Lab #5 Report: Localization

Team #1

Jensen Curtiss
Nick Dow
Dane Gleason
Sadhana Lolla
Sienna Williams

RSS Spring 2023

April 15, 2023

1 Introduction

Sienna Williams

The racecar will competing against others in a fast closed loop track. To be able to do this, the robot must be able to localize itself on the map based on lidar scans of its surroundings.

Lab 3 gave the robot the capability to find walls and create a line based on noisy lidar data using linear regression. This capability will be useful in Lab 5 as it will allow us to compare the lidar scans from various particle positions against the lidar scan that the racecar actually sees.

The average pose estimate of the racecar is calculated through three steps: the motion model, the sensor model, and the particle filter. The motion model takes in the previous particle estimates, odometry data from the racecars motors and adds Gaussian noise to update the positions of the particles. The sensor model takes in the current particle positions, the laser scan from the racecar and returns the probability of the racecar being at each of the particle positions based on the laser scan. The particle filter uses both these models to periodically update the particle pose and publish the transform. Whenever the car moves, odometry data will be received and the motion model will be used to update the array of particle positions. Whenever sensor data is received, the sensor model will update the probabilities of the particle positions and resample the particles. The initialization of the particles is essential as well because the kidnapped robot problem, where a robot must localize itself without knowing its initial position is extremely difficult. Once the racecar's localization capabilities were successful in simulation experiments were conducted to qualify and quantify the localization's performance. Videos were taken of the robot in simulation and real life for comparison of the accuracy of the perceived location. The convergence rate of the particle poses was calculated by measuring the percentage of particles that fall within a 0.1 meter radius of the average. These two metrics allowed us to determine that our MCL localization algorithm was successful and that the racecar is ready for path planning in Lab 6.

2 Technical Approach

2.1 Motion Model

Nick Dow

In this section, we describe the general procedure of the motion model and the key decisions we made in it's design.

2.1.1 General Procedure

The Motion Model was designed to take in odometry data generated by the racecar's VESC module, the controller for the car's motors, and use that information to estimate the car's movement. The data itself takes the form of a ROS Twist message which contains the translational and angular velocities VESC's ROS node estimates from the movement of the car's motors and steering. We know the car is in the 2D plane of the floor, so the relevant measurements for estimating pose are the x and y translational velocities and the angular velocity in the xy plane. To turn these velocities into useful measurements, we take the time since the last motion model update in seconds and multiply that into our velocities to get the car's displacement since the last update. These displacements are from the robot frame of reference, so we use a matrix transformation from the robot to world frame to convert the displacements into the world frame. Once the displacements are in the world frame, we simple add them to the current pose of the car, which is already in the world frame.

2.1.2 Noise

If the motion model got perfect odometry data and updated enough to overcome the geometric inconsistencies that get introduced when theta rapidly changes, then we could purely rely on one particle to determine the robot's location if it were given a correct initial position. Alas, the world is not ideal and noise enters the model, causing it's estimate of pose to drift from the ground truth over time. To overcome this noise, we must introduce noise to our own updates to allow our particles to spread out to cover a number of possible positions to prune later with Lidar data. To not generate improbable particles, we structured the noise

to reflect how the car moves and how outside noise enters the model.
To this end, we sample, for each dimension independently, our noise from normal distribution

$$\mathcal{N}(0, \textbf{displacement measurement} \cdot \textbf{dimension factor})$$

That is, we sample from a Gaussian distribution centered on the measurement with a standard deviation equal to the size of the measurement scaled by a factor determined for that dimension (x, y, θ). These factors were experimentally derived through simulation experimentation aiming to optimize localization performance across noise levels.
Our noise is centered on the actual odometry measurement while diverging from the measurement in a distribution that reflects roughly how likely that noise is. By scaling the standard deviation of the normal distribution by the size of the displacement update, we acknowledge in our model that larger measurements bring larger potential for noise.

2.2 Sensor Model

Sadhana Lolla

The goal of the sensor model is to take in a set of measured scans z_k and the ground truth scans from a potential particle x (given by ray tracing) and compute the probability that the measured scan resulted from the same particle that generated z_k.

2.2.1 Calculating Probabilities

For every scan in the measured range, we calculate the results of four intermediate distributions and then weight and normalize them according to a set of specified weights α. The four distributions are as follows:

1. If the ray tracing algorithm determines that there should be an obstacle detected in the map at distance d, we want to determine the probability that the observed scan actually detected said obstacle. We do this by modeling a Gaussian with mean d and a tunable standard deviation (throughout this lab, we used 0.8 as the standard deviation. The idea here is that if there is a scan in z_k with a value close to d, we will assign a higher probability to that measured scan. The PDF of this probability distribution p_hit is as follows, where $z_k^{(i)}$ is the ith scan of the measured scan:

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

2. We also want to calculate the probability of a short scan– i.e. it's possible that the lidar scan has a scratch or hits the vehicle itself, so it sees a short scan even if there shouldn't be one in the ground truth. The probability of seeing a short scan should decrease as d, the ground truth distance calculated by the ray tracing, gets larger– if the closest object is very far away, we should see a short scan very rarely. We represent this using the following PDF:

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

3. If the observed lidar scan has a very large distance (equal to the maximal distance observable), then this scan is likely erroneous due to bouncing off of a reflective surface and never returning to the car. Therefore, we do not want to assign a low probability so that we don't filter out particles (as described in section 2.3). So, we add a large spike to the probability distribution here as described below.

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

   ϵ is a tunable parameter here that we set to 1 so the probability distribution is well defined.

4. We also weight a small random probability, defined below.

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

The final probability is given by normalizing $p(z_k^{(i)}|x_k, m) = \alpha_{hit} * p_{hit} + \alpha_{short} * p + short + \alpha_{max} * p_{max} + \alpha_{rand} * p_{rand}$.

2.2.2 Precomputing Probabilities

Since computing the probabilities mentioned above at every timestep is quite computationally expensive, precompute a 200 x 200 table of probabilities. The rows of this table represent the different observed scans (varying the value of $z_k^{(i)}$), and the columns represent the varying ground truths d. Since the original scans are given in meters in the world frame, which is a continuous domain, we discretize the original scans into pixels so that we can precompute the above table. We do this by dividing the measured values by the map resolution.

2.2.3 Evaluating a scan

For a given scan z_k, we multiply $\Pi_{i=0}^{N} p(z_k^{(i)}|z_k, m)$ to determine the final probability that the measured scan z_k resulted from the particle with ground truth minimum distance d.

2.3 Particle Filter

Sadhana Lolla

The goal of the particle filter was to take into account the point clouds generated by the motion model and filter out particles with a low probability of producing the measured scans at every timestep. We first initialized the position of the robot by providing the approximate location of the robot: we need to provide this information, otherwise we will have to solve the much harder of localizing the robot in a map with no additional information (also known as the kidnapped robot problem).

To create our initial point cloud, we draw $N = 200$ points from a normal distribution with the pose estimate as the mean and 0.1 as the standard deviation. Then, as the robot moves, we translate these points according to the motion model and add further noise. When adding noise, we face the classic exploration/exploitation tradeoff, where we want to include points that are somewhat far from the robot's current pose to account for shifts or drifts in the motion model. However, too much noise will result in points spreading too fast and all of the points will be too far from the correct robot pose, so we will not get an accurate estimate of the pose. Details of the noising scheme we used are described in 2.1.2. At the same time, whenever we receive new observations from the laser, we calculate the probabilities as described in section 2.2, and sample from the points at a rate proportional to their probability. It is important to note that we sample with replacement, so the number of points N in the model at any point does not change. The goal of resampling is to discard points that are unlikely to represent the car's location and upsample points that most accurately represent the car's pose. Finally, to obtain a point estimate of the car's pose, we average the sampled points.

2.4 Adapting Code to Robot

Dane Gleason

The adaptation of our MCL algorithm to the robot was a key step in achieving desired performance in this lab. While the simulation and unit testing provided by the staff was useful for rapid iteration and evaluation of our model, the remaining step of integration inherent to all robotics was another key step in the process. Our goal with the adaptation was to maintain our high level of simulated performance in the real world while keeping constraints and realities in mind such as computational efficiency, real-world measurement noise, and discrepancies between the simulation model and the real car.

Conveniently, our implementation worked almost right away once we dealt with hardware interfacing and middleware such as changing ros topics or dealing with various errors. Not so conveniently, though, this interfacing was quite a bit more of a challenge than anticipated. We ran into a variety of errors such as the lack of initial pose required for the MCL algorithm to work. Another challenge we faced were inconsistencies once our implementation was successful. While our visualization of the car in RVIZ worked at some times, other times it did not and required restarting on the order of 5-6 times to get a repeatable result. We believe that this is attributed to the ros processes themselves and not a bug in our code, such that it was an inconvenience but not completely inhibiting for the success of the lab.

3 Experimental Evaluation

Sienna Williams and Dane Gleason

Several experiments were conducted to qualitatively and quantitavely assess the racecar's localization capabilities. The car was placed in an easily recognizable spot on the map of the Stata basement. This initial pose location was then fed into the car through the /intialpose topic. Then, the car was driven around the basement and the particle poses as well as the odometry was visualized through Rviz.

We qualitatively assessed that the car's localization was accurate by viewing the car's location in simulation and in real life simultaneously. We verified that the car was always able to locate itself accurately when driven in straight lines, circles and around curves. We tested the localization using a manual drive of the car as well as running our wall follower code from Lab 3 on the car. Videos of the car in simulation and in real-life can be viewed on our team's website. The link is as follows: https://rss2023-1.github.io/website/labs/5/. The videos displayed are the localization along with the wall follower code, around a corner, and with the particle pose convergence in both simulation and real life.

We quantitatively assessed how well the car's MCL algorithm was able to converge the particle poses within a 0.1m radius. We wanted to ensure that despite the random noise added to the motion model and the random sampling of points in the sensor model, that the particle poses would converge to the correct location. We published the percentage of points that were within a 0.1m radius from the center of all the particle poses and compiled the data over time as shown in Fig. 1.

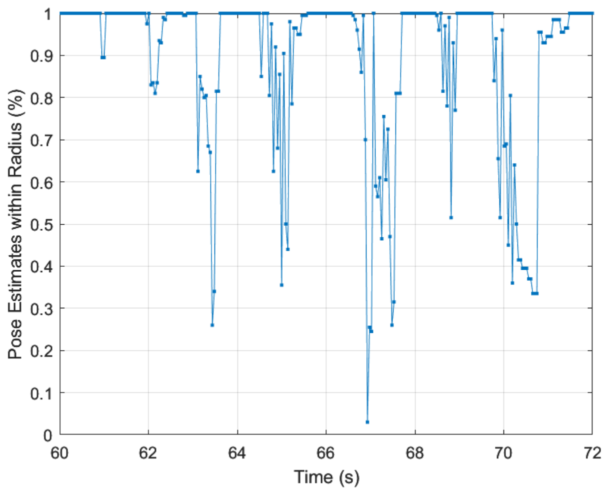

Figure 1. Percentage of pose estimates within a 0.1m radius subject to disturbances over time.

As seen in the data peaks in Fig. 1, the car was subject to a variety of disturbances by including quick accelerations and turns to introduce divergence in the estimated poses. This process is shown in the videos of our car featured on our aforementioned website. With the percentage of pose estimates within the 0.1m radius over time and disturbances, we could then calculate the convergence rate by estimating an average positive slope. As shown in Fig. 2, the slopes were created between the lowest point of each peak and the respective peak's first convergence to 100%. The average of the slopes were taken resulting in an average convergence rate of 227% per second or about 50% in 0.2 second.

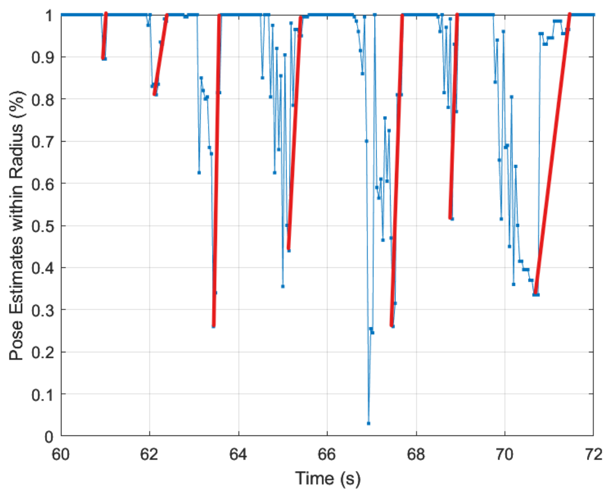

Figure 2. Pose estimate slope estimates for convergence rate calculation.

As this lab prepares us for the final challenge, we must look forward to future implementations to evaluate this measurement. Per course staff, the final challenge will see cars travelling up to 10 m/s. Using Fig. 2 as a reference, we can see that disturbances usually introduce a 50%-60% decrease in pose estimates falling within the set radius. If there is a disturbance in the final challenge such as a turn, we would expect a similar response. Using our convergence rate from earlier, we find that at 10 m/s, the car would travel 2 meters before converging completely. Assuming linearity of convergence rate, 1 meter after a disturbance would present 75% convergence, certainly not a bad result. Consequently, we find the convergence rate to be extremely satisfactory for current use and satisfactory for future implementation.

Although we have taken both qualitative and quantitative evaluations of our real-world performance, there are further experiments that we feel could improve our evaluation and thus our performance in the future. Most notably, given more time we would have liked to quantify our estimated average pose over time to the ground truth in the real world. This was the central feature in testing our simulation for good reason as it would provide a much more intuitive and insightful look into the discrepancies between our simulated and real-world performance. Realizing this in the amount of time given was challenging, mostly in establishing a real world ground truth. We considered using running our wall follower a straight section of wall, but the oscillations made it difficult to produce a ground truth we were confident in. Because of the oscillations, we could not simply set out ground truth as a line parallel to the wall at our reference distance and get an accurate result. Though we are sure there is a more savvy approach, the other challenges presented by this lab took up the rest of our time. We also hope to conduct future experiments to balance computational power and performance, such as the number of estimate poses.

4 Conclusion

Nick Dow

In our semester-long progression towards the final challenge of racing our robot in Stata basement, we have taken another large step forward in this lab. Using Monte Carlo localization, our robot can localize itself in a mapped environment. This technique involved implementing a motion model, implementing a sensor model, integrating the two models in simulation, and porting them to the robot. Using odometry data, the motion model updates a list of potential positions of the robot, known as particles. The sensor model then prunes and resamples these positions using Lidar data, a map of the environment, and a probabilistic model of the likelihood of Lidars measurement given a particle's position. Integrating the two models involved some interfacing code, tuning an array of parameters, and debugging their interactions. Finally porting the code onto the car wasn't trivial given some of the hardware hurdles we've encountered in the past, but was eventually successful. Experimentally, our particle filter performs well in both simulation and on the robot platform. We confirmed good performance qualitatively by comparing videos of the robot in simulation and reality to show localization performance. We quantitatively verified this performance with convergence rate metrics. With localization now behind us, we are now able to approach planning a path through an environment in which we can now localize ourselves.

5 Lessons Learned

5.1 Nick Dow

I have learned the effectiveness of identifying bottlenecks in projects and how to structure our workflow around it. We usually stick to a division of work decided upon at the start of the lab. During this lab, I switched from a physical porting role to implementing the motion model because the original team member assigned to the role was coming back later than me from spring break. Getting the motion model done was an obvious bottleneck to finishing the particle filter which itself was a bottleneck to implementing it on the car. So by identifying this bottleneck and reallocating the work, we made the lab less of a time crunch and easier on ourselves.

5.2 Sienna Williams

I learned about how important it is to check all the resources available to me when running into challenges. There were several errors that we ran into when trying to adapt our code to the robot that we spent hours trying to debug only to find that the solution was on piazza. It also helped to discuss our problems and approaches with other groups to avoid being stuck on one thing for too long. It was also nice to be able to switch roles with my other group members when I came back from spring break late and this allowed us to finish the project smoothly.

5.3 Sadhana Lolla

I really enjoyed this lab overall! I thought that the group split work evenly and finished tasks proactively, enabling us to make progress quickly. We rearranged tasks to reduce bottlenecks and speed up progress. Like Sienna, I found the Piazza really helpful for this lab, since it helped us debug some issues much faster than us debugging alone. Finally, although we ran into some challenges (I could not submit to the autograder due to incompatibility with M1 Macs, and Velodyne adaptations), I thought this lab went really smoothly.

5.4 Dane Gleason

Over the duration of this lab, I have learned that being open minded when handling tasks is not only advantageous for productivity but also personally enriching. As Nick mentioned, some time through the lab we found it better to reassign tasks due to availability. With this, I ended up working on integration and got a lot of hands on experience that reinforced my understanding of hardware-software interactions and how I can write software more mindfully in the future to be more conducive to integration. I also learned the value of having teammates to debug with. With each error we ran into, there was always one of my team members to offer a solution that the rest of hadn't thought of. Overall, this was a very rewarding lab to be a part of.