Lab 5: Determining Robot Pose in a Known Environment Using Monte Carlo Localization

Team #11

Timber Carey
Marcos Espitia-Alvarez
Subin Kim
Amy Shi
Christian Teshome

6.420 Robotics: Science & Systems

April 10, 2024

1 Introduction

Subin Kim

An autonomous system's ability to determine its location within an environment is crucial for tasks ranging from navigation to interaction with the surroundings. The process of localizing the robot racecar system inherently involves managing uncertainty and noise associated with motion and sensor model data in real-world scenarios, and updating the particles based on the data received from the odometry. A localization algorithm is crucial in the racecar's ability to perform and compete effectively, as it aids in understanding the racecar's current pose, path planning for the next lab, as well as in maneuverability and speed optimization for the competition.

The Monte Carlo Localization (MCL) method in this lab employs a particle filter approach to tackle uncertainty with the racecar localizing through a probabilistic approach. This is a non-parametric method for implementing the Bayesian filter through random sampling, which represents a probability density function by a set of random particles. Each particle (hypothesis) carries an associated weight that reflects the posterior probability of the state given all prior information. The greater the number of particles, the more accurate the representation of the probability density function. MCL is used for its ro-

bustness in recovering from localization failures with the resampling strategy, focusing on areas with high likelihood.

1.1 Lab Objectives

The primary goal of this lab is to implement and evaluate a particle filter to localize a car within a static map. This process involves:

- Developing an understanding of motion and sensor models that form the basis of the MCL algorithm,
- Constructing a simulation-based particle filter algorithm and validating it against ground truth data in a controlled 2D racecar simulation environment, and
- Adapting the algorithm for use in the physical racecar.

1.2 Overview of Technical Approach

Our approach for solving the localization challenge consists of the following:

- Motion Model: Updating the state of particles based on the robot's movements, incorporating the stochastic nature of physical actuation.
- Sensor Model: Interpreting LiDAR and other sensory data to assess the likelihood of a robot's pose.
- Resampling: Selectively refining the set of particles to concentrate on more probable hypotheses, increasing localization accuracy each step.
- Algorithm Evaluation: Testing the implementation against various noise models and real-world disturbances.

2 Technical Approach

2.1 Motion Model

Timber Carey

The purpose of the motion model is to update the pose of a set of particles using wheel odometry for use in the particle filter. The poses are expressed in the world frame and the odometry is a local quantity, Δx, which represents the measured change of pose between time $k-1$ and time k. If we were to use a deterministic motion model (with no added noise) in the particle filter, all of the particles would end up in the same place, making the particles redundant. Therefore, we must inject noise into the motion model, which will make the particles spread out as the car moves. This accounts for any uncertainty that exists in the odometry information.

The first consideration when designing the motion model was how to add noise. We wanted the noise to be centered around zero so that the particles would have an equal probability of moving in either direction. We also wanted the noise to have a greater likelihood of being closer to zero and lower likelihood for greater noise values. This criteria lends itself to a Gaussian distribution with a mean of zero.

The next consideration after choosing a distribution type was how to choose the parameters of that distribution, namely the standard deviation of the dx, dy, and $d\theta$ noise values. With a standard deviation too small, the points would not spread out sufficiently for the particle filter, but with a standard deviation too large, there would be many points too far from the robot to be a reasonable prediction of the next pose, and not enough closer points for the particle filter to choose from. Ultimately, the standard deviation parameters were chosen empirically, by isolating the motion model and testing different values in simulation as well as considering the real-world scale of the standard deviation. Our chosen standard deviations for the Gaussian noise distributions were 0.2 m for the x and y directions and 3° for the θ direction.

The code for the motion model was implemented as a sequence of simple steps. The 'MotionModel' class includes an 'evaluate' function that takes as input a set of particles (each with x, y, and θ) and odometry data, and returns a matrix of updated particles. First, the odometry data is decomposed into its component parts dx, dy, and $d\theta$. Then, a random noise value chosen from the specified distribution is added to each particle. The new state for each particle is then calculated by rotating the noisy odometry data into the frame of reference of each particle and adding it to the previous pose. Finally, the orientation angles are normalized to be between $-\pi$ and π and the new particle poses are returned.

The motion model uses a dead reckoning approach to update the pose of each particle, which is, on its own, subject to cumulative error. However, the addition of noise to the model and additional manipulation in the sensor model and particle filter can prevent this buildup of error in locating the robot. In the following sections, we will see how the spreading of particles from adding noise will be collapsed back into a small region for localization.

2.2 Sensor Model

Amy Shi

The sensor model is responsible for calculating the probability of a given particle and the likelihood of a hypothetical pose. To account for various real-world scenarios, the probability of each particle is considered amongst four cases. Firstly, the probability between an estimated distance and what is observed must be considered. To do this, a Gaussian distribution centered around the ground truth distance between the robot and the obstacle is used. This is

because the expected distance from the robot to an obstacle is the ground truth distance, so the highest probability would occur when the measured distance between the obstacle and the robot matches the ground truth distance.

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

The second case considers the probability between the LiDAR scan detecting an obstacle and the distance the LiDAR scan covers. Assuming that obstacles are relatively uniformly distributed in an environment, the probability that the LiDAR detects an obstacle closer to the robot is higher than the LiDAR detecting an obstacle further away.

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

The third case is accounting for the probability of the LiDAR beam potentially reaching the maximum range and returning data that results from reflections and such. By creating a large spike in the probability distribution, this allows the algorithm to account for potential uncertainties while also preventing discounting the weight of probable particle distances.

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

Finally, the last case considered is related to the probability of random events affecting the estimated distance.

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

The four probability cases are then assigned weights and summed together for the total probability of a given particle.

$$\begin{aligned} p_(z_k^{(i)}|x_k, m) = \alpha_{hit} \cdot p_{hit}(z_k^{(i)}) + \alpha_{short} \cdot p_{short}(z_k^{(i)}) \\ +\alpha_{max} \cdot p_{max}(z_k^{(i)}) + \alpha_{rand} \cdot p_{rand}(z_k^{(i)}), \end{aligned} \tag{5}$$

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

In the ROS implementation, it would be computationally taxing to calculate the probability for each particle within each hypothetical LiDAR scan. Thus, the most efficient solution is to have a precomputed table of probabilities for a range of ground truth and estimated distances. To do this, an iterative approach was first utilized to test the accuracy of the algorithm. Then, the iteration was replaced with array operations for increased processing speed.

The precomputed table was structured as a 2D numpy array, with the observed distances on the y-axis and the hypothetical ground truth distances on the x-axis. In the final and more efficient algorithm, the four cases of probabilities were represented as numpy arrays with the same axes as the precomputed table. Then, the p_{hit} array was normalized across the columns of the table because we needed to account for discretized p_{hit} values. Finally, the weighted sums of the probabilities were normalized across the columns of the precomputed table to ensure that the sum of the probabilities will add to 1. This ensures that the particles are properly weighted and the probability distribution is properly maintained.

To evaluate the probability of hypothetical particles and the overall likelihood of a pose, the measurements from the particle scans and robot observation were first converted from meters to pixels. Then, the distance values were clipped to be between 0 and the maximum value of the precomputed table. Finally, the probability of each particle within a scan was multiplied to return the overall likelihood of the pose. After implementing the more efficient algorithm, the computation time was almost 300% less than the original run time.

Table 1: Evaluating Algorithm for Efficiency

	Precompute	Evalute
Iterative	0.0549s	0.00421s
Array Operation	0.0444s	0.00151s
Speed Up %	125%	280%

As illustrated above [5, Tab. 1], the refactored code for both functions were significantly faster than the original algorithms.

2.3 Particle Filter

Christian Teshome

Now that we have the sensor model and motion model working, the final step is using those models to determine the location of the car via Monte Carlo localization. The first step in Monte Carlo localization is to initialize a large number of particles at random locations on the map. Each particle represents a guess at the location of the car. Whenever we receive updates from the sensor model or the motion model, we update our set of particles, moving them closer and closer to the actual location of the car.

First, let's look at how particle locations are initialized. Each particle has an x value (in meters), a y value (in meters), and a θ value (in radians). When the particle filter starts up, every particle has its x, y, and θ all set to 0. The actual initialization step occurs whenever a user clicks "2D Pose Estimate" on

RVIZ, which publishes the user's clicked pose to the "/initialpose" topic. When our callback function receives this pose, it sets the particles around that pose randomly using three normal distributions, The standard deviations for x and y were 2 meters, while the standard deviation for θ was $\pi/6$. We decided on using relatively big values here because we wanted our particle filter to work well even if the initial pose is significantly off from the actual location of the robot. However, if our standard deviations are too big, then it would take it much longer to converge onto the location of the car.

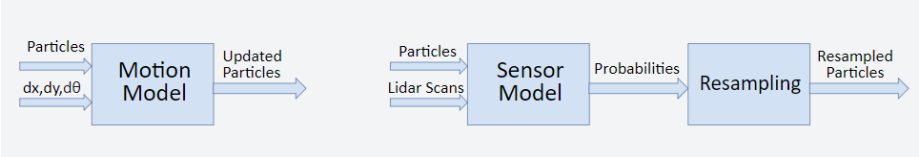

Figure 1: A diagram demonstrating the functionality of our motion model and sensor model

Once the particles are initialized, the motion model and sensor model can start updating them. First, let's consider the motion model. The motion model gets called whenever the particle filter receives new odometry data from the car. This data is stored in the twist component of the odometry, specifically the linear x and y velocities, and the angular z velocity. The motion model expects to receive data in the form of dx, dy, and $d\theta$, so in order to convert the odometry data into this form, we multiplied the twist values by the time difference (in ms) since the last time odometry data was received, which we called dt. Once this is computed, we then pass these values into the motion model, along with our current particle locations, call its evaluate method, and then reassign our list of particles to the output.

At the same time, the sensor model is also used to refine our particle locations. The sensor model gets called whenever the particle filter receives new LiDAR scan data from the car. These LiDAR scans, along with our current particle locations, are passed in to the sensor model's evaluate function, which returns a list of probabilities of size num_particles. Each probability represents the probability that the car is at the corresponding particle location, given the most recent LiDAR scans. This list of probabilities is used to resample our particle locations, so that the most likely particle locations appear more often in our set of particles. This resampling is done with replacement to allow more probable locations to take up a larger portion of our set of particles.

Whenever the particles are updated, either via the motion model or the sensor model, we compute the average position of the particles and publish it. However, there are multiple notions of "average" that we could use to represent our best guess at where the car is. We ultimately decided that we would com-

pute the arithmetic means of the x and y coordinates of each particle. For the θ values, in order to account for the fact that they wrap around 2π, we decided to use a circular mean instead. Essentially, we convert each θ to an x and y coordinate on the unit circle, and then take the arithmetic mean of those x and y values. The "average" θ is then just the angle that corresponds to the average x and average y value from above. We found that a circular mean was the best way to represent our guess at the robot's orientation.

3 Experimental Evaluation

3.1 Simulation

Marcos Espitia-Alvarez

Evaluation of our particle filter in the simulation was enabled by the knowledge of the robot's ground truth pose with respect to the map frame. While running our wall follower, which would regularly make jarring movements, we found that our particle filter was able to approximate the pose in the x direction within 0.4 m, in the y direction within 0.25 m, in the yaw angle within 3°.

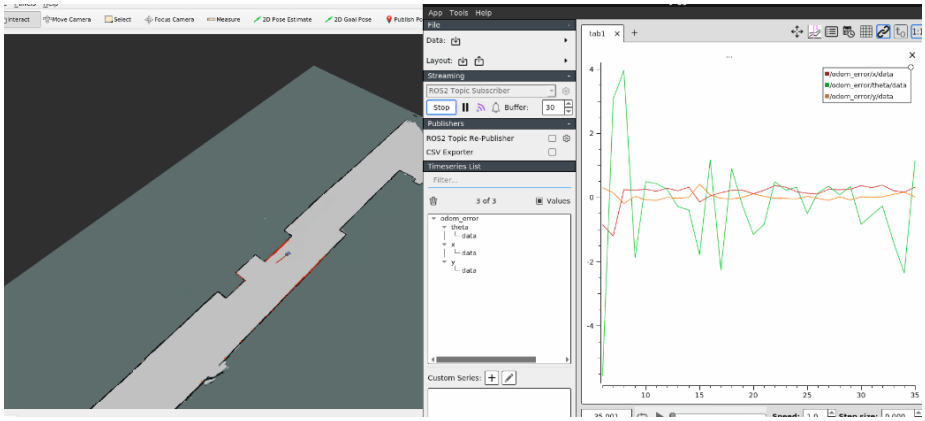

Figure 2: Data collection of odometry error over time, showcasing the visualization tool used and simulation

In simulation, the odometry of the car was assumed to be published without any noise. In the ROS2 node of the particle filter, callback functions were created for when laser scan data or odometry data were published to topics. In these callback functions, another function call was calculated the average pose of the particles, and this average pose was published to a separate odometry message with the transform between the map frame and the average pose frame being published. Another node then listened to the two published odometry messages, one for the car and the other for the average particle, which best represents the best guess for where the car is using our particle filter, and did

the necessary computation and publishing of our error, which we defined as the difference between the best guess of a state variable subtracted by the actual value of that state variable. Because any of the states could have a value of 0, using percent error could yield an undefined amount of error. A state having a value of 0 would also yield a percent increase or decrease equal to 0 or undefined. With the error being the delta of the state values, there are no risks of division by 0 errors, and the computation of the error is trivial meaning it would not affect the runtime performance of our robot.

$$error_i = \Delta_{x_i} = x_{i_{ground}} - x_{i_{particle}} \tag{6}$$

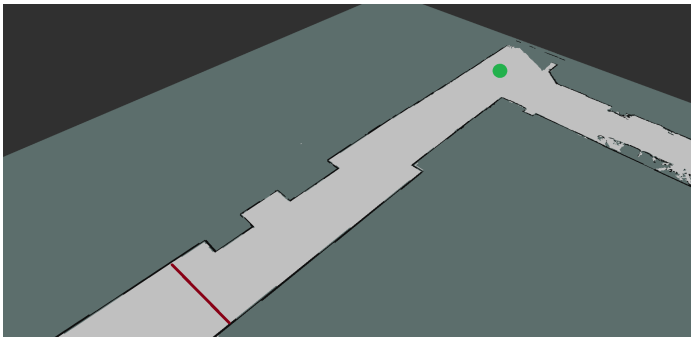

Figure 3: Capture of hallway used for the experiment. The green dot represents the car starting point and the red line determines the finish line for the trials

Experiments were run in simulation on a long narrow hallway of the Stata basement at the same position for each trial. The car would move autonomously in simulation through the help of a wall-follower controller, developed in a previous lab, which takes in LiDAR scans and approximates the position of a wall when enough scans meet certain distance and angle criteria. The controller then steering angle commands to the car to maintain a specified distance from the wall. The hallway contains varying wall geometries, forcing the car to make sharp turns to maintain its desired distance. This is important, as it allows us to determine the particle filter's performance, even under non-steady state conditions. Finally, To visualize the error in realtime and collect data we used PlotJuggler, a tool developed for ROS to provide superior time-series visualization capabilities.

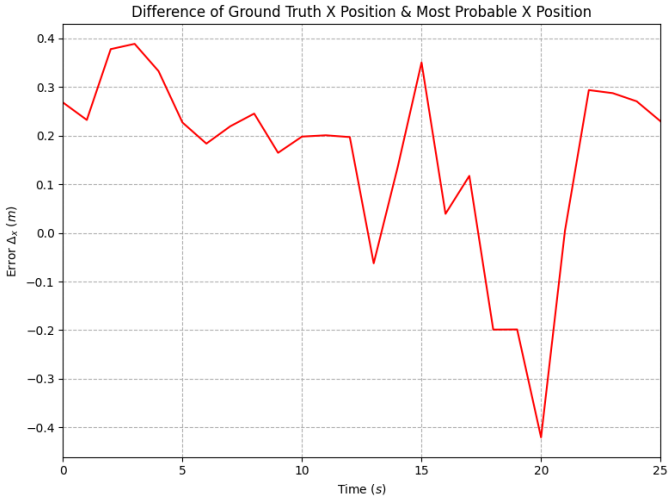

Figure 4: Plot of error between the car ground truth x position versus the most probable x position (average). The difference ranged between -0.4 m to 0.4 m, which is less than the length of the car

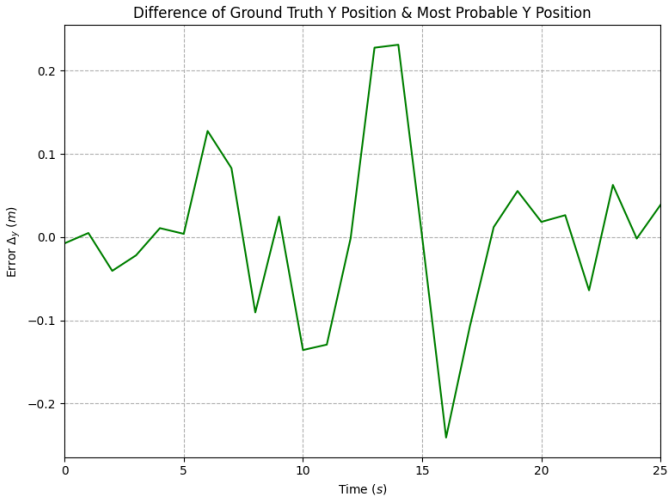

Figure 5: Plot of error between the car ground truth y position versus the most probable y position (average). The difference ranged between -0.25 m to 0.25 m, which is less than the width of the car

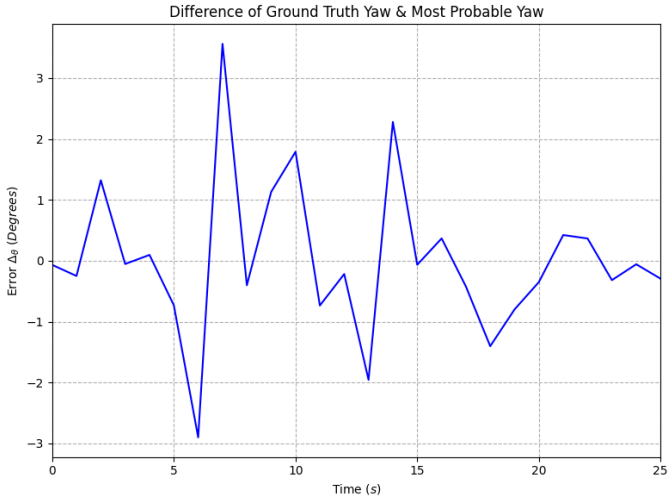

Figure 6: Plot of error between the car ground truth yaw angle versus the most probable yaw angle (average). The difference ranged between $-3°$ and $3.5°$, which is roughly aligned with the front of the car

Spikes in the plots for the data correspond to moments of aggressive maneuvering by the car, such as passing the cubby on the left of the hallway image. Nonetheless, the error in the x, y, and yaw angle were kept between -0.4 m and 0.4 m, -0.25 m and 0.25 m, and $-3°$ and $3.5°$. For reference, the car is roughly 0.3m (y) $\times$ 0.5m (x). Even under non-ideal conditions, the particle filter performed well and under steady-state conditions, the best guess of the car's pose lands within the area of the body. Because the placement of the vehicle in the simulation also provided the initial guess for the particle filter, there was convergence to the correct solution every run, taking around 6 seconds evidenced by recordings of the car.

3.2 Real Life

Timber Carey (primary author), Marcos Espitia-Alvarez

Unlike the simulation, we did not have access to a ground truth pose for the robot with respect to a static frame, which requires precise known fixed locations in the world that we could easily translate to the pixel map. However, we qualitatively found that our particle filter gave a rough estimate of the robot's position when given a decent initial guess of the robot's pose. When playing with the model, we were able to observe several different cases of initial guesses that yielded different rates of success. When the initial orientation guess given to the algorithm is near $180°$ from the actual robot's orientation, the algorithm is consistently unsuccessful. This is because the motion model reads the odometry from the robot and begins moving the particles in the opposite direction to the actual robot drive direction. Once it gets a little ways away, the algorithm

cannot correct itself because the noise in the motion model is not high enough to make the algorithm consider points farther away.

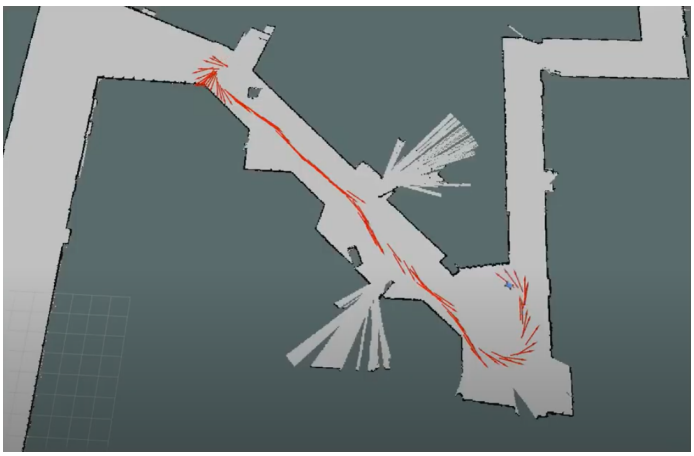

Figure 7: Current and previous average poses of the particles estimating the position of the robot (represented by the car model) as it is driven in realtime inside the Stata basment

On the other hand, if the orientation of the initial guess is near the actual robot's initial orientation but there is some translational error (typically within 2-3 meters), the localization algorithm can self-correct. This happens because the sensor readings are still relatively similar and the algorithm can continuously choose a new guess that reduces the translational error, as the particle with a closer scan will have a greater probability. Finally, initial guesses far away from the robot (simulating the kidnapped robot problem where the robot was teleported from the guess position to its real-world position) were also consistently unsuccessful. We hypothesize that overcoming an erroneous initial guess would require the placement of random particles throughout the map with each resampling, even if the particle filter converged on a solution. In this way, the filter remains open to converging to a far-off solution, which is necessary when correcting from suboptimal solutions.

4 Conclusion

Subin Kim

In this lab, we successfully implemented the Monte Carlo Localization (MCL) method to determine the racecar's pose within a known environment. By incorporating Gaussian noise into the motion model, we observed that the particles naturally spread from their initial positions, effectively simulating the real-world uncertainty that autonomous systems must contend with. The sensor model played a pivotal role in this process, determining the likelihood of the robot's

position for each pose using a pre-calculated table of values. This allowed for a nuanced evaluation of each particle's pose, by using the weighted sums and the multiplications of probabilities to refactor our localization approach accurately.

Our efforts to improve the efficiency of the algorithm were fruitful, particularly our decision to refactor the codes for eliminating for loops as well as for implementing the localization on the real race car from the simulation. The particle filter's application, leveraging both motion and sensor models, facilitated dynamic updates and resampling of particle locations based on odometry and LiDAR scans. We initiated the particle distribution by sampling from a Gaussian (normal) distribution around the initial pose set for the racecar, laying a robust groundwork for the implementation process. Furthermore, our tests in the simulated environment, combined with our wall-follower code, consistently validated the robot's location accurately as it moved through the map. This confirmed the effectiveness of our particle filter against the ground truth odometry.

Moreover, our methodology included rigorous testing of the motion model, sensor model, and particle filter both individually and collectively. These individual/collective tests were conducted in simulated environments and actual real-world scenarios, ensuring comprehensive validation of our system. A pivotal adaptation involved down-sampling the LiDAR data to 100 points for each particle in real-world tests, a modification made to the existing logic that only worked for the simulated environment.

Looking ahead to the next phase, we plan to implement several improvements to enhance the performance of our localization system. We aim to adjust the parameters within the motion and sensor models to improve the system's robustness to noise from the odometry data. Another significant enhancement will involve dynamically optimizing the number of particles used in the algorithm, striking an optimal balance between computational efficiency and localization accuracy. We also intend to integrate simultaneous localization and mapping (SLAM) techniques, which will enable the algorithm to be applied in environments where the map is not a known priori. Addressing the "kidnapped robot problem" is another crucial objective, where we will develop strategies to effectively re-localize the robot in scenarios where it is moved to an unknown location without prior indication. Lastly, we plan to conduct extensive testing of the system under a variety of environmental conditions to ensure its reliability and robustness.

The successful execution of the MCL method within this lab has laid a solid foundation for advancing to the next lab's challenge: path planning.

5 Lessons Learned

5.1 Timber's Lessons Learned

Throughout the course of this lab, I gained valuable insights on the technical aspects of robot localization, as well as lessons in communication and collaboration with my team. By testing and debugging each of the sections of the Monte Carlo Localization algorithm, I was able to thoroughly understand the mathematical and programmatic processes necessary to determine a robot's pose on a map. One important technical lesson that I learned was how to use array operations to eliminate 'for' loops and make a function more efficient. Instead of iterating continuously over an array of values, it is much more efficient to perform a large operation on the entire array at the same time.

Beyond the technical lessons learned, I also was able to practice communication and collaboration skills with my team. By helping to debug sections of code that my teammates had written, I was able to practice asking my teammates to explain their thought processes and communicating about where issues may have arisen and potential solutions. Some long nights were spent with my teammates working on this lab, and I learned that having my teammates present to bounce ideas off of, even if every person is not actively working on some part of the lab, is very beneficial and makes the work go much easier than simply working alone.

5.2 Marcos's Lessons Learned

While working on Lab 6, I gained valuable experience implementing a localization method and, unexpectedly, with trust. I acted in more of a support role during this lab, meaning I had to have a necessary level of understanding of each portion of the lab. I especially gained familiarity with probability and I was opened to the technique of discretization of continuous spaces to make runtime performance on hardware much more efficient. Furthermore, I improved my familiarity with ROS. I was tasked with the evaluation of our particle filter and I had to spend time figuring out the best ways to extract meaningful data from the onslaught of messages being published.

Nonetheless, there were times when I was absent from my teammates, having to take care of coursework and exams in other classes. I feel anxious when I am not in control, when I do not have a say in matters that will affect me later. However, I did not have the time to be caught up, so I learned to trust my teammates' work. I am grateful that they were able to help me and I look forward to helping them more in return, when they have moments where they need to step away to take care of other business.

5.3 Subin's Lessons Learned

In working on the MCL lab, I found myself navigating through both challenges and discovery. This was a good opportunity to bridge the gap between lecture-comprehension of theoretical robotics concepts and the practical applications of deepening the understandings of each module we covered to successfully get the localization algorithm to completion. Adopting each part of the algorithm, and adapting those algorithms from simulation to the real race car highlighted the week's experience for me, as applying theoretical models to the real-life application had unpredictable potential errors. The process of applying problem-solving skills to tackle individual parts individually and collectively instilled a deep appreciation to finally see the work come to fruition.

For working jointly with my team members for extended periods of time, I learned to appreciate their dedication even more for collectively working as a group, doing their best to contribute whilst also having to meet other deadlines for other classes. I also feel closer to my team members as we spent a lot of time together working toward the same goal, discussing amongst ourselves the problem as well as in approaching a consensus with the proposed solution. There's an old saying we used to say in the military: "if you want to go fast, go alone, if you want to go far, go together". In this case, having each others' back and working together allowed us to go fast and far.

5.4 Amy's Lessons Learned

While many of the theories behind Lab 5 were presented in lecture, it was hard to know how to apply them to real world scenarios until we implemented them for this lab. Overall, I learned so much about the purpose of a sensor model, motion model, and particle filter. I learned what the probabilities are used for in terms of localization, and I learned how the likelihood of the robot being in a specific position is calculated. It was a very insightful lab. Besides finally understanding the theory and mathematical expressions, I also learned a lot about how to write an efficient program. For my previous projects, the computation time never mattered as much, so this lab really pushed me to learn how to identify time-consuming algorithms and refactor it for efficiency.

As for lessons learned within the team, I can now better understand each teammates' communication style and know how to keep everyone on the same page. This lab brought our group closer together because while the code was quite difficult to debug at times, our team was there for each other to bounce ideas around and be an emotional support. Personally, I understand the group dynamic better, and I believe this will help us work together even better in the future.

5.5 Christian's Lessons Learned

Throughout this lab, I learned that determining the location of a car is actually a very difficult problem which requires a lot of very specific and efficient computations, although it's a very easy problem to state. This has also been my first time seeing a real life system have this much dependence on probability. Once we had our first draft at an implementation, I also learned how to methodically debug very complex pieces of software and hardware in a structured way. For example, turning off one of the models and seeing what the other model does by itself. Also, echoing a ROS topic to see whether or not it's successfully being published to and seeing what is inside those messages.

Regarding the lessons learned relating to teamwork, this lab has given me a lot of practice at effectively utilizing the strengths of my teammates. Since this class has people from a lot of different majors, each teammate is able to apply the knowledge they've gained from other courses to whichever portion of the lab they were working on. I was also able to practice my collaboration skills in general, when explaining concepts to my teammates or asking them for help with a certain task related to my portion of the project. My team has provided a great example to follow with regards to effective teamwork, so I've been able to follow their example.

Formatting/editing by Timber Carey