Lab #5 Report: Monte Carlo Localization

Team #15

Lauren Carethers
An Bo Chen
Brian Li
Claire Lu
Rachel Lu

RSS

April 15, 2023

1 Introduction

Author(s): Lauren

The purpose of this lab is to successfully determine our robot's position (otherwise known as its pose) in a known environment through the implementation of Monte Carlo Localization (MCL). Localization is critical to include in the racecar's programming such that the car can a) know its current location and b) use this knowledge to make future actions.

MCL is an algorithm that uses particle filtration to find a distribution of possible poses. At initiation, the configuration space is filled with a random distribution of particles. Each time the sensor is activated, particles (or poses) from this space are passed through a Bayes Filter which recursively reduces the particles in the space until they converge with the actual position of the robot.

The racecar is fit with sensors that allow us to determine both the state of our car and the environment its contained within. To solve MCL and localization in general, we must utilize odometry, which uses the cars input sensor data to estimate the change in its position over time.

Specifically, the odometry utilizes the car's previous pose and integrates control and proprioceptive measurements to determine its current pose. The calcu-

lation of odometry features uncertainty modelling which incorporates a pose belief $P(\mathbf{x}_k|\mathbf{u}_{1:k})$ — the probability distribution over possible poses given the available measurements — and a motion model as a conditional probability $P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k)$ — typically determining a pose belief based on the prior pose of the robot, where $\mathbf{x}_k$ is defined in eq. (1).

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \epsilon_k) \tag{1}$$

Using these concepts, our team set out to accomplish the following:

1. Implement MCL in the racecar simulator
2. Augument the simulated odometry with various types of noise and compare it against the ground truth odometry
3. Adapt our implemented MCL to work in our car
4. Conduct experimental analysis of our MCL's performance to verify its accuracy

2 Technical Approach

Our MCL incorporates a motion model, sensor model, and particle filter to determine our car's pose.

2.1 Technical Problem

2.2 Monte Carlo Localization

2.2.1 Motion Model

Author(s): Lauren

Our motion model determines the probable future poses of the racecar. It works by executing the following process for all of the poses provided at each scan:

1. Converting the input odometry vector $\Delta\mathbf{x}$ written out as $[dx, dy, d\theta]$ into the world frame $\Delta\mathbf{x}_R$ so that it can be directly applied to the last known pose $[x_0, y_0, \theta_0]$, where R is the rotation matrix
2. Applying that transformed odometry to the car's last known pose to determine the new probable pose $[x, y, \theta]$

3. Adding randomized Gaussian noise to the new probable pose for each particle

The formulae used for these calculations are shown in eq. (2), eq. (3), and eq. (4) below.

$$\Delta \mathbf{x}_R = \Delta \mathbf{x} * R \tag{2}$$

$$R = \begin{bmatrix} cos(\theta_0) & -sin(\theta_0) \\ sin(\theta_0) & cos(\theta_0) \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \Delta \mathbf{x}_R + \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} \tag{4}$$

2.2.2 Sensor Model

Lookup Table

Author(s): Rachel

After updating particle positions using the motion model, the sensor model was used to assign likelihood weights to each particle. The probability of each particle was calculated based on the laser scanner data, which is the type of sensor used in our implementation. These probabilities were used to resample the particles, giving particles with higher probabilities a better chance of being selected in the next iteration. This process ensures that the particle estimations are continuously improved with better hypotheses, resulting in a more accurate estimation of the racecar's position. However, computing the probabilities of each particle at every iteration can be computationally expensive, involving complex functions. To improve efficiency, we created a probability lookup table with pre-computed probabilities for discretized inputs.

To construct the lookup table, we used eq. (5) which defines the likelihood of recording a specific sensor reading z_k from a given hypothesis position x_k in a known static map m at time k.

$$p(z_k|x_k, m) = p(z_k^{(1)}, ..., z_k^{(n)}|x_k, m) = \prod_{i=1}^{n} p(z_k^{(i)}|x_k, m) \tag{5}$$

The lookup table was constructed by discretizing the range values. The input of the lookup table is the measured distance $z_k^{(i)}$ and the actual distance d, and it returns a probability. To calculate the probability, we used eq. (6) and eq. (7). The different probabilities are computed using eq. (8), eq. (10), eq. (11), and eq. (9). The grid of input values $z_k^{(i)}$ and d were discretized, and the probabilities were stored in a 200 by 200 lookup table. The rows of the lookup table correspond to the z values, and the columns correspond to the d values.

$$\alpha_{hit} + \alpha_{short} + \alpha_{max} + \alpha_{rand} = 1 \tag{6}$$

$$\begin{aligned} p(x_k^{(i)}|x_k, m) = \alpha_{hit} \cdot p_{hit}(z_k^{(i)}|x_k, m) + \\ \alpha_{short} \cdot p_{short}(z_k^{(i)}|x_k, m) + \\ \alpha_{max} \cdot p_{max}(z_k^{(i)}|x_k, m) + \\ \alpha_{rand} \cdot p_{rand}(z_k^{(i)}|x_k, m) \end{aligned} \tag{7}$$

$$p_{hit}(z_k^{(i)}|x_k, m) = \begin{cases} \eta \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(z_k^{(i)} - d)^2}{2\sigma^2}) & \text{if } 0 \leq z_k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$p_{short}(z_k^{(i)}|x_k, m) = \frac{2}{d} \begin{cases} 1 - \frac{z_k^{(i)}}{d} & \text{if } 0 \leq z_k^{(i)} \leq d \text{ and } d \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$p_{max}(z_k^{(i)}|x_k, m) = \begin{cases} 1 & \text{if } z_k^{(i)} = z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$p_{rand}(z_k^{(i)}|x_k, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

When calculating the probabilities for each z and d value, the calculated p_{hit} values are normalized across columns of d values, and the columns of the lookup table itself are also normalized. This normalization ensures that the probability distribution function has a sum of one for all the probabilities. It is crucial to ensure that the resampling and probabilities computed in the evaluation function work as intended. After constructing the lookup table, it can be used in the evaluate function.

<u>Evaluate</u>

Author(s): Claire

After pre-computing the lookup table, we use pairs of simulated and actual distance values to look up probabilities from the table and then evaluate the final probabilities of a given list of particles. First, we simulate 100 laser scans from a given particle's position and orientation in addition to the actual scan data that we are given. Then, we convert both the simulated and actual scan data into pixel units, and clip all values to be between 0 and the width of our table (200), which is the maximum pixel value. Then, we iterate through the given list of particles. For each particle, we further iterate through the lists of simulated and actual scan data, index into our table with each (simulated, actual) pairs of data, and multiply each resulting probability value together to get the final probability for that particle.

The simulated scans correspond to observations, z, and actual scans correspond to true distances d, so we used the simulated scan value as the row index and actual scan value as the column index to access cells in our table. For instance, if our simulated scan data for one particle was $[0.6, 0.7, ...]$ and actual scan data was $[0.5, 0.3, ...]$, we would compute the probability by doing sensortable[0.6][0.75] $\cdot$ sensortable[0.7][0.3] $\cdot$ sensortable[..][..] $\cdot$ ... and so on.

The probability of each pair of (simulated, actual) scan data being produced by that particle is independent from all of the other pairs. Since these probabilities are independent, if we want the probability of one particle producing all of the distance pairs, we multiply the individual probabilities together. We use this process to compute the probability of each particle, and return the entire list of particle probabilities as the final output of the sensor model.

2.2.3 Particle Filter

Initialize Particles

Author(s): An Bo

Once we have both the motion model and the sensor model, we use them to implement a complete MCL algorithm in particle filter. However, before that, we need to generate a random spread of particles around an initial pose. Without this type of initialization, implementing and reasoning about our localization would become difficult as we would not have any particles to work with.

From the initial pose, we can obtain an initial position, orientation, and a covariance matrix. Using the initial position, we can calculate the initial values of x and y. With the initial orientation represented as a quaternion, we can use the formula that converts quaternions to a yaw to compute the initial value of theta. Let q_x, q_y, q_z, and q_w be the variables representing the corresponding components in the quaternion. Then, the formula to calculate theta is as

follows:

$$\theta = \arctan 2(2 \cdot (q_w \cdot q_z + q_x \cdot q_y), 1 - 2 \cdot (q_y^2 + q_z^2)) \tag{12}$$

The covariance matrix takes the following form:

$$C = \begin{bmatrix} \sigma_x^2 & ... & ... & ... & ... \\ ... & \sigma_y^2 & ... & ... & ... \\ ... & ... & ... & ... & ... \\ ... & ... & ... & \sigma_{q_z}^2 & ... \\ ... & ... & ... & ... & \sigma_{q_w}^2 \end{bmatrix} \tag{13}$$

Using the covariance matrix, we can obtain standard deviations for the initial values by taking the square root of the corresponding elements along the diagonal of the matrix. For our implementation, we use σ_{q_w} as the standard deviation for the initial theta value since the other σ_q^2 elements are zero.

We then generate the desired number of initial particles, typically around 200, to keep our particle filter performant. For each particle, we calculate the corresponding x, y, and θ values from a normal distribution with the mean as the initial value and the standard deviation from the covariance matrix.

Locking

Author(s): An Bo

An important implementation detail in our particle filter is the use of thread locks in the main body of the MCL algorithm. A lock is a synchronization object that allows only one thread to access a shared resource at a time. This prevents multiple threads from accessing the same data simultaneously, which can lead to race conditions and other concurrency issues. If the motion model and sensor model are updating the list of particles independently, we need to ensure that the models do not conflict with each other and modify the same data at the same time.

Sensor Model Resampling

Author(s): An Bo

At the beginning of each iteration of the particle filter, we resample our current particles from a previous particle probability distribution. However, at the very first iteration, we can only use the initial random spread of particles since we have not generated the particle probabilities yet. As we resample, the particles with higher associated probabilities will appear more often in our set of particles, improving our estimated pose predictions as time goes on.

Motion Model and Sensor Model Update

Author(s): Brian

In order update the positions of our particles, we use our motion model, which takes in the current particles and odometry received from the car and returns the updated particles. We also update the probability associated with each particle by sending our current particles and range data from the most recent LiDAR scan to the sensor model. After these two steps are complete, we save a copy of the updated particles and probabilities since we will use them in the next iteration of the algorithm.

Estimated Pose

Author(s): Brian

Finally, to get the estimated pose of the car, we take a weighted average of all the particles, where the weights, w_i, are the normalized probabilities associated with each particle i as shown in eq. (14).

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{\theta} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & ... & x_n \\ y_1 & y_2 & ... & y_n \\ \theta_1 & \theta_2 & ... & \theta_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ ... \\ w_n \end{bmatrix} \tag{14}$$

We then publish this average pose to both an Odometry and a Transform message, making sure to convert Euler angles to quaternions when appropriate.

3 Experimental Evaluation

3.1 Simulation Testing

3.1.1 Error Testing

Author(s): Claire Lu

In simulation, we were able to test our particle filter by comparing the trajectory that it computed with the actual trajectory taken, and by quantifying the difference between our estimated pose and the ground truth pose.

First, we were able to compare our estimated trajectory with the actual trajectory taken. To plot our estimated trajectory, we published a transform that contained the x, y, and θ of our estimated pose. The Gradescope autograder

then plotted our trajectory against the ground truth and staff solution under three different noise conditions, which can be seen below in Figure 1.

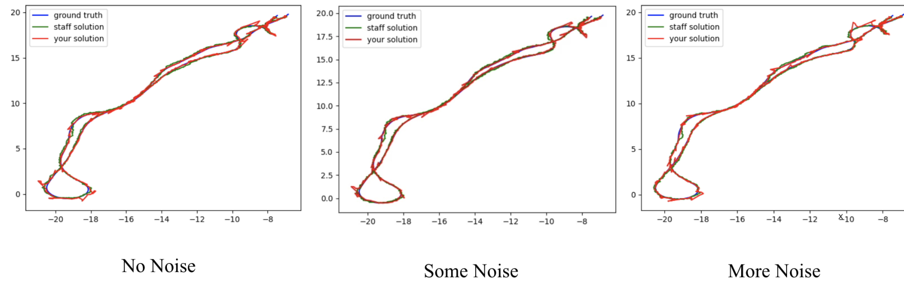

Figure 1: Plotted Trajectories vs. Ground Truth in Simulation

As shown in the plots, our estimated trajectory almost perfectly traces the ground truth trajectory. This demonstrates that our estimated pose remained accurate to the real pose as the car moved around the map. There is some jaggednesss in our trajectory across all three noise conditions, especially on turns and loops, which likely represents a delay in updating our calculated θ compared to how quickly the car is actually turning. Nonetheless, our estimated pose in simulation is very accurate to the real pose.

In addition to the plotted trajectories, we computed cross-track error while the car ran in simulation. We defined cross-track error as the component-wise difference between the ground truth position and our estimated position. We found the ground truth position by listening to the transform published between base_link (the robot) and map (the world frame). We then found the difference between the x and y components of the transform with our own estimated pose, and published this error in a custom PoseError message that we made. Then, we plotted the error over time in Figure 2.

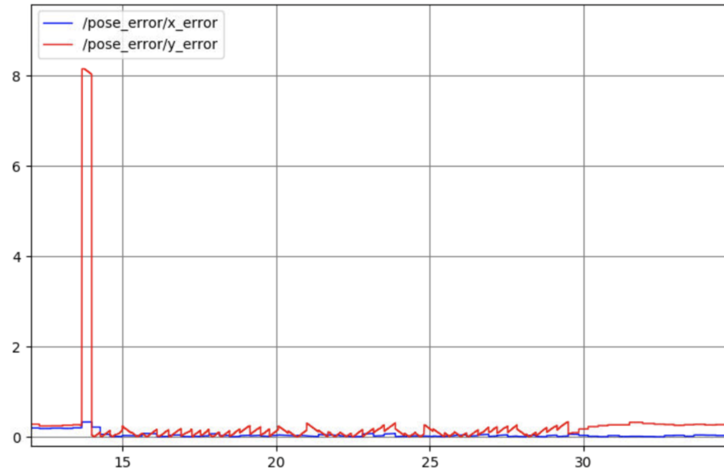

Figure 2: Cross Track Error of the Racecar in Simulation

We can see a large initial spike in y_error, then both x_error and y_error remain very close to zero for the rest of the car's movement. This large initial spike is due to the fact that we start with a stationary initial pose estimate, and this stationary estimate may be quite off of the actual position. However, once the car starts moving and we collect fresh odometry and sensor data, we see that the error quickly drops to near zero and stays there until the car stops moving (around 30 seconds). We can see that the x error is essentially zero while the y error exhibits a sawtooth shape close to zero. We hypothesize that the sawtooth shape is due to both the periodicity of data collection and pose calculation, and the fact that we are continuously recalculating the estimated pose. Since we are not updating our estimated pose in real time, we can see a "build-up" in error as our old estimate gets further and further from the current true position until we thread lock again and re-update our estimated position. This creates the peaks in error, and the sharp dips down again are when we re-update our estimated pose and it is correct at that exact moment in time. Overall, though, our error stays consistently close to zero, further proving that our particle filter is extremely accurate in simulation.

3.2 Physical Environment Testing

3.2.1 Convergence Testing

Author(s): Brian Li

In addition to analyzing cross track error from our error graphs, we also looked at convergence of our particle filter by defining a metric called effective sample size (ESS) as shown in eq. (15) where w_i is the probability associated with particle i.

$$ESS = \frac{1}{\sum_{i=1}^{n} w_i^2} \tag{15}$$

When the ESS is small, this indicates that our particles are not very diverse and somehow correlated with each other since some particles have much higher weights. This means the particles may not be exploring the full state space or accurately representing the posterior distribution, which in both cases indicates the filter is not converging to the true pose of the car. When the ESS is large, our particles are diverse and relatively independent of each other, which corresponds to faster convergence. Thus, since we want our particle filter to converge quickly, a higher ESS is desirable. Note that ESS is always a number that ranges between 0 and the number of particles in our filter. For our convergence test, we chose to drive our car around an open corner in Stata basement and record ESS values every time the particle filter published a new estimated pose.

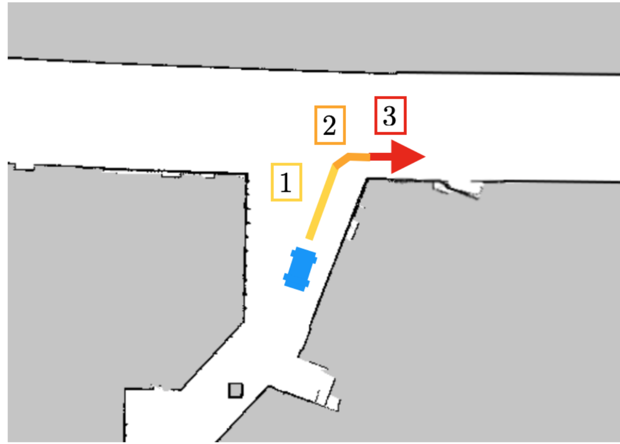

Figure 3: Map of open corner in Stata basement. Numbers correspond to different regions in the ESS plot below.

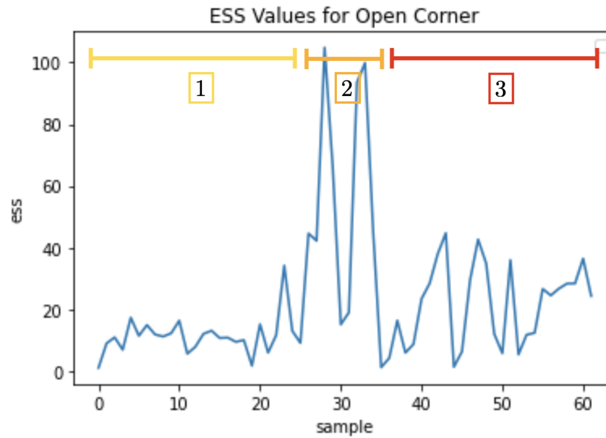

Figure 4: ESS values plotted from the initiation through completion of the turn

From this test, we can see that as our car approaches the corner, the ESS is around 10-15 particles. As we turn the corner around sample 30, the effective number of particles peaks at around 100. Then, after rounding the corner, the ESS drops down to around 40 particles on average. This is roughly what we expected and makes sense since as the car approaches and leaves the corner, it's driving in a relatively simple hallway. The filter is fairly confident in the car's pose which is why the particles are similar to each other. However, as the car rounds the corner, the number of features in the environment increases, so the particles diversify to account for decreased confidence in the car's true pose.

3.2.2 Visualization

Author(s): Rachel

One method to evaluate the success of the localization algorithm on the physical racecar is through visual inspection while the program is running. Figure 4 displays a snapshot from a video illustrating our particle filter in action on the physical racecar. The image showcases the known map of Stata, the particle distribution visualization, the inferred position visualization, and the laser scan data in the inferred position's coordinate frame. The red particles surrounding the racecar represent the particle distribution, and the blue arrow indicates the inferred position. As the racecar moves around Stata, the laser scan data, particle distribution, and inferred position all update dynamically based on the racecar's physical position.

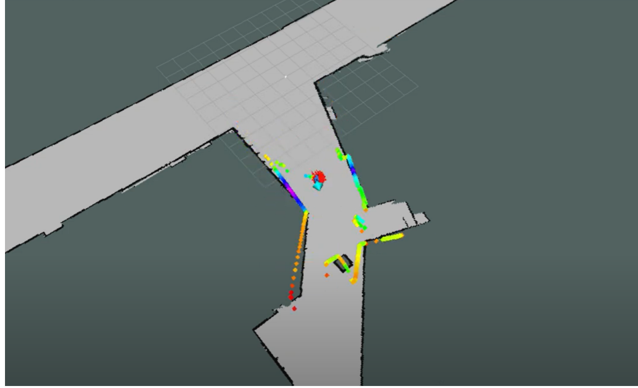

Figure 5: Visualization of the racecar in RViz for the localization test on the physical racecar. The lidar scans line up well with the walls and the estimated position matches well to the actal position.

The visualization confirms that our localization algorithm is functioning properly and accurately. Our lidar scan data correlates well with the map of Stata, and the inferred position closely tracks the position of the physical racecar. This, combined with the numerical evidence, provides strong support for the particle filter's ability to effectively and efficiently determine the racecar's location using the MCL method.

4 Conclusion

Author(s): Rachel

4.1 Accomplishments

In summary, our implementation of the MCL algorithm has enabled our program to determine the racecar's location with reasonable accuracy in both simulation and the physical environment. We achieved this by developing a motion model to predict particle evolution over time during resampling and a sensor model that uses a lookup table to determine the probability of particles being in their respective location. The motion model updates the particles based on odometry readings. While our testing in simulation and the physical environment demonstrated that our particle filter operates as expected, there are still some limitations and areas for improvement in the future.

4.2 Next Steps

4.2.1 Initial Pose

To begin running our particle filter, we must first initialize the particles. Currently, we accomplish this by using an initial "guess" of the robot's location, which works well in simulation but is less effective for the physical racecar. During testing, it proved challenging to determine the racecar's initial position accurately, and any discrepancies between the initial position and the initialized particles resulted in localization errors. To address this issue, we need to improve how we determine the initial location of the racecar and the corresponding particles. Instead of placing the racecar in a position and guessing the corresponding location on RViz, we can try to choose a well-defined location on RViz and place the racecar in that position in the physical environment. Additionally, we can try to identify the origin position (0, 0, 0) and place the racecar there. A better method of determining the initial position will enhance the accuracy and robustness of our algorithm when running on the physical racecar.

4.2.2 Path Planning Integration

Now that we have a functional MCL algorithm, we can integrate it into our path and motion planner. Determining the precise physical location of the racecar at any given time is a critical component of a path planning algorithm. Accurate localization of the racecar is essential for autonomous operations, planning, and control. It enables us to determine how far the racecar is from a designated goal and allows the path finding algorithm to correct itself when the racecar deviates from the desired path. Localization is an indispensable aspect of achieving our ultimate objective of having a racecar drive autonomously with maximum efficiency and effectiveness.

5 Lessons Learned

Lauren:

When working on the motion model in Lab parts A and B, I learned more about the importance of frame rotation and orientation when calculating a particle's new pose. While the execution proved to be rather simply for Lab part A, it took me a while to fully comprehend what vectors I needed to transform and how to transform them. Thus it was helpful to talk with my teammates about this portion. In terms of collaboration, one difficulty we had was timing given that the majority of us went away for spring break. We front-loaded the

simulation part of the lab before break, while our teammate that stayed at MIT during spring break began working on implementing our particle filter. Once we returned, we worked to get on the same page and troubleshoot issues.

Brian:

The big takeaway I had from this lab is that it's one thing to implement an algorithm but figuring out how to test and evaluate the performance of that algorithm is a challenge in and of itself. When we first started the experimental evaluation portion of this lab, I wasn't sure how we would test things like the convergence rate of our particle filter since we didn't have ground truth available for the actual racecar. It was only after doing some research and brainstorming among the team that we came up with the idea of using ESS as a convergence metric. In hindsight, ESS is far from perfect and there are definitely other, better, methods to do this test. However, I thought it was pretty cool to see how the ESS metric confirmed some of our intuitions about the behaviour of our particle filter, such as how the particles diversify when the car turns corners. I also greatly enjoyed working with my teammates over the course of this lab. In the beginning, we had some challenges with our particle filter in simulation, and it was great to see how we were able to offer each other a fresh set of eyes when figuring out bugs in the code as well as correct each other's conceptual misunderstandings about the particle filter.

An Bo:

In this lab, I learned how to implement a MCL algorithm for the particle filter by utilizing a motion model and sensor model. Although I found the process challenging, I eventually succeeded in integrating and applying the models to implement the particle filter. I appreciated the modularization of the models, which allowed me to focus solely on understanding their specifications without worrying about their implementation. However, I did find the lab instructions somewhat unclear and difficult to follow in terms of implementing the MCL algorithm in the particle filter. Initially, I was able to develop a working version of the particle filter, but it failed to pass the autograder. Fortunately, my teammates were able to provide valuable assistance in debugging the conceptual errors in the particle filter. We were able to collaborate effectively, using hints and clarifications from office hours to improve our understanding of the problem. This experience highlighted the importance of teamwork in solving complex technical problems and the value of having fresh perspectives to identify problems that may be overlooked.

Rachel:

This lab taught me the significance of comprehensively grasping the concepts and intricacies of an algorithm before coding it. Developing the sensor model and motion model separately was relatively straightforward compared to writing the particle filter. However, without a clear comprehension of the MCL

algorithm and how the sensor and motion model amalgamate in the particle filter, we encountered unforeseen bugs. While programming the particle filter, we utilized RViz to visualize the code. Regrettably, due to a conceptual misunderstanding about whether the particles should converge into a single point or not, we failed to detect a crucial bug in our motion model. We wrongly believed that particle convergence was the intended result instead of a spread of distinct particles. Although we presumed that passing the unit tests meant our models were functional, we discovered a bug in the way we added noise, which the unit tests did not detect. Going forward, I have learned not to assume that different components of our program work correctly just because they pass the unit tests. Furthermore, conceptual understanding is vital and indispensable to detect errors in the way we implement algorithms. In terms of teamwork and collaboration, our team effectively divided the lab into modules and developed them separately. This allowed us to focus on specific tasks and ensure that each module was implemented correctly. Communication was key in ensuring that each member was on the same page and understood their respective responsibilities. When it was time to combine the modules, we were able to do so seamlessly and efficiently, resulting in a functional localization algorithm.

Claire:

This lab was a good experience for me in terms of requiring me to truly understand the theoretical basis for the concepts that we were implementing. I felt that in past labs, our biggest priority and challenge was debugging the implementation and working with ROS architectures, but since such a large part of this lab was on getting it to work in simulation, there was a larger emphasis on having the right approach in the first place. Both motion and sensor models were a new concept to me, so I enjoyed learning how they individually worked and how to fit them together in the particle filter. I was also introduced to the concept of thread locking, and really being concerned with optimization to make our model run as efficiently as possible. In terms of teamwork, I learned that it's still really important to talk to TAs and professors for help. We've mostly been able to ask friends and peers in the class when we got stuck, but in this case our biggest problem was a conceptual misunderstanding, and it was talking with the professor that ultimately fixed our problem.