Lab 5 Report: Localization

Team # 9

Artemis Pados
Selinna Lin
Min Khant Zaw
Reng Zheng

6.4200 RSS

April 12, 2025

1 Introduction - Artemis Pados

In this lab, we designed a localization system that allows a robot (our racecar) to estimate its position and orientation (its "pose") within a known environment (Stata basement). Localization is a fundamental capability in robotics that is crucial for navigation, mapping, and decision-making. The system we built uses **Monte Carlo Localization (MCL)**, which utilizes a particle filter to maintain and continuously update a set of hypotheses about the car's location.

Our approach combines two key components: a **motion model** and a **sensor model**. The motion model predicts how the car's pose changes over time based on odometry (wheel encoder readings). However, these readings are noisy and imperfect and thus the motion model includes randomness to reflect uncertainty in movement. The sensor model evaluates how likely a given pose is by comparing predicted sensor readings to actual ones.

Bringing it all together to perform localization, the **particle filter** maintains a cloud of particles where each particle represents a possible pose. As the robot moves and sensor data is received, the motion model and sensor model update accordingly based on the new data. The particle filter resamples the particles, favoring those with higher likelihood (weights), to yield a belief of the location of the car that is concentrated at more likely location.

Overall, this lab's work is a demonstration of how a moving robot's location can be estimated with probabilistic methods in the face of noise. This work on localization is foundational for a robust future autonomous system: without

knowing where it is, a robot cannot plan routes, avoid obstacles, or interact meaningfully with its environment.

2 Technical Approach

2.1 Technical Introduction - Reng Zheng

Localization is a common task in robotics to determine the location of the robot in space. In our case, we are using a pre-scanned map of the Stata basement, our primary testing site, and attempting to keep track of where the racecar is in the basement during movement.
We do this with the following components:

1. The prescanned map of Stata basement.
2. Ingested LIDAR and odometry data.
3. A motion model to evolve particles that represent potential positions of the racecar in space.
4. A sensor model to evaluate the probability is represented by any given particle.
5. A particle filter to prune unlikely robot positions.

The odometry data is ingested by the motion model to evolve the points via the provided kinematic information. Noise is added to the evolution in the motion model to represent noise in the odometry data. The evolved points and LIDAR scans are fed into the sensor model, which uses a precomputed table of probabilities to determine the likelihood the robot is at any given point. The particle filter then resamples with replacement the evolved points using the precomputed likelihoods, before passing the new set of particles back to the motion model for the next time step.

2.2 Motion Model - Reng Zheng

The motion model ingests odometry data $\vec{\text{odom}}_k = [\Delta X_k^{\text{racecar}}, \Delta Y_k^{\text{racecar}}, \Delta\theta_k^{\text{racecar}}]^T$, representing the racecar's motion from its reference frame in terms of Cartesian position changes and rotations. As such data always has some noise and uncertainty, we model this by adding a small Gaussian noise to the odometry values each time before they are used to evolve a given particle representing a potential racecar position. We will denote the modified odometry data as $\vec{\text{odom}}_k^{(i)}$, where i is the index of the particle and its associated noise-augmented odometry.
To evolve the particles, we now need to convert the odometry into the map frame by applying the following formula:

$$\delta\vec{X}_k^{(i)} = \begin{bmatrix}\cos(\theta_k) & -\sin(\theta_k)\\ \sin(\theta_k) & \cos(\theta_k)\end{bmatrix}\begin{bmatrix}\Delta X_k^{(i)}\\ \Delta Y_k^{(i)}\end{bmatrix}_{\text{racecar}}$$

where θ_k is the heading of the racecar in the map frame at the current timestep and $\delta\vec{X}_k^{(i)}$ is the change of the particle position in the map frame at timestep k. We can now add $\delta\vec{X}_k^{(i)}$ to the vector $\vec{X}_k^{(i)}$, which represents the guess of the particle i about the current position of the racecar in the map frame. This yields $\vec{X}_{k+1}^{(i)} = \vec{X}_k^{(i)} + \delta\vec{X}_k^{(i)}$, or the evolved point i at timestep $k+1$. Furthermore, because we are using Ackermann Steering, we know that heading changes from the racecar's frame are additive to the map's frame, meaning we know $\theta_{k+1} = \theta_k^{(i)} + \delta\theta^{(i)}$. Thus, we get the following equations for the motion model evolution:

$$\begin{aligned}\vec{X}_{k+1}^{(i)} &= \vec{X}_k^{(i)} + \delta\vec{X}_k^{(i)} \\ \theta_{k+1}^{(i)} &= \theta_k^{(i)} + \delta\theta_k^{(i)}\end{aligned}$$

2.3 Sensor Model - Artemis Pados

In our pipeline for our racecar's localization, the sensor model assigns likelihoods to each particle based on how well that particle's predicted laser scan matches the actual scan observed by the robot. These likelihoods are critical for the MCL framework where particles representing more plausible car poses are given higher weights (and are thus more likely to be sampled in the resampling step of the particle filter).

Our sensor model evaluates the probability $p(z_k \mid x_k, m)$ of an observed scan z_k given a particle pose x_k and a known map m. This is computed as a weighted sum of four model components:

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

where

- $z_k^{(i)}$ is the i-th beam of the observed scan,
- d is the expected range from the particle's pose (found via raycasting),
- and $\alpha_{\text{hit}} = 0.74$, $\alpha_{\text{short}} = 0.07$, $\alpha_{\text{max}} = 0.07$, $\alpha_{\text{rand}} = 0.12$ is given.

Within this weighted sum, each term models a particular physical phenomenon. Specifically,

- p_{hit} is a truncated Gaussian centered at d with standard deviation $\sigma_{\text{hit}} = 0.5$, i.e.

$$p_{\text{hit}}(z \mid d) = \eta \cdot \exp\left(-\frac{(z-d)^2}{2\sigma_{\text{hit}}^2}\right), \quad 0 \le z \le z_{\text{max}},$$

- p_{short} represents unexpected short readings, i.e.

$$p_{\text{short}}(z \mid d) = \begin{cases} \frac{2}{d}\left(1 - \frac{z}{d}\right), & 0 \le z \le d \\ 0, & \text{otherwise} \end{cases};$$

- $p_{\max}$ represents a mass at the biggest possible range, i.e.

$$p_{\max}(z) = \begin{cases} 1, & z = z_{\max} \\ 0, & \text{otherwise} \end{cases},$$

- and p_{rand} is uniform distribution over all possible ranges, i.e.

$$p_{\text{rand}}(z) = \frac{1}{z_{\max}}.$$

For all combinations of discrete expected distances d and measured ranges z, we precompute $p(z_k^{(i)} \mid d^{(i)})$ and store them in a 201×201 lookup table where each column (for a fixed d) is normalized to sum to 1.

At runtime, we use the `PyScanSimulator2D` library to generate simulated LIDAR scans for each particle pose $x_k^{(j)}$. These simulated measurements $d^{(j,i)}$ approximate what the LIDAR would have returned if the robot were actually at $x_k^{(j)}$. We then discretize both the real LIDAR observation $z_k^{(i)}$ and the simulated distances $d^{(j,i)}$, and use them to index into the precomputed sensor model table. This gives us the likelihood of observing $z_k^{(i)}$ given the expected measurement $d^{(j,i)}$ for each beam i.
The total likelihood of observing z_k given particle $x_k^{(j)}$ and map m is then computed as:

$$P(z_k \mid x_k^{(j)}, m) = \prod_{i=1}^{n} p(z_k^{(i)} \mid d^{(j,i)}),$$

which serves as the importance weight for particle $x_k^{(j)}$ in the particle filter.

Overall, this model balances physical accuracy with computational efficiency. The precomputed table trades memory space for fast runtime performance even when evaluating many particles.

2.4 Particle Filter - Selinna Lin

The particle filter integrates the motion model (Section 2.2) and sensor model (Section 2.3) to implement our Monte Carlo Localization algorithm.

First, the particle filter initializes particles around a gaussian to provide a guess of where the robot's location is with a random spread of particles around the clicked intial pose. This happens whenever a new pose for the robot is initialized on the map.

The odometry callback passes processed odometry data into the motion model for updating particles with accounted noise. The laser scans callback passes

scanned data from the LiDAR into the sensor model to evaluate for the weights of the particles (i.e. the likelihood that the particle is accurate in estimating the odometry of the robot). We use these weights to resample the particles and publish the estimated pose as a transformation between the /map frame and a frame for the car's expected base link, which is /base_link_pf for the simulator and /base_link for the real car.

It is worth noting that we ran into race conditions, and so we added a lock from the `threading` library such that the lock would be acquired whenever the particles are being updated.

3 Experimental Evaluation - Min Khant Zaw / Selinna Lin

3.1 Simulation - Min Khant Zaw

To evaluate the performance of our particle filter, we first tested it by running our wall follower in simulation. We had the car follow the wall over a long run, making it turn around the corner and obstacles. As we can see from Fig. 1, the pose estimated by our particle filter, which is indicated by the red arrow, aligns correctly with the pose of the robot even when the robot is turning around the corner.

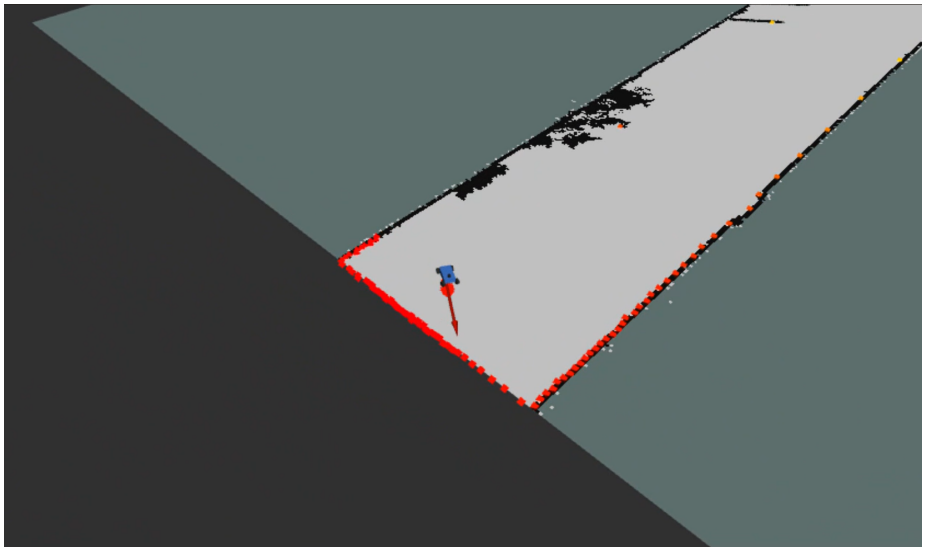
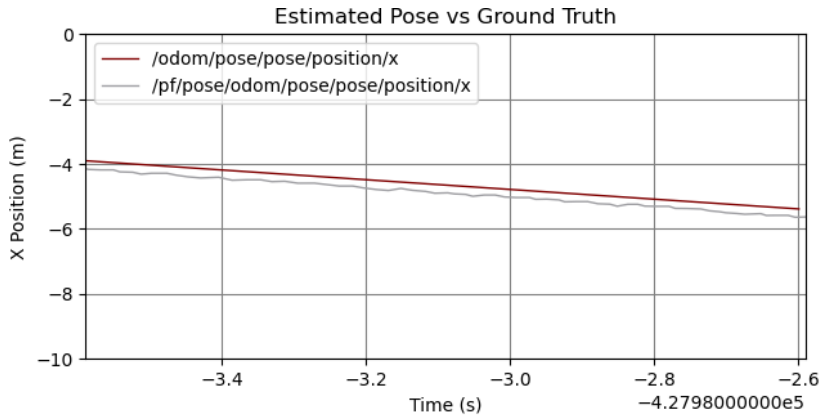
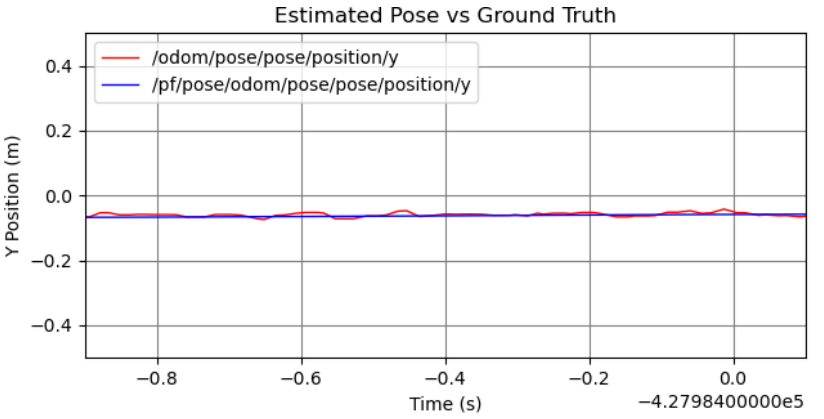


Fig. 1: The picture shows that the estimated pose correctly aligns with the actual pose of the robot.

We then plotted the estimated pose from the topic /pf/pose/odom and the ground truth from the topic /odom using the rqt graph to numerically visualize the performance of our particle filter. We first plotted the x and y-coordinates of the estimated pose and the ground truth. We found that the y-coordinate of the estimated pose stays closer to the ground truth than the x-coordinate (Fig. 2). This behavior is due to having more noise in x than y in our motion model since there are more features in x than y in the map.



(a) x-coordinate

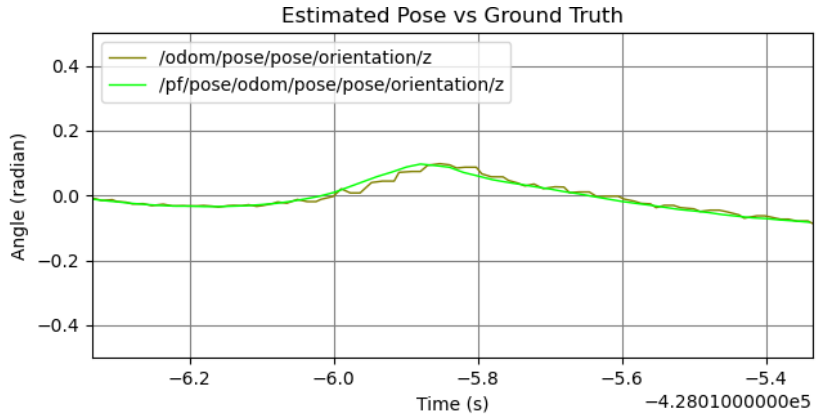


(b) y-coordinate

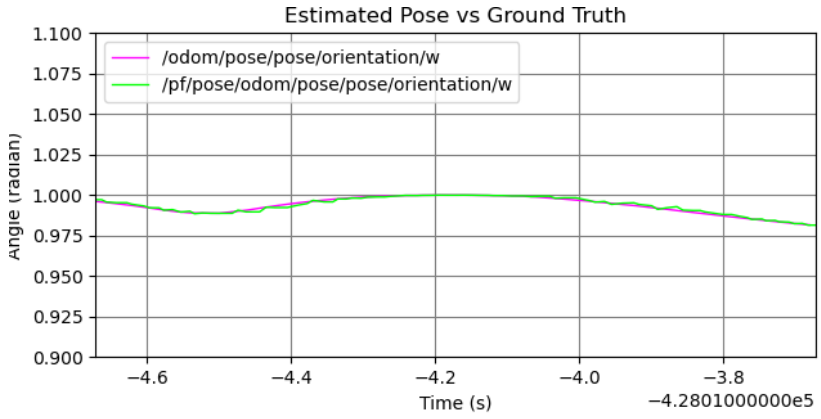
Fig. 2: Graphs (a) and (b) display the x-coordinate and y-coordinate of the estimated pose and the actual pose respectively. **Both x and y-coordinates of the estimated pose stay close to the ground truth.**

We also plotted z and w of the orientation of the estimated pose and the ground truth to compare them. We saw that both z and w of the estimated pose remain

almost the same as the ground truth (Fig. 3). We can see that even when z and w of the estimated pose diverge from the ground truth, they converge back fairly quickly.



(a) z of orientation



(b) w of orientation

Fig. 3: Graphs (a) and (b) display the z and w of orientation of the estimated pose and the actual pose respectively. **Both z and w of orientation of the estimated pose stay close to the ground truth.**

We calculated the absolute error of the estimated pose and found the maximum and minimum errors (Table 1). We can see that our particle filter is able to estimate the pose of the robot exactly as the ground truth at some locations, giving us the minimum errors of 0 in all of them.

Table 1: Maximum and minimum error values in x, y coordinates and z, w of orientation.

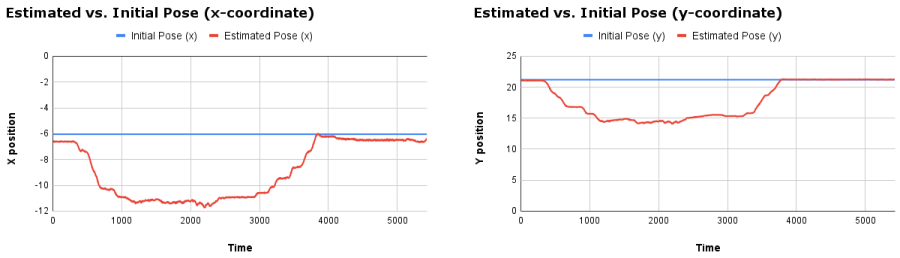
Maximum Error in x-coordinate	0.6742
Minimum Error in x-coordinate	0
Maximum Error in y-coordinate	0.01856
Minimum Error in y-coordinate	0
Maximum Error in z of orientation	0.1244
Minimum Error in z of orientation	0
Maximum Error in w of orientation	0.02944
Minimum Error in w of orientation	0

3.2 Real World - Selinna Lin

After successfully implementing and testing our localization algorithm in simulation, we transitioned to testing it on the actual car. Our goal was to verify whether the car could correctly identify its position on the map when initialized at a location that roughly corresponds to its real-world starting point, and whether our MCL algorithm could accurately estimate future poses as the car moved.

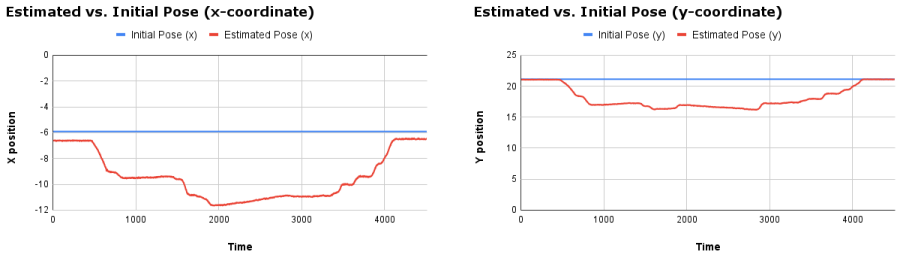
To evaluate this, we initialized the car at a known starting position, drove it around an arbitrary path, and then returned it to the original location. We then checked whether the estimated x and y positions converged back to the initial values. This served as a consistency check: successful convergence would indicate that the vehicle correctly recognized it had returned to a previously visited location, thereby validating the accuracy and robustness of the localization system.

We conducted three trials of this experiment. The results of this experiment indicate that the localization algorithm successfully identified previously visited positions. Figures 4, 5, and 6 present the graphical results for all three trials. In each case, the estimated poses demonstrate a clear convergence back to the initial position, confirming the system's ability to recognize and return to a known location. Minor deviations observed across trials can be attributed to human driving error.



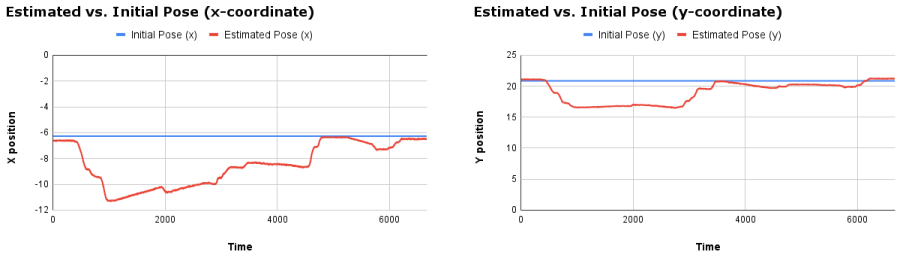
(a) x-coordinate comparison. (b) y-coordinate comparison.

Fig. 4: We show **Trial 1** results depicting the Estimated vs. Initial Pose for x- and y-coordinates.



(a) x-coordinate comparison. (b) y-coordinate comparison.

Fig. 5: We show **Trial 2** results depicting the Estimated vs. Initial Pose for x- and y-coordinates.



(a) x-coordinate comparison. (b) y-coordinate comparison.

Fig. 6: We show **Trial 3** results depicting the Estimated vs. Initial Pose for x- and y-coordinates.

4 Conclusion - Artemis Pados/Selinna Lin

In this lab, we successfully designed and implemented a full Monte Carlo Localization (MCL) pipeline to estimate our car's pose within a known map using noisy sensor and motion data. Our system integrates three critical components: a probabilistic motion model that updates particle positions based on odometry data, a sensor model that assigns weights to particles by comparing simulated and real LIDAR measurements, and a particle filter that resamples these particles to track the most likely pose over time. Each of these components was grounded in probabilistic robotics theory, and we made deliberate design decisions to balance computational ease with effective modeling.

Through our experimentally gathered data and real-time observations in both simulation and the real world, we have validated the robustness and accuracy of our MCL pipeline in estimating the vehicle's pose while car is in motion. These encouraging results not only confirm the effectiveness of our approach, but also lay a strong foundation for future refinement and enhancements.

Handling kidnapped robot scenarios as well as multi-modal distributions in a robust manner is a natural next step for the localization capabilities of our racecar. Further, the current system we have implemented assumes a static map. Extending our work to account for dynamic obstacles (such as pedestrians or other cars in a real world scenario) would make our localization efforts more resilient. Regardless, this localization module will serve as the backbone for future autonomous behavior such as path planning and control and we look forward to expanding our work in a reliable and effective manner.

5 Lessons Learned

5.1 Artemis Pados

In this lab, I focused a lot on the sensor model for the particle filter. I found that doing the math by hand for this (and formally stepping through the derivation of the Bayes filter) really helped solidify my understanding of probabilistic reasoning in robotics. It was rewarding to see how the theoretical tools we discussed in lecture and interacted with mathematically directly apply and work on physical systems like the racecar. Working in parallel with my teammates, who handled the motion model and integration with the particle filter, emphasized how effective communication and a shared grasp of the math allowed us to not only intertwine our separate working parts, but also wholistically develop a well-functioning system. This week's lab has deepened my appreciation for both the theory and engineering behind robots and autonomous systems.

5.2 Selinna Lin

Overall, I feel that the lab's most challenging part was integration on the car as well as any debugging we ran into on the way. We were pretty quick to implement each component (motion model, sensor model, and particle filter) individually. I worked on the particle filter, and since we were developing in parallel, I had to trust that my teammates' modules would work as expected, since mine depended on theirs. I think we did a solid job communicating both technically and logistically. In terms of technical, we all understood what the components did and how they would come together even if each one of us didn't work on all the parts - which would be inefficient if we did. In terms of logistical, we had clear team goals and deadlines in mind of when we most definitely wanted something done and the means of achieving that, as well as what things we can push back to focus on more important tasks. I'm really happy with what we were able to pull off by the end of this lab, and I'm excited to see how our team continues to work together moving forward!

5.3 Min Khant Zaw

For this lab, integrating the algorithm on the robot was the most challenging because we faced various errors and issues such as the differences between simulation and the actual robot. I definitely learned how to think while debugging and how to use various tools such as rqt graph in this lab since we could not figure out how to use it in previous labs. Our team was able to finish a working algorithm earlier than the deadline, but optimizing it to perform better on the actual robot took longer than we expected. The communication between our team members was solid since there was no issue caused by miscommunication as far as I remember. The more labs I have worked on, the more I realized that everything might not go smoothly when we test our algorithms on the robot, and that it is important to start working on the lab as early as possible.

5.4 Reng Zheng

Overall, the lab's most challenging section was coordinating the integration of parallel components. Although we were well-communicated, the increased complexity of the tasks made integration much harder to debug than in previous labs. However, because we started development and integration early, this was trivial to solve and we always left Stata basement before midnight. Overall, I'm happy with how we approached the lab, given that we had little conflict and low stress throughout. I think that the takeaway from this lab, then, is that starting early, communicating often, and integrating early are all good strategies to parallel development.