Lab 5 Report: Localization

Team 1

Vasu Kaker
Ben Evans
Emmanuel Anteneh
Johlesa Orm
Michelle Zbizika

6.4200/16.405 (RSS)

April 11, 2024

1 Introduction

Author of section: Johlesa Orm

The goal of this lab was to utilize odometry information of a robot and harness the principles of probability in order to localize the robot's position in a known field. Specifically, we use The Monte Carlo Localization (MCL) Algorithm in order to achieve this. The value of this algorithm can have great use in mapping or adjusting the speed and controls of an autonomous vehicle ahead of time based on predicted pose, which can lead to a smoother driving experience. MCL can be a great addition to any existing driving algorithm (e.g. Wall Follower: Lab 1) to handle situations like turning a corner or encountering obstacles with more stability.

Overall, the main parts of the MCL Algorithm are as follows: of the scans received from the robot, add noise to each of the particles found in the scan in a motion model. Then, compute the probabilities of the robot being at the location of the particle with a sensor model. From there, as new information about the odometry is received, continually update the probabilities and compute the average pose to get the predicted pose of the robot in a particle filter. Our solution used 200 particles and added noise to the obtained particles using a Gaussian distribution. In the following sections, we dive deeper into the calculations behind the algorithm and its performance.

2 Technical Approach

Author of section: Michelle Zbizika

2.1 Motion Model

Motion model takes odometry data and updates the particles to reflect probable future states.

$$\begin{bmatrix} dx_{rotated} \\ dy_{rotated} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} \tag{1}$$

$$\text{particle } += \begin{bmatrix} dx_{rotated} \\ dy_{rotated} \\ \delta\theta \end{bmatrix}$$

Then we normalize the updated particle angles to the range $[-\pi, \pi]$. We introduce noise the the particle values by adding a random number from a Gaussian distribution centered at 0 with a standard deviation of 0.3 meters to make a probable set of particles and predict the next pose.

2.2 Sensor Model

Given the updated particles, we want to predict the probabilities of each of them occurring, and use the probabilities to predict the odometry of the robot. The sensor model implements this in the following. For each particle, we sum the chance of it occuring from 4 different distributions: p_{hit} chance of a known obstacle, p_{max} chance of a very large measurement, p_{short} chance of a short measurement, and p_{rand} complete random. In order to minimize computation time, the z_i and d values are discretized, and a lookup table representing the sensor model sum of these probabilities $p(z_k^{(i)}|x_k, m)$ is precomputed and stored. The rows of the table represent discretized z_i values and the columns are discretized d values, the values in the table are the probability of measuring any discrete range. Numpy arrays are used throughout to increase efficiency. We use the following parameter values $\alpha_{hit} = 0.74, \alpha_{short} = 0.07, \alpha_{max} = 0.07, \alpha_{rand} = 0.12, \sigma_{hit} = 0.0$.

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

$p(z_k^{(i)}|x_k, m) = \alpha_{\text{hit}} \cdot p_{\text{hit}}(z_k^{(i)}|x_k, m) + \alpha_{\text{short}} \cdot p_{\text{short}}(z_k^{(i)}|x_k, m) + \alpha_{\max} \cdot p_{\max}(z_k^{(i)}|x_k, m) + \alpha_{\text{rand}} \cdot p_{\text{rand}}(z_k^{(i)}|x_k, m)$ We normalize the p_{hit} values by column values d so that they add to 1.

The sensor model also has an evaluate function, which evaluates how likely each particle is given the observed scan. Scans and observations are scaled by

the resolution and LIDAR scale to map scale, and clipped to the range $[0, z_{max}]$. We convert the distances in the LIDAR observations and the result of ray casting from meters to pixels. dividing by (map resolution*lidar scale to map scale).

We then return the probabilities, which are the multiplication of all the observation and scans. We downsampled the real life LIDAR from 1081 particles to 200 particles.

The reason why we implement a sensor model that takes into account these probabilities is that measurements in the real world are imperfect and often contain noise, and by taking those into account, it is more likely that our prediction captures the true future pose of the robot.

2.3 Particle Filter

After we have implemented the sensor and motion model, we then utilize them in the particle filter. The overall structure of the particle filter as follows: it gets the current pose of the robot, uses the sensor and motion model to predict the next possible positions and their likelihood, and takes the average of these positions to get future odometry. The particle filter is implemented as a node and initializes the sensor model and motion model. When it receives a laser scan, it uses the laser scan to evaluate the new probabilities of particles, and then resamples and updates the particles based on the probabilities. When the particle filter receives odometry data, it uses the motion model to evaluate and update the probabilites of future particles. After every new update, the particle filter takes the "average" location of these particles and publishes this pose and the array of particles to odometry publisher and pose array publisher.

We get the "average" location of the average x value of all the particles, the average y value of all the particles, and the circular average of the θ of all the particles. One drawback of this is that if the particles have a multimodal distribution, the average that we calculate might be in a very unlikely location. (1) (2)

3 Experimental Evaluation

Author of section: Benjamin Evans

3.1 Initial Testing

For testing, we convert the predicted location from the model to a point in the physical (or simulated) world, and compare this to where the robot actually is and take this difference in meters.

The robot was first tested in simulation by running wall follower, and measuring the error between the predicted location and the ground truth location in the subsequent robot path. To evaluate the error of the system in simulation,

the node finds the ground truth position by listening to the transformation between map and base link, and then takes the difference between this location and the predicted location to get the error. We publish this error to get the graph shown in figure 1. In this Error vs. Time graph, the robot initially has some error but it able to localize shortly after. It accumulates significant error after turning a corner but is again able to localize quickly.

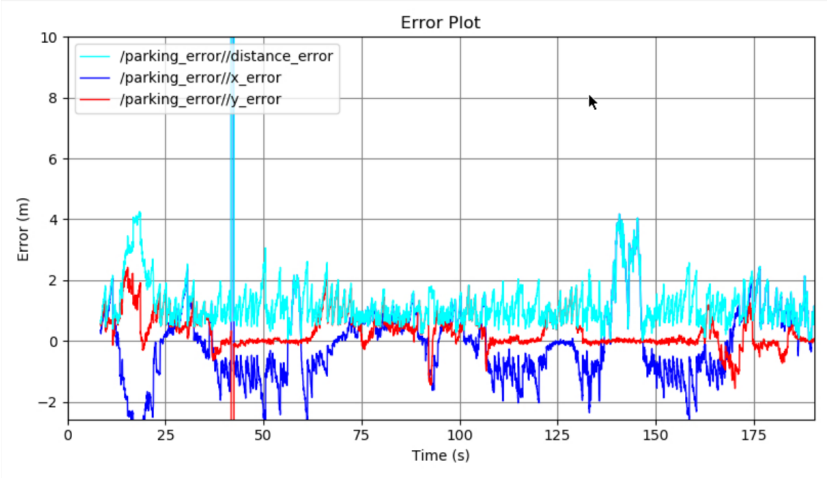

Figure 1: **Localization Error in Simulation**

3.2 Simulator Robustness - Noisy odometry

In the simulator, at varying noise levels, we measure the distance error as the robot performs the wall following task. We also record the amount of time it takes for the error to converge to an approximate minimum. These noise levels, errors, and convergence times are listed in table 1. In real life, the localization will likely have convergence times much different from these.

3.3 Real World Testing

Unfortunately due to technical difficulties, we were unable to thoroughly test our model on the physical robot. Prior to these technical issues we were able to get the robot to localize itself in the basement of MIT building 32. Data was not acquired for this, but qualitatively it seemed to operate fairly quickly and accurately. More testing on the robot is currently in progress, but data will not be available in time for this report.

Noise level (xy, m)	**Noise level (θ, rad)**	**Avg. Error (m)**	**Avg time to converge**
$\sigma = 0.1$	0.01	3.9	DNC*
0.1	0.05	4.2	DNC*
0.1	0.1	1.3	0.5
0.3	0.01	1.2	1
0.3	0.05	3.6	DNC*
0.3	0.1	1.1	2.5
0.5	0.01	1.0	0.5
0.5	0.05	0.9	1.33
0.5	0.1	0.9	0.625
0.7	0.01	2.7	DNC*
0.7	0.05	4.2	DNC*
0.7	0.1	2.1	DNC*

Table 1: **Convergence With Varied Noise Levels**
*DNC = Did Not Converge

4 Conclusion

Section Author: Emmanuel Anteneh

As a whole, we were successful in implementing robotic localization using particle filter in simulation. The main remaining work is to adapt our implementation to work on the physical race car, and conduct further experimental analysis in real world environments.

Implementing particle filter (aka MCL) required implementing motion model and sensor model. Given our estimations of the robot's current pose and orientation, motion model uses odometry data to update our estimations to reflect their likely future states. Sensor model then predicts the probabilities of each estimation being the true position and orientation. The models are then both utilized by particle filter which continuously estimates potential future poses/orientations for the robot, and weights the estimations by the likelihood that they reflect the true state of the robot, while averaging the current set of estimations to get future odometry.

Evaluating our particle filter in simulation yielded good results, as error stays low in most environments and noise values. While our implementation was successful in simulation, running MCL on the physical racecar introduced challenges that we still need to address, such as resolving issues with running out of storage when running the algorithm. Moreover, once these problems are resolved, we will need to evaluate and further tune our particle filter to achieve ideal performance.

5 Lessons Learned

Johlesa Orm: This was a very challenging lab which tested our abilities and endurance in debugging and learning new concepts like probabilistic robotics. I gained an appreciation for the usefulness of this algorithm after spending a lot of time with the models and seeing the fruits of our labor work in simulation and in real life. Another difficult part of this lab was the time management and collaboration. I acknowledge that since we started after spring break, it was difficult for everyone to coordinate times to work on the project together. However, we can definitely work on starting the project earlier and communicating with each other more frequently in order to find and fix problems sooner. Personally, I need to work on communicating any work needs to be split up so that I don't get burnt out and none of my teammates do. I think this lab was a great lesson for me going forward, and I can't wait to work with my team for the upcoming lab.

Emmanuel Anteneh: This lab was a lot of work, and was very helpful in getting intimately familiar with Monte Carlo Localization. It was one thing to learn about MCL in lecture, but actively implementing and debugging the algorithm taught me a lot and helped me gain a deeper appreciation for it. In the future, I want to start working earlier, as I have a tendency to procrastinate. I want to put less of a burden on my teammates and reduce deadline stress.

Ben Evans: I feel overall disappointed in my performance with this lab. Part of that is rooted in becoming ill partway through, but that would have been less of an issue if I hadn't procrastinated. In any case I felt that my communication with my teammates was inadequate for much of that time, and I will definitely work to do better. All that said, it was really interesting to see how this algorithm worked in real time on our robot (when it was working). The technical issues were very frustrating and definitely affected us more than they should have if we had been able to transition from simulation to the robot earlier.

Vasu Kaker: I really enjoyed this lab. I was fairly involved in the particle filter development, and in moving the code onto the robot. We were able to get the code tested and running with the robot, but didn't manage to get the figures in on time. I found the experience to be overall pretty satisfying and fun. I really enjoyed learning about MCL and deeply understanding the concepts behind it.

References

[1] RSS_Course_Staff, "Lab 5: Monte carlo localization (mcl)," Available at https://github.com/mit-rss/localization (2024/04/10).

[2] S. Thrun, W. Burgard, and D. Fox, *Probabalistic Robotics*, 1st ed. Cambridge MA: MIT Press, 2005.