Lab 5 Report: Localization

Team #13

Kaleb Blake
Lucian Covarrubias
Seth Fine
Jesse George
Ivory Tang

6.141 Robotics, Science, and Systems - Spring 2022

April 2, 2022

Editor: Kaleb Blake

1 Introduction

By: Seth Fine

Throughout this lab, we explored the open problem of localization in robotics. We learned different versions of the localization problem under different constraints of a known or unknown global map. In this lab, we worked with a known global map of the Stata basement where we would eventually test our robot. Under these constraints, we were tasked with implementing the Monte Carlo Localization algorithm. After implementing the algorithm and passing unit tests, we first tried our code in simulation. This allowed us to fine tune parameters that modelled noise using Gaussian distributions. Next, once we were confident in our implementation used in the simulations, we used our algorithm for real time localization on the robot within a known map. Throughout our implementation and testing, we needed to optimize for runtime efficiency which forced us to utilize and explore fast python packages, discretization, and safe multi-threading in order to create an effective solution.

2 Technical Approach

2.1 Overview

By: Lucian Covarrubias

In order to implement a localization algorithm, there are multiple moving parts.

Firstly, a motion model is necessary to simulate the change in position of the vehicle as it moves in real life. This motion model must incorporate noise, as odometry in vehicles is inherenty noisy. The motion model is used to update the locations of every simulated particle used in the localization algorithm as the car moves.

Secondly, a sensor model is necessary to represent the confidence of recieving a certain sensor reading given the true world around us. This model must be tailored to express the possibility of sensor errors in multiple ways, which will be discussed in detail later.

Finally, a particle filtering mechanism must be implemented which uses the motion model to move particles such that they follow the general motion of the car, and uses the sensor model to remove points which have very unlikely poses given the true pose of the vehicle.

2.2 Motion Model

By: Seth Fine

The motion model is called when the particle filter receives a new odometry message from the robot. This happens approximately 50 times per second. Hence, this sub-module is called very often meaning it must execute its computations rapidly in order to provide current and relevant information for the particle filter. To accomplish this, our implementation of the motion model makes heavy use of the numpy python package. An additional consideration for the motion model implementation is that the odometry data is noisy. This means how far the robot reports to have travelled and how far the robot reports to have turned might not be correct. Therefore, we add randomness to the received odometry data through Gaussian distributions to model that the true current state of the robot is likely a small delta away from the position calculated using the received odometry data. Figure 1 demonstrates this property. We are then able to return to the particle filter the current positions.

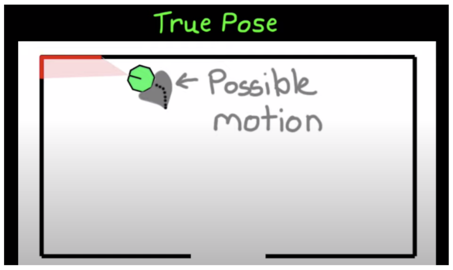


Figure 1: Due to noise in the odometry data, the true position of the robot is likely within the grey region. The motion model uses Gaussian distributions centered around the odometry data to model this noise.

2.3 Sensor Model

By: Ivory Tang

The purpose of the sensor model is to assign likelihood weights to each particle. In the next iteration, particles with higher likelihood weights will be more likely to be resampled and those with low weights will not. In short, we receive sensor readings from the lidar scan, and we must determine the likelihood of each particle existing. Each particle is given a fake "scan" via ray casting. We then compare the raycasted scan with our real scan to get probabilities for each measurement.

To do this, we provide a breakdown of the likelihoods into four components as follows:

1. Probability of detecting a known obstacle in the map
$$p_{hit}(z_k^{(i)}|x_k, m) = \begin{cases} \eta * \frac{1}{\sqrt{2*\pi*\sigma^2}} * exp(-\frac{(z_k^{(i)}-d)^2}{2*\sigma^2}) \text{ if } 0 \leq z_k < z_{max} \\ 0 \text{ otherwise} \end{cases}$$
2. Probability of a short measurement. Maybe due to internal lidar reflections (scratches or oils on the surface), hitting parts of the vehicle itself, or other unknown obstacles (people, cats, etc).
$$p_{short}(z_k^{(i)}|x_k, m) = \frac{2}{d} * \begin{cases} 1 - \frac{z_k^{(i)}}{d} \text{ if } 0 \leq (z_k^{(i)} < d \text{ and } d \neq 0 \\ 0 \text{ otherwise} \end{cases}$$
3. Probability of a very large (aka missed) measurement. Usually due to lidar beams that hit an object with strange reflective properties and did not bounce back to the sensor
$$p_{max}(z_k^{(i)}|x_k, m) = \begin{cases} \frac{1}{\epsilon} \text{ if } z_{max} - \epsilon \leq z_k^{(i)} \leq z_{max} \\ 0 \text{ otherwise} \end{cases}$$

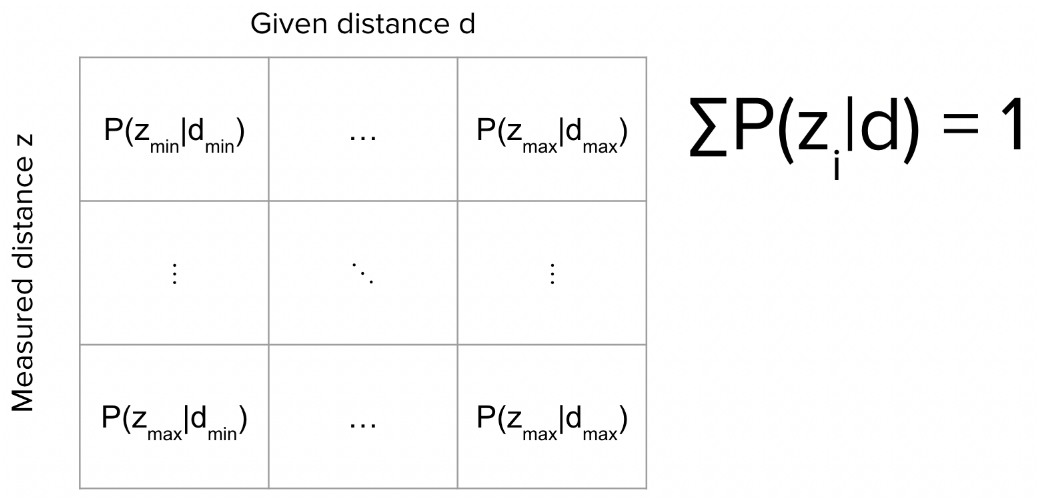
4. Probability of a completely random measurement. Just in case of an asteroid strike.
$p_{rand}(z_k^{(i)}|x_k,m) = \begin{cases} \frac{1}{z_{max}} \text{ if } 0 \le z_k^{(i)} \le z_{max} \\ 0 \text{ otherwise} \end{cases}$

The equation we solve for then depends on a weighting of these four probabilities that contribute to the total probability.

$$p(z_k^{(i)}|x_k,m)$$
$$= \alpha_{hit} * p_{hit}(z_k^{(i)}|x_k,m) + \alpha_{short} * p_{short}(z_k^{(i)}|x_k,m) + \alpha_{max} * p_{max}(z_k^{(i)}|x_k,m)$$
$$+ \alpha_{rand} * p_{rand}(z_k^{(i)}|x_k,m)$$
$$= 0.74 * p_{hit}(z_k^{(i)}|x_k,m) + 0.07 * p_{short}(z_k^{(i)}|x_k,m) + 0.07 * p_{max}(z_k^{(i)}|x_k,m) + 0.12$$
$$* p_{rand}(z_k^{(i)}|x_k,m)$$

$$\alpha_{hit} + \alpha_{short} + \alpha_{max} + \alpha_{rand} = 1$$

Once we compute the model, we need to normalize each row. This ensures that we have a probability distribution which represents the probabilities of getting a variety of measured values given a set real value.


Given distance d

Measured distance z

$P(z_{min} \mid d_{min})$	…	$P(z_{max} \mid d_{max})$
⋮	⋱	⋮
$P(z_{max} \mid d_{min})$	…	$P(z_{max} \mid d_{max})$

$$\sum P(z_i|d) = 1$$

Figure 2: Our final pre-computed table will be formatted as above.

Many of these calculations are time-intensive, so we perform a discretization of the range values to speed up our model. In essence, this grid of discretized values will have inputs of the actual distance d and the measured distance z_k^i and will store a probability. Because this grid can be used even after the car moves and changes location, it is much more efficient to just access the value we need in this pre-computed grid instead of calculating the probability each time. The second reason for the discretization is so we can numerically normalize the $p(z_k^i|x_k, m)$ function.

2.4 Particle Filter

By: Lucian Covarrubias

Particle filtering is the process of taking the simulated possible point locations of the racecar, and using the sensor model to figure out which possible locations are the most likely. Given the likelihoods of our possible positions, we can filter out points with low probability and keep ones with higher probability in order to get a more accurate estimate of the car's true position. Figure 3 depicts an example starting initialization for possible locations in red, with the goal to filter out points which don't match closely to the true car location in pink.

Particle filtering is very important for localization because it allows us to collect the simulated points around the most likely locations of the vehicle. Without particle filtering, noise from the motion model would make the simulated points diverge very quickly and the estimated location of the vehicle would similarly diverge from the truth. Adding in particle filtering allows noise to be used in the motion model while still keeping points near the true location of the vehicle.

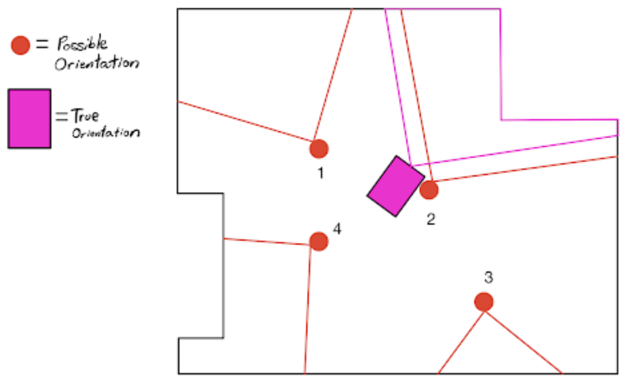


Figure 3: An initial set of potential poses for the vehicle (red) along with the true pose of the vehicle (pink). Fake scans are ray-casted from each potential point for comparison purposes.

The particle filter's functionality is relatively simple. Given the current orientations of all simulated points, generate fake lidar scans for each point and compare the scan to our true scan. Figure 3 demonstrates the initial problem, along with fake scans from each point. Using the sensor model, the program calculates the similarity between each potential scan and the real scan. In Figure 4, we can see each scan, with the scan from point (2) matching the real scan in pink much more than any other.

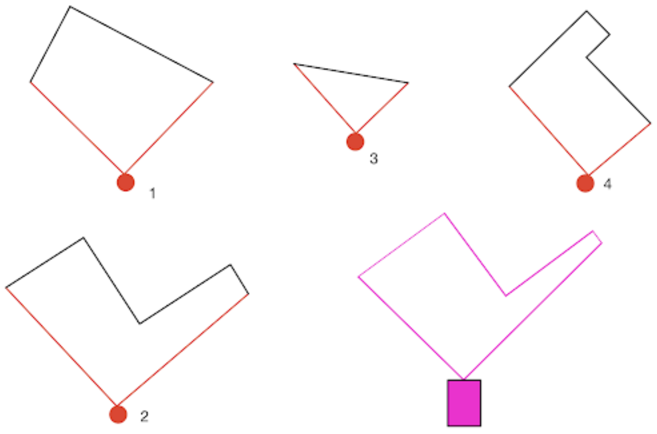


Figure 4: Each scan separated for clarity. Every scan will be compared to the true scan (pink) to determine the likeliness of each point.

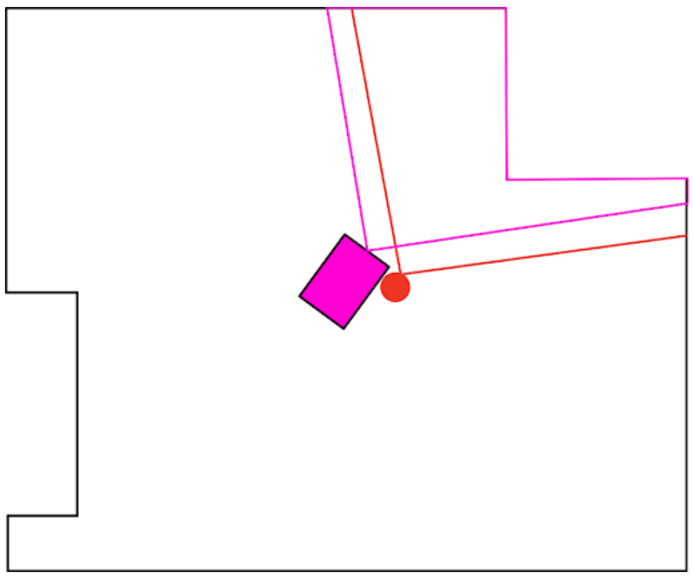


Figure 5: A potential final set of points after filtering.

Once all point probabilities are calculated, the particle filter randomly re-samples another set of points from a normalized distribution created from those probabilities Figure 5. As we can see in Figure 5, multiple points may be sampled at the same location if it is highly likely. Points with higher probability are going to be re-sampled more and points with low probabilities may not be re-sampled.

This effectively removes unlikely points and allows the model to focus more on areas that are close to the car's true position.

3 Simulation and Experimental Evaluation

By: Kaleb Blake

We tested our implementation using the racecar simulation in Rviz. We conducted three types of tests on our particle filter: motion model on and sensor model off with deterministic set to True, motion model on and sensor model off with deterministic set to False, and both motion model and sensor model on with deterministic set to False. While running each test we made sure to visualize our set of particles and the estimated odometry of the racecar calculated based on the current state of the particles. We had the cone parking algorithm running, so that we could see the particles update based on where the cone is and how the car navigates to it. We evaluated each test qualitatively, because we knew what should happen for each test from our understanding of the particle filter and our implementation of it. In the motion model on and sensor model off with deterministic set to True, the particles all performed the same motion as the racecar, but since they begin with slightly different orientations and positions as the racecar, they all don't end up at the cone like the racecar does. Figure 6 shows these results.

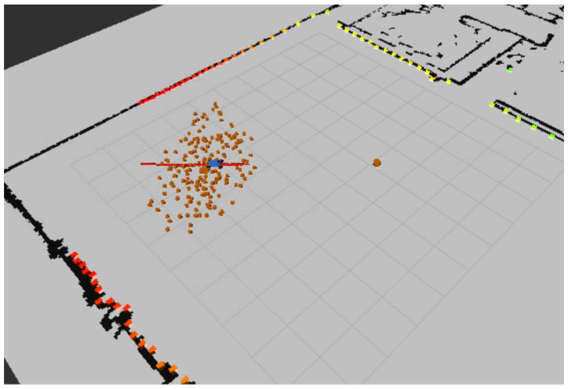


Figure 6: Racecar simulator in Rviz demonstrating the motion model on and sensor model off with deterministic set to True. The particles all performed the same motion as the racecar, but they all don't end up at the cone like the racecar does.

Since the sensor model was off there was no resampling of the particles based on which particles were most likely to be correct. In Figure 7 the motion model on and sensor model off with deterministic set to False test is shown. It had slightly different results, as expected. Instead of the particles all performing the same odometry commands as the racecar they performed commands with noise added, as deterministic was set to False. This could especially be seen when the racecar was stationary and the particles moved in an erratic fashion about their original stopping point.

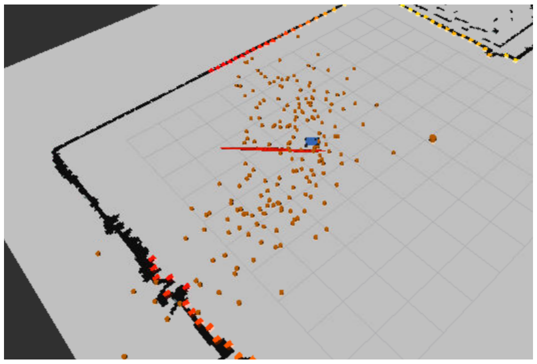


Figure 7: Racecar simulator in Rviz demonstrating the motion model on and sensor model off with deterministic set to False. The particles did not perform the same motion as the racecar, and end up farther from the racecar.

With both motion model and sensor model on and deterministic set to False the full particle filter capabilities could be observed. The particles are always near the cars true position and the estimated odometry from the particles follow the racecar trajectory closely. The particles are the orange clump under the racecar seen in Figure 8. This is because unlike in the previous test the particles are resampled and the particles most likely to be correct are kept, so all the particles end up close to the racecar.

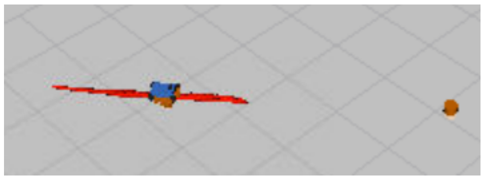


Figure 8: Racecar simulator in Rviz demonstrating both motion model and sensor model on and deterministic set to False. The particles very closely followed the racecar, and can be seen as the orange clump under the racecar.

4 Conclusion

By: Lucian Covarrubias

In this lab, we implemented a MCL algorithm to allow the robot to identify its location in a known map. This involved modelling the motion of our vehicle via non-deterministic methods, modelling our sensor with randomness that represents the possibilities of failure in our system, and implementing particle filtering that takes in our simulated particles and uses our sensor model to find and keep the most likely points. The operation time of our algorithm was vastly improved by using techniques such as memoization and vectorized computations through the Numpy library, allowing real time updates of every particle location.

5 Lessons Learned

Kaleb: This lab was by far the most challenging of all the labs, and that allowed me to learn a lot in each of the technical, communication, and collaboration aspects of this course. I thought the technical ideas of this lab were very interesting. It is quite a difficult problem to estimate a robot position with just a LIDAR, but the way we were shown to do it was very creative and impressive to see. I enjoyed learning about how to incorporate uncertainty and noise through probability. I also thought it was very difficult to successfully implement multiple models together into a complete algorithm, but it was very satisfying once it worked. It was helpful for debugging to test the models individually on their own as well as individually in the whole algorithm. I think I was able to better communicate my progress on the code with the team after I finished working and problems that I solved and problems that still need to be solved. I could work better at understanding where other people leave off and what problems they have encountered. Since this lab was so challenging, collaboration was so important. Most of my progress was definitely made when I was working with other members of the team. Not only is collaboration with my team important, but I found it to be very helpful to get help from TAs at office hours. I learned to know when to ask for help, though I was late to ask for help this lab.

Lucian: Through this lab, I learned the intricacies of working on a limited system which doesn't have infinite computation ability, along with managing a system that needs to work in real time. Precomputation and memoization combined with discretizing our continuous spectrum of data allowed us to vastly increase the speed of our algorithm, which is a very valuable lesson for the future. The importance of threading for preventing race conditions was very interesting to me, as I've never had to work with asynchronous systems in Python before. As far as team organization, I've learned a lot about how setting concrete expectations with each team member is very important for making sure that everyone knows what they need to do. This makes sure that certain people don't take on too much work and that the whole process is more evenly divided.

Seth: This lab had us use new techniques such as discretization and threading. It was cool to see the effects of these different techniques on both making the code easier to implement in the case of discretization and also the efficiency effects of threading. To this end, some of the newer aspects of this lab led our group to struggle through some tough to comprehend errors which forced us to be systematic in how we debugged as our algorithm and the simulation environment have many different interacting components. One thing that could help our team in the future is beginning labs earlier in the cycle allowing us more time to run ideas through each other and troubleshoot incoherent errors with TAs.

Jesse: The main parts of this lab that I found the most interesting were the methods I have never used before, such as threading and the use of a particle filter with simulated noise. For the particle filter, I had known of the concept before, but it was my first time implementing it in actual code. The corresponding result was actually very impressive with only 200 particles, but it was very computationally expensive to run. I now feel like I have a deeper understanding of why it tends to be avoided in many mobile robots with limited computational capacity despite it's good performance. Threading, however, was an entirely new concept to me, so it was interesting to learn about how to optimize code by having certain operations running simultaneously while avoiding data races. It feels like a genuinely useful skill for me to use in the future for other projects, especially for tasks involving expensive computations that could be parallelized to save time or CPU usage.

Ivory: The most interesting aspect of this lab to me is seeing how Monte Carlo localization helped our racecar navigate more complex surroundings and thus increase its autonomy. Whereas before, we were simply following a wall or line, now we can navigate a complex environment such as the stata basement due to our localization algorithm working. There was significantly more troubleshooting in this lab than previous labs, mostly involving the particle filter where we were trying to get the motion model and sensor model, which were working individually, to work together in filtering just the desired particles. Other important lessons this lab taught were the importance of precomputing values so that our algorithm would run fast enough when we add lots and lots of particles as well as threading which is particularly important when you have multiple models working jointly. Our team realized the importance of starting these labs earlier, ie getting more progress done before spring break, so that we would have ample time to debug near the due date, and we hope to apply this lesson to the remaining assignments. Lastly, we found that visualization tools in Rviz were especially helpful for debugging in this lab and helped us to catch a few of our errors (for instance, when we turned off our sensor model to see if our motion model was doing the right thing). Being able to see how the particles acted in simulation helped us make the necessary changes to our model before we tested it out on the real racecar.