Lab 5 Report: Localization Team 13: Five Guys Drive-Thru

Allen Baranov James Morin Jian Ming Chen Remeyn Mechura Ryan Xiao

6.4200/16.405, Robotics: Science and Systems April 8, 2024

1 Introduction

(Authored by Remeyn Mechura, proofread by James Morin)

Previously, we have designed and tested a few programs to autonomously run our robotic car in specific scenarios. We have successfully programmed a generic wall following algorithm combined with a safety controller which can maintain a set distance from a detected wall on either side of our car and, if necessary, stop to avoid an imminent collision. This is a useful, albeit limited program based on onboard lidar scanners feeding distance and direction information back to the car to facilitate navigation. Following this, we designed, tested, and implemented an object chasing and line following algorithm which utilized the onboard optical cameras to detect a specific color using HSV (hue, saturation, and value) filtering. To help in that task, we designed a homography transform to take the two dimensional image from the camera and extract distance data, which we then used to implement our line following algorithm.

Now, we want to be able to autonomously navigate to any desired position in a known environment. There are a few future goals we are working towards, such as using our car to race against the other teams around a track, race around the Stata Center basement, and navigating on a pool deck. To make progress toward these ultimate goals, the goals of this lab are to be able to:

1. Determine our car's location with respect to a global, fixed coordinate frame.
2. Design and implement a Monte Carlo Localization algorithm which takes the cars current pose and odometry data to create a probabilistic set of points mapping out our most likely future location.
3. Confidently and expeditiously navigate through the known environment to safely arrive at our desired location, while accounting for unpredictable noise that may be encountered along the path of the car.

Accomplishing these tasks will enable us to operate our vehicle in a wide range of locations and environments. We will be able to take a map of an area, such as a hallway, classroom, or the floor of a building and have a robust understanding of where our robot is at any time. In the future, this will enable us to pick a location for our car to drive to, hit start, and let it run. There will still be scenarios in which this does not operate fluidly, but finishing this lab will be a significant achievement in our journey towards autonomy.

2 Technical Approach

(Authored by Ryan Xiao and Allen Baranov, proofread by Jian Ming Chen and Ryan Xiao)

Our goal in this lab is to develop an algorithm that finds our car's position and orientation in the Stata basement. This is known as the localization problem. In real life, we don't have GPS to localize our car. Instead, we have the car's sensors and odometry along with a map of the Stata basement. We seek to implement a motion model to track the car's movements using odometry data and a sensor model using LiDAR scans to track where the car is in the environment. Once these are developed, we will take those models and use them to construct a complete Monte Carlo Localization algorithm.

2.1 Motion Model

(Authored by Ryan Xiao, proofread by Jian Ming Chen)

The first component of the localization algorithm is the motion model, which describes how the robot's position will change in response to a control input. In order to understand the model, it is important to understand how the robot represents its current position and orientation. In the model, we use the robot pose, X_k^W, defined relative to the world, to encode the robot's position. This lets us pinpoint where the robot is in the map at the current state. Because the robot is not able to move up or down, we only need to use a 2D pose consisting of the x and y positions and orientation angle. In other words,

$$X_k^W = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}$$

Once we have the position, our model will calculate the robot at the next state using odometry data, ΔX_k^r, as a control input. The odometry data gives us an estimated position change of the robot over time, according to data from motion sensors, also represented as a pose. However, to obtain the robot's pose at the next state, we cannot directly add the odometry data to the robot's pose at the current state. This is because the odometry data was collected in reference to the local robot frame, r, instead the world frame, W. The robot oftentimes will not have the same orientation as the world frame. Therefore, a transformation needs to be applied to the odometry data before we can obtain the next state. To do this, we use a 2D homogeneous transformation matrix:

$$T_r^W = \begin{bmatrix} R_r^W & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} cos(\theta_r^W) & -sin(\theta_r^W) & 0 \\ sin(\theta_r^W) & cos(\theta_r^W) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where θ_r^W is the robot orientation at the current state, θ_k. Applying this transformation matrix rotates the odometry data from the robot frame, r, to the world frame, W:

$$T_r^W \Delta X_k^r = \Delta X_k^W$$

However, we do not want the robot's next state to be deterministic. This is because in the MCL algorithm, the robot is simulated over many iterations. We call each robot state iteration a particle. So, if the calculation was deterministic, the particle would end up in the same position on every iteration.

To account for this, we use a 3x1 noise matrix given here:

$$N_k = \begin{bmatrix} N_{k,x} \\ N_{k,y} \\ N_{k,\theta} \end{bmatrix}$$

For each position and orientation is incorporated into the calculations for the next state. The noise term is randomly chosen from a normal distribution centered about 0. The standard deviations we use in our model are $\sigma_x = \sigma_y = \sqrt{0.2}$ for position and $\sigma_\theta = \sqrt{0.01}$ for orientation. These values were obtained through rigorous testing in simulation. Values of standard deviation that were too low did not produce a particle spread wide enough for the MCL algorithm to produce a good match. Similarly, a standard deviation that was too high produced a particle spread too sparse, also not allowing the algorithm to produce sufficient matches. Combining our terms, we can obtain the motion model, consisting of a linear system of equations representing the pose of the robot at the next state, shown here:

$$X_k^W = X_{k-1}^W + \Delta x_k^W + N_k = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} cos(\theta_r^W) & -sin(\theta_r^W) & 0 \\ sin(\theta_r^W) & cos(\theta_r^W) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta \theta_k \end{bmatrix} + \begin{bmatrix} N_{k,x} \\ N_{k,y} \\ N_{k,\theta} \end{bmatrix}$$

2.2 Sensor Model

(Authored by Ryan Xiao, proofread by Jian Ming Chen)

Once we have the motion model, our MCL can simulate it over multiple iterations to generate a cloud of particles showing possible locations the car could be at the given state. This will be explained more in the Particle Filter section below. Once the particle cloud has been calculated, we need to assign probabilities to each particle denoting the likelihood that the particle represents the true position of the robot. Each particle will simulate LIDAR scan data by taking ray casts. This data will then be fed into our probabilistic sensor model, given by the equation below (see appendix for p_{hit}, p_{short}, p_{max}, and p_{rand} equations):

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

This equation is calculated for each individual sensor scan, $z_k^{(i)}$, in the ray cast dataset, with $\alpha_{hit} = 0.74$, $\alpha_{short} = 0.07$, $\alpha_{max} = 0.07$, and $\alpha_{rand} = 0.12$.

In order for the MCL algorithm to work, it needs to run the simulation over many iterations. How many iterations decides how many particles there are to run through the sensor model. It would be infeasible for us to do an exact LIDAR scan simulation with every point. So in our actual implementation, we speed up our calculations by discretizing range of values that the ray casts can measure into 200 possible buckets and pre-computing the probability values of each bucket beforehand and placing them in a 200x200 lookup table. This will drastically speed up our calculations as we can directly pull probabilities from the lookup table for each ray cast rather instead calculating the probability at every scan with the possibility of recalculating a value that had previously already been calculated.

2.3 Particle Filter (MCL)

(Authored by Allen Baranov, proofread by Ryan Xiao)

The reason we develop a motion model and sensor model is to combine them into a particle filter. A particle filter is an algorithm used for recursive state estimation. It's commonly used when the state of a system is not fully known but can be inferred from noisy sensor measurements over time. The particle filter is made up of "particles", which are predicted states of the system. In our case, each particle will predict the position and orientation of the car. The particles are updated and resampled based on odometry and LiDAR data provided by the car, which is where our motion model and sensor model will come in handy.

The particle filter uses a two step process to refine the state estimate: prediction and correction. During the prediction step, each particle's state prediction is updated using our motion model. We use each particle's current pose and the car's odometry data to update the pose prediction.

Once the particles' predictions are updated, we can perform the correction step. Here, we weigh each particle by its performance using our sensor model. We define performance as how likely the car's LiDAR measurement is given a "ground truth" LiDAR scan from the particles' pose. Once we weigh each particle, we resample and select particles at random based on the weights. This step can be thought of as a raffle drawing, where particles that align closely with sensor information are given more tickets in the raffle drawing. By resampling, we select for particles that accurately capture the car's pose,

improving the accuracy for future iterations. Repeating these two steps allows us to maintain a set of particles that closely matches the car's pose.

Aside from this two step process, we also needed to generate a pose for the car using the particles. The way we made a final prediction on the car's pose was by calculating the average position and orientation of all particles. The formulas for average position and orientation are as follows:

$$\bar{x} = \frac{1}{N} \sum x_i$$

$$\bar{y} = \frac{1}{N} \sum y_i$$

$$\bar{\theta} = \tan^{-1} \left(\frac{\sum \sin(\theta_i)}{\sum \cos(\theta_i)} \right)$$

The average position is a simple average, while the average orientation is computed using a transformation from polar coordinates to Cartesian coordinates to take the average before transforming back to polar coordinates.

After implementing the particle filter, we are ready to run algorithms such as wall following on the car while keeping track of its position in simulation.

3 Experimental Evaluation

(Authored by James Morin And Jian Ming Chen, proofread by Allen Baranov and Remeyn Mechura)

We implemented all of these principles first in simulation in various sections of the Stata basement, then in real life in the Stata basement outside of 32-082. In both cases, a pre-existing map of the Stata basement was used to simulate the lidar scans of the points in the point clouds. Several experiments were run in simulation in order to test what positional variance led to the most reliable and accurate localization, and the resultant value was verified on the racecar.

3.1 Simulation Setup

(Authored by James Morin and Jian Ming Chen, proofread by Allen Baranov)

The first step was to make sure that we correctly implemented the math of these algorithms. We were able to test both the motion model and sensor model with unit tests offered by the class. These confirmed that our odometry increments and our precomputed tables were calculated accurately. Then, putting both models together required calling each model in certain callback functions. After making this work correctly, we were ready to test in simulation.

A significant part of our technical approach is the Gaussian noise added to our motion model, so a lot of our time was focused on tuning the standard deviation of the noise as described below in Section 3.2. Our tests consisted of running the localization algorithm with a set amount of noise and driving the simulated racecar around the map using our wall following module. Qualitative data was then taken by visually comparing the estimated pose of the racecar to its actual location. Quantitative data was taken utilizing internal calculations to compute results such as convergence rate and positional error. Both types of data were then used to measure the success and tune our localization algorithm as described in Section 3.2 below.

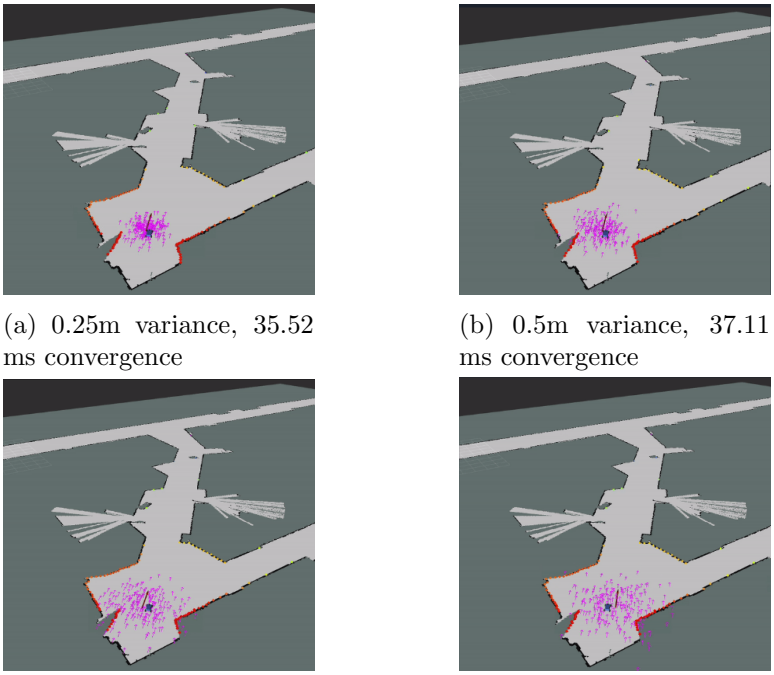
3.2 Simulation Results

(Authored by James Morin, proofread by Allen Baranov)

To make our algorithms perform as accurate as possible, the only variable to alter is the variance introduced in the odometry steps for the points in the point cloud within the motion model. We tested a number of values of the positional variance to determine how well they performed in different experiments.

Our first experiment was conducted in order to determine how changing the variance might affect the runtime of the algorithm. We want to run as many cycles of localization as possible per second, so increasing runtime by a significant factor would be bad for the algorithm's overall performance. Therefore,

we observed the convergence time for a stationary robot at different values of positional variance. The results are shown in Figure 1 below.



(a) 0.25m variance, 35.52 ms convergence

(b) 0.5m variance, 37.11 ms convergence

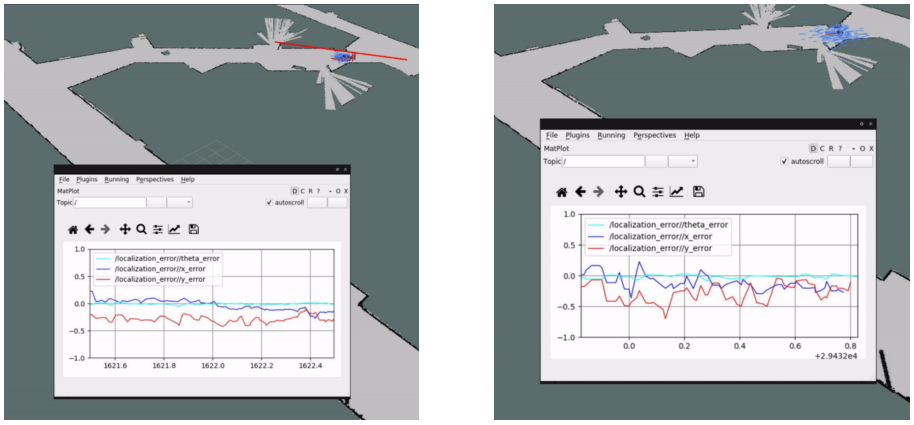
(c) 0.75m variance, 41.26 ms convergence

(d) 1.0m variance, 46.33 ms convergence

Fig. 1. Comparison of convergence time (ms) versus the variance in spacial noise (m). There is a slight positive correlation between variation and convergence time, but the magnitude of this change is not much.

According to these tests, the magnitude of the positional variance will not greatly affect the convergence time of our particle filter, so the computational speed is relatively unaffected by this parameter. Since the computational speed is not greatly affected by the size of the variance, we wanted to observe which values for the variance led to the best performance of our localization algorithms, without placing an upper bound on variance. For these tests, we ran our localization in simulation along with wall following code from Lab 2. First, we tested our simulation by setting the robot to drive along a section of the hallway with several features, with the results described in Figure 2 below.

This experiment alone would imply that we would optimally choose a small positional variance for our localization to function as accurately as possible. However, experimenting with only this section of the map does not tell us the



(a) 0.1m variance, 0.4m error

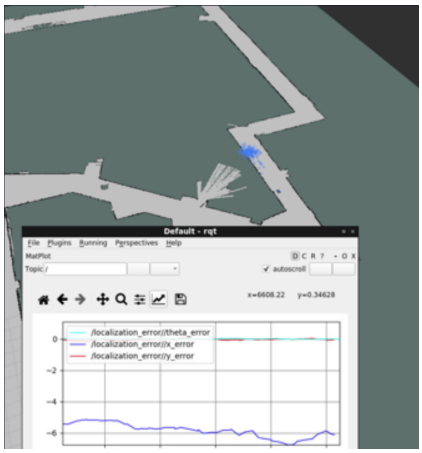
(b) 1.0m variance, 0.8m error

Fig. 2. Comparison of positional error (m) versus the variance in spacial noise (m) for a hallway with many features. There is a clear negative correlation between variation and measured error, so for these and similar circumstances, we would desire to have a smaller variance to decreased error in localization.

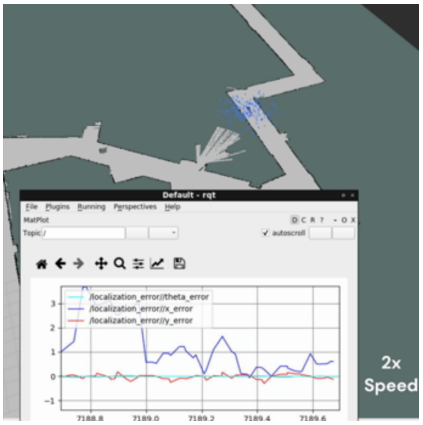
full picture. This section of the map outside of 32-082 has a lot of variation and features that enable the lidar comparisons to see significant differences when comparing to various points from the point cloud. However, there are also sections of the map without many features. For example, there is a hallway northwest of 32-082 where the walls are very flat, so as the robot moves down this hallway, there aren't many differences to be seen in the robot's lidar scans. A comparison in the performance of different variances is shown in Figure 3 below.

Even though this is a difficult section of the map, we still must be prepared to succeed in this region. Therefore, when we expect a map that is full of identifiable features, a variance of around 0.2m is good because it affords greater accuracy, but when there are sections of the map without many features, a variance of 2.0m or greater may be required to avoid completely losing the true position of the robot.

Now that we know exactly how we want to set the variance in the odometry data error, we are ready to test our localization on the racecar.



(a) 0.2m variance, failed localization



(b) 2.0m variance, successful localization

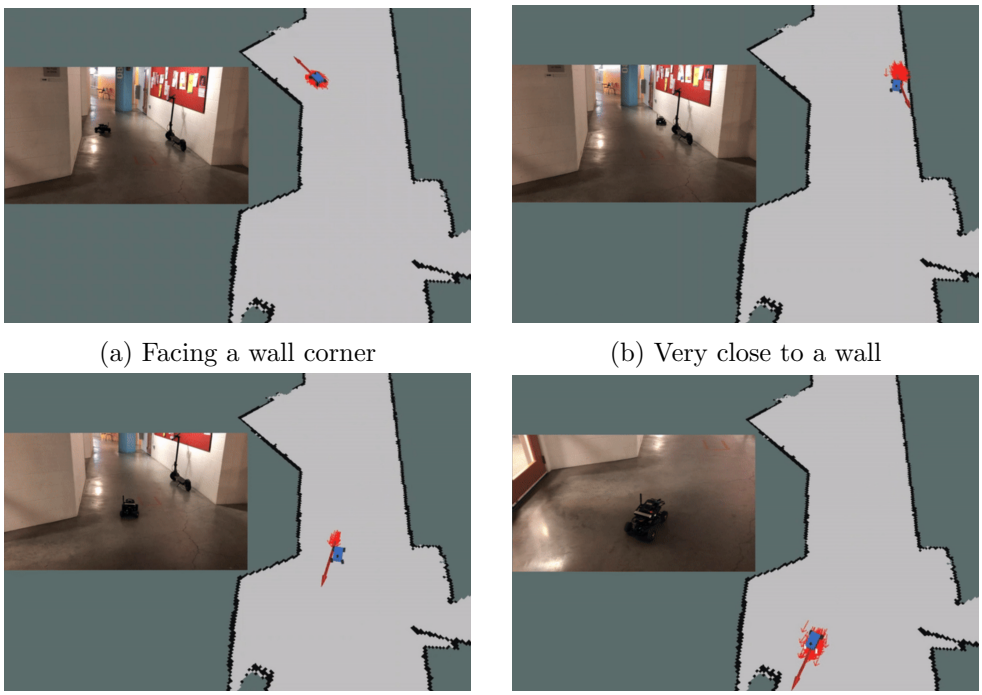
Fig. 3. Comparison of localization success for different variance in spacial noise (m) for a hallway without many features. While 0.2m variance was well within the range of successful parameters in the previous experiment, it was inadequate in these circumstances. Note how the point cloud and robot are completely separated, which is catastrophic localization failure. A very large variance of 2.0m was required in order for the robot to succeed in localization in this region.

3.3 Robot Implementation and Results

(Authored by Jian Ming Chen, proofread by Remeyn Mechura)

The real world evaluation mainly comprised of qualitatively analyzing the localization model in RViz and comparing it to the robot in the real map. Using RViz, we could discern the accuracy of the racecar localization, spread of particles, and robustness to sudden changes. Adapting our localization code for the racecar did not introduce any great obstacle to us, nor did it require much alteration. From our experiments in simulation, we decided to run the racecar with a spacial variance of 0.2m. We used teleop to control the racecar in the section of the hallway outside of 32-082. This section of the hallway had a distinct pillar that provided a great reference point for the car's localization. We would drive the racecar in chaotic movement patterns, such as making sudden sharp turns and going in circles, to provide as much sudden change to not only the robot's odometry but pose at the same time. We then observed how the racecar's localization plotted in the basement map matched it's position in the real world. During all our trial runs, the

racecar's pose in RViz matched almost perfectly to the racecar's real world pose. Several frames from our live tests with the racecar are presented in Figure 4 below.



(a) Facing a wall corner

(b) Very close to a wall

(c) Moving at full speed

(d) Stationary end position

Fig. 4. Several frames from our localization in real life comparing the racecar's localized pose in RViz and its real world pose.

In (a) and (d) where the car is at rest, we can see that the particle filter is very precisely centered around the true location of the racecar. The racecar knew where it was even very close to a wall as shown in (b). When moving at full speed as shown in (b) and (c), the point cloud does not always rest completely centered on the position of the racecar as described by the last localization, but these estimates do follow closely behind as each iteration updates the point's position based on probability and odometry. In all cases, the localization had no issues with determining the robot's location.

4 Conclusion

(Authored by Remeyn Mechura, proofread by James Morin)

In this lab, given a preacquired map of the environment with which to simulate lidar data, we successfully designed a localization algorithm that eliminates the issue of drift that dead reckoning based on odometry would have. Our motion model takes hundreds of particles around the car and updates their pose given the odometry readings and random noise that we generated for the simulator. Then, our sensor model uses a delicate balance of accuracy and computational efficiency to determine which particles are most similar to the actual racecar pose. To that end, we took the lidar scan information from the car, compared it to simulated lidar scans generated on the map for each particle from the motion model, and assigned weighted probabilities to determine the most likely position of our car. Finally, the Monte Carlo Localization scheme allows us to combine the two models, resampling the particles and removing the ones that have low probability of being at the same location as our car until we are left with a small region of high probability.

Using a map, lidar scans, and odometry data from the car, along with a starting location, our particle filter is able to keep a fairly accurate account of the position of our racecar in the map with an acceptable amount of error. While we accept that there will be some error in the pose accuracy, this localization offers the immense benefit that our racecar's perceived and actual pose will never drift apart over time due to odometry/IMU error buildup. We aim to solidify and build upon this in the next lab where we will introduce, among other things, motion planning, which will help us to complete all of the challenges that we are building towards.

5 Lessons Learned

5.0.1 Allen

On the technical side, I learned how many layers we can add to our models and how deep they can go. For example, our sensor model equation is massive and takes into account many different possibilities. I was surprised by how we were not introduced to a simpler formula, and that this was the level of depth needed to capture the idea. On the communications side, I have really hammered in the idea of not saying "like" and other filler words. This is a problem that has been pervasive my personal life as well as my

communications in this class. At this point, it's such a habit for me to catch myself that I've learned how much of a communications blocker it really is. It makes conversations harder to follow and makes me seem like a less confident speaker. Now, when I catch myself, I feel like my point comes across more clearly.

5.0.2 James

When it comes to robot autonomy, the lack of human intervention means that the robot's programming must be extremely robust. Any input that could possibly be received by the robot's sensors needs to be accounted for. In this project, the localization worked the best when we used portions of the map with clear features that made the lidar scans of different nearby points look significantly different. However, regions of the map without many unique features, like a long, straight hallway, introduced a problem for a robot without a sparse enough point cloud. The robot did not consistently make it through the hallway with accurate localization, and the point cloud often fell entirely out of sync with the robot. This taught me how an optimization made in a subset of possible inputs may lead to unreliability in performance when other possible inputs are encountered.

5.0.3 Jian

In terms of technical lessons, this was one of the most theory intensive labs by far. I had to spend a lot of time reviewing past lectures and searching online to fully understand the MCL algorithm. This reminded me how important it is to grasp the material in lecture as it will be pivotal in implementation in future labs. As for communication lessons, this lab really challenged my time management skills as it was the first lab where a lab report was also required. It meant I had to schedule different components of the lab clearly in order to have everything done in time. Delegating tasks was also a good learning opportunity in this lab as it required all hands on deck for success.

5.0.4 Remeyn

This was a particularly involved lab, being the first time we had a report due in addition to all of the other things similar to the previous lab. A lesson I learned was that I really need to learn more about computer programming fundamentals and time management. The majority of the issues I have encountered with this class revolve around the programming portion. This lab was kind of an exception in that it took me a long time to understand what was going on to begin with, making the time available to actually do work rather scarce, and all of the coding was pretty much done before I even looked at it. It is cool to see how this kind of stuff works though, and that does provide a bit of motivation to continue learning. Something else I have come to appreciate through this class is teamwork.

5.0.5 Ryan

Probability can be a very useful tool in robotics and autonomous machines. Previously, I only had exposure to probability in the context of finance and statistics. While I was studying the derivations behind probability laws and solving problems with various distributions, there was always a wonder of whether I would actually be able to apply these concepts o fields that I was considering for the future. This lab opened my eyes to the role that probability can play in the realm of robotics. Localization is an important step toward autonomy and the MCL algorithm that we use relies on probability to work. Normal distributions allow us to add noise to our otherwise "perfect" models, It allow particles to spread around a specific point, giving the algorithm more opportunities to find a best matching particle to represent the robot's true position.

6 Appendix

6.1 Sensor Model Equations

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

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

$$p_{\max}(z_k^{(i)}|x_k, m) = \begin{cases} \frac{1}{\epsilon}, & \text{if } z_{\max} - \epsilon \leq z_k^{(i)} \leq z_{\max} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$p_{\text{rand}}(z_k^{(i)}|x_k, m) = \begin{cases} \frac{1}{z_{\max}}, & \text{if } 0 \leq z_k^{(i)} \leq z_{\max} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$