Lab 5 Report: Localization

Team 10

Foland, Lexi
Fortt, Julia
Kim, Evan
Krebs, AZ
Peale, Will
RSS

April 12, 2025

1 Introduction: AZ Krebs

Accurate localization - estimating a robot's position and orientation within a known environment - is a crucial ability for autonomous mobile robots in the real-world. From self-driving cars to warehouse robots, being able to figure out where it is in its environment is crucial for robots to complete safe and effective operation. However, achieving reliable localization is a challenging problem caused by noise from sensors and imperfect motion models. To tackle this challenge, we implemented the Monte Carlo Localization (MCL) algorithm. MCL provides a probabilistic framework for tracking a robot's pose by representing the belief state with a set of weighted particles. We implemented the algorithm in a 2D simulated environment where it was tested under various noise conditions to evaluate accuracy and convergence and then adapted the MCL implementation for the physical racecar and conducted experiments to evaluate the real-world performance.

2 Technical Approach: AZ Krebs

The implementation of MCL in this lab followed a modular approach based on probabilistic robotic principles. MCL operates by maintaining a belief distribution over a robot's pose using a finite set of weighted particles. These particles represent hypotheses about the robots location, which are iteratively updated through the motion and sensor models as new data becomes available. The overall process consists of three primary steps: sampling (motion model update), weighting (sensor model update), and resampling.

1: **Algorithm sample_motion_model_odometry**(u_t, x_{t-1})**:**

2: $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$

3: $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s}$

4: $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5: $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1 \delta_{\text{rot1}} + \alpha_2 \delta_{\text{trans}})$

6: $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3\ \delta_{\text{trans}} + \alpha_4(\delta_{\text{rot1}} + \delta_{\text{rot2}}))$

7: $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1 \delta_{\text{rot2}} + \alpha_2 \delta_{\text{trans}})$

8: $x' = x + \hat{\delta}_{\text{trans}}\ \cos(\theta + \hat{\delta}_{\text{rot1}})$

9: $y' = y + \hat{\delta}_{\text{trans}}\ \sin(\theta + \hat{\delta}_{\text{rot1}})$

10: $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

11: *return* $x_t = (x', y', \theta')^T$

Figure 1: An algorithm for computing the effect of odometry deltas on particles. The sample function allows random noise to be added, which both simulates real-world error and prevents early convergence to a specific particle.

The motion model predicts the robot's new state based on proprioceptive sensing. Odometry readings served as the control input which allows a prediction of the next pose of the robot based on its previously predicted state. To account for noise from the actuators and sensors as well as potential wheel slippage, zero-mean Gaussian noise was added to the prediction of the next state. This stochastic model spreads the particles appropriately and reflects the uncertainty in the prediction. This is also done to guarantee the model does not converge too quickly and allows full exploration. In a perfect world with no sensor noise or wheel slippage, there would still be noise to prevent this "too quick" convergence. In simulation, both deterministic and stochastic versions of the motion model were implemented, with the deterministic model being used for unit testing, while the stochastic model was used to simulate real-world localization.

The sensor model evaluates the likelihood of each particle's predicted observation given the actual sensor reading. Each actual laser scan was compared to what the predicted particles would measure as if they were at that location on the map in the real-world. These "simulated" laser scans for the particles were calculated by projecting scan endpoints into the coordinate frame of each particle and evaluating its proximity to the known obstacles in the map, also known as ray-casting. Rather than recomputing the probability of each particle at each time step, we developed a discretized precomputed table of probabilities,

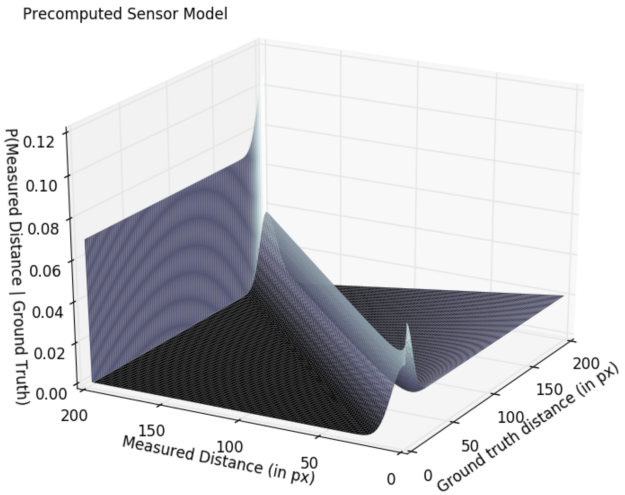

Figure 2: The likelihood surface representing the probabilities reflected in the precomputed sensor model table. Peaks correspond to higher probability regions. The surface is measured as an intersection between measured and ground truth distances.

for measured distance vs. ground truth distance, to speed up computation. To construct this likelihood function, there were four probability components, hit, short, max, and random. Each component was weighted by a tunable parameter. The likelihoods were then normalized and used as particle weights in the update step.

Equations

For a range measurement, there are typically a few cases to be modeled in determining $p\big(z_k^{(i)} \mid x_k, m\big)$:

1. Probability of detecting a known obstacle in the map
2. Probability of a short measurement. Maybe due to internal lidar reflections (scratches or oils on the surface), hitting parts of the vehicle itself, or other unknown obstacles (people, cats, etc.)
3. Probability of a very large (aka missed) measurement. Usually due to lidar beams that hit an object with strange reflective properties and did not bounce back to the sensor
4. Probability of a completely random measurement. Just in case of something unexpected.

We typically represent (1) with a Gaussian distribution centered around the ground truth distance between the hypothesis pose and the nearest map obstacle. If the measured range exactly matches the expected range, the probability is maximum. If the measured range is $z_k^{(i)}$ and the ground truth range is determined (via ray casting on the map m from pose x_k to be d, then we have that:

$$p_{hit}\big(z_k^{(i)} \mid x_k, m\big) = \begin{cases} \eta \, \dfrac{1}{\sqrt{2\pi\,\sigma^2}} \exp\Big(-\dfrac{\big(z_k^{(i)} - d\big)^2}{2\,\sigma^2}\Big) & \text{if} \quad 0 \le z_k^{(i)} \le z_{max}, \\ 0 & \text{otherwise.} \end{cases}$$

where η is a normalization constant such that the Gaussian integrates to 1 on the interval $[0, z_{max}]$. For this problem, set $\eta = 1$.

Case (2) is represented as a downward sloping line as the ray gets further from the robot. This is because if the unknown obstacles (people cats, etc.) are distributed uniformly in the environment, the lidar is more likely to hit ones that are closer (think about how area scales...). This likelihood can be modeled as:

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

Case (3) is represented by a large spike in probability at the maximal range value, so that reflected measurements do not significantly discount particle weights. While this is really a delta function $\delta(z_k^{(i)} - z_{max})$, it can be approximated by a uniform distribution close to z_{max}, for small ϵ. For this problem, choose $\epsilon = 0.1$.

$$p_{max}\big(z_k^{(i)} \mid x_k, m\big) = \begin{cases} \dfrac{1}{\epsilon} & \text{if} \quad z_{max} - \epsilon \le z_k^{(i)} \le z_{max}, \\ 0 & \text{otherwise.} \end{cases}$$

Case (4) is represented by a small uniform value, to account for unforeseen effects.

$$p_{rand}\big(z_k^{(i)} \mid x_k, m\big) = \begin{cases} \dfrac{1}{z_{max}} & \text{if} \quad 0 \le z_k^{(i)} \le z_{max}, \\ 0 & \text{otherwise.} \end{cases}$$

These four different distributions are now mixed by a weighted average, defined by the parameters $\alpha_{hit}, \alpha_{short}, \alpha_{max}, \alpha_{rand}$:

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

Note that in order for $p\big(z_k^{(i)} \mid x_k, m\big)$ to be a probability distribution we must have that:

$$\alpha_{hit} + \alpha_{short} + \alpha_{max} + \alpha_{rand} = 1.$$

Resampling

Resampling was then performed to adjust particle positions and improve guess accuracy. It's important to note, however, that the best particle is not just chosen, instead particles are redistributed based on their weights, effectively disregarding unlikely hypotheses and increasing the number of more probable ones.

This was first all done in a 2D racecar simulation environment using ROS2. The simulation included artificially injected noise into the odometry and laser scan data to replicate realistic operation. The MCL was then adapted to operate on the physical racecar.

3 Experimental Evaluation: Lexi Foland

Figure 3: The highlighted green portion represents the path taken by the wall follower program during testing of the particle filter in simulation. The speed was set to 1 m/sec. The car was paused at the end of the trajectory to ensure no unexpected movements occurred while stationary.

3.1 Simulation Testing

We first tested the particle filter's accuracy in simulation; this involved running repeated trials with different amounts of noise added to the motion model's odometry readings. This method allowed observation of the filter's success as well as an opportunity to tune noise parameters. Noise levels were implemented as scaling arguments to the sampling functions utilized in our motion model's algorithm.

Each evaluation consisted of an approximately 25-second execution of the wall follower program on a portion of the map (Figure 3) while the particle filter simultaneously ran. On each timestep, the ground truth pose of the racecar was compared with the output from the particle filter. An error vector consisting of the components x and y was calculated between the true and predicted translations. The magnitude of this error vector (Equation 1) was plotted via RQT. The true rotation of the car, in radians, was compared to the predicted rotation, and the magnitude of the angle error was also recorded, with calculations accounting for the circularity of radians (Equation 2).

$$\mathcal{E}_{\text{trans}} = \sqrt{(x_{pred} - x_{real})^2 + (y_{pred} - y_{real})^2} \tag{1}$$

$$\mathcal{E}_{\text{rot}} = |((\theta_{pred} - \theta_{real} + \pi) \mod 2\pi) - \pi| \tag{2}$$

With no noise added, we evaluated the translational error (Figure 4) and the rotational error (Figure 5) of the particle filter. The poor accuracy in both metrics likely results from the particle cloud converging quickly to one point and thus becoming deterministic. Next, we added Gaussian noise with a scale factor of 0.1 to our translational calculations and with a scale factor of 0.2 to our rotational calculations. We similarly evaluated the translational (Figure 6) and rotational (Figure 7) errors again; the racecar performed better in both metrics with this level of noise. Note that these plots begin not at time 0, when RQT first began running, but rather when the wall follower began and the program started publishing error metrics.

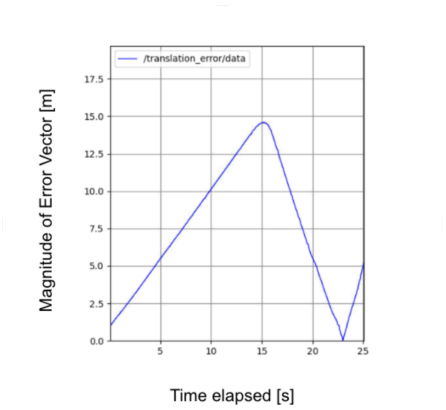

Figure 4: With no noise added to the motion model, this is a plot of time elapsed in simulation versus the magnitude of the translational error vector between ground truth and the estimated pose from the particle filter. Values were variable throughout the trial, indicating a low accuracy of the filter without noise.

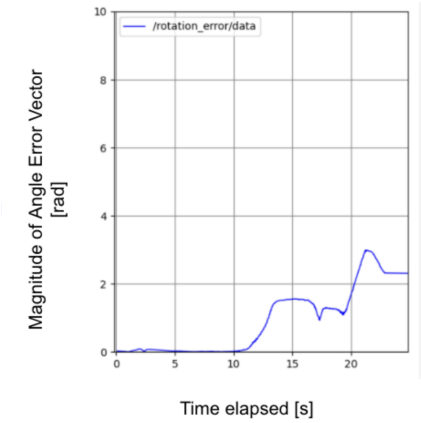

Figure 5: With no noise added to the motion model, this is a plot of time elapsed in simulation versus the magnitude of the rotational angle error between ground truth and the estimated pose from the particle filter. Error increased throughout the trial, indicating a poor accuracy with this level of noise.

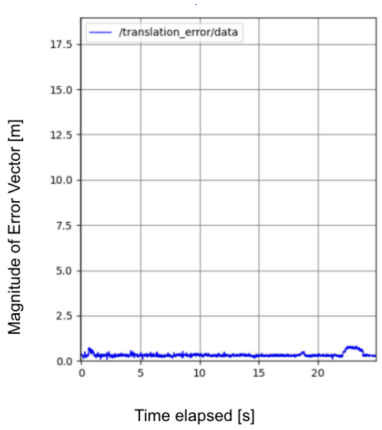

Figure 6: With 0.1-scaled Gaussian noise added to our translational calculations, this is a plot of time elapsed in simulation versus the magnitude of the translational error between ground truth and the estimated pose from the particle filter. Error stayed relatively low throughout the trial, indicating this level of noise was beneficial for the particle filter.

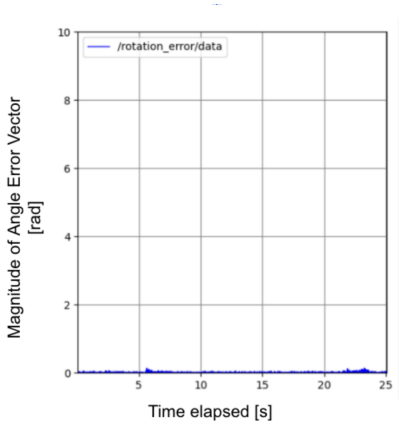

Figure 7: With 0.2-scaled Gaussian noise added to our rotational calculations, this is a plot of time elapsed in simulation versus the magnitude of the angle error between ground truth and the estimated pose from the particle filter. Error stayed consistently low throughout the trial, indicating that our filter benefited from this level of noise.

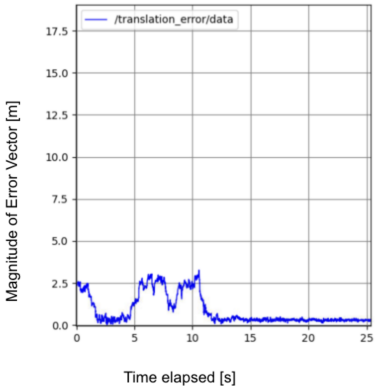

Figure 8: With 0.2-scaled Gaussian noise added to translational calculations, this is a plot of time elapsed versus the magnitude of the translational error vector between ground truth and the estimated pose from the particle filter. Error experienced some variability in the beginning before converging to a lower value later on. This indicates that this noise level may be slightly too high for our program.

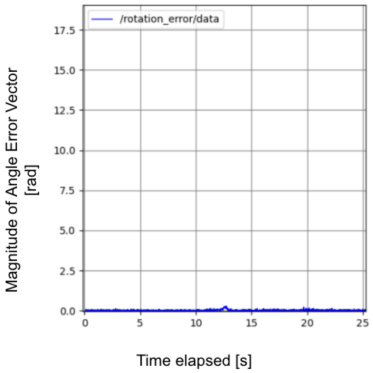

Figure 9: With 0.4-scaled Gaussian noise added to rotational calculations, this is a plot of time elapsed versus the magnitude of the angle error between ground truth and the estimated pose from the particle filter. The angle error stayed consistently low throughout the execution of the program, indicating resilience to this level of noise.

Lastly, we gave our translational calculations 0.2-scaled noise and our rotational calculations 0.4-scaled noise. Translational error stayed volatile until about halfway through the trial (Figure 8), while rotational error stayed consistently low (Figure 9). Comparing the results of these three trials, we adopted the 0.1-0.2 noise scaling as our noise parameters for the motion model.

3.2 Hardware Testing

Hardware testing was less straightforward because there was no "ground truth" to measure. To account for this, we created three paths of waypoints in the Stata basement, then mapped them into our simulation (Figure 10). For each, three trials were recorded: each began with placing the racecar at the "starting marker" in the real world, then initializing the racecar's position in simulation as similarly as possible. The racecar was then driven via teleoperation to the "ending marker," and the particle filter's pose estimate was recorded. This pose estimate was then compared to the simulated location of the "ending marker," which had been recorded before the trials began. The magnitudes of the translational error vector (Equation 1) and the angle error (Equation 2) were recorded for each trial, and then averaged together (Figure 11).

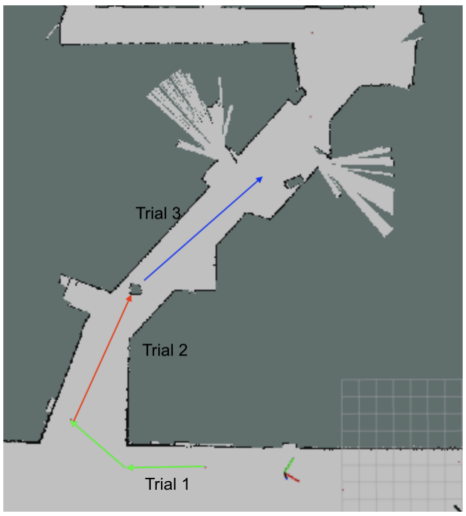

Figure 10: A depiction of the three paths, each marked in a different color, used to test the effectiveness of the particle filter on hardware. The green line represents the path used in Trial 1, the red represents Trial 2, and the blue represents Trial 3.

	Average Translational Error Across 3 Trials [m]	**Average Angle Error Across 3 Trials [rad, deg]**
Trial 1	0.340 m	0.404 rad, ∼23.173 deg
Trial 2	0.767 m	0.239 rad, ∼13.694 deg
Trial 3	0.974 m	0.166 rad, ∼9.511 deg

Figure 11: A table representing the results of the three trials, averaged over three attempts per trial. While Trial 1 showed satisfactory results in the translational error, Trials 2 and 3 showed sub-optimal readings of error. Conversely, Trials 2 and 3 showed much better results in orientation error, while Trial 1 reflected a struggle to fully track the orientation. The variability in results reflects the need for an investigation into how the filter functions in different areas of the map.

The averaged error metrics were within a decent range, always less than a meter off (translational) and 25 degrees off (rotational). However, we recognize that trials 2 and 3 were significantly off-target and are exploring possible explanations for this issue. Further tuning of the motion model's noise values may help. While 0.1-0.2 scaling was effective in simulation, it did not fully succeed on the real racecar - our particle filter might have become too deterministic. On the other hand, we may not have tested high enough noise resilience in simulation,

causing our filter to react unexpectedly to inaccurate sensor readings and wheel slippage. Lastly, trials 2 and 3 were conducted in the same hallway of the Stata basement, an area frequented by students, random clutter, and open classroom doors. Objects distorting sensor readings create edge cases in our code, and we plan to explore methods for making the particle filter more resilient to this issue.

4 Conclusion: Julia Fortt

Given the results of our evaluation metrics, we as a team have concluded that our current localization model is a fairly effective implementation of the MCL algorithm. As briefly mentioned in previous sections however, there are changes we'd like to make to improve the robustness of our model. On the racecar, our model currently performs better in some areas of the Stata basement map than others. This could be for a variety of reasons - a lack of distinctive features in some hallway sections, debris or people acting as unexpected obstacles, overfitted noise, or a number of other possibilities. As such, time permitting, we would like to do additional evaluation runs with the live racecar and noise tuning in simulation to eliminate some of these potential issues and thereby improve overall performance.

4.1 Lexi Foland

Although I had heard about robot localization, I had no reference for how to implement such a technique. Creating a particle filter with my team felt like an introduction to a whole new world of concepts. As a math major, I was pleasantly surprised at how important probability is to effective localization, especially in the Bayes Filter. Additionally, I learned how to get creative with the error metrics; this lab challenged us by taking away the measure of a "ground truth" while working with hardware. I enjoyed finding ways to circumvent this issue with my teammates, and the experience taught me which metrics to value in which situations. Lastly, I learned the importance of version control in programming; early in the lab, I made a mistake in the programming of our filter that was able to be fixed because of version control. Without managing the history of our code, this issue would not have been resolved.

4.2 Julia Fortt

This lab taught me a lot about the strengths and weaknesses of Monte Carlo Localization. Though it makes sense in hindsight, I was surprised to see how unexpected obstacles/a lack of distinctive nearby features could cause the model to struggle even when tuned. This lab also showed me how difficult coming up with effective evaluation methods/metrics can be, regardless of how important they are. It took some time for us as a group to decide how best to evaluate the performance of our localization model on the live racecar, simply because there

was no "straightforward" metric to look at. Both of these things have made me more curious about how MCL algorithms are implemented and used in the real world.

4.3 Evan Kim

Working on this lab taught me a lot about the intricacies of using randomness to model the uncertainties and noise in the real world. It also forced me to think about different evaluation metrics for how to quantify these uncertainties. I gained valuable experience in tuning algorithm parameters to balance accuracy and computational efficiency, which is crucial for the implementations in this class. Additionally, I developed a deeper appreciation for how theoretical concepts in probability translate to solutions for complex problems like robot localization.

4.4 AZ Krebs

I learned a lot about bridging physical systems and algorithmic reasoning in this lab. Mechanical systems almost never behave perfectly. Friction, wheel slip, backlash, noise, etc. all cause this disparity between theory and practice, but this lab taught me how to handle those imperfections using probabilistic modeling.

4.5 Will Peale

This lab gave me a better understanding for the complexity behind localization in robotics. While the theory behind Monte Carlo Localization seemed pretty simple, implementing it on a real robot was much harder than I expected. I learned how to debug failures in the algorithm by breaking the math into components. I also learned how visualization tools like RViz can be used for diagnosing unexpected issues. I came away with a better understanding of how localization methods can help robots know their position in noisy stochastic environments.