Lab 5 Report: Localization

Team 6

Penny Brant
Yatin Chandar
Fritzgerald Duvigneaud
Nico van Wijk
Kristine Zheng
6.4200: RSS

April 15, 2023

1 Introduction (Penny, Yatin, Fritz, Nico, Kristine)

When given a map, a common task for a robot might to be localise itself within that map. Without a map, but with knowledge of its location, a robot might be tasked with building the map itself. Simultaneous location and mapping (SLAM) is the combination of these challenges, when a robot must locate itself in its environment whilst also building a map of its surroundings.

SLAM is a problem based by any robot that will need to enter novel or constantly changing environments. For example, self-driving cars will often have to drive through areas of dynamic obstacles and constant construction that may not be reflected in GPS maps. Thus, these principles are key building blocks for any robust autonomous system.

In this report we explain how we implemented Localization within both a simulated and physical racecar. We also describe explorations of the full SLAM process through google cartographer.

2 Technical Approach (Penny, Yatin, Fritz, Nico, Kristine)

2.1 Overview (Penny, Yatin, Fritz, Nico, Kristine)

Our approach to Localization was to utilize Monte Carlo Localization (MCL). The principle behind this approach is to use a range of "particles" that are randomly drifting to represent potential "states" of the robot - i.e its possible positions in space. Then a Bayesian probability estimate is used to determine the likelihood of each particle being the ground truth location based upon received sensor data. By constantly re-sampling the drifting particles based upon these probabilities, we see a convergence of the particles towards the car's true location. Theoretically, this can used to combat odometry drift from internal sensor responses.

To construct the MCL, we had to code three primary components. Firstly, a **motion model** that would take each particle and move it at each timestamp based upon the odometry data of the car. Secondly, a **sensor model** that would take in the LiDAR data of the vehicle and compare it to each particle and estimate the probability of that particle's correctness. Finally, a **particle filter** that used the two components described here and constantly updated the particles whilst publishing an estimate of the car's location based upon the particle data. These models are described in detail in the forthcoming sections.

Lastly, to explore other aspects of SLAM, particularly the mapping component, we also incorporated Google Cartographer into our simulated environment. This allowed us to build a map of the provided basement environment in real time.

2.2 System Setup

We worked on a race car with Hokuyo 2D Lidar sensor LiDar data on a NVIDIA Jetson TX2 Embedded Computer. For simulation, we tested it on RViz in ROS with 100 LiDar data points, the odometry data is obtained for wheel encoders.

2.3 Motion Model (Penny, Yatin, Fritz, Nico, Kristine)

The purpose of the motion model is to take the current particle locations and internal odometry of the car as input and then return new particle locations. Here, the current particle locations are stored in world frame, but the robot internal odometry are represented in car frame, which is represented in Fig.1
The first step in this process is to use the internal odometry, Δx, measured by the odometry tracking wheels and recorded in the frame of the car. Here, **Car frame** refers to the frame of reference pointing along the direction of the car at the moment of measurement. To calculate the updated position of the car, in equation.1 we used a rotation matrix to rotate the internal odometry from the

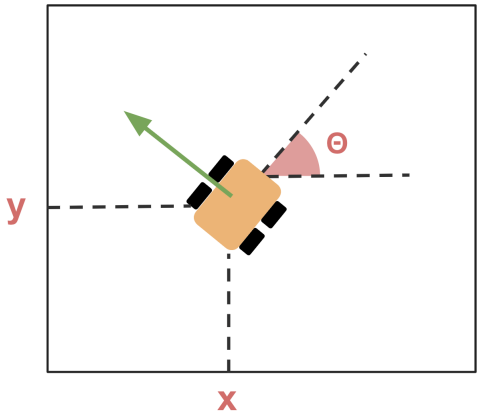

Fig. 1: Representation of the robot location and internal odometry, the robot location is represented by x, y, θ, and the odometry is the green vector in car frame representing the change in robot location

car's frame to the world frame before adding it to the current position in world frame.

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} cos(-\theta) & -sin(-\theta) & 0 \\ sin(-\theta) & cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \Delta x + \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{1}$$

We added noise to the internal odometry data to get our particle positions to diffuse and account for noise and drift in the system. We experimented with Gaussian and uniform noise in 6 distributions with varying parameters across different trails.
1) Gaussian noise with the distributions $\mathcal{N}(0, 0.02)$, $\mathcal{N}(0, 0.04)$, $\mathcal{N}(0, 0.06)$, where the first value is the mean and the second value is the standard deviation.
2) Uniform noise with the distributions of $\mathcal{U}(\frac{-1}{50}, \frac{1}{50})$, $\mathcal{U}(\frac{-2}{50}, \frac{2}{50})$, $\mathcal{U}(\frac{-3}{50}, \frac{3}{50})$, where the first value denote the lower bound of the uniform distribution and the second value is the upper bound of the distribution.
This noise was generated as a random 3x1 vector and then added to Δx to make a new noisy change of pose for each particle.

2.4 Sensor Model (Penny, Yatin, Fritz, Nico, Kristine)

The sensor model determines how likely it is to record the given data in a LiDAR scan given the current robot location from the motion model. We start with the likelihood of observing a single LiDAR scan data given a hypothesis $z_k^{(i)}$ in

a known static map m at time k:

$$p(z_k^{(i)}|x_k, m) \tag{2}$$

The total likelihood of the LiDAR is therefore computed as the product of the likelihoods of each of the n range measurements in the scan:

$$p(z_k|x_k, m) = p(z_k^{(1)}, ..., z_k^{(n)}|x_k, m) = \Pi_{i=1}^{n} p(z_k^{(i)}|x_k, m) \tag{3}$$

Furthermore, the likelihood of each range measurement is the sum of 4 scenarios:

1. Probability of detecting a known obstacle in the static map.

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

2. Probability of a short measurement.

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

3. Probability of a very large (aka missed) measurement.

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

4. Probability of a completely random measurement.

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

We used the given weights of $\alpha_{hit} = 0.74\, \alpha_{short} = 0.07\, \alpha_{max} = 0.07\, \alpha_{rand} = 0.12$ so the weighted sum of the four cases is equal to

$$\begin{aligned} p(z_k^{(i)}|x_k, m) = \alpha_{hit} \cdot p_{hit}(z_k^{(i)}|x_k, m) + \alpha_{short} \cdot p_{short}(z_k^{(i)}|x_k, m) + \\ \alpha_{max} \cdot p_{max}(z_k^{(i)}|x_k, m) + \alpha_{rand} \cdot p_{rand}(z_k^{(i)}|x_k, m) \end{aligned} \tag{4}$$

In the implementation, we pre-computed these summation sensor model values to a table of size 201x201, visualized in Fig.2, which discretized the range values and made the calculations more efficient. Therefore, we can index this table based on the ground truth distance d and sensor measurement x_k to quickly evaluate $p(z_k^{(i)}|x_k, m)$ for each particle.

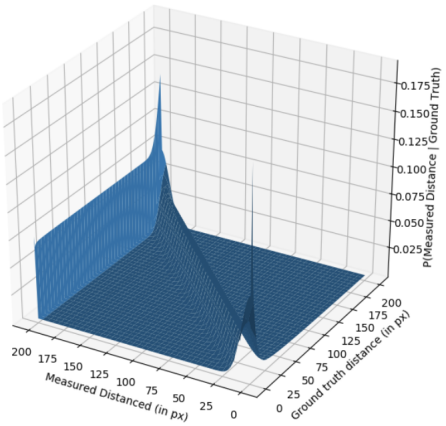

Fig. 2: A probability density function that shows the likelihood of each sensor reading given a fixed ground truth distance.

2.5 Particle Filter (Penny, Yatin, Fritz, Nico, Kristine)

The third component of our implementation is the particle filter. This module combines and utilises both our motion model and sensor model, and the way the different components interact is illustrated in Fig.3. The particle filter uses the motion model to calculate new particle positions, evaluates the probability of these particles using sensor model, and re samples the particles to form an estimate for the location of the robot. A visual demonstration of this process is illustrated in Fig.4

2.5.1 Motion Model

The particle filter uses the motion model to update the particle locations. The program tracks the time difference between the last seen odometry data and current odometry data, Δt between the current time and time of last odometry data, scale the odometry by this difference in time. Then, we update the particle positions using the motion model described in the motion model section, where noise is added to help the particles diffuse. These positions were directly used to update the particle filter's particles.

2.5.2 Sensor Model

The sensor model updates the likelihood probabilities of the different particles based on LiDAR data. When the particle filter received LiDAR data, the LiDAR data is averged every 100 data points for performance speed to 100 values and then passed into the sensor model. When the sensor model returns a likelihood

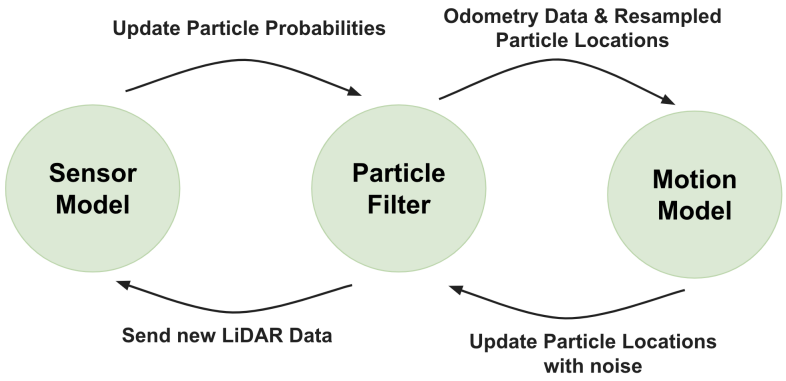

Fig. 3: Visualization of system overview of particle filter incorporates motion model and sensor model into updating accurate locations for particles.

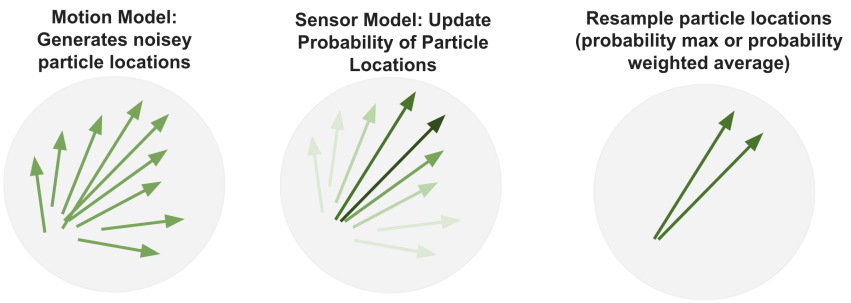

Fig. 4: Visualization of system design for particle filter's process of re-sampling particle locations. The left is noisy initialized locations by motion model, which sensor model evaluates the probability of, and locations using a probability weighted re sampling or taking the max probability point.

probability for each particle, the particles are then re-sampled based upon these probabilities. This aligns the particles in general more towards the ground truth.

2.6 Particle Re-Sampling

For calculating the pose estimate, we explored two different methods. The first of these was taking the particle with the maximum likelihood probability and estimate this particle's odometry as the truth:

$$\arg \max_{P(x_{i,k}|x_{i,k-1},u_k)} x_{i,k} \tag{5}$$

The second method we explored was taken an expected value of the odometry by combining all the particles and weighting them according to their probabilities:

$$E(X_k) = \sum_{i=0}^{n} P(x_{i,k}|x_{i,k-1}, u_k)x_{i,k} \tag{6}$$

These integrations function in parallel, so an important technical aspect of our solution was to build in a way to thread these processes. We achieved this by identifying that the mutual step taken by both processes was updating the position of our particles. Hence, we abstracted this updating to a separate function that was called from both parallel processes and locked with a threading lock so that it could only be updated by each process in turn.

2.7 Google Cartographer (Penny, Yatin, Fritz, Nico, Kristine)

Google Cartographer is software that can provide real-time SLAM in both $2D$ and $3D$ environments. As our MCL didn't include any mapping, being used only for localisation, we decided to also incorporate Google Cartographer into our simulation environment. As part of this process, we attempted to follow instructions in a github provided by RSS course staff.

During this process we encountered a number of errors. Multiple packages were not automatically imported and had to be manually installed. Furthermore, ros-melodic had be to resourced multiple times, despite us creating a separate workspace entirely for google cartographer. Once the isolated workspace had been built though, we were able to import a cartographer config from the staff's github to our regular racecar workspace and then launch Cartographer in our simulator.

Here, we faced another roadblock. To properly map the environment, we wanted to drive fully around the virtual map space we were provided, however, our wall follower solution was prone to following specific loops whereas our parking controller did not work at all due to the lack of a transform existing between the

cartographer's map frame and our car's frame body. This was because the simulator still launched its own distinct map frame that we had to work around. Hence, we explored solutions for manually driving the car in simulation.

The first angle we attempted was connecting joystick controllers, such as a PlayStation's ds4. However, the online drivers for this were broken and outdated. Hence, we instead installed the teleop-twist-keyboard package to be able to send messages directly from our keyboard controls. These messages were of type Twist and would not move the car on their own. Hence, we created our own separate node that would subscribe to the Twist messages and then publish AckermannStamped messages to drive our car in simulation. With these new controls, we were able to explore our map environment in google cartographer. We recorded a video of this SLAM in action which even included a visible loop closing event.

2.8 Real World Deployment (Penny, Yatin, Fritz, Nico, Kristine)

After successfully modeling noisy sensor data and odometry data, as well as implementing a particle filter that was able to successfully localize the car in the simulator, we attempted to localize the physical race car in the Stata basement. During this process, we encountered various technical and infrastructure issues, which we tackled in sequence. The car had battery issues, which prevented us from debugging our particle filter transformation implementation physically. We quickly resolved this by exchanging batteries with another team.

Next, we encountered our largest roadblock. While trying to run the particle filter, we expected that the car's sensors would start streaming data to the filter node and publish to the topics we could access in Rviz. Unfortunately, the particle filter neither published the dispersed particles nor the odometry data from the car, making it difficult to determine the failure mode of our implementation.

After ensuring that all the parameters were modified so that the node would access the car's initialized topics and tracing through the particle filter code to ensure the math was correct, we realized that the node running on the car did not self-initialize its current pose. We would have to manually initialize its pose in the Stata basement, much like in the simulator. However, this process was more difficult, as it required finding a location where we could consistently align the laser scan data with the map and setting it as the origin to initialize the car's pose. This location-finding procedure would not have been necessary if the particle filter was running in real-time since the initial position could have been set immediately to where the node was first activated. However, more optimization of the code is needed for this step.

3 Experimental Evaluation (Penny, Yatin, Fritz, Nico, Kristine)

3.1 Overview

Our experiments on our particle filter in simulation yielded valuable insights into its performance. We structured these tests first by testing the aforementioned two methods of computing the optimum particle from the filter (Expected value vs maximum likelihood particle **(**(Arg_Max))). We tested each method's performance across different speeds and noise levels. We also tested two noise distributions: zero-mean Gaussian and zero-mean continuous uniform.To additionally test the robustness of the particle filter in simulation, we analyzed qualitative convergence rates – how quickly the error converges – with mean squared positional error **(MSE)***.
For expected value particle and maximum probability particle:

$$MSE = (ground\ truth\ position - predicted\ prediction)^2$$

*Note that Cross Track Error is the distance between the current position and the correct path, adjacent to the square root of MSE. It can be used for the same purpose of evaluating the particle filter.
In the live testing, we experienced hardware challenges on the racecar we shared with other teams that hindered the collection of numerical evidence as described in section

3.2 Simulated Experimental Results (Penny, Yatin, Fritz, Nico, Kristine)

Each test consisted of a complete lap around a rectangular section of the virtual map, passing obstacles. An Example Video Here
In addition to quantitatively measuring the mean squared position error between the car and the optimal particle throughout the tests, we qualitatively measured the convergence behavior. The x-axis of the graphs are not in seconds but are rather proportional to the time it took to complete the test. The MSE along the y-axis is not normalized and is measured in square meters. The results and analysis of those tests are as follows:
Expected Value particle and Gaussian Distribution Mean Squared Positional Error
The squared error trends show that mean error decreases at all speeds with increasing standard deviation of the noise. Surprisingly, the error throughout the tests also decreases with increasing speed. Initially, we believed that this might be solely attributable to the higher speed leading to less time for error to accrue over the test, but observing the tests confirmed that the particles tracked the car much more closely in both angle and position at higher speeds. The best performing test was done at the highest standard deviation and highest speed: 0.06 and 10 m/s respectively. We believe that increasing standard deviation

Fig. 5: Expected Value particle and Gaussian Distribution Mean Squared Positional Error

results in better performance, because having particles that span a wider range on average allows the particle filter to better "react" to rapid changes in position, as there are more likely to be particles at a given time at the new position. 5

Maximum probability particle and Gaussian Distribution Mean Squared Positional Error

The performance trends of the particle filter using the maximum probability particle closely resembled those of the expected value particle, but the error metrics were higher across the board. We believe that this may be due to the following reason: The max probability particle changes very rapidly as the car moves, especially in the presence of outliers which can cause error to accrue more quickly over time. However, the expected value is more robust to outliers. 6

When qualitatively characterizing the error convergence behavior across tests, the following criteria was used: A score of poor was given to tests whose positional and angular particle/ground-truth error diverged to reach high values throughout the test, or which remained steadily high. A score of fair was given to tests whose errors rapidly converged to low values but which still started with significantly high initial error. A score of good was given to tests whose errors showed slight divergent behavior but which remained low. A score of great was given to tests whose errors remained markedly low with slight variation throughout the duration of the test.

[]

Using a Gaussian distribution and the expected value particle, we noticed that at low standard deviations across speeds, the angular and positional error tended to converge to low values while starting at relatively high values. Both convergent and divergent behavior was observed with lower mean errors for intermediate

Fig. 6: Maximum probability particle and Gaussian Distribution Mean Squared Positional Error

Qualitatively describing the convergence behavior across tests

Expected Value	**1 m/s**	**5 m/s**	**10 m/s**
$\mathcal{N}(0, 0.02)$	Fair	Great	Fair
$\mathcal{N}(0, 0.04)$	Good	Fair	Great
$\mathcal{N}(0, 0.06)$	Great	Great	Great

Table 1

Expected Value Uniform	**1 m/s**	**5 m/s**	**10 m/s**
$\mathcal{N}(0, 0.02)$	Poor	Poor	Poor
$\mathcal{N}(0, 0.04)$	Poor	Poor	Poor
$\mathcal{N}(0, 0.06)$	Poor	Poor	Poor

Table 2

Arg_Max	**1 m/s**	**5 m/s**	**10 m/s**
$\mathcal{U}(\frac{-1}{50}, \frac{1}{50})$	Poor	Poor	Poor
$\mathcal{U}(\frac{-2}{50}, \frac{2}{50})$	Poor	Good	Good
$\mathcal{U}(\frac{-3}{50}, \frac{3}{50})$	Poor	Great	Great

Table 3

standard deviations. At the highest standard deviations, at low speed there was slight divergent behavior, while at high speed error was consistently low.**??**
[]
[]
Using a Gaussian distribution and the maximum probability particle, we observed that for most tests, the error was divergent and large in magnitude. However at the higher end of speed and standard deviation, impressively convergent and low error behavior was observed. table. 3
The angular/position error across tests with uniform noise probability distributions immediately and rapidly diverged in all cases, showing slightly slower divergence rates at high speeds.2,4

3.3 Live Testing Results (Penny, Yatin, Fritz, Nico, Kristine)

We deployed our implementation of the particle filter on a physical race car and observed real time updating of particle positions using simulation. Fig.7 shows a side by side comparison of our car in the stata basement and calculated position of the particle filter, and a video of our live testing results can be found here at this link. In this test, the car ran the wall following code and the

Arg_Max Uniform	**1 m/s**	**5 m/s**	**10 m/s**
$\mathcal{U}(\frac{-1}{50}, \frac{1}{50})$	Poor	Poor	Poor
$\mathcal{U}(\frac{-2}{50}, \frac{2}{50})$	Poor	Poor	Poor
$\mathcal{U}(\frac{-3}{50}, \frac{3}{50})$	Poor	Poor	Poor

Table 4: Qualitative descriptor of optimal particle convergence across test using probability max to pick the particle positions. We found that calculating the particle position only using the max probability point resulted in poor convergence across different settings of noise and robot speed.

Fig. 7: Position of the car in real life in the stata basement compared to the particle positions calculated by the particle filter in Rviz shown on the right. The calculated particle positions in the Rviz correspond to a accurate representation of the car's position and orientation with respect to the stata map.

particle filter inferred correct positions of the car. Additionally, another testing video of the wall following code running in the stata basement can be found at this link - in this video, while the car drove closer and closer to the wall in real life, various hardware issues prevented the model car in the simulator from mirroring the position of the real car. This can be seen in the video, where the simulated car continues straight, the real car approaches the side wall, and the localization particles follow the position of the real car, since those are being updated correctly.

4 Conclusion (Penny, Yatin, Fritz, Nico, Kristine)

Ultimately, despite a number of road blocks during our technical design, we believe that we achieved the primary goal of this lab. As shown during our results video, our autonomous race car successfully managed to follow walls around an entire room, including the corners. However, this path − whilst common to many routes that the car may need to take − does not include edges cases that would challenge our design.

To improve our lab, we would need to have more quantitative measures of evaluating our results under different conditions, especially ones that target these mentioned edge cases. Our lack of these is partly due to time constraints but this is also a reflection of our larger time allocation. Collecting these measures earlier might have allowed us to adjust and fix some of our more complicated strategies and perhaps account for these edge cases.

In the end, it was disappointing to not be able to complete the corner cutting algorithm, albeit the maths involved did guide our later distance calculation when we reverted to a look ahead approach for turning. All was not in vain though as our work on Hough Transforms was encouraging. We can apply a lot of what learnt (both specifically about the transforms and more generally about

procedure) towards the next lab involving computer vision.

5 Lessons Learned

5.1 Penny

After spring break, I had a disruption in my travel schedule and was stuck abroad for 2 days without wifi, which meant I couldn't work on the lab. I found the lab instructions at times hard to follow, and as a result we had multiple versions of the code being developed until we merged them. However, in the end I'm proud of how my team handled our challenges and worked through this lab. In this lab, I learned valuable lessons regarding time management, programming, testing, and debugging, as well as presenting our work. I'm excited to learn more about localization and trajectory planning!

5.2 Fritz

During the lab, I learned the importance of understanding the trade-off between time management and quality in engineering. While I greatly enjoyed coming up with interesting algorithms to tackle the problem at hand, I realized that it is easy to fall into a long spiral of debugging during their implementation. And just as in industry or academia, time is not unlimited. To respect mine and my teammates' time, and to meet the assignment's deadline, I had to learn to compromise on the complexity of the techniques we used.

5.3 Nico

The hardest challenge with this lab was adjusting to continual adversity. For myself personally, an infected injury during Spring Break led to a rough start to the lab where I was not in a healthy position to contribute to my team until quite close to the initial deadline. When this was compounded with Penny dealing with personal circumstances that affected her own involvement, an unfair burden was placed on Kristine, Yatin and Fritz, hence we all understandably fell behind schedule. However, as a team, I am proud of the way we have been working hard to catch up. Allocating time between ourselves to both labs 5 and 6 simultaneously and attempting to optimise our time.

From a technical perspectives, we also faced a number of unexpected challenges. From teleop breaking regularly, to having issues displaying in rViz. At one point during the course of this lab, my catkin became so corrupted that the TAs recommended that I delete my entire docker image and start fresh. Since a lot of these issues are difficult to predict, I think the main takeaway from this is learning more ways to generally debug or reset my environment. It's also a

lesson in allocating extra time for unexpected errors.

5.4 Yatin

This lab was difficult. Ito took me a while to fully understand the steps needed to guess the location of the car given noisy data and little prior information, which made it harder still to find issues in the code that I wrote. My teammates were instrumental in aiding my understanding, and we were able to tackle the myriad hardware and software issues that got in our way during this lab. However, I learned a lot, and would love to have a better experience learning about robotics techniques in the future.

5.5 Kristine

This lab was quite interesting since we had a lot of flexibility in our implementation of the particle filter and additions to localization. Additionally, since a large part of this lab was over Spring Break, I learned lessons for planning, time organization, and momentum for completing a project. Additionally, as the lab drew to an end, I realized the car had technical difficulties and couldn't run our code, which might've been resolved faster if we planned to use car earlier and coordinated different with the other teams. Importantly, as other team members have mentioned, it is critical that we fully understand our implementation and how to evaluate it by our deadlines, an imperfect technical solution but sound reasoning, rather than spending long hours debugging and fixing code.