Lab 5 Report: Designing a Monte Carlo Localization System for an Autonomous Racecar

Team 7

Pranav Arunandhi
Thomas Fisher
Brady Klein
Calvin Maddox (editor)
Gustavo Ramirez

Robotics : Science and Systems

April 15, 2023

1 Introduction - Calvin Maddox

Localization is an important part of designing an autonomous system. In our labs thus far we have designed systems that function strictly based on information gathered directly at the time the system is being run, but localization allows for this behavior to be extended beyond the reactionary to being planned in advance of any actions taken.

In lab 5 we were tasked with devising an algorithm that given a map and initial pose for our robot could localize the robot's position relative to the map. We were able to accomplish this by synthesizing information from two major categories – sensor information and odometry information. We then aggregate the information from these models into a Monte Carlo particle filter to compute the final estimate of our robot's pose. For this lab we tested in two environments – one simulated, and one in the Stata Building basement, and we argue that our implementation is able to accurately localize our robot in both, and is robust to many different types of movement such as turns and switchbacks.

2 Technical Approach

2.1 Motion Model - Gustavo Ramirez

In our Monte Carlo Localization (MCL) particle filter implementation for the robotic racecar, the Motion Sensor component plays a crucial role in estimating the racecar's pose given the wheel odometry data. This section describes the implementation and key decisions made for the Motion Sensor component, including the choice of odometry source, noise addition, and motion model.

The wheel odometry derived from dead-reckoning integration of motor and steering commands was chosen over the Inertial Measurement Unit (IMU) data for the motion model. This choice is based on the observation that wheel odometry is less noisy than IMU data except under extreme operating conditions.

We obtain odometry data from the /odom and /vesc/odom topics as TwistWithCovariance messages, which express velocity in free space with uncertainty. The TwistWithCovariance message is defined as follows:

```
1 # This expresses velocity in free space with uncertainty.
2
3 Twist twist
4
5 # Row-major representation of the 6x6 covariance matrix
6 # The orientation parameters use a fixed-axis representation
    .
7 # In order, the parameters are:
8 # (x, y, z, rotation about X axis, rotation about Y axis,
    rotation about Z axis)
9 float64[36] covariance
```

From the TwistWithCovariance message, we extract the linear and angular velocities (dx, dy, and dtheta) to be used in our motion model. These values, combined with the accumulated pose in the global frame, allow the Motion Sensor component to update particle poses effectively.

We implemented a custom motion model that adds random noise to the deterministic model derived in Question 1 of Part A. The noise addition is essential for particle spread, ensuring the particle filter's effectiveness in estimating the racecar's pose. The noise coefficients define a normal distribution with standard deviation σ that will be added to each of the pose values, resulting in a more realistic representation of the racecar's motion.
The noise coefficients (σ_x, σ_y, and σ_θ) were empirically determined to achieve optimal performance.

The MotionModel class initializes with noise coefficients, providing methods for constructing transformation matrices and evaluating particle updates based on odometry data. The evaluate method adds noise to odometry, applies the noisy odometry to each particle, and updates particle poses accordingly.

The transformation matrix, T_{Wc}, is constructed using the current particle pose (x, y, θ):

$$T_{Wc}(x, y, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

The updated particle pose, T_{Wi_new}, is computed as:

$$T_{Wi\_new} = T_{Wi} \cdot T_{\delta} \tag{2}$$

where T_{Wi} is the transformation matrix for the current particle and T_{δ} is the transformation matrix for the noisy odometry.

The Motion Sensor component of our MCL particle filter implementation for robotic racecar localization accurately estimates the racecar's pose using wheel odometry data and a custom motion model with added noise. By carefully considering the odometry source, noise coefficients, and motion model, our implementation is both technically sound and efficient.

2.2 Sensor Model - Thomas Fisher

The sensor model component of the localizer assigns probabilities to the particles received from the motion model to determine how likely it is that each particle's pose is representative of the car's ground truth pose. The sensor model does this by using a probability distribution over several factors to calculate a likelihood for each particle which describes the certainty of whether the car would have produced the observed laser data from that particle's position. Figure 1 illustrates the goal of the sensor model visually.
Particles that receive a higher probability score and which should theoretically be nearer to the true position of the car are more likely to be chosen by the particle filter (see Section 2.3) as where the car believes itself to be. As time progresses over the course of a run, the noisy odometry from the motion model combined with the described selective probabilistic refinement results in a cohesive, accurate group of particles which represent a close approximation to the true location of the car on the map.

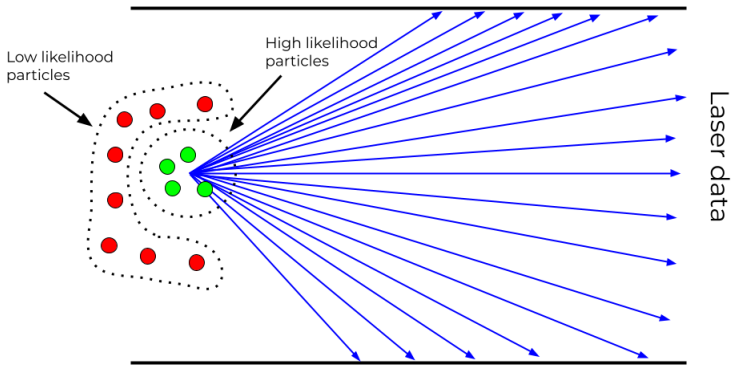

Fig. 1: The sensor model scores each particle on the likelihood of it having generated the observed laser data.

2.2.1 Probability Model

The probability model used is the sum of 4 separate components which each represent different ways that laser data can interact with the car's environment. A brief description of each component is listed below:

1. p_{hit}: Similarity between predicted range and measured range.
2. p_{short}: Distance of the measured range from the source.
3. p_{max}: Added weight to most distant range measurements.
4. p_{rand}: A small random value representing unexpected interactions.

Each of these probability components has a corresponding coefficient, and together their sum describes the distribution over which the predicted ranges (from the particles) and the measured ranges (from the laser data) are sampled:

$$p_{particle} = \alpha_{hit} p_{hit} + \alpha_{short} p_{short} + \alpha_{max} p_{max} + \alpha_{rand} p_{rand} \tag{3}$$

The equation above is a simplification in that it does not explicitly display all of the input dependencies for each of the probability components, however it illustrates the structure of the distribution well. Our team found that using the coefficient values below led to strong performance in our localizer.

$\alpha_{hit} = 0.74$

$\alpha_{short} = 0.07$

$\alpha_{max} = 0.07$

$\alpha_{rand} = 0.12$

2.2.2 Efficiency Improvements

The implementation of the sensor model provides several areas for improving performance of the localizer. The first relates to the probability model and involves pre-computing values to reduce computation at runtime (Figure 2). The second reduces the amount of laser data being analyzed through downsampling (Figure 3). Lastly, the sensor model is run at a lower frequency to reduce the number of relatively time-intensive cycles.

The laser data provided by the car's hardware produces range values which are always in the range $[0, z_{max}]$, and predicted ranges for the particles follow the same constraint. This allows us to know beforehand an approximate idea of what input pairs could be used to sample the probability distribution. We leverage this fact by pre-computing values from our probability distribution over a discretized grid of inputs. The effect of this approach is that during runtime we only have to look up the computed probability value with our $z_k^{(i)}$ and d values as table indicies instead of performing a calculation.

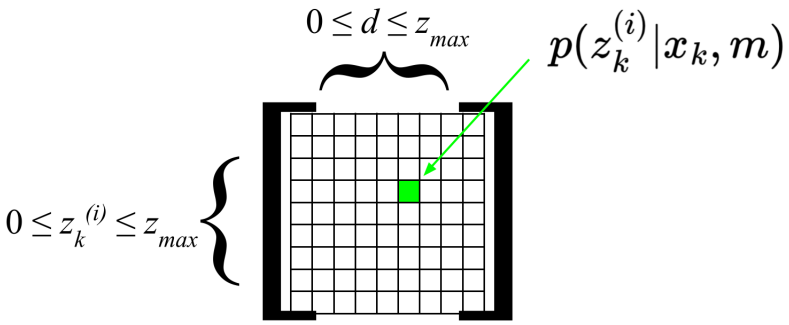

Fig. 2: Pre-computed probability values over a discretized grid of inputs.

A second, more low-level strategy to reduce the amount of work performed during runs is to downsample the data by reducing the observed ranges our algorithm considers. Our sensor model reduces the number of laser beams to be equal to the number of particles, which was about a factor of 10 for the values used in the lab. As each particle must compute its likelihood based on all observed laser ranges, this reduction in necessary computation is significant for a component which runs at 20 Hz. Figure 3 below gives a sense of how downsampling from very dense data leaves the key features of the observations intact.
A last optimization comes by running the sensor model at 1/10 the frequency of the motion model due to the higher number of operations required per cycle of execution by the sensor model. Letting the motion model run many times in a row without the sensor model is an acceptable drawback to make, as the sensor

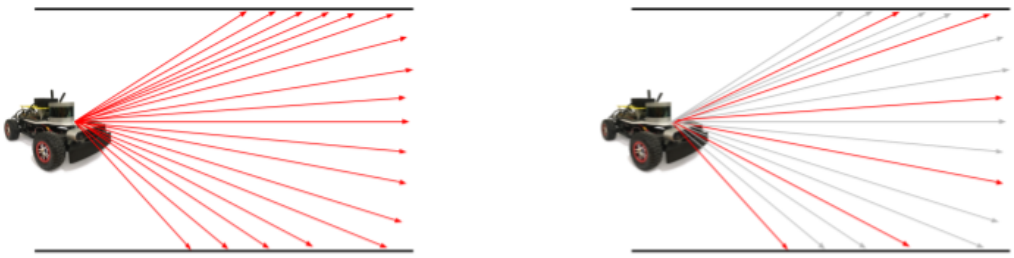

Fig. 3: Downsampling laser data (dropped observations in gray) reduces required computation.

model will begin assigning probabilities to the particle poses (which will likely have begun to stray) from the cycle it picks up on and correct accumulated drift.

2.3 Particle Filter - Pranav Arunandhi

The particle filter provides an estimate of the racecar's position and heading. This is achieved by maintaining a set of particles representing a running set of estimated poses for the racecar - which the motion model and sensor model update in tandem - and then using the techniques described in Section 2.3.1 to take a representative average of the particles to determine an estimated pose for the racecar, a process overviewed by Figure 4.

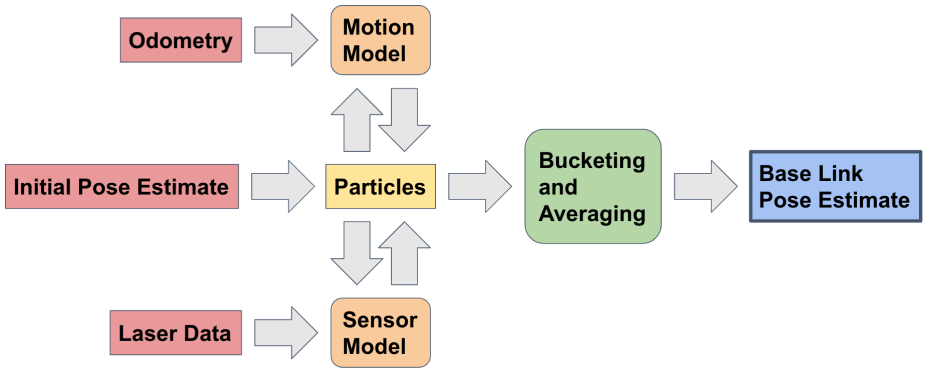

Fig. 4: An overview of the flow of information in the particle filter.

The particle filter subscribes to the the `/initial_pose` message, which is published to by RViz when the user selects the "2D Pose Estimate" button and clicks

a point and heading on the map that is relatively close to where the racecar is in real life. This provides the particle filter with a tuple of 3 values representing the `(x, y)` position and the heading `theta` of the racecar. Following this, a set of particles is generated by adding noise to the original pose using independent normal distributions with standard deviations of .25 for the position dimensions and .5 for the angular dimension.

This set of particles is used by the motion model when a message is received on the `/odom` topic; as described in Section 2.1, the particles are propagated forward in time to reflect the motion of the racecar - this directly updates the stored set of particles. The same set is also used by the sensor model when a message is received on the `/scan` topic to provide each particle a probability of being the true pose of the racecar based on the LIDAR scan data - these probabilities are rescaled to sum to 1, and treated as a distribution. This distribution is then used to resample the poses, with noise being added with standard deviations of .1 for the position dimensions and .5 for the angular dimension. Due to the sensor model being very computationally intensive, we chose to only run the sensor model on every 10^{th} LIDAR scan, as it is most effective at re-concentrating the points that have been spread out by the noise introduced in the motion model.

2.3.1 Pose Averaging Algorithm

Using the maintained set of particles, we want to provide an estimate for the racecar's pose. We noted the possibility of the distribution of particles being multimodal in any of the given dimensions, in which case a simple average may not necessarily provide an accurate estimate of the racecar's position in that dimensions; see Figure 5 for an example of a failure mode with taking the average of the entire distribution of points.
To get around this, we introduce the concept of buckets, which discretizes our data into fixed-width sections in each positional dimension; see Figures 6 and 7 for examples of bucketing in 1- and 2-dimensions. We chose to not discretize in the heading dimension because the distribution in that dimension tended to not be as widely spread out, especially within each bucket, and would not have significantly improved the performance of the averaging algorithm.

After bucketing the data, we then select the subset of points that are found in the most densely populated bucket to average to form a pose estimate. For the positional dimensions, we used a standard averaging algorithm along each dimension; for the heading dimension, we used the Mean of Circular Quantities algorithm:

$$\bar{\alpha} = \arg\left(\sum_{j=1}^{n} e^{i\cdot\alpha_j}\right). \tag{4}$$

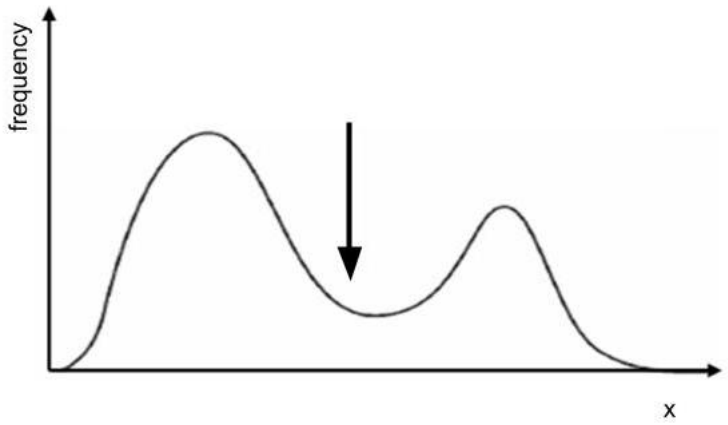

Fig. 5: A multimodal distribution, with the position of the simple average indicated with an arrow; notably, this average is not well representative of the data points with high frequency.

We recognize that bucketing may still lead to suboptimal pose estimates depending on the distribution of the points - for example, it is possible that certain discretizations could lead to a cluster of points being split over multiple buckets. However, this can be circumvented by using sufficiently large buckets such that this is unlikely, and even in the case that it does occur, there should be sufficiently many points populating the same bucket such that the estimate is reasonably close to the expected optimum. However, using buckets that are too large could re-introduce the problem of multimodal distributions. By balancing these factors, we found that a bucket size of .5 provided the desired performance.

Many design decisions were made by comparing their performance against quantitative and qualitative benchmarks. We go into this further in section 3.

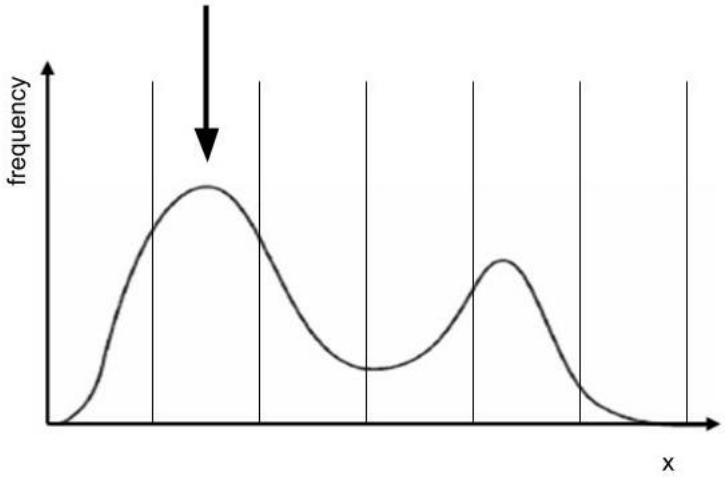

Fig. 6: A bucketed multimodal distribution, with the position of the average of the most densely populated bucket indicated with an arrow; notably, this average is more representative of the data points with high frequency.

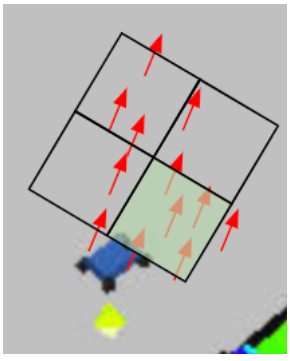

Fig. 7: A bucketed multimodal distribution, with the position of the average of the most densely populated bucket indicated with an arrow; notably, this average is more representative of the data points with high frequency.

3 Experimental Evaluation - Brady Klein

Upon successful implementation of the motion model, sensor model, and particle filter, our team employed a variety of methods in order to gauge the performance of our system both qualitatively and quantitatively. This section describes those methods and presents evidence in support of the conclusion we drew from this lab.

3.1 Random Driver

In order to begin testing our implementation, our team needed to determine a method to navigate the car in simulation and run localization on the navigation. We ultimately constructed a ROS node that would drive the racecar along a quasi-random path. In order to achieve this, we reused much of the framework from the parking controller in lab 5. The main change we made to this code was that instead of coming to a stop in front of the cone, the car would continue to drive straight and then loop around before making another pass by the cone.

Another idea our team had before deciding on this approach was to construct a "random walk" algorithm that would randomly alter the steering angle of the car on every iteration of the node's callback function. What ultimately deterred us from this implementation was that we wanted to be able to record a rosbag on the order of about one minute and this method would almost guarantee our car running into a wall before the one minute mark. This consideration is what led us to the more contained and cyclic implementation detailed previously.

Once this random driver was complete, our team was able to record rosbags which would allow us to analyze quantitatively the performance of our particle filter.

3.2 Gradescope Tests

The first assessment of our particle filter localization was by submitting our code to the RSS Gradescope assignment. We can see one of the tests that the autograder ran in figure 8. While there was definitely some noise in these tests, our team was extremely satisfied with how our system performed in this test suite.
One odd feature of these tests that we noticed was that our system appeared to be noisiest on the long, straighter sections near the middle of the path in figure 8. While we are not fully certain of the cause for this noise, one hypothesis that we had was that on these longer stretches of the path when the lidar scans may be returning similar scans at each time step, our localization algorithm likely has multiple particles that have similar probabilities of producing that scan data. Although we do not have a concrete answer for this noise, its magnitude was small enough in the scope of the system as a whole. This meant that the system

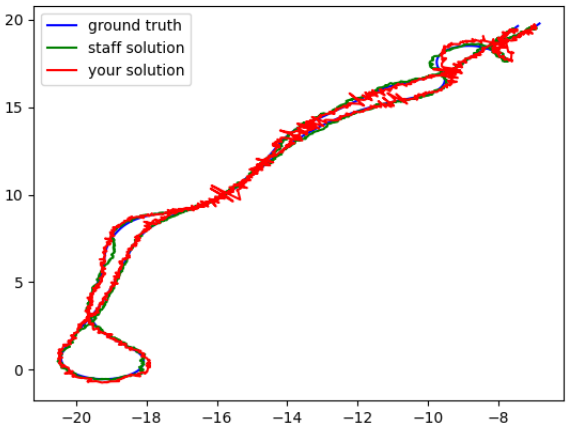

Fig. 8: Overlay of the traveled path, our system's localization, and the staff solution in Gradescope

was able to return to a steady state and the noise did not affect its ability to locate the car in later sections of the path.

3.3 Euclidean and Angular Error Measurements

With the random driver implemented and sufficient performance on the gradescope tests, our team was ready to implement our own error measurements in the simulation. For this lab especially, the simulation proved dramatically easier to evaluate our system's performance because we were able to subscribe to the car's ground truth pose. With this information, we could easily compare it with the localization system's estimation of the car's position.

Our team used two main metrics to evaluate the performance of the system: Euclidean distance and angular error. Because the particle filter relies on random noise, we varied how we defined the noise value in order to find the best performance. We defined a "low noise" value of 0.0025 and a "high noise" value of 0.005.

In figures 9 and 10 below, we can observe the Euclidean error of our system with low and high noise respectively. We found that the higher noise value produced better results with a mean of 0.329 meters and a standard deviation of 0.200 meters. The low noise test was not far off with a mean of 0.560 meters and a standard deviation of 0.374 meters. From these results, we concluded that our system's performance increased as noise decreased, however in both cases

the magnitude was small enough to produce effective localization. The spikes in this data is likely a product of the probabilistic nature of the particle filter algorithm.
We next used these same noise values to evaluate our system's angular error in figures 11 and 12 below. Similar to the results of the Euclidean measurements, we found that the system performed better with the higher noise value. With high noise our system had a mean error of 0.035 radians and a standard deviation of .042 radians. With low noise our system had a mean error of 0.046 radians and a standard deviation of .044 radians. Again we observed spikes in our data although less regular than in the Euclidean error. We can again contribute these spikes to the probabilistic nature of the algorithm.
An interesting takeaway that we had from this testing was that the noise values we used for our own testing differed from the value we used for the gradescope testing. From this we hypothesize that the ideal noise value is dependent on the environment that the car is in.

3.4 Visual Assessment in Real World

The next step of our assessment process was to observe our particle filter node running in a real world environment. Before analyzing any quantitative data in this environment, we first analyzed the performance of the system in this environment qualitatively. After some parameter adjustments, our team determined that the results from navigating in a physical environment were quite satisfactory. While difficult to quantify, these qualitative observations of the particle filter played an imperative role in our team's adjustments of the system and was arguably more helpful than any of the other metrics we observed because we could receive feedback that was instantaneous, reproducible, and intuitive. Because of this, the visual assessment was the first metric our team turned towards whenever we tested in a new environment, either physical or simulated.

3.5 Real World Measurements

To compute quantitative error metrics of our localization model in the real world, our team developed the following process. Launch the localizer in the Stata basement. Use our initial pose sub subscriber to get coordinates for a start and endpoint of a straight path. As the car navigates this path, record a rosbag listening to linear error. Use our rosbag utility file to plot this error.

With this process we could compute the distance between the car's estimated position and the path in real time. We decided to follow a straight path because this would alleviate the human error that would arise in driving the car.

Despite developing this process, our team ultimately ran into hardware issues and were unable to collect this data. In preparation for the final challenge, we hope to be able to revisit this topic.

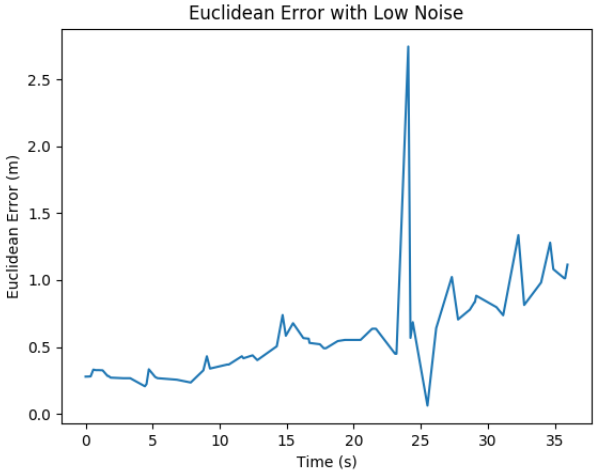

Fig. 9: Linear error in a simulated environment with motion model noise set to 0.0025.

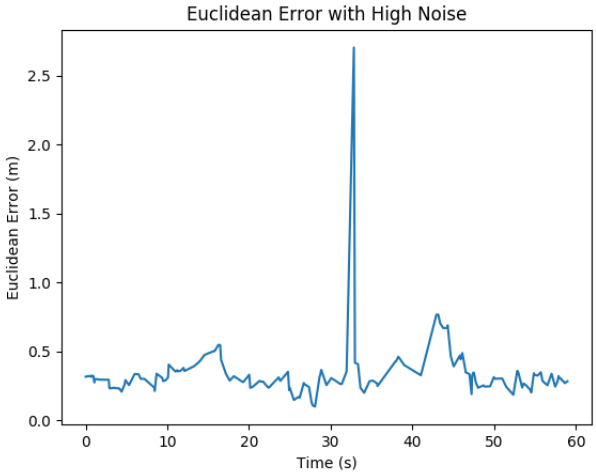

Fig. 10: Linear error in a simulated environment with motion model noise set to 0.005.

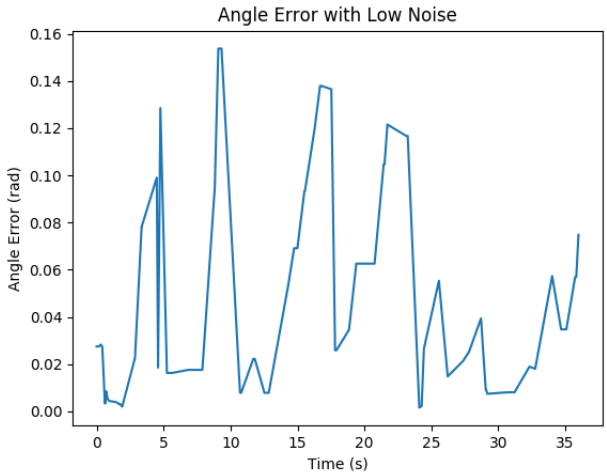

Fig. 11: Angular error in a simulated environment with motion model noise set to 0.0025.

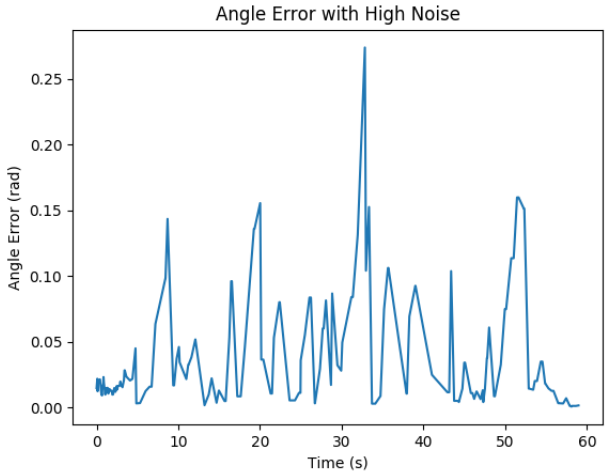

Fig. 12: Angular error in a simulated environment with motion model noise set to 0.0025.

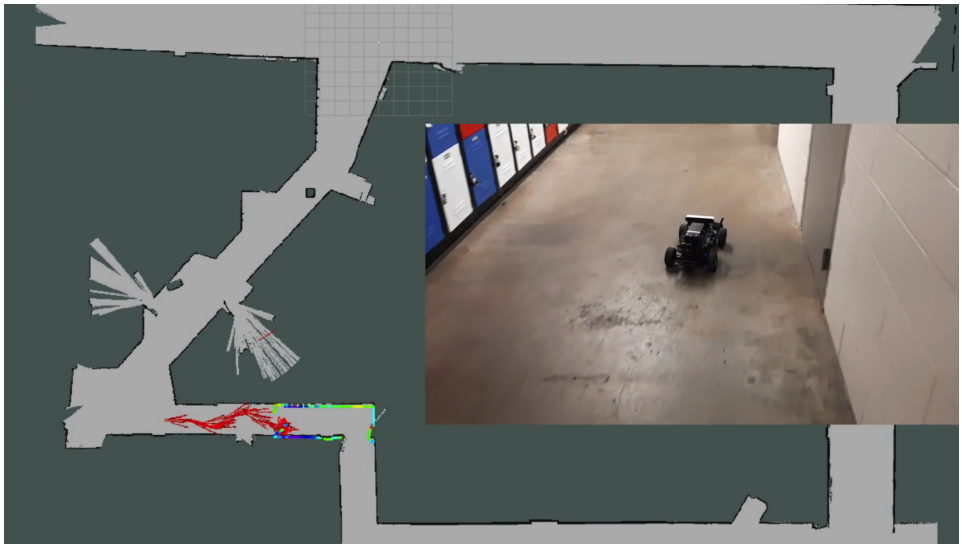

Fig. 13: Screen capture of our particle filter running in the Stata basement while being manually driven

4 Conclusion - Calvin Maddox

In this lab we successfully implemented a localization algorithm that given a map and initial pose of our robot is able to continuously localize the robot within the provided map. We use two models – an odometry model and a sensor model – in conjunction with a Monte Carlo particle filter to achieve this, and argue that based on our results in the simulated environment as well as preliminary real-word confirmation, this algorithm can reliably succeed in this task.

In addition to the evaluation metrics we have presented in this report, there are a few additional metrics we would like to add, given additional time. First would be additional testing in the real-world environment. While the simulation is able to provide a consistent testing environment for confirming the efficacy of our algorithm, there are additional factors that can arise when testing on a physical robot such as errors in sensors, and we would like to see how resilient our robot is to errors like this. Second would be parameter tuning, especially in regards to the amount of noise added to the odometry model, in order to continue optimizing the accuracy of our algorithm.

Even without these additions however, we still have implemented a robust algorithm capable of performing reliable localization, and are ready to integrate it into the next lab in order to perform autonomous path planning on the robot.

5 Lessons Learned

5.1 Pranav

This lab helped me develop my debugging skills, as there were a lot of ways for our code to appear as though it was working in some cases and fail in other cases. It also required me to be creative when approaching the various technical issues that we needed to solve in order to develop a high-functioning solution. The division and parallelization of work across the different tasks, along with the necessary integration at the end of the pipeline, necessitated that we communicated our progress effectively, so that everyone was on board with what was happening.

5.2 Thomas

Lab 5 presented unique challenges in coordinating team member schedules with spring break splitting the start and end of the lab. Our team relied on group planning before break and strong communication after our return to ensure an uninterrupted workflow and a clear understanding of the statuses of different lab components as deadlines approached. While it was sometimes difficult in the moment, this kind of complex time/effort coordination is a skill I'm happy to have improved with my teammates during this lab.

5.3 Brady

This was a lab that I felt necessitated a team effort. The problem of localization is extremely challenging. While this is an area of autonomous vehicles that I had never really put much thought into, it is an invaluable part of the overall system we are building. I felt that this lab really showcased our team's ability to work towards an end goal and deliver a functional result. This lab highlighted how important communication and collaboration are to delivering in any team environment. I feel like our team grew in our ability to produce code, test it, and optimize toward an end goal.

5.4 Calvin

More than some others, I felt this lab really tested our skills at being able to integrate work from different team members as while we had a clear division of labor, each component had to work in tandem with the others and so was crucial to the performance of the algorithm as a whole. Additionally, I felt that this lab also tested our ability to test our code, as there were additional failure modes that we wanted to catch, and required additional thinking on our part to properly evaluate.

5.5 Gustavo

This lab gave us the opportunity to strengthen our coordination and communication skills as a team. As each member was responsible for different components of the project, we had to work cohesively to integrate these components into a unified system. The challenge of achieving this on a tight timeline, broken up by spring break, necessitated efficient communication and collaboration among team members. While we were each mainly responsible for one component of the project, this lab showed me importance of developing a thorough understanding of each component of the overall system.