Lab Report: Localization

Team 21

May Huang
Mohammadou Gningue
Haley Sanchez
Nikhil Kakarla
Neil Chowdhury

Robotics: Science and Systems

April 14, 2023

1 Introduction

1.1 Lab Objective

Our team's objective for this lab was to implement a localization algorithm on our robot. Monte-Carlo Localization (MCL) enables our robot to locate itself within a known environment given sensor data and a motion model, providing information on the robot's position and orientation. A successful, robust, and reliable localization algorithm will allow our robot to locate and navigate through any environment autonomously. Given current pose data and an objective endpoint, the robot can navigate to that point by evaluating the relative error in its current pose and modifying its path of motion. We implemented our MCL algorithm by breaking down into three components: the motion model, the sensor model, and the particle filter.

1.2 Defining Success

In order to define success for this lab, we had a variety of metrics. First, we had unit tests for the motion and sensor model that would allow us to determine if they were working correctly. For the particle filter, we wanted to be able to watch in simulation as the particles followed the robot. Moreover, we expected the particles to follow the robot even when we introduced noise into the system. The main definition of success was that we did not want the particles to drift as the robot continued driving. Finally, we wanted our MCL algorithm to work on the robot. The main indicators of success was that the Lidar scans matched the map of Stata, which would show that the robot has an accurate idea of its location. Also, we want the standard deviation of our particle locations to be relatively low even in the presence of noise. This indicates that we have an accurate idea of the location of the robot and are not sampling from a wide range of possible locations.

2 Technical Approach

2.1 Developing a Motion Model

2.1.1 Algorithm

In order for our robot to find itself within a given environment, we use a motion model that relies on exteroceptive sensors. This allows for tuning to some amount of uncertainty, in the case that the robot was placed in an unexpected location, orientation, or moved in the middle of driving. Given a distribution of previous particle locations and recent odometry data, our model applies the shifts from the odometry data as well as a rotation matrix to the initial particle positions, then returns an updated pose.
In practice, the robot generates particles, uses the odometry data to assign probability values for each of the particles, evaluates where particles will fall after some displacement or movement of the racecar, and prunes data and the process repeats again.

2.1.2 Noise Function

Random noise is also injected into the system to cause the particles to spread as the robot moves. On every odometry update, we add random noise from a normal distribution from 0 to 0.01 to both position and orientation measurements. This method helps account for inconsistencies in sensor data, for example if the robot drifts or slips while in motion.

2.2 Designing a Sensor Model

2.2.1 Algorithm

The goal of this model is to be able to assign likelihood weights to particles given an observed laser scan. This is done by first pre-computing a probability table in order to reduce computation power needed to complete this task and increase the computational efficiency. Within the table we discretized probabilities across p_hit values and then calculated the likelihood weights according to the equation in Figure 1. We then discretized across columns in the pre-computed table (by d) to make sure that the probabilities across the columns add up to 1. The likelihood of a scan is computed as the product of the likelihoods of each of n range measurements in the scan.
In this algorithm particles are given in the form of [x, y theta] and need to be put through ray casting in order to get the ground truth lidar scans. We then

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

Figure 1: Equation for Likelihood of a Particle Given an Observation

compare these particle ground truth lidar scans with the lidar scan of the measured scan (observation). Once, we get the probability from the pre-computed table, we can then squash the probability by raising them to the power of $\frac{1}{2.2}$. Lastly, we convert values from the lidar scans from meters to pixels by dividing them by `self.map_resolution*lidar_scale_to_map_scale` parameters.

2.2.2 Down-sampling

2.3 Implementing Particle Filter in Simulation

2.3.1 Algorithm

2.3.2 Design Choices

2.3.3 Tuning Noise

When noise was added to the motion model there was no tuning or examples that highly relied on noise. So when the motion model and sensor model came together to implement the particle filter, we needed to re-evaluate the noise scaling factor. We need noise so that the re-sampling process goes smoother and so that the values do not collapse on one another. In figure 2, we experimented with various noise scaling factors. The .001 light grey line shows that with too little noise our absolute error between the ground truth and the noisy odometry is very high. The .1 black line shows that with too much noise the model struggles to keep absolute error low. The .02 scaling value with the blue line shows that this level of noise is able to keep absolute error low as time increases. Based on this data, we used a noise scaling factor or .02 in our particle filter.

2.3.4 Increasing Algorithmic Speed

We navigated a tradeoff between using more particles, which would lend itself to greater locational accuracy, and algorithmic speed. We increased algorithmic speed by downsampling lidar scan data, to where we were working with 100 data points instead of over 1000. We used interpolation to create a smooth curve of the lidar data and then sampled that curve. This process allowed for faster computation on less data points as well as a major reduction in lidar Scan noise. Additionally, our MCL algorithm avoided recomputation by using caches when processing commonly used data, such as scan ranges. Using a precomputed probability table instead of computing probability following every raycast scan greatly increased speed. Without this precomputation, the MCL algorithm would not be able to track the robot when moving at high speeds.

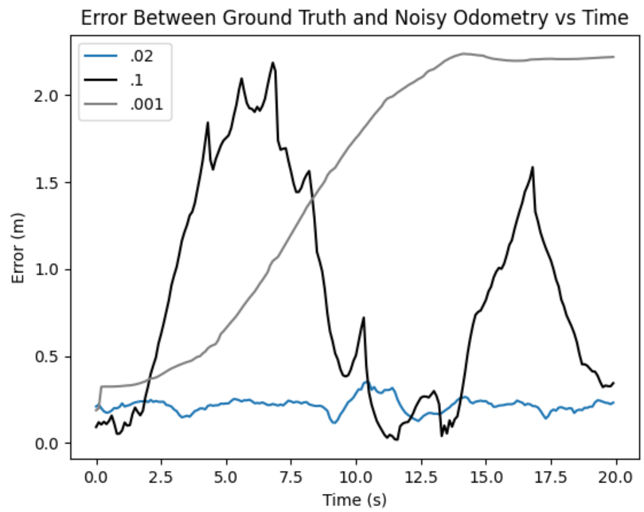

Figure 2: Impact of Various Noise Scaling on Absolute Error

Finally, computing using NumPy operations over for loops also helped speed up operations as NumPy operations are very efficient.

2.3.5 Autograder Results

2.4 Localization on Physical Robot

After getting the localization to work in simulation, it was time to transfer the code onto the robot itself and test it in the real world. The transfer process was relatively easy, only requiring us to change a few topic names and update some of the code. The main challenge was that we needed to tweak our locking mechanisms to work on the physical robot, where the hardware runs at different speeds than in simulation. After getting the locking mechanism to work, we were able to eliminate the concurrency issues and the model worked on the robot. We visualized our results in RViz and used rqt plot to gather data about our particle distribution.

2.4.1 Overall Results

Our localization worked very well on the robot. When running it live on the robot, we visualized the data in Rviz. An example output of our data is shown in the below figure. In the output, the base layer is a map of the Stata basement. This is overlayed with the live feed of the lidar scan data. Moreover, we added visualizations for our particles as well as the base link guess of where our robot was. As shown below in figure 3, the model is working because the lidar scans very closely match the walls of the Stata basement in the map. This means that the robot has an accurate idea of where it is on the Stata map and the external readings match the internal location estimate. Driving at different speeds, we were able to show that the lidar scans continually matched the walls around the robot. Therefore, the localization algorithm was functional and effective as the robot moved around its environment.

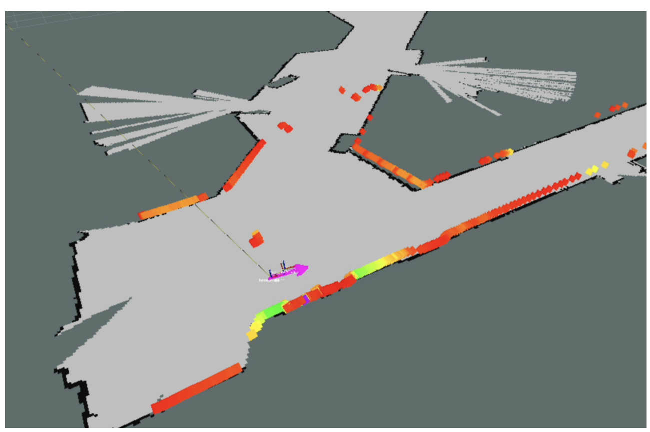

Figure 3: Functional Output Display in RViz of Localization

2.4.2 Testing

After visually verifying that our algorithm was working, the next step was to run tests to ensure that the localization algorithm was robust and effective under a number of different conditions. First, we tested initializing the robot at different locations. Next, we tested initializing the robot at a different location in the physical environment than where it thought it was in simulation. The results of this experiment are shown below. Our localization algorithm was able to quickly determine that the robot was in the wrong location and adjust the estimation until it was accurate. Below are images of the image outputs at the start of initialization and a few seconds later. In the first image, the lidar scans clearly do not line up with the walls of the Stata map. However, within a few seconds the robot updated its estimated pose and the lidar scans again very closely mirrored the map. This process is shown in the second and third images. This shows that the robot was able to localize and "find" itself in its environment even when the initial guess was incorrect. Additionally, we included a graph in figure 5 of the standard deviation of our particles during this trial. In the graph, the particles first start very scattered with high standard deviation as the robot tries to find itself on the map. Quickly, the robot is able to sample the most correct particles and the standard deviation drops to the baseline level created by our artificial noise. This is proof of our algorithm working, as it is able to sample the correct particles and create a very accurate estimate quickly.

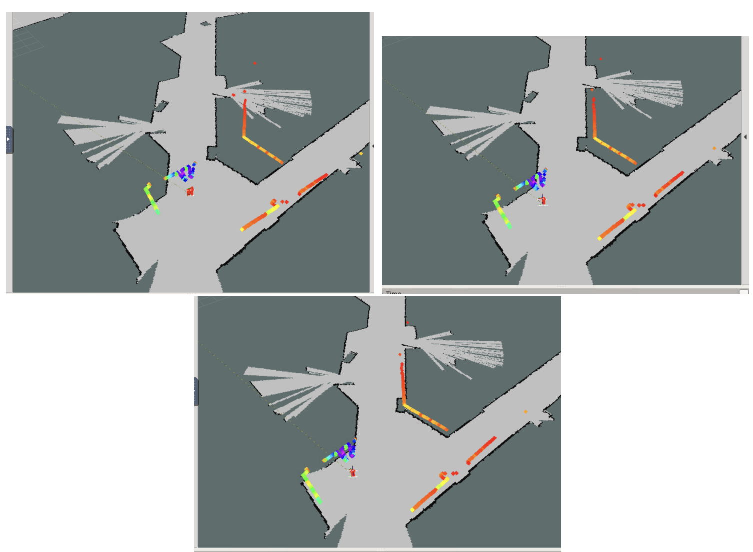

Figure 4: Robot initialized incorrectly and able to localize

Additionally, we wanted to test our localization algorithm with the robot driving at different speeds. Although visually it appeared that the algorithm was working, we conducted another experiment driving the robot first quickly and

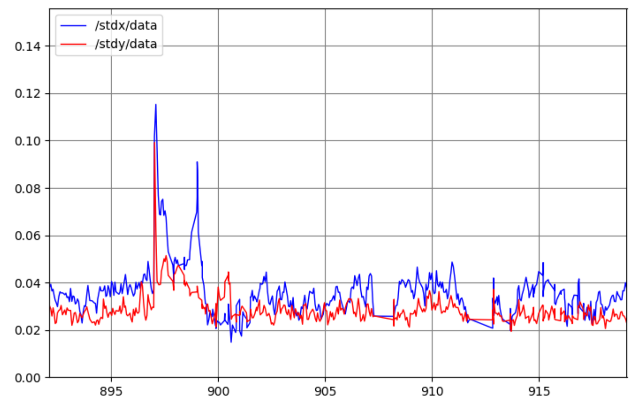

Figure 5: Standard Deviation of x,y coordinates of particles with bad initialization

then slowly. The graph of the standard deviation of the particles is shown below in figure 6. The key result is that, after the initial initialization, the standard deviation did not increase too much and was able to return to its base levels quickly. This is proof that our localization algorithm is robust to driving at different speeds and is computationally quick enough to follow the robot even at its max speed.

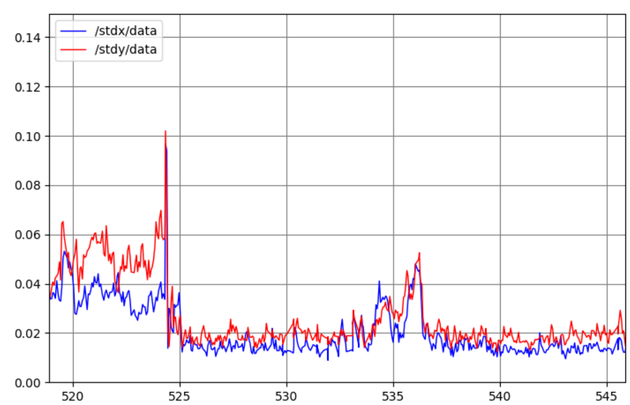

Figure 6: Standard Deviation of x,y coordinates of particles while driving

In conclusion, transferring the localization algorithm onto the robot allowed us to verify the accuracy of our algorithm. By testing different initialization conditions and driving speeds, we found that our localization algorithm is correct, robust, and efficient.

3 Conclusion

We achieved our goal of implementing a successful and effective algorithm for MCL localization with our particle filter. We delegated the motion model and sensor model algorithms within our team, which allowed our team to work on them in parallel and to be more efficient. We were able to use unit tests to verify that the motion and sensor models were functioning correctly. Once we put these algorithms to use within the particle filter, we were able to verify its results through the auto grader. In simulation and through unit tests, we then knew that our algorithm was functioning correctly. After that we moved to adapting our algorithm to function on the physical robot. We had to make slight alterations to our original code. Then we completed multiple experiments with the physical robot to verify that the localization was working correctly in Stata basement. From our experiments, we found that our particle filter algorithm functions effectively in the real world.

Next steps include completing further physical experiments on the robot. Another experiment that we could complete is running wall follower and then comparing the actual distance from the wall with the measured distance from the wall with localization. Additionally, we could experiment with various locations if we are able to attain a map of the area like we had for Stata basement.

Overall, we completed implementing our functioning and effective particle filter algorithm that lets us use localization on the robot. So far, we have worked with lidar scans, ZED camera, and implemented localization. We look forward to learning about path finding and implementing it with the physical robot.

4 Lessons Learned

4.1 Haley

Since this lab was started prior to spring break, it was very crucial for us to take notes so that we could pick up right where we left off. Prior to break we had a good conceptual understanding of the lab and how the components mapped to the individual assignments from beforehand. Having good documentation for technical work is super imperative if there is a break or new people joining in on the project. By keeping documentation the project is able to get done smoothly.

4.2 May

This lab was challenging because it required clear and consistent communication between team members while working in parallel with eachother. Being able to do the individual pre-assignment was very key for understanding how to approach these algorithms from a conceptual lens, and debugging was a huge part of successfully implementing and testing our code. I think planning ahead for securing robot time and using it efficiently, as well as communicating successfully with other teams on access to equipment, will be something that continues to be extremely valuable to practice and crucial to finishing work by deadlines.

4.3 Mo

4.4 Neil

4.5 Nikhil

I learned a lot about this lab. The primary lesson for me was dealing with concurrency issues throughout the code. Because of the structure of ROS, callbacks are not thread safe. Therefore, I had to apply locking to each of the nodes to make sure there were no concurrent transactions that would cause the program to crash. Additionally, I learned a lot about communication and parallel development. For example, our team began to comment code and document our code better. This allowed us to both pick up where we left off more easily as well as code individually. In this way, we were able to streamline our development process and spend less time trying to figure out what other people's code was doing. Therefore, I learned a lot about both technical programming and team communication throughout the course of this lab.

5 Credits Page

5.1 Nikhil

- Localization on Physical Robot: ALL

5.2 Neil

- Motion Model: Noise Function
- Sensor Model: Down-sampling
- Autograder Results: X

5.3 Mo

- Implementing Particle Filter in Simulation: Algorithm
- Implementing Particle Filter in Simulation: Design Choices
- Autograder Results: X

5.4 May

- Introduction
- Motion Model: Algorithm
- Implementing Particle Filter in Simulation: Increasing Algorithmic Speed

5.5 Haley

- Sensor Model: Algorithm
- Implementing Particle Filter in Simulation: Tuning Noise
- Conclusion