeuverController()*, employs PID control for forward and rotational movements, ensuring precise adjustments to robot speed and heading.
- *TurnManeuverController()* focuses on PID control for accurate rotations, featuring a specialized method for the final turn to a specified heading.
- *SmartManeuverController()* integrates PID control for both forward and angular velocities, with the introduction of an acceleration limit which prevents abrupt accelerations, leading to smoother and more controlled motions.

The motion controller's main function involves initializing LCM communication, subscribing to relevant channels such as odometry and pose, and processing received paths to update the robot's motion accordingly.

C. Gripper assembly

We designed a Rack-Pinion mechansim assisted forklift gripper for lifting boxes during warehouse pick and place tasks. Our design consists of Pinion (2 Nos.), Rack (2 Nos.), Fork (1 Nos.), and Motor box (2 Nos.; to house XL320 motors). The combined assembly can be seen in Figure 12 below.

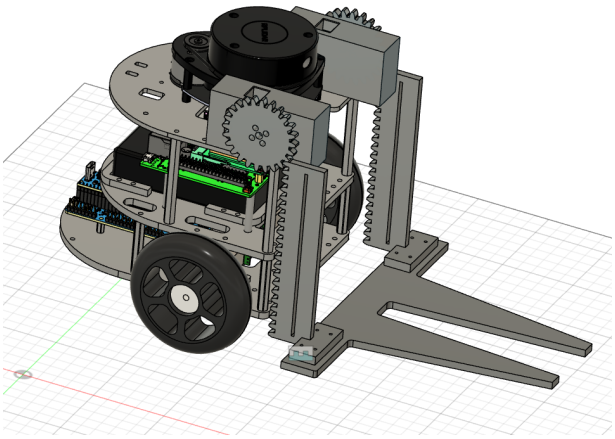


Fig. 12: Gripper assembly on MBot

The forks on the front of the truck move up and down the vertical track i.e. the rack using gear tooth meshing (gear ratio of 1:1) with the pinions. The maximum load lifting capacity is 250-350 grams, suffice corresponding to the weight of the blocks (approx 30-40 grams) with a factory of safety equal to 10. OLLO rivets were used for joining the pinions with the XL320 motors. Th racks were joined using an L-shape wide M8 screwed base to the fork in order to reduce the gear teeth pressure/rubbing.