Lab 5 Report: Localization

Team 11

Suchitha Channapatna
Luis De Anda
Anastasiia Kutakh
Amber Lien
Prince Patel

Robotics Science and Systems

April 12, 2025

1 Introduction

Localization is a fundamental challenge in autonomous robotics as it enables a robot to determine its position within a known environment. In this lab, we implemented Monte Carlo Localization (MCL) on the racecar. The ability to accurately localize is essential for navigation, path planning, and obstacle avoidance — all key components of an autonomous system.

Our implementation follows the particle filter approach, which maintains a set of hypothesized poses (particles) and updates their likelihood based on sensor measurements and motion data. This probabilistic method is particularly well-suited for robotics applications because of its robustness to sensor noise and environmental uncertainties.

The technical problem addressed involves fusing odometry data with laser scan measurements to continuously estimate the racecar's pose. Our solution is comprised of three main components: a motion model that predicts particle movement based on odometry, a sensor model that evaluates particle likelihood based on laser scans, and a particle filter that integrates these models to estimate the racecar's pose.

This localization system allows the racecar to determine it's position in a previously mapped environment solely based on LiDAR data. The implemented MCL algorithm not only provides accurate position estimation, but also effectively handles the inherent uncertainty in robotics systems through a probabilistic framework. Our technical approach prioritizes computational efficiency and real-time performance, making it suitable for deployment on our physical racecar.

2 Technical Approach

2.1 Problem Formulation

The localization problem involves estimating the robot's pose (x, y, θ) within a known map given a stream of sensor measurements and control inputs. Formally, we estimate the posterior probability distribution $p(x_t|z_{1:t}, u_{1:t})$, where x_t is the robot's pose at time t, $z_{1:t}$ are the sensor measurements up to time t, and $u_{1:t}$ are the control inputs up to time t.

MCL addresses this problem by approximating this distribution with a set of weighted particles. Each particle represents a possible pose, and its weight reflects the likelihood of being the true pose given the sensor measurements. Ray casting is used to generate simulated LiDAR ranges from each of these particles, which can then be compared to the true LiDAR ranges from the racecar. The algorithm follows a predict-update-resample cycle, where:

1. Particles are propagated according to the motion model (Prediction)
2. Particle weights are updated based on sensor measurements (Update)
3. Particles are resampled according to their weights (Resampling)

2.2 System Architecture

Our implementation follows a modular design with three main components, each encapsulated in a separate Python file and depicted in Fig. 1:

- **Motion Model:** Predicts how particles move based on odometry data, incorporating both deterministic updates and noise models to account for real-world uncertainties.
- **Sensor Model:** Evaluates the likelihood of each particle's pose given laser scan measurements using a precomputed probability lookup table for efficiency.
- **Particle Filter:** Initializes particles, integrates the motion and sensor models, manages particle weights and resampling, and estimates the robot's pose from the particle distribution.

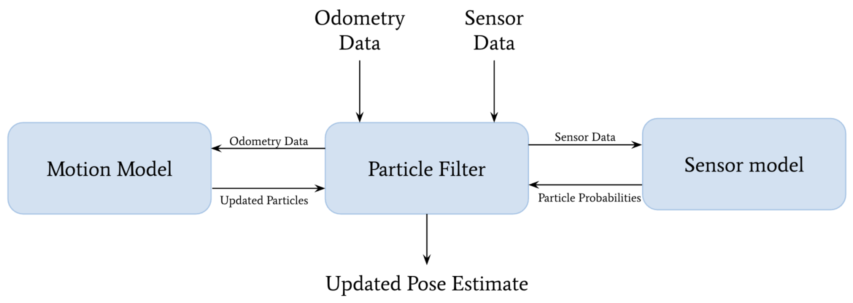


Fig. 1: **The system architecture.** Odometry and sensor data are provided to the Particle Filter module. Odometry data is then passed to the Motion Model which updates the set of particles accordingly. Sensor data is passed to the Sensor Model which returns particle probabilities based on the LiDAR scans. A new set of particles is aggregated and used to compute a pose estimate.

2.3 Motion Model

The motion model implementation includes:

1. **Parameter Setup**: Declaration of ROS parameters for noise configuration.
2. **Noise Update**: Computation of noise based on current linear and angular velocities.
3. **Particle Update**: Transformation of odometry data from the robot's frame to the world frame, computation of pose in the next time step, and addition of noise.

The motion model updates particle poses based on odometry data. Given a particle at pose $(x_{t-1}, y_{t-1}, \theta_{t-1})$ in the world frame and an odometry reading $(\Delta x, \Delta y, \Delta\theta)$ in the robot's frame, the new pose (x_t, y_t, θ_t) is computed as:

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + \begin{bmatrix} \cos(\theta_{t-1}) & -\sin(\theta_{t-1}) \\ \sin(\theta_{t-1}) & \cos(\theta_{t-1}) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} + \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \tag{1}$$

$$\theta_t = \theta_{t-1} + \Delta\theta + \epsilon_\theta \tag{2}$$

where ϵ_x, ϵ_y, and ϵ_θ represents noise added to model real-world uncertainties. Empirically, we found uncertainty increases with speed. These observations were incorporated into the following three noise models:

1. **Gaussian Noise**: Models sensor and motion uncertainty using a Gaussian distribution. There is a baseline standard deviation that is additionally linearly scaled by speed.

2. **Uniform Noise**: Models noise uniformly spread within a given range. The size of this range is linearly scaled by speed.

3. **Exponential Noise**: Models one-sided uncertainty (e.g., wheel slip) using a baseline scaling factor, also linearly scaled by speed.

Future data showed that Gaussian noise provided the most realistic behavior and was thus selected as the default model (Section 3.2). The noise parameters are tunable through ROS parameters, allowing for adaptation to different environments and hardware configurations.

By combining deterministic updates with stochastic noise, the motion model aims to capture the uncertainty in the robot's movement.

2.4 Sensor Model

The sensor model evaluates how well each particle's hypothesized pose matches observed laser scan data. The sensor model implementation includes:

1. **Table Precomputation**: Creation of the lookup table during initialization.

2. **Map Integration**: Processing of the occupancy grid map for ray casting.

3. **Likelihood Evaluation**: Computation of particle likelihoods given observed laser scans.

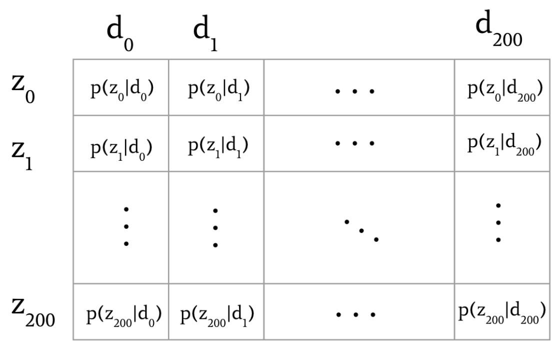
For efficiency, we precompute a lookup table of probabilities based on four components:

1. **Hit Model (p_{hit})**: A Gaussian probability centered at the expected distance, with weight $\alpha_{hit} = 0.74$.

2. **Short Model (p_{short})**: A linear probability that decreases with distance, modeling early reflections, with weight $\alpha_{short} = 0.07$.

3. **Max Model (p_{max})**: A spike probability at maximum sensor range, modeling failures to detect obstacles, with weight $\alpha_{max} = 0.07$.

4. **Random Model (p_{rand})**: A uniform probability over the sensor range, modeling random noise, with weight $\alpha_{rand} = 0.12$.



	d_0	d_1		d_{200}
z_0	$p(z_0 \mid d_0)$	$p(z_0 \mid d_1)$	$\cdots$	$p(z_0 \mid d_{200})$
z_1	$p(z_1 \mid d_0)$	$p(z_1 \mid d_1)$	$\cdots$	$p(z_1 \mid d_{200})$
	$\vdots$	$\vdots$	$\ddots$	$\vdots$
z_{200}	$p(z_{200} \mid d_0)$	$p(z_{200} \mid d_1)$	$\cdots$	$p(z_{200} \mid d_{200})$

Fig. 2: **The precomputed table calculated in the sensor model.** The columns are possible ground truth distances and the rows possible measurements. Entry (i, j) in the table gives $p(z_i|d_j)$, the probability of measuring z_i given that the true distance is d_j.

The equations defining the probability models above are presented below, with the total probability being given by a weighted sum of these models:

$$p_{hit}(z|x,m) = \eta \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-x)^2}{2\sigma^2}\right) \quad \text{if } 0 \leq z \leq z_{max} \text{ else } 0 \tag{3}$$

$$p_{short}(z|x,m) = \frac{2}{x}\left(1 - \frac{z}{x}\right) \quad \text{if } 0 \leq z \leq d \text{ and } d \neq 0 \text{ else } 0 \tag{4}$$

$$p_{max}(z|x,m) = \frac{1}{\epsilon} \quad \text{if } z_{max} - \epsilon \leq z \leq z_{max} \text{ else } 0 \tag{5}$$

$$p_{rand}(z|x,m) = \frac{1}{z_{max}} \quad \text{if } 0 \leq z \leq z_{max} \text{ else } 0 \tag{6}$$

$$p(z|x,m) = \alpha_{hit} \cdot p_{hit} + \alpha_{short} \cdot p_{short} + \alpha_{max} \cdot p_{max} + \alpha_{rand} \cdot p_{rand} \tag{7}$$

where z is the observed range, x is the particle pose, and m is the map.

To compute these probabilities efficiently at runtime, we precompute a 201×201 lookup table indexed by discretized ranges as shown in Fig. 2. For each possible ground truth distance d_j, we compute the likelihood for each possible measurement z_i values (for i, j discrete). This approach significantly reduces computational overhead during runtime evaluation. Note that this additionally changes the equation defining $p_{max}(z|x,m)$, now given by:

$$p_{max}(z|x,m) = 1 \text{ if } z_i = z_{max} \text{ else } 0 \tag{8}$$

By vectorizing these operations with NumPy, we achieve the required real-time performance (> 20Hz) specified in the lab requirements.

2.5 Particle Filter

The particle filter integrates the motion and sensor models to perform MCL. It follows the predict-update-resample cycle:

```
Initialize particles around the initial pose;
while running do
    if odometry received then
        Update particles using motion model;
        Estimate robot pose from particles;
        Publish pose estimate and visualizations;
    end
    if laser scan received then
        Compute particle weights using sensor model;
        Normalize weights;
        Resample particles based on weights;
        Estimate robot pose from particles;
        Publish pose estimate and visualizations;
    end
end
```

Algorithm 1: Particle Filter Algorithm

2.5.1 Particle Initialization

Particles are initialized around the provided initial pose with added Gaussian noise, creating a diverse initial distribution.

2.5.2 Pose Estimation

Initially, we investigated k-means clustering to handle potential multimodal distributions (where particles might form distinct groups representing different possible locations). K-means clustering would allow us to identify the largest cluster of particles and compute the pose estimate from that cluster alone, potentially providing more robust estimates in environments with ambiguous features.

However, we ultimately decided against using k-means clustering for two reasons: (1) its computational overhead would impact our real-time performance requirements, and (2) in our testing environment, the particle distribution was typically unimodal due to the particle initialization and relatively small and feature-rich map that provided sufficient disambiguating information.

Instead, we implemented a simpler approach, computing the robot's pose estimate as the weighted average of particle poses. For position (x, y), we use arithmetic mean:

$$x_{est} = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad y_{est} = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{9}$$

For orientation θ, we use circular mean to properly handle angle wrapping:

$$\theta_{est} = \text{atan2}\left(\frac{1}{N}\sum_{i=1}^{N}\sin(\theta_i), \frac{1}{N}\sum_{i=1}^{N}\cos(\theta_i)\right) \tag{10}$$

This approach offers a good balance between computational efficiency and accuracy for our unimodal particle distributions. For future work in larger environments where multimodal distributions are more likely, the k-means approach could be revisited with optimizations to reduce its computational cost.

2.5.3 Resampling

We use weighted random selection to resample particles, favoring those with higher likelihood. This process naturally focuses on promising hypotheses while maintaining diversity.

2.5.4 Concurrency

We employ thread locks to safely update particles across different callbacks (odometry, laser, and pose initialization), preventing race conditions.

2.5.5 Visualization

We publish visualization markers to inspect the particle distribution and pose estimate in RViz, aiding in debugging and demonstration.

Our particle filter is parameterized with ROS parameters, allowing for easy tuning of the number of particles, resampling frequency, and other parameters affecting performance and accuracy.

3 Experimental Evaluation

3.1 Technical Procedures

We evaluated our localization system in simulation and on the physical racecar, focusing on:

- **Position Error**: The Euclidean distance between estimated pose and ground truth.

- **Orientation Error**: The absolute angular difference between estimated and ground truth orientations.
- **Convergence Time**: The time required for the particle filter to converge to a stable estimate.
- **Computational Speed**: The frequency at which we could publish a new pose estimate.

Simulated tests used a pre-recorded map, generating ground truth data by running the robot along predefined trajectories (Fig. 3). For the physical racecar, we marked known positions on the floor and measured the difference between the estimated and actual positions.

We conducted three sets of experiments:

1. **Noise Model Comparison**: We compared Gaussian, uniform, and exponential noise models to determine which best captures real-world odometry uncertainties.
2. **Parameter Tuning**: We varied key parameters (noise scale) to optimize performance.
3. **Real-World Testing**: We evaluated the system in a dynamic environment with imperfections and varying lighting conditions.

3.2 Results

3.2.1 Noise Model Comparison

We ran our wall follower, generating a track and recording rosbags of the racecar's x-error, y-error, and θ-error, obtained by finding the difference between the estimated and true x and y positions. θ-error was obtained by converting quaternions to Euler angles and calculating the difference while accounting for wrap-around.

We varied the noise models, finding that the Gaussian noise model performed best with average position errors of 0.32 m in x, 0.02 m in y, and an orientation error of 0.31 radians. The uniform noise model resulted in larger errors: 0.35 m in x, 0.12 m in y, and 0.48 radians in orientation. Results for uniform and Gaussian models over time are shown in Figures 4 and 5 respectively. The exponential noise model performed worst, drifting before the car began to move, suggesting it is not suitable for modeling odometry noise.

3.2.2 Convergence Analysis

An initial pose for the robot was published at (-8.0 m, 18.0 m). Particles were generated by adding Gaussian noise with a standard deviation of 1.25 m to this location, their orientations randomly generated.

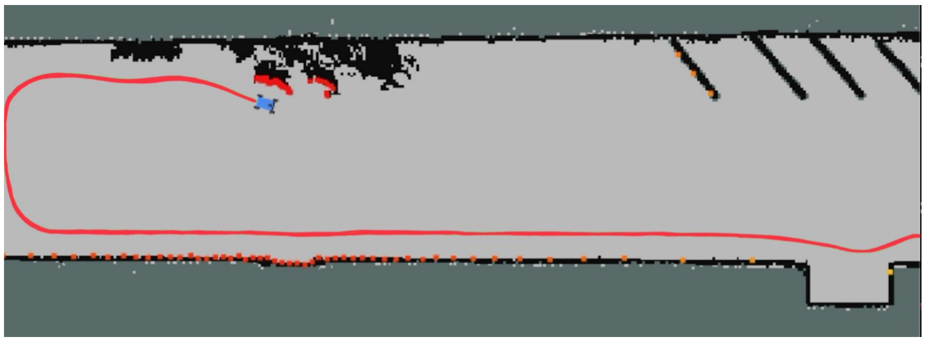


Fig. 3: **The curve depicts the path followed during our experimental evaluations.** Our track includes a straight path in addition to a turn and noisy objects.

We defined convergence to be when no particles were outside a tolerance around this initial position, where tolerance was defined as a circle centered at (-8.0, 18.0) with radius 0.3m, approximately the car's width.

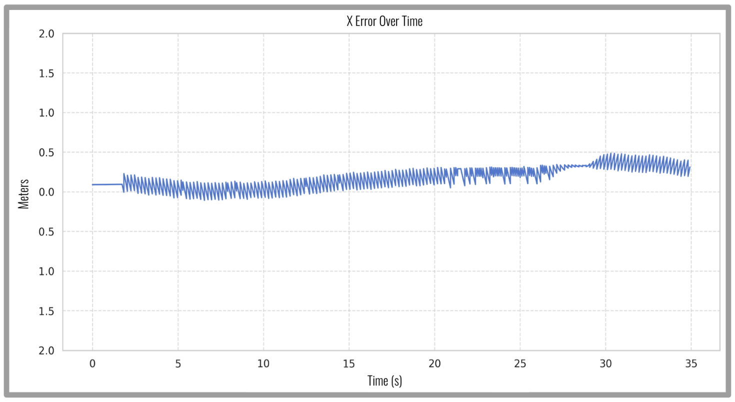
The car remained static as the Gaussian noise model added noise. With baseline standard deviation 0.005 m, our particle filter converged within 0.5 seconds, with an average of 41 particles (out of 200) remaining out of tolerance, resulting in a 79.5% convergence rate. Increasing the standard deviation to 0.04 m increased convergence time to 1.25 seconds, with similar final performance (77.52% convergence rate) (Fig. 6, Fig. 7).

3.2.3 Real-World Performance

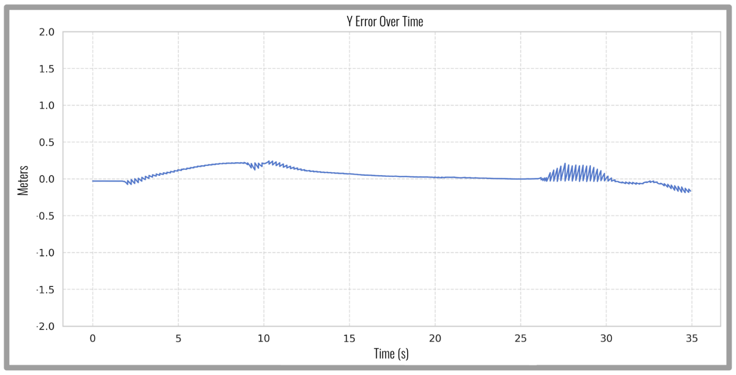
Analysis of real-world performance is difficult due to the lack of a ground truth measurement. When the particle filter is applied to a real-world, teleoperated driving scenario, the estimated pose and actual position of the racecar are the only observable metrics. In an attempt to establish a ground truth, a measuring tape was used to measure the position of the racecar relative to some feature in the environment. Then, using a rosbag of the robot's traveled path, that same feature was identified in the visualized map. The distance between the published estimated pose and that feature was measured using the rviz measure tool. There is room for error in this method as identifying the exact pixel of the feature on the map is very difficult/error-prone. We provide the results below (Table 1), but acknowledge that error analysis is best done in simulation.

3.2.4 Computational Speed

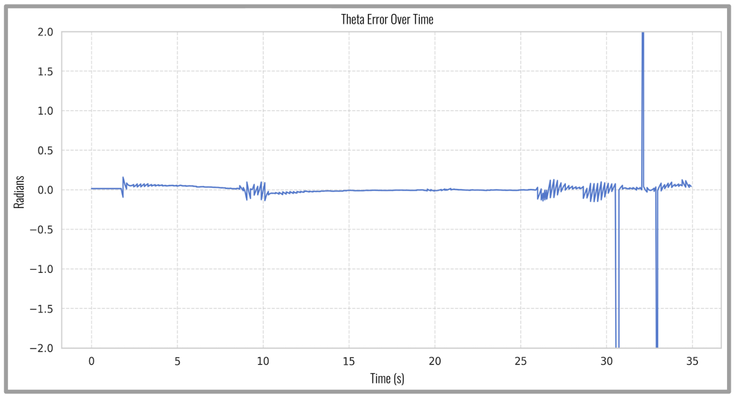
Our implementation achieved a runtime frequency of 25 Hz on the racecar's onboard computer, exceeding the minimum requirement of 20 Hz. The



(a)

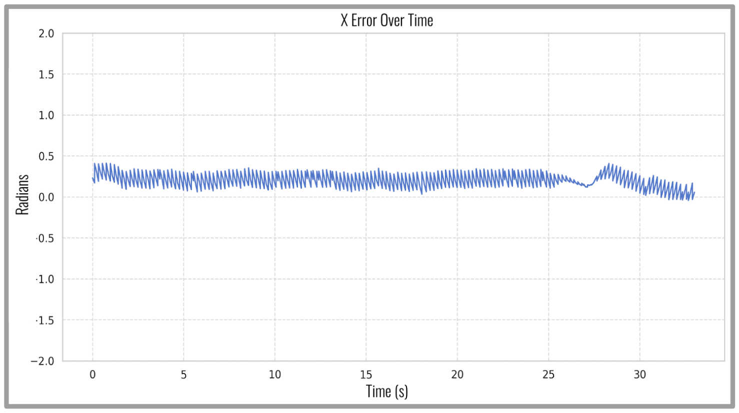


(b)

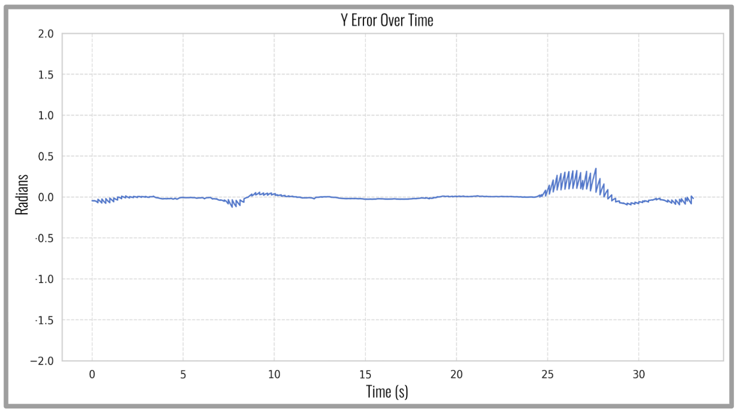


(c)

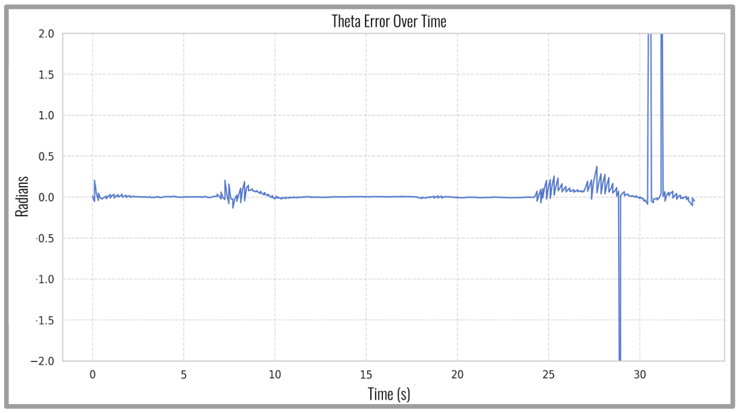
Fig. 4: **The graphs of x, y, and theta errors using noise drawn from the uniform distribution.** The x and y errors stay near zero, though x-error notably drifts over time. Theta error remains near zero radians.



(a)



(b)



(c)

Fig. 5: **The graphs of x, y, and θ-errors using noise drawn from a Gaussian distribution.** The x and y errors stay near zero, with x-error moving towards 0 even as time goes on. θ-error remains near zero radians.

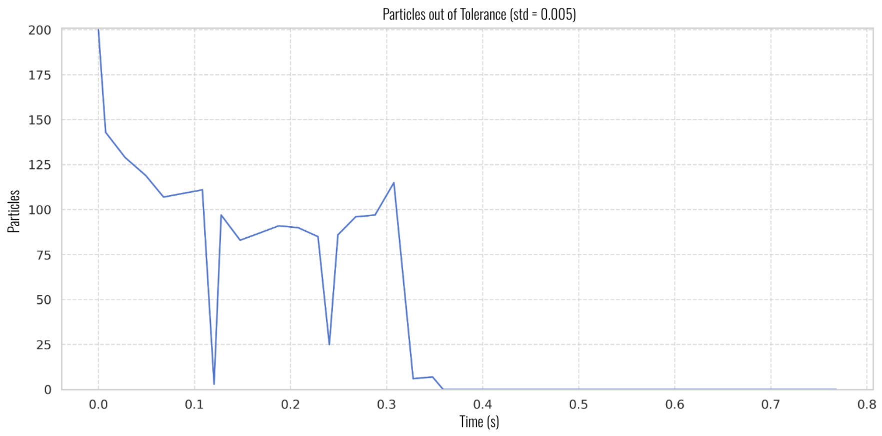


Fig. 6: **The number of particles within tolerance when the Gaussian noise model has a baseline standard deviation of 0.005 m.** The particles out of tolerance dropped from 200 to 0 within 0.5 seconds, demonstrating robust convergence.

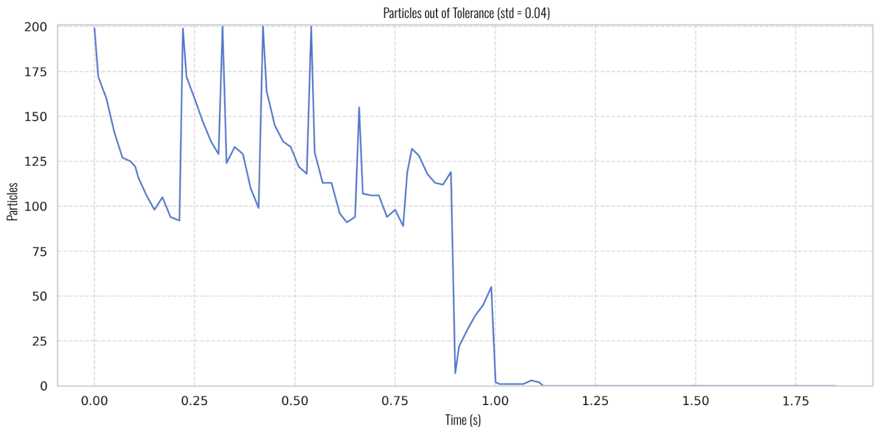


Fig. 7: **The number of particles within tolerance when the Gaussian noise model has a baseline standard deviation of 0.04 m**. The particles out of tolerance dropped from 200 to 0 within 1.25 seconds. Notably, there is more oscillation compared to the standard deviation 0.005 m case due to the increased noise. However, convergence remains resilient to increases in noise.

Table 1: **Localization Error Across Trials**

Trial	X Error (m)	Y Error (m)
1	0.07	0.14
2	0.09	0.11
3	0.16	0.12
Average	0.11	0.12

precomputation of the sensor model lookup table and vectorized operations in NumPy were key to achieving this performance.

The experimental results demonstrate that our MCL implementation localizes the robot with sufficient accuracy for autonomous navigation. The Gaussian noise model provided the best balance between exploration (maintaining particle diversity) and exploitation (converging to the correct pose). The system's real-time performance and sub-car-length accuracy make it suitable for deployment in controlled and dynamic environments.

4 Conclusion

In this lab, we successfully implemented Monte Carlo Localization for our racecar, enabling it to accurately determine its position within a known environment. Our solution integrates odometry data and laser scan measurements through a probabilistic framework that effectively handles the uncertainties inherent in robotic systems.

Key achievements of our implementation include:

1. Development of a modular architecture with separate motion and sensor models, facilitating independent testing and tuning.
2. Implementation of a robust particle filter that efficiently samples the pose space and converges to accurate estimates.
3. Achievement of real-time performance (25 Hz) through algorithmic optimizations like precomputed lookup tables and vectorized operations.
4. Demonstration of sub-car-length accuracy in real-world testing, with average position errors of 0.11 m in x and 0.12 m in y.

Our localization system forms a crucial component of the autonomous navigation stack we have been building throughout the course. Combined with the mapping capabilities from previous labs, it provides the foundation for path planning and control in subsequent labs.

While our current implementation meets the requirements of the lab, several improvements could enhance its performance:

1. Further experimentation with noise models and parameters to better capture real-world uncertainties.

2. Implementation of adaptive particle counts that increase during high uncertainty and decrease during stable operation.

3. Enhanced handling of the “kidnapped robot” problem through global localization when confidence drops.

4. Integration with simultaneous localization and mapping (SLAM) to update the map based on new observations.

As we move forward, this localization system will provide crucial pose estimates for path planning and execution, bringing our racecar closer to full autonomy.

5 Lessons Learned

All members contributed equally to this report. Every section was written by and revised by multiple members to ensure an accurate and well-written report. Individual lessons learned and reflections are listed below.

5.1 Suchitha Channapatna

This lab stressed the importance of computational efficiency. In addition to learning about particle filters, I was able to appreciate the need for a precomputed probability table and vectorized code. I also saw how to implement thread locks between two ros callbacks. I found this interesting as it combined topics from RSS and a concurrency class I am taking as well. As I reflect, I think our team made significant improvements in testing and analysis. We focused on keeping trials consistent and on computing the most informative quantities.

5.2 Luis De Anda

This lab definitely tested our ability to divide and conquer the tasks and capability to collect data. Thankfully our team is capable of assigning tasks based on our skills, but it shows the importance of having a diverse group to be capable of simultaneously accomplishing different tasks. When it comes to testing, the importance of establishing rules for test cases was noticeable. When trying to test the different noise models, the test we went with provided the least amount of variability to ascertain that only the noise models were impacting the particle filters performance. Just want to mention, the importance of practice when it comes to the presentations has become visible to me. In retrospect, I notice how the more I had practiced the content I wanted to go over, the more confident I was in presenting it. All in all, lab 5 was fun but still tested our ability to be efficient to accomplish the labs goals and show success.

5.3 Anastasiia Kutakh

This lab deepened my technical knowledge of localization algorithms. Working with particle filters showed me how important computational efficiency is for real-time applications. I found the vectorization techniques particularly useful for improving runtime performance. Our implementation successfully balanced accuracy with processing speed requirements. For the final lab and project, I plan to focus on completing technical components earlier while maintaining our rigorous testing approach. The filter convergence results were rewarding when our system performed well in the final demonstration. Overall, Lab 5 was challenging but provided valuable experience in both coding and system integration that I'll carry forward.

5.4 Amber Lien

In Lab 5, I learned much more about data collection using rosbags and data visualization and the importance of computational efficiency. Lab 5 was additionally a more challenging lab compared to previous labs and, as a result, we had less time to prepare for the briefing than usual. I've become more efficient in preparing for them as a result. Overall, Lab 5 was a challenging but rewarding experience. In the final lab and final project, I hope to be even more comfortable with presentations while aiming to have technical components completed more promptly.

5.5 Prince Patel

Technically, I gained insights into optimizing computational performance in robotics algorithms through vectorization and efficient data structures. This experience will be valuable for future projects requiring real-time processing. Regarding collaboration, I discovered the importance of creating comprehensive tests for each component before integration, which saved us significant debugging time during the final system assembly.