BotLab Report - Section PM Team 5

Jayaprakash Harshavardhan, Nithish Kumar, Surya Pratap Singh
{jharsh, nithishk, suryasin}@umich.edu

***Abstract*—This project aims to create software and algorithms for an MBot, which is a two-wheel differential drive robot, that enables path traversal, autonomous mapping, and SLAM-based localization. We use motor control using particle filters, mapping, PID tuning, and odometry, together with path planning features. This enables the mbot to perform a variety of task autonomously like traversing a unknown environment and mapping it out. Apart for just traversing we also aim to To facilitate object identification and move it as per the task . To accomplish this , we also design and construct a forklift through 3D printing. Our group also takes part in four competition events to assess the overall system's performance with these components ..**

I. Introduction

Growing advancements in robotics technology emphasize the benefits of self-governing mobile robots across several domains and use cases, particularly in search and rescue and exploration. Mobile robots were first used in disaster relief, as demonstrated by Delmerico [1] and Kruijff-Korbayová [2], who effectively used them in earthquake recovery operations. The use of mobile robots in intricate, uncharted areas was realized by Yamauchi [3] and Endo [4]. Still, an environment map is necessary for these tasks. Autonomous exploration of an unfamiliar region is necessary for mobile robots to acquire a comprehensive and precise map. Mobile robots must create maps and achieve location in order to participate in the Defense Advanced Research Projects Agency's Underground Challenge [5], which places a high priority on robot autonomy in exploration.

The most common examples cited for multiple ground robots working together would most definitely be the Amazon warehouse robots. But even these robots have a pre-made layout of the land given from some other source. But there are many cases in the real world where the pre-made map of the environment is not available and the bot may have to move in uncharted territories. Autonomous path planning is a huge need of the hour as the bot may move into an environment with too many obstacles and the bot may scan its surroundings in real time. The need for the algorithms to be computationally efficient while also being robust enough so that possible expensive equipment is not damaged in transit.

This project makes use of the traditional Mbot design, which consists of two parallel wheels with a caster at the back and two differential drive motors. One MBot's initial design is modified to include a forklift attachment for block movement. This robot also has a 2D-LIDAR, whose laser beams are used in SLAM, for environment perception. This is a critical aspect of the robot as it enables, mapping of the environment which can be used for autonomous navigation later. A 3-DOF odometry system based on motor encoders and an IMU are utilized for posture estimation. Using both give a very accurate estimate on the location of the robot. We work on the robot through remote SSH access, which communicates pertinent data via LCM. High-level communication between the MBot's firmware and the sensing instruments is made possible by the use of a Jetson Nano board, and the Pico Controller runs the firmware to operate the robot and read data from the IMU.

This project focuses on improving autonomous implementations such motion control, SLAM, and optimal path-finding, as well as firmware code (for better odometry and motor control). We go over how each of these components was implemented, the findings from each, and the inferences we may make from the information.

II. Methodology

A. Motion Controller and Odometry

1) $Caliberation$ - We have to calculate each motor's PWM (Pulse Width Modulation) command from a wheel velocity using the PWM calibration function, which accepts the polarities and calibration data from an MBot. When both encoders operate, the polarity is accurate, and when we can ascertain that neither the punch nor the slope of the calibration function for a single motor has a magnitude greater than 0.10, we deem a calibration procedure successful.
2) $Odometry$ - We compute the angular velocity of the mbot using the measured velocities of each wheel based on the encoder ticks of its motor in the basic functionality. The below equations specify formulas for the odometry pose computations, linear body velocity $\hat{v}_x$, and angular body velocity $\hat{w}_z$. R is the wheel radius, $\hat{w}_L$ is the angular velocity of the left wheel, and $\hat{w}_R$ is the angular velocity of the right wheel. We only use the φ from the IMU data in the revised odometry to calculate

$\hat{w}_z$. A revised calculation can be seen in the below equation for $\hat{w}_z$. T is the duration of the firmware loop. All other computations remain the same.

$$\hat{v}_x = R(\hat{w}_L - \hat{w}_r)/2$$
$$\hat{w}_z = -(\hat{w}_L + \hat{w}_r)/2$$
$$x_t = x_{t-1} + \hat{v}_x \cos\theta_{t-1} T$$
$$y_t = y_{t-1} + \hat{v}_x \sin\theta_{t-1} T$$
$$\theta_t = \theta_{t-1} + \hat{w}_z T$$
$$\hat{w}_z = (\varphi_t - \varphi_{t-1})/T$$

3) $MotorControl$ To develop a closed-loop controller for wheel velocity controllers, we join the feed-forward term with a feedback term based on PID. Moreover, we also include low-pass filters to lessen noise in the output and reference terms. These consist of the measured velocities of every wheel, the commanded v and w, and the PMW output following the combination of the feed-forward and feed-back terms. The three parameters of a PID controller which are, the derivative term K_D, percentage term K_P, and integral term K_I are the same for both wheels even though we have distinct PID filters for each wheel's velocity. To address an integral windup, we utilize distinct filters for each motor's K_I gain and reset it if it accumulates to 1.0. The difference between the measured $\hat{w}_L$ and the required wheel velocity w_L is our input to the PID controller when we consider the left wheel which is similar to the case of the right wheel. Before the total is fed into the PMW calibration function, the PID output is added to w_L (or subtracted, if the wheel polarities multiply by -1). The tuning procedure is supported by the graphing of w_L and $\hat{w}_L$ parameters. This process starts with setting K_P,K_I and K_D to 0 and execute a straight path, $\hat{w}_L$ reaches w_L after a delay. Then we start by increasing K_P until $\hat{w}_L$ doesn't come close to w_L. If slight oscillations appear reduce K_P to the previous hypothesis. Then increase K_D until the bot appears to jerk, as soon it starts to jerk reduce K_D to the previous observation. Then increase K_I until the steady state value of $\hat{w}_L$ is close to w_L. In our initial attempt at tuning, we add filters with variable time constants to improve acceleration and turns after fine-tuning the PID parameters without low-pass filters. This results in sharp swerves in trajectories that resemble squares. In our second try, we start by progressively increasing the low-pass filters' time constants from 0 until abrupt velocity shifts are executed smoothly. The square path veers somewhat as a result of this. After applying the aforementioned technique to adjust our PID values, we observe that the swerves vanish. Shifts in v and w are carried out smoothly, a square run has no gaps, and the graph of w_L and $\hat{w}_L$ for a straight path shows what we expect (a minor branch, no oscillations, and an accurate steady state).

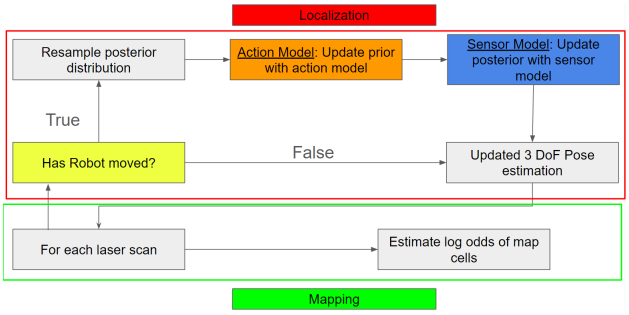

Fig. 1. SLAM components

4) $MotionControl$ We depend on the given $SmartManeuverController$ works reasonably well with our revised odometry and gives an accurate enough movement to reach the intended target.

B. Simultaneous Localization and Mapping (SLAM)

In this project, we focus on improving sophisticated simultaneous localization and mapping (SLAM) systems for autonomous robots, especially in warehouse conditions. Here we have used use occupancy grid-based mapping and Monte Carlo localization algorithms.

1) Mapping: First, we have the Mapping module that uses LIDAR sensor data to continuously update the occupancy grid map of the bot's surrounding environment. Each laser ray is processed individually during LIDAR scanning. As these rays interact with their environment, the Mapping component adjusts the log-odds values in the relevant map cells. Cells through which the laser passes are given a rise in odds, indicating a higher occupancy likelihood. Cells that travel through the laser without being struck experience a fall in odds, suggesting a lesser possibility of occupancy. This ongoing adjustment process generates a detailed and constantly changing map of the surroundings. To ensure perfect ray casting on the occupancy grid, we adopted Bresenham's line algorithm, taught in the lecture.

2) Monte Carlo Localization: Monte Carlo Localization (MCL) represents a localization method built upon particle filters. Its implementation necessitates three essential components: an action model for pose prediction, a sensor model to estimate pose likelihood based on sensor data, and diverse particle filtering functions such

as sample generation, particle weight normalization, and determination of the weighted mean pose.

a) Action Model: The Monte Carlo Localization (MCL) in our Botlab begins with the Action Model, which simulates the robot's movement. This component utilizes odometry data containing both translation and rotation details to calculate the robot's motion. To address real-world movement inaccuracies, a probabilistic method is employed, introducing noise into the motion calculations to realistically depict uncertainties. A random number generator is utilized to mimic the effects of these uncertainties on the robot's motion. The parameters k1 and k2, representing error, are fine-tuned through iterative testing to align with the error distribution of robot rotation and translation. Throughout the SLAM process, this tuning aims to ensure the accurate inclusion of the robot's pose within the particle population's pose distribution. The model is outlined in Algorithm 1.

Algorithm 1 Action model

Pre-requisites: $\hat{x}_t, \hat{x}_{t-1}, x_{t-1}$
Output: x_t, y_t, θ
$(\Delta x, \Delta y, \Delta\theta) \leftarrow (\hat{x}_t - \hat{x}_{t-1}, \hat{y}_t - \hat{y}_{t-1}, \hat{\theta}_t - \hat{\theta}_{t-1})$
Equations:
$\delta_{Rot1} \leftarrow atan2(\Delta y, \Delta x) - \hat{\theta}_{t-1})$
$\hat{\delta}_{Rot1} \leftarrow Normal\_dist(\delta_{Rot1}, k1 * \delta_{Rot1})$
$\delta_{trans} \leftarrow \sqrt{(}\Delta x^2 + \Delta y^2)$
$\hat{\delta}_{trans} \leftarrow Normal\_dist(\delta_{trans}, k2 * \delta_{trans})$
$\delta_{Rot2} \leftarrow \Delta\theta - \delta_{Rot1}$
$\hat{\delta}_{Rot2} \leftarrow Normal\_dist(\delta_{Rot2}, k1 * \delta_{Rot2})$
$x_t \leftarrow x_{t-1} + \hat{\delta}_{trans} * cos(\theta + \hat{\delta}_{Rot1})$
$y_t \leftarrow y_{t-1} + \hat{\delta}_{trans} * sin(\theta + \hat{\delta}_{Rot1})$
$\theta_t \leftarrow \theta_{t-1} + \hat{\delta}_{Rot1} + \hat{\delta}_{Rot2}$

b) Sensor Model: The Sensor Model determines the likelihood of sensor readings given the state of each particle. It assesses the possibility between the particle's state and the observed environmental data by analyzing the occupancy grid and the present LIDAR scan data. This assessment involves scrutinizing each LIDAR ray against the map and gauging the probability of the particle's precise positioning.

c) Particle Filter: The Particle Filter plays a crucial role in localization within the SLAM system, employing a particle filter algorithm to manage and continuously update a collection of particles representing potential robot states. Each particle's state is modified based on the robot's movements (informed by the Action Model) and sensor readings (guided by the Sensor Model). Additionally, the Particle Filter incorporates a resampling step, wherein particles are chosen according to their weights, indicating the likelihood of each particle accurately representing the robot's actual state. This feature enables the initialization of particles at specific poses or their random distribution across the map, offering adaptability in diverse operational scenarios.

The SLAM system within the Mbot framework merges sensor inputs and motion data to generate a detailed map of the environment and precisely determine the robot's position within it. Employing a particle filter for localization effectively handles uncertainties arising from robot motion and sensor measurements. The system's modular architecture, which divides action modeling, mapping, and sensor processing, offers the necessary flexibility and adaptability for efficient operation across different environments. Figure 1illustrates a block diagram demonstrating the interaction among the SLAM system components.

C. Planning and Exploration

1) A Algorithm:* A* combines the perks of Dijkstra's algorithm and the greedy best-first search. It creates a balance between investigating potential paths (using heuristic knowledge) and assuring optimality by accounting for goal costs incurred thus far.

A* employs a heuristic function h(n) to estimate the cost of getting from node n to goal. This heuristic helps the search process by offering an informed estimation of how likely a specific node is to attain the goal.

Cost Evaluation A* assesses each node using two factors. Goal Cost (g-cost): This is the cost incurred from the start node to the current node. Heuristic Cost (h-cost): This is the projected cost of getting from the current node to the desired node. Total estimated cost (f-cost): A* adds the goal and heuristic costs to calculate the total cost of reaching the goal node from the current node. It prioritizes nodes with lower total costs by using the equation f(n) = g(n) + h(n).

A* has two sets: The Open Set contains nodes that have been detected but not yet evaluated. Nodes in the open set are acceptable for expansion. A closed set contains nodes that have previously been analyzed. When a node is evaluated, it moves from the open set to the closed set.

Node Expansion: A* expands nodes in the open set by taking into account their neighbors. It selects the node with the lowest overall cost (f-cost) for expansion, giving priority to nodes closest to the objective. A* ends when the goal node is located in the open set. At this step, the path from the start node to the goal node is recreated using the parent pointers established throughout the search.

Optimality: A* assures finding the shortest path under specific conditions: The heuristic function h(n) must be acceptable, which means it should never overestimate the true cost to reach the goal. The heuristic function must also be consistent (or monotonic), which means that the estimated cost from any node to the goal is always less

than or equal to the cost of reaching a neighboring node plus the estimated cost from that neighboring node to the goal.

Algorithm 2 A-Star Algorithm

```
Input: start, goal(n), h(n), expand(n)
Output: path
if goal(start) then then
    return makePath(start)
open ← start; closed ← ∅
while open ≠ ∅ do
    sort(open)
    n ← open.pop();
    kids ← expand(n)
    for all kid in kids do
        dist ← obstacleDist(kid)
        if dist > radius then
            if kid is not in closed then
                kid.f_cost ← g_cost + h_cost
                open ← kid
            if goal(kid) == true then
                return makePath(kid)
    closed ← n
return ∅
```

2) Exploration: Exploration comprises implementing processes to navigate and chart an environment, beginning with a start point and end with a return to the starting position. Several major components of the preconfigured implementation are described below:

Frontier Identification: A frontier denotes the limit between explored and undiscovered areas. The process for locating all frontiers and charting a journey to the nearest one is pre-established.

State Machine Integration: By including a state machine, the exploration program can determine the robot's current state and status, allowing for more informed decision-making about future actions.

The Motion Planning server is in charge of creating routes for the exploration program and determining the practicality of assigned destination sites.

D. Forklift

The MBot's forklift mechanism lifts and stacks cardboard crates using a timing belt system. A timing belt of 400mm length with a 3:1 ratio is utilized to transfer motion from the motor. We used USB to TTL adapter and AX to XL320 adapter method and installed mbot_xl320_library to rotate the servo motor. Since we are using 3:1 ratio timing gears the servo only has to provide 1.5 times full circle and instead of directly providing rest to 0 position.

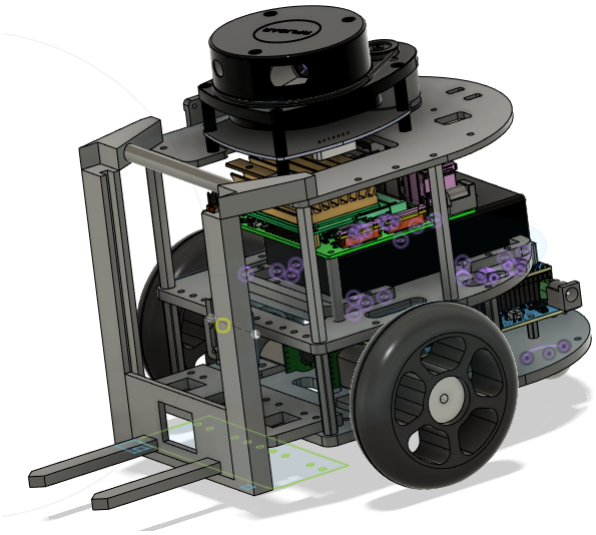

Fig. 2. MBOT CAD Design with Forklift

We have a U-frame that is utilized to firmly balance the fork to a separate 3D-printed base plate on which the motor is mounted. The forklift's fork is attached to the timing belt. To ensure reliability, we selected to 3D print the majority of the assembly in PLA with a 20% infill. We used nylon to 3D print the shaft that holds the other timing gear. Daisychain Dynamixel XL-320 motors are utilized for the movement of the timing belt. We 3D printed a bracket to attach the motor to the center right of the Mbot's front, just far enough away to avoid blocking the camera. To improve stability, the u-frame connects to the top of the Mbot using a clamp, preventing unwanted movements, wobbles, and rotation.

III. Results

A. Motion Controller and Odometry

For the $MotorControl$, the below tables detail the analysis of the motor calibration slope and intercept from five concrete floor calibrations of our team's MBots.

Positive case	slope(R)	Intercept(R)	slope(L)	Intercept(L)
MEAN	0.0594	0.052363	0.05137	0.05936
VARIANCE	2.543E-7	5.9144E-5	5.996E-7	3.3796E-5

Negative case	slope(R)	Intercept(R)	slope(L)	Intercept(L)
MEAN	0.05273	-0.06774	0.058872	- 0.0628
VARIANCE	1.874E-7	1.47E-5	3.278E-7	1.044E-5

We add a PID to both the left and right motors and also add the low pass filter to remove the noise. However, we were getting a lot of errors while applying a low pass filter hence we decided to get rid of it as even without it we were able to get decent data. The below

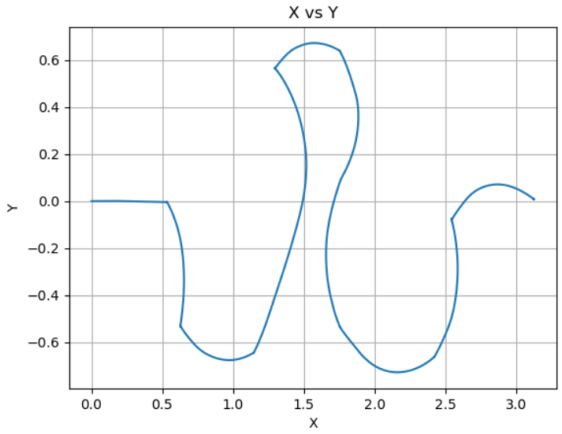

Fig. 3. Without changes made in motion controller

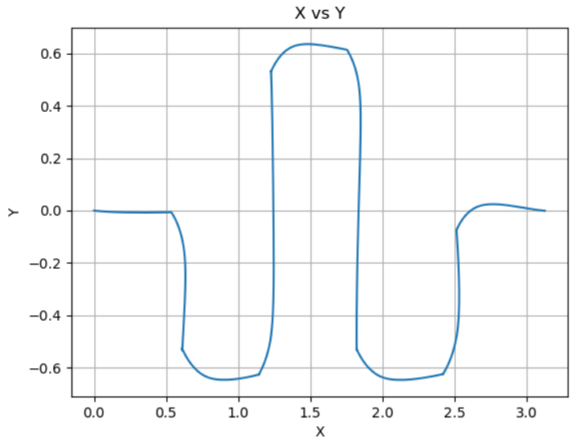

Fig. 4. After changes made to motion controller

table shows the gain employed by both motors to ensure smooth motion.

K_P	K_I	K_D
1.0	0.0001	0

We show our plots with and without our changes in motion controller while trying through the maze in checkpoint 1. Given in Fig 3 and 4, we can clearly observe a huge improvement after adding the changes and hence emphasizes on the value of adding PID controller to the motor to ensure smooth locomotion . Odometry output is shown by the blue line in a (x, y) coordinate frame.

To fine-tune the motor control further and ascertain the accuracy of the refined motion controller, we use the Motion Controller to direct the robot to walk around a square 4 times. Fig 5 shows the resulting odometry plots

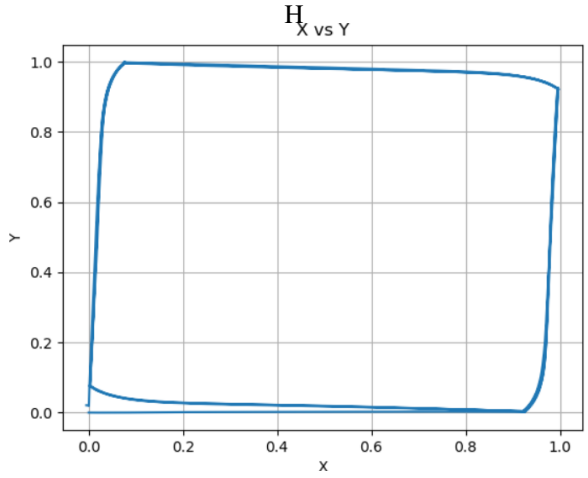

Fig. 5. Odometry data for drive square 4x scenario

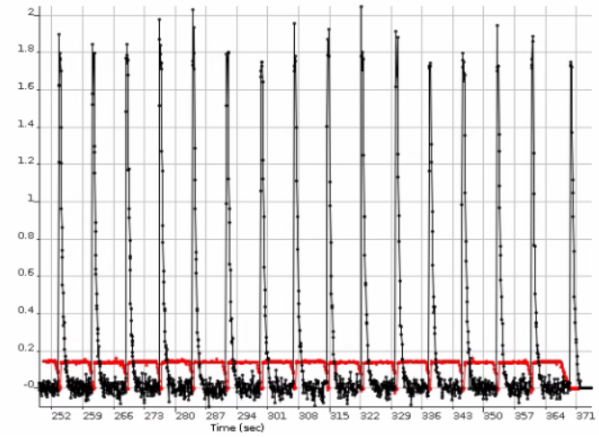

Fig. 6. Measured linear and angular body velocity for Fig5

and the odometry readings following traversal, which are anticipated to be x = 0, y = 0, and θ = 0.

Lastly, we plot the measured linear and angular body velocities, as seen in Fig 6. We can see some noise in the angular velocity measurements which can be improved upon later and subsequent changes to the low pass filter which we have trouble implementing. The black line in Fig 6 is for measured angular velocity and correspondingly the red line propagates to the measured linear velocity.

B. Simultaneous Localization and Mapping (SLAM)

A dependable and efficient algorithm was crafted via a systematic and step-by-step approach, encompassing the creation of an occupancy grid, integration of Monte Carlo Localization, and development of a comprehensive SLAM system. The system's accuracy in mapping, localization, and navigation within intricate environments is evidenced through diverse metrics and visual aids.

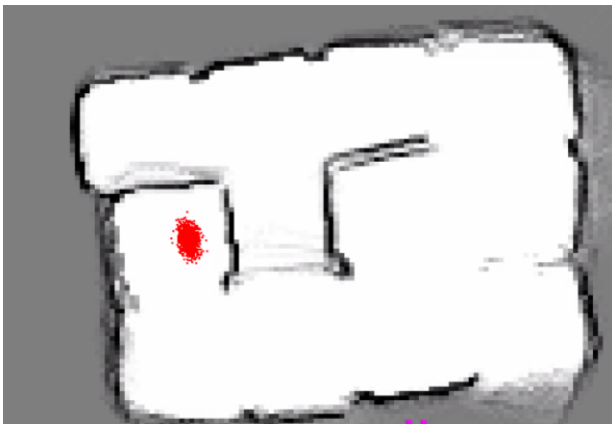

Fig. 7. Map for drivemaze.log

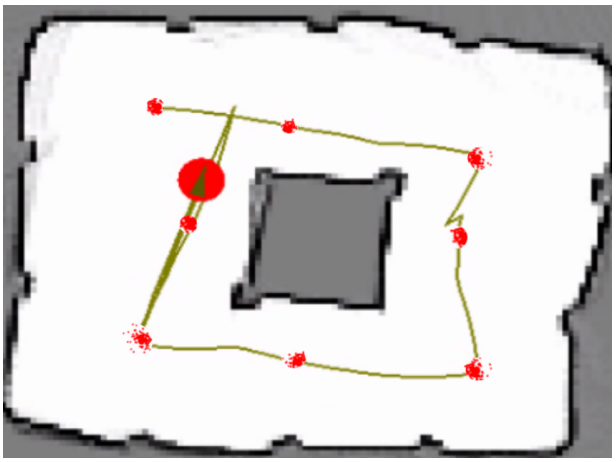

Fig. 8. 300 particles drawn at regular intervals

This section observes the outcomes of mapping, action model, sensor model, and particle filter, alongside the overall components of the SLAM system. Data analysis, graphical depictions, and comparative assessments against established ground-truth benchmarks substantiate the conclusions.

1) Mapping: Our primary hurdle involved constructing an occupancy grid map using robot poses extracted from log files. The grid employed was notably intricate, featuring cells no larger than 10 cm and log-odds values spanning from -127 to 127, ensuring a meticulous portrayal of spatial probability. The Botlab-gui software aided in the mapping endeavor by utilizing the ground-truth postures from the provided log file. Figure 7 illustrates the resulting map.

2) Monte Carlo Localization: We integrated action and sensor models as well as diverse particle filtering functions into MCL. The action model incorporated odometry and laser scans, making use of wheel encoders and other sensor inputs like the IMU or computer vision methods. Meanwhile, the sensor model determined pose likelihood by analyzing laser scan data against an occupancy grid map.

a) Action model: The action model equations contain uncertainty parameters, detailed in Table I below. These values were established via experimental tuning and simulation trials to strike a balance between precision and computational speed.

TABLE I
UNCERTAINTY IN PARAMETERS

Parameter	Description	Value
k1	Rotation uncertainty	0.0025
k2	Translation uncertainty	0.0005

b) Sensor model and Particle Filter: The duration for updating the particle filter across various particle quantities was documented, as illustrated in Table II. An assessment was conducted to determine the maximum number of particles feasible at a 10Hz rate, estimated to be approximately 2050. In Fig. 8, the drive square task employing 300 particles demonstrates our utilization of the Monte Carlo localization model.

TABLE II
NUMBER OF PARTICLES VS TIME TO UPDATE

Particle numbers	Time(s)
100	12.11
500	26.89
1000	47.38
2050 (approx)	97.80

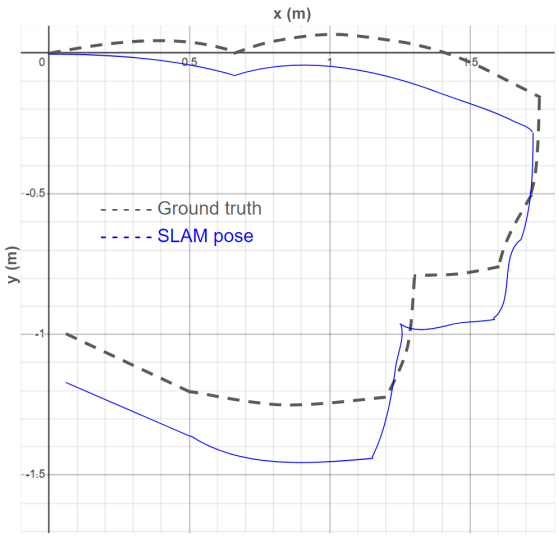

Fig. 9. SLAM pose vs ground truth pose

The precision of the SLAM system underwent assessment through a comparison between the calculated poses and the ground-truth data, as depicted in Figure 9. Various statistical metrics, as detailed in Table III, were computed to gauge the system's accuracy.

TABLE III
ACCURACY

Accuracy metric	Value(m)
RMSE	0.1463
Max. Abs	0.1725

C. Planning and Exploration

1) Planning: The A-star algorithm excels at accurately navigating paths without colliding with any obstacles, although its computation time was a bit long. To improve it for competitive settings, changes were made to simplify paths by removing waypoints using the prune method. This improvement, together with fine-tuning cost evaluations, improves efficiency while maintaining algorithm precision.

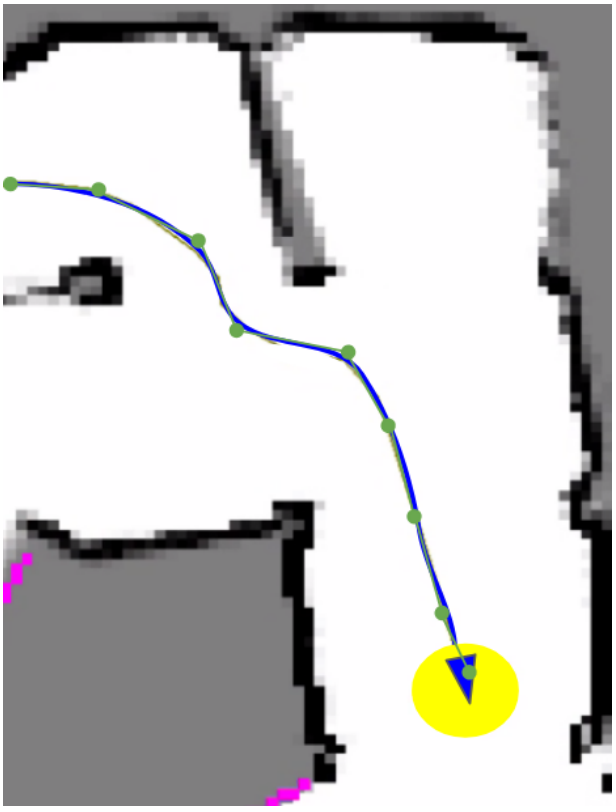

Fig. 10. A star planned path (Green) and the Actual Path of the MBOT (Blue)

TABLE IV
A-STAR TESTS RESULTS

Tests	Min	Max	Mean	Std. Deviation
Maze grid	3853	6503	4876	1083.2
Narrow constriction	131	359721	116663	152685.8
Wide constriction	1302	36001	13228	16276.2
Convex grid	469	2031	1320.5	1086.6

2) Exploration: The exploration algorithm always looks for the nearest frontier to travel to. It demonstrates that our goals are nearing frontiers because it will reduce calculation time and allow for real-time updates. Our approach allows the robot to avoid obstacles by detecting them before moving to the target and calculating the next frontier. When there is no frontier on the map, the robot returns to its initial place and completes the exploring process.

IV. DISCUSSION

A. Motion Controller and Odometry

The calibration results show that the calibration function of our MBots has greater variation in its intercept than in its slope. We observe that because the slope is more dependent on the mechanical design of the motors than it is on the static friction that needs to be overcome across calibrations. There could also be more reasons like calibration's working area may differ, and non-systematic elements like uneven flooring or slippage may also have an impact on the static friction. When the MBot pose's θ value is estimated using the IMU instead of just the encoder readings, the odometry output is improved; the readings correspond to the locations of the waypoints in the basic maze. Furthermore, the dead-reckoning position computed using the enhanced odometry is significantly more representative of the four-square path that the MBot performed than the rudimentary odometry for outcomes that depend on the Motion Controller. The tuning procedure is shown in the three plots of a square-path traversal at various points in the Motor Control tuning. The traversal and odometry outputs from the lack of motor control are reasonably accurate. Nevertheless, it results in slight swerves and a greater displacement of the final position odometry in x, y, and θ. The plot with the most precise odometry at the final position is the one that PID tuning to show a very accurate and smooth traversal.

B. Simultaneous Localization and Mapping (SLAM)

The generated maps were discovered to be less precise than anticipated. This discrepancy can be attributed to several factors, such as sensor noise, insufficient sensor resolution, or deficiencies within the mapping algorithm. Enhancements could be made by refining the mapping

algorithm to effectively handle sensor noise and interpret sensor data with greater precision. The SLAM system exhibited larger-than-anticipated errors in localization and mapping. These errors may stem from compound inaccuracies in both the action and sensor models, along with potential deficiencies in the particle filter algorithm. A possible resolution involves the development of a more robust action model that comprehensively considers robot and environmental factors, as well as enhancing the sensor model to interpret environmental data more accurately.

C. Planning and Exploration

When implementing A-star, we noticed most of the successful cases had around x seconds of evaluation time, though there were cases where the execution time was way quicker. After some fine-tuning of the cost function, the M-bot was able to effectively complete the planning using the A-star algorithm and accurately map the surroundings where the frontiers were discovered. Sometimes the robot moves in a curved trajectory, and when it turns in the opposite direction, it achieves the desired position. We discovered that one wheel rotates first, requiring an adjustment in PID parameters for the wheel, which will resolve the issue. The bot was able to return to the starting point after exploration was done, but it has some positioning errors, primarily because of the issues we faced with the odometry. The waypoints created using the A star were more than adequate to navigate through the maze, thereby avoiding obstacles and walls. The prune_node function was useful to reduce the total number of waypoints. During the exploration process,our method determines the closest free cell to the robot's destination point. To improve efficiency, we can determine the frontier closest to the robot's current position rather than the first valid path to it. Fixing the odometry will drastically change the performance of the M-Bot by precisely positioning the bot at the desired waypoints.

V. Future Works

The main setback we faced was the implementation of the forklift. As our slots for the fasteners in the 3D-printed part were worn out, the assembly was affected because of this. While current execution can complete the planning and SLAM task at hand, we see room for improvement in these algorithms. Our future efforts will focus on improving our odometry, planning, and exploration methods' efficiency by fine-tuning the cost function, and motion controller and increasing their application to a variety of test conditions.

VI. Conclusion

In the MBot Lab, we've instilled autonomy in the given robot across various levels of abstraction, crafting motion, SLAM, and exploration functionalities. Through diverse tests, we successfully showcased these autonomous abilities, enabling the robot to discern its position from an unknown pose and autonomously engage in exploration tasks. For path planning, we used a fast and efficient path planning algorithm, i.e. A* algorithm to move the mobile robot to target poses. In this project, we also designed a forklift with a gear-pulley mechanism to lift objects in a warehouse scenario.

VII. References

1) Delmerico J, Mintchev S, Giusti A, Gromov B, Melo K, Horvat T, Cadena C, Hutter M, Ijspeert A, Floreano D, Gambardella LM. The current state and future outlook of rescue robotics. Journal of Field Robotics. 2019 Oct;36(7):1171-91.
2) Kruijff-Korbayová I, Freda L, Gianni M, Ntouskos V, Hlaváč V, Kubelka V, Zimmermann E, Surmann H, Dulic K, Rottner W, Gissi E. Deployment of ground and aerial robots in earthquake-struck amatrice in italy (brief report). In2016 IEEE international symposium on safety, security, and rescue robotics (SSRR) 2016 Oct 23 (pp. 278-279). IEEE.
3) Yamauchi G, Nagatani K, Hashimoto T, Fujino K. Slip-compensated odometry for tracked vehicle on loose and weak slope. Robomech Journal. 2017 Dec;4:1-1.
4) Endo D, Nagatani K. Assessment of a tracked vehicle's ability to traverse stairs. Robomech Journal. 2016 Dec;3:1-3.
5) Dang T, Tranzatto M, Khattak S, Mascarich F, Alexis K, Hutter M. Graph-based subterranean exploration path planning using aerial and legged robots. Journal of Field Robotics. 2020 Dec;37(8):1363-88.