Lab 5 Report: Tracking Robot Location

Team 24

Harry Heiberger (Editor)
Ayyub Abdulrezak (Editor)
Amanial Abraham
Michael Chen
Nicole Khaimov

6.4200 (Robotics: Science and Systems)

April 15, 2023

1 Introduction

[Harry Heiberger]
In labs 3 and 4, our team implemented algorithms for wall following and object detection. These areas were similar in that they allowed our robot to understand and function within new environments. Given distinctive color patterns or a straight wall to follow, our robot was able to look at its surroundings and perform complex tasks wherever it was placed. However, in many robotics applications, inferring your environment from just the objects in the vicinity is not enough. For example, areas such as motion planning and maze navigation require a more detailed understanding of the environment than we've had previously. Thus, giving the robot a detailed understanding of its own location is a critical problem in autonomous robotics.

One important area of research within this problem is localization. Given a map of the environment and starting location, localization algorithms attempt to maintain an understanding of the robot's location on the map throughout its movement. At first glance, this may seem as simple as directly transcribing the robot's movement on the map. However, noise in these measurements makes it much more difficult to maintain accurate tracking. A variety of algorithms exist in the current literature to resolve this issue. One especially notable solution is the Monte Carlo localization algorithm. At a basic level, this algorithm functions by maintaining a set of probabilistic guesses of the robot's location that, in a weighted average, converges on the actual robot's location.

In this lab, our group implements the Monte Carlo algorithm in order to solve this problem of localization. We then analyze its performance using a variety of qualitative and quantitative metrics. By the end, our robot was able to robustly track its location on a map in a variety of different driving conditions and environments, something we hope to harness in future labs to solve more complex problems such as path planning.

2 Technical Approach

[Aman Abraham and Harry Heiberger]

In this lab, we implemented the Monte Carlo Localization algorithm in order to solve our problem of keeping track of the robot's location on a map. This algorithm functions via a set of probabilistic guesses of our robot's location that, in a weighted average, converges on the actual robot's location. These guesses, called particles, each represent an independent guess of our robot's orientation and position. Our implementation of this algorithm can be split into three main components: the motion model, the sensor model, and the overarching particle filter code that brings these two models together.

2.1 Motion Model

[Aman Abraham and Harry Heiberger]

The motion model is responsible for updating the position of our particles using the odometry data of our robot. In other words, whenever the robot moves, we update all of our particles so that they move in roughly the same way such that they track the movement of the robot. However, all particles do not move in the same direction. Instead, we add in some random Gaussian noise to each particle's movement direction to account for the inaccuracies in our odometry sensor measurements. A visualization of this action can be seen in Fig. 1. This works because, with many particles, there is a high likelihood that some of the particles given our random noise will move in the actual direction of the robot regardless of any odometry error. In other words, adding this noise has the effect of removing error in the estimated position of our robot and making it roughly converge on the robot's actual path over time. We tuned the amount of noise added through rigorous testing and eventually found that a noise standard deviation between 0.2-0.4 led to the best tracking. Our data for this is discussed in further depth in a later section.

2.2 Sensor Model

[Ayyub Abdulrezak]

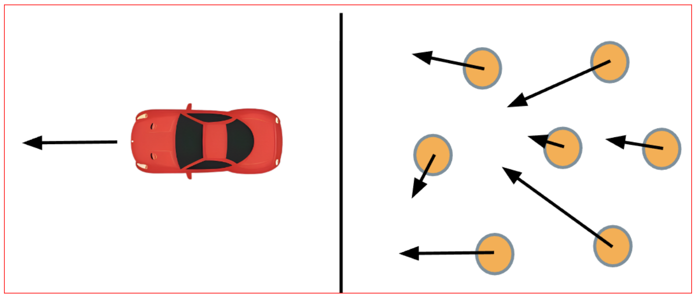


Fig. 1. Example update step of our motion model.

After using the motion model to update the positions of each particle, the sensor model handles computation of the likelihood that each particle matches the robot's actual location. It does this via ray casting from the particles (to simulate a laser scan from them) and comparing it to the actual laser scan data of the robot. Particles whose raycasts are similar to the actual sensor readings have a much higher probability of matching the actual robot's position.

Mathematically, the sensor model takes as inputs a sensor reading z_k, a hypothesis position x_k, and a map m, and produces as output a probability by taking a weighted sum of the likelihood of several cases. The likelihoods are split into the cases p_{hit}, p_{short}, p_{max}, and p_{rand}. They cover the scan hitting an obstacle, getting a very short value back (i.e. close interference), getting a very large value back (i.e. far interference, or environmental, reflective interference), and measurements that are entirely random (errors and largely unforeseeable environments). These cases are split such that they can cover a wide variety of the types of things that the scans might run into, and the weight of each component was tuned in order to maximize performance.

The formulas for these components are as follows. In these formulas, d represents our raycast distance, z_{max} is the maximum distance our lidar can output, and η is a normalization constant such that the Gaussian integrates to 1 on the interval $[0, z_{max}]$.

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

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

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

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

After calculating our probabilities, we exponentiate them to the power of $\frac{1}{2.2}$. This functions as a squashing factor that makes the point of highest probability in our probability distribution less pointed. In other words, it provides more leniency on what's considered a strong match between the raycast and actual Lidar data. It was found in our testing that using this squashing parameter led to a better tracking of our robot's actual position.

2.3 Particle Filter

[Michael Chen]

Once we created the motion model and the sensor model, we used them to construct the Monte Carlo localization algorithm. On a high level, the goal of this overarching component was to use the motion model to take in the odometry data to update the positions of each of the particles. It then uses the sensor model to assign probabilities to each particle based on the likelihood of being in a particular location. These probabilities are then used to resample the most likely positions of the particles, thereby removing particles that are far from the car's actual position. Finally, every time the particles update, the average pose of the particles is determined and the transform is published.

The first step to creating our particle filter is initializing a starting pose for the vehicle using an initial "guess" for its location. The reason we start with this type of initialization is because global localization without a known starting point, otherwise known as the "kidnapped robot problem", is extremely difficult. The pose is initialized when an arrow is dropped on the screen in RVIZ. The pose is extracted from the clicked position to determine a position which should be very close to where the car is in real life. The determined position is used as the center around which particles are randomly generated following a Gaussian distribution with noise in the form of a tunable standard deviation constant.

Another tunable parameter is the number of particles utilized. As proven in the extensive literature of this algorithm, Monte Carlo localization guarantees perfect convergence on the actual car location with infinite particles. Thus, we tuned this value by increasing it as high as we could while maintaining a real-time performance rate of our code. We ended up finding that 800 particles performed robustly while maintaining an update rate of greater than 20 hz.

As our robot moves in the real world, odometry data is collected from its various sensors. When this happens, a callback in our particle filter is triggered through a ROS subscriber, and these measurements are passed as input into the motion model. The positions of the particles are then updated accordingly using the motion model.

Similarly, as LIDAR data is collected on the vehicle, it's used to assign probabilities using the sensor model. First, the approximately 1000 datapoints of laserscan data are down-sampled to approximately 200 for efficiency reasons. Then, the laser scan is fed into the sensor model in order to calculate the probability that each particle matches the actual car's position. These particles are resampled by a selection of particle indices based on the probabilities outputted by the sensor model. This results in more particles where there's a higher probability of their existence as well as particles that are poor predictions being removed. Because both the odometry and scan data influence the particle array, we used thread locking to prevent multiple simultaneous edits.

Each time the particles update by either the motion model or sensor model, we recompute our estimate of the robot's pose. This is done by taking an average of the x and y position of all of our particles. The new orientation is set as the average orientation using a circular mean calculation in order to avoid the issues of directly taking the average of circular data. The formula of our circular mean is as follows:

$$\alpha = \operatorname{atan2}\left(\frac{1}{n}\sum_{j=1}^{n}\sin\alpha_j, \frac{1}{n}\sum_{j=1}^{n}\cos\alpha_j\right) = \operatorname{atan2}\left(\sum_{j=1}^{n}\sin\alpha_j, \sum_{j=1}^{n}\cos\alpha_j\right)$$

Finally, this newly estimated position of our pose is then broadcasted for use in other systems.

3 Experimental Evaluation

[Harry Heiberger]

After implementing our Monte Carlo localization algorithm, we moved on to analyzing its effectiveness and accuracy. This evaluation was performed through a variety of both quantitative and qualitative metrics that are discussed in the following sections.

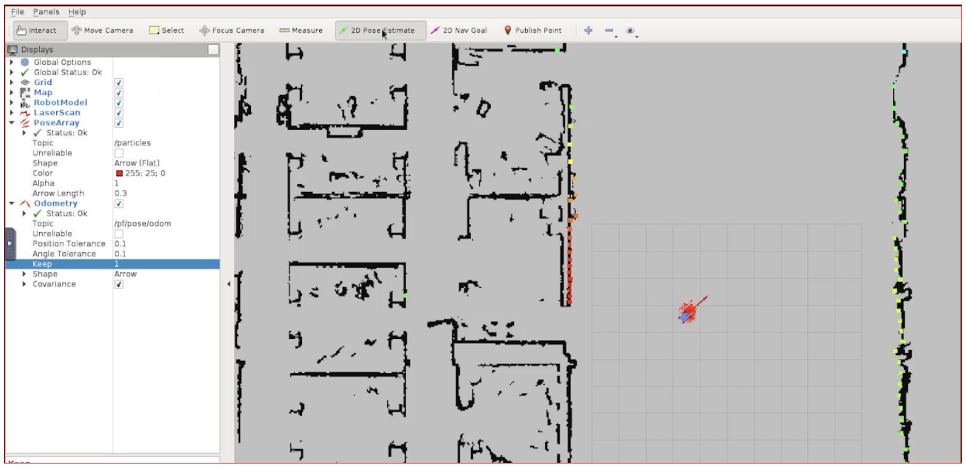
3.1 Localization Simulation Performance

[Harry Heiberger]

Before running our algorithm on our physical racecar, we first performed an evaluation in simulation. This was done in order to avoid the noise and uncontrollable variables of working in the real world. Recall from our technical approach that we have to add some Gaussian noise to our motion model update step in order for the particles to converge on the correct robot location. A big part of our simulation testing was tuning the standard deviation of this noise in order to get the best possible convergence.

Tests were performed by simply running our localization algorithm in the simulator with a chosen amount of noise and driving the simulated robot around a constant path. Quantitative data was taken through internal calculations that were printed at the end of each test. Our quantitative data for the simulator tests at each noise standard deviation are shown in Table I. The average number of steps to convergence refers to the average number of particle resamplings that occur before the particles are all within a 0.1-meter margin of the average particle location. It is important to note that this value refers to the time it takes for all particles to agree on an estimate of the location of the robot and not whether that agreement is the correct robot location. Likewise, average particle convergence time refers to the average number of seconds before the particles are within a 0.1-meter margin of the average particle location.

Qualitative data was taken by observing whether the estimated location of the robot matched closely with its actual location. Video recordings of the simulator at each noise standard deviation can be found in the Appendix. In the videos, the car model represents the actual position of the simulated robot, the red dots represent the Monte Carlo particles, and the red arrow represents the estimated location of the robot's position or the weighted average of these particles. A sample screenshot from our 0.4 noise video is shown in Fig. 2. This screenshot was taken at a point where the estimated location is matching closely with the robot's actual location



Global Options	
Global Status: Ok	
Grid	✓
Map	✓
RobotModel	✓
LaserScan	✓
PoseArray	✓
Status: Ok	
Topic	/particles
Unreliable	☐
Shape	Arrow (Flat)
Color	🟥 255; 25; 0
Alpha	1
Arrow Length	0.3
Odometry	✓
Status: Ok	
Topic	/pf/pose/odom
Unreliable	☐
Position Tolerance	0.1
Angle Tolerance	0.1
Keep	1
Shape	Arrow
Covariance	✓

Fig. 2. Sample image from our simulator testing videos.

Looking at our data, a noise standard deviation of 0 led to a very quick particle convergence time. This makes sense as with no added noise, the particles will never diverge once they are stacked in the same position. However, this lack of noise also meant a very poor qualitative tracking performance as there is no noise to offset the errors between our simulated and real robot positions. In other words, the particles will continually move further from the robot's actual position once they lose it.

In contrast, adding particle noise led to much better tracking performance. Even with a small noise standard deviation of 0.05, the particles were able to remove the estimated position error over a few update steps and roughly converge on the robot's actual path. Looking at our quantitative data, increasing noise increases both of our convergence time metrics. This makes sense as a larger particle spread means it takes longer for the particles to resample such that they are within a 0.1-meter margin. However, larger amounts of noise also led to a faster rejection of discrepancies between the simulated and real robot position, something that is more desirable than this increase in convergence time especially as the convergence time is small enough to be unnoticeable at the standard deviations tested. This faster error rejection occurs because the particle updates are able to take larger steps toward the actual robot position if a discrepancy occurs. Looking over our qualitative videos, our algorithm tracked the real position of the robot best with a noise standard deviation between 0.2 and 0.4. Higher noise values started to cause instability in the estimated position and eventually a loss of tracking. This is most likely because with the same number of particles, a very large spread means that it is less likely for the randomly moving particles to be over our robot's position.

Overall, as shown through the close tracking of the robot's actual position in our videos, our Monte Carlo localization algorithm performed robustly in simulation after choosing an adequate noise value.

Table I
SIMULATION PARTICLE CONVERGENCE ANALYSIS

Odometry Std_Dev (#)	Average Number of Update Steps to Convergence (#)	Average Particle Convergence Time (s)
0.00	0.01	0.0212
0.05	1.22	0.0540
0.10	1.79	0.0647
0.20	2.12	0.0717
0.40	2.35	0.1068
0.80	2.55	0.1152
1.60	2.95	0.1200

3.2 Localization Real-World Performance

[Nicole Khaimov]
Similar to the evaluation in the simulator, the quantitative analysis of our localization algorithm on the physical racecar centered on the use of metrics to confirm the optimality of our implementation of noise addition to the odometry data in the motion model. By running these tests outside of simulation, we could be confident that our implementation was not overly reliant on the abstraction of the environment in the simulator and could handle the significant increase in noise and uncertainty as we transitioned to the real-world environment.

In this set of tests, we used teleop to manually control the car as it drove along the path shown in Fig. 3 at variable speeds. This path, approximately 60 meters in length, includes both right and left turns. We ran multiple trials while varying the standard deviation σ^2 of the Gaussian noise added to the odometry. Unlike in simulation, however, it was necessary to constrain the range of values to those that would work reasonably well for the purposes of localization. When in simulation and using extreme values for σ^2 like 0 or 1.6, we can still drive the car around and visualize it, although the particles will quickly veer away from the car's location. In the real world, it becomes difficult to test because we use the visualization to show the car's location according to the particles; as we lose tracking and the car vacates the area of the map, there is no useful information on the output of our algorithm besides its extreme inaccuracy. We determined that it would be most useful to collect data over a limited set of values that we observed were, at a minimum, able to track the car for more than a couple of seconds.

Table 2 shows the results of our localization evaluation in the real world. The definitions for "Average Number of Update Steps to Converge" and "Average Particle Convergence Time" are the same as in the previous section. We also computed the Euclidean distance between the actual location of the car at the end of each trial and the location according to our algorithm shown in RViz. These values are reflected in the "Distance Error" column.

The convergence times across these standard deviations were quite consistent, all within 0.1 seconds of one another. However, there is a significant difference in the distance errors observed across these trials. Error decreased as standard deviation increased within the tested range, aligning with our expectations based on the simulation tests that adding particle noise improves tracking performance. The lowest observed error was 0.2069 meters, using a noise standard deviation of 0.4. Like with the simulation data, increasing this standard deviation further resulted in too much noise being added to the system and, thus, a loss of tracking.

Qualitatively, we determined that the localization algorithm was able to closely follow the path of the physical racecar by observing the alignment of the lidar

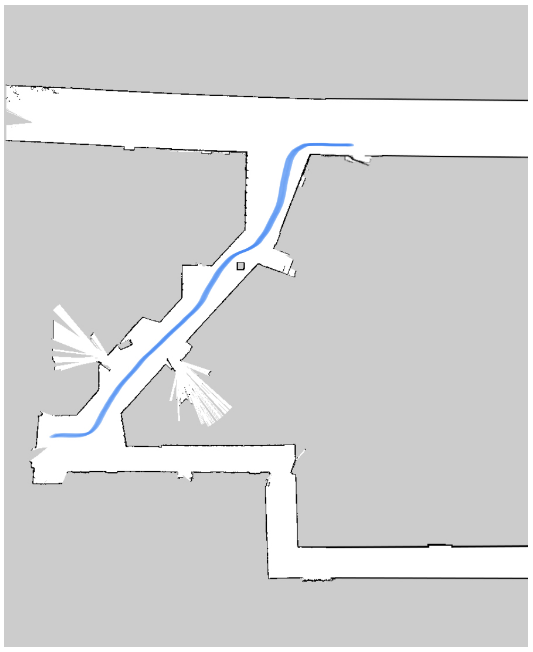
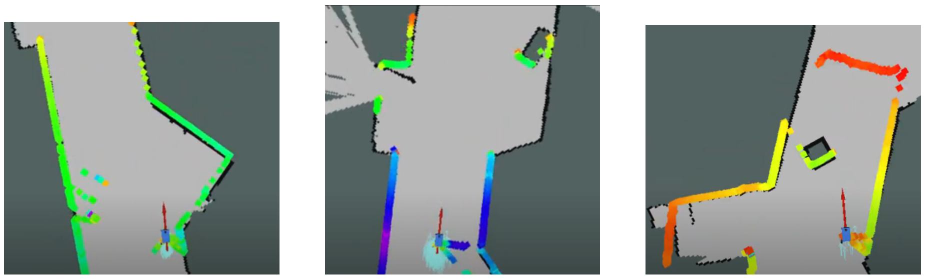


Fig. 3. Path of length approximately 60m driven in real-world localization tests shown in blue, using variable speeds throughout. This map shows a portion of the Stata basement.

Table II
REAL-WORLD PARTICLE CONVERGENCE AND ERROR ANALYSIS

Odometry Std Dev (#)	Average Number of Update Steps to Convergence (#)	Average Particle Convergence Time (s)	Distance Error (m)
0.05	3.16	0.194	0.2819
0.1	6.17	0.289	0.2210
0.4	2.15	0.197	0.2069

scan visualization with the map using RViz. Some sample observations are shown in Fig. 4. This alignment demonstrates the accuracy achieved by our implementation because the obstacles and walls observed in the lidar scan line up with the expectation of where the racecar is on the map.



$\sigma^2 = 0.05$ $\sigma^2 = 0.1$ $\sigma^2 = 0.4$

Fig. 4. RViz screenshots from each of our 3 real-world tests with different values for the standard deviation of the Gaussian noise added to odometry. The close alignment of the racecar's lidar scan visualization with the map features indicates the accuracy of the localization.

4 Conclusion

[Nicole Khaimov]
In this lab, we were able to successfully implement and test an algorithm for localization using an existing map. Using a modular approach to developing the particle filter of our Monte Carlo localization algorithm, we worked iteratively to achieve an implementation that worked well in simulation and then transitioned to tuning variables to find the pose of the physical racecar.

Essentially, given a map of the environment, an approximate starting location, Lidar sensor data, and odometry readings, we are able to accurately track the location and pose of the racecar in the global environment. Through testing, we determined that the tracking capabilities are resistant to odometry drift over time as well as noise in the lidar data because we use both readings to correct for one another and give us the most probable pose estimate at the current time.

Moving forward, we will be combining this work with our previous progress in wall-following, color detection, and line-following in an endeavor to achieve path-planning capabilities. As the implementation of a localization algorithm allows us to compute the racecar's current location in the global environment, we will be able to plan a path for navigating to a goal pose even in cases when the goal is out of view or obstructed by obstacles.

5 SLAM with Google Cartographer

[Michael Chen]
We utilized Google Cartographer to conduct

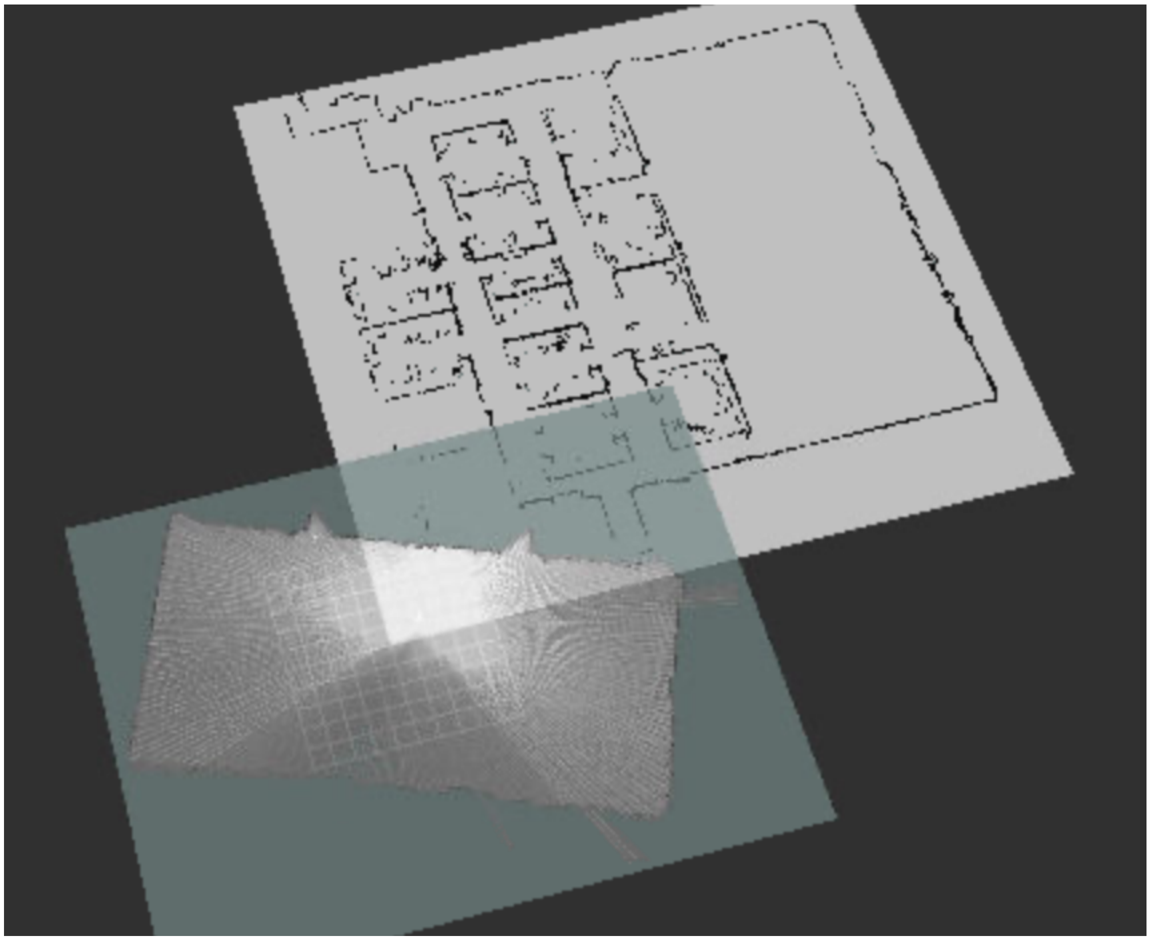


Figure 2: Caption

6 Appendix

6.1 Simulation Particle Convergence Videos

Link: https://drive.google.com/drive/folders/1cVB9wVoEBjukfyPjxQNePI-OJ9amcjCL?usp=sharing

7 Lessons Learned

[Harry Heiberger]
For me, lab 5 was a very fun assignment that helped train my ability to understand and implement complex algorithms. Getting a full understanding of the Monte Carlo localization algorithm required more than just going to lecture. Instead, I had to read through a variety of research papers that went into more detail. Being able to digest technical papers about a complex subject and then turn that into a code implementation is something I'll be doing throughout my career so it was good to get training on it now. Additionally, this lab was my first experience working with a probabilistic algorithm. These are useful as you get into more advanced computer science classes, so it was nice to get exposure to them.

On the communications side, this lab was very useful for improving my ability to communicate a technically complex topic. I was responsible for implementing much of the algorithm code. Thus, I had to explain my code and design choices to my teammates in a way that was easy for them to quickly understand. I then had to simplify these ideas even further to condense them into our briefing presentation. I found this training helpful and will definitely use these skills again as I go into my future career.

Finally, on the collaboration side, working with a team of five on a complex codebase was a very valuable experience. Writing code that is easy for your teammates to understand is a critical skill, as well as finding ways in which to fairly divide up the workload. By the end of the lab, our group was able to come up with good strategies for working on our codebase collaboratively, something that should be helpful as we get closer to the final project.

[Ayyub Abdulrezak]

[Michael Chen]

[Nicole Khaimov]
Throughout this lab, it was really interesting to see the initial modular components build up to a more complex model for localization. On the technical side, I appreciated this opportunity to learn about probabilistic algorithms as well as methods for improving efficiency in computationally expensive processes.

This will be useful in future implementations of more complex tasks, requiring additional data processing within a still limited execution time. Additionally, presenting on the technical approach in the briefing and elaborating on the experimental evaluation within the report gave me the opportunity to learn about the differences in effective communication between these types of information. Whereas explaining a technical approach focuses on rationale for design choices, describing an evaluation of the approach centers more on analyzing the results and their significance. Collaboration, once again, played a significant role in this lab as we had to work together as a team to discuss possible approaches, bring together the different parts of our implementation, and eventually deploy it on the racecar.

[Aman Abraham]

During this lab, I gained invaluable experience in using simulation as a tool for understanding and implementing complex algorithms. The simulation process allowed me to visualize the behavior of the algorithm and identify any issues in the implementation. Through multiple iterations and adjustments, we were able to receive a good understanding and score on the simulation.. This experience provided me with a deeper appreciation for the power of simulation in the development and validation of algorithms.

Furthermore, the lab built upon the communication skills we had developed in previous assignments. It pushed us to work collaboratively and effectively convey our thoughts on the algorithm's design and implementation. We learned to articulate our ideas clearly and concisely, ensuring that every team member was on the same page. This lab not only fostered better communication within the group but also equipped us with the skills necessary to present complex concepts to broader audiences, a vital ability for our future careers.

Lastly, the lab offered us critical insights into the concepts and implementation of the Monte Carlo localization algorithm. Through the process of research, coding, and simulation, we were able to explore the algorithm's intricacies and gain a comprehensive understanding of its workings. This hands-on experience allowed us to appreciate the practical applications of the algorithm and its relevance in real-world scenarios. Overall, the lab provided a solid foundation for our understanding of probabilistic algorithms and prepared us for more advanced topics in computer science.