Lab 5 Report

Team 3

Introduction (Nicholas, Lennie)

The focus of this lab is localization, which is determining the position and orientation of the racecar within a known environment using relevant sensor data. Developing a working Monte Carlo Localization (MCL) algorithm is an essential part of higher-level navigation algorithms in our goal of developing a robust robotic system. Knowing the position and orientation in space is crucial for such systems to have more autonomy in planning out paths, for example.

In this lab, we implemented a working MCL that determines and updates the position of the racecar given odometry and sensor data. Since MCL relies on random sampling and statistical modeling to approximate the position of the robot as it moves, our MCL avoids the problem of having to quickly compute generic PDFs (probability density functions) that makes implementations of the Bayes Filter intractable in practice.

Following the lab specifications, we implemented the motion and sensor models based on our formulas from Part A, then coded the particle filter to publish the new “average” particle pose in simulation for Part B. We then refactored our implementation to work on the physical racecar for Part C. In the following sections, we will expand on our technical approach as well as evaluate how our MCL performs both in simulation and in the Stata basement.

Technical Approach (James, Nicholas, Lennie, Autumn, Michael)

Our implementation of the MCL is split into three modules: motion model, sensor model, and particle filter. In the **Initial Setup**, we go over the relevant data acquired from the racecar that the MCL needs. The **Technical Approach** section explains the theory behind MCL and clarifies how each building block contributes to localizing the racecar. Finally, the **ROS Implementation** section illustrates specific design choices we made.

1) Initial Setup

Our setup consists of a known map OccupancyGrid which gives us a defined space the racecar is in, Odometry data, and LaserScan data. We use the twist (linear for position and angular for

orientation) component of the Odometry message to find the change in velocity, then dividing by the time difference from the last received Odometry message to get the change in position/odometry $(dx, dy, d\theta)$. This odometry is then passed to and used by the motion model to update the particles. The LaserScan data provides the sensor model information about distances to nearby obstacles and is used to calculate the probability of each particle existing given the observation and the map.

2) Technical Approach

The MCL is split into 3 main modules in our implementation: the **motion model**, **sensor model**, and **particle filtering**, where each serves a different purpose. In the following sections, we will explain more in-depth about how these work and relate to each other.

Motion Model: The motion model takes in the old particle pose and the current odometry data, then returns an updated pose with the odometry applied to the old poses. For each of the particles, we represent it as the following matrix with the rotation matrix:

$$\text{particle} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix}$$

In the odometry transform matrix, we include random noise obtained from a noise node to the odometry to represent uncertainty and allow the particles to spread out as the car moves. The noise is calculated by a normal distribution with variance depending on the odometry positional data. The variance is set based on the movement of the car (dx and dy) but has a **fixed** variance for the angle. If dx or dy is 0, a fixed variance is set to make sure the particles are always moving. On the other hand, the angle always has a fixed variance as we always need to be adjusting for the angle of the racecar even if it is not turning a lot. Overall, this design choice makes sure that the particles cover more range when the racecar is in motion. Making a separate node also allows us to easily change the type of noise as well as remove it for optimization and debugging purposes.

$$\text{odometry transform} = \begin{bmatrix} \cos(d\theta + \text{noise}) & -\sin(d\theta + \text{noise}) & dx + \text{noise} \\ \sin(d\theta + \text{noise}) & \cos(d\theta + \text{noise}) & dy + \text{noise} \\ 0 & 0 & 1 \end{bmatrix}$$

When given no noise, the model is deterministic and the particles follow the racecar's movement exactly.

We multiply the two matrices (particle $\mathbf{x}$ odometry transform) to get the new particle values, and the motion model returns the particles array with this transform applied to each.

Sensor Model: The sensor model defines how likely it is to record the sensor reading z_k from a hypothetical position x_k in a known, static map m at time k. The model assigns likelihood weights to each particle by computing the product of likelihoods of the range measurements in the scan. This means that good hypotheses will have more of a chance being resampled, which allows the particles to converge and collapse the spreading of particles from the motion model back down.

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

$$= \prod_{i=1}^{n} p(z_k^{(i)}|x_k, m) \tag{2}$$

Since particle positions are updated every time, we **precompute** the sensor model by discretizing the range values into 200 possible values: we evaluate it on a grid of actual distance d (from raycast) and measured distance z (from sensor), and we store the values in a look-up table to be much more efficient and improve publishing frequency. For look-up, the sensor model sets LIDAR ranges to 0 if they are negative and caps them at 200 if they are over that (units are converted from meters to pixels for processing). This precomputation also allows us to numerically normalize the probability easily.

To compute $p(z_k|x_k, m)$, we consider 4 cases:

1) Detecting a known obstacle in the map: We represented this as a Gaussian distribution centered around the ground truth distance between the hypothesis pose and the nearest map obstacle. If the measured and expected distances are similar, then this probability is very large.

2) Short measurement due to unknown obstacles: This was represented as a downward sloping line as the ray gets further away from the robot since we expect the lidar to pick up on obstacles closer to the racecar.

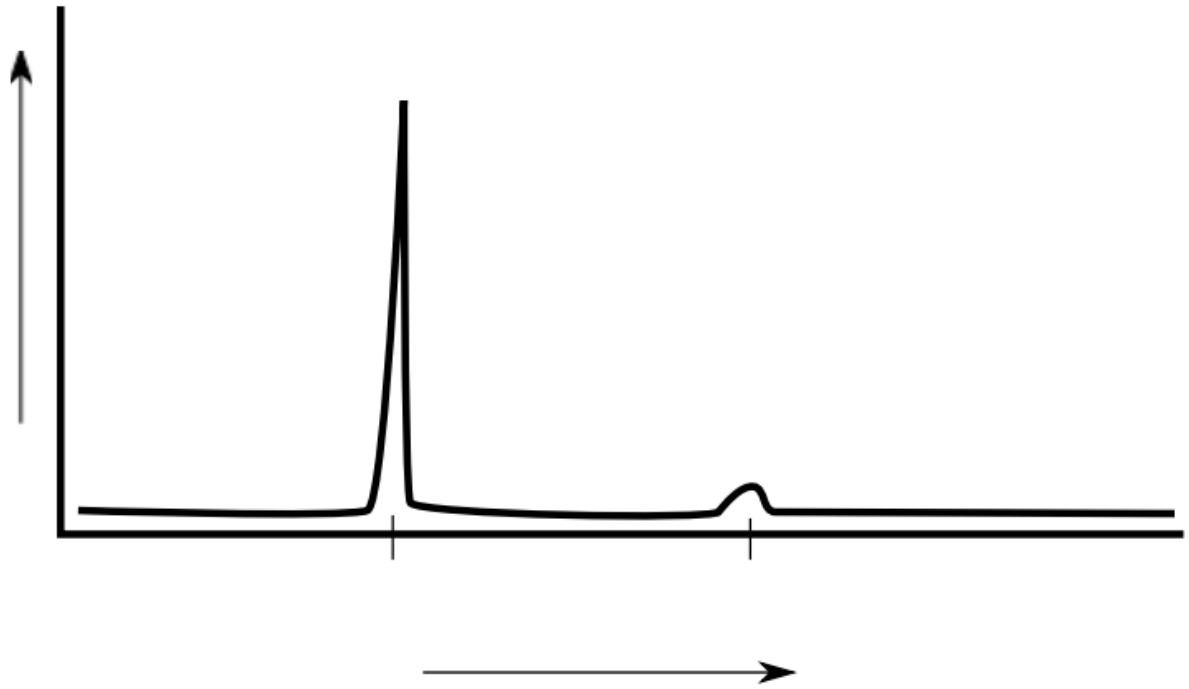
3) Large or missed measurement usually due to strange reflection: We represented this as a uniform distribution with a large spike at the maximal range so that the reflected measurements do not significantly discount particle weights.

4) Random measurement: This is just represented by a small uniform value to account for completely unforeseen events.

The sensor model returns p as a weighted average of these 4 distributions with the weights adding up to 1.

Particle Filter: We initialize particles using a normal distribution around the chosen point (in the case of the simulator, the chosen point is always the ground truth robot position) so that they can spread out and be resampled to the proper position based on LIDAR scans. This is done when starting the simulation or choosing a point to re-init (2D pose estimate on the simulator) and it prevents the kidnapped robot problem from happening.

The particle filter first uses the sensor model to compute the particle probabilities, and these probabilities are than "squashed" (taken to the power of 1/2) in order to ensure that the probabilities for our model for likely distances is not "overly precise." Ironically, this can lead to issues as if the model is too precise, particles are unlikely to fall within a "sharp" peak, which can mess up the model. We squash the peaks to make the models less precise so there is a wide enough range for the particles to fall into. (CITE THESE SOURCES USING IEEE FORMAT FOR THIS POINT: https://papers.nips.cc/paper/1998-kld-sampling-adaptive-particle-filters.pdf (THIS IS WHERE THE PICTURE COMES FROM) AND https://web.archive.org/web/20170209092754/http://www.cc.gatech.edu/~bboots3/STR-Spring2017/Lectures/Lecture4/Lecture4_notes.pdf)



We then normalize the probabilities and resample the particles. Next, using information from the robot (twist component) and previous odometry data (if we have any) we find the current odometry data and, using the motion model, we update the particle positions and publish the updated pose and the transform.

When it came to publishing the pose, using average or median wouldn't work, especially if the distribution is multi modal since the "average" pose would be entirely different. Thus, we compute the average pose of the robot by first finding the largest cluster among the particles using the unsupervised machine learning model DBSCAN. While it is computationally more intensive, the improved accuracy is much more important and justifies the use of clustering mean over a simple mean. Then, we take the average x and y positions of the particles and the circular mean of their angles θ to arrive at the average pose. Using the newly determined pose, we compute the convergence distance and publish the transform.

3) ROS Implementation:

Our particle filter is implemented in a single node in ROS in particle_filter.py. In the initialization, we subscribe to Odometry data received from /vesc/odom and LaserScan data from /scan, which are used by MotionModel and SensorModel respectively. We realized while doing the lab however, that we needed to negate the odometry values while refactoring the code to work

on the physical racecar. In addition, we also subscribe to /initialpose, which allows us to receive pose initialization requests in the form of a PoseWithCovarianceStamped message. After running MCL, the pose estimate (with respect to the /map frame) represented in the pose field of the Odometry message is published through /pf/pose/odom.

In order to visualize the particles and where they are pointing in RViz, we also publish a PoseArray to /pose_array_topic. This is done in a publish_pose_array() helper function which gets called at the end of the odom_callback(msg) and laser_callback(msg) functions after the particles are updated. In our code, we also make sure to add a thread lock to prevent self.particles from being modified by both odom_callback and laser_callback.

For evaluation, we publish the convergence distance to /convergence_distance after publishing the estimated pose at the end of the callbacks. This checks to see if the convergence distance, the average distance across all particles to the average pose, is within a certain threshold (0.05) and publishes 1 if it is.

Experimental Evaluation (Michael, Lennie, Nicholas, James)

In order to evaluate the functionality of our MCL and use the results to iterate on our implementation, we designed and measured a few metrics to compare the accuracy of our code. The three main components—the motion model, sensor model, and particle filter—all need to be tested. The motion and sensor models were tested using the unit tests provided by the course staff, and all the tests passed.

We debugged the particle filter by turning off the sensor model first, then the motion model.

With only the motion model, we noticed that the particles gradually expand outwards in the direction of movement because of the lack of resampling and weighted probability that lowers the chance of bad hypotheses from passing through, which the sensor model provides. We also tested it with and without noise: in the simulation, we observed that the MCL approximation loses accuracy significantly on turns without noise. In the simulation, we use odometry data but in the real word, we run the localization at the same time as wall follower and use the average distance from wall to help us determine the ground truth position relative to the average position.

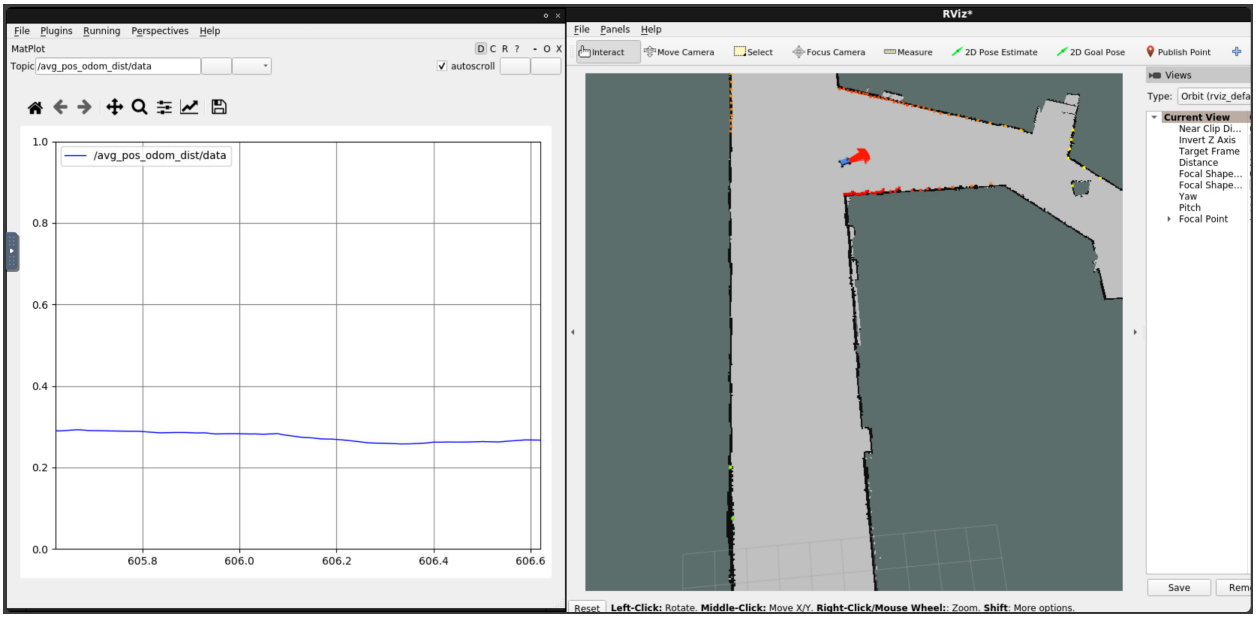


Figure 1: The distance from the average pose of the particle filter and the ground truth odometry pose of the robot remains constant at ~0.3, even while the robot is turning. This metric encapsulates the overall accuracy of the particle filter in simulation. The particle filter manages to consistently discover the position of the robot and follow it, even while the robot is moving in both x and y and is turning.

Another metric we had is that after publishing the estimated pose at the end of the callbacks, we check if the convergence distance, the average distance across all particles to the average pose, is within a certain threshold (0.05) and publishes 1 if it is.

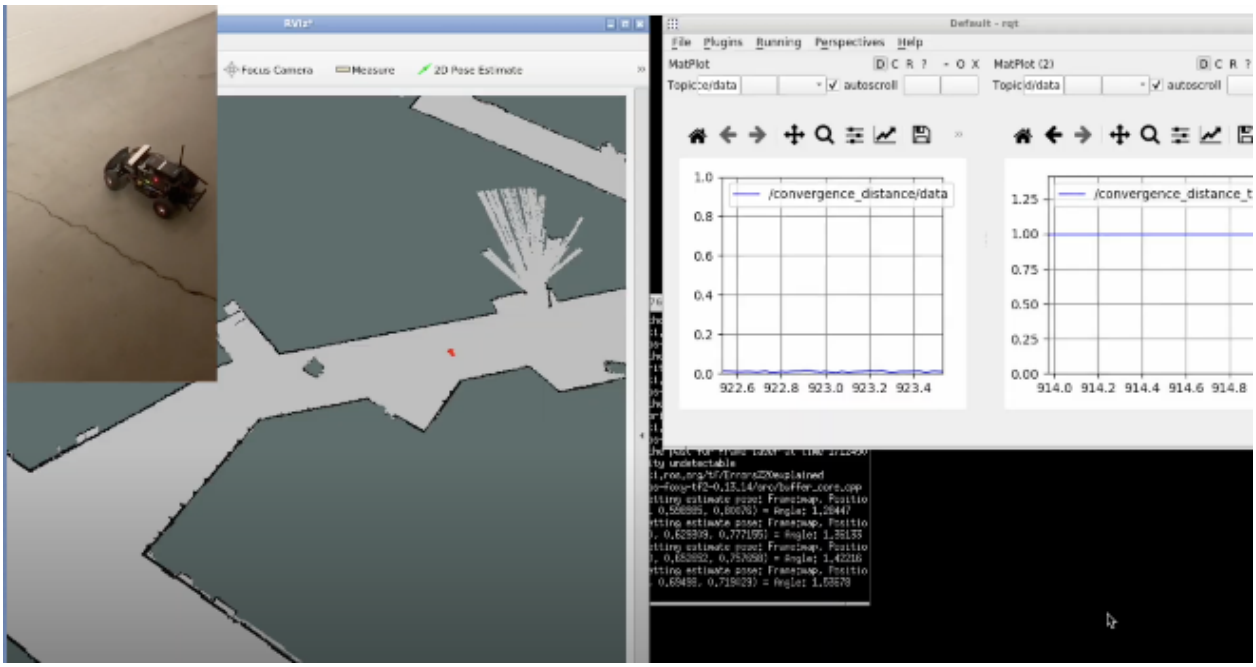


Figure 2: The convergence distance remains relatively steady while the racecar is driving around. We calculated this because it allows us to determine how quickly the particle filter can work, and if it is working correctly.

Conclusions (Lennie, Autumn)

In this lab, we achieved a working Monte Carlo localization algorithm that meets the objectives of this lab that our racecar can use to navigate its surroundings. We were careful in our coding to keep things modularized and easy to understand for other teammates to build upon, and we have robust comments and descriptive variable/function names that provide necessary context for people to understand the code.

One area we can improve upon is having a more streamlined debugging process, incorporating it in every step so bugs are caught early on. While we were much better about it this lab as compared to the previous labs, we can improve this with more stringent code review and having more robust performance evaluations that quantitatively evaluates our results, a point emphasized in the briefing feedback. While we didn't have much time today between receiving our briefing feedback and the report deadline to fully get the relevant evaluation metrics, we will improve on this in lab 6 by thinking about how we can measure our code *while* coding, and it will allow us to write more persuasive reports as well.

Given the increase in difficulty of this lab compared to previous ones, we made sure to start on this lab early so that we had enough time over the weekend to make sure it is working properly on the racecar. In addition, this lab was convenient to split up among our members as there are clear modules and it was great practice in working as a team. For lab 6 and the final project, we will continue to strive to achieve these goals.

Personal Reflections:

Nicholas: Over the course of this lab, I've learned the power of randomized algorithms in being able to quickly approximate a variety of quantities. I had learned of Monte Carlo algorithms from my time in MIT's algorithms classes, but never had the chance actually to implement them and see how powerful they are. Taking problems that are normally absurdly difficult to compute quickly and simplifying them down by approximating, discretizing, and randomizing is something I can really appreciate and something I plan to make much use of in industry and in research.

I've also learned something about how to communicate a message using presentations properly. I've already understood the power of diagrams and presenting data in a powerful way from my other classes, but not how to design slides that articulate a message straight to the point. I've now gained experience in designing concise, impactful slides, and I think that'll be quite helpful going forward in my programming career.

Finally, I found that coordinating with teammates ahead of time is of utmost importance when working on a project. One of the most efficient things I did this lab was to break complex tasks down for my teammates ahead of time so everyone could take on bite-sized chunks and work in parallel quickly. This prevented confusion, overlapping of tasks, and potential conflicts. Additionally, it allowed for efficient utilization of time and resources, as tasks could be evenly distributed and carried out simultaneously.

Overall, I found this lab challenging but quite rewarding. I'm looking forward to Lab 6, where we'll be able to put this localization into practice and design a complex navigation algorithm!

Autumn: During this lab, I gained a deeper appreciation for effective communication. It became evident that our collective communication fell short of what was necessary, making the lab considerably more challenging, particularly given its complexity. Moving forward, I am committed to being more proactive in my communication efforts. Reflecting on the past weekend, I realize that I could have communicated my availability better. It seems like a lesson I revisit regularly, but this week, I made an effort to outline deadlines while the team was present.

Initially, it seemed promising, but adhering to the schedule proved difficult amidst everyone's busy schedules.

James: I've learned numerous lessons from this lab including technical knowledge in Monte Carlo Localization, and the value of communication and planning. The lab taught me new information regarding localization, and implementing a motion and sensor model to map out particles via odometry data. Math applications were introduced to me through this lab such as updating the particle's position in accordance to odometry information and calculating the probability weights of the particles. The concept of localization and usage of noise to map out likelihood of robot position was another idea brought forth to me by this lab. Additionally, having effective communication helped in distributing the work and breaking up the assignments into a manageable task for the deadline.

Michael: Given the inherent difficulty of Lab 5, there were many valuable learning experiences. From the technical side, I initially found the key concepts of Monte Carlo Localization (MCL) to be quite abstract. Since my initial work focused on designing the motion model, I understood how it worked on its own, but I struggled to grasp how it functioned alongside the other components of MCL. I also found it challenging to visualize the effects of different motion model noise. However, working with the particle filter in simulation really helped me gain a deeper understanding of how the motion model and sensor model work together to create a powerful particle filter. It also became apparent that without appropriate noise in the motion model, the particles will struggle to track the robot, especially while making turns or other aggressive maneuvers.

This lab also reiterated the importance of active communication as a team. It has become clear that it can be difficult to schedule things last-minute when people's schedules are so busy. For this reason, I think it would be very valuable for everyone to communicate weekly availability to the team so we can best organize the work for the lab. With my busy baseball schedule, I have tried my best to let my teammates know which days I do not have games. This lab, I felt like I did a pretty good job optimizing my time that

Lennie: From this lab, I learned a lot about how the Monte Carlo Localization (MCL) works. I understood the general theory from lecture, but this lab really made me think about the details needed to implement it correctly. For example, one big one is that taking the simple mean of the particles is flawed when it follows a multi-modal distribution, so we needed to take the cluster mean instead. In addition to the technical portion, I also learned a lot about communication and knowing that I need to do a better job conveying my schedule as well as making sure that one teammate isn't doing too much for the lab.