Lab 5 Report: Localization

Team 9

Zhenyang Chen
Kwadwo Yeboah-Asare Jr.
Meenakshi Singh

6.141/16.405 Robotics: Science and Systems

April 25, 2022

1 Introduction

Author: Meenakshi
Localization is a problem within robotics that focuses on determining the position of the robot within a known map. The goal of RSS Lab 5 was to understand and implement the Monte Carlo Localization algorithm in three parts: the motion model, the sensor model, and the particle filter.

1. The motion model portion of the lab involved taking the wheel odometry coming from integration of motor and steering commands and applying it to the current position of the robot.

2. The sensor model portion of the lab involves taking the position of the particles determined by the motion model and assigning the particles weights based on their likelihood. This step increases the chances of particles that are more likely to represent the robot position being sampled on successive iterations. In our approach to the sensor model, we constructed a probability look-up table where each column contained normalized Lidar measurements d_k and each row contains the distance z_k from raycasting at the particle. We did this in order to reduce the computational complexity of having to recalculate probabilities for each particle.

3. The particle filter takes the results of the motion model and sensor model. Using the odometry data and the motion model we update the particle positions. Then, we use the sensor model to compute the particle probabilities and resample the particles. For each update of the motion or sensor model, we calculate the average particle pose and publish that transform.

2 Technical Approach

2.1 Motion Model

Author: Nana
For the lab we applied the method from the probabilistic robotics textbook to get the position the robot in the world frame given an odometry.
The method comes in three steps:

1. Rotate the car in the direction of the odometry vector.
2. Move the car in the direction of the odometry using the dy and dx values
3. Rotate the car the remain degree to allow its total change in orientation to match the odometry.

The equations for theses various steps are outlined below

First rotation in direction of odometry vector

$$o_{rot1} = arctan2(dy, dx)$$

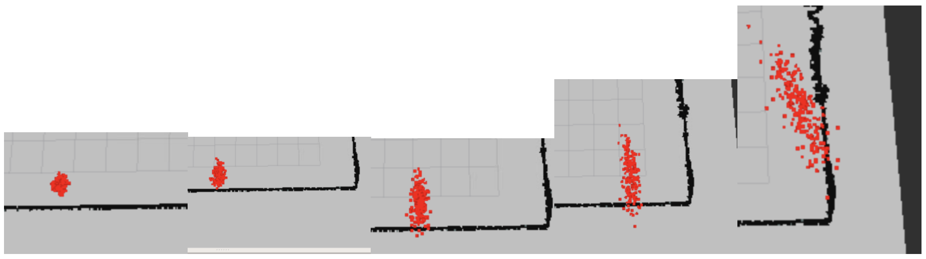
magnitude of movement in direction of odometry

$$o_{tran} = mag(dx, dy)$$

remaining rotation to get the correct orientation for the robot

$$o_{rot2} = d_o - o_{rot1}$$

To add noise to the model we simply add some gaussian noise with a suitable standard deviation to the d_{theta}, dy and dx values given by the odometry.
Here are images of the particles spreading out as the motion model progresses without resampling.


Progression of particles without resampling

With resample particles stay clumped together at the location of the cars

2.2 Sensor Model

Author: Zhenyang

Introducion

After updating the estimated state for each particle, we will create measurements for each particle. Theses measurements will be used to compare with the real measurement at time t, provided by the Lidar on the car and tell us which particle has the largest possibility to represent the current robot state. These probabilities will be used in the resampling process when we build the particle filter.

2.2.1 Probability of Sensor Model

The probability model of sensor contains several parts: gaussian noise, higher probability of hitting objects closer to the robots, maximal range, and random noise. These noises can be represented by the following equations:

$$p_{hit}(z_k^{(i)}|x_k, m) = \begin{cases} \eta \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z_k^{(i)}-d)^2}{2\sigma^2}\right) & \text{if} \quad 0 \leq z_k \leq z_{max} \\ 0 & \text{otherwise} \end{cases}$$

$$p_{short}\left(z_k^{(i)}|x_k, m\right) = \frac{2}{d} \begin{cases} 1 - \frac{z_k^{(i)}}{d} & \text{if} \quad 0 \leq z_k^{(i)} \leq d \text{ and } d \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

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

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

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

2.2.2 Precomputed Sensor Model

In our application, we are interested in finding a probability that represent how likely a Lidar measurement represents a real robot state. Where m is the given map, x_k is the real state and z_k is the measurement coming in.

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

When using the particle filter, we need to calculate this probability every time step which involves in calculating n (number of the particles) probability. To reduce the computational complexity, we will build a probability look up table where the column is the distance from Lidar measurement d_k and the row is the "real" distance z_k given by the particles. We built up a table with 201*201 dimension, start from $d_k = 0$ and $z_k = 0$. To make the probability reasonable, we also normalized the Gaussian distribution and probability for each column. And the figure below shows the what look up table looks like.

2.2.3 Evaluate function

:
For the evaluate function, we need to return probability value of each particle using equation (1), given the estimated motion state and the Lidar measurement. We first rescaled the state of particles and measurements of the Lidar, turning them from meters to pixel. And then to make the probability reading process more efficient, we utilized the slicing techniques provided by the numpy.array. We tiled up the rescaled Lidar measurements data and used it as a column indices and did the same for the particle states.

2.3 Particle Filter

After building the motion model and sensor, we combined them together to make the whole particle filter. The idea is simple. Every time we receives odometry data, we update the states of all particles using motion model. When we receive Lidar data, we do the ray-casting and evaluate particles, which return a probability weights for us.

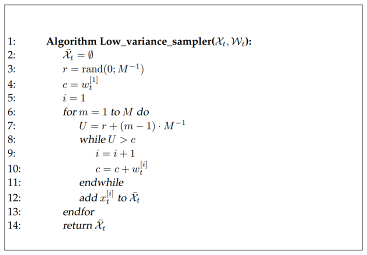
2.3.1 Initialize Position

:
To initialize the position we publish the 2D position from Rviz to the localization before we start running anything. All the particles are initialized in a small radius around the initialized point the the motion model and re-sampling are run. The motion model applies the odometry to the set of particles every time it receives a new odometry and the re-sampling re-picks particles based on which are more likely to be the correct position of the car.

2.3.2 Low variance re-sampling

: Author: Zhenyang
After testing our particle filter with a simple version of resampling, we realized the particles will converge very fast during the resampling process and lose diversity after several iteration. Though the variance of the particle set is decrease, the variance of the particle set as an estimator of the true belief increase [1]. After reviewing textbook, we tried a new resampling algorithm with variance reduction.

The basic idea of low variance resampling is to add a random offset when resampling. In this way, the algorithm can cover the whole sampling space in a more systematic and efficient way while following the probability distribution and avoid losing samples when there is no updating. The detail process of this algorithm is shown below. Further experiments and comparisons with our simple resampling algorithm will be discussed in the experiment part.



```
Algorithm Low_variance_sampler($\mathcal{X}_t, \mathcal{W}_t$):
    $\bar{\mathcal{X}}_t = \emptyset$
    $r = \text{rand}(0; M^{-1})$
    $c = w_t^{[1]}$
    $i = 1$
    for $m = 1$ to $M$ do
        $U = r + (m - 1) \cdot M^{-1}$
        while $U > c$
            $i = i + 1$
            $c = c + w_t^{[i]}$
        endwhile
        add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$
    endfor
    return $\bar{\mathcal{X}}_t$
```

Figure 1: Low Variance Resampling

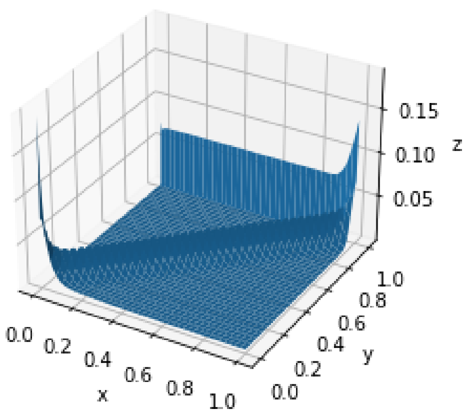


Figure 2: Precomputed Sensor Model

3 Experimental Evaluation

To test our localization on the real race car, we initialized the car in simulation to the same position in simulation and moved the real car. Ideally the position of the car in simulation matches that of the car in simulation and the particles should also be centered on the simulation race car.

3.0.1 Angle and distance error

The graphs below are the the errors of the published position vs the actual position of the car. These graphs are taken during various situations. Below is a graph of the distance and angle error as the particles converge. We see that

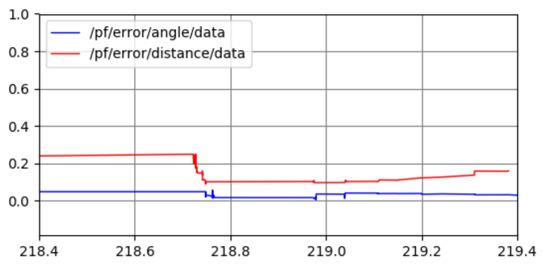


Figure 3: Errors converging

the the distance error converges more quickly than the angle. This is probably due to the fact that the standard deviation for the noise added to the distance and angle odometries are the same and this affects the angle much more since the relative scale is much greater for the angle data.

We see a similar pattern during turns were the distance error barely changes but the angle error swings wildly before settling down again.

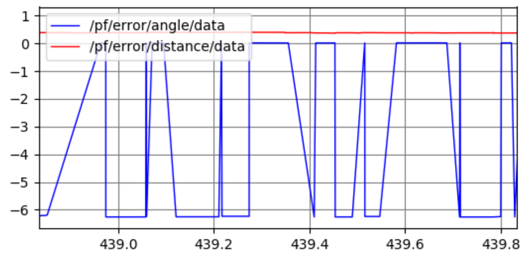


Figure 4: Errors converging

3.0.2 Normal resampling vs Low Variance resampling

When comparing the two resampling methods. Numpys random choice with weighted probabilities and the low variance resampling presented in the text-book. The benefit of the low variance filter is it slows down the convergence and avoids converging too early from the chart below we see one such case were the angle and distance converged on the wrong value and stay there. More noise could be added to fix this but it is inconsistent how much you might need to add for different situations so a more slow convergences is more consistent.

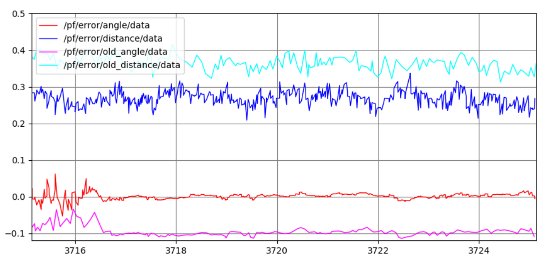


Figure 5: Errors converging

3.0.3 Number of particles

The last test was the number of particles. The behaviour of the localizer did not change however the publish rate of the average pose of the particles moved from 53hz to 48hz which is well above the 20hz mark mention in the lab workbook.

4 Conclusion

Author: Meenakshi
Over the course of lab 5, we learned about Monte Carlo Localization. We first understood the mathematical theory behind the concepts of the motion model, sensor model, and particle filter. Then, we implemented the models in simulation and refined them until we were satisfied with their accuracy. Finally, we tested our particle filter on the racecar in Stata basement and refined it.

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 Meenakshi

Due to personal circumstances, I was unable to help with many of the technical aspects of the lab early on. However, despite the beginning circumstances, I was able to gain experience with collecting and processing data from recorded rosbags. I also gained a better understanding of the challenges that come with localization problems. Although I have a theoretical understanding of the model utilized, I am not as familiar with the problems that came with testing it on the robot. Another thing to improve moving forward is to document our progress along the way in a clear manner because we had methodical approaches to improve the performance of the particle filter based on testing, but the iterative changes were not documented so looking at the code between versions can feel like large gaps in knowledge.
My biggest takeaways from this lab is about the importance of communication within the team during the face of external factors. I am very apologetic that my situation contributed to the team falling behind. Even though I could not control those external factors, I could have been more responsible by actively communicating with my teammates so that the division of work was more fair to them.

5.2 Zhenyang

The biggest lessons I learned from this lab is to see the large gap between theory and engineering practice and see the importance of communication and collaboration in an engineering challenge. Particle filter is a kind of Bayesian filter using Mente-Carlo method to sample motion state and estimate the real robot states. Though the idea seems simple, the mathemacical derivation looks messy and straight forward. When it comes to implementation, we need to deal with more realistic cases. For example, to reduce computation complexity, we pre-compute the sensor model, and to mimic the real motion state of the robot, we add noise in motion model. This details and tricks are not necessarily included in the formulas, but it is definitely worth noting and thinking, and they result in the correct performance in real application.
This week is challenging for me and for the team. For the last two weeks, I lost connection with my teammates. One of them feels not well, the other one decided to drop the class before briefing. This leads to a lag in progress and makes me realize everyone is important and responsible in an engineering team. I believe with coming back of my teammate and the join of new teammate, we can do much better in the Lab6. Keep working, keep updating with others.

5.3 Kwadwo

Main takeaway is sometimes trying other methods for the sake of it can lead to great results. After reading through the method outlined in the textbook, I found it much easier to understand and implement than the rotation matrix. Outside of that this lab was particularly challenge. We were largely left with a team of two to complete the lab since one teammate fell sick and the other dropped the class. If anything it did show me that having a full team is extremely important for any engineering project as with what we did manage to do it would have been much easier to accomplish with a full team.

References

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.