Lab 5 Report: Localization

Team 16

Lasya Balachandran
Fiona Gillespie
Adrian Gutierrez
Katherine Lin

Robotics: Science and Systems

April 15, 2023

1 Introduction *(Adrian)*

Our group developed localization on our race car so that our car would be able to estimate its position within a given map. We were able to develop this program with our use of Linux, Git, and the ROS skills we have gained over the last few weeks.

Localization is essential in any autonomous vehicle. When our race car is able to localize itself within a given map it can then start to make autonomous decisions within that map to avoid walls and reach goal points. Through this lab, we were able to develop a simulated version of our car operating its localization program in RVIZ and attempted to translate the program from simulation over to hardware.

For this lab, we are gathering Odometry and LiDAR data from our race car to produce an estimation of our position within our map. To do this we implemented a Monte Carlo Localization (MCL) program that uses a sensor model and motion model to compute our estimated position.

Our motion model receives Odometry data from the race car which is then used in conjunction with our previously estimated position to compute our new position for the next time step. Then our sensor model uses a probabilistic model to later refine the estimated position of our race car. These two run jointly to build our MCL implementation.

2 Technical Approach *(Adrian, Lasya, Fiona)*

In order to create an MCL program that could determine a race car's position and orientation in a known environment, we focused on three main aspects: a motion model to estimate the position of the car, a sensor model to predict the likelihood of particle poses on future iterations, and a particle filter to remove unlikely particles and update the particle poses based on the motion sensor models.

2.1 Motion Model *(Adrian)*

Our motion model was created to estimate our position given Odometry data gathered by the race car and an estimate of our position at the previous timestep. Using these two components and by adding some noise to our odometry we are able to get a sufficiently good estimate of our position.

2.1.1 Changing frames *(Adrian)*

An important aspect of the motion model is making sure all our calculations are being done with respect to the world frame. Figure 1 shows a good example of this. On the left, we see that the car's position p_r^W is in the world frame while on the right we see that everything, such as race car 2's position (p_{r2}^{r1}) is with respect to race car one's frame. Before we do any meaningful calculations we must make sure that any positions or changes in positions are all in the same frame. For us, that is the world frame since we are trying to localize within our given world/map.

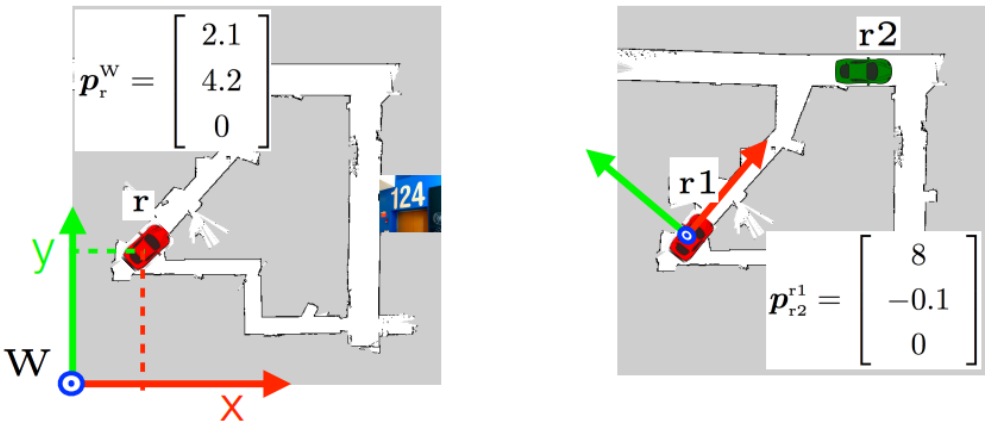

Figure 1: Coordinate Frames

The inputs we receive to calculate our new position are Odometry data and our previously estimated position. The problem is that the Odometry data is in the race car's frame while our estimated position is in the world frame. Equation 1 shows us how we go about changing frames and Section 2.1.2 shows how we use these new values to get our new position.

2.1.2 Estimated Positions *(Adrian)*

To calculate our new estimated position we receive two main inputs: our position recorded at the last timestep and Odometry data gathered by the race car. This position is a set of 200 coordinates, or particles, that define where we believe our race car to have been at the previous time step. Then our Odometry, which is a 1 x 3 vector of [Δ x, Δy, $\Delta\theta$] is applied to each particle. This is done through the equation

$$p_c^W = R_r^W p_c^r + p_r^W \tag{1}$$

where

$$p_c^W = \begin{bmatrix} x_{current} \\ y_{current} \\ \theta_{current} \end{bmatrix} \tag{2}$$

$$R_r^W = \begin{bmatrix} cos(\Delta\theta) & -sin(\Delta\theta) & 0 \\ sin(\Delta\theta) & cos(\Delta\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$p_c^r = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{4}$$

$$p_r^W = \begin{bmatrix} x_{previous} \\ y_{previous} \\ \Delta_{previous} \end{bmatrix} \tag{5}$$

By applying this equation to each particle we acquire the new position of each particle. However, to make our implementation more robust we add in random Gaussian noise defined with a mean of 0 and different standard deviations for each variable. For x it is a standard deviation of 0.07, y is 0.05, and θ is 0.07.

2.2 Sensor Model *(Lasya)*

We created a sensor model to assign likelihood weights to each particle with four main components: finding the probability of a scan given the ground truth and observed distance measurements of map obstacles (e.g. walls, pillars, etc) from the car, creating a precomputed table of the scan probabilities based on distances in pixels, converting the distances to pixels, and using the precomputed table to find the probability of each particle.

2.2.1 Finding the Probability of a Scan *(Lasya)*

In order to find the probability of a scan i given the car's estimated pose x_k, we considered the probabilities four possible situations: detecting a known object, a short measurement such as an internal LiDAR reflection or hitting parts of the car, a very large or missed measurement such as a LiDAR beam that hits an object and does not reflect back, and a random measurement accounting for any other possible situations.

1. **Known object:** We represented the probability of detecting a known object as a Gaussian distribution centered around the ground truth distance between the estimated distance and the nearest map obstacle. In this case, the probability is maximum if the measured distance z_k to the nearest obstacle is the same as the ground truth range d. We represented probability using Equation 6

$$p_{hit}(z_k^{(i)}|x_k, m) = \begin{cases} \eta \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z_k^{(i)}-d)^2}{2\sigma^2}} & \text{if } 0 \leq z_k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

   where η is the normalization constant such that the probabilities across all i possible scans for a given d sum to 1.

2. **Short measurement:** We represented the probability of a short measurement was represented using a downward sloping line since the LiDAR was more likely to hit closer objects. We represented probability using Equation 7

$$p_{short}(z_k^{(i)}|x_k, m) = \frac{2}{d} \begin{cases} 1 - \frac{(z_k^{(i)})}{d} & \text{if } 0 \leq z_k \leq d, d \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

3. **Very large measurement:** We represented the probability of a very large measurement by a spike at the maximum observed range value so that the measurements did not significantly affect the particle weights. While the situation could be represented by the delta function $\delta(z_k^{(i)} - z_{max}$, we approximated it as a uniform distribution close to z_{max} for small ϵ. We represented probability using Equation 8

$$p_{max}(z_k^{(i)}|x_k, m) = \begin{cases} \frac{1}{\epsilon} & \text{if } -\epsilon \leq z_k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

   where $\epsilon = 0.1$.

4. **Random measurement:** This probability accounted for any other possible situation, and we represented it using a small uniform value, as seen in Equation 9

$$p_{max}(z_k^{(i)}|x_k, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Using these different situations, we then evaluated a weighted sum, as shown in Equation 10, to find the overall probability of the scan given a pose on a map.

$$p(z_k^{(i)}|x_k, m) = \alpha_{hit} \cdot p_{hit}(z_k^{(i)}|x_k, m) + \alpha_{short} \cdot p_{short}(z_k^{(i)}|x_k, m) + \alpha_{max} \cdot p_{max}(z_k^{(i)}|x_k, m) + \alpha_{rand} \cdot p_{rand}(z_k^{(i)}|x_k, m) \tag{10}$$

where $\alpha_{hit} = 0.74, \alpha_{short} = 0.07, \alpha_{max} = 0.07, \alpha_{rand} = 0.12$, and $\sigma = 8.0$.

2.2.2 Precomputed Table *(Lasya)*

For computational efficiency, we then discretized the input distances z_k and d by converting the distances to integer pixel values from 0 to 200 and created a table with each of the possible probabilities based on equation 10. For p_{max}, we assumed ϵ to be 1 pixel, so the function was 1 at z_{max} and 0 otherwise. The equations of p_{hit}, p_{short}, and p_{rand} were the same as in Equations 6, 7, and 9. After finding the value of each cell in the table, we normalized the values across each column such that for every d, all of the z_k values summed to 1. We generated Figure 2 based on this precomputed table.

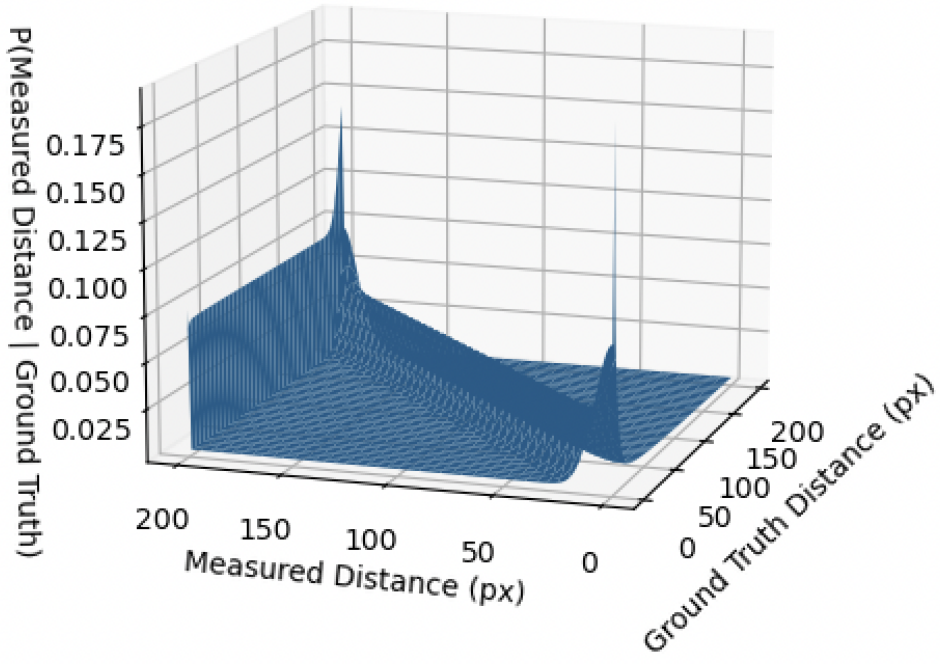

Figure 2: 3D Plot of Precomputed Table of Probabilities of Observed Distance Given Ground Truth Distance

2.2.3 Converting to Pixels *(Lasya)*

In order to convert our n x 3 array of particle poses $[x, y, \theta]$ from meters to pixels, we first passed the array into the given ray tracer to produce an n x m array of ground truth ranges, where m was the number of LiDAR beams per scan. This procedure resulted in n sets of LiDAR messages with each set corresponding to a particle pose. We also divided each of the distances by the product of the map resolution and the LiDAR scale to map scale of 1.0 to convert the observed distances from meters to pixels.

2.2.4 Using Precomputed Table to Find Probabilities of Particles *(Lasya)*

Once we converted the distances to pixels, we found the overall probability of each particle. The probability of a particle pose occurring is the probability of the observed distance occurring given each of the corresponding LiDAR distances. We independently considered each of the LiDAR distances with respect to the observed distances, so we expressed the overall probability of each particle pose using Equation 11.

$$p(x_k^{(i)}) = \prod_{i=1}^{m} p(z_k^{(i)} | x_k, m) \tag{11}$$

We also squashed the probabilities by raising them to the power $\frac{1}{2.2}$ to make the probability distribution less peaked.

2.3 Particle Filter *(Lasya)*

Using our motion and sensor models, we then constructed the Monte Carlo Localization (MCL) algorithm to remove unlikely particles. Our particle filter is composed of four main components: initializing the particles, using the motion model to update the particle positions, using the sensor model to resample the particles, and finding and publishing the average particle pose as a transform. Figure 3 shows a block diagram of our particle filter.

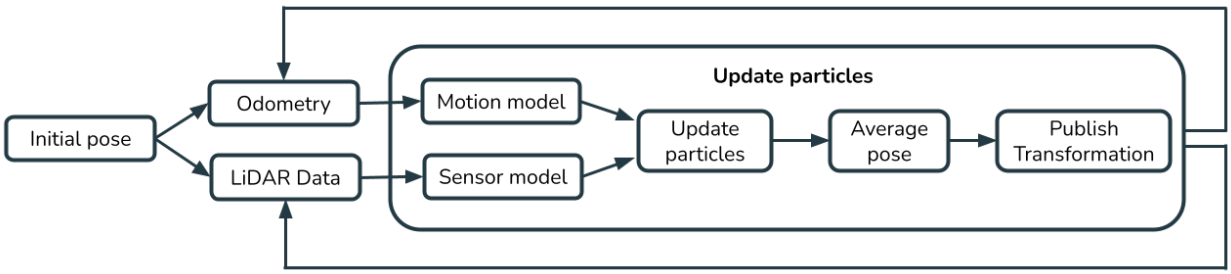

Figure 3: Block Diagram of Particle Filtering System

2.3.1 Initializing Particles *(Lasya)*

We first initialized a 2D pose estimate, which returned a PoseWithCoverianceStamped object. We then initialized 200 particles as a 200 x 6 array using a Gaussian distribution with the columns of the array representing the x position, y position, z position, roll, pitch, and yaw. For the mean of the Gaussian distribution, we used the x and y position values from the pose estimate for the x and y positions and the z value from the Euler from quaternion conversion of the orientation for the yaw. All of the other values were 0 since the car does not move in those directions. After creating the particles, we removed the third, fourth, and fifth columns to create an array of 200 particle poses representing $[x, y, \theta]$.

2.3.2 Using Motion Model to Update Particle Positions *(Lasya)*

Our car periodically received odometry which could be passed into the motion model to update the particle positions. We first found the change in time, dt, between successful calls of the odometry callback and multiplied this by the x position, y position, and z orientation of the velocity to obtain the distances dx, dy, and $d\theta$, respectively, traveled in dt. After finding dx, dy, and $d\theta$, we passed in the particles and a list of [dx, dy, $d\theta$] into the evaluate function of the motion model to create a new set of particle positions.

2.3.3 Using Sensor Model to Resample Particles *(Lasya)*

Our car also received LiDAR data which could be passed into the sensor model as the observed distances. We first ray casted the particles to LiDAR distances and converted the estimated and observed distances to pixels. After converting the distances to pixels, we used the precomputed table to find the probabilities of each of the particles and normalized the probabilities across all of the particle poses to find their relative likelihood. We then resampled the particles according to their probabilities.

2.3.4 Finding and Publishing Average Particle Pose *(Lasya)*

After updating the particles through either the motion model or the sensor model, we determined the average particle pose which could be published as part of an Odometry message. We calculated the average x and y positions using a weighted sum and calculated the average angle using a weighted circular sum, as shown in Equations 12 and 13.

$$(\bar{x}, \bar{y}) = \sum_{i=1}^{n} p_i \cdot (x_i, y_i) \tag{12}$$

$$\bar{\theta} = \tan^{-1}\left(\frac{\sum_{i=1}^{n} p_i \cdot \sin(\theta_i)}{\sum_{i=1}^{n} p_i \cdot \cos(\theta_i)}\right) \tag{13}$$

We set the x and y positions of the odometry pose to $\bar{x}$ and $\bar{y}$, respectively. We also converted the orientation of the pose representing roll, pitch, and yaw from Euler values (0, 0, $\bar{\theta}$) to a quaternion. All other pose values were left as the default value of 0. This pose was published to "/pf/pose/odom." The odometry was also transformed between the map frame and the car's expected base link ("base_link_pf" for simulation and "base_link" on the hardware) to make the LiDAR points line up with the map.

2.4 Simulation to Hardware *(Fiona)*

After implementing our Monte Carlo Localization algorithm, the final step was refactoring and tuning our system to work on the hardware. In theory, there were only a few key components that needed to be changed to transition from simulation to hardware. (The practical transition will be further discussed in Section 3.2.) The main differences on real hardware were the LiDAR data, ROS topics, and particle initialization.

2.4.1 Downsampling LiDAR Data *(Fiona)*

The real LiDAR scan uses over 1,000 beams, which provides high granularity. However, many of these beams are redundant. To improve computational performance, we downsampled to just 100 beams without loss of information. This downsampling process involved reading every 11th beam to still provide even coverage over the whole scan.

2.4.2 Updating ROS topics *(Fiona)*

When moving from simulation to hardware, some of the specific topics that are published to must be modified. In particular, the odometry is read from `/odom` in simulation, but in hardware it was updated to be read from `/vesc/odom`. The particle filter frame was also changed from `/base_link_pf` in simulation to `/base_link` on hardware. As these changes are subtle and hard to keep track of in a single file, we used two separate launch files. This allowed us to easily run `localize.launch` for the simulation configuration and `localize_real_env.launch` for the hardware configuration.

2.4.3 Initializing Particles *(Fiona)*

In simulation, there was a simple button in the visualization software to provide a 2D pose estimate. However, this requires a way to display a graphical program. When running software on the racecar, a laptop must use secure shell (SSH) to connect to the racecar's computer. This process unfortunately does not include a way to display graphics. So, the visualization program is run on the laptop instead, but due to computer security protocols, the visualization button on the laptop cannot directly send information to the racecar's computer. Instead, we followed the staff's advice of determining the initial pose just on the laptop, copying this information, and then sending it directly between the laptop and the racecar without the visualization software. This process worked surprisingly well, as our LiDAR data gathered from the real world aligned nicely with the map in the visualization. Figure 4 shows side-by-side the LiDAR data projected on a map in simulation and the physical racecar in the Stata basement

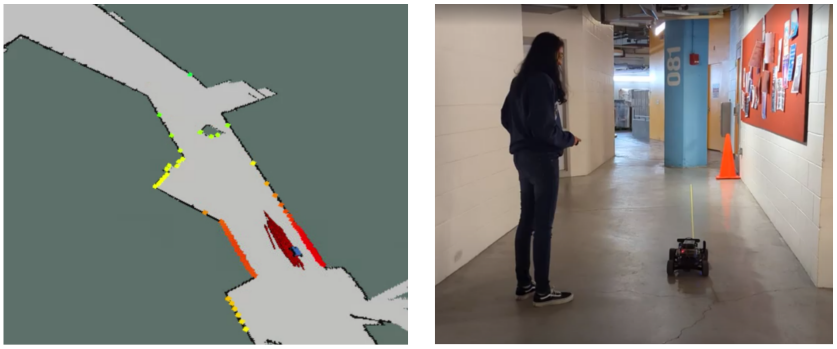

Figure 4: Our initial pose estimate worked well on hardware. The left picture (a) shows the visualized pose estimate with the LiDAR data well aligned to walls. The right picture (b) shows where the physical racecar was in Stata basement, facing a column.

3 Experimental Evaluation *(Katherine, Fiona)*

We evaluated our localization algorithm in both simulation and hardware. We were able to successfully validate our algorithm in simulation, testing important metrics such as cross-track error and robustness to noise in the odometry data. The hardware implementation was able to hold an initial position estimate and localize reliably within a short-range, but it was unreliable past approximately 15 feet from the initial position. Due to time constraints, we were unable to fully debug this last issue. We will discuss our evaluation process for both simulation and hardware in the following subsections.

3.1 Simulation Evaluation *(Katherine)*

Once we had a working particle filter implementation, the next step was to evaluate how effectively our particle filter could find the odometry of the car. To do this, we looked at several possible things to measure.

To start, we compared our distribution of particles to the average pose that we took to be our final estimated odometry.

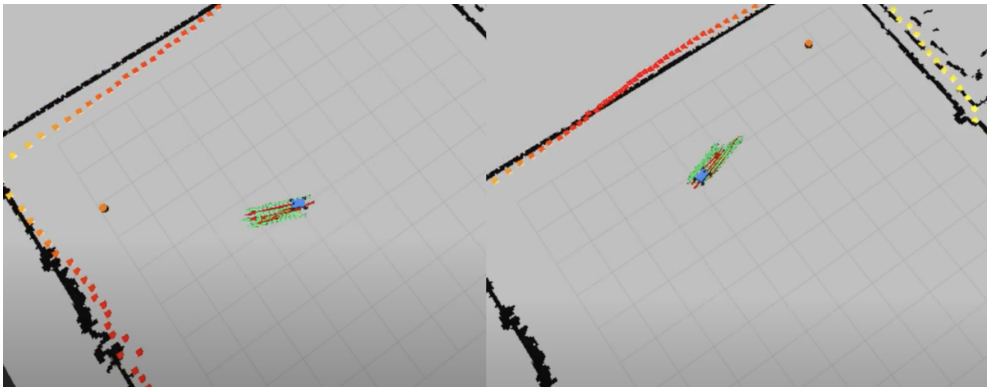

Figure 5: The green pointers are particles in the distribution we generated, and the red arrows are the averages we took of the distribution. The red arrows were our final estimated pose.

In Figure 5, it is evident that within the wide range of green pointers, the red arrows appear to cluster much tighter around the true pose of the car. Our final calculated pose appeared to be much more accurate and consistent than the distribution, showing that the average that we took was a good heuristic that improved our particle filter's accuracy.

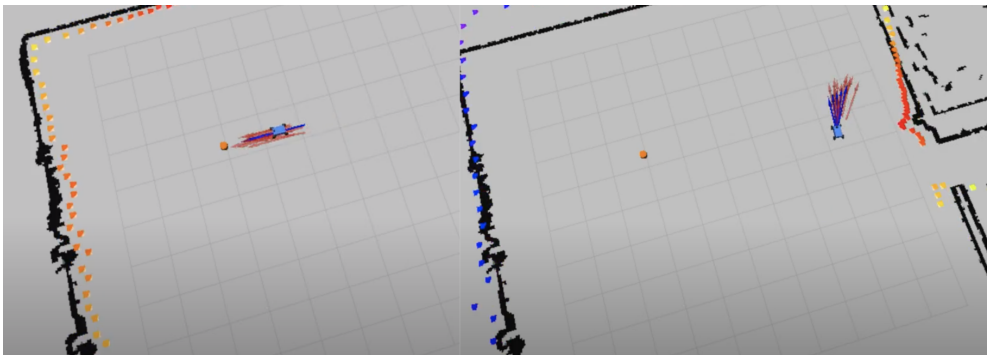

Figure 6: Again, the red arrows mark the estimated pose we calculated, whereas the blue arrows show the ground truth. On the left, the car is traveling in a straight line, whereas on the right, the car is turning.

We also compared our estimated odometry to the ground truth in Figure 6. Something to take note of is that we kept track of more red arrows at a time, meaning some of the red arrows are from several time frames ago. This can explain some of the "lagging" that some of the red arrows appear to do in the figure. While there does appear to be some variation in the red pointers unexplained by the lagging, we can see that they cluster tightly around the blue, and so are quite accurate.

Finally, we looked at what the laser scan would look like if the robot were actually at the estimated pose. The default laser scan is what the robot's LiDAR sees (as a ground truth). However, if we transform this laser scan into the frame of the estimated odometry, then we can see what the robot would see if the estimated odometry were completely accurate. The closer the estimate is to the ground truth, the better the transformed laser scan will match with the wall. In both Figures 5 and 6, we are shown the transformed laser scan. It largely hugs the wall, as seen on the left of Figure 6 and right of Figure 5, with some small shifts off due to errors in orientation as pictured in the left of 5. In all of these visual metrics, our particle

filter appears to be quite successful.

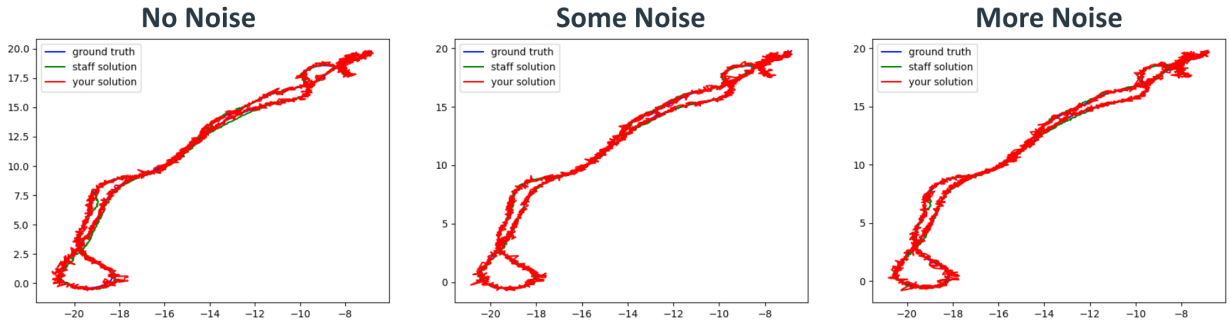

Figure 7: Given a ground truth path that the car traveled in simulation (in blue), we turned our particle filter's estimated odometry from the path into its own path (in red). On the left, there was no noise injected into the odometry, in the middle, there was some noise injected, and on the right, there was more noise.

However, we also evaluated the data quantitatively to assess its consistency. To this end, we first looked at the overall path generated by our estimated odometry, versus the true path traveled, with varying noises injected into the odometry. Figure 7 shows our estimate right on top of the ground truth, regardless of the varying noise. Overall, the time-average deviation from the trajectory was about 0.25 m for all three trials, compared to the staff solution, which was about 0.20 m. We can conclude that our particle filter is consistent given its performance, and it is robust to noise.

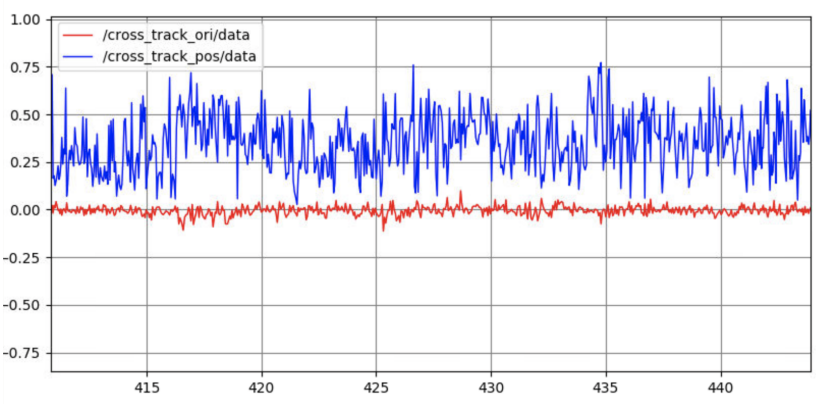

Figure 8: Cross-Track Error of a Given Path

We also looked at cross-tracked error for both the orientation of the odometry and the position. Figure 8 is a graph corresponding to the difference between the ground truth and our estimated odometry, produced by a simulated car on a randomized path. The position error primarily stays between 0.25 and 0.5 m, and the difference in orientations between the truth and the estimate are very close to zero. So we can conclude that our particle filter was effective and consistently had low error.

3.2 Hardware Evaluation *(Fiona)*

After verifying that our Monte Carlo localization implementation was robust in simulation, we modified our system for hardware as discussed in Section 2.4. However, we faced many challenges during our transition

to hardware which will be discussed in the following subsections.

3.2.1 Unexpected Joystick Behavior *(Fiona)*

One large problem we encountered was that the joystick commands would not match the racecar's position in simulation. At the beginning of this week, we noticed that the car in simulation would only move forwards or backwards along a single axis, regardless of what joystick commands were used to control the physical car. Our first idea to resolve this was to redownload the `base` folder, which handles all of the joystick code. We reasoned that perhaps a file had been changed erroneously, which could cause this unexpected behavior. However, that did not resolve our issue, so we went to office hours to look for more ideas on debugging this behavior. The next day in office hours, another team had experienced the same issue on the same racecar that we were using for testing. Since two teams with different code had encountered the same issue on the same racecar, the staff suggested we test our system on another car. This was a very helpful idea, as it should help us determine whether the issue was caused by our code or by the hardware. We were able to test our code on another car, and the joystick behaved as expected - the simulated car and physical racecar both moved the same way.

3.2.2 Tuning Noise *(Fiona)*

After the joystick issues were resolved, we next wanted to check if the position estimate seen in the visualization matched how the physical car moved in Stata basement. Our first question was after the particle initialization, what would happen if we did not command the car to move? Ideally, the position estimate should also not move, since that would match the ground truth. However, we noticed that our position estimate would drift significantly, even showing the estimate moving down the hallway and turning a corner, despite the physical racecar never moving. Figure 9 shows this an example of the far extent of the drifting behavior after 11 seconds.

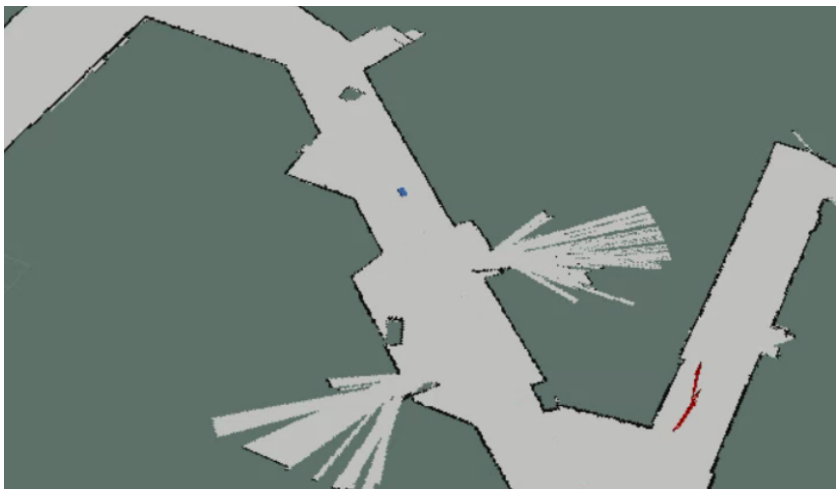

Figure 9: Although the car was initialized to the position shown by the blue car in the middle, and never commanded to move, the position estimate severely drifted, even turning a corner and reaching the pose shown by the red arrows after 11 seconds.

An initial explanation for this was perhaps part of our particle filter implementation was wrong or had not been correctly updated from simulation to hardware. We went over our implementation almost line-by-line with several TAs, but they did not find any large discrepancies.
The next idea was perhaps we were adding so much noise that the particle filter found it likely that we were moving. During the simulation development, we had experimented with different amounts of noise. We had also experimented with adding noise to either the incoming odometry data, the updated particle poses, or both. During initial hardware testing, we had raised the noise but likely by too much. Since we

were experiencing significant drift, we tried only adding noise to the updated particle poses and we reset the noise level to the best result from the simulation. After these changes, we again initialized the car and did not command the car to move. This time, the position estimate was also stationary in the visualization, as seen in Figure 10. Tuning the noise was key in fixing the large drift issues.

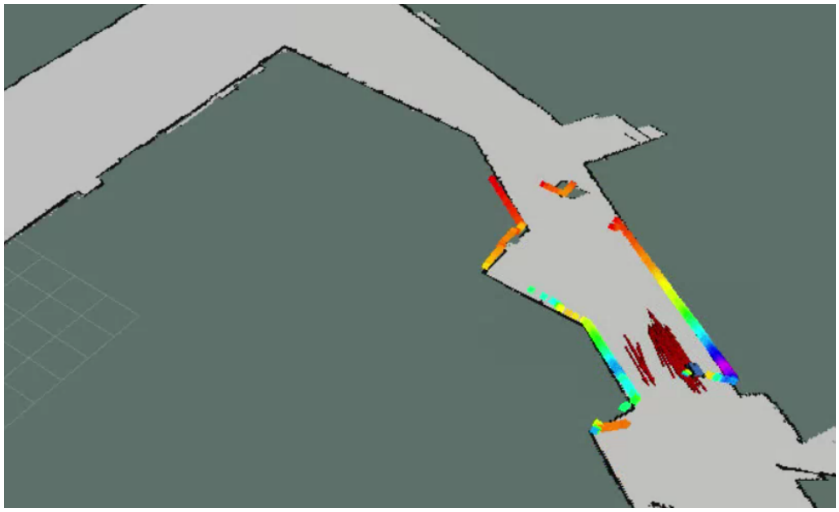

Figure 10: The car was initialized to the position shown by the blue car and not commanded to move. After reducing noise, the position estimate is shown by the red arrows. This time after 11 seconds, the red arrows and blue car align well, which is an improvement over the large discrepancy in Figure 9.

3.2.3 Range Testing *(Fiona)*

After fixing the drifting issues, we started to test the range of reliable localization. We started the racecar outside of 32-082 and drove it down the hallway. However, the estimated position only went as far as the red arrows in Figure 11, which was approximately 15 feet.

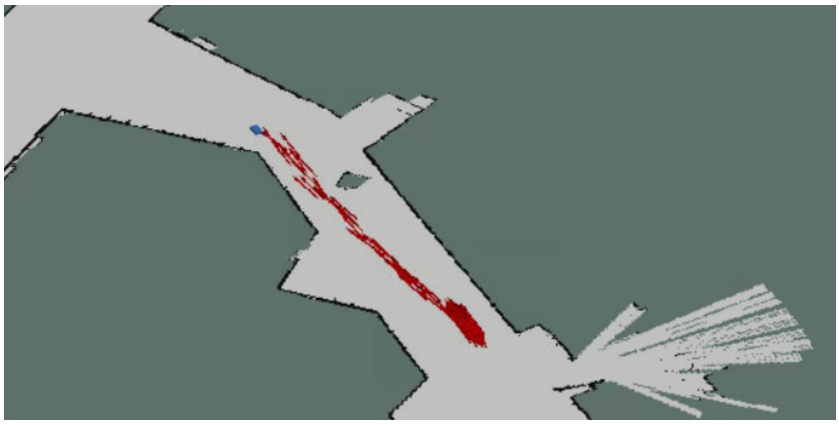

Figure 11: The localization is reliable for about 15 feet, as shown by the range of the red arrows. Beyond this range, the localization produced inaccurate accurate results.

Past this range, the physical racecar and estimated position did not match. The racecar was driven down the hallway and to the left, but the estimated position moved very unreliably. Figure 12 shows a test where the car is visualized to have turned right, but the physical racecar in this test had turned left instead. Furthermore, the blue car's location on the map does not match up with where the red odometry markers appear. This discrepancy was another point of confusion for us.

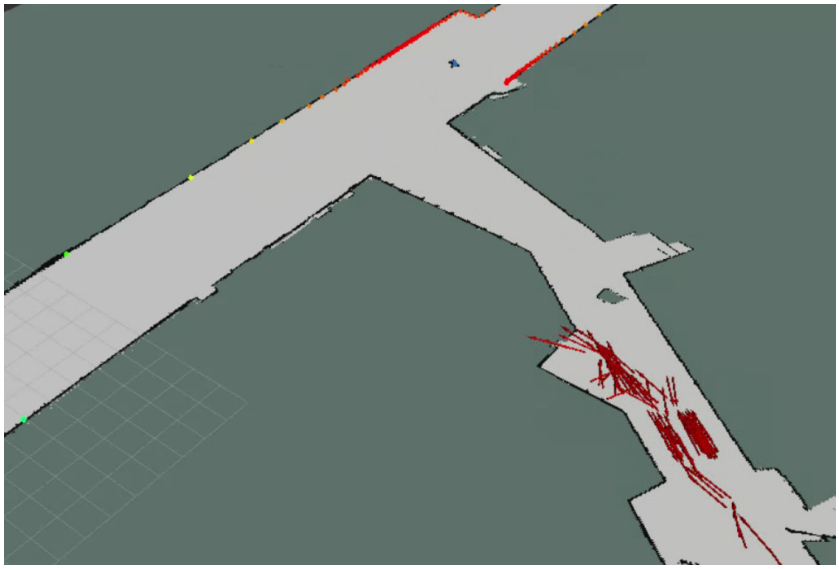

Figure 12: The localization past approximately 15 feet is unreliable. Here the car is visualized to be on the right of the far hallway, when the physical car had actually turned left.

Due to the time constraints and tight turnaround between this report and the next lab, the staff strongly suggested to wrap up this lab and switch to the upcoming lab. Therefore, we unfortunately were not able to fully debug this range issue. Despite this limitation, it was still exciting to have a reliable initial pose estimation and short-range localization solution validated on hardware.

4 Conclusion *(Katherine)*

In this lab, we designed a module that would allow a robot to localize where it was on a map as it traveled through space. This module was the result of combining a sensor and motion model into a particle filter that would return the estimated pose the robot was at. Though our algorithm was not without flaws, we were able to localize successfully in simulation, as evaluated through extensive qualitative and quantitative means.

On hardware, we were able to hold an initial pose estimate and successfully localize over a short range. A key limitation and next step for the future would be to improve the localization to handle longer ranges. However, no robot can be autonomous without additional algorithms. We hope in the future to improve what we noted above but also take advantage of path planning to integrate this module with previous modules we've worked on, making this robot equipped to autonomously operate in more robust scenarios that will imitate real world cases.

5 Lessons Learned

5.1 Lasya

For the technical part, I became more familiar with localization as a whole since I did not have much experience in this area before. I also learned gained a lot of experience debugging our car. We originally ran into several issues with our odometry not appearing as we expected, and I think this will definitely help us going forward in trying to detect issues in our solution.

For communication, I think our team generally communicated well, letting others know what we had worked on and any issues with the car. I think we also communicated well regarding dividing up the work, especially in the beginning when we worked on the motion and sensor models. However, while we were able to split

the work, I think that we could have improved on explaining the parts we did to the rest of the team so that it was easier to debug if the person who worked on the part wasn't there. In addition, I think handoffs between other teams in the pod has improved since the first lab. Multiple teams in the pod ran into issues with our two cars, and we were all able to exchange batteries/cars to accommodate the other teams.

In terms of collaboration, our team originally divided up the work between the motion and sensor models, which I found to be helpful in terms of efficiency. We also ran into issues with submitting on the autograder on Gradescope, and we were able to streamline the submitting process by relying on GitHub. In addition, we took turns running the launch files so that we all knew how to test the car.

5.2 Fiona

Overall, this lab has been the hardest yet that we've faced - on the technical side and with hardware. Despite these challenges, I think as a team we have developed significantly and still made solid progress.

The technical portion of this lab proved quite challenging. The technical approach was much more involved than previous labs, and also included material that was very new to all of us. There were also some advanced software techniques such as multi-threading. Since this lab was larger in scale, debugging became much more difficult. Even when we went to office hours, the TAs would also not be very sure in what our problems were. I personally struggled with the sheer amount of time we'd spend both in office hours and in lab on our own, with minimal improvements and results to show for the effort. Ultimately, we were able to achieve very strong results in simulation and some results in hardware. A lesson learned here was balancing trying to get something perfect with knowing when it is time to move on to other tasks.

For hardware, we faced many odd bugs and challenges. Our joystick would suddenly start driving the car strangely, or moving the car in simulation when it shouldn't be able to. It was difficult to isolate whether there was a code issue or hardware issue, which made debugging slow.

In terms of collaboration, we've also learned a lot in this lab. There were some debugging-heavy work sessions where it was difficult to get all four of us actively involved. In the future, it could be helpful to have a mini team meeting at the beginning of a work session to discuss what needs to get done and how best to use everyone's time to make it happen. It would be great to also rotate who is "leading" the mini meeting so we all get experience with that. We've also improved collaboration throughout our pod. Despite the more technically challenging lab, handoffs seemed smoother as we've built stronger connections with the other teams in the pod.

With all these lessons learned, I'm looking forward to moving into lab 6 and the final challenge with my team.

5.3 Adrian

With each lab, I believe the teams within our pod are working more efficiently to solve any issues we might be having. We are able to make smooth handoffs and communication through our channels is seamless. Even when it seems like some problems might be forming we are always all quick to address them. Within my team, it definitely can be an issue to decide when we can all work since we are all busy but even so I believe we have been adequately communicating to make sure we all are largely working together to solve the problems in our lab. We are also able to partition our work so everyone can feel like they contributed and learned something from every aspect of the lab. I do believe we could improve upon the efficiency of our work sessions. Sometimes we may come together to work but not all of us are quite sure what we are all going to be doing. This can sometimes slow us down. So communicating this offline first would be helpful for us.

The technical side has been more difficult. Although we have been getting much more familiar with how to run all programs and how to solve familiar problems (topics being incorrect, problems with packages not

being built, etc) we ran into a lot of issues with the new content of the lab particularly when we attempted to put our implementation on the race car. And although we worked long hours on the lab, not all these issues were resolved. I believe a good way to make sure we are able to meet all the deliverables for our future projects is to ask our peers from other teams for help. This might give us some insight into where they might've made and resolved mistakes that we are facing.

5.4 Katherine

This lab was the most technically difficult for me thus far, and I definitely struggled with getting off the ground and gaining a basic understanding of how our particle filter could find the pose of the robot. However, I think that our team's coordination and teamwork has grown by bounds since we first met, and they were super helpful in that endeavor. Technically, the rest of the team also struggled, although that was through a combination of Gradescope issues and car issues as well.

Our greatest strength that we've learned is our willingness to try out any suggestions a team member may make, even if it may be time-consuming to implement; through this method, we were able to substantially improve our simulation particle filter. This is an indication of our team dynamic, as well as our pod dynamic. As we've run into increasingly challenging issues with our cars, our pod has been increasingly accommodating and helpful in suggesting possible fixes or informing us of potential problems.

Finally, something else that we've learned to do while collaborating is compartmentalization. Often, since not everyone can be editing the same files at once, it is hard to have all four people working on one thing at once. But when we split up into two groups, we were able to implement the sensor and motion models independently and more efficiently. It also allowed each team member a greater chance to contribute.