Lab #5: Localization: Using Motion and Sensor Modelling to Detect a Racecar's Location

Team #5

Cole Paulin
Erina Yamaguchi
Preston Tower
Waly Ndiaye

6.4200/16.405: RSS

April 14, 2023

1 Introduction

1.1 Overview and Motivations

During this lab, we designed a particle filter using Monte Carlo localization to be embedded into our robot. The purpose of designing our particle filter was to enable our car to be able to estimate its position on a known map environment, such as the Stata basement. This lab builds on the preceding labs during which we designed a wall follower, safety controller, line follower, and parking controller. It is a crucial next step towards making the racecar more autonomous because it allows for the robot to plan with a higher level of perspective as we work to implement path planning in the coming weeks. This lab will be helpful for reaching goals that are outside the car's immediate vicinity, such as driving to an end location. Its functionality can also be paired with previous labs, such as line following, for running our racecar on the Johnson Track for the Final Challenge. In order to evaluate the success of these systems, we set our goals to minimize the effects of increasing odometry noise on our particle filter's time-average deviation from the trajectory, and minimize position and angle error (both in simulation and in the real world). The deliverables for Lab 5 were split up into four sections: designing the motion model, designing the sensor model, implementing the particle filter, and testing/deploying code onto the car.

1.2 Goals and Implementation

Our group decided to start our focus on developing our Motion and Sensor models, which later combined to be our particle filter. As an overview, our particle filter worked as follows; after being initialized with a starting position, we updated our particle probabilities by passing data into the Motion or Sensor Models to continually estimate our current position on the map.

In addition, we had many ROS nodes we dealt with and had to figure out how to configure them to work with our implementation. In general, the robot subscribed to the Odometry and LaserScan data for the Motion and Sensor models respectively. Our current odometry pose, errors, and all of the particle poses were published to enable our localization functionality and for data collection.

During both simulation and real-world testing, we collected data to analyze our localization performance. With this information, we will be able to make educated decisions on for our upcoming path planning lab, along with choosing between our localization algorithm and the course staff's version for future usage on our racecar.

2 Technical Approach

2.1 Motion Model

The function of the motion model is to take the possible state estimates of the robot's location, provided by the particle filter, which will be propagated to the next time step using the given odometry data, representing the control input to change the position of the racecar.

2.1.1 Calculating Future States

The current pose data of the particles are provided in the world frame in which the car is located. The odometry data, however, are provided in the racecar frame. Therefore, transformation matrices must be constructed in order to calculate the current pose in the racecar's frame and place it back in the world frame. The transformation matrix is defined as:

$$T_r^W = \begin{bmatrix} cos(\theta) & -sin(\theta) & x \\ sin(\theta) & cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix}$$

Using this transformation matrix, the next state X' can be computed by multiplying X, the current state with the odometry matrix.

$$\begin{bmatrix} cos(\theta') & -sin(\theta') & x' \\ sin(\theta') & cos(\theta') & y' \\ 0 & 0 & 1 \end{bmatrix} = T_r^w * \begin{bmatrix} cos(\Delta\theta) & -sin(\Delta\theta) & \Delta x \\ sin(\Delta\theta) & cos(\Delta\theta) & \Delta y \\ 0 & 0 & 1 \end{bmatrix}$$

We can simplify the computation without applying matrix multiplication for the new cartesian coordinates because we only need information about the last column.

$$x' = cos(\theta) * \Delta x - sin(\theta) * \Delta y + x$$
$$y' = sin(\theta) * \Delta x + cos(\theta) * \Delta y + y$$

Additionally, the new θ value can be calculated by just adding the change in θ from the odometry to the current θ value.

$$\theta' = \theta + \Delta\theta$$

2.1.2 Implementing Noise

The noise is assumed to be a normal Gaussian distribution. The noise is added to the current pose data to account for any errors in the odometry data and inaccuracies in motion due to factors like slip. Each position and angle is represented with a mean of 0, and the distribution was found experimentally.

2.2 Sensor Model

The goal of the sensor model is to determine a probabilistic model of the robot's actual position. This is important because in Monte-Carlo Localization, the robot learns from the different possible locations (particles), and then hones in on the actual location. The robot uses its LiDar Scan data, and compares it across each of the particles, and returns the probability that the robot is where that particle lies. The sensor model aids in the process of filtering the particles, until the robot has an accurate real-world position.

2.2.1 Calculating Probability

Each particle is represented with a probability that it corresponds to the robot's location on a static, known map. On the known map, we can extrapolate certain known features, such as hallways, corners, and anything that a LiDar scan could sense. The main components involved in the probability model are the actual LiDar scan that the robot has, and the difference between that scan and the particles. The distance to known obstacles on the map is represented by d. And the difference between each particle and the LiDar scan is represented by z. There are 4 different equations that we follow to give each particle a probability.

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

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

$$p_{max}(z_k^{(i)}|x_k,m) = \begin{cases} \frac{1}{\epsilon} & \text{if} \quad z_{max} - \epsilon \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$p_{rand}(z_k^{(i)}|x_k,m) = \begin{cases} \frac{1}{z_{max}} & \text{if} \quad 0 \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The first equation is based on how aligned the scan is with the particle. The second takes scan data that are too large into account. This is because some LiDar scans never reflect back, so they may read as erroneously large distances. The model also takes into account its counterpart: short scans. These could be a product of scans that were reflected immediately by the physical LiDar scan. This will show some data that is extremely close to the robot, even though it may not be. The last aspect is some random probability distribution that we add to ensure that our model does not overfit. These different parts are then weighted (by importance), and then averaged to return our final probability. This average is based on the equation

$$p = \alpha_{hit} \cdot p_{hit} + \alpha_{short} \cdot p_{short} + \alpha_{max} \cdot p_{max} + \alpha_{rand} \cdot p_{rand} \tag{5}$$

In the Sensor Model, we used α values of $\alpha_{hit} = 0.74$, $\alpha_{short} = 0.07$, $\alpha_{max} = 0.07$, and $\alpha_{rand} = 0.12$. The α_{hit} value is the largest because this is the most important feature of determining the validity of the particle position. The LiDar Scan being aligned with our map is the most important aspect of a correct position.

2.2.2 Increasing Efficiency

In the physical robot system, these calculations must be done extremely efficiently. As shown in 2.2.1), there are many different calculations that must be made for every particle. This will take too much computational time, so a more efficient method is necessary. One way to combat this is to precompute an array of probabilities that we can then look up in the future. The only changing variables are d and z, so if we make z our rows, and d out columns, then we can use the equations to precompute a probability array based on the discretized z and d values. Now, the robot will find a z and d value for each particle, then it only has to compare those two values against our precomputed matrix to find the probability. This worked great, as the robot was able to run with a faster computation that did not impede its overall ability.

2.2.3 Evaluate Function

The Evaluate function was implemented to return the probability of accuracy at each particle. This was done by taking in all of our particles, and comparing them to our observation (LiDar Scan data). Often, the raw data did not correspond well to integer z and d values, so the model must perform some operations to clean up the calculations. To do this, we followed a series of steps…

1. Scale the data down to account for our meters-to-pixels scale.

2. Clip any data that does not fall within our maximum values of z and d in the array, and also any erroneous negative values
3. Convert these new arrays into integers

As a final step, we returned a list of probabilities based on the different z and d values. The raw probability was often unnecessarily too large, so the model scales down each of these probabilities so that they are more easily used by the particle filter.

2.3 Particle Model

The goal of this lab is to implement localization using a particle filter. Particles are the estimated poses of the racecar, with a weight, or likelihood that it is the true odometry value. The previous two models are used to update particles through odometry data and resample the particle filter using laser scan data. The particles are initialized as a random normal distribution from the initial pose position. However, they all are equally likely to be the true pose. From there, they will converge to the estimated actual position.

2.3.1 ROS Implementation

The implementation of our Particle Filter Included involved utilizing three subscribers and three publishers. The first of these subscribers involved listening to Odometry Data and aided in updating particle pose (odom_topic). Velocity and change in time were used to calculate the estimated odometry message, which was to be passed into the motion model. The second was subscribed to LaserScan data coming in from our Lidar scanner (scan_topic). Downsampled observations were passed into our Sensor Model to update particles by reevaluating their probabilities. Lastly, our third subscriber was for setting the initial pose before the 1st occurrence of the Sensor/Motion model being called (/initialpose). After initialization, this callback published our racecar's starting position on RVIZ through the use of one of our publishers.

As for our publishers, the one that helped publish our racecar's current estimated position was our final Odometry publisher, /pf/pose/odom. It published our pose after we used a weighted average on all the particles and their likelihoods/probabillities. For data collection and visualization purposes, we had publishers that reported angle (/ang_error) and position error (/position_err) regarding the car's estimated position compared to its actual position. A publisher that reported the odometry of all particle poses to visualize their behavior in RVIZ was also implemented (/pf/pose/all_poses).

2.3.2 Implementing the Motion Model

The true odometry data is unknown, but we can calculate the approximate change in robot position through the 'Twist' data, which provides linear and angular velocity. Recording the change in time from the previous call of the odometry callback function, we have an estimate of how the racecar and the particles should move. As mentioned in the technical approach, each of the particle data has random normally distributed Gaussian noise to take into account the actual noise in sensor data and errors in motion with the actuator. Without the sensor model, we can observe the particle filter diverging with time as more noise is added to its prediction.

2.3.3 Implementing the Sensor Model

The sensor model takes the LiDAR data in order to evaluate a prediction of its location in the world frame. The original data has approximately one thousand points, but we downsample to about ten percent of the data, or one hundred points. As mentioned in the technical approach section, the sensor model ultimately outputs a probability distribution for each of the particles based on the lidar data. This probability data places more weight on particles that would best match the laser scan data, which we can use to resample the particles. To do our downsampling, we used the np.random.choice() operation, and correlated each LaserScan point with the probability regarding its likelihood.

2.3.4 Pose Output

In order to initialize the position, the 2D point estimate in RVIZ was used. Before running the odometry and motion model, the robot's state must be estimated. Using this estimate, the odometry and LiDAR data update the position accordingly to match the collected data with its knowledge of the surroundings. Therefore, the two motion models both update the particles. To make sure these updates do not interfere with each other, thread locking is implemented. After each particle update, the estimated pose position is evaluated through a weighted average because we know that some of the particles carry more weight than others. Utilizing a weighted average minimized the risk of having a few outlier particles skew the car's estimate of its current location. To visualize the particles, as well as their spread and convergence, /pf/pose/all_poses and /pf/pose/odom are published for visual analysis.

3 Experimental Evaluation

3.1 Particle Model: Simulator Implementation

The particle filter was run using three roslaunch files. The first was simulate.launch to simulate a model of the racecar and fake scan data in the environment. Additionally, the parking_sim.launch from the visual servoing package was used to drive the racecar, or to keep it in place. We chose to use the visual servoing package over our wall follower since the parking detection feature allowed us to try a wider array of test conditions that involved localization and our car. Lastly, the localize.launch in the localization was run to implement the particle filter. After simulation testing, localize_real_env.launch was used instead, in order to run the racecar in a real life scenario instead.

3.1.1 Simulation Performance

Our first method of testing our particle filter's performance was through utilizing the course's Gradescope Autograder functionality. The Autograder tested our particle filter's performance in conditions with no odometry noise, some noise, and more noise. For the case with no noise, we had a time-average deviation from the trajectory of .275m, as shown below.

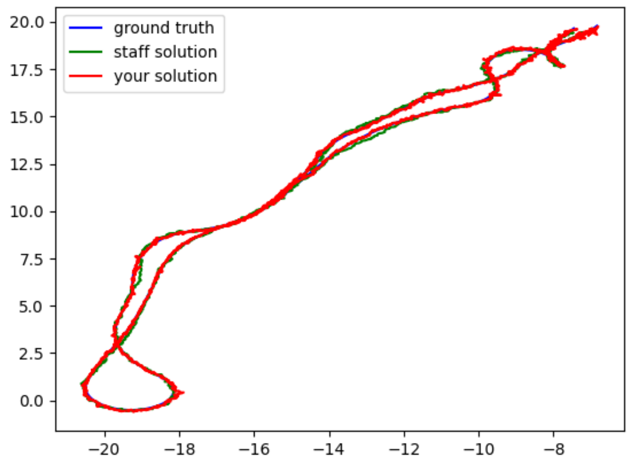

Figure 1: Time Average Deviation from trajectory when in a condition with no odometry noise

In the case with some noise, we deviated 0.267m, as shown below.

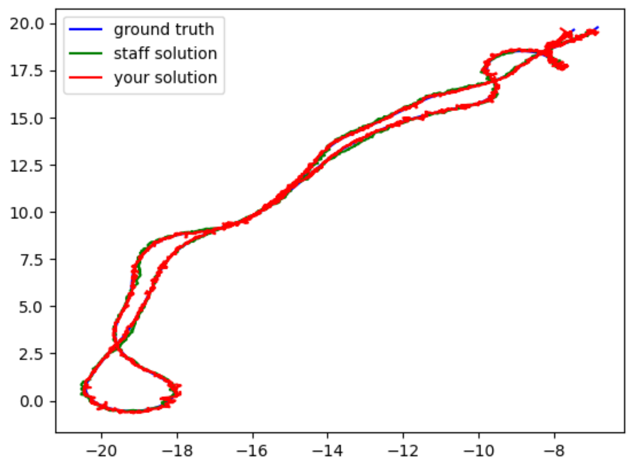

Figure 2: Time Average Deviation from trajectory when in a condition with small amounts of odometry noise

In the last case with the most noise, we deviated .268m, as shown below.

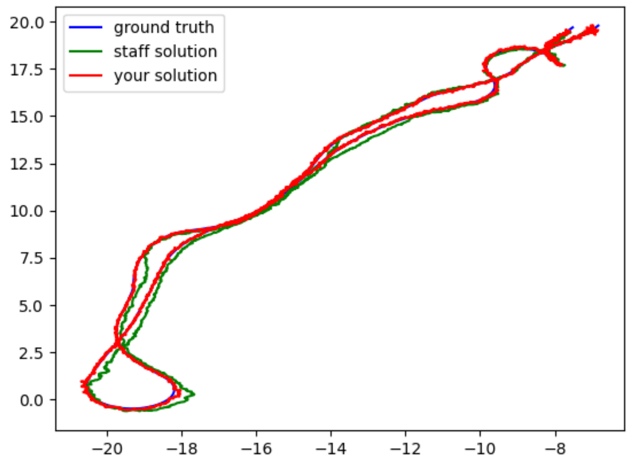

Figure 3: Time Average Deviation from trajectory when in a condition with larger amounts of odometry noise

From the simulation testing, we were able to see our car's performance was consistent with different levels of noise, and that our performance did not decrease as the noise was increased. Based on our score, we found that we were at least 94% accurate for all the different variations of odometry noise, with an average .27m time-average deviation. Knowing that our car's simulation performance was at a currently optimal level, we decided to shift our focus to deploying the localization on the car in real-life.

3.1.2 Error Analysis

Unlike in hardware, the true odometry pose of the racecar is known in simulation. Therefore, we can evaluate the performance of the particle filter by evaluating the error in position and angle measurement. For position, the Euclidian distance between the actual racecar and the estimated racecar position was published. Additionally, the angular error was evaluated as well, as the difference between the true and estimated angle. Figure 4 shows the angle error varied by approximately 0.03 radians. However, the mean of its estimates remained close to the true value. In comparison, Figure 5 shows that the error varied between 0.2 to 0.35 meters, with the mean closer to 0.25, rather than 0. Numerical analysis explained further in the next section, with testing in hardware reveals that this offset could have come from the distance between the estimate of the base link pose and the LiDAR position.

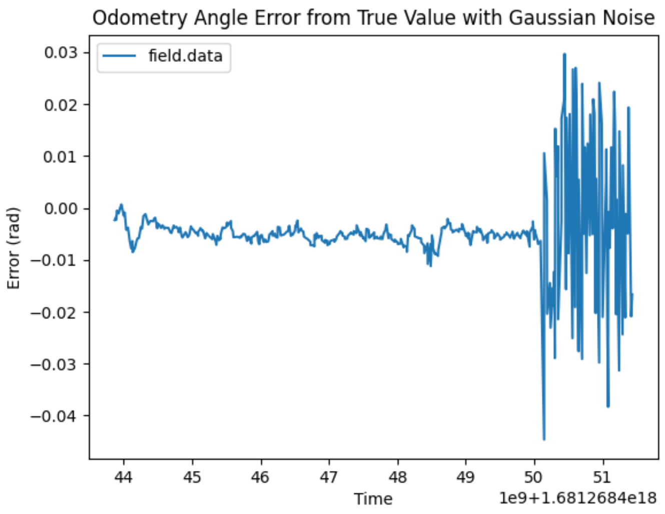

Figure 4: Angle error of the racecar's position in simulation centered around 0 shows the particle filter evaluates close to the true pose angle.

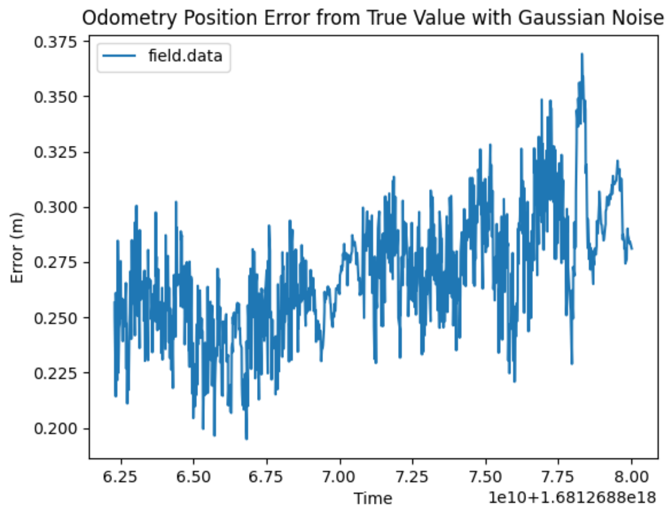

Figure 5: Position error of the racecar's position in simulation shows that an offset exists, but has consistent performance, with variations of less than 0.1 meters from the mean.

3.2 Localization: Racecar Implementation

3.2.1 Initial Pose Estimation

The car requires an initial estimated pose which is near to the car's true pose in order for the real-time pose estimation to begin. We provided this initial pose estimate to the car using RViz to retrieve the pose coordinates in the map frame, which we then fed to the particle filter. The car could reliably converge on its true pose if our initial estimate pose was near to the true pose. We provided increasingly inaccurate initial poses and found that the particle filter would usually converge on the true pose, but would, in certain areas of the hallway, converge on a different pose which appeared to represent a local maximum of estimation likelihood. It was particularly sensitive to variations in the rotation of the estimated pose, such as if the estimation was rotated by 90 degrees or more from the car's true orientation.

3.2.2 Measuring Accuracy

Once our software implementation ran without error on the racecar hardware, we performed an experiment to assess its accuracy. We placed the race car in the middle of a straight segment of hallway which we could easily identify on the RViz map visualization. We then moved the car by incrementally increasing its distance from the wall. We measured the true distance with a tape measure and measured the localization routine's estimated distance using the RViz measurement tool. Both distances were measured from the wall to the car's origin: the center of the rear axle. We performed that process once with the car facing directly away from the wall (i.e. aligned with the wall's normal direction), and again with the car facing parallel to the wall (90 deg right of the wall's normal direction).

The results of those measurements are shown in figure 6 below. Initially, we observed that the car's position estimate was close to correct when the car was facing parallel to the wall, but displaced by a large constant offset when facing away from the wall. We postulated at this point that the position estimate routine was actually reporting the estimated position of the car's LIDAR sensor, but the position was being displayed in RViz as that of the car's origin. To account for this, we subtracted the distance between the car's origin and the LIDAR sensor (0.29 m) from the position data we recorded, and the position estimate error was greatly reduced (this correction is shown by the green line in fig. 6). We believe this frame transformation also accounts for the constant offset observed in figure 5 of approximately 0.29 m. With this correction applied, we found that the car was able to reliably estimate its distance from a wall to within 0.2 m across a range of true distances.

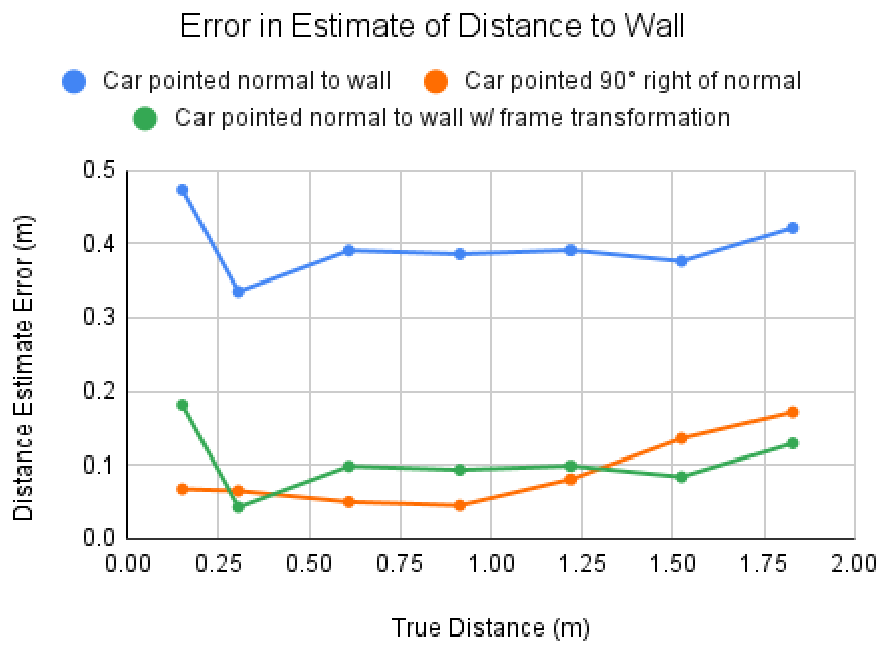

Figure 6: Racecar is able to estimate its distance from a wall within 0.2 meters of the true distance

3.2.3 Driving within Known Hallways

We drove the car along a segment of hallway which was within range of its WiFi router and observed that its position estimate closely matched the car's true position in the hallway. Because the debugging process was so time-consuming, we did not collect data to fully characterize this behavior, but we collected videos for future reference.

4 Conclusion/Lessons Learned

4.1 Conclusion

Our implementation of a particle filter which estimates the car's pose in a known environment was successful. In real-world operation, the particle filter was able to converge on a pose estimate which was accurate to within 0.2 m, which is about half of the length of the car. The particle filter was able to maintain an accurate pose estimation while the car was driven along hallways in the presence of noise due to objects and passerby. This demonstrated capability fulfills the lab's purpose of preparing the car for more autonomous action. Our successful particle filter will have a crucial role in our future projects which involve path planning and execution, and autonomous racing and city driving.

4.2 Lessons Learned

4.2.1 Erina

In all of the labs, the move from simulation to hardware is never smooth. However, this lab was especially true where we had issues in building the packages and launching them. An especially difficult part was solving the clock skew error in the system, which showed that the particle filter was running as expected, but could not publish due to the clocking error. I think some of these issues could have been better approached and have caused less stress if more office hours were attended earlier in the hardware integration.

4.2.2 Cole

This lab had many different moving parts, between the motion model, sensor model, and particle filter. The most efficient way for our team to work is to divide the work, then combine and implement each separate part into one final product. The topics are getting more difficult in the class overall, and it can be difficult to know how every part of the robot works at a more in-depth level. When the team got to the testing portion of our efforts, there was a difficult time debugging because everybody had gained expertise in different areas, and we never communicated how our respective parts worked. For example, one might know how the sensor model works, but then it could be difficult to combine this with the particle filter because of a lack of knowledge. In the future, our team will make a more intentional effort to dive into everybody's work so that when we have to debug, everybody can contribute efficiently.

4.2.3 Waly

During the process of attempting to deploy localization on our racecar, our team had many hardware and software difficulties along the way. These included, but were not limited to, not having the sim_ws folder on our racecar, clock skew issues, and our map not finding our tf data. While working through these problems, we often received help from other teams who had gone through similar issues and fixed them. This was incredibly crucial in not only helping us solve some of our issues, but also aided in allowing us to figure out what was even the problem in the first place. Our experiences this lab showed to me the benefits of discussing the lab with other teams, which is something we have not done too much in the past. In the future, we hope to maintain good relations with other teams, so that we can help groups when familiar issues arise and also get assistance when possible.

4.2.4 Preston

This lab was difficult. Our team worked hard on the individual system components, which were complex, then spent a long time struggling to integrate them into hardware, which eventually worked for reasons still mostly unknown to us. I was unable to participate in most of the work done for this lab, but contributed to the collection and analysis of data from the functioning hardware. Personally, this lab reminded me to over-communicate about my conflicts rather than assuming/hoping they would resolve soon. If I had done so, my team would have been at least prepared for my absence and adjusted earlier, which would have relieved some of their strain.

5 Addendum

In order to not clutter the report, we listed out by team member which sections they primarily authored. The listing is recorded below.

5.1 Waly

- 1.1 Overview and Motivation
- 1.2 Goals and Implementation
- 2.3.1 ROS Implementation
- 3.1.1 Simulation Performance
- 4.2.3 Waly (Lessons Learned)

5.2 Cole

- 2.2 Sensor Model
- 2.2.1 Calculating Probability
- 2.2.2 Increasing Efficiency
- 2.2.3 Evaluate Function
- 4.2.2 Cole (Lessons Learned)

5.3 Erina

- 2.1 Motion Model
- 2.1.1 Calculating Future States
- 2.1.2 Implementing Noise
- 2.3 Sensor Model

- 2.3.1 Implementing the Motion Model
- 2.3.2 Implementing the Sensor Model
- 2.3.3 Pose Output
- 3.1 Simulator Implementation
- 3.1.1 Error Analysis
- 4.2.1 Erina (Lessons Learned)

5.4 Preston

- 3.2 Localization: Racecar Implementation
- 3.2.1 Initial Pose Estimation
- 3.2.2 Measuring Accuracy
- 3.2.3 Driving within Known Hallways
- 4.1 Conclusion
- 4.2.4 Preston (Lessons Learned)