Lab 5 Report: Monte Carlo Localization

Wells Crosby, Paarth Desai, Ian Gatlin
Samay Godika, John Posada

6.4200/16.405 Robotics: Science and Systems

April 15, 2023

1 Introduction

Author: Paarth Desai

The focus of this lab is to solve the problem of localization both within a simulation environment and with our robot. Localization is the determination of a robot's position and pose (orientation) within an environment, and is a very important aspect of any robotics system, specifically with autonomous vehicles. This importance is due to the fact that our robot must know its own position and pose to interact with its environment effectively, as for example, if we have an autonomous car which can perfectly plan paths, but cannot accurately estimate its position, the robot may believe it is on a viable path to its desired location, where in reality it could be driving into the sidewalk. This is an extreme example of course, but it demonstrates the importance of localization.

To solve this problem, we implement Monte Carlo Localization, or particle filter. On a high level, this algorithm generates a large number of hypothetical robot poses and positions based on initial conditions, and then assigns a probability to each pose based on sensor data, with the probability representing the likelihood that the pose is the actual position of the robot. By generating these probabilities, we can estimate the robot's ground truth(real world) pose by examining the distribution of the probabilities. The actual implementation of the algorithm is, of course, more complicated, and will be explained in detail in the remainder of this report.

Unfortunately, during the course of this lab, the car which we typically use had some issues with its hardware, and was given to the staff for repairs. This occurred at an untimely time for us, as it was just a day before we planned on testing our algorithm on the car, and thus we were unable to get extensive tests on our robot for this lab. With that said, we did have some time after the car was fixed to test in some capacity, however it was not an ideal situation.

2 Particle Filter Algorithm

We've found that the implementation of a particle filter pipeline for estimating the pose of our robot has allowed us to be robust to perturbations in the environment and unknown noise in sensor readings. With an initial pose estimate, the algorithm is able to converge to a pose that is very close to its true pose and can then be used in control.

2.1 Motion Model

Author: Wells Crosby

The goal for the motion model is to take our set of particles and the imperfect odometry reading from the car and find the projected forward particle poses using the odometry reading with noise added for each.
For each particle we will take the odometry reading from the car and add some Gaussian noise to each value, then project the particle forward with that new odometry reading to find the new particle pose. In this process our particles will spread out and explore different ways that the odometry readings could have been inaccurate, giving us confidence that at least some of them will be close to the true pose.
The calculation for getting the projected particle from the original particle and a noisy odometry reading is as follows:

$$(\text{new particle}) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot (\text{noisy odometry}) + (\text{original particle}) \tag{1}$$

where θ is acquired from each particle's pose.

2.2 Sensor Model

Author: Ian Gatlin

Now that we have all of these particles, we want to see which ones could be representing the car. To do this, we will use our sensor model. To determine which particles are probable to be where the car is, the sensor model compares LIDAR scans from the actual car and simulated LIDAR scans from the particles to compute the probability that a particle is where the car is.
This process is very computationally expensive, since you have to compute a probability for each beam for each simulated LIDAR scan. Each particle has 100 simulated LIDAR beams to compare, and there can be hundreds of particles. To solve this problem, we precomputed the possible probability outcomes of actual LIDAR beam range compared to a simulated LIDAR beam range in a 200x200 table. To be able to put the ranges in a table, we have to discretize the range values to integers. We did this by first converting distance values (ranges) from LIDAR scans to how many pixels long that distance would be in a map, and then rounding the pixel values to the nearest integer. We capped the distance a particle could be at 200 pixels, and accounted for this in the sensor model table with the $p_{\max}$ case (3) as defined in the lab handout. Once our motion model probability formula and our implementation was built to work with discrete values on a limited range, we could compute the probabilities for each entry in the table before runtime.
When the sensor model is running, it generates simulated LIDAR from the particles by ray casting. It then rounds the simulated data to the nearest pixel integer. Next, it looks up the probability in the precomputed table that the LIDAR beam of the simulated scan matches the actual LIDAR beam. Next, the product of all these probabilities for each particle is computed and "squashed" which means raising it to the power of $\frac{1}{2.2}$. The sensor model returns a vector of these probabilities for every particle to the particle filter, where all of these modules of localization are integrated together.

2.3 Pose Estimation

Author: Wells Crosby

Given the set of particles we have found and refined with our motion and sensor models, we need to publish our best guess of the car's actual pose. To do this we are going to take a weighted average of the particle's x, y, and θ values. We will use the probabilities assigned to each particle from the sensor

model as the weights.
For the x and y position of our estimated pose we will simply use the weighted average of the x and y values of the particles. For getting the theta of our estimated pose we will need a slightly more complicated approach. The technique we will use is a weighted circular mean.
This works by first translating the theta value of each particle into the associated x and y values of that angle's position on the unit circle. We get these x and y values with $\cos(\theta)$ and $\sin(\theta)$ respectively. With these x and y values of each particle, we will take a weighted average of each of these, then feed those averages into atan2 to find the angle created by them. This gives us our weighted circular mean of the θ values, so we can now return a pose with our found average x, y, and θ values.
By using a weighted averaging for each value, we get a better representation of where the car likely is and become more resistant to multimodal particle distributions.

2.4 Effect of Parameters

Author: John Posada

Within our implementation, we are also able to modify parameters based on the needs of the robot for different applications, such as the need to have very quick pose estimate updates, or the need to have more confidence in our pose estimates.
One parameter is the number of particles that are used at every update. Nominally, we utilize 200 particles as a baseline in order to provide reasonably confident estimates of pose while being able to maintain an adequately high update rate. By increasing the number of particles, we are able to evaluate a finer distribution of possible poses near the robot at every update and so produce a more precise pose estimate at the cost of computation time.
Within the motion model, the main parameter that affects the performance of the localization algorithm is the magnitude of the noise applied to the particles during the odometry callback. By applying more noise to the position and orientation of the particles, the algorithm covers a larger range of space around the robot. However, this means that the overall particles are more sparse, and so the pose estimate returned by the algorithm is likely to be less precise. It is critical to the localization problem that some noise be added in order to provide confidence in the pose estimate and be robust

to any undetected perturbations in the environment or sensors, but not so much that the pose estimate itself is noisy. A way to balance the increase of noise with estimate precision is to increase the number of particles, but this increases computation time, as mentioned above.
Within the sensor model, a few parameters play a role in determining the success of the particle filter. Increasing the number of simulated raycasts used to evaluate the likelihood of each particle allows us to have a finer range of simulated scan data, which increases our confidence in the probabilities calculated for each particle, but comes at the cost of computation time. Additionally, everytime the particles are resampled based on their probabilities, a certain fraction of them are deleted and replaced with those that were sampled. By decreasing the fraction of particles that are sampled, the algorithm, on average, keeps only high probability particles, and so propagation at each time step retains a good estimate. However, if the particles themselves are spatially sparse, then the system becomes sensitive to slight perturbations in sensor readings.
Overall, there is a tradeoff to each parameter within the particle filter algorithm. These can be chosen experimentally with trial-and-error, to ensure the algorithm is neither too reliant on odometry data nor too reliant on LIDAR data. Onboard the robot, it is critical that the parameters are limited so as to keep the computation time at each update within a reasonable range. Otherwise, the robot will not be able to locate itself quickly enough to issue proper controller commands, which can hinder performance and be a safety issue.

3 Experimental Evaluation

Author: Paarth Desai

In the simulation environment, we used the provided Gradescope tests as our primary metric of success. These tests, and the figures provided by them, allowed us to visualize exactly how our pose estimate compared to the ground truth pose, within the simulation environment of course. We were also able to compare our solution with the staff's solution, providing us with a standard which we knew to be attainable.
At first, our solution was quite poor, as can be seen in the solution below. This is the "more noise" test, and our estimate deviated quite a lot from the ground truth pose.

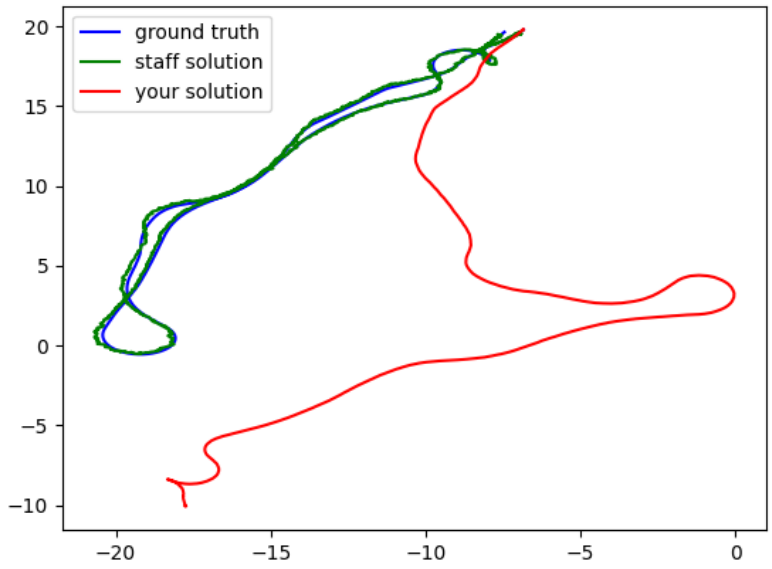

Figure 1: First autograder attempt.

In this test, we found that our issue was that we did not have enough noise applied to our particle filter, as our particle filter was not modifying the odometry data enough, leading to our generated particles being too similar to our odometry data, which we know is not very accurate. This leads to our particle filter not generating enough different positions, and in this case, this narrow search space did not include the actual solution, leading to a poor pose estimate.

Given this, we knew that our model needed improving, so we continued to iterate, specifically adding more noise, and were able to achieve a solution which had an average deviation from ground truth of 0.3915 meters, shown in the plot below.

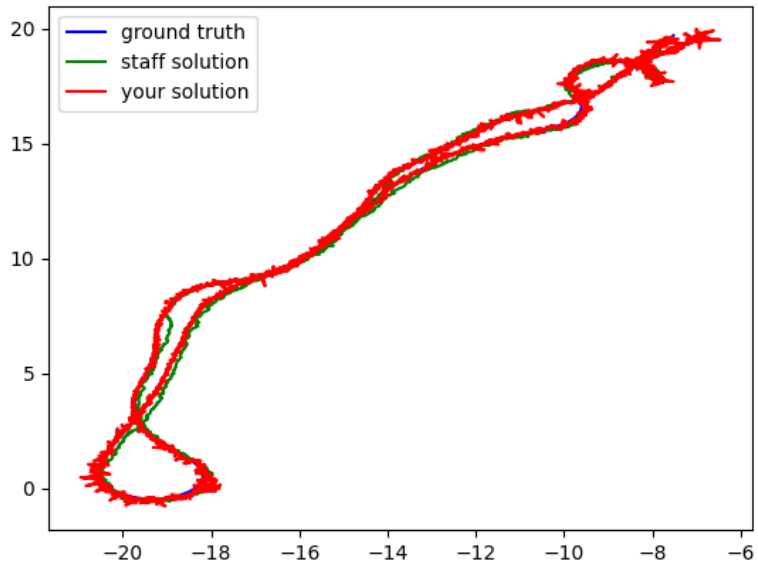

Figure 2: Second autograder attempt.

This solution was obviously significantly better than our previous one, however it is clear to see that the particle filter is not returning a "smooth" pose estimate, and is instead fluctuating significantly. To improve it, we continued to guess and check different noise values, as well as change our initial conditions, which was discussed in detail in part (wherever). This led to the solution which can be seen below.

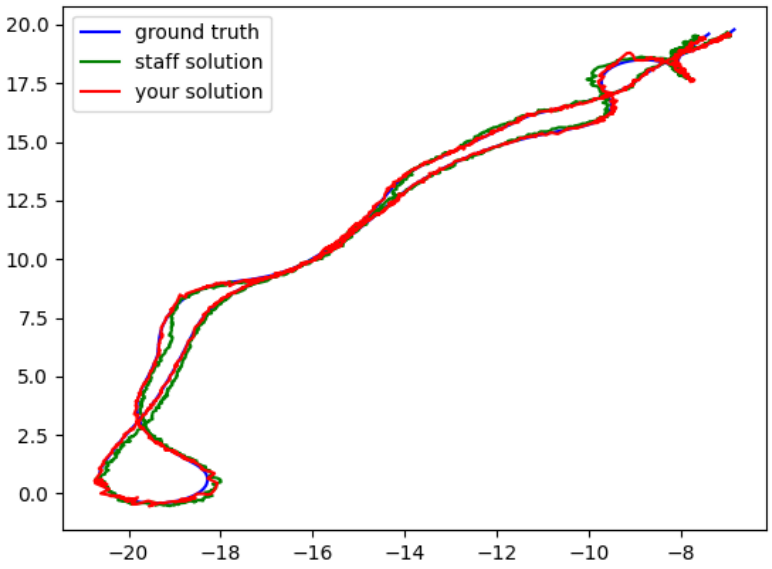

Figure 3: Third autograder attempt.

This solution has an average deviation of 0.1677 meters from the ground truth, and this was good enough to beat the staff's solution. Naturally, we were very pleased with this result, so we were happy to now move on to testing our implementation on our robot in the real-world environment.
The figure below shows the results from the three main runs which we did on the autograder, each after significant changes to our algorithm. The three different bars per run represent tests with no odometry noise, some odometry noise, and more odometry noise, as created by the course staff. As can be seen, our algorithm performed significantly better on the final run.

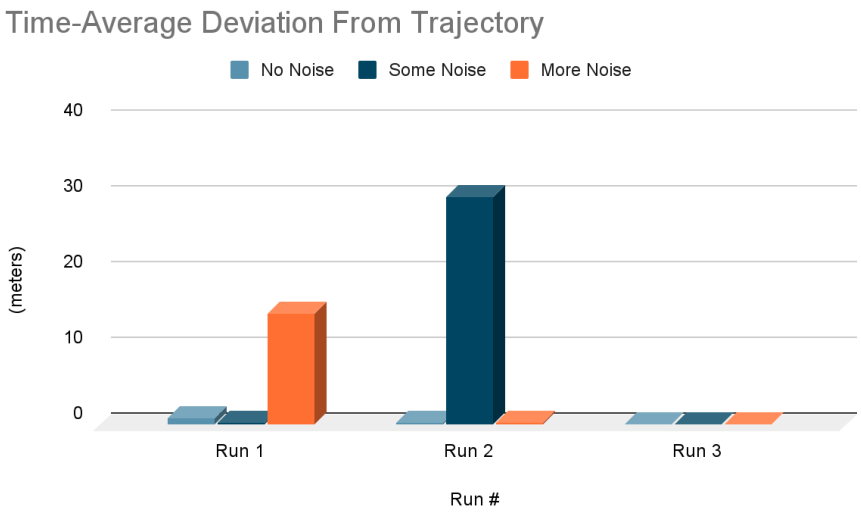

Figure 4: Bar chart of average deviation from given trajectory per autograder attempt.

4 Conclusion

Author: Samay Godika

In conclusion, our team used the Monte Carlo method to successfully enable our robot to determine its location through its surrounding environment. As described above, there were two key parts of this lab's implementation: the Motion Model and the Sensor Model. After generating random particles, all of which depict the robot's possible position, the motion model moves these particles in their respective directions based on odometry data given. The sensor model uses these locations of the particles to see which ones correspond with the information the robot itself is detecting. Those particles are given a higher weight, and as this process continues, the robot is able to have clarity of its location. Essentially, our program sends out hundreds of possible robot locations and uses the LIDAR data that the robot receives to weed out the particles that don't match with the LIDAR. To ensure that our implementation is robust to faulty odometry data, we introduce noise to the data. The robot then calculates the particles with respect to this randomized odometry. Thus, if incorrect odometry was provided, this would get handled

and shifted by the noise, and our robot would still be able to localize itself correctly! Nevertheless, there are still places we could improve. In the past week, our car's router had a serious issue in which it began smoking up. Due to this, our team was unable to test extensively with the car. Although we were able to get a functioning localization implementation running on the car, we would like to work on our map transform more and make it better for the future. Additionally, our robot's LIDAR has been detecting points immediately on the top of the vehicle at all times, even when there is clearly nothing there. To get around this, we have had to change our code to account for this special case and take note not to include it in our calculations. In the future, we are planning to generalize our code further such that it would be able to handle these outlier cases from the LIDAR sensor.
On the whole, the robot is able to localize itself very well. Testing it in the simulator, we were able to see how it functions with all ranges of noise. Running our algorithm on our real robot was also a big learning experience, and we took careful note of how to evaluate this, as described in the sections above. Of course, there is scope for improvement; we plan to work on that in the next few weeks to have our robot fully autonomous and prepared for the final challenge!

5 Lessons Learned

5.1 Wells Crosby

In this lab the main communication/collaborations lesson was in how we manage our time. We were able to split the tasks up well so that we could get each part of the lab done quickly and effectively, but we ran into issues with being able to meet all together. In previous labs we have tried to all meet at the same time so that we can work together as much as possible but this created issues in this lab. Finding a time when we are all able to meet is hard when we all are busy with other things in our lives. This became a significant issue when we were only able to all meet the day before the presentation was due. When we met, we found that the car had been broken and so we would not be able to do any physical testing that day. If we had not worried too much about all of us being there we would have been able to test earlier and more frequently. In the future we are going to de-emphasize needing to all be there when we meet so that we can have more frequent

meetings. On the technical front, the major lessons for me were in creating performant code. This lab required iterating through many particles many times, so optimizing the iteration is key for good performance, especially since python is so slow to iterate. I learned a good amount about numpy vectorization in order to make these iterations much faster.

5.2 Paarth Desai

From a technical standpoint, I learned a lot in this lab, as the problem of localization was one that I had never considered when thinking about robotic systems, yet it is incredibly crucial for any robot. It was very interesting to see and implement the Monte Carlo algorithm for localization, as this was not necessarily an intuitive solution in my eyes, but it obviously has very good results, and thinking about why that has allowed me to learn more about general methodology of problem solving in robotics. From a group perspective, there were a few big takeaways for me, as we had some issues with working on the robot due to hardware issues. For one, it has become clear that we need to prioritize finishing tasks without the robot even more, as we cannot assume that any given task will be easy with the hardware, as unexpected issues can always occur, regardless of how simple an implementation may seem. Another huge takeaway is something Dylan mentioned at our briefing, which is that we should consider working in smaller groups from time to time. Currently, we typically only work in our group of 5, which is a great environment, and is certainly very productive, however sometimes, it is hard to find time where the 5 of us are all free earlier in the week, and in these cases it would be good for a smaller subset of our team to meet and get some work done, just so that we give ourselves more time before the deadline.

5.3 Ian Gatlin

From a technical perspective, I learned a lot about how to represent mathematical ideas in python code using regular control flow and numpy. When working with more complex data structures such as matrices, regular python operations are not sufficient and you have to use numpy. There is a lot of documentation for this library and it was good practicing reading documentation on a variety of different numpy methods. I think for this lab we also learned how important it is to get code on the robot early and so we can

do simulation and physical optimization of the code in parallel, rather than perfecting it in the simulation first. This will allow us to finish labs faster. From a communication perspective I think we did a good job communicating where we were having problems with getting our parts of the lab done and collaborating to fill those gaps. I think a big thing we need to work on though is working separately. It is nice to only worry about the work when we are meeting together so it isn't on my mind when I'm trying to work on other work, but it puts us in time crunches. This is something we can work on for the next lab and final challenge.

5.4 Samay Godika

I have learned a lot from a technical standpoint over the past few weeks. From different types of object recognition algorithms to the math behind the Monte Carlo transformations, it has been interesting to first read about and then actually implement solutions. Furthermore, working as a team has gone smoothly, and I feel like our team has a good dynamic. At the beginning of the lab, our team would divide the lab into sections, and discuss each member's interests. Based on one's own specific skill sets and interests, each person got to work on something they were excited about. I really like that approach, as I feel that working in parallel has made us efficient. At the same time, we help each other if someone is stuck on their respective sections. I have learned a lot about how communicating when I'm stuck is very important, especially in a group project where everyone is ready to help. This prevents us from being stuck on something for too long and instead uses the resource of a fresh pair of eyes to look at the issue! Additionally, I learnt how it is important to take notes when someone is talking. While speech is an important form of communication, it is easy for key points to be forgotten. Writing it down could be useful to refer to days or even a week later when we start working on that particular thing. Additionally, those notes could be shown to others who weren't present when the words were spoken, and then everyone would be well informed! Thus, on the whole, I have learnt a lot from a technical and communication standpoint, and I look forward to the next few weeks of further work!

5.5 John Posada

One of the key lessons from this lab was how important it is to split up objectives across the team, and it's something I think we did very well as the lab progressed. This allowed each of us to optimize our portions of the code and debug quickly when any problems arose. Additionally, having a strong mathematical foundation from the first part of the lab allowed us to get a running start into the bulk of this lab, so a lesson there is that it pays to spend some time preparing before actually getting into the thick of it. Our team faced a lot of setbacks and problems during the lab, whether it was hardware issues, not having the robot to test on, or just not knowing how to fix bugs we were facing. In every case, though, I found that our team was really able to keep pushing through and strive for reachable objectives everytime we met that made sure that even if we didn't finish everything we wanted to, we reached a point that we can all be happy with. In general, it seems that even if the project is not fully realized as expected from the beginning, progress made and technical concepts mastered are still valuable things we can take from each of the labs. On a final note, I really learned to love numpy a lot more than I did before. Vectorizing our large operations really improved computation time and even though we struggled a lot in understanding the bugs we faced using numpy, it paid off to optimize really large operations that had to run several times a second.