Team 2 PM ROB 550 BotLab Report

Zariq George*, Aaron Tran*, Yang Zhang+

***Abstract*—This report details the completion of the BotLab portion of ROB 550 in Winter 2023 by group two for the PM section. In this project, a differential-drive robot with 2D scanning lidar and a MEMS 3-axis IMU is programmed to navigate and explore closed environments. The software developed here includes a cascade PID robot frame controller, particle filter for localization, a occupancy grid map for mapping, and an A* implementation for path planning.**

I. Introduction

MBOT is a robot that moves with a differential drive, two parallel wheels with a rear caster. Each motor is equipped with a magnetic wheel encoder. The robot also has a scanning 2D Lidar, a Raspberry Pi camera, and a MEMS 3-axis IMU. The MBot has a Wi-Fi module, which we use to SSH into and program the onboard Raspberry Pi 4B (RPi) computer module. The low-level motor control is handled on a Raspberry Pico.

Robots like the Mbot have many applications such as Autonomous navigation, Autonomous exploration, smart homes, autonomous driving, military, etc. To achieve these functions, the Mbot mainly consists of motor control, path plan, mapping, and localization.

Precise motor control is the foundation for the Mbot which drives the Mbot and enables the Mbot to accurately move as intended. Path planning enables the Mbot to find a way to get to the desired pose and avoid obstacles. Without it, Mbot can not make use of environmental information gained from mapping. Mapping uses a LIDAR sensor to scan the environment and store information. The localization part is essential for the Mbot to know its location and help create an accurate map. After combining these parts together, the Mbot is able to simultaneously localize and map an unknown environment.

II. Methodology

A. Motion and Odometry

1) Odometry: Our odometry makes use of encoders and the IMU. The encoders are capable of measuring wheel displacement, which we translate into a forward and heading displacement for the robot. For the IMU, we only use them on board gyroscope, which we also use to determine heading. Equation 1 describes how we take the right and left encoder readings been converted to linear wheel travel, Δs_R and Δs_L, and obtain the change in heading, $\Delta\theta$, the linear displacement, Δd, the change in x position, Δx, and the change in y position, Δy.

$$\begin{aligned} \Delta\theta &= \frac{\Delta s_R - \Delta s_L}{b} \\ \Delta d &= \frac{\Delta s_R + \Delta s_L}{2} \\ \Delta x &= \Delta d * cos(\theta + \Delta\theta/2) \\ \Delta y &= \Delta d * sin(\theta + \Delta\theta/2) \end{aligned} \tag{1}$$

Encoders are affordable and easy to use, but they inherently suffer from a number of systematic errors that can induce large uncertainty. Backlash due to gearboxes, any amount of wheel slip, inaccuracies in wheel diameter measurements, differences in wheel diameters, wheelbase measurement uncertainty, encoder resolution – all of these contribute to uncertainty in the encoder readings, and more importantly, the odometry readings. With careful attention, the uncertainty present in forward velocity and displacement measurements can be kept acceptably low, but turn velocity and displacement is still difficult with just encoders. For this reason, we defer to the gyroscope when we can.

The gyroscope works well most of the time but is subject to drift, as it integrates the change in heading over time to output readings. This unfortunately includes any bias the sensor may have. Still, the gyroscope readings tend to be more accurate and we have found that the robot performs much better using Algorithm 1. We have found that it is better to use the gyroscope almost always. The only time we do not want to use the gyroscope is when the robot is sitting still or close to still. This is when the drift in the gyroscope is large in comparison to the readings themselves. This lead us to implement Algorithm 1 to determine our detected change in heading.

We validated the use of this algorithm and quality of our odometry by comparing odometry readings to expected displacement values (measured by hand). This

*Zariq George and Aaron Tran are with the Department of Robotics, University of Michigan, Ann Arbor, MI 48109, USA. E-mail: {georgz, aartran}@umich.edu.

+Yang Zhang is with the Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109, USA. E-mail: zhanya@umich.edu.

Algorithm 1 Gyrodometry

$\Delta\theta_{gyro} \leftarrow IMU$
$\Delta\theta_{enc} \leftarrow Encoder$
$\Delta\theta_{GO} \leftarrow |\Delta\theta_{gyro} - \Delta\theta_{enc}|$
if $\Delta\theta_{GO} > \epsilon$ **then**
    $\Delta\theta_{odom} \leftarrow \Delta\theta_{gyro}$
else
    $\Delta\theta_{odom} \leftarrow \Delta\theta_{enc}$

was a qualitative process and we only tuned the gyrodometry threshold when we noticed it was affecting our controller. We initially set it to an arbitrarily low number and have changed it to be $\Delta\theta_{GO} = 0.08$. This led to qualitatively satisfactory results in driving in a square and returning to the same starting position. The repeatability of this is shown in Figure 9.

2) Control: Our robot frame controller is a feed forward cascade PID controller, whose formula is shown in the Eq. 2.

$$Output = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \tag{2}$$

The whole structure of our Controller is shown in Figure1. From Figure1, we can know that. A motion controller takes in desired x, y, and $theta$ values and converts them into set-point forward and turn velocities. These are both fed forward through the open loop model to determine base PWM values and through a low pass filter to be used as inputs for the first PID controller. The first PID controller filters the set-point forward and turn velocities which are then fed into the second PID controller. For the second PID controller, we pass the filtered forward and turn velocity set-points and compare them with filtered measured forward and turn velocity set-points. This outputs right and left wheel speed PWM commands to add on top of our feed forward model. The final PWM commands are then sent to the motors via the pico.

To facilitate the design of the motion controller, the pose of the robot was represented in polar coordinates as shown in Fig. 2

The motion controller utilized the kinematic motion model shown in Eq. 3 and control law in Eq. 4 to drive the robot to its goal pose. The gains (k_d, k_α, k_β) influence the linear velocity (v) and angular velocity (ω) of the robot.

$$\begin{bmatrix} \dot{d} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -K_d d\cos\alpha \\ K_d \sin\alpha - K_\alpha \alpha - K_\beta \beta \\ -K_d \sin\alpha \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} K_d d \\ K_\alpha \alpha + K_\beta \beta \end{bmatrix} \tag{4}$$

To ensure stability of the motion controller, k_d, k_α, k_β were chosen to satisfy the strong stability condition in Eq. 5 [2].

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ -\frac{2}{\pi} & 1 & \frac{5}{3} \end{bmatrix} \begin{bmatrix} k_d \\ k_\alpha \\ k_\beta \end{bmatrix} > \mathbf{0} \tag{5}$$

B. Simultaneous Localization and Mapping (SLAM)

1) Mapping: For the mapping part, we use Breshenham's algorithm to label which cells are within the lidar ray's range and then use log odds to score the ray. If the laser passes the cell, the score will be small. If the laser terminates at the cell, the score will be high, which will help create a map. After the map is generated, a threshold value of 0 log odds is used to determine whether a cell on the map is an obstacle. Values larger than 0 will indicate that the cell is an obstacle. Lower values will indicate free cells.

a) Breshenmam Algorithm: Let x_i, y_i be the center-of-mass of $\mathbf{m_i}$

$$\begin{aligned} &r = \sqrt{(x_i - x)^2 + (y_i - y)^2} \\ &\phi = \operatorname{atan2}(y_i - y, x_i - x) - \theta \\ &k = \operatorname{argmin}_j |\phi - \theta_{j,\text{ sens}}| \\ &\text{if } r > \min\left(z_{\max}, z_t^k + \alpha/2\right) \\ &\quad \text{return } l_0 \\ &\text{if } z_t^k < z_{\max} \text{ and } \left|r - z_t^k\right| < \alpha/2 \\ &\quad \text{return } l_{\text{occ}} \\ &\text{if } r \le z_t^k \\ &\quad \text{return } l_{\text{free}} \\ &\text{endif} \end{aligned} \tag{6}$$

where r is distance to the cell, ϕ is angle to cell, α is dimension of cell, k is beam index, z_t^k is reading k from, scan at time t. For the project, we use $l_{occ} = 3.5, l_{free} = -1$.

2) Monte Carlo Localization:

a) Action Model: For the action model, we propagate the particles through the dynamics of the system with some noise. The noise is based on our understanding of the robot's uncertainty in turning and moving forward. The noise value was tuned until we determined that the resultant distribution matched what we observed well. The parameters are defined in Figure 3.

Based on the definition, the equations are as follows. For k_1, we choose 0.0005, for k_2, we choose 0.001.

$$\begin{aligned} \varepsilon_1 &\sim \mathcal{N}(0, k_1|\alpha|) \\ \varepsilon_2 &\sim \mathcal{N}(0, k_2|\Delta s|) \\ \varepsilon_3 &\sim \mathcal{N}(0, k_1|\Delta\theta - \alpha|) \end{aligned} \tag{7}$$

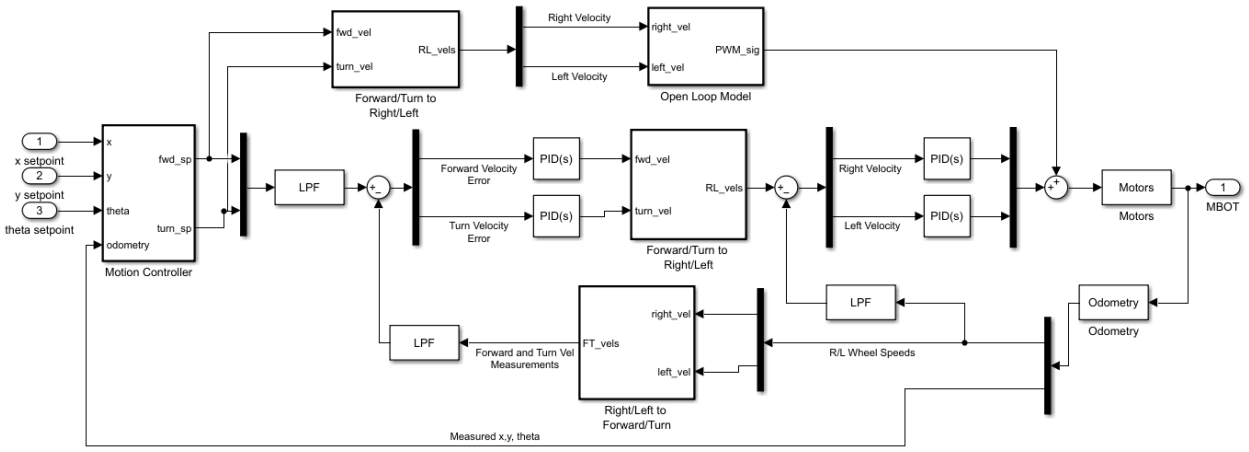
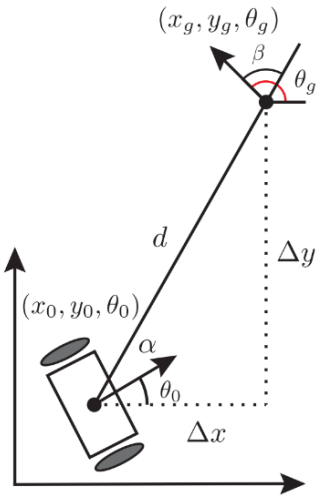


Fig. 1: Structure of Controller



$$\Delta x = x_g - x$$
$$\Delta y = y_g - y$$
$$d = \sqrt{\Delta x^2 + \Delta y^2}$$
$$\alpha = \text{atan2}(\Delta y, \Delta x) - \theta$$
$$\beta = \theta_g - \theta$$

Fig. 2: Polar Coordinate Representation of Robot Pose [1]

Therefore, the next pose estimated from odometry is as follows.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta s + \varepsilon_2)\cos(\theta_{t-1} + \alpha + \varepsilon_1) \\ \Delta s + \varepsilon_2)\sin(\theta_{t-1} + \alpha + \varepsilon_1) \\ \Delta\theta + \varepsilon_1 + \varepsilon_3 \end{bmatrix} \quad (8)$$

b) Sensor Model: The sensor model describes the information that robot is able to provide with it's sensors.

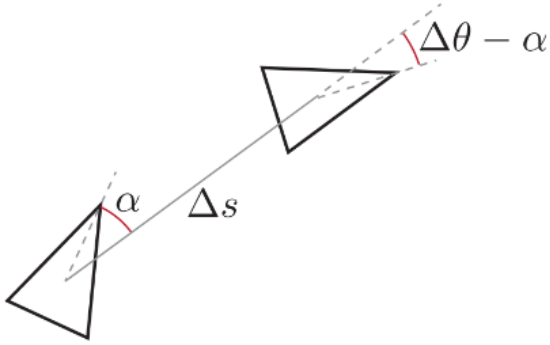


Fig. 3: Action

In this study, the sensor model provides the probability of a LIDAR reading (z_t) matching the pose of the robot, $x_t = (x\ y\ \theta)^T$, given the map of the environment (m) as shown in Fig. 4.

The simplified likelihood field model was utilized to determine the likelihood of $p(z|x_t, m)$ as shown in Alg. 2. It calculates $p(z|x_t, m)$ using the endpoints of the rays $(x_{z_t^k}\ \ y_{z_t^k})^T$ from Eq. 9 and the log odds of the occupancy grid, $\log o(m|z_t, x_t)$. The relative location of the robot's LIDAR in it's local coordinate is denoted as $(x_{k,sens}\ \ y_{k,sens})^T$, and the angular orientation of the sensor beam relative to the robot's heading direction as $\theta_{k,sens}$.

$$\begin{bmatrix} x_{z_t^k} \\ y_{z_t^k} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{k,sens} \\ y_{k,sens} \end{bmatrix} + z_t^k \begin{bmatrix} \cos(\theta + \theta_{k,sens}) \\ \sin(\theta + \theta_{k,sens}) \end{bmatrix} \quad (9)$$

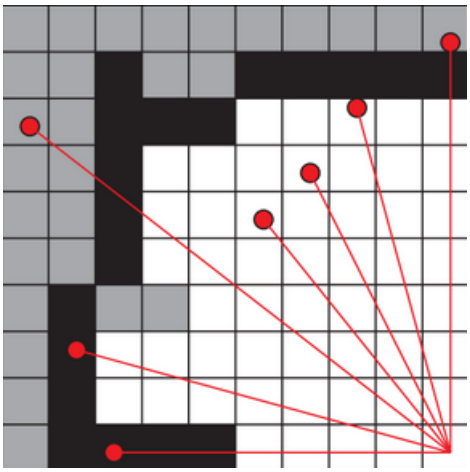


Fig. 4: Laser Scan on Map

Algorithm 2 Simplified Likelihood Field Model

Require: d_c ▷ The length of a cell in the map
Require: ζ ▷ Fraction of cell near end of ray
$q \leftarrow 0$
$H(x, y) : \log o(m(x, y))$ ▷ Log odds at x, y on map
for all k **do**
  if $H(x_{z_t^k}, y_{z_t^k}) > 0$ **then**
    $q = q + H(x_{z_t^k}, y_{z_t^k})$
  else if $H(x_{z_t^k} + d_c, y_{z_t^k} + d_c) > 0$ **then**
    $q = q + \zeta H(x_{z_t^k} + d_c, y_{z_t^k} + d_c)$
  else if $H(x_{z_t^k} - d_c, y_{z_t^k} - d_c) > 0$ **then**
    $q = q + \zeta H(x_{z_t^k} - d_c, y_{z_t^k} - d_c)$
  else
    $q = q$
return q

c) Particle Filter: The particle filter is a multi-modal localization filter capable of representing different underlying distributions. The fact that it is multi-modal is important to the kidnapped robot problem. By placing a hypothesis over multiple different positions, the robot can recover from being displaced by some external force to an entirely new position. Additionally, we expect the lidar readings to follow a composite distribution of Gaussian, exponential, and uniform. Imposing or assuming any one distribution shape, as a Kalman Filter does, would be detrimental to the performance of our system. The particle filter starts with an initial amount of randomly distributed particles spread across either the entire map or around a prior pose. These particles are propagated forward through the action model after which we evaluate the likelihood of seeing the sensor readings given the new particle positions. The particles are assigned normalized weights proportional to these likelihoods and then resampled using the particle weights.

3) Combined Implementation: The combined implementation is shown in figure 5.

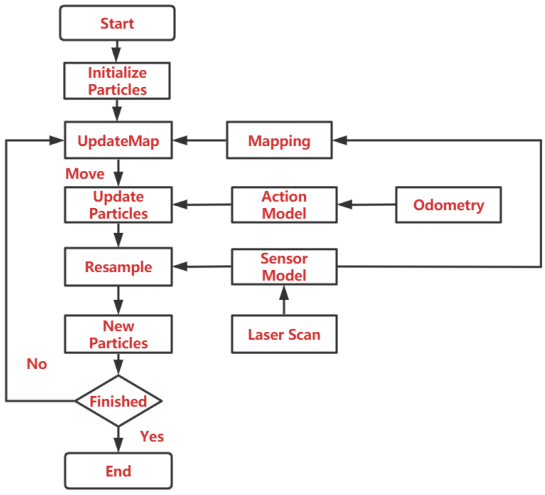


Fig. 5: Diagram of Slam Algorithm Using Particle Filter

C. Planning and Exploration

1) Path Planning: We utilize the A* algorithm (Algorithm 3) to efficiently find the optimal path to the goal position in a 2D grid. A* combines the strengths of two different algorithms: Dijkstra's Algorithm and the Greedy Best-First search. Dijkstra's algorithm is a Breadth First Search algorithm that can find the optimal path using a g-cost, which encapsulates the cost to move from the start position to the current node position. However, it is computationally expensive as it explores in all directions, making no use of our prior knowledge on the goal's position. Greedy Best-First search is a Depth First Search algorithm that finds a path using a heuristic. It simply measures the distance from child nodes to the goal position to determine the next child to expand into. This method works well for open spaces but struggles with obstacles. A* makes use of both the g-cost and the heuristic cost (h-cost). This leads to a tunable, efficient algorithm that can guarantee an optimal path.

2) Exploration:

a) Frontier: The frontiers are obtained by BFS. It expands from the frontier and the specific algorithm is as Table 4.

b) Exploration Strategy: The steps to explore the map are defined below.

1) Find frontiers
2) Computes the centroid of each frontier
3) Select the nearest centroid as the next target
4) Use BFS to compute the nearest navigable cell
5) Plan the path from the current pose to the desired pose using A* algorithm

Algorithm 3 A*

if $goal(start) = true$ **then**
  return $makePath(start)$
$open \leftarrow start$
$closed \leftarrow \emptyset$
while $open \neq open$ **do**
  $sort(open)$
  $n \leftarrow open.pop()$
  $kids \leftarrow expand(n)$
  for all $kid \in kids$ **do**
    $kid.f \leftarrow n.g + g(kid) + h(kid)$
    if $goal(kid) = true$ **then**
      return makePath(kid)
    if $kid \cap closed = \emptyset$ **then**
      $open \leftarrow kid$
  $closed \leftarrow kid$
return $\emptyset$

Algorithm 4 Frontier Expansion Algorithm

$Initialize \quad Queue, VisitedCell$
$StartPose \rightarrow VisitedCell$
while $VisitedCell \neq \varnothing$ **do**
  $Neighbors = 4 - Connected$
  for $q \in Neighbors$ **do**
    if $q \in visited$ **then** continue
    else if $q \quad is \quad frontier$ **then**
      $growfrontier$
    else if $q \quad is \quad empty \quad cell$ **then**
      $q \rightarrow VisitedCell$
      $q \rightarrow Queue$

III. Results

A. Control

1) Calibration: Table I, Table II, and Figure 6 detail the calibration constants we obtained for determining the relationship between motor PWM and wheel speeds.

2) Wheel Speed Open Loop and PID Controller: The wheel Speed Controller is a PID controller with input filtered by a low-pass filter. The parameters are shown in Table III.

BOT	$+Slope$	$+Intercept$	$-Slope$	$-Intercept$
108	0.0058	0.0840	0.0057	-0.0989
008	0.0059	0.0773	0.0059	-0.0781
107	0.0064	0.0993	0.0062	-0.1055

TABLE I: Left Motor Calibration Values

BOT	$+Slope$	$+Intercept$	$-Slope$	$-Intercept$
108	0.0055	0.0923	0.0061	-0.0969
008	0.00616	0.106	0.00597	-0.136
107	0.0064	0.0868	0.0064	-0.0854

TABLE II: Right Motor Calibration Values

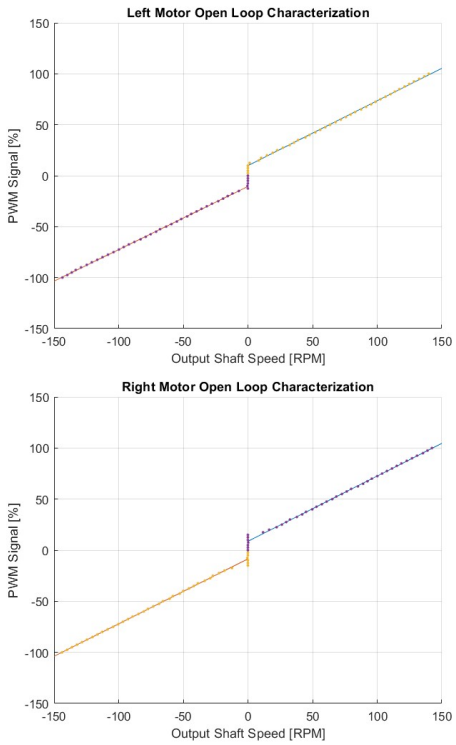


Fig. 6: Open loop characterization plots for MBOT 107

K_p	K_d	k_i	$dFilterHz$
3	0	0.01	25

TABLE III: Wheel Speed Controllers Parameters

3) Robot Frame Velocity Controller: The wheel Speed Controller is a PID controller with input filtered by a low-pass filter. The parameters are shown in Table IV.

K_p	K_d	k_i	$dFilterHz$
3	0	0.01	25

TABLE IV: Frame Velocity Controllers Parameters

To test the controller, we run the mbot with different velocities and the result as shown in Fig. 7 and Fig. 8.

4) Motion Controller: From Figure 1, we know the structure of the whole Motion Controller. The parameters are shown in Table V.

Controller	k_d	k_α	k_β	$dFilterHz$
Smart	3.0	50	-10	10

TABLE V: Gains of Smart Controller

Fig. 9 shows the robot's dead reckoning estimated pose as the robot is commanded to drive a 1m square 4 times.

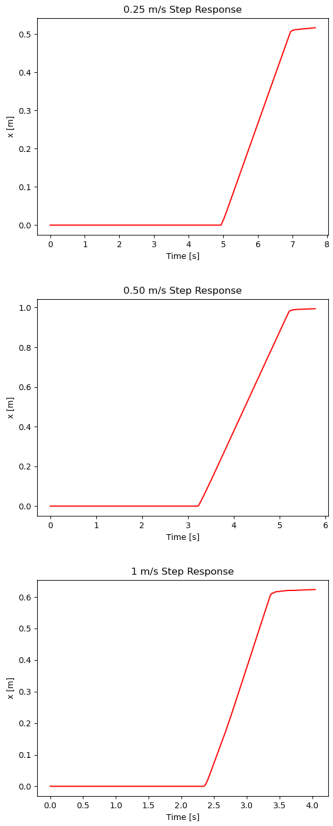


Fig. 7: X position vs time

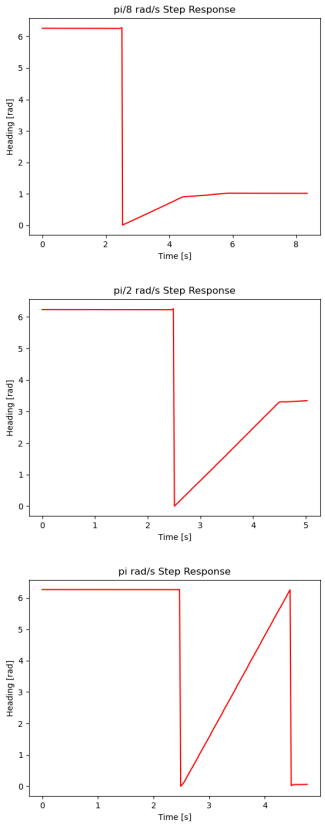


Fig. 8: Heading vs time

Fig. 10 shows the robot's linear and rotational velocity as it drives one loop around the square.

The control algorithm for getting between way-points is described in Fig. 2 and Eq. 3,4,5.

B. Simultaneous Localization and Mapping (SLAM)

1) Mapping: Fig. 12, shows the plot of a map generated using the robot's occupancy grid mapping algorithm with prerecorded data to validate it's accuracy.

2) Monte Carlo Localization:

a) Action Model: From Eq. 7, we know that there are four parameters that reflect the uncertainty which shows in Table VI.

k_1	k_2	min_dist	min_theta
0.005	0.001	0.00025	0.002

TABLE VI: Uncertainty Parameters in Action Model

Values of $\Delta\theta$ and Δdis were chosen to decide the magnitude of k_1 and k_2. Their values were changed incrementally based on their performance until changes no longer yielded a substantial gain in performance. A similar process was used to determine min_dis and min_theta.

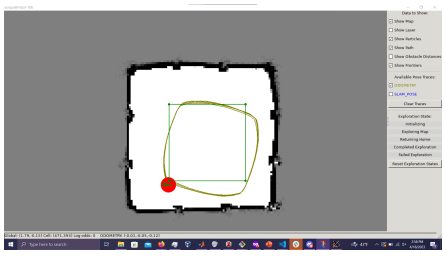


Fig. 9: Estimated pose as the robot drives a 1m square 4 times

b) Particle Filter: To determine the capabilities of the particle filter, the time to update the sampled poses was measured for an increasing number of particles as shown in Table VII. The maximum number of particles the filter could handle before performance significantly deteriorated was approximately 5000 particles.

3) Combined Implementation: Fig. 12 illustrates the difference between the robot's calculated SLAM pose and the ground truth pose from the pre-recorded data.

Fig. 14 illustrates the distance between the SLAM pose and the true pose as a function of time resulting in a RMSE between the two poses of 0.17m.

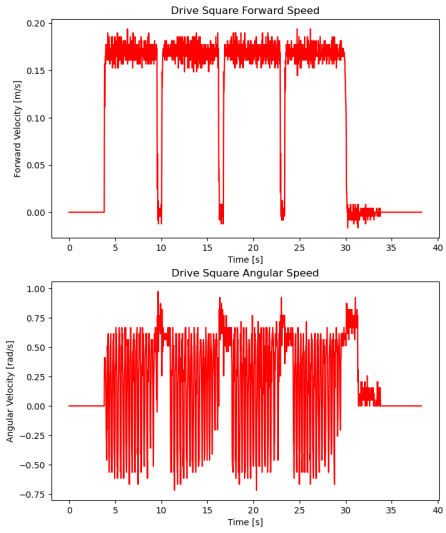


Fig. 10: Linear and rotational velocity while driving in a square

Particle Count	Update Time (ms)
100	5.140
300	20.697
500	32.790
1000	70.655
5000	268.474
10000	570.000

TABLE VII: Particle Count vs. Update Time

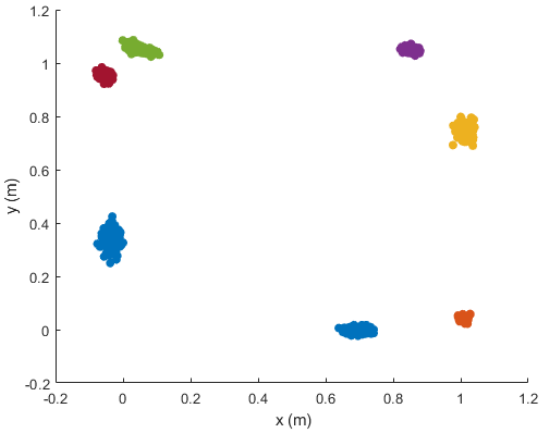


Fig. 11: Particle Distribution in Square Path

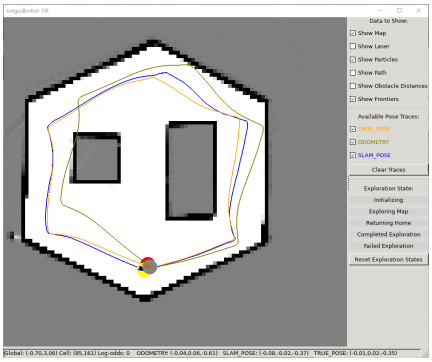


Fig. 12: Estimated Pose From Slam System Compared With True Pose

C. Planning and Exploration

1) Path Planning: Table VIII and IX show the timing statistics in μs for various grid types. The former shows the data for successful path generation, and the latter shows the data for failed path generation. The maze test is of greatest interest to us, as it is the most complicated grid the robot was designed for. A mean of 43.415 ms is acceptable for our performance needs.

Grid	Min	Mean	Max	Median	Stddev
Convex	145	2016.5	38888	0	1871.5
Maze	7106	14950	23078	7236	7783.05
Narrow	3019	43415	122655	122655	56034.7
Wide	3009	43446	124199	124199	57101

TABLE VIII: Timing statistics on successful A* tests

To test the path planning, we plot a figure showing the planned path in an environment. The result is shown in Fig. 15

IV. Discussion

For the competition, our team ranked 1st and perform well in the first two events. From Fig. 7, Fig. 8, and Fig.9 we can see that the motion controller works well. However, we found that at high speeds, our robot fails to make maps of good quality. This is evident in Fig. 15.

Due to the time limitations, we were not able to fully validate our path planning. This means that operating the Mbot requires waypoints or teleoperation. We are satisfied with the timing and performance of A* in the test script, but it fails to operate properly when using click to move. Ensuring that A* works properly would be vital for a fully functional robot.

One challenge that we did not undertake was the kidnapped robot problem. It involves picking up the

Grid	Min	Mean	Max	Median	Stddev
Convex	43	55	67	0	12
Empty	38	53.8	90	43	19.083
Maze	39	46.8	68	41	10.7778
Narrow	44	60	76	0	16
Wide	42	42	42	0	0

TABLE IX: Timing statistics on failed A* tests

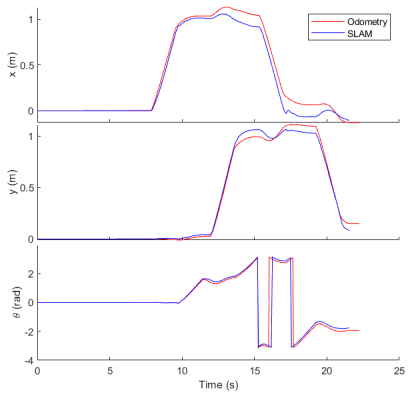


Fig. 13: SLAM and odometry difference

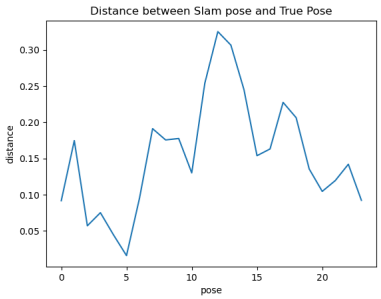


Fig. 14: Distance between Slam Pose And True Pose

robot and placing it in a completely unknown position. This can be detected using the IMU. The robot loses its understanding of its position, which makes it difficult to localize. Assuming that the robot already has a map, this can be solved by playing a uniform distribution over the entire map. High probability points will become clear due to resampling and the sensor model. However, the robot must move in order to further refine this distribution, hopefully making a much more distinct mode in the distribution. To do this, we would look at lidar readings and move in directions where the lidar range readings, on average, are the largest. The robot would move only small amounts and wait to receive lidar readings then move a small amount again until the variance in the position of the top 10% of particles is acceptably low. This should ensure that the robot can relocalize after getting lost.

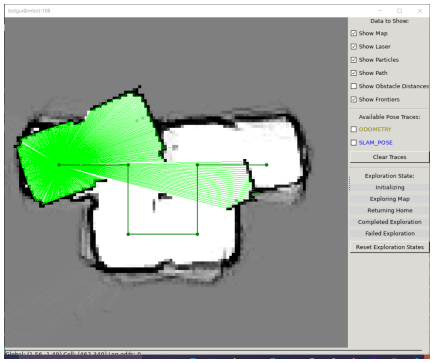


Fig. 15: Path Planning

These improvements would help us meet the goals that were set at the beginning of the project. In the interest of improving the robot past the expectations, we would be interested in implementing a computer vision system capable of semantically classifying objects in view. We could pair this with the lidar or the IMU to implement sophisticated SLAM frameworks, such as V-LOAM[3] or Fast-LiO2[4].

REFERENCES

[1] P. Gaskell, "Rob550 winter 23 lecture series," 2023.
[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.
[3] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: low-drift, robust, and fast," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2174–2181.
[4] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022.
[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2006. [Online]. Available: http://www.probabilistic-robotics.org/
[6] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.com/books?id=wGapQAAACAAJ