Lab 5 Report: Monte Carlo Localization

Team 7

Bradley Albright
Ethan Ardolino
Jiahai Feng
Tahmid Jamal
Tian Lin

6.141

April 5, 2022

1 Introduction

To navigate through an area with a known mapping, a robot needs to have an accurate estimate of its whereabouts. This is the basic premise of localization – understanding the position and rotation, known as the pose, of a robot in relation to its surroundings as it traverses through an environment. The Monte Carlo Localization algorithm (MCL) is an example of one approach to solving this problem. From a rough estimate of an initial position, the algorithm tracks the movement of the vehicle by keeping internal measurements of the distance traveled and the radians turned. To combat noisy data collection, MCL maintains a list of possible positions the robot may be in and periodically filters these points. A final pose is estimated by combining these clouds of possible locations.

A less trivial problem is localizing a robot when there is not a given reference map, known as simultaneous localization and mapping (SLAM). This is generally the case in real-world applications of autonomous robots. To tackle this challenge, Google Cartographer was used in order to create a map of an unknown area while also keeping track of where the vehicle is located within that area.

The following algorithms will be used for achieving higher level objectives such as navigating to specific places on a map.

2 Technical Approach

2.1 Motion Model (Bradley)

In order to effectively make use of the odometry data to track our robots location, we need to understand the effect of noise on the odometry so that we can best negate it. Instead of storing only our best guess of our robot's location the algorithm stores a plethora of guesses of possible locations that the robot may be positioned. How we update this list of possible robot locations to include the noisy odometry data is called the motion model.

In each update step the motion model is given the set of possible robot poses, locations and directions, and the odometry data. The algorithm's first task is to convert the odometry data from local measurements relative to the old pose to global movement. We do this by rotating the local pose to the global pose since it is measured from a proposed robot location. Now that the pose is rotated into the same frame as the global axis we can add the change in x and y to the old pose in order to get our new global pose.
To add noise to the filter we use the following model. There are three different

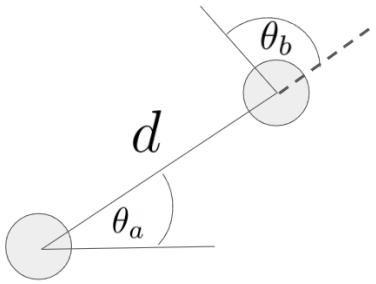


Figure 1: Model of the various angles and distances between poses

measurements that will have noise:

- θ_a: The angle between the initial heading and the vector between point a and point b.
- d: Distance to the next point.
- θ_b: The angle between the vector from point a to point b and the final heading.

This leaves us with three parameters to tune. By using a gaussian distribution we can distribute points with a highest density closest to the original odometry

measurements. We use the following equations for this purpose:

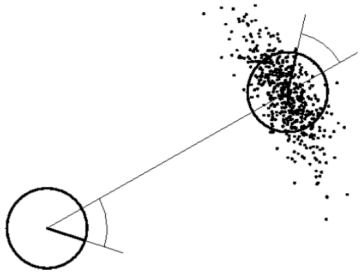
$$\hat{\theta_a} = \theta_a - sample(a1 * \theta_a + a2 * d) \tag{1}$$

$$\hat{d} = d - sample(a3 * d + a4 * (\theta_a + \theta_b)) \tag{2}$$

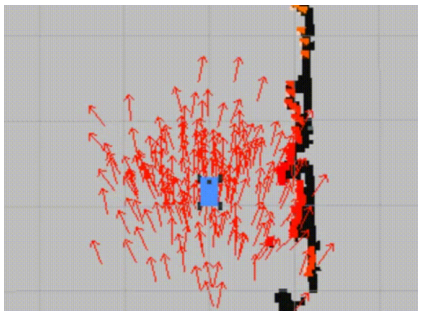
$$\hat{\theta_b} = \theta_b - sample(a1 * \theta_b + d) \tag{3}$$

$$\tag{4}$$

Where the sample function multiplies the arguments by a random value picked from a Gaussian distribution centered at zero and $a1$ through $a4$ are noise parameters. An example of the distribution of points based on an odometry measurement is as follows.



(a) Distribution of possible robot locations based on the initial odometry measurements



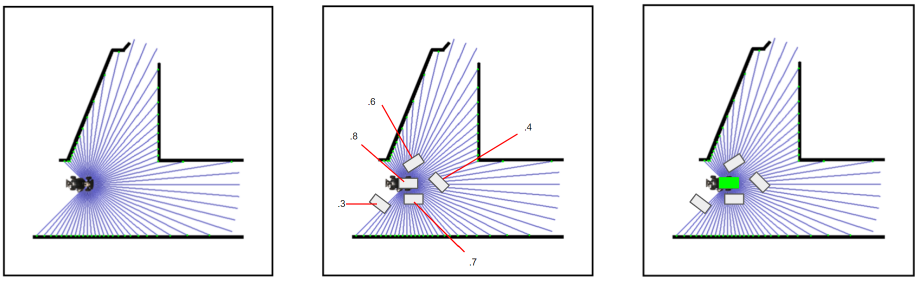
(b) Arrows each represent a possible pose for the robot. The poses gradually spread out over time as the noise compounds without filtering

Figure 2: Example robot location distribution (right) vs robot pose distribution in simulation

To cover any other extra noise we also added random perturbations to the sampled poses in both the x and y directions. Using this added perturbations we hope to capture all the possible locations that the robot could end up in. The next couple steps will act to filter these possible locations based on sensing of the external environment from the LiDAR.

2.2 Sensor Model (Tahmid)

With the motion model providing a plethora of various poses, we would like to identify the poses that are closest to the ground-truth pose of the robot. We use a sensor model that relays how well a simulated LIDAR scan about a pose in a known map setting matches the LIDAR scan read from the car. The models assigns probabilities to the poses based on how well they match sensor data, and we estimate the true pose from the highest likelihood poses.
To evaluate the probability of a certain pose given its simulated LIDAR scans from raycasting, we will develop a probability distribution to account for various



(a) LIDAR data is treated as ground-truth

(b) Sensor model assigns a probabilities.

(c) High likelihood poses are chosen.

Figure 3: Overview of how sensor model is used to identify pose

scenarios. Say that $d^{(i)}$ is the ground truth distance of the i-th range measurement of a LIDAR scan, and $z^{(i)}$ is the i-th simulated range measurement for pose x in map m. Thus, we can determine the likelihood of any pose by evaluating the following

$$p(z|x,m) = \prod_{i=0}^{n} p(z^{(i)}|x,m)$$

To evaluate $p(z^{(i)}|x,m)$ we consider multiple scenarios.

2.2.1 Probability of detecting a known obstacle

For the most part, we expect that that the calculated distance from an obstacle in m match the respective LIDAR scan to that obstacle (i.e. in the same direction relative to the racecar). We model this by saying $z(i)$ is normally distributed about the ground truth distance.

$$p_{hit}(z^{(i)}|x,m) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z^{(i)} - d^{(i)})^2}{2\sigma^2}\right)$$

2.2.2 Probability of detecting an unknown obstacle

It's possible that the LIDAR scan detected an object that is not accounted for in m such as a stray person or chair. We assume that these objects are distributed uniformly, so it's more likely that the LIDAR hit them closer than farther.

$$p_{short}(z^{(i)}|x,m) = \frac{2}{d} \cdot \left(1 - \frac{z^{(i)}}{d}\right) \cdot \mathbb{1}_{0 \le z^{(i)} \le d^{(i)}}$$

2.2.3 Incorrectly detecting large distances

Due to the reflective nature of objects, it's possible that the LIDAR beam may bounce around a little, leading to measurements that have maximum distance

measurements. We account for this by saying $z^{(i)}$ is sampled from an approximate delta distribution at that maximum distance.

$$p_{max}(z^{(i)}|x,m) = \mathbb{1}_{z_{max}-1 \leq z^{(i)} \leq z_{max}}$$

2.2.4 Completely random measurements

Finally, as a generalization for cases we cannot account for and erroneous measurement errors, we will say $z^{(i)}$ is sampled from a uniform distribution of all possible distances.

$$p_{rand}(z^{(i)}|x,m) = \frac{1}{z_{max}}$$

2.2.5 Final distribution

Once we've established all the aforementioned distributions, we can have our final distribution of $x^{(i)}$ sample from the distribution created by their weighted sum, where all the α values are tunable parameters but must sum to 1.

$$p(z^{(i)}|x,m) = \alpha_{hit} \cdot p_{hit} + \alpha_{short} \cdot p_{short} + \alpha_{max} \cdot p_{max} + \alpha_{rand} \cdot p_{rand}$$

To ease computation times, we discretize our probability distribution over a square grid consisting of possible $z^{(i)}$ and $d^{(i)}$ pairs. To ensure the grid is accurate, we discretize the component distributions first before combining them.

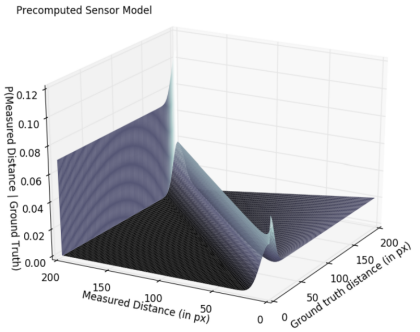


Figure 4: A view of how the final precomputated probability table looks like

2.3 Particle Filter (Jiahai)

We use a particle filter to localize the robot, updating on sensor and odometry data. We use 200 unweighted particles to represent the distribution of poses. To obtain a point pose estimation, we simply take the arithmetic mean of the translation component of all the poses. For the rotation component, we cannot simply take the average of the angle, because 2π and 0 radians are equivalent. Instead, we take the arithmetic mean of the quaternions, and rescale it to form a valid quaternion.

2.3.1 Odometry Update

Every time we receive an odometry message, we estimate the relative pose transformation of the robot from the last odometry update. The odometry message contains the instantaneous rate of change of the robot pose, so we estimate the relative pose transformation since the last update by multiplying the rate of change with the time elapsed since the lats update.

As elaborated earlier, the odometry model then applies the odometry update to all the particles, but with injected noise.

2.3.2 Sensor Update

The sensor updates come in the form of LIDAR scans. We pass the scan into the sensor model, downsampling if necessary, and obtain the posterior probabilities for all the particles. Then, we resample with replacement the particles based on the posterior probabilities.

To prevent race conditions, a mutex lock is placed on the particles during odometry and sensor updates.

2.4 Google Cartographer (Tian)

To experiment with simultaneous localization and mapping (SLAM), we installed a well-known SLAM package, Google Cartographer. In particular, we used Ninja-build to build the melodic cartographer distribution, reflective of our Robotic Operating System (ROS).

The cartographer collects data from numerous onboard sensors including LiDAR, cameras, and inertial measurement unit (IMU) to determine the position of the sensor and simultaneously create a map of the sensor's surroundings.

2.4.1 Cartographer Visualization

After installing Google Cartographer on our racecar, we launched 'cartographer.launch'. Then we manually drove our racecar around the Stata basement using 'teleop' to collect information on the location for mapping and localization. By subscribing Rviz to the correct topic (/cartographer_map), we were able to visualize the Google Cartographer processed sensor data.

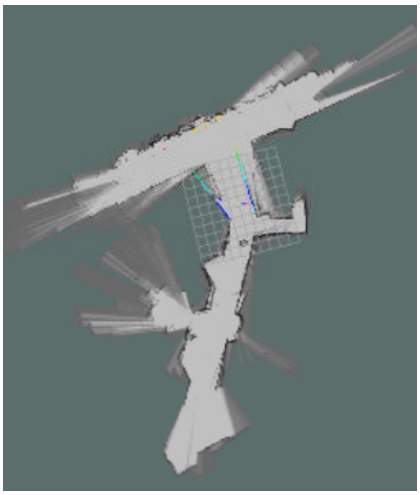


Figure 5: The Stata basement map as captured by Google Cartographer after manually driving the racecar the basement. The position of the racecar is shown in the map.

3 Experimental Evaluation

3.1 Simulation (Jiahai)

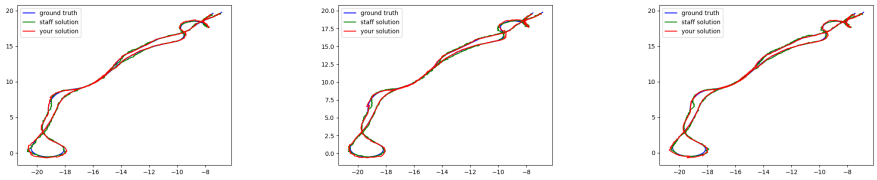
We relied heavily on the use of simulation for experiments. The provided autograder on gradescope runs the localization algorithm in a simulation, tracks the point pose estimate and compares it with the ground truth in the simulation. A metric is also computed from each simulated run that measures the mean absolute deviation from true trajectory. That is:

$$d = \frac{1}{T} \int |x_{est}(t) - x_{true}(t)| \, dt$$

The autograder runs the particle filter on three runs. All three runs use the same ground truth trajectory, but varies in the amount of noise in the simulated odometry data, which tests the algorithm's robustness to noise. The plots of the estimated trajectory, the ground truth trajectory, and the trajectory obtained by the reference particle filter implementation are shown in Figure 6.

	No noise	Less noise	More noise
Our particle filter	0.278	0.278	0.268
Reference particle filter	0.192	0.209	0.196

Table 1: Values of d (mean absolute deviation) in meters for the 3 noise conditions



(a) No noise (b) Less noise (c) More noise

Figure 6: Estimated, ground truth, and reference estimated trajectories under three noise conditions

Table 1 shows the mean absolute deviation d for the three noise conditions. Interestingly, we note that similar to the reference implementation, the performance of the particle filter does not degrade appreciably as the amount of odometry noise increases. This suggests that the noise in the odometry update model is sufficient to account for the simulated odometry noise.

3.2 Physical racecar (Ethan)

After achieving great performance in simulation, the next aspect of the lab was to implement our Monte Carlo Localization on the physical robot.

There were a few clear changes that had to be made. First, we had to down sample the racecar's lidar data. In simulation, the scan is made up of 100 scans whereas in reality it is just over 1000 scans. Many of these lidar points are redundant and will slow down computation, so instead we take a small subset of the scans. We did so by taking 100 evenly spaced points from the original data.

Next, we had to change a few topic name within our algorithm. This consisted of changing the odometry topic to "/vesc/odom" and the particle filter frame from "/base_link_pf" to "/base_link". In simplest terms this allows our particle filter to utilize the real-world measurements from our racecar. Furthermore, we no longer listened to noisy odometry data, as unlike the simulation the physical robot already has noise in the data.

Lastly, we made a few adjustments to allow for visualizing our particle filter in real time. This started by creating a node that published the transform from the robot to the map. This was done so that the car would move in RViz. We then created a launch file to source the stata basement as the reference map. Here are the results from our experiments driving around the physical racecar. It worked out very well, as evident by the figure above. Our MCL was excellent at localizing the robot even as we swerved around the environment. The biggest obstacle in getting this to work was setting an initial pose of the robot in Rviz.

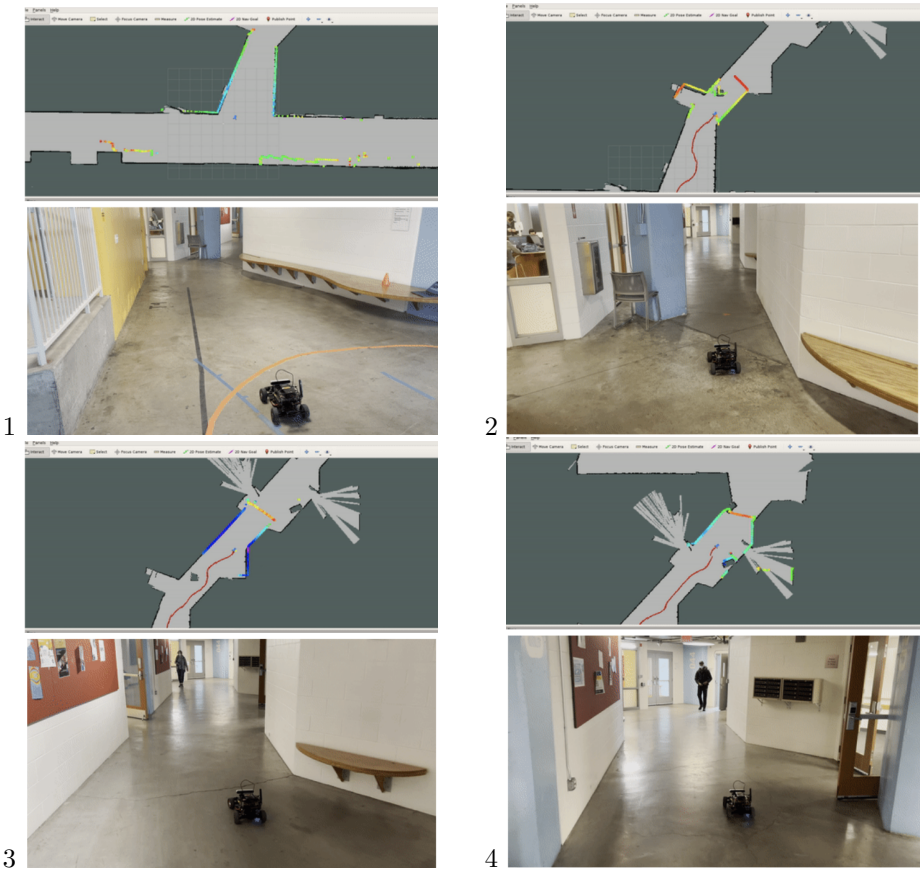


Figure 7: In each screenshot, the bottom picture shows the robot traversing throughout the stata basement and the top picture shows the real-time graphics produced by RViz.

Without a pose initialized, the particle filter will not begin to execute. After various efforts to implement this, we eventually settled on simply hard-coding a starting pose. Although not best practice, it was a simple work around for the time being.

4 Conclusion (Tian)

In this design phase, we successfully processed our LiDAR scan and odometry data to localize our race car in a familiar map (the Stata basement). Our motion model accounts for possible noise in the racecar's odometry data by generating guesses for possible poses. Our sensor model assigns probabilities to the poses generated by the motion model considering the racecar's sensor data. We com-

bined our motion and sensor models to create a particle filter that estimates the racecar's pose. Finally, we were able to take the lab one step further by performing SLAM on our physical racecar.

We completed this phase of the lab successfully. Now that we know where the racecar is on the map, the next step is to use this information for path planning.

5 Lessons Learned

Presents individually authored self-reflections on technical, communication, and collaboration lessons you have learned in the course of this lab.

5.1 Tian's Reflection

My main technical takeaway is to look for the official source when installing software. I ran into many errors, such as missing packages and compatibility issues that can be resolved by straying away from the lab instructions. Instead, I can refer to the official website which mentions how to deal with some of the errors I came across.
In terms of communication, our zoom presentation was different from previous in-person presentations. The main difference is that only one person has control over the slides. I learned to effectively and subtly communicate how to transition to the next slide without interrupting the flow of the presentation.
In terms of collaboration, I ran into more issues with my portion of the lab than expected, and due to scheduling, I couldn't spend more time with the physical robot. I learned that I can ask my teammates for help by providing clear instructions on what I need.

5.2 Jiahai's Reflection

I learnt a lot from this lab about the importance of visualization for tuning and debugging robotics algorithms. Our particle filter implementation did poorly on the gradescope autograder at first, but it seemed to work fine locally. By visualizing the pose clouds represented by the particles, I was able to deduce that the sensor model required more noise than what was initially present.

5.3 Bradley's Reflection

This lab I was humbled again by attempting to change to many things too quickly. In order to speed up the motion model I needed to refactor my code. Unfortunately when I did I started with large test cases that were difficult to debug. Once I started to work with the smaller examples it was much easier to find out what was going wrong. When making big changes to code I need to constantly remind myself to slow down so I don't break things.
For communication I learned from Tian how to effectively communicate that you want a slide change from someone else during a presentation. She would make

use of phrases such as "on the next slide" and "the next step" which seamlessly integrated into her explanation of a topic as well as gave a heads up to switch slides without being intrusive. I will try to integrate this into my collaborative presentations in the future.
For the collaboration piece I believe that I could have been more proactive in explaining my work on the motion model. If I had left some more comments and gave a quick explanation to the group when they needed to start using it then it would've been a lot clearer as to how to make use of the code I made.

5.4 Tahmid's Reflection

A technical achievement for this lab would be creating 2 versions of the sensor model. Although the sensor model isn't called as often as the motion model since the probability table only has to be precomputed once, it was still good practice creating a version that utilized more numpythonic code. On a similar note, I noticed that even though the 2 versions were the exact same, one would not pass tests while the other would. This leads to my communication achievement for this lab where I improved my ability to work with others such as TAs to find issues with unit tests because code is only as good as the test that validates it.

5.5 Ethan's Reflection

Lab 5 certainly tested my perseverance. Although relatively easily to make a few modifications to implement in part c, it took an egregious amount of time to debug RViz on the racecar. I do feel accomplished after finally completing it. They were non-trivial bugs about the backend of ROS so I didn't take away much from this.

In terms of communication, I felt that I better managed my time in the briefings. In our last presentation we went over the eight minute mark, so this week I focused on solely stating the essential information and to keep it moving. From the feedback we received as a group, it seemed to go very well.

Lastly, after this lab I certainly have a deeper appreciation for collaboration. On paper it may seem like a more effective plan to delegate and have each member take an individual section. However, I found this week that two people working together were more than twice as productive when working together. This would imply that (sometimes) it is better to have multiple people work together on multiple sections rather than each person working alone. Bouncing ideas off of each other and having another set of eyes for sanity checks is critical in problem solving.