Lab #5 Report: Robot Localization

Team # 6

Sera Hamilton
Carlos Sanchez
Yuewei Liu
Miguel Padilla
Willow Pickernell

RSS

April 12, 2025

1 Introduction

This lab is about the problem of localization, which consists of using sensors to determine the position and orientation of the robot within a known map. Localization is crucial within a robotic system as once the robot knows where it is within a map, it can then use this map for navigation purposes. To solve the problem of determining the robot's position we implement Monte Carlo Localization. MCL works by creating an initial set of particles to estimate the robot's location within the map. These particles are then updated using Bayesian updating techniques to get an improved estimate of the robot position.

We broke this down into three parts:

1. A motion model to update the positions of the particles based on the robot's velocity and change in time
2. A sensor model to calculate the likelihood of each particle using input from the LiDAR
3. A particle filter which relies on the two models to update the set of particles and taking the average of them as the robot's location

We first simulated the robot using ROS2 in RViz and then implemented the algorithm on the actual car, which is detailed below.

2 Technical Approach

Our technical approach consists of the development of a motion model, sensor model, and the integration of the modules into a full particle filter for Monte Carlo Localization.

2.1 Motion Model

Given a particle cloud, our motion model updates the pose of each of the particles by the odometry reading, Δr, reported by ROS2. We compose the new pose of each particle $T^W_{r_{t+1}}$ from the previous pose $T^W_{r_t}$:

$$T^W_{r_t} \cdot T^{r_t}_{r_{t+1}} = T^W_{r_{t+1}}, \ \textit{where} \ T^{r_t}_{r_{t+1}} = \Delta r.$$

Since this is a physical system and thus subject to error, we sample two gaussian distributions, both centered at 0 (i.e. mean $\mu = 0$) with $\sigma = 0.005$ for the first and $\sigma = 0.15$ for the second. Sampling from these distributions—for each pose axis (x and y) and rotation (θ)—allows us to model the error of the odometry independent of the source or magnitude of errors.
Initially, we sampled from only the first distribution for both the position and the rotation of the odometry reading, but after consulting with course staff we decided to have distinct $\sigma_{position} = 0.005$ and $\sigma_{rotation} = .15$ to allow a higher variance for the noise applied to the rotation of the odometry reading.

2.2 Sensor Model

To update the particles based on the lidar data, we needed to develop a sensor model. We first computed a sensor model lookup table that allowed us to lookup the likelihood $p(z^i_k|x_k, m)$, which yeilds the probability of sensor reading z^i_k given the hypothesis position x_k and map m at time k.

The likelihood can be broken down into the sum of four different probabilities:

1. p_{hit} : The probability of detecting a known obstacle on the map

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

2. p_{short} : The probability of a short measurement due to hardware issues or unexpected obstacles

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

3. p_{max} : The probability of a missed measurement due to reflective materials

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

4. p_{rand} : The probability of an arbitrary, erroneous measurement

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

The likelihood is then the weighted average of the four distributions, where $\alpha_{hit} + \alpha_{short} + \alpha_{max} + \alpha_{rand} = 1$:

$$p(z_k^{(i)}|x_k, m) = \alpha_{hit} p_{hit}(z_k^{(i)}|x_k, m) + \alpha_{short} p_{short}(z_k^{(i)}|x_k, m)$$

$$+\alpha_{max} p_{max}(z_k^{(i)}|x_k, m) + \alpha_{rand} p_{rand}(z_k^{(i)}|x_k, m)$$

From these equations, we created a precomputed table using parameters $\alpha_{hit} = 0.74, \alpha_{short} = 0.07, \alpha_{max} = 0.07, \alpha_{rand} = 0.12, \sigma = 0.8m, \eta = 1, \epsilon = 1$. The table, pictured below in Figure 1, had a size of 201x201, representing a range of 0 to 200 pixels (and as such $z_{max} = 200$. Further, a row index i represents the distance of the observation, z_k^i and a column index j represents the actual distances d.

Additionally, the table must be normalized so that each column represents a probability. First, since we are working with discrete values, we normalize p_{hit} by dividing each by the sum of all the p_{hit}'s in the column. Then the whole table is normalized by dividing each column by the sum of the entries in the column. From this, we have completed precomputed table as pictured below in Figure 1.

From here, we can develop an evaluate function to generate the likelihood of each particle given the current observation z_k.
To do this, we first simulate what the expected ranges, d, would be for each particle. We divide both z_k and d by the map resolution and lidar scale to map scale to convert the values to pixels for ease of use with the lookup table, and clip the values between 0 and z_{max}. Then, the likelihood of a scan for each particle is the product of the likelihoods of each measurement in the scan, which we can find in the lookup table.

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

2.3 Particle Filter

Using the sensor and motion model, we can construct a particle filter to localize our robot within a map. First we initialize a set of 200 particles around our guess of where the robot is in the map, as pictured in Figure 2. Let the initial guess be x_0, y_0, θ_0. We then initialize our particle positions by drawing from a Normal ($N(\mu, \sigma)$) distribution , $X \sim N(x_0, 0.3), Y \sim N(y_0, 0.3)$. We tried a few different values for standard deviation and landed on $\sigma = 0.3$m as values like

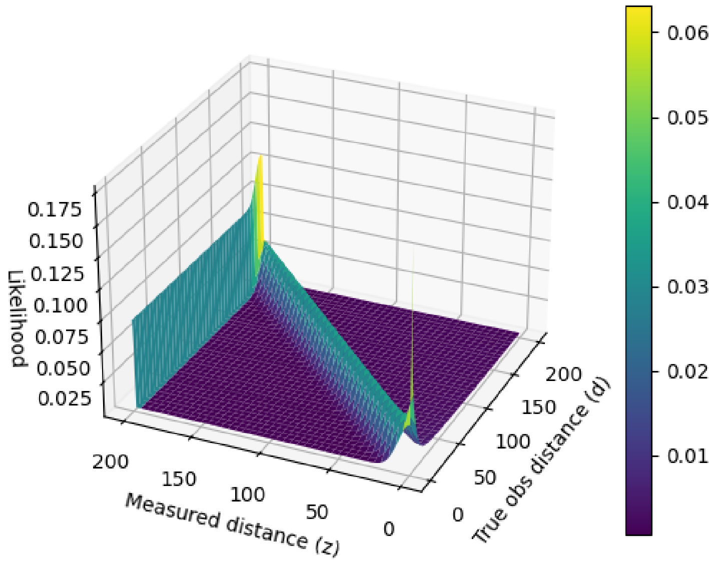


Figure 1: Plot of the precomputed sensor model table. We can see that along the diagonal as the measured distances get closer to the true distances, the probabilities of the scan increase.

$0.5, 1.0$ were too high and $0.1, 0.2$ too low. Our set of theta values were chosen from a uniform distribution, $\theta \sim Uni(-15 + \theta_0, 15 + \theta_0)$. We initially tried a normal distribution, but because we wanted our theta values to be more evenly distributed around our guess without being too large we decided on a uniform one.

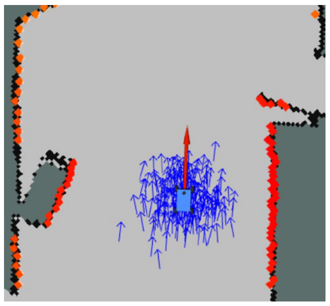


Figure 2: Particles (blue) initialized around an initial guess of the car location.

From here, we add in the motion and sensor model. Each time we get new odometry data, the motion model as described in 2.1 is used to update the particle positions. Each time we get new sensor data, the sensor model is used to evaluate the likelihood of each particle given the sensor data. From here, we resample the particles based on their likelihoods. Additionally, every time the particles are updated we compute and publish the new average position of the particles as our estimate of the car's location, along with a transformation from "map" to "base_link" to relate the world frame to the body frame. A video of our implementation can be viewed here. In the video, we can see that as the car gets closer to more irregular geometries it is better able to orient itself in the map.

3 Experimental Evaluation

We tested the performance of our localization implementation in both simulation and the real world.

3.1 Simulation

In simulation, we individually tested the primary components of our localization algorithm: the motion model, sensor model, and combined model.

When testing the motion model, we published the particles on the Stata basement map in Rviz. We used the pose estimate tool to reposition the robot and observed the resulting particle distribution and estimated pose of the robot given our motion model. We tested both a deterministic model and one with noise from a normal distribution centered at 0, and varying standard deviations. Noise was applied to x, y, and theta.

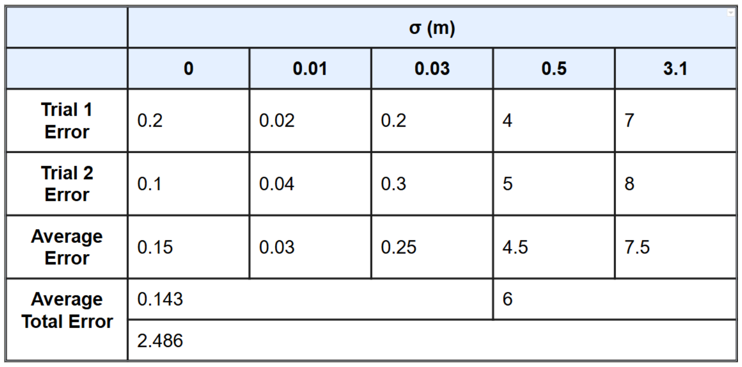
We performed the same process with the addition of the sensor model. We overlaid our localization estimate with the laser scan to qualitatively evaluate how well our localization aligned with the ground truth. Due to latency in the simulation, we decreased the number of particles from 500 to 200. The combination of 200 particles (100 post-downsampling) greatly reduced latency.

We rigorized our evaluation by quantifying the translational error between our estimate of the robot's pose and the ground truth pose.

$$\epsilon_{pose} = \sqrt{(\Delta x^2 + \Delta y^2)}$$

for $\Delta x = x_{real} - x_{obs}$, $\Delta y = y_{real} - y_{obs}$

We obtained ground truth odometry pose values through a provided ros2 topic. To track error over time, we turned on our wall-follower and graphed the translational pose error. We varied noise levels by adjusting the standard deviation of our sampling distribution, as depicted in Figure 3 below.



	σ (m)				
	0	**0.01**	**0.03**	**0.5**	**3.1**
Trial 1 Error	0.2	0.02	0.2	4	7
Trial 2 Error	0.1	0.04	0.3	5	8
Average Error	0.15	0.03	0.25	4.5	7.5
Average Total Error	0.143			6	
	2.486				

Figure 3: Simulated error for varying noise levels on a straight line drive.

Simulated error was also plotted for one trial and it was confirmed that error remained stably low across the run, for the noise parameter we ultimately selected (0.01).

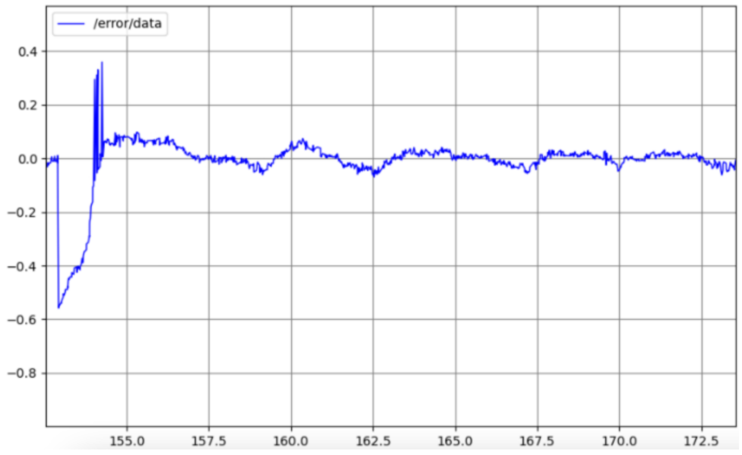


Figure 4: Error for a simulated drive.

3.2 Real World

Once we were satisfied with our algorithm's performance in simulation, we tested it on the robot. We performed a similar process as in simulation but had to measure error slightly differently since we no longer had a ground truth topic to grab the real pose. Instead, we performed a qualitative and quantitative assessment of our robot's performance as follows.

We initialized the full localization algorithm and started the provided Rviz map in order to visualize our particles and pose estimates. We were also able to visualize how well our estimated laser scan aligned with walls in the map.

In our first test, we aligned the robot at a distance of one meter from a wall and initialized the pose of the robot to that same position and location on the map in RViz. Essentially, we are starting the robot in the same spot both in the real world and on the RViz map. Then, we drove the robot through varying locations of the Stata tunnels and observed how well the laser scans aligned with the map walls.

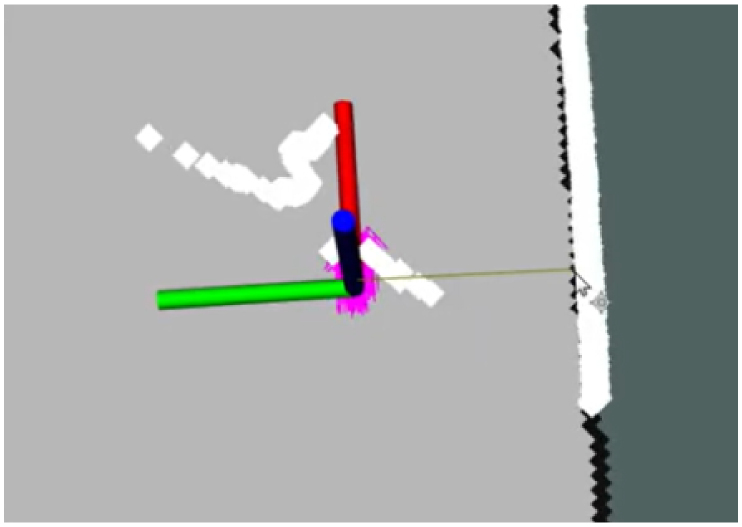


Figure 5: Distance from the wall as measured based on the robot's calculated position in the map, (thin olive-colored horizontal line) pulled from RViz.

We then performed a quantitative assessment of our localization by starting the robot at a known location matching in Rviz and real life. We then drove the robot until some stopping point and compared the robot's ending position with the Rviz estimate using known landmarks both in the real world and the map. We then measured the orthogonal distance from the wall (real) and the estimated distance from the wall (RViz), and computed the error or difference between these measurements. We tested this on a straight hallway, a short curved hall, and a longer loop. Error estimates are provided in Figure 7 below:

We see that the localization performed accurately, with an average error of just 4.42 centimeters. In the stata environment, this level of accuracy would be sufficient to safely navigate obstacles at moderate speeds.

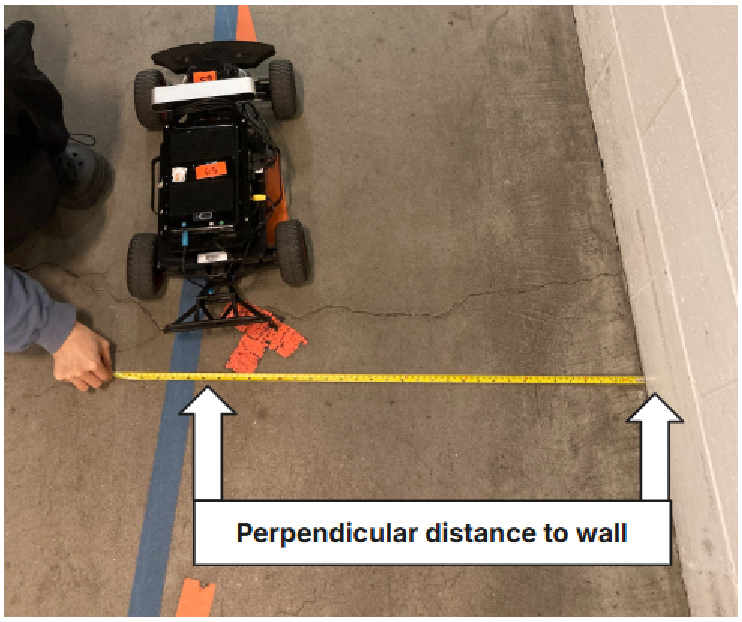


Figure 6: Real life measured distance from the right wall for corresponding trial with above screen capture from Rviz (prior figure.)

	Trial 1 (Straight)	Trial 2 (Short Curve)		Trial 3 (Longer Loop)	
RViz Meas. (cm)	144	175	74.9	275	110
Actual (cm)	139	182	75	269	114
Error (cm)	5	7	0.1	6	4
Average Error (cm)	4.42				

Figure 7: Error from real position measurements vs calculated location.

4 Conclusion

Overall our localization implementation on the robot with particle filtering was highly successful and the robot was able to accurately deduce its location to within a 4.4 centimeters of the true location. The algorithm performed well, especially along complex geometries as the sensor model had more unique information to pinpoint the location in the space. However, an area for improvement could be in situations with less distinct features, such as when traveling down a straight, plain hallway, as in this case the robot has less information to determine where exactly it is in the map.

Further work could also be done to ensure the particle filter works accurately in real time, even at higher speeds. This would require optimizing our code to be more efficient and experimenting with different angular noise ranges to reduce the observed lag in angle prediction.

Additionally, one next step we could take would be to implement a full SLAM system in which the robot is actively mapping its environment while localizing itself within the space.

5 Lessons Learned

Sera Hamilton

I learned the importance of strong communication and ensuring everyone was on the same page. When we were debugging a problem, it was important the before we even questioned why the system was going wrong we would first describe how each section (the motion model, sensor model, and particle filter) should be behaving. Then, we would go through our implementation line by line to make sure that what was written matched with how everyone expected it to behave. On the more technical side, I learned the importance of having multiple sources of measurement, such as both the motion and sensor model, worked together to create a better estimate of an average position than either of them would have alone.

Carlos Sanchez

One of the lessons I learned from this working on this lab is the importance of scheduling time to meet as an entire group. This lab had a lot of modules that built of each other. It was much more helpful when we met as a group and alternated between worked independently and as a group. This method help us find a rhythm for productivity. Additionally, from a technical standpoint, it was interesting to learn about a sampling based approach to localization. I was able to apply some of the skills I learned in statistics and probabilities courses to a functional robot.

Yuewei Liu

It's extremely important to make sure all team members are on the same page about the state of the code and what problems are currently being worked on and by whom. One challenge we encountered this lab was that due to the nature of collaboration with git, people were often coding asynchronously from their local copies of the code. This made it unclear who was working on what part, and at times redundancy and conflicts occurred that could have been avoided with more explicit communication. This lab also taught me about the thought that goes into designing a system with design constraints (for the particular hardware) such as functioning at high frequencies. It was cool to see how lookup tables and precomputing for discretized inputs could drastically decrease runtimes.

Miguel Padilla

I learned a lot about the importance of organization, time management, and effectively communicating the division of work. With this lab we had clearer deadlines of when we wanted certain parts to be finished. We also more strictly laid out times when everyone was available to meet and enforced meetings outside of class time. This helped us feel much more organized as compared to previous labs. Additionally, I learned about the difficulties of debugging and pinpointing the problem. For instance, some issues we ran into were less with our conceptual understanding of the algorithm and more with our actual implementations.

Willow Pickernell

Communication regarding changes made to the project, how they affect the project, and how said changes function are crucial for the technical aspect of projects like these. There were times when our group met and someone who had changed something wasn't able to make it, which caused us to spend time decoding the affect their change had before we could build on top of it. In terms of technical skills, I learned about how Monte Carlo localization works in practice and how to work with the various pieces of hardware the robot has to determine how the environment around it looks and where it is in the environment as a whole from that information (and from the odometry data).