Lab 5 Report: Monte-carlo Localization for Autonomous Race Car

Team 4

Adelene Chan
Alan Chen
Paul Gregory
Eghosa Ohenhen

6.4200 Robotics: Science Systems

April 7, 2025

1 Introduction (Paul)

The focus of this lab was to develop a probabilistic localization system for a miniature autonomous racecar operating in a known map. Localization, or estimating a robot's position and orientation within its environment, is a fundamental challenge in robotics. The ability to localize is a crucial step toward enabling autonomous navigation, forming the basis for path planning and decision-making.

In this lab, we implemented Monte Carlo Localization (MCL), also known as a particle filter, to estimate the racecar's pose using LiDAR data and a known map. The MCL algorithm maintains a set of particles, or hypotheses, representing potential locations of the robot, and continuously updates them using probabilistic motion models and sensor measurements. Over time, the filter converges toward the true position, even in the presence of noise. This lab builds directly on the foundational tools and concepts from earlier labs. We extended our experience using data from the racecar and ROS-based programming. The challenge shifted from reactive control to accurate state estimation under uncertainty, reflecting the increasing complexity of real-world robotics tasks.

We encountered a number of practical challenges, such as dealing with noise, transitioning from the perfect simulation world to the imperfect world of the physical racecar, ensuring stable convergence, and dealing with software dependency issues between different group member's implementations.

Knowing where the racecar is in its environment is a key piece of having an autonomous vehicle. Moving forward, we will build on the knowledge of where the robot is, transitioning to allowing it to get where it needs to go.

2 Technical Approach

Developing the particle filter for localization involves incorporating two parts: a motion model and sensor model. Starting with a spread of initial guesses as to where the robot is located, the motion model can use the wheel odometry to determine the where the next pose will be. The particle filter takes this new set of particles and identifies how probable each particle is based on the current LiDAR scan data and resamples the particles to create a new spread of particles with higher likelihood of being the correct pose.

2.1 **Motion Model** (Adelene)

As the racecar moves, we take into account the wheel odometry messages which include translational velocities in the x and y directions, the angular velocity in the z direction, and the time difference between the last and current messages. This allows us to integrate each velocity to obtain a displacement pose, Eq. 1, with respect to the previous robot frame, $k-1$. We had two implementations, one that applied this simpler independent integration and a second which followed the arc length, given that the translational motion is more realistically coupled with the angular motion. However, with sufficiently small time steps, independent integration for displacements is likely enough for an effective motion model.

$$\mathbf{\Delta x} = \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} \tag{1}$$

Since each particle is a previous pose $k-1$ with respect to the map frame, we can convert the poses into transforms and compose the transforms to give us the transform from the map to the current k robot frame.
Wheel odometry will not be completely representative of the actual motion of the racecar due to noise, friction, tilting, and additional physical disturbances, so we need to add noise to the term $\mathbf{\Delta x}$ in order to provide more guesses as to what the racecar pose is after accounting for noise. We chose a Gaussian distribution for dx, dy, dz respectively
The transform is applied to each of the current particles $\mathbf{T}_k^W = \mathbf{T}_{k-1}^W \cdot \mathbf{T}_k^{k-1}$. Eq. 2 shows what transform composition for a single previous pose k and a Δx with noise.

$$\mathbf{T}_k^W = \begin{bmatrix} cos(\theta_{k-1}) & -sin(\theta_{k-1}) & x_{k-1} \\ sin(\theta_{k-1}) & cos(\theta_{k-1}) & y_{k-1} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta_{\Delta x}) & -sin(\theta_{\Delta x}) & x_{\Delta x} \\ sin(\theta_{\Delta x}) & cos(\theta_{\Delta x}) & y_{\Delta x} \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The resulting transforms are then converted into poses of the form shown in Eq. 3

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} \tag{3}$$

We return the list of these poses which present a new set of particle guesses of the new robot poses.
With the addition of noise, which is necessary to capture the imperfect translation of wheel odometry to exact pose with real world disturbances, the motion model will lead to a divergence of particle poses. This leads to the sensor model which will take in these particles and return a set of more likely particles. This will help the particles to converge to a pose estimate.

2.2 Particle Filter (Alan)

In our particle filter, we maintain a prior of the current pose. Then, if we make an observation O, we update the prior using Bayesian inference.

We represent an approximate prior, parameterized by a set of poses P_i (which we will call particles), each with equal weight. We treat each P_i as a vector (x, y, θ). Each of those represents a Gaussian with a fixed mean and covariance. Our Prior is then

$$prior = \frac{1}{n} \sum_i \mathcal{N}\left(P_i, Cov_{fixed}\right)$$

where Cov_{fixed} is chosen to be a constant, to make the prior somewhat smooth:

$$Cov_{fixed} = \begin{pmatrix} 5\text{cm} & 0 & 0 \\ 0 & 5\text{cm} & 0 \\ 0 & 0 & 5\text{degrees} \end{pmatrix}^2$$

We estimate the posterior contributed by the part $\mathcal{N}\left(P_i, Cov_{fixed}\right)$ with a single likelihood number, using the average likelihood across all points in $\mathcal{N}\left(P_i, Cov_{fixed}\right)$:

$$likelihood(P_i, O^*) = E_{P^* \sim \mathcal{N}(P_i, Cov_{fixed})} P_{O \sim Obs_{P^*}}(O = O^*)$$

We describe how we compute an estimate of this function in section 2.2.1.

We use this as the likelihood for all points in $\mathcal{N}\left(P_i, Cov_{fixed}\right)$. This gives us a approximate posterior which is a weighted mixture of Gaussian.

To use the posterior as the next inference step, we need to change it to the prior format which has uniform weights. We do this by sampling each P_i' independently from the posterior. Note that we would still need to resample if we have weights in our prior, to prevent very low weight parts that contributes little to modeling the distribution.

Note that we assume the next data we accept is independent of the previous one. This is a highly flawed assumption; in particular, if the racecar is still, The same sensor data would be used multiple times, causing the algorithm to be overconfident in one location.

2.2.1 Estimating the likelihood

Let z be the ground truth pose, and let the observation be drawn from $O \sim Obs_z$. Let Z be the global distribution of z.

Let G be the Gaussian

$$\mathcal{N}(0, Cov_{fixed})$$

We would like to estimate the likelihood function, given by

$$\begin{aligned} & likelihood(P_i, O^*) \\ =& E_{P^* \sim \mathcal{N}(P_i, Cov_{fixed})} P_{O \sim Obs_{P^*}}(O = O^*) \\ =& E_{x \sim G} P_{O \sim Obs_{P_i + x}}(O = O^*) \end{aligned}$$

We estimate *likelihood* by mapping P_i into a feature space $feature(P_i)$ in F, and approximate $likelihood(P_i, O^*)$ as if we only know $feature(P_i)$:

$$likelihood(P_i, O^*) \approx likelihood^*(feature(P_i), O^*)$$

we want $likelihood^*$ to best approximate *likelihood*. A good candidate to use is the global average likelihood, conditioned on $feature(P_i)$:

$$\begin{aligned} & likelihood^*(f, O^*) \\ =& E_{x \sim G, P^\dagger \sim P | feature(P^\dagger) = f} \left[P_{O \sim Obs_{P^\dagger + x}}(O = O^*) \right] \\ =& P_{x \sim G, P^\dagger \sim P, O \sim Obs_{P^\dagger + x}}(O = O^* | feature(P^\dagger) = f) \end{aligned}$$

The overall distribution of $P^\dagger + x$ is the mixture of all priors, which we approximate by the ground truth distribution Z.

Rewriting with $z = P^\dagger + x$ gives

$$likelihood^*(f, O^*) \approx P_{x \sim G, z \sim Z, O \sim Obs_z}(O = O^* | feature(z - x) = f)$$

If we have observation samples labeled with ground truth (which we have in the sim):

$$(z, x, O, feature(z - x))$$

Then, with a small categorical feature space, we can estimate this directly.

On the real racecar, since we have no ground truth z to use, we can use a localization model (such as using a particle filter based on sim data, and then filtering it through manual inspection) to estimate z. We can then compute this quantity assuming that those estimates are actually the ground truth.

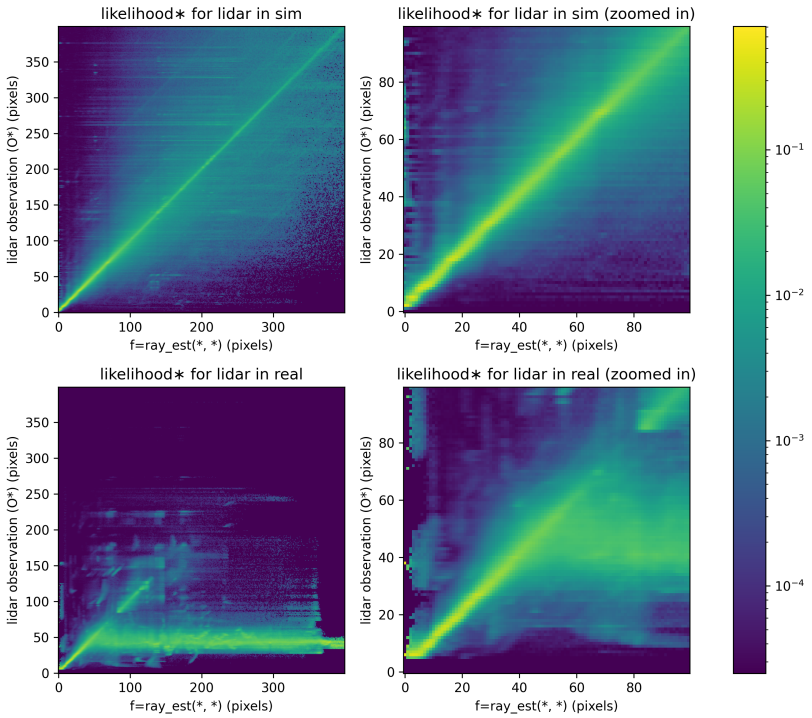

Figure 1: Plots of $likelihood^*$ computed from data. For the real racecar, ground truth comes from accepting points from the sim localization model if lidar data likelihood is above a certain threshold. For data used on real racecar here, the lidar is angled downwards, causing it to only see about 3 meters (60 pixels) in front.

2.3 Sensor Model (Alan)

To make use of lidar observations, we use the method in section 2.2.

For a single observation at angle θ, we compute the likelihood using the feature $feature(P_i) = ray\_est(P_i, \theta)$ where $ray\_est(P_i, \theta)$ is the distance of an approximate ray tracing from P_i and θ.

We make use of multiple observations from different θ by taking the **geometric mean** of the likelihood obtained by each individual θ. We find that the observations are highly dependent, so we could not have multiplied the probabilities.

Examples of $likelihood^*$ from sensor data is shown in fig. 1.

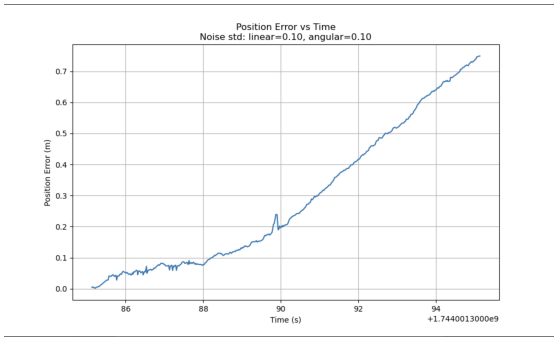

Figure 2: Our faulty error plot when gaussian noise with standard deviation of .1 is added to our odometry data.

3 Experimental Evaluation

3.1 Simulated Racecar Evaluation - Noisy Odometry (Paul)

In a real-world robotics system, sensor data is rarely perfect. While simulations often provide idealized, noise-free data, robust localization algorithms must be able to handle noisy and uncertain inputs in order to remain robust and accurate when transitioned to the real world. To test the resilience of our Monte Carlo Localization (MCL) system under realistic conditions, we evaluated its performance using simulated odometry data corrupted with Gaussian noise. By systematically varying the magnitude of noise applied to both linear and angular velocity measurements, we observed how well the particle filter could maintain accurate pose estimates in the presence of uncertainty. This evaluation serves to approximate the types of errors encountered on the physical racecar and provides insight into the limitations of our filter, as well as opportunities for improvement. By comparing the estimated pose against ground truth location, we can quantify the error introduced by noisy odometry and assess the effectiveness of our current motion and sensor models.

To obtain this data, we create a new ROS node that inputs the odometry, adds gaussian noise at a specified standard deviation, and publishes this noisy odometry to be used by the particle filter. This node then captures an error over time plot for 10 seconds while our wall follower is running on the simulation racecar.

Unfortunately, we ran into an error when attempting to capture these plots for our final particle filter, as software/hardware requirements made it difficult to run this implementation node with our particle filter node, and we did not plan enough time to debug this issue. Moving forward, we have already worked towards simplifying our particle filter to resolve some of these requirements. We include one of these plots in 2; it diverges as the particle filter node does not run properly with the evaluation node.

We also include plots captured on an earlier version of our particle filter in 3

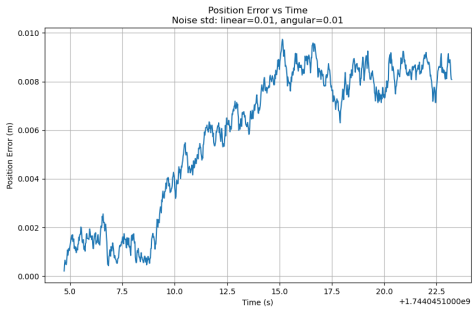

Figure 3: Our error plot when gaussian noise with standard deviation of .01 is added to our odometry data.

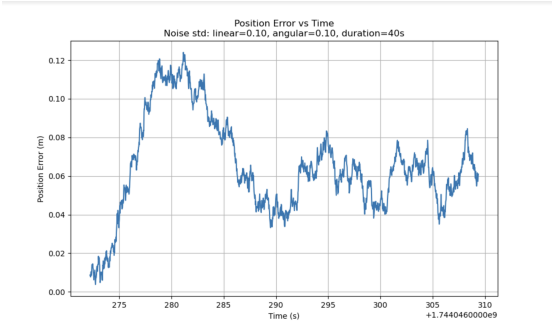

Figure 4: Our error plot when gaussian noise with standard deviation of .1 is added to our odometry data.

and 4, showing how our evaluation works. As you can see, at higher levels of standard deviation, the particle filter becomes worse at estimating the car's location.

3.2 Simulated Racecar Evaluation - Convergence (Eghosa)

Convergence of the pose estimation is an important metric that signifies how confident the controller is about the pose of the robot at that time t. Naturally, a fast convergence plot with minimal spikes afterwards is desired. To measure the convergence of the pose estimation particles, we measured the standard deviation of the particles while running localization on a wall follower. All of the convergence plots were generated while following a straight line. To show the robustness of the pose convergence, we plotted the convergence of the robot in 3 states: no noise, 0.01 gaussian noise (Figure 5) and 0.1 noise (Figure **??**). As expected, the more gaussian noise that was added, the more variance that appeared in the standard deviation of the pose estimate, even when it had

converged. Due to some issues with our racecar sensor model, the convergence plots diverged. This is something that later addressed after this first evaluation of the racecar's localization.

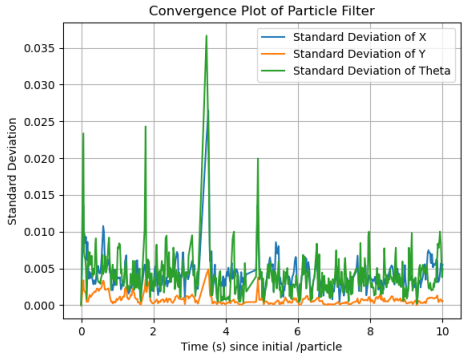

Figure 5: Convergence Plot in Simulator with 0.01 Gaussian Noise
The blue, orange, and green show the standard deviations of the spread of the x, y, and theta of the particles. The higher spikes indicate less certainty in the estimated pose. The small amount of noise led to some points of uncertainty but mostly certain pose estimate. This was while running the wall follower.

3.3 Real Racecar Evaluation (Adelene)

Moving from simulation to the real racecar involved accounting for several differences: the laser scan data having a greater number of beams, more noise in the wheel odometry, increased sensor noise, obstructions present in the scan, and other factors. While in simulation, we had access to the ground truth odometry which allowed for direct comparison with the pose estimate. However, the real racecar does not have a ground truth odometry as the objective is to estimate its pose. In order to identify correct localization, we compared the laser scan with the predicted laser scan as produced by ray tracing with the estimated pose. This allows us to check for accurate localization by seeing whether the two align. We acknowledge that this requires unique features in the map to compare with as it is difficult to differentiate sections of a similar looking hallway. Figure 7 shows the localization accuracy of a unique feature in the Stata basement where the actual laser scan lines up with the ray-traced scan of the estimated pose very closely. We note that there exists a line in front of the racecar which was a hardware issue with the LiDAR pointing towards the ground leading it to see a smaller range.

We also used convergence plots to see how quickly and how confident the racecar was at estimating its pose. This does not necessarily indicate localization accuracy as the algorithm can converge on an incorrect location. However, a faster

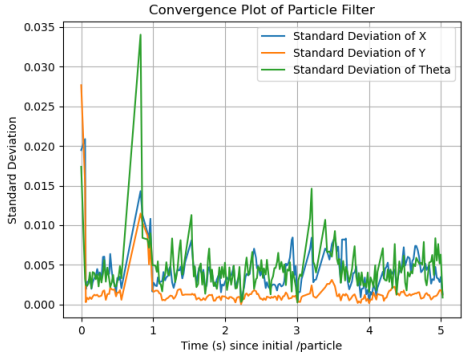

Figure 6: Convergence Plot in Simulator with 0.1 Gaussian Noise
The blue, orange, and green show the standard deviations of the spread of the x, y, and theta of the particles. Like the 0.01 noise spread, there were spikes of uncertainty but largely certain pose estimates over time. This was while running the wall follower.

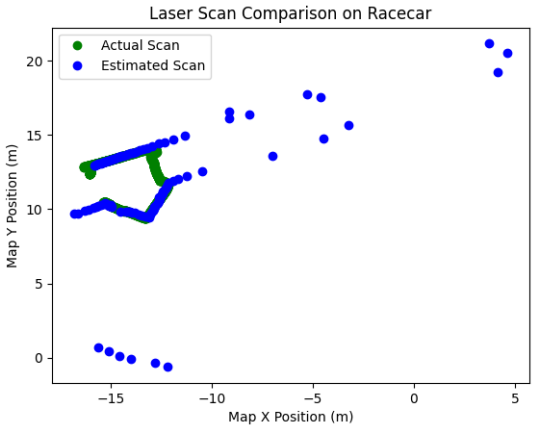

Figure 7: Laser Scan of Stata Basement, Outside Classroom
The blue shows the ray-traced estimated scan based on the estimated pose while the green shows the actual laser scan in the perspective of the estimated pose. The laser scan in green lines up well as indicated by the high overlapped regions.

convergence on the pose estimate allows for using this localization as a building block for other racecar algorithms which may require faster performance. Figure 8 shows that in a map area with a high number of unique features, the particle filter leads to higher confidence in its estimated pose as shown by the smaller standard deviations in the convergence plot.

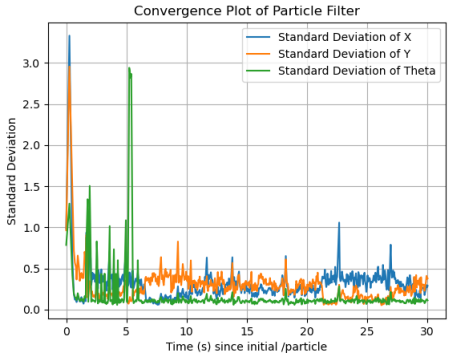

Figure 8: Convergence Plot of Stata Basement, Outside Classroom
The blue, orange, and green show the standard deviations of the spread of the x, y, and theta of the particles. The pose estimates converge fairly quickly but to a value higher than that of the simulation, in the 0 to 0.5 range. This makes sense given the greater amount of real world noise.

3.4 Discussion (Adelene)

Evaluating the particle filter was more complex than previous labs as it required analyzing different parts whether that was robustness to noise, confidence, and accuracy. Additionally, having multiple parts to the algorithm, the motion model, sensor model, and particle filter, added to complexity of identifying where something may have gone wrong. Therefore, it was important that we tested each part in isolation before putting them together. This heavier emphasis on evaluation led to spending more time on different analyses from simulation to the real racecar.
These tests helped us discover ways to improve our algorithm such as a convergence-divergence issue, efficiency issues, and also quantify and qualify how well this particle filter performed. The particle filters we developed successfully localize the racecar in the static map, meeting the system and lab requirements. However, we observed areas to improve, such as our algorithm's efficiency. Moving from the simulation to real racecar where ROS topics changed, real-world noise, and efficient code became even more important as there exists time delays and different levels of data being sent such as a larger set of scan data. Though we have a working particle filter, an improved particle filter will be more effi-

cient, effective in various environments, and execute at a faster rate to allow the racecar to localize while moving more quickly or in more similar feature areas.

4 Conclusion(Paul)

This lab focused on implementing Monte Carlo Localization to estimate the pose of a miniature autonomous racecar using odometry and LiDAR data. Localization is an important piece of making our racecar autonomous, as it must be able to accurately determine its position within its environment before making movement decisions.

To implement localization, we used a particle filter. We represented the robot's belief as a set of weighted hypotheses, or particles, each encoding a possible pose. At each timestep, we applied a motion model to propagate the particles based on odometry data, then updated their weights by evaluating how well the predicted LiDAR observations aligned with actual scan data. This alignment was computed using a ray tracing model, which estimated the expected range measurements from each particle's perspective. The result was a posterior distribution over the robot's pose that evolved over time, maintaining robustness in the presence of uncertainty.

We first validated this system in simulation, where ground truth data allowed us to evaluate localization accuracy directly. We measure estimation vs ground truth error at different levels of noise as well as plotting the convergence of our particle filter. On the real racecar, we adapted our system to handle more complex, noisier inputs, and validated localization indirectly by comparing actual LiDAR scans to predicted ones. We also again used convergence plots to assess the filter.

This lab demonstrated the effectiveness of our particle filter for localization, providing an important step towards autonomous navigation. In future labs, we will build on the ability to localize to implement path planning for our racecar.

5 Lessons Learned

(Adelene)

This lab showed me about why well-organized, documented, and accessible code is important in development stages. There exists a tricky balance of having something be cleanly abstracted and something that can still be debugged easily. I'm glad that our team went through some of the growing pains of figuring out what workflows work and don't work as well as setting some standards as to how we want to develop and test our code moving forward, whether that's developing evaluation metrics earlier or ensuring code compatibility between devices and the real racecar. I am grateful for having great teammates to lean on and learn from and am grateful we have different skillsets and perspectives to bring. I also learned about the importance of bringing up difficult conversations as that is necessary to process what is working and what is not working,

in order to grow as a team.
(Paul)
In this lab, I learned about the importance of having synergy between implementation and evaluation, as evaluation can be used for testing an implementation to make it better. I also learned about the importance of documenting code, as when working on a team it can sometimes be very difficult to understand what someone elses code is doing or how to run it.
(Eghosa)
I learned a lot about keeping in touch with a team this lab and also about the difficulty of migrating multiple products into one. I have worked on github repos with multiple people before, but usually we start from scratch, working together to implement an idea. However, for this lab, we have each came in with our own working implementations and side projects and had to integrate them together not only via Github, but also when sshed onto the robot's WiFi. Trying to push our code from the robot to Github was interesting since we were essentially on the same machine, so a lot of the usual abstractions used when collaborating with Github couldn't be used. This was a very valuable insight for me.
(Alan)
I learned that oftentimes software may not be fully cross-platform; It is important to make sure anything runs on all teammates computers first. I also learned that, in an ROS system, it is more clean to use a "ROS Node" as interface rather than Python functions or classes.