Lab #5: Robust Monte Carlo Localization via Implementing, Tuning, and Evaluating Probabilistic Sampling and Motion Models

Team #7

Edward Cheng
Audrey Lee
Christopher Liem
Drew Tufto
Richard Yeboah (Editor)

Robotics: Science and Systems

April 12, 2025

1 Introduction

Yibo Cheng

Navigation involves two critical steps: constructing a reliable map of the environment and accurately following it. Despite huge advancements in high-definition mapping, precise pose estimation is still challenging for an autonomous vehicle. In this lab, we implemented Monte Carlo Localization, a robust particle filtering method that iteratively estimates the robot's orientation and position. This lab lays the foundation for the future use of path planning and pure pursuit control. By integrating and extending these components into a unified localization system, we take a significant step toward enabling fully autonomous navigation in realistic environments.

We are provided with an accurate map of the basement of the Stata Center. The objective is to estimate the robot's position within this known environment as it navigates through it. Relying solely on odometry for localization is unreliable due to the accumulation of drift over time, and sensor data such as camera or LiDAR readings can often be noisy or ambiguous. To overcome these limitations, our model combines both motion and perception data within a probabilistic framework. Specifically, we leveraged a motion model to update particle positions based on noisy odometry, a sensor model to evaluate the likelihood of each particle using LiDAR scan data, and a resampling strategy to reinforce high-confidence pose estimates. This fusion of information enables the

robot to maintain a consistent and accurate estimate of its location, even in the presence of uncertainty.

2 Technical Approach

2.1 Motion Model

Drew Tufto

The goal of the motion model is to update our particles in a manner that is consistent with the motion of our actual car. In a real deployment of our vehicle, our odometry information is subject to noise which makes a deterministic motion model infeasible, as we would quickly drift to an incorrect location. Instead, we model the possible locations we could be after a given noisy odometry and previous state, and sample from that distribution to update our particles.

2.1.1 Technical Implementation

The model takes wheel odometry as input. Specifically, the input is the incremental displacement in the local frame of the robot, given as the vector.

$$\Delta X = \begin{bmatrix} \Delta x & \Delta y & \Delta\theta \end{bmatrix} \tag{1}$$

This vector is derived by combining the vehicle's motor outputs and steering inputs through dead reckoning, and is published to the `/Odom` topic.

The heart of the model is the propagation equation that updates the position of each particle (x, y) and heading θ. We first model the noise as zero-mean Gaussians, applied independently to each dimension of the odometry. Then, we apply a simple twist matrix to the odometry position to transform it into the global frame. Combining the transformed position with the absolute change in theta, we obtain the global state vector for each particle.

Mathematically, given a particle with state $[x,\ y,\ \theta]$, the update is performed as follows:

$$\begin{aligned} dx_{\text{noisy}} &\sim \mathcal{N}(\Delta x, \sigma_x^2) \\ dy_{\text{noisy}} &\sim \mathcal{N}(\Delta y, \sigma_y^2) \\ d\theta_{\text{noisy}} &\sim \mathcal{N}(\Delta\theta, \sigma_\theta^2) \end{aligned} \tag{2}$$

$$\begin{aligned} x_{\text{new}} &= x + \cos(\theta)\, dx_{\text{noisy}} - \sin(\theta)\, dy_{\text{noisy}}, \\ y_{\text{new}} &= y + \sin(\theta)\, dx_{\text{noisy}} + \cos(\theta)\, dy_{\text{noisy}}, \\ \theta_{\text{new}} &= \theta + d\theta_{\text{noisy}}, \end{aligned}$$

where the terms dx_{noisy}, dy_{noisy}, $d\theta_{\text{noisy}}$ represent the odometry data perturbed by Gaussian noise. Note that the noise is sampled independently for each par-

ticle. Our resulting model utilized the following variance values at deployment:

$$\sigma_x^2 = 0.1(m), \sigma_y^2 = 0.1(m), \sigma_\theta^2 = \texttt{np.deg2rad}(20)$$

Finally, our motion model returns an N x 3 matrix of the updated particles, where each particle is described as

$$P_i = \begin{bmatrix} x_{new} & y_{new} & \theta_{new} \end{bmatrix} \tag{3}$$

2.2 Sensor Model

Christopher Liem

The sensor model's goal is to assign likelihoods to each particle for our particle filter to use.

2.2.1 Likelihood Model Introduction

Let L be the number of beams in our LiDAR scan. For a LiDAR scan acquired at iteration k, $z_k^{1:L}$, define the probability that each particle $\hat{p}_k^{(j)}$ produces the scan given the particle's hypothesized position $x_k^{(j)}$ and a static map m at time step k by

$$p\left(z_k^{1:L} \,|\, x_k^{(j)}, m\right)$$

Assuming that the LiDAR measurements are independent,

$$p\left(z_k^{1:L} \,|\, x_k^{(j)}, m\right) = \prod_{i=1}^{L} p\left(z_k^{(i)} \,|x_k^{(j)}, m\right) \tag{4}$$

Given a particle pose estimate $x_k^{(j)}$, we can compute the ground-truth distances $d^{1:L}$ that an ideal LiDAR would observe using ray-casting on m from pose $x_k^{(j)}$. However, assigning likelihoods to each particle at every iteration is computationally expensive, so we precompute a lookup table that discretizes our continuous and complex distribution at runtime. The columns of the lookup table represent the ground truth distances which lie between 0 and $z_{\max} = 200$ and the rows correspond to the LiDAR measurements, resulting in a L by $z_{\max} + 1$ table. Discretization is discussed in section 2.2.2.

2.2.2 Likelihood Model Discretization

For this section, let d be the ground truth range of a particular measurement. We model the likelihood of each range measurement as a mixture of likelihoods of four different events:

1. A known obstacle in the map is detected. The event can be modeled as a Gaussian distribution centered around the ground truth distance between our predicted pose and the nearest known obstacle on the map

$$p_{\text{hit}}\left(z_k^{(i)} \,|\, x_k, m\right) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(z_k^{(i)} - d\right)^2}{2\sigma^2}\right) & \text{if } 0 \le z_k \le z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where σ is the standard deviation of the Gaussian. For our implementation, $\sigma = 8.0$.

2. A short measurement is detected, possibly due to unknown obstacles and internal LiDAR reflections. The distribution for this event is a downward sloping line.

$$p_{\text{short}}\left(z_k^{(i)} \mid x_k,\, m\right) = \frac{2}{d}\begin{cases} 1 - \frac{z_k^{(i)}}{d} & \text{if } 0 \le z_k^{(i)} \le d \text{ and } d \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

3. A large measurement is detected, possibly because LiDAR beams did not bounce back to the sensor. This is a uniform distribution.

$$p_{\text{max}}\left(z_k^{(i)} \mid x_k,\, m\right) = \begin{cases} \frac{1}{\epsilon} & \text{if } z_{\text{max}} - \epsilon \le z_k^{(i)} \le z_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

4. A completely random measurement is detected. This is modeled as

$$p_{\text{rand}}\left(z_k^{(i)} \mid x_k,\, m\right) = \begin{cases} \frac{1}{z_{\text{max}}} & \text{if } 0 \le z_k^{(i)} \le z_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The law of total probability allows us to write the likelihood of range measurement i as

$$\begin{aligned} p\left(z_k^{(i)} \mid x_k,\, m\right) = {} & \alpha_{\text{hit}} p_{\text{hit}}\left(z_k^{(i)} \mid x_k,\, m\right) \\ & + \alpha_{\text{short}} p_{\text{short}}\left(z_k^{(i)} \mid x_k,\, m\right) \\ & + \alpha_{\text{max}} p_{\text{max}}\left(z_k^{(i)} \mid x_k,\, m\right) \\ & + \alpha_{\text{rand}} p_{\text{rand}}\left(z_k^{(i)} \mid x_k,\, m\right) \end{aligned} \tag{9}$$

where α_{hit}, α_{short}, α_{max}, and α_{rand} are the priors of acquiring a hit, short, max, or random measurement respectively. In order for this to be a proper probability distribution,

$$\alpha_{\text{hit}} + \alpha_{\text{short}} + \alpha_{\text{max}} + \alpha_{\text{rand}} = 1 \tag{10}$$

We chose $\alpha_{\text{hit}} = 0.74$, $\alpha_{\text{short}} = \alpha_{\text{max}} = 0.07$, and $\alpha_{\text{rand}} = 0.12$.

2.3 Particle Filter

Christopher Liem

In this section, we discuss how to utilize the motion and sensor models to implement Monte Carlo Localization on our robot in simulation and the real world.

2.3.1 Particle Initialization

We begin by initializing the particles. We subscribe to a ROS topic, which retrieves the robot's pose estimate from when the algorithm began. This yields a `PoseWithCovariance` message with information about position, orientation, and covariance. The robot's x and y coordinates are extracted from the positional information in Cartesian form. The rotation around the z-axis, θ, can be extracted by converting the orientation, which is a quaternion, to yaw using the formula

$$\theta = \arctan\left(2 \cdot (q_w q_z + q_x q_y),\ 1 - 2 \cdot (q_y^2 + q_z^2)\right) \tag{11}$$

The covariance matrix of the message takes the form

$$\sum = \begin{bmatrix} \sigma_x^2 & \cdots & \cdots & \cdots & \cdots \\ \cdots & \sigma_y^2 & \cdots & \cdots & \cdots \\ \cdots & \cdots & \sigma_z^2 & \cdots & \cdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \sigma_\theta^2 \end{bmatrix} \tag{12}$$

Each entry along the diagonal σ_k^2 represents the variance of k and it follows that σ_k is the standard deviation of k. Each particle $\hat{p_i}$ can be initialized in the form

$$\hat{p_i} = \begin{bmatrix} X_i & Y_i & \Theta_i \end{bmatrix} \tag{13}$$

where X_i, Y_i, and Θ_i follow the Gaussian distributions

$$\begin{aligned} X_i &\sim \mathcal{N}(x, \sigma_x) \\ Y_i &\sim \mathcal{N}(y, \sigma_y) \\ \Theta_i &\sim \mathcal{N}(\theta, \sigma_\theta) \end{aligned} \tag{14}$$

Sampling each particle from a Gaussian distribution effectively adds noise to our particles, accounting for any uncertainty within our odometry estimates. Furthermore, this process guarantees that our particles are initialized within the scope of the robot's initial pose. We could've initialized the particles randomly across the Stata basement, but doing so may require numerous particle updates to converge to the robot's actual pose, which is inefficient.

We store our particles in a vector P. If N is the number of particles, then P takes the form

$$P = \begin{bmatrix} \hat{p_1} \\ \hat{p_2} \\ \vdots \\ \hat{p_N} \end{bmatrix} = \begin{bmatrix} X_1 & Y_1 & \Theta_1 \\ X_2 & Y_2 & \Theta_2 \\ \vdots & \vdots & \vdots \\ X_N & Y_N & \Theta_N \end{bmatrix} \tag{15}$$

2.3.2 Particle Updates

When we receive `LaserScan` data from our LiDAR, we use the sensor model to evaluate the probability of each particle existing given the new data, which

we downsampled to 100 points for efficiency. This is the update step of Monte Carlo Localization. We store the probabilities in a vector W_N.

$$W_N = \begin{bmatrix} p(\hat{p_1}) \\ p(\hat{p_2}) \\ \vdots \\ p(\hat{p_N}) \end{bmatrix}$$

where $p(p_i)$ is the probability that the ith particle exists. We then choose whether to resample during this iteration. Let S be a Bernoulli random variable with parameter p ($S \sim \text{Ber}(p)$). If $S = 1$, then we don't resample and multiply the weights of each particle by the probabilities that they exist. Suppose that w_i is the current weight of the ith particle. The new weights will be

$$W = \begin{bmatrix} w_1 \cdot p(\hat{p_1}) \\ w_2 \cdot p(\hat{p_2}) \\ \vdots \\ w_N \cdot p(\hat{p_N}) \end{bmatrix} \tag{16}$$

If $S = 0$, then we resample. First, we transform the particle probabilities by taking their cube roots and normalizing the result. In other words, we have

$$W_N = \begin{bmatrix} \frac{p(\hat{p_1})^{\frac{1}{3}}}{\sum_{i=1}^{N} p(\hat{p_i})^{\frac{1}{3}}} \\ \frac{p(\hat{p_2})^{\frac{1}{3}}}{\sum_{i=1}^{N} p(\hat{p_i})^{\frac{1}{3}}} \\ \vdots \\ \frac{p(\hat{p_N})^{\frac{1}{3}}}{\sum_{i=1}^{N} p(\hat{p_i})^{\frac{1}{3}}} \end{bmatrix} \tag{17}$$

This process prevents issues that arise when averaging with weights that are tiny compared to others. We take a new sample of N particles from our current particles which are distributed according to their probability of existing as given by W_N. P will be updated according to the particle we take and the corresponding entry in W will be updated with the particle's weight in W_N. For example, if our first sampled particle is $\hat{p_{50}}$, then we update $P^{(1)} = \hat{p_{50}}$ and $W^{(1)} = W_N^{(50)}$. Note that notation such as $W^{(i)}$ is used to denote the ith entry of W. Resampling increases the accuracy of our robot's pose estimation because it is likely to keep the particles that have high probabilities of existing while filtering out particles that aren't as accurate, discarding unimportant information.

When we receive odometry data, we update our particles. This is the prediction step of Monte Carlo Localization. From the `Odometry` message, we extract the linear velocity in the x and y direction, which is given by $\dot{x}$ and $\dot{y}$, respectively. Moreover, we extract the angular velocity along the z axis, given by $\dot{\theta}$. Furthermore, we compute the amount of time that passed since we last received odometry data. Using all these parameters, we use our motion model to update the pose of each particle.

2.3.3 Pose estimation

After updating our particles, we can estimate our robot's pose $r = \begin{bmatrix} x & y & \theta \end{bmatrix}$. Our robot's current x and y coordinates are given by a weighted average of the respective coordinates of the particles. In other words, we have

$$x = \frac{\sum_{i=1}^{N} w_i X_i}{\sum_{i=1}^{N} w_i} \text{ and } y = \frac{\sum_{i=1}^{N} w_i Y_i}{\sum_{i=1}^{N} w_i} \tag{18}$$

where w_i is the weight of the ith particle. Our robot's current yaw is computed using a weighted circular average of the yaw of each particle.

$$\theta = \arctan\left(\sum_{i=1}^{N} w_i \sin\Theta_i, \sum_{i=1}^{N} w_i \cos\Theta_i\right) \tag{19}$$

We published this estimated pose to both `Odometry` and `Transform` messages, where the position of each message is given by $\begin{bmatrix} x & y & 0 \end{bmatrix}$ and the orientation by calling the built-in `quaternion_from_euler` function from the `tf_transformations` library.

$$q_x, q_y, q_z, q_w = \text{quaternion\_from\_euler}(0,\ 0,\ \theta) \tag{20}$$

We store all x, y coordinates of the robot in a `Marker` message of type `LINE_STRIP` to visualize previously estimated robot poses in real time.

3 Experimental Evaluation

3.1 Unit Testing

Audrey Lee

Before integrating the components of our system, we wanted to ensure that the outputs from both models (motion and sensor) aligned with our expectations. Figure 1 demonstrates the effect of injected noise on our motion model. As expected, higher noise levels lead to increasingly erratic and diverse particle trajectories, even when following the same control commands. Notably, the particles also diverge more from the expected position over time, reflecting noise accumulation. This motivates the need for the sensor model to correct drift by anchoring motion model estimates to real-world observations.

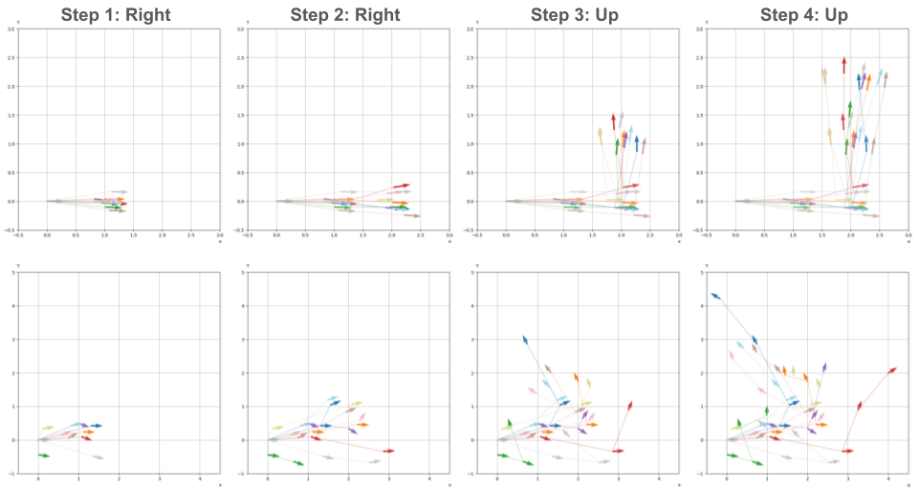

Figure 1: **Increased noise causes more deviation in our motion model**. The intended odometry commands are two unit steps to the right and two unit steps up. The top 4 images show small deviations with a noise of $\sigma_{x,y} = 0.1,\ \sigma_\theta = 5°$. The bottom 4 images show much more unpredictable deviations with a larger noise of $\sigma_{x,y} = 0.5,\ \sigma_\theta = 30°$.

To verify the sensor model's behavior, we created 6 synthetic particles by adding noise to a ground truth LaserScan. Intuitively, the particles with less noise should more closely match the ground truth measurements and be assigned higher probabilities by the sensor model. This relationship is reflected in Figure 2, reinforcing the validity of our sensor model.

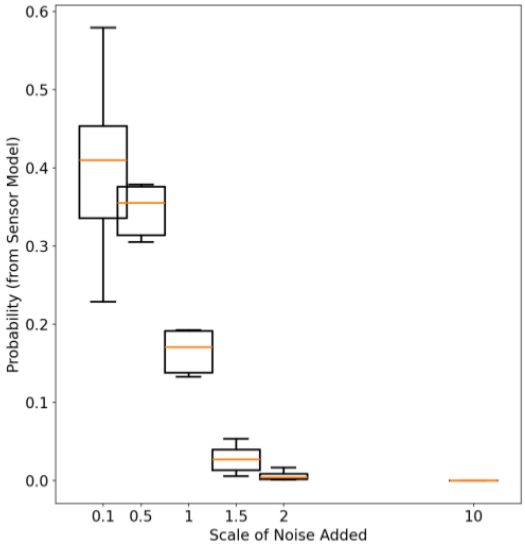

Figure 2: **Our sensor model prioritizes particles that are similar to the ground truth**. The particle with the least added noise (left-most, $\sigma = 0.1$) was assigned the highest probability, with probabilities decreasing as more noise was added. The sensor model assigned negligible probability to the particle with the most added noise ($\sigma = 10$).

3.2 Simulation

Richard Yeboah

Within RViz, we initialized the bot with various random initial poses and orientations. The simulated robot was commanded with constant +x-velocity as a base case.

For our filter, we set the number of sampled particles to 200 and the number of beams to 100. We found that this was an appropriate benchmark to begin testing on both the simulation and the robot.

3.2.1 Testing

In order to both test out our filter and create a preliminary calibration for the robot, we varied the amount of noise added while running each initial condition. Visually, RViz reported pose estimations with a green arrow and particles with red arrows.

3.2.2 Results

With no noise, our simulation produced a very accurate pose estimation relative to the ground truth and closely clumped particles. As we increased noise, the deviation in pose estimation increased as expected, yet it still stayed roughly direct toward ground truth.

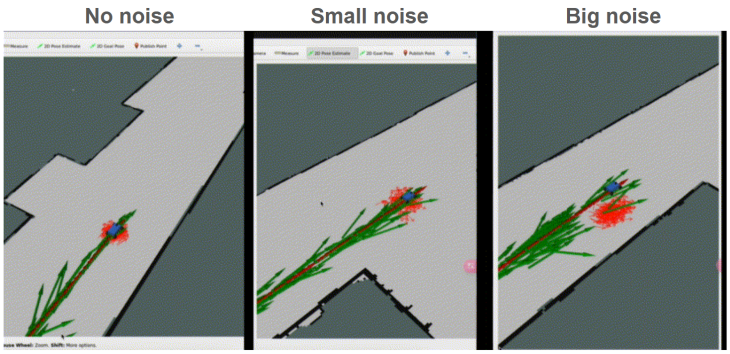

Figure 3: Testing the pose estimation and particle sampling with no noise, low noise, and large noise.

3.3 On Robot

Audrey Lee

Much of our initial robot evaluation was qualitative, as the system behaved strangely due to lag. We troubleshooted by decreasing the number of particles and optimizing our code until pose updates happened in real time and provided stable, continuous estimates. One example of optimization, replacing for loops with numpy operations, can be seen below.

```
# Original Code (For Loops)
probabilities = []
for ground_truth in scans:
    indices = np.array(list(zip(observation, ground_truth)))
    p = self.sensor_model_table[indices[:,0], indices[:,1]]
    p = np.exp(np.sum(np.log(p)))
    probabilities.append(p)
```

```
# Optimized Code (Numpy Operations)
observations = np.tile(observation, (scans.shape[0], 1))
probabilities_array = self.sensor_model_table[observations, scans]
probabilities = np.exp(np.sum(np.log(probabilities_array), axis=1))
```

To quantitatively evaluate our system, we defined a "ground truth" by manually driving along a straight line 1 meter away from a wall. We mapped real-world coordinates to pixels in RViz and graphed the expected wall location in green (Figure 4). The wall was diagonal, so the x-position changed as we drove alongside it. Across 3 forward-driving trials, we observed an average x-error of

0.26 meters. Due to shakier manual control when driving backward - visible as waviness on the right half of Figure 4 - the average x-error increased to 0.5 meters.

Estimating the y-position was more challenging as the robot was in a long hallway. However, the y-position plot follows expected trends: it decreases while driving forward, flattens during the turn, and returns to approximately the starting value ($y \approx 7$) as we drive back.

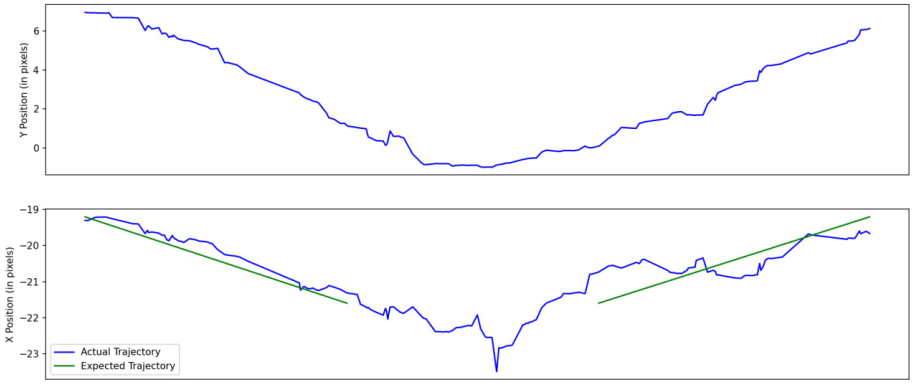

Figure 4: **Estimated X (bottom) and Y (top) locations while driving along a straight wall.** The expected x-location is shown in green (excluding the turn in the middle). Our MCL system closely tracks the expected location, even when driving forward (left) and backward (right).

For further qualitative evaluation, we drove halfway around the Stata basement until connection was lost, logging the estimated trajectory (Figure 5). The system maintained reliable localization over time in both feature-sparse (e.g., wide hallways) and feature-rich (e.g., narrow corridors with poles) environments and was robust to dynamic obstacles like pedestrians.

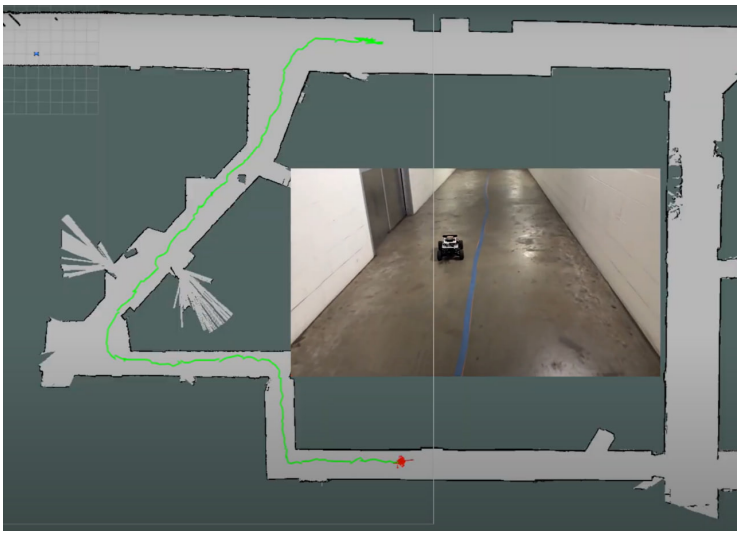

Figure 5: **Localization in the Stata basement**. Our system maintains reliable localization in a realistic use case: driving around unique hallways, tight turns, and moving pedestrians.

4 Conclusion

We successfully designed and implemented a robust Monte Carlo Localization (MCL) system capable of accurately estimating the robot's pose within a known map of the Stata Center basement. Our approach integrates a well-tuned motion model, a probabilistic sensor model based on LiDAR data, and an adaptive resampling strategy to maintain a stable and precise localization estimate. Both simulation and real-world experiments demonstrated that our system performs reliably and remains resilient in the face of common challenges such as sensor noise, odometry drift, and dynamic obstacles.

While the core functionality of our localization pipeline is complete, there are several areas for further refinement. For instance, improving computational efficiency would allow us to support a higher number of particles in real time, increasing accuracy without sacrificing performance. Additionally, incorporating more adaptive or intelligent resampling techniques could further improve robustness in ambiguous or feature-sparse environments. For a more rigorous quantitative evaluation, we could instruct the robot to follow 1 meter from a wall around the basement, providing a clearer ground truth trajectory against which to benchmark our system's accuracy.

5 Lessons Learned

5.1 Edward Cheng

Throughout this lab, I significantly deepened my understanding of code optimization. Previously, I often relied on traditional for loops, which are intuitive but inefficient, especially for large datasets or real-time applications. During this lab, I learned how to leverage NumPy and array slicing techniques, which drastically improved the efficiency and speed of our implementation. This shift in approach allowed us to handle complex operations more elegantly and with less computational overhead. Additionally, I came to appreciate the value of unit testing. When debugging complex systems, its often unclear where issues originate. Writing unit tests for each submodule allowed us to quickly isolate and identify the source of bugs, making our development process much smoother and more reliable.
About communication and collaboration. While we initially divided the workload by assigning different components to each team member, we quickly realized that this approach was not ideal. Many tasks could not proceed until those foundational parts were completed. This created bottlenecks and idle time for some teammates. In hindsight, a better strategy would have been to pair up on the critical modulestwo people working on the motion model and two on the sensor model. This parallel collaboration would not only lead to better design choices, but also facilitate debuggin process.

5.2 Audrey Lee

Although all of our lab members were finally in Boston during this lab (a first for us), adjusting to being a 5-person team and shaking off the dust from spring break made communication more challenging than I think any of us initially expected. There were definitely instances of confusion around team members' availability, and I think that couldve been avoided by more vigorous communication about our meeting schedule and obtaining confirmation from all of the group members that they would be able to attend the planned work sessions. Because there were so many moving parts in this project, we also got confused about who was working on which part and how much progress they had made - to address this, our instructors recommended having a working document or permanent place to put individual updates. Although we received this recommendation a little bit too late for this lab, I think it would be a great idea for us to implement moving forward.

On the technical side, this lab highlighted the value of peer programming and debugging with many extra eyes. As we were debugging both individual modules and the full system, we ran into a lot of silly syntax, naming convention, and configuration errors that couldve been easily avoided with a second pair of eyes to review what we programmed. This slowed us down during this lab, but I think that it was a valuable lesson to help us improve our efficiency and save us

some headache for the final lab and competition. Overall, while this lab was a struggle, both technically and in terms of teamwork, I believe that we came out of it with a lot of great tools and lessons that will help us succeed throughout the rest of the class.

5.3 Christopher Liem

From a collaboration and communication standpoint, I learned how to efficiently distribute tasks between team members with flexibility. Initially, we assigned tasks as follows: Drew works on the motion model, Audrey works on the sensor model, I work on the particle filter, and Yibo works on SLAM. Drew finished his task swiftly and offered to help me with coding the particle filter because he thought that the motion model wasn't a lot of work. The particle filter was one of the biggest tasks to implement, so splitting up the functions between me and Drew greatly enhanced efficiency. Moreover, I was able to remain flexible and help Audrey debug her sensor model implementation before working on the particle filter which relied on a working sensor model.

From a technical standpoint, I learned how crucial it is to understand the entire algorithm before attempting to code, even when I may not be responsible for implementing certain functions. The particle filter relied on the sensor and motion models to function, so having a fundamental understanding of how they work made implementing the particle filter significantly easier. Moreover, I learned how we can add "randomness" in our algorithm to account for uncertainties within our hardware such as drift while driving straight, noisy odometry data, and more. Overall, this lab helped me understand how to use Monte Carlo Localization to estimate our robot's pose on a given map, which was interesting to see.

5.4 Drew Tufto

5.5 Richard Yeboah

Having just joined this team for this lab, it has been an incredible mountain to climb meshing into a team that worked so well together, but I think I have made a lot of progress in doing so. My team members have been very open about helping me with questions I've had about existing code, new code, and division of work.

On the technical side, I mostly helped out with the conceptual side of code because I felt like that was my main point of understanding. I feel like I could've stepped out of my comfort zone by helping implement more code directly, and that's something I'm currently working toward in the next lab. I credit my teammates once again for being so adept with the code that the journey from concept to code was quite easy.