Lab 5 Report: Implementing Monte-Carlo Localization

Team 13

Xavier Sanchez (2A)
Insuh Na (16)
Inimai Subramanian (6-2)
Russell Perez (6-2)

RSS

April 12, 2025

1 Introduction

In Lab 5, we aimed to build a robust algorithm for localizing our racecar's position in a known environment. Given an initial pose, subsequent poses can theoretically be constructed deterministically, by integrating the vehicle trajectory at each timestep. However, when using hardware we need to account for some unknown distribution of random noise. Because of that noise, the deterministic calculation quickly diverges from the ground truth car pose. We solve this problem with a 'particle filter', an algorithm that uses random sampling and environmental validation to converge a distribution of particles to the car's true position. Looking back to Labs 3 and 4, we have so far experimented with specialized path following algorithms, which used a wall or line as our predefined path. This lab's focus on localization allows for robust global positioning, a necessary prerequisite for generalized path planning and following in Lab 6. Operationally, the particle filter involves three distinct components: a motion model to update each of the car's estimated positions, a sensor model to determine the likelihood of each estimate, and a resampling scheme to shift the estimated distribution towards more likely poses. After implementing the particle filter we evaluated its effectiveness in simulation using real-time ground truth data, and then on the real racecar with sim-to-real checkpoints.

2 Technical Approach

2.1 Motion Model

The motion model takes input from the robot's odometry to calculate the average pose of the robot. Dead-reckoning (integrating the drive commands of the robot over time) provides odometry to estimate the distance and direction the robot has travelled within a certain time period.

At each timestep t, we have the car's initial world frame pose r_{k-1}^{W}, and its instantaneous velocity v^{k-1} with respect to that pose. Integrated over the timestep Δt, that velocity becomes the relative displacement $(\Delta r)^{k-1}$, termed the **relative odometry**:

$$(\Delta r)^{k-1} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{pmatrix}^{k-1} \tag{1}$$

The relative odometry is rotated to the world frame, then added to the initial pose to find the pose at the subsequent timestep $t+1$.

$$r_k^W = r_{k-1}^W + \mathbf{C}_{k-1}^W * (\Delta r)^{k-1} \tag{2}$$

In our motion model, we initialize many particles at $t = 0$ which together form a stochastic distribution describing possible locations for the car's true position. At each timestep, the above position update is calculated for *each particle* using the current relative odometry. To handle many particles in real-time, our code uses *NumPy* operations to perform simultaneous matrix multiplication between homogeneous transforms representing each particle and the relative odometry. Each particle is updated as follows:

$$\begin{bmatrix} \mathbf{C_k^W} & \begin{pmatrix} x \\ y \end{pmatrix}_k^W \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{C_{k-1}^W} & \begin{pmatrix} x \\ y \end{pmatrix}_{k-1}^W \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{C_k^{k-1}} & \begin{pmatrix} x \\ y \end{pmatrix}_k^{k-1} \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix} \tag{3}$$

2.2 Sensor Model

In order to decide which particles in our distribution best represent the car's position, which will be important for updating our distribution towards convergence, we need some way to validate the particles with respect to what we know about the local environment. Using our environment map, we can calculate what each particle "sees" relative to its pose, and compare that to what the car should be seeing from its pose, given by the current LIDAR scan. Using a function provided by the TAs, a ray tracing algorithm outputs a pseudo-LIDAR scan originating at each particle.

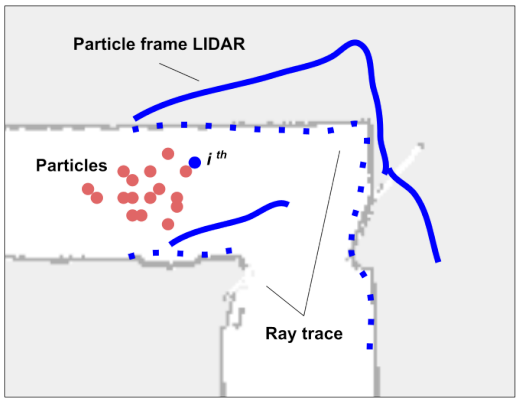

Figure 1: Each particle's ray traced environment should be compared to the LIDAR scan at its pose

We constructed a sensor model to determine how likely it is for a single ray traced scan to be a noisy version of the corresponding LIDAR scan from the car. Mathematically, the sensor model is a probability distribution constructed from a series of heuristics about the noise the LIDAR scan could introduce. For a single particle of interest, one axis represents a LIDAR measurement, and the second axis to the corresonding ray traced distance.
The four components of our sensor model:

$$p_{hit} = \begin{cases} \frac{2}{d} \cdot \left(1 - \frac{z_k}{d}\right) & \text{if } 0 \leq z_k \leq z_{max} \\ 0 & \text{else} \end{cases} \tag{4}$$

Assumes Gaussian noise around the true value of the LIDAR scan

$$p_{short} = \begin{cases} \frac{2}{d} \cdot \left(1 - \frac{z_k}{d}\right) & \text{if } 0 < z_k \leq d \\ 0 & \text{else} \end{cases} \tag{5}$$

Assumption of linearly increasing noise as scan distance approaches 0, from unexpected occlusions. The increasing probability comes from increasing area as radius increases from point in 2D space

$$p_{max} = \begin{cases} \frac{1}{\epsilon} & \text{if } z_k = z_{max} \\ 0 & \text{else} \end{cases} \tag{6}$$

Assumes high noise at maximal scan distance, from LIDAR not de-

tected at the receiver due to high absorption

$$p_{rand} = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_k \leq z_{max} \\ 0 & \text{else} \end{cases} \tag{7}$$

Assumes constant, low noise from disturbances otherwise unaccounted for

$$p_{total} = p_{hit} + p_{short} + p_{max} + p_{rand} \tag{8}$$

The total probability is the sum of these components

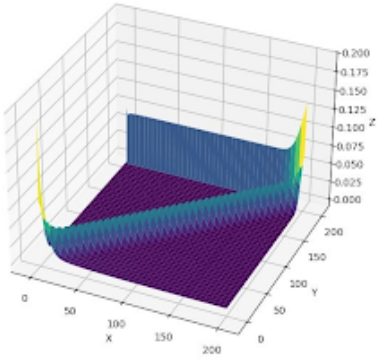

Figure 2: Sensor Model: Noise Probability Distribution

Particles are associated with multiple probabilities, one for each pair of LIDAR measurements and ray traced distances. All of these independently sampled probabilities p^i_{total} are then multiplied to find an overall likelihood for that particle. The sensor model outputs such a probability for each particles in our position distribution.

$$p_i = \prod_{z_k,d} p_{\text{total}}^{z_k,d} \quad \forall p_i \in P \tag{9}$$

2.3 Particle Filter - Bringing everything together

Now that the particle distribution has been updated with odometry (motion model), and the probability found for each particle being the ground truth pose (sensor model), we can improve the degree to which the distribution as a whole represents the ground truth pose. This is done through resampling. New particles are randomly selected from the post-motion model distribution according to the sensor model probabilities. The total number of particles does

not change, and 2.5% of particles are kept from the initial distribution. This 2.5% ensures that the particle filter does not filter out all of the low likelihood poses, ensuring that the estimation can use new data to recover from a false convergence.

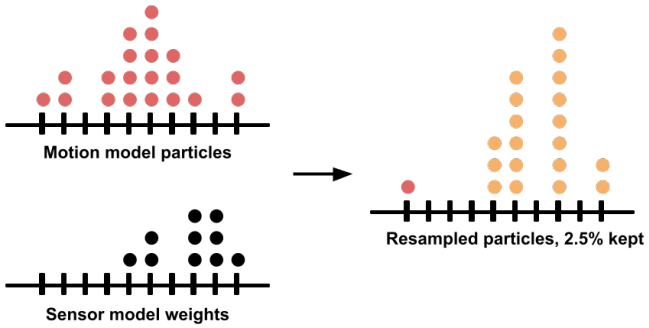

Figure 3: Simplified 1D depiction of particle resampling

After resampling, the particle filter publishes its current estimate for the race-car's pose, taken as an 'average' over the pose distribution at that time step. Our 'averaging' procedure is a simple average over X and Y, and a simple 'circular' average over θ.

$$x_{\text{avg}} = \frac{1}{N}\sum_i x_i, \quad y_{\text{avg}} = \frac{1}{N}\sum_i y_i, \quad \theta_{\text{avg}} = \tan^{-1}\left(\frac{\sum_i \sin\theta_i}{\sum_i \cos\theta_i}\right) \tag{10}$$

We chose this definition for effectiveness, interpretability, and simplicity. For unimodal distributions, higher particle density in a given location gives more weight to the pose estimate, as expected. When dealing with a multimodal distribution, in most cases a simple average would translate the uncertainty of the particle filter to the final pose estimate. This makes it obvious when the algorithm is unable to handle an environment.

3 Experimental Evaluation

3.1 Simulation - Russell

Our initial evaluation involved testing the motion and sensor models with unit tests, all of which passed. After developing the particle filter, we validated its

performance in software using various testing procedures.

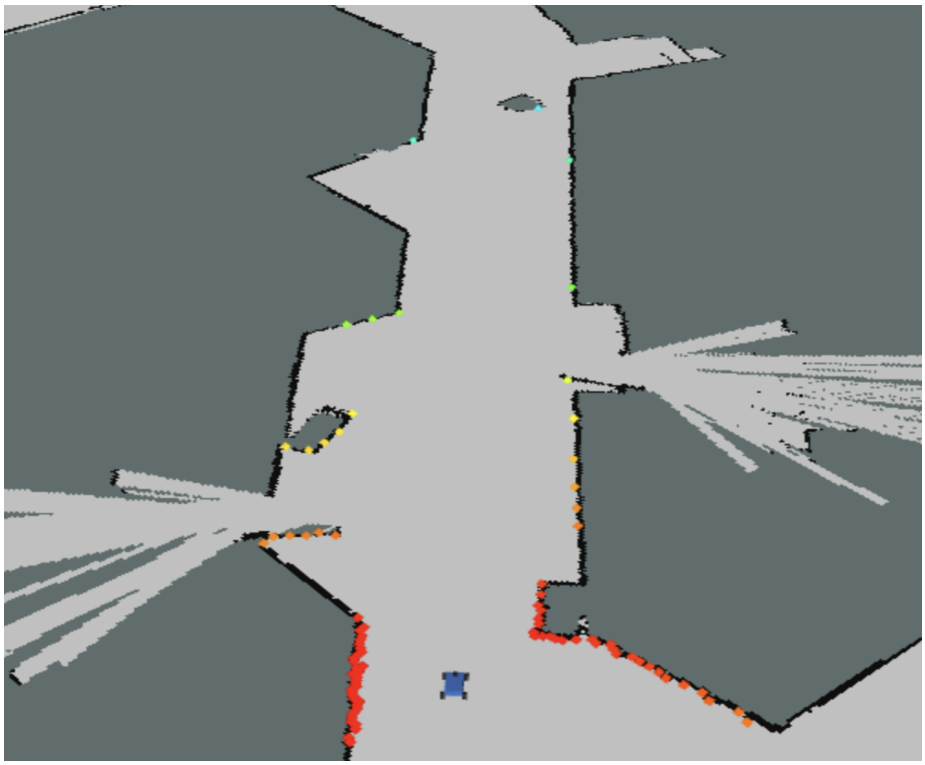

Figure 4: Feature-rich hallway right outside the lab

To evaluate performance under varying noise levels, we recorded a ROSbag of the car driving through the feature-rich hallway outside the lab using our wall-following implementation (Figure 4). This environment was ideal for localization due to the abundance of visual features. Recording a ROSbag allowed for repeatable tests and controlled conditions. We varied only the noise levels in dx and dy, keeping dt constant at $\frac{\pi}{15}$, and ensured equal noise values for dx and dy.
To evaluate performance, we defined two key metrics:

$$\text{distance error} = \sqrt{(x_{\text{ground truth}} - x_{\text{average pose}})^2 + (y_{\text{ground truth}} - y_{\text{average pose}})^2} \tag{11}$$

$$\text{angle error} = ((t_{\text{ground truth}} - t_{\text{average pose}} + \pi) \mod 2\pi) - \pi \tag{12}$$

Here, t represents orientation relative to the world frame. Both errors were published in real time and visualized using rqt. We also computed the overall

average of these metrics across the run. These final average values served as key performance indicators.
During testing, the 2D pose estimate was manually initialized in RViz to start the particle filter.

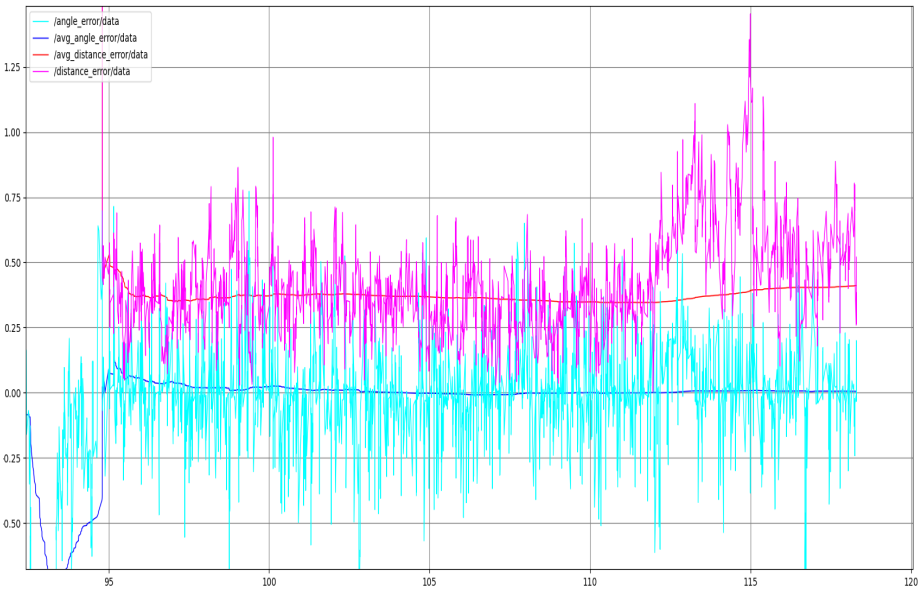

Figure 5: Simulation run in feature-rich hallway using noise levels $(0.1, 0.1, \frac{\pi}{15})$

With noise levels of $(0.1, 0.1, \frac{\pi}{15})$ in the x, y, and theta directions respectively, the filter showed consistent error trends. As seen in Figure 5, the average angle error converged toward zero, while the distance error stabilized around 0.375 meters. We repeated this with other noise levels:

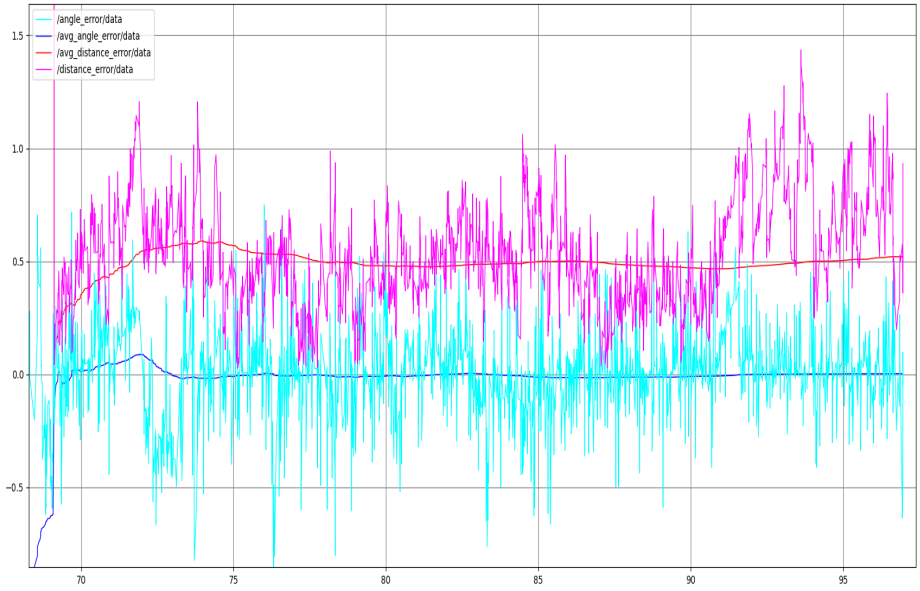

Figure 6: Simulation run in feature-rich hallway using noise levels $(0.2, 0.2, \frac{\pi}{15})$

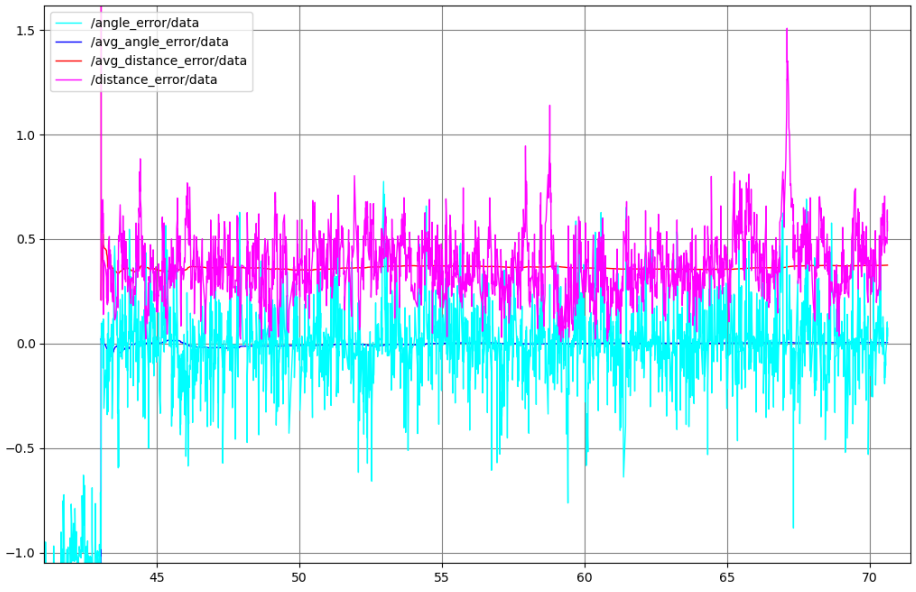

Figure 7: Simulation run in feature-rich hallway using noise levels $(0.4, 0.4, \frac{\pi}{15})$

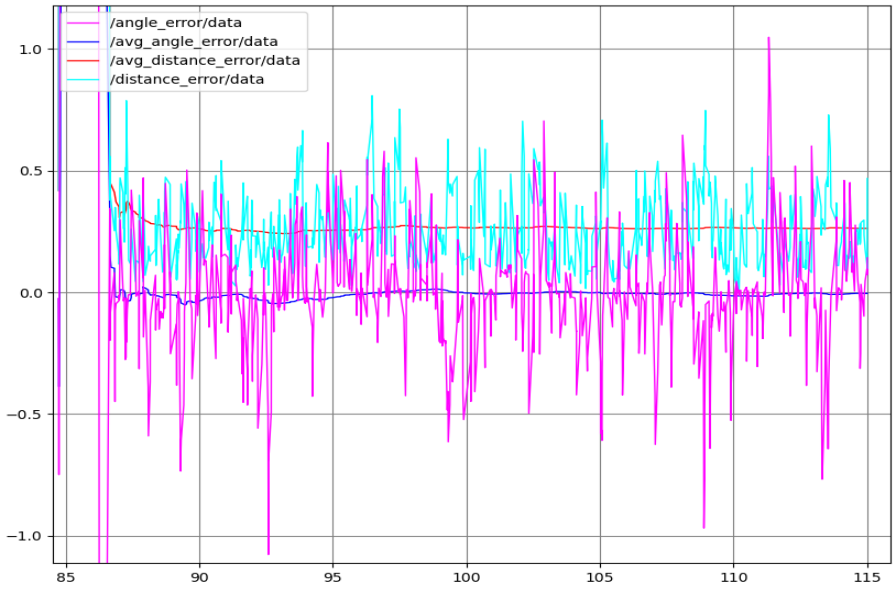

Figure 8: Simulation run in feature-rich hallway using noise levels $(0.8, 0.8, \frac{\pi}{15})$

Noise Level	Average Distance Error (meters)	Average Angle Error (radians)
(0.1, 0.1, $\frac{\pi}{15}$)	0.41	0.0056
(0.2, 0.2, $\frac{\pi}{15}$)	0.52	0.0026
(0.4, 0.4, $\frac{\pi}{15}$)	0.38	0.0051
(0.8, 0.8, $\frac{\pi}{15}$)	0.26	0.0043

Table 1: Resulting average error values from differing noise levels on feature-rich hallway

Across all runs in the feature-rich hallway, average distance errors stayed within a narrow range, averaging 0.39 meters. Average angle error was 0.0044 radians (0.25°), indicating consistent orientation accuracy.

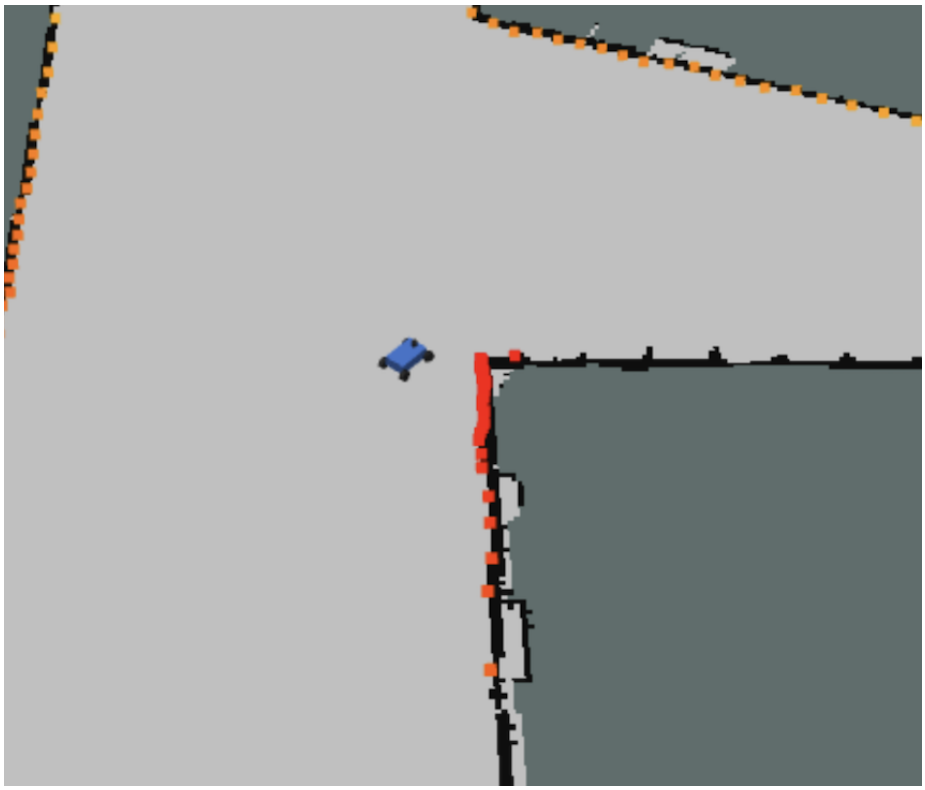

Figure 9: Turn from a feature-less hallway

We repeated the same tests on a different ROSbag. Again, only noise levels were changed. This time, we wanted to see how well the particle filter could handle a turn in a feature-less hallway as in Figure 9.

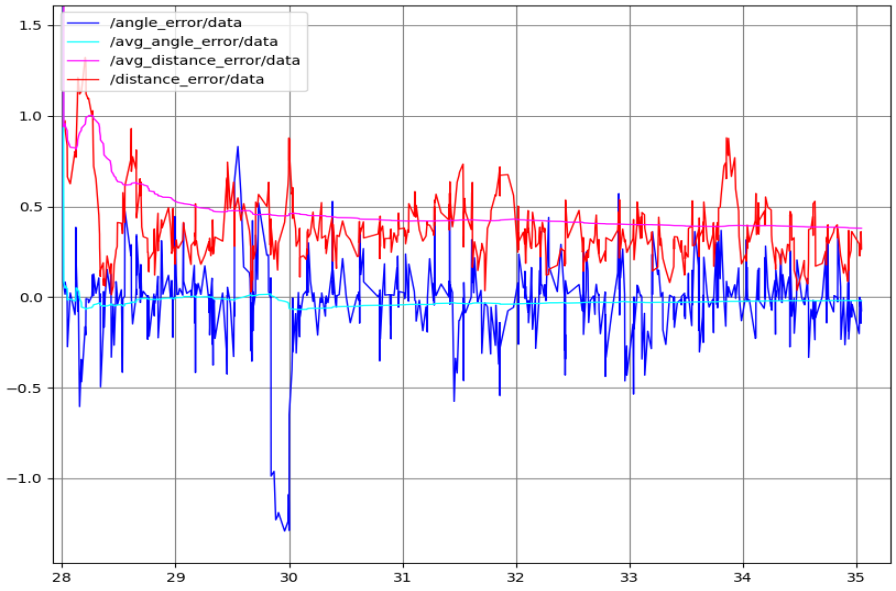

Figure 10: Simulation run in turn from feature-less hallway using noise levels $(0.1, 0.1, \frac{\pi}{15})$

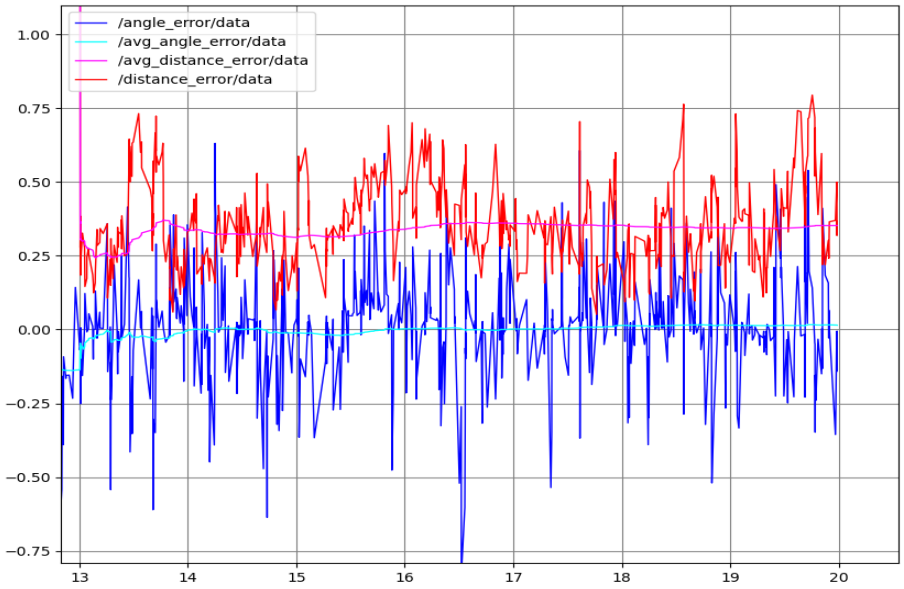

Figure 11: Simulation run in turn from feature-less hallway using noise levels $(0.2, 0.2, \frac{\pi}{15})$

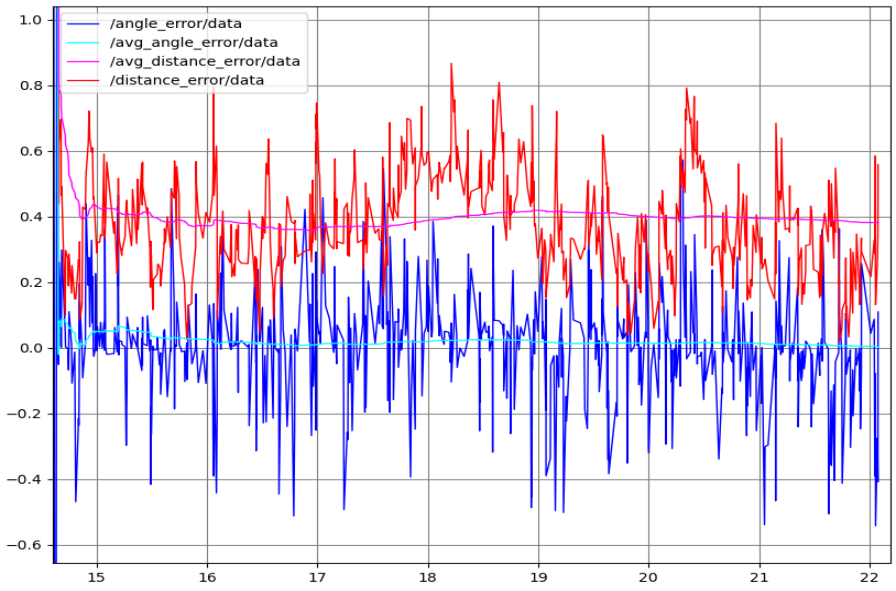

Figure 12: Simulation run in turn from feature-less hallway using noise levels $(0.4, 0.4, \frac{\pi}{15})$

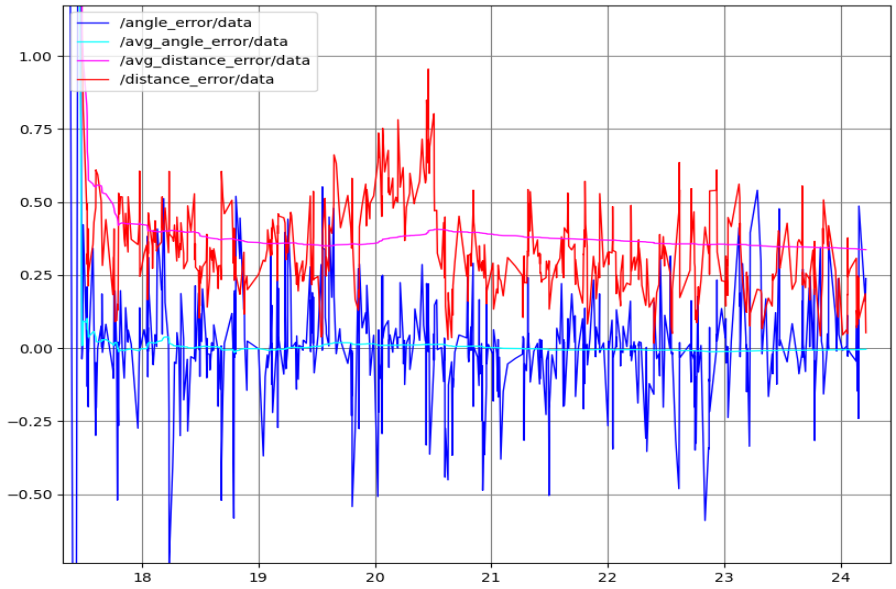

Figure 13: Simulation run in turn from feature-less hallway using noise levels $(0.8, 0.8, \frac{\pi}{15})$

Noise Level	Average Distance Error (meters)	Average Angle Error (radians)
$(0.1, 0.1, \frac{\pi}{15})$	0.38	0.02
$(0.2, 0.2, \frac{\pi}{15})$	0.35	0.015
$(0.4, 0.4, \frac{\pi}{15})$	0.38	0.0012
$(0.8, 0.8, \frac{\pi}{15})$	0.34	0.004

Table 2: Resulting average error values from differing noise levels on turn from feature-less hallway

Despite fewer landmarks and a turn, the filter remained stable. The average distance error was 0.36 meters, and average angle error was 0.01 radians (0.57°). These experiments demonstrate the particle filter's reliability across noise levels and environments. Errors stayed low and consistent, indicating strong localization performance even under degraded sensing conditions.

3.2 Racecar - Inimai

Testing the system implementation on the racecar involved evaluating the localization algorithm both statically and dynamically (with teleop maneuvers).

3.2.1 Qualitative Evaluation

We tested the localization algorithm by placing and driving the car under various conditions in the Stata basement.

Static Localization. The car was placed in front of a column in the feature-rich hallway, in the sparse hallway (bike storage), and at the corner between them. Localization was accurate in feature-rich areas but consistently off by about 1meter in the x-direction in the sparse hallway due to a lack of distinguishing features.

Dynamic Localization. We drove the car through routes including the T-junction, the corner, and the sparse hallway, while monitoring the particle filter and average pose in RViz. The pose tracked well in rich areas but drifted in sparse ones. The algorithm remained responsive even during zig-zag paths and high-speed motion, showing the robustness of the motion model.

3.2.2 Quantitative Evaluation

Since ground-truth odometry wasn't available, we used static localization for benchmarking. We selected three checkpoints across different feature environments and recorded their positions in RViz using the Measure tool.

Checkpoint 1 was in a feature-rich hallway, Checkpoint 2 at a T-junction, and Checkpoint 3 in a sparse hallway. They are representative of unique scenarios in our map. We measure their real-life coordinates to compare with our simulation.

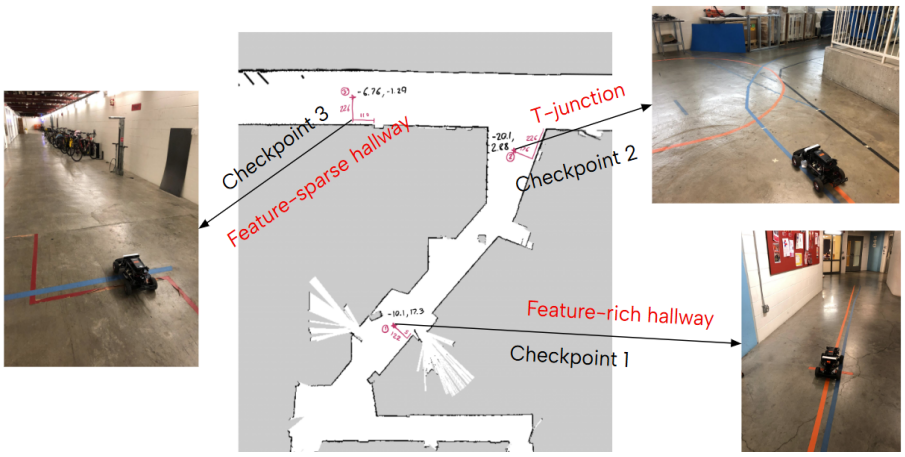

Figure 14: The three checkpoints used for static localization evaluation placed on the map and shown in real-life.

After obtaining coordinates for the true and localized positions, we calculated

the straight-line distance between them. Table 3 shows the raw data and error per trial. Multiple trials were conducted to ensure consistency.

Checkpoint	Checkpoint Type	Real (x,y) coordinates (m)	Localized (x,y) coordinates (m)	Distance Error (m)
1	Feature-rich Hallway	(0.05, 1.22)	(0, 1.35)	0.14
2	T-junction	(2.26, 1.26)	(2.09, 1.05)	0.27
3	Feature-sparse Hallway	(1.10, 2.26)	(2.75, 2.25)	1.65

Table 3: Resulting distance error values for each checkpoint. Distance error increases with decreasing features available.

The localization algorithm performed well in areas with sufficient features (Checkpoints 1 and 2), showing minimal distance error. At Checkpoint 3, the lack of features led to an x-direction error of 1.65m, though the y-coordinate remained accurate.

4 Conclusion - Inimai

Our team was able to build a robust localization algorithm implementing Monte-Carlo localization. The primary components of this system include the motion model (which updates the current particles' poses) and the sensor model (which computes the likelihoods for each particle to be the ground-truth position as weights). These come together for the particle filter, which uses these models to filter and update the racecar's average pose.

Through extensive testing and implementation, our motion model is resilient to noisy observations from odometry on the real racecar. Our sensor model is also able to run efficiently through the use of a pre-computed probability table and numpy array improvements that optimized our methods. This allows us to achieve low distance and angle error between the ground truth and average pose for simulation, and the checkpoint verification for the real car. Our system is also able to respond quickly to noise, reliably converge to the car's actual location, and track the car's position reliably while driving.

For the future, we hope to improve our implementation by adding further optimizations to the code to improve its runtime speed, improving initialization capabilities in feature-sparse regions, and explore new locations / maps for better generalization. We aim to use these methods paired with path-planning methods in the next lab.

5 Lessons Learned

5.1 Xavier

From a technical perspective, I learned that thinking through an algorithm is one thing, but that programming something that works is another thing entirely, not to mention writing efficient code. From this perspective, it is very important to leverage the strengths of your teammates *and of yourself*. It is important to understand how things are working, and to learn how to do something for yourself, but there are diminishing returns on struggling through something that a teammate could do in their sleep instead of using your skills on another branch of the project. Communication is an integral part of this, because teammates need be willing to understand each other's strengths, and how to structure the project so that each person's skills are fully leveraged. Additionally, a reflection after implementation on *how* something was done and *why* an initial attempt didn't work works in the learning while still allowing for quick and robust progress. Lastly, I learned that sometimes just being there to support your team is important, even when there is not much you are physically assigned to do. Being there to offer a helping hand, bounce ideas off one-another, and support the team's morale towards reaching the goal is another important part of what it means to be a team.

5.2 Russell

One important lesson I learned from this lab is the value of starting early. We had the opportunity to begin work before Spring Break and continue over the break, but I didn't take full advantage of that extra time. This ultimately made the later stages more rushed than they needed to be. Additionally, getting sick the Monday after Spring Break caused me to miss key communications and updates, and it took a while to catch up. This experience reinforced how important it is to be present and up to date, especially in collaborative projects.

I also learned the importance of fully understanding the goals of the lab before diving into the work. Our group spent a significant amount of time uncertain about how to proceed with experiments on the Racecar. In hindsight, taking more time early on to clarify what data we needed to collect and how to do so would have saved us time and confusion down the line.

One technical skill I improved on was collecting experimental data in simulation using ROSbags. This was a valuable experience, and I know this technique will be useful in future labs. I'm excited to continue building on these skills as the course progresses.

5.3 Insuh

I learned that it is important to take a break and plan when things aren't working out. The action of organizing things that are going to be done can help progress quite a bit.

5.4 Inimai

Technical: From this lab, I deepened my understanding of probabilistic robotics, particularly through the implementation of Monte-Carlo Localization (MCL). I learned how to design and tune a motion model that accounts for real-world odometry noise and how to construct an efficient sensor model that uses precomputed probabilities to evaluate particle likelihoods. Working with numpy and optimizing array-based operations taught me the importance of computational efficiency in real-time systems. Overall, the lab reinforced how algorithmic design, optimization, and validation come together to build robust, real-time localization systems.

CI: This lab emphasized the importance of clear, precise communication within a technical team. Collaborating with teammates on dividing tasks like motion and sensor modeling required consistent updates, shared documentation, and frequent check-ins to ensure alignment and integration of our modules. Presenting our findings to the staff helped me practice distilling complex technical information—like the workings of our particle filter—into accessible explanations.