Section 2 Team 1 ROB 550 BotLab Report

Yulun Zhuang, Hao Liu, Morgan Sun

***Abstract*—Autonomous mobile robots are useful in a variety of environments. In the MBot lab, movement control, obstacle detection, maze exploration, and self-localization functionality will be developed on a mobile robot platform.**

I. Introduction

AUTONOMOUS mobile robots are reshaping the way people live and work. In complex environments, it may be impossible to ensure continuous communication with the robot, thus requiring autonomous capability. The MBot Lab is designed to explore the fundamentals of robot autonomy by developing a robot with autonomous mapping, localization, and exploration capabilities. Four increasingly complex competition tasks provided validation of these functions. In this report, the systems used to implement these functions, as well as resulting performance characteristics, will be discussed.

Fig. 1: Snapshot of the MBot robot

II. Methodology

A. Odometry

The MBot is a unicycle model robot shown in Figure 1 and therefore its state can be represented as a position (x, y) and an angle θ. Odometry is used to enable awareness of the MBot's own state/pose. To this end, the MBot is equipped with motor encoders on the left and right drive motors, as well as an IMU and LIDAR. The encoders and IMU will be used to implement odometry functionality.

1) Wheel Speed Calculation: An unusual method of performing this calculation was designed in order to ensure smooth behavior at low speed. This method is described in detail by Algorithm 1. This calculation method was validated by manually setting the wheels to various speeds and comparing the calculated velocity to expectations.

Algorithm 1 Wheel Speed Calculation

Input: $\Delta n_{enc},\ \Delta T$
Output: v_{wheel}
  $Q \leftarrow PushFront(\Delta n_{enc})$
  if $Q.length > 10$ **then**
    $PopBack(Q)$
  $n_{cuml}, T_{cuml} \leftarrow 0, 0$
  for $\Delta n_i \in Q$ **do**
    $n_{cuml} \leftarrow n_{cuml} + \Delta n_i$
    $T_{cuml} \leftarrow T_{cuml} + \Delta T$
    if $n_{cuml} > 15$ **then**
      break
  $v_{wheel} \leftarrow EncoderToMeters(\frac{n_{cuml}}{T_{cuml}})$

2) Gyrodometry: To avoid issues with drift, the IMU was not used to characterize linear motion or velocity; instead, encoder-based odometry was used. However, during turns, slight wheel slippage means that encoder-based odometry is not the optimal approach. Instead, Algorithm 2 is used to calculate rotation rate using both IMU and encoder measurements[1].

Algorithm 2 Gyrodometry

Input: $\Delta\theta_{gyro},\ \Delta\theta_{odom}$
Output: $\Delta\theta$
  $\Delta_{G-O} = \Delta\theta_{gyro} - \Delta\theta_{odom}$
  if $|\Delta_{G-O}| > \Delta\theta_{thres}$ **then**
    $\Delta\theta \leftarrow \Delta\theta_{gyro}$
  else
    $\Delta\theta \leftarrow \Delta\theta_{odom}$

Gyrodometry effectively uses encoder-based odometry most of the time, while incorporating IMU data only when gyro and odometry data differ substantially, mitigating the effects of IMU drift. The tunable parameter $\Delta\theta_{thres}$ represents the degree of confidence in IMU- as opposed to encoder-based odometry.

3) Dead Reckoning: Dead reckoning was used as a minimum viable localization technique. This method works by integrating measured linear and rotational velocities. More specifically, the process described by Algorithm 3 is used.

Algorithm 3 Odometry

Input: $\hat{\mathbf{x}}_{t-1},\ \Delta n_r,\ \Delta n_l,\ \Delta\theta_{gyro}$
Output: $\hat{\mathbf{x}}_t$
$\Delta R \leftarrow K_{enc} \times \Delta n_r$
$\Delta L \leftarrow K_{enc} \times \Delta n_l$
$\Delta d \leftarrow (\Delta R + \Delta L)/2$
$\Delta\theta_{odom} \leftarrow (\Delta R - \Delta L)/b_{wheel}$
$\hat{x}_t \leftarrow \hat{x}_{t-1} + \Delta d \cos(\theta + \Delta\theta_{odom}/2)$
$\hat{y}_t \leftarrow \hat{y}_{t-1} + \Delta d \sin(\theta + \Delta\theta_{odom}/2)$
$\hat{\theta}_t \leftarrow \hat{\theta}_{t-1} + Gyrodometry(\Delta\theta_{gyro},\ \Delta\theta_{odom})$

$\hat{\mathbf{x}}$ is the robot state estimated by odometry, and K_{enc} is the encoder constant calculated by $2\pi r_{wheel}/(gear\ ratio \times encoder\ resolution)$.

Parameters are wheel radius r_{wheel} and the wheelbase distance b_{wheel}, which are given by physical measurement/specification; they were fine-tuned by driving the robot in a square, with the objective of matching odometry output to the actual robot path. Dead reckoning was validated by driving the robot in a square and observing the odometry-based trajectory estimate.

B. Motion Control

Motion control can be separated into three hierarchical levels: wheel controllers, a velocity controller, and a motion controller.

1) Wheel Controller: The wheel controllers are responsible for maintaining the speed of individual wheels in response to set-point inputs by controlling the duty cycle of PWM signals sent to wheel motor drivers. Two wheel controller designs were compared: simple open-loop control, as well as a PID controller.

The open loop controller works by linearly mapping the wheel speed set-point to an output duty cycle. To account for hardware variation, independent linear mappings were used for each wheel and direction for a total of four mappings. Each mapping was determined using a calibration procedure wherein a wheel's duty cycle was swept from 0% to 100% while encoders were observed to track the actual wheel speed; linear regression was then used to determine the slope and intercept associated with each linear mapping.

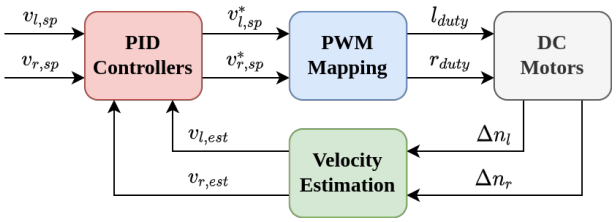


Fig. 2: Wheel speed feedback controller

The closed loop controller (shown in Figure 2) differs from the open loop controller by adding an additional term to the input set-points. This term is determined using a PID kernel which takes as input the error between the true and set-point wheel speeds. Additional controller features, such as an output low-pass filter, were experimented with, but did not yield noticeable improvements and were therefore not implemented.

Due to the highly nonlinear behavior of the hardware components of the robot system, it was not feasible to use an established systematic PID tuning method such as Nicholas-Ziegler. Instead, PID parameters were tuned through trial-and-error, with the following objectives:

- When given identical left and right set-points, the robot moves with minimal curvature
- After a step change in the input set-point, the robot does not oscillate noticeably
- After a step change in the input set-point, the robot reaches a steady state in $<<1$ sec oscillation.

2) Velocity Controller: The velocity controller can be considered to be the next level of abstraction on top of the wheel controllers. The velocity controller receives linear and rotational velocity set-points, and determines what left and right wheel velocity set-points should be given to the wheel controllers based on the following equations:

$$\begin{aligned} v_l &= v_{fwd} - \frac{r_{wheel}}{2} \times v_{rot} \\ v_r &= v_{fwd} + \frac{r_{wheel}}{2} \times v_{rot} \end{aligned} \tag{1}$$

Controller performance was characterized via testing with linear and rotational step inputs.

3) Motion Controller: The motion controller further abstracts robot control by taking an input path of waypoints, and generating velocity set-points that result in the robot following the path. To avoid stopping at every turne, intermediate waypoints are assigned a wider tolerance than the final one.

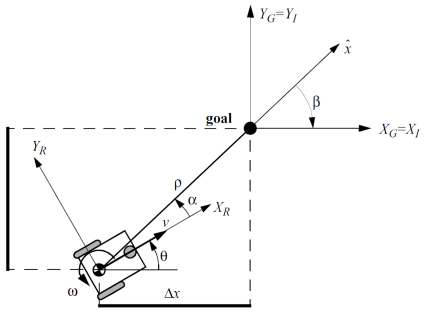


Fig. 3: Feedback controller for nonholonomic model[2]

$$\begin{aligned}\rho &= \sqrt{\Delta x^2 + \Delta y^2} \\ \alpha &= -\theta \arctan 2(\Delta y, \Delta x) \\ \beta &= -\theta - \alpha\end{aligned} \tag{2}$$

From the kinematics of nonholonomic drive model (Equation (2)), the control law of the motion controller is given by Equation (3).

$$\begin{aligned}v_{fwd} &= K_\rho \rho \\ v_{rot} &= K_\alpha \alpha + K_\beta \beta\end{aligned} \tag{3}$$

The motion controller uses the nonholonomic model, and computes the linear and angular velocities such that the deviations in x and y direction are driven to zero. This controller was slightly modified by adding a maximum acceleration constraint in order to prevent undesirable behaviors (e.g. wheel slip and tip-overs) after large step changes. The modified controller was validated by using it to drive laps around a one-meter square.

C. Simultaneous Localization and Mapping (SLAM)

Occupancy-grid based mapping was combined with Monte Carlo localization to create a SLAM system.

1) Mapping: Mapping was performed by maintaining an 2D occupancy grid, with each cell containing the estimated log-probability of an obstacle existing at the corresponding location. The mapping system receives LIDAR and robot pose data as input. Received LIDAR scans are first converted to global frame; this conversion takes into account the motion of the robot during the scan by interpolating from the previous to the current robot pose estimate. The start- and end-points of each ray in the scan are then mapped to their corresponding grid cells. The log-probability in the endpoint grid cell is reduced by k_{hit}, while the log-probabilities in cells between the start- and end-points, as calculated by the Bresenham line algorithm[3], are increased by k_{miss}. The mapping system was tested by running it on previously recorded LCM log data that was provided by the instructors.

2) Monte Carlo Localization[4]:

a) Action Model: A turn-travel-turn model was used to characterize the behavior of the robot in response to velocity control, and δ_{rot1}, δ_{trans}, δ_{rot2} correspond to these motion respectively. This model is described in more detail by Algorithm 4

The error parameters in the model k_1 and k_2 correspond to the error distribution of robot rotation and translation and were tuned via trial-and-error while testing the SLAM system. The objective of tuning was to ensure that the pose distribution represented by the particle population consistently included the true pose of the robot throughout SLAM operation.

Algorithm 4 Odometry Action Model

Input: $\mathbf{x}_{t-1},\ \hat{\mathbf{x}}_t,\ \hat{\mathbf{x}}_{t-1}$
Output: $\mathbf{x}_t$

$\Delta x \leftarrow \hat{x}_t - \hat{x}_{t-1}$
$\Delta y \leftarrow \hat{y}_t - \hat{y}_{t-1}$
$\Delta\theta \leftarrow \hat{\theta}_t - \hat{\theta}_{t-1}$ ▷ wrap to π for all angles
$\delta_{rot1} \leftarrow \arctan 2(\Delta y,\ \Delta x) - \hat{\theta}_{t-1}$
$\delta_{trans} \leftarrow \sqrt{\Delta x^2 + \Delta y^2}$
$\delta_{rot2} \leftarrow \Delta\theta - \delta_{rot1}$
$\hat{\delta}_{rot1} \leftarrow \textbf{gaussian}(\delta_{rot1},\ k_1\delta_{rot1})$
$\hat{\delta}_{trans} \leftarrow \textbf{gaussian}(\delta_{trans},\ k_2\delta_{trans})$
$\hat{\delta}_{rot2} \leftarrow \textbf{gaussian}(\delta_{rot2},\ k_1\delta_{rot2})$
$x_t \leftarrow x_{t-1} + \hat{\delta}_{trans}\cos(\theta + \hat{\delta}_{rot1})$
$y_t \leftarrow y_{t-1} + \hat{\delta}_{trans}\sin(\theta + \hat{\delta}_{rot1})$
$\theta_t \leftarrow \theta_{t-1} + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

b) Sensor Model: A sensor model was necessary in order to quantify the likelihood of each particle in the particle filter. In the case of Botlab, this was done by quantifying how consistent incoming LIDAR scan data is with the particle poses associated with a given particle. Two methods of doing this were compared. The first method consisted of a raycasting algorithm, while the second method is a simplified model introduced in class.

c) Particle Filter: Each particle is assigned a likelihood using the sensor model. The particle filter uses these likelihoods as weights in a low-variance re-sampling algorithm to propagate particles between consecutive timesteps.

3) Combined Implementation: The combined SLAM implementation consists of the action model, the sensor model, and the particle filter; these components are arranged as shown in Figure 4. In addition to maintaining a map of the robot's environment, SLAM is also responsible for outputting a single estimate of the robot's pose at each timestep. To do this in an outlier-robust way, the most likely 10% of particles are selected, and their likelihood-weighted mean pose is used as the SLAM pose estimate.

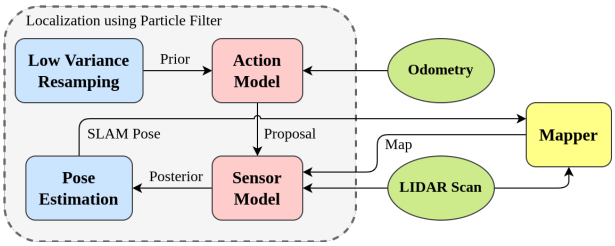


Fig. 4: The SLAM implementation block diagram

The SLAM implementation was validated by running it on several pre-recorded LCM log files, as well as on real-life test environments.

D. Planning and Exploration

Planning and exploration components are the highest level of abstraction, making use of SLAM and motion control systems. The map maintained by SLAM is used to identify unexplored areas within the robot's environment, and the motion controller is used to execute paths that explore these areas.

1) Path Planning: The occupancy map maintained by SLAM serves as the basis for path planning. Observed unoccupied cells which are sufficiently far from observed occupied cells are considered traversable, while all other cells are considered untraversable. Path planning is implemented using the A* algorithm;. to constrain the branching factor of the A* search, only cells sharing an edge or corner were considered to be connected.

The heuristic metric used is a modified version of the Manhattan distance and is described by the equation

$$\begin{aligned} h_{1,2} =& \sqrt{2}\min(x_2 - x_1, y_2 - y_1) \\ &+ \{\max(x_2 - x_1, y_2 - y_1) \\ &- \min(x_2 - x_1, y_2 - y_1)\} \end{aligned} \tag{4}$$

Note that, since our A* implementation is constrained to search for paths that only move along cardinal and ordinal headings, this heuristic will never overestimate and is therefore admissible. This A* implementation was tested on a range of test cases provided by the instructors, and the algorithm's runtime and output optimality were observed.

2) Exploration: Frontiers are identified by searching for unobserved cells which are adjacent to observed unobstructed cells. During exploration, A* attempts to find paths from the robot to the center of each frontier, and the nearest frontier is selected as an exploration target. This serves as the core loop which drives the exploration process.

If the center of a frontier lies within an untraversable area, Breadth First Search (BFS) is applied to the frontier center to find the closest traversable cell, which is selected as the exploration target instead. If A* fails to find a traversable path, the obstacle radius is adaptively reduced such that more cells become traversable; this continues until A* successfully finds an alternative path.

3) Localization with Unknown Initial Pose: State transition logic is shown in Figure 5, while data transmission between LCM nodes is shown in Figure 6. One exploration node is run in order to move around while avoiding obstacles, while another instance of SLAM runs in parallel, attempting to converge on the robot's pose. During particle resampling, 20% of particles are set randomly on this map. The top likelihood 80% particles are identified and the variance in their position, as defined by $Var(\mathbf{x}) + Var(\mathbf{y})$, is calculated. The robot will repeatedly explore the maze until this metric remains under a threshold (0.04) for 10 consecutive timesteps, at which point the robot is considered localized.

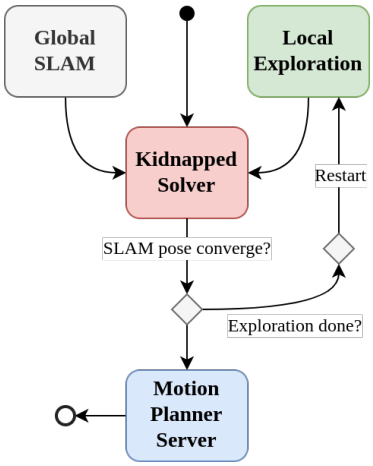


Fig. 5: State transition logic for bonus

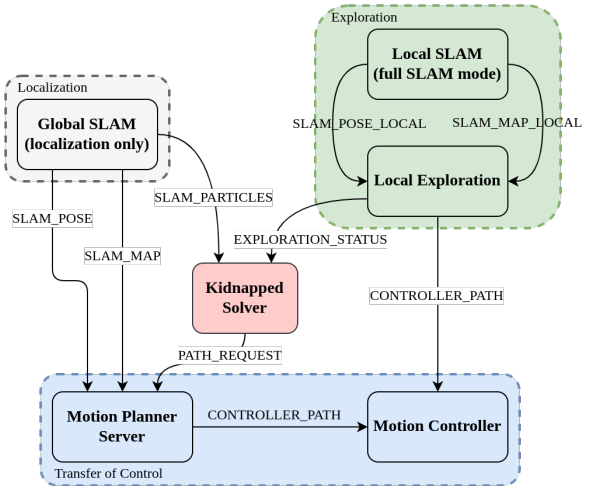


Fig. 6: LCM graph with global and local channels

III. Results

A. Odometry

1) Wheel Speed Calculation: Velocities reported by the algorithm described in Algorithm 1 were consistent with expectations. This algorithm was implemented to improve the robustness of velocity calculations at low speeds; however, it is likely that this level of robustness was unnecessary since in practice the robot always operated at fairly high speeds.

2) Gyrodometry: According to Borenstein et. al. [1], the angle delta threshold at which to begin using gyrodometry is usually set to be 0.125°. The authors chose a value of 0.09°, which empirically perfomed better, perhaps due to slightly less reliable encoders.

3) Dead Reckoning: The final tuned values of r_{wheel} and b_{wheel} are 0.04175 and 0.15, respectively. Figure 11 shows the tuned dead-reckoning pose estimate trajectory as the robot drives in a square. A substantial amount of drift, especially during turns, can be seen. However, this drift is gradual and consistent. The authors interpreted this as indicating odometry to be sufficiently precise over short timeframes.

B. Motion Control

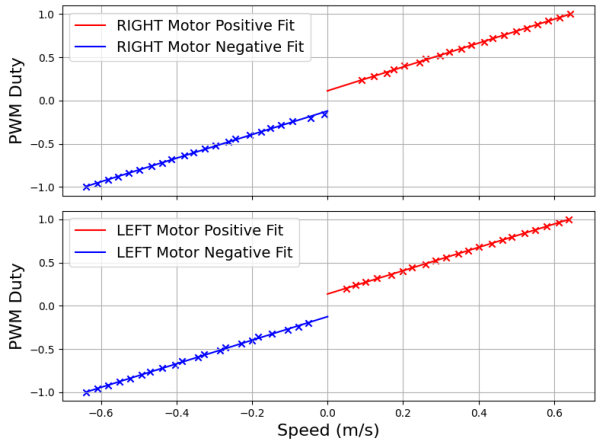


Fig. 7: Motor PWM linear mapping calibration

1) Wheel Controller: Figure 7 shows the fitted linear mappings for the open loop controllers, with final calibration parameters shown in Table I. Variation in these parameters across robots is likely the result of a combination of factors, including:

- Tolerances in mechatronic hardware components
- Tolerances in the motor driver circuitry
- Tolerances in the PWM circuitry on the RasPi

TABLE I: Motor calibration parameters

	Type	Positive	Negative
Left Wheel	Slope	1.3672	1.3760
	Intercept	0.0779	-0.0725
Right Wheel	Slope	1.3712	1.3735
	Intercept	0.0826	-0.0931

When given symmetric set-points, the open loop controller resulted in noticeable differences in left and right wheel speeds. The closed loop controller was able to dramatically mitigate this difference, although it wasn't able to completely eliminate it. PID parameters were tuned to a point where the robot did not exhibit noticeable oscillations in response to step changes in set-points, and appeared to reach a steady-state velocity almost immediately (Figure 8).

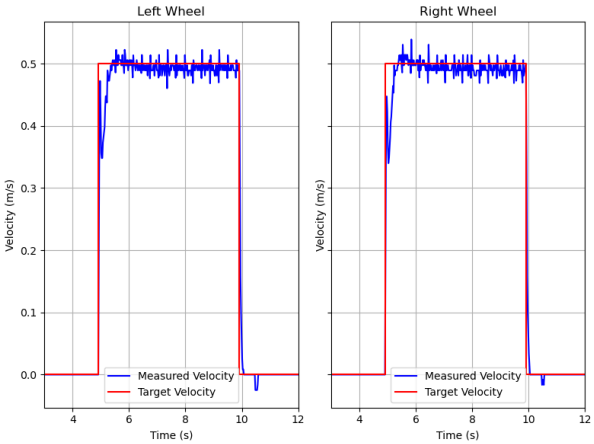


Fig. 8: Step response of wheel speed PID controller

2) Velocity Controller: Figure 9 shows the robot's response to step inputs of various velocities. From the figure, only a very small overshoot can be observed in linear velocity and almost no overshoot can be observed in angular velocity. Both velocities converge to the command speed within 0.5 seconds. Figure 10 shows the robot's position and heading with several step inputs along the x-axis in a roll. There are only slight offsets in the y-axis and its heading is almost negligible. The movement along the x-axis meets the estimation. Both tests validate the velocity controller's performance. Due to the satisfactory performance of an open-loop design, closed-loop velocity control was not used to avoid additional tuning and complexity.

During the experiments, the maximum and mimimum velocities for a stable driving are 0.8 m/s and 0.1 m/s in translation movement while 2π rad/s and $\pi/6$ rad/s in rotation movement.

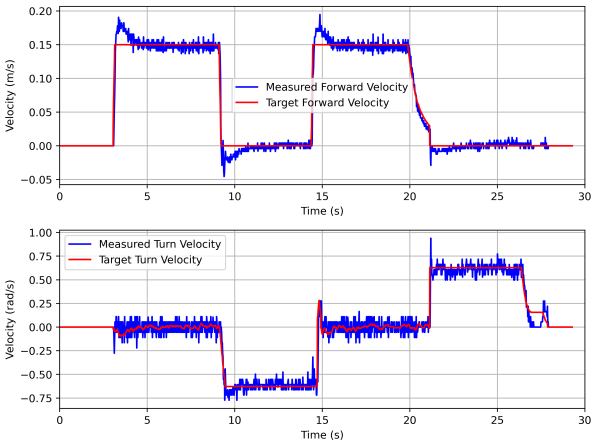


Fig. 9: Step response of linear and angular velocities

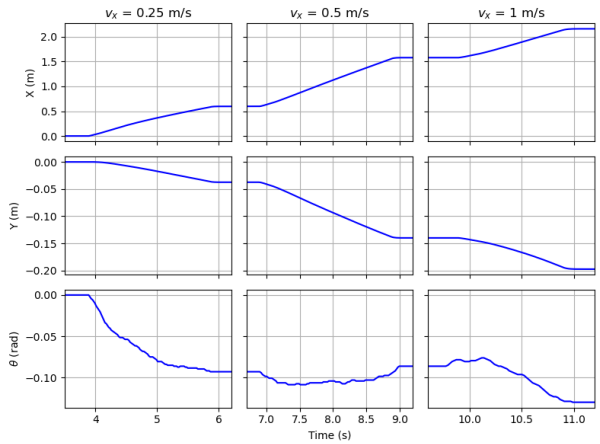


Fig. 10: Robot position and heading for different step inputs

TABLE II: PID gains and lowpass filter parameters. τ is the time constant and f is the frequency.

	P Gain	I Gain	D Gain	τ	f
Wheel Speed PID Filter	0.4	2.4242	0.0436	0.04 s	50 Hz
Wheel Speed Lowpass Filter	\	\	\	0.08 s	50 Hz
Wheel Encoder Lowpass Filter	\	\	\	0.04 s	50 Hz

3) Motion Controller: Figure 11 shows the robot's dead-reckoning pose as it drives four laps around a square pattern. Although drift in both the robot's estimated and true pose was observed, these results nonetheless verify that the motion controller is capable of following complex paths with multiple waypoints. Additionally, the implementation of an acceleration constraint eliminated wheel slip and tip-overs.

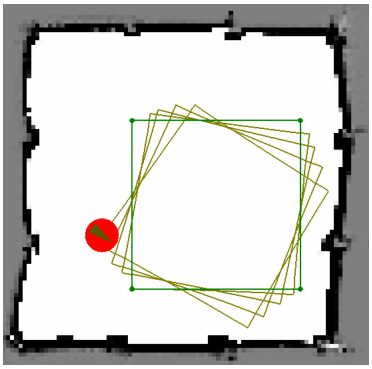


Fig. 11: Dead-reckoning odometry as the robot drives around a square four times.

Figure 12 shows the SLAM result after driving 4 loops in a 1 m $\times$ 1 m square. With the help of SLAM, the robot can localize itself more accurately and its actual path is exactly the same as shown in Figure 12. The start position and the end position are within a 3-cm difference, validating the functionality of the controller.

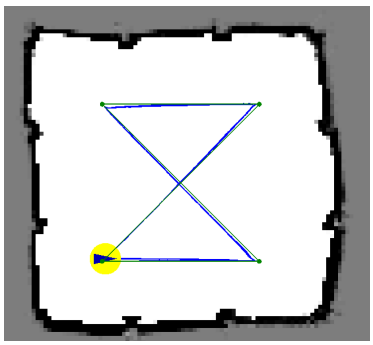


Fig. 12: SLAM while driving 4 circuits in event 1 map

C. Simultaneous Localization and Mapping (SLAM)

1) Mapping: Figure 13 shows the results on *tobstacle_slam_10mx10m_5cm.log*. The map explored is of high quality with clear edges and boundaries, validating the good mapping function.

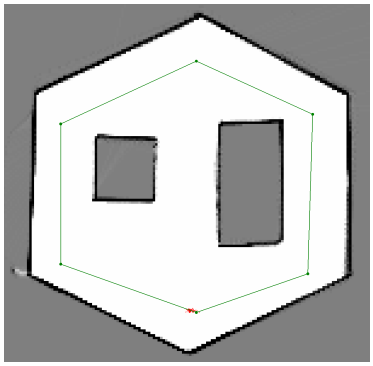


Fig. 13: Mapping on *obstacle_slam_10mx10m_5cm.log*.

2) Monte Carlo Localization: Table III shows the final tuned values of action model parameters k_1 and k_2. These values were found to result in tight particle distributions that covered the true pose.

TABLE III: Uncertainty parameters of the action model

Action Model Parameters	$\mathbf{k}_1$	$\mathbf{k}_2$
Value	0.08	0.03

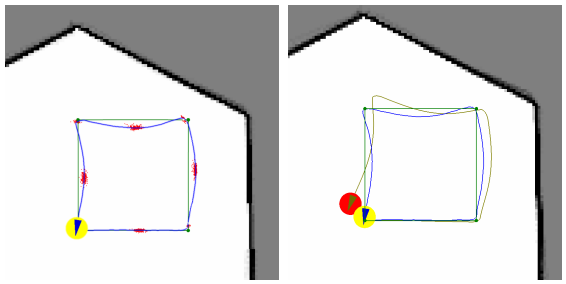
It was found that the raycasting sensor model cost too much runtime to be used, except at unfeasibly slow update rates. Therefore, the simplified sensor model was used for the remainder of the lab.

TABLE IV: Update speed of the particle filter with different numbers of particles on RasPi

	Particle Number			
	100	300	500	1000
Update Time (us)	20380	51111	83274	160187

Table IV shows the SLAM update rate associated with various particle counts. The estimated maximum number of particles our filter can support running at 10Hz on the rPi is around 600.

Figure 14a shows particle distributions at the midpoints and corners, demonstrating tight and accurate groupings. Figure 14b shows error between odometry and SLAM poses on *drive_square_10mx10m_5cm.log*. Accumulation of error due to odometry drift is clearly visible, demonstrating the need for a SLAM system to correct this drift.



(a) Particles distribution on the midpoints and corners (b) SLAM and Odometry poses deviations

Fig. 14: Validation of the SLAM system

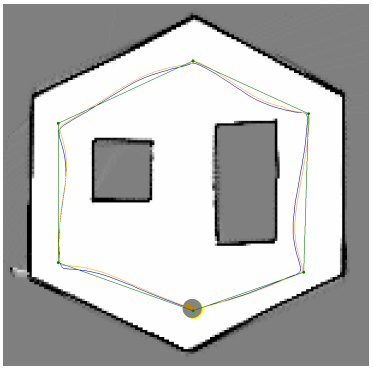


Fig. 15: Validation of accuracy of the SLAM system on *obstacle_slam_10mx10m_5cm.log*.

Figure 15 compares the ground-truth pose and the SLAM pose. They align well and the RMS error between them is only 0.0604. At the same time, as mentioned in the previous section, Figure 12 shows the SLAM path recorded in event 1 and it aligns well with the actual position of the robot in the lab. When it returned to its starting point, the position error is within 3 cm, showing a convincing SLAM system accuracy.

D. Planning and Exploration

1) Path Planning: Figure 16 compares the planned A* path during event 2 to the actual robot trajectory. The robot doesn't perfectly follow the planned path, but still reaches the final pose with high accuracy; this is an intentional consequence of the decision to widen tolerances associated with intermediate path waypoints.

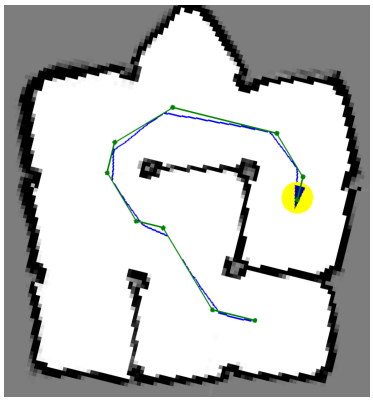


Fig. 16: Comparison of deviations between the planned path and the driven path in the map of event 2

TABLE V: Statistics of A* execution times (tested on Intel Core i7-12700H)

Test Cases	Timing (us)			
	Min	Max	Median	STD
Empty Grid	63	151234	1885	59915
Filled Grid	9	44	20	13.65
Narrow Grid	46	163026	46	76619
Wide Grid	17	25332	25332	11641
Convex Grid	11	127	58	45.14
Maze Grid	1070	6964	1947	2263

Table V shows the time takes for the A* algorithm to find the shortest path for different test cases. Though all of them can be finished within 1 second, it's still not as fast as expected. A potential reason for that is all the waypoints are initialized by opening new memory space to save memory usage each iteration. However, these memories might not be cleaned properly leading to some memory leaks and decreased operation speed.

During testing, it was observed that the A* implementation yielded suboptimal paths that included swerving motions. This was caused by a low-precision approximation of $\sqrt{2}$, rendering the heuristic used inadmissible. After increasing the precision of this approximation, output paths were of optimal length, suggesting the A* implementation was working properly.

2) Exploration: Figure 16 shows the SLAM map after event 2, while figure 17 shows the map built during event 3. During both events, the robot achieved full exploration without collisions, and subsequently returned to within 3cm of a home position. Resultant maps were of high quality and contained minimal artifacts.

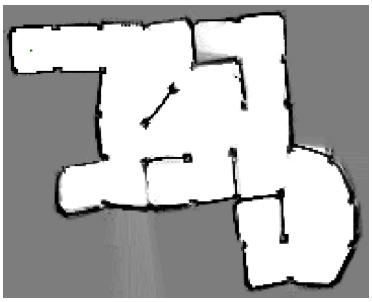


Fig. 17: Exploration result from competition event 3

3) Localization with Unknown Initial Pose: In the bonus event of competitions, the robot was able to begin from an unknown initial pose in a pre-mapped environment, and correctly converge upon its true pose while exploring the environment without hitting obstacles. The robot was then able to return to a predefined home pose with high precisions. Note that before the localization converges, the lower motion controller only receives command path from exploration agent, and switch to path from motion planner server after that (Figure 6).

Fig. 18: Localization with unknown initial pose and sampling augmentation in a known map

IV. Discussion

Overall, the authors were able to achieve the core autonomy goals of motion, SLAM, and exploration. Nonetheless, several opportunities for improvement presented themselves during the project.

A. Odometry and Motion Controller

Using a robust odometry calibration methods like UMBmark[5] can eliminate most of the system and non-system errors and lead to a more accurate pose estimation. Using a more advanced motion controller could enable the following of advanced paths such as splines, which could have enabled smoother driving.

B. SLAM

Several compromises were made in the SLAM system to account for runtime constraints. The most significant was the use of a simplified sensor model which only checked three occupancy grid cells per LIDAR beam; this lacked robustness to noise or transient errors and more robust designs might yield improvement. Additionally, the authors observed that brief bouts of uncertainty in SLAM pose would quickly degrade or destroy the SLAM map, and thus SLAM could potentially be made more robust by implementing a system to modulate the speed of map adjustments based on pose confidence.

C. Exploration and Planning

Path planning could also have been improved further. Most notably, the authors' implementation of A* searched for paths in (x, y) space, which resulted in paths with minimum length but not a minimum amount of turning. This could have been solved by changing our A* implementation to search for paths in a 3D (x, y, θ) space as opposed to a 2D (x, y) space. Additionally, A* runtime was found to be disappointingly long and likely could be improved via code profiling and optimization. Alternatively, more advanced path planning alternatives such as RRT could be experimented with.

V. Conclusion

Throughout the MBot lab, the authors implemented robot autonomy at various levels of abstraction by developing motion, SLAM, and exploration capabilities. Through a variety of tests as well as competition tasks, the authors were able to demonstrate these autonomous capabilities, cumulating in the ability to successfully localize the MBot starting from an unknown pose.

References

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

[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[3] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems journal*, vol. 4, no. 1, pp. 25–30, 1965.

[4] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[5] J. Borenstein and L. Feng, "Umbmark: A benchmark test for measuring odometry errors in mobile robots," in *Mobile Robots X*, vol. 2591. SPIE, 1995, pp. 113–124.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2006.

[7] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005.