Lab #5 Report: Localization

Team #6

Tiffany Horter
Nisarg Dharia
Andrew Manwaring

6.141 - Robotics: Science and Systems

April 1, 2022

1 Introduction - Tiffany Horter

The goal of this lab was to develop a working implementation of localization for our racecar when given an initial position.

Localization is important to enable a higher level planning activity. If the robot does not know where it is, it can only react to its environment and cannot plan out its actions. To accomplish the end goal of creating a self-driving car, the ability to understand where a car is located in the map will allow for directions to be given to the car such as "drive to the mall" as the robot will now have an understanding of where it is in the world and where relative to that is its' end location.

To achieve this goal, we implemented Monte Carlo Localization to accurately determine the current pose of the robot. In order to implement the algorithm, we created a motion model and a sensor model and combined the two to create the final particle filter. The motion model uses the odometry data and the previous location to project the new location of the robot. As odometry data is inherently noisy, we added additional noise to account for possible imperfections in the data. The sensor model takes in the observed data from the lidar as well as the different particles and returns the probability of the robot being located at each particle. The particle filter combines these two models to determine the pose of the car and therefore localize the car, by weighting the particles from the motion model by the weights of the probabilities from the sensor model.

Through this lab, we explored the application of probabilistic models for determining the location of the robot and evaluated the performance of our Monte

Carlo implementation.

2 Technical Approach

2.1 Motion Model - Nisarg Dharia

Given the previous pose estimates of the racecar and a set of odometry data, the motion model's goal was to predict the new possible locations of the car. This was done by first converting the odometry data from the robot frame to the world frame and adding it to the previous pose estimate particles, and then incorporating some randomness to offset any noise in the odometry data.

Rotating the Odometry
Because the odometry data was given in the frame of the robot, the first step was to convert it into the world frame. For each particle i, we took the orientation θ_i and used it to build the rotation matrix

$$R_r^w = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix} \tag{1}$$

From this, we then took the particle's previous position $(x_i,\, y_i)$ and the change in position from the odometry $(\Delta x,\, \Delta y)$ to calculate the new position $(x_i',\, y_i')$ of each particle as

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} + \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{2}$$

Lastly, to get the new orientation of each particle, we simply added the previous orientation θ_i to the odometry's change in orientation $\Delta\theta$ to get the new orientation θ_i' as

$$\theta_i' = \theta_i + \Delta\theta \tag{3}$$

Adding Randomness Once we had the updated location of each particle (x_i', y_i', θ_i') we needed to add some randomness to account for noise in the odometry measurements. This was done by sampling an independent random value for the x, y, and θ components of each particle from a Gaussian distribution and adding it to the corresponding particle's pose. Given these random values $r_{x.i}, r_{y,i}$ and $r_{\theta,i}$, the final pose of each particle would then be

$$\begin{bmatrix} x_i" \\ y_i" \\ \theta_i" \end{bmatrix} = \begin{bmatrix} x_i' \\ y_i' \\ \theta_i' \end{bmatrix} + \begin{bmatrix} r_{x,i} \\ r_{y,i} \\ r_{\theta,i} \end{bmatrix} \tag{4}$$

Note that for each of the random values added, the Gaussian distributions were centered around 0 so as to not favor any particular directions. However, choosing the standard deviations was more difficult, as we found that as the odometry noise took on greater values, the standard deviation needed to increase as well. For the real car, we tested a range of values between 0.001 and 0.1 for all three directions (x, y, and θ). Ultimately, we used anecdotal evidence to determine that a standard deviation of 0.04 worked best in the x and y directions, while 0.03 performed best for the randomness in θ.

2.2 Sensor Model - Tiffany Horter

The goal of the sensor model is to develop a model of the probability that our car is in each particle's location. To do this, the sensor compares the scan data and the "ground truth" scan values of the ray-traced particles to determine the probability at each location. Essentially, the probabilities represent the fit to the observed lidar scan from each simulated scan from sampled particles.

Precompute sensor model table
Since it would be computationally expensive to recompute the probability of each scan as it comes in, we precomputed a table for the sensor model to be able to lookup the probability of measuring any given discrete distance for all distances the robot or simulated scans measure. To create the table, we made a 2D array with rows representing ground-truth values and columns representing observed distances, both ranging from 0 to the maximum scan value which we defined as 200. The probability of a given scan from an observation is determined by combining four probability scenarios detailed below, where z_k is the ground truth value, x_k is the hypothesis pose, m is a given map, z_max is the largest scan value, d is the observed distance, and σ is one of the scaling constants:
- (1) One possibility is that there is an exact robot scan match to a known map obstacle:

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

- (2) Another possibility is that it represents a measurement of the maximum range:

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

- (3) It is also possible that it is a shorter than ground truth measurement such as an unmapped obstacle (like a bike) or interference from the robot:

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

- (4) Finally, it could also be a random signal:

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

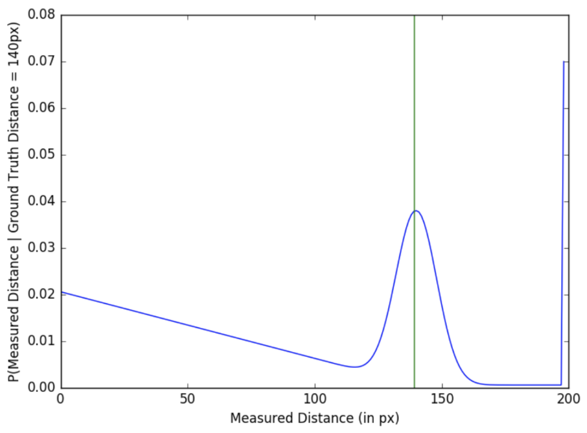


Figure 1: Probability distribution of the probability scenarios combined based on ground truth values and observed distances.

The probability of any entry in the table is the weighted sum of the probabilities of each scenario multiplied by their associated constant weight.

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

As the sensor model was performant with the default constant values of $\alpha_{short} = 0.07, \alpha_{hit} = 0.74, \alpha_{max} = 0.07, \alpha_{rand} = 0.12, \sigma_{hit} = 8.0$, we chose to keep the constants as they were. As probabilities for a given distance should sum to 1 by the rules of probability, we normalized the probability for the p_{hit} probability scenario and each value of d (i.e. across each column) to sum to 1. Finally, to maintain access to this table and only compute it once, we cached this sensor model table.

Evaluate how likely each particle is given the observed scan
For a set of particles and an observed lidar scan, the sensor model evaluates how likely each particle is to be the real location given the observed scan. Once

again, due to computational efficiency, we down-scaled the lidar data to 100 evenly spaced observation beams. To find the ground truth values, we then ray-traced the sampled particles since straight line values should remain constant. Then we scaled both the rays and the lidar to pixels by multiplying by the map resolution * *lidar_scale_to_map_scale* and clipped the resulting values to acceptable ranges between 0 and the largest scan distance. For visualization and testing purposes, we then published the pixelized lidar scan to understand what the robot was seeing. Because the values in the lookup table were discretized, we rounded lidar and scan to access the correct entry in the table. Then, iterate through all particles to find each one's probability of being the actual scan observed by multiplying the looked up probability of each beam at the given particle and distance.

$$\text{weight of particle} = \prod_{i=1}^{n} p(z_k^{(i)} | x_k, m)$$

Finally, we squashed the probabilities by raising them to the given power of $-1/2.2$ to flatten the probability distribution.
With the probabilities of each particle that the sensor model returns, we can use these as weights in the particle filter and therefore be able to determine the actual location of the robot.

2.3 Particle Filter - Nisarg Dharia

The main goal of the particle filter was to combine the estimated particle poses from the motion model and weights from the sensor model to determine the approximate pose of the car. To achieve this, secondary objectives such as initializing the car's location on the map and adding visualizations to better view the localization performance were also necessary.

Initializing the Racecar's Location
In order to begin localization, the car needed to know its approximate starting position. This process began by first waiting for a map to load, and then waiting for a pose to be selected on the map representing the car's approximate starting location. Once this initial pose was selected, the particle filter then generated 200 particles randomly distributed around this pose using a Gaussian distribution with mean 0 and standard deviation 0.5 to represent the possible starting poses of the car. Each of these poses was given a uniform initial probability of 1/200. Note that higher numbers of particles were tested but hindered performance too much to be usable. Finally, once this whole process was completed, a flag was set to signal that the motion model and sensor model were ready to begin running.

Updating Particle Locations
Once the particles were initialized, they were ready to have their locations updated as the car moved throughout the map. To do this, every time a new

odometry message was received, the particle filter first extracted the linear velocity in the x and y directions and the angular velocity around the z axis. Next, it took the time difference between the current odometry message and the previous one and multiplied it by the extracted velocities to get the displacements $(\Delta x, \Delta y, \Delta \theta)$. This displacement along with the current particle positions was then sent to the motion model, with the resulting particle positions replacing the current ones. Note that as a result of this process, the new particle poses are more spread out to reflect the uncertainty in the car's actual position, as show in figure 2.

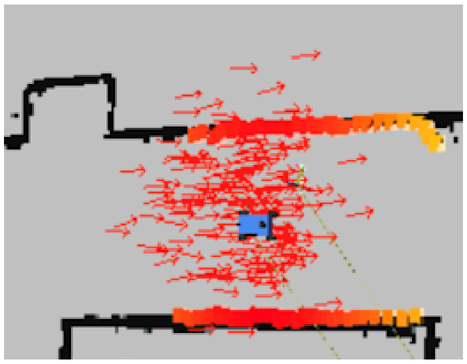


Figure 2: A spread of 200 particles displaying the possible poses of the car after going through the motion model, with each red arrow representing a single particle

Updating Particle Weights and Resampling After initialization, the particles were also ready to have their probabilities updated based on how accurately each one would have matched the LiDAR data. Thus, whenever LiDAR data was received, the scan was passed to the sensor model along with the particles in order to have the particle weights updated. Given these new updated weights, the particle filter then resampled 200 particles from the current set (including duplicates), with the probability of a sample being chosen equal to it's weight. This process resulted in a much more clustered set of particles representing the most probablistic poses for the car as shown in figure 3.

Determining the Average Particle Pose Given the locations and probabilities of 200 possible poses of the racecar, the particle filter then needed to determine what the single most likely pose was for the car. To do this, we first considered selecting the particle with the highest probability of being accurate based on the sensor model as the car's most likely position. However, relying on a single particle caused our average pose to jump around from particle to particle, which resulted in an erratic sequence of poses. Our second approach was to take an even average across all particles, but this again led to undesirable results. Specifically, it gave too high a consideration to particles that were

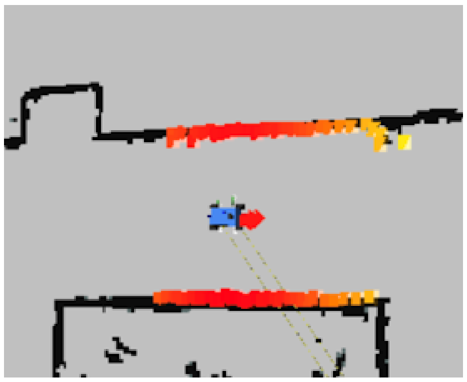


Figure 3: A cluster of 200 particles displaying the possible poses of the car after going through the sensor model and being resampled. Note that the large red mark in front of the car is actually 200 individual arrows representing the particles

unlikely to be correct, skewing the average away from the real location. In the end, the averaging method that gave us the most success was taking a weighted average, where the pose of each particle was weighted based on the probability that it was correct as given by the sensor model.

The weighted average approach had benefits over the max particle approach, as taking the average gave a smoother transition of the car's pose over time. It also helped alleviate some of the issues with the even average approach, as the heavier weights for the high probability particles meant that more consideration was being given to poses that were more likely to be correct. However, one drawback to this approach was that if we had a multimodal distribution of particles, the pose determined would likely be somewhere in between the clusters making it guaranteed to be incorrect. Thankfully, this didn't occur often in our testing, but it is certainly an area of improvement for the future.

Visualizations

In order to help with debugging and demonstrating correctness, we also included a couple of visualizations. First, given the scan data from the LiDAR on the real car, we published a copy of the scan with respect to our predicted location for the car. This helped with debugging as we could then see where the real car saw obstacles throughout time, and compare it to where the projected car was seeing obstacles. If the scan lined up with the obstacles in the map, we knew that the projected car's location was approximately correct. Second, we also wanted to visualize the spread of our particles to ensure they were exploring an appropriate variety of poses. To do this, we published a PoseArray message with a pose for each of the 200 particles we were keeping track of. This helped us ensure that the amount of randomness added and frequency of resampling were appropriate for keeping track of the car. Finally, we published a few error graphs to visualize our car's performance as discussed in section 3.

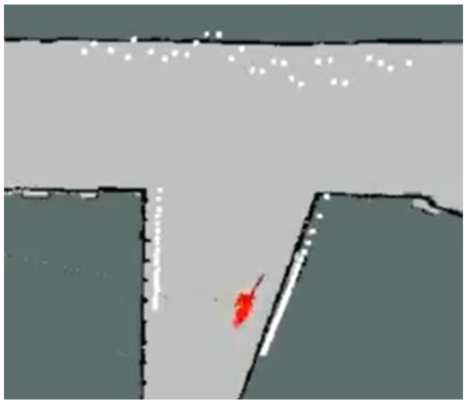
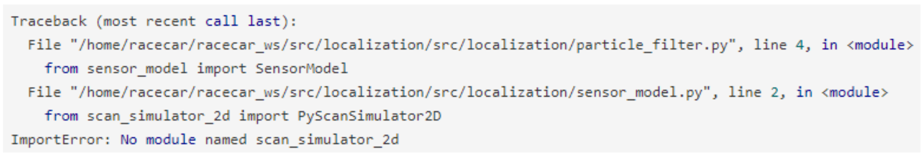


Figure 4: An image of the racecar's projected pose in RViz with visualizations. The white dots represent the laser scan seen by the actual car at the current point in time, and align almost perfectly with the walls seen by the projected car. The red cluster is a group of 200 particle arrows representing the possible current locations of the car

2.4 Running Physical Racecar - Andrew Manwaring

Setup Error Due to an error and limited time to troubleshoot the racecar



```
Traceback (most recent call last):
  File "/home/racecar/racecar_ws/src/localization/src/localization/particle_filter.py", line 4, in <module>
    from sensor_model import SensorModel
  File "/home/racecar/racecar_ws/src/localization/src/localization/sensor_model.py", line 2, in <module>
    from scan_simulator_2d import PyScanSimulator2D
ImportError: No module named scan_simulator_2d
```

Figure 5: Setup error message presented when running localize.launch, resulting in the team using rosbags to test our localization

workspace, the group was unable to run the localization code directly on the car in real time. After exhausting all troubleshooting notes posted, our team decided to record a rosbag as a workaround to fully test the capabilities of our localization. This was robust in predicting how the code would run on the actual racecar because the rosbag recorded the same /vesc/odom and /scan topics that the motion and sensor model would use in real time to localize the racecar. The rosbags proved faster than real life testing could've been for debugging the localization, a helpful lesson for the future.

3 Experimental Evaluation

3.1 Localization in Simulation - Andrew Manwaring

In the simulations, quantitative data was collected to demonstrate the capabilities of our localization implementation. In the given localization tests on Gradescope, the car was able to predict its location within an accuracy of 0.27 meters. The Gradescope tests provided the localization with odometry, scan, and presumably map data. The car drove around in a loop, where it was compared to the ground truth expected from the given data. The tests had an average error of 0.262 meters, 0.270 meters, and 0.266 meters with increasing odometry noise. The addition of noise progressively throughout the tests did not have any affect on the performance of the localization, demonstrating robustness in the face of significant odometry noise. The output trajectory graph vs ground truth and staff solution is shown in the figure 6.

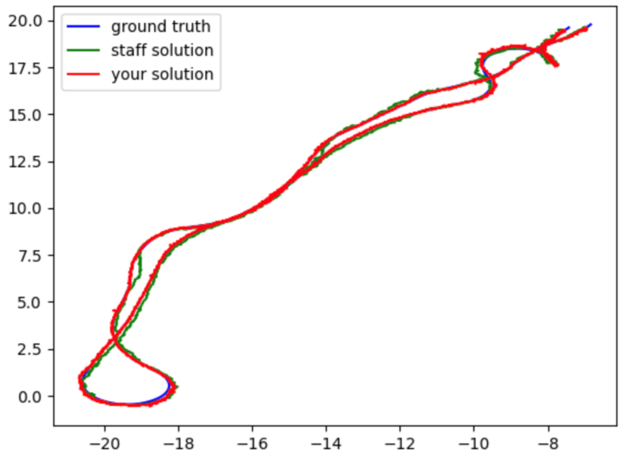


Figure 6: Simulation trajectory generated in the highest odometry noise Gradescope test using our localization controller. The red trajectory, our solution, nearly perfectly covers the ground truth, resulting in an average error of 0.266 meters.

In more of our tests on the simulation, the localization continued to perform well. In simulation, the car was driven around some portion of the simulated room using wall follower. The two tests consisted of inputing 0 and 0.1 as our odometry noise parameter and measuring the distance and angle error between the particle filter's predicted pose and the ground truth. Each graph represents the car's distance and angle error as it moves along some portion of the wall around the room. During this time, the car is constantly turning and correcting to maintain its path along the wall. Figure 7 and 8 represent the distance and angle error with 0 and 0.1 odometry noise, respectively.

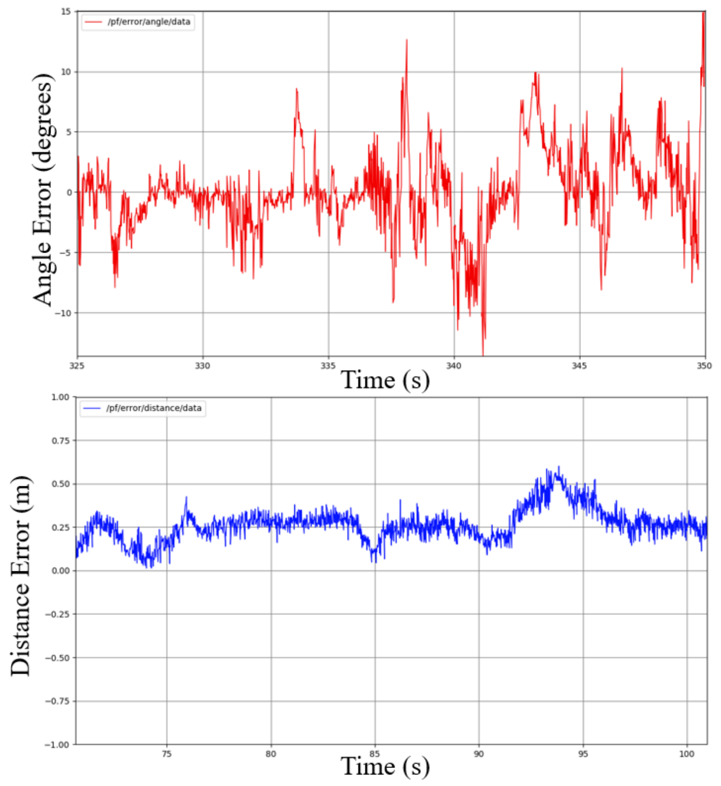


Figure 7: The angle and distance errors in simulation with no odometry noise, as the racecar ran wall follower.

Figures 7 and 8 demonstrate a similar level of error to the Gradescope tests, and show robustness in responding to localization drift. The distance errors with and without odometry noise hovers around 0.25 meters. We chose to show the angle error as a signed quantity to indicate the direction of the error as it indicates that the angle is oscillating around 0 degrees rather than drifting off to one side in particular. The angle error showed significantly more noise, but stayed within +/- 5 degrees the majority of the time, showing self-correcting behavior. In the odometry noise angle error graph, the angle error hits -20 degrees, but corrects itself back to nearly 0 degree error. This shows that the localization controller is able to respond to drift and correct itself before the localization and ground truth diverge. The odometry noise of 0.1 was considered to be larger than our smaller X, Y, and θ variance could easily handle, so the similar distance and angle error between the 0 and 0.1 odometry noise proved localization capability.

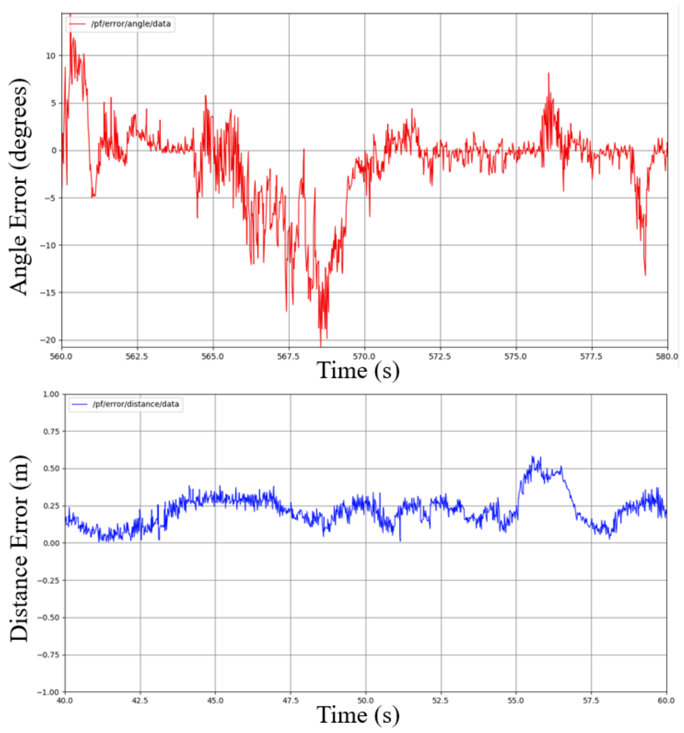


Figure 8: The angle and distance errors in simulation with very noisy odometry ($odometry\_variance = 0.1$), as the racecar ran wall follower.

3.2 Localization on Racecar - Andrew Manwaring

The localization on the physical racecar was evaluated qualitatively using a comparison of landmarks in the video of the racecar to the predicted location in rviz simulator. Using the unique shape of the Stata basement, such as the pillars and changing characteristics of the wall, the robot's location on the map in reality was easy to estimate on the Stata basement map. This was contrasted with the pose displayed in rviz for the localized car location. Additionally, the laser scan data from the car's run lined up like a trace when the car had localized it self well. With these methods it was determined that the car was localizing itself within the Stata basement accurately. This was not perfect, as was evident from drifts and offsets of the pose vs the actual location, however, the localization was able to correct itself when entering more characteristic areas, where each sampled location would have a very different scan. The link below shows a syncronized video of the real time racecar driving video with a simultaneous localization on a map, demonstrating our method for localization performance evaluation.

Localization on Racecar Video

4 Conclusion - Tiffany Horter

As a result of this lab, our racecar can now identify its location on a map when it knows its initial position. Our localization implementation was robust to noise and performed well in the simulator across increasing scales of noise with 94.5%, 93.8%, and 94.3% percent accuracy respectively compared to the staff solution. Since we ran into unresolvable hardware problems on the robot, we came up with an out of the box solution - recording a rosbag to enable us to complete the lab. Using this method, we were able to demonstrate that localization would work on the car. On the car, through a qualitative evaluation of relative landmarks in the simulated map and in real life, our localization was fairly accurate – even when it experienced drift, once it found a more characteristic and identifiable space it was able to correct itself. Therefore, both in simulation and on the car, the localization was highly effective.

Several improvements could still be made to improve the performance of this phase, such as running it directly on the car and continued tuning of parameters. By tuning parameters such as the constants relating to the probabilities determined in the sensor model, our implementation could more accurately determine its location by relying on a closer estimate of where it is likely to be. Some future work is to run this module on the car simultaneously as it is driving around to be able to implement the next design phase on the robot in real time. In the next design phase, we will use the robot's ability to ascertain its location to implement planning of higher-level goals beyond simply reacting to its environment.

5 Lessons Learned

5.1 Andrew Manwaring

During this lab I learned some helpful debugging tricks in ROS. The biggest one there was the use of rosbags. Using the rosbag data, we were able to "run" the robot over and over without waiting to charge the robot or do a lengthy setup. This allowed for the debugging process to be done much faster for this lab and ultimately allowing us to tune our localization to work well. I also continued to hone in on my numpy skills as we sought to improve efficiency in our localization. The localization methods used in this lab, although taught in lecture, were mostly unfamiliar to me. I learned thoroughly the algorithm that we used to localize the vehicle.
Additionally, I learned more about working in a team. Throughout the first few labs I have really enjoyed working with Tiffany and Nisarg and feel that we have been productive while finding enjoyment in tackling these challenges

together. As we move on to new teams, I will certainly use a lot of the technical and teamwork skills that my team members have taught me.

5.2 Nisarg Dharia

On the technical side, this lab was key at improving my skills with ROS while learning about localization techniques. Regarding ROS, I was able to master the use of rosparam to quickly tune parameters and gained experience with a variety of visualization tools such as PoseArray, Odometry, and LaserScan to help with debugging. In doing this, I learned more and more about how various amounts of noise and scaling can impact the performance of a localization model.

This lab also helped me improve as a teammate both with sharing the workload and communicating while integrating other people's work. Specifically, with only 3 people on our team we each had to do work independently from one another to get tasks done in time, which helped me be a little more independent. More importantly though, once we began combining our work I learned a lot about how crucial clear communication is during the integration process to avoid misunderstandings and delays.

5.3 Tiffany Horter

From a technical point of view, I developed an understanding of the localization algorithm and the basics of probabilistic robotics. This lab also allowed me to review my knowledge of probability and apply it to a real application. In addition, I built on my numpy manipulation skills to eliminate the for-loops to go faster. Once again, I learned that simulation is very different than reality and that more time must be allocated to debug those differences.

For this lab from the communication perspective, I focused on how to better present our error and consider what metrics would be most effective to indicate our results. Additionally, I learned that while working together remotely, it is important to communicate what you are working on so that multiple people aren't simultaneously writing the same code. I'll miss working together with Andrew and Nisarg – I think that we collaborated very well. The positivity of my teammates always made debugging much more bearable. I will take the lessons that we learned in terms of teamwork, especially our willingness to always jump in and help a teammate, to the next team I am working with.

6 CITATIONS

Localization README (Figure 1)