Team 19: Stephanie Howe, Mary Lau, Prajwal Mahesh, Sean Crozier, Uche Okwo

6.141

2022 April 02

Lab 5 Report: Where Are We? Localization with Team 19

Introduction

Localization, the problem of determining a robot's position and orientation in a known environment, is critical to enabling our racecar to autonomously navigate its surroundings. We solve this problem of localization by implementing a Monte-Carlo particle filter, which was able to accurately localize in both simulation and on our physical racecar. For the problem of localization, we have a map of the environment we expect our robot to be in. To carry out effective localization, we must strategically fuse this prior map knowledge with our robot stream of proprioceptive/controls and exteroceptive/sensor data in order to actively maintain the location and direction on the map that the robot believes itself to be at (aka its pose). What is unique about the Monte-Carlo particle filter is that it initializes and maintains a distribution of possible states, multiple hypotheses of where the robot believes itself to be. As the robot moves and collects more control and sensor information, it shifts and evolves these belief particles accordingly. Eventually, the distribution of particles will converge to an accurate estimation of the robot's pose. Our implementation of this process, which is iteratively carried out by our localization model, breaks down into the following two stages. First, given a set of particle beliefs, we need to implement a module that can predict the state of these particles in the next timestep, based on the robot's odometry (its controls and proprioceptive data). Second, we require a module that would apply the robot's exteroceptive sensor data to update these particle predictions.

Motion Model (Mary)

We implemented this first module which predicts the motion of the robot based off of the mathematics of transformations and rotation matrices. Each belief or particle is represented by the coordinates (x, y, θ); x and y represent its position on the map, and θ represents its steering angle, as a rotation about the z-axis. The module also accepts an odometry reading, represented as: $(\bar{x}, \bar{y}, \bar{\theta})$. To update a particle with the odometry, we convert both into a rotation-translation matrix, and multiply the odometry's matrix with the particle's matrix.

$$R_{particle}R_{odom} = \begin{bmatrix} cos(\theta) & -sin(\theta) & x \\ sin(\theta) & cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\bar{\theta}) & -sin(\bar{\theta}) & \bar{x} \\ sin(\bar{\theta}) & cos(\bar{\theta}) & \bar{y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} ... & ... & x\bar{x} + y\bar{y} + 1 \\ ... & ... & \bar{x}sin(\theta) + \bar{y}cos(\theta) + y \\ 0 & 0 & 1 \end{bmatrix}$$

From this product, we can extract the x and y of our predicted particle. As for the angle of our predicted particle, we simply add the θs.

$$x' = x\bar{x} + y\bar{y} + 1y' = \bar{x}sin(\theta) + \bar{y}cos(\theta) + y\theta' = \theta + \bar{\theta}$$

We apply this calculation to all of the particles. In addition, to ensure that our particles are exploring with enough variation to account for noise from the odometry readings, we implemented an option to inject Gaussian noise into our particles at the end of the update stage. As such, we ensure that our particle filter will have considered a sufficiently varied spread of possible hypotheses for our robot's pose.

Sensor Model (Stephanie)

The sensor model is an important step in localization because it assigns each possible particle a likelihood. Assigning probabilities influences the next iteration of sampling, giving preference to particle hypotheses with a stronger likelihood of being correct. These probabilities represent how well a hypothesized particle's sensor reading matches the car's actual LIDAR sensor reading.

Our steps to implement the sensor model were three fold: generate probabilities for individual range values, store these probabilities in a precomputed table, and finally do an array lookup when calculating particle probabilities.

The likelihood of a particle's scan is the product of the likelihood of each range measurement in such a scan. The formula is as follows: $p(z_k|x_k, m) = \prod_{i=1}^{n} p(z_k^{(i)}|x_k, m)$

There are a few different probabilities that must be considered when calculating the likelihood for an individual range. These include the case of detecting something that is known, receiving a short measurement, receiving an overly large measurement, or getting a completely random range measurement. The formulas for each case, respectively, are as follows:

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

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

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

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

The final probability of the particle is computed as a weighted average of each of these cases, each with their own weight parameter $\alpha_{hit}, \alpha_{short}, \alpha_{max}, \alpha_{rand}$.

In order to eliminate the need to recompute these formulas for each range of each particle, we stored precomputed range probabilities in a discretized table. This table is a 200x200 grid representing different values of d and $z_k^{(i)}$, the particle scan's range value and the LIDAR range value, respectively. This table makes calculating whole particle probabilities much more efficient at runtime, where we then calculate the product of each range probability by looking them up in the precomputed table.

Particle Filter (Uche)

The particle filter is the synthesis of the motion model and sensor model for the purpose of localization using Monte Carlo methods. Something being Monte Carlo simply refers to the process of using randomization to solve deterministic problems. In this case, the problem is given a location and a map of the environment, to figure out where in the map the robot is as it moves around. The reason why a starting location must be given is that there is no current solution for the kidnapped robot problem, which is exactly the problem of figuring out a location given no information about where you start (as if the robot had been kidnapped and placed in a random location). The particle filter maintains a list of particles (each particle pose has an associated probability). In addition, it operates two separate callbacks, one for the robot's lidar scan and one for the robot's odometry. When the callback for the lidar is received, the particle filter program puts all the measurements through the sensor model, along with the particle poses, to output a list of likelihood values, which gives the probability that the corresponding particle is the actual pose of the robot conditioned on the lidar measurements. To carry out the Monte-Carlo part of localization, the robot then takes these likelihoods, and using the priors that it keeps track of for each particle, the particle filterupdates the particles' probabilities via Bayes's Rule. After that, the particle filter resamples the particles based on these updated probabilities.. This resampling process weights the particle filter towards particles that remain highly probable consistently. In the odometry callback, the particle filter updates the poses of particles using the motion model. To help combat noise within the odometry readings and to prevent resampled particles from becoming more similar to each other, we add noise to the motion model updates. After these callbacks, the robot publishes a pose as the result of localization. The particle filter selects the average of the most likely particles' positions and the average particle rotation. The reasoning for only selecting the most likely particles for the position of the robot is bimodal distributions, where the probability distribution is centered around two different values. In taking a pure average, the robot will end up selecting a position in between the 2 values, which itself may be highly unlikely. However, if angles become bimodal, this is considered a problematic situation that should not be handled separately, which is the reason for the distinction in the handling of the averaging of the particles to produce the reported pose. While our team's particle filter works well in simulation, there is a possible race condition that remains in the code: if odometry callback is running and the lidar callback is called (or vice versa), since we did not use any synchronization primitive, the particles will desync, in this case meaning that whichever process finishes first will be overwritten by the other, so in either case the particles will be updated based off of old data. This could be resolved by a mutex, but in practice, we haven't found a reason to introduce the possible deadlocks a mutex could cause. If more real world testing reveals this to be an issue, synchronization primitives from the python `threading` module could be introduced in order to resolve the problem.

Experimental Evaluation (Praj)

To confirm that our implemented Monte Carlo localization algorithm worked in practice to estimate the robots pose in a given map, we evaluated it in several stages. The ros localization package was first tested on a simulated robot model. Using the rviz gui, the robot was manually given an initial pose and was then allowed to drive autonomously around the map using a previously developed parking controller. Performance was evaluated at a high level by visualizing the calculated robot pose output and confirming qualitatively that it closely matched the robot's actual behavior. The test was repeated with the simulated robot at various starting positions and orientations around the map as well as with different robot driving maneuvers to ensure that the motion model, sensor model, and particle filter were robust to different situations the robot could be in. Once we confirmed that our implementation roughly behaved as expected, we moved on to more quantitative analyses. The localization package was submitted for testing in another simulated environment where it was put to test on a known path. In this series of tests, the odometry information given to our motion model was based off this ground truth path with varying levels of added noise to replicate noisy real-world sensor data. Localization performance was then evaluated by how closely the calculated robot pose matched the ground truth path. Specifically, the simulator measured time-averaged deviation from the true, known trajectory in meters, giving us a numerical result for optimization. To compensate for the simulated noise, we adjusted the total number of particles as well as the spread of these particles at each time step by changing the range of noise added by the motion model. This had to be configured so the hypothetical particles spread out enough that the robot's true position was among the guesses, but not so far that our particle cloud became too sparse to capture the robot's true position. The tuning of these noise parameters was done experimentally—first adjusting individual values independently in order to understand how they affected the result, followed by combining the results and optimizing for best performance. With this tuning process, we were able to achieve fairly accurate localization regardless of added noise and with low deviation over fairly long paths. Image XX1 shows the result of one of these simulated runs. It is clear that the calculated path in red closely matches (and in this case obscures) the ground truth path, indicating that our localization implementation was able to accurately track the robot as it moved through its environment. Image XX1 also depicts a shortcoming of our current implementation as the calculated path appears jagged and is full of higher frequency noise. Further filtering may be required to smooth these results.

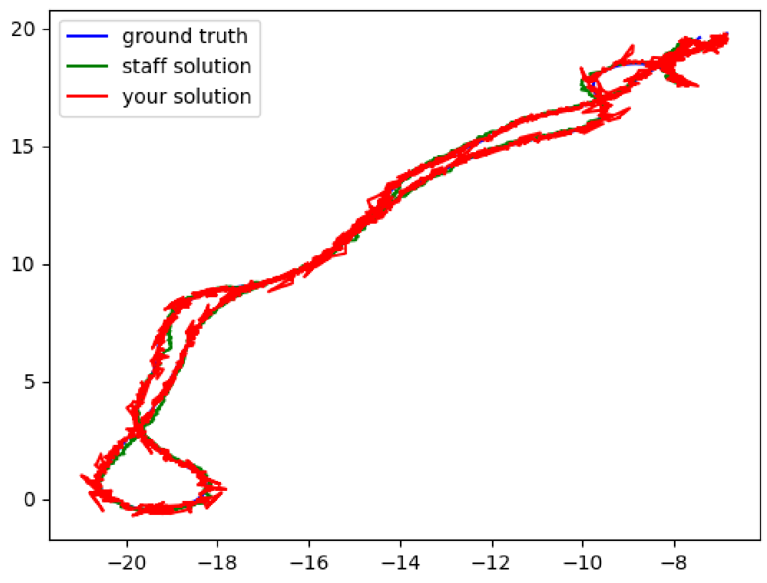

Figure 1: ImageXX1: Robot path as calculated by our localization system (red) closely matches and obscures the ground truth path (blue)

After testing and optimizing the localization code in simulation, we moved the package to the physical racecar platform for real-world testing. To evaluate our system, we placed the robot at a known location in the basement of Building 32 and manually drove it around different hallways. Because different levels of odometry and lidar noise were present on the racecar, re-tuning of parameters was required for good performance. We began with a high range of added noise in the motion model and progressively dialed it down until the calculated pose was well behaved and followed the motion of the robot. The data from different runs of the robot were recorded into rosbags for later analysis and once again, rviz was used to visualize the robot as it moved through the map frame using poses calculated by our localization implementation. In the initial tests we managed to run on the racecar, localization performed poorly with heavy skew relative to ground truth as shown by the red trail in image XX2. Increasing the noise range added to the motion model, as was done for image XX3, did help to lessen this problem as it allowed for the particle hypotheses to better account for odometry noise. Unfortunately, simply tuning the noise parameters was not able to alleviate the problem completely. Upon further investigation we found that the map model used by the scan simulator was incorrect, which prevented the sensor model from effectively paring down unlikely particles.

Images/lownoise.png Image XX2: Localized path (red) skews heavily from ground truth when not enough noise is added to distribute particles and account for odometry noise

Images/highnoise.png Image XX3: Localized path (red) skews less from ground truth when more noise is added to distribute particles and account for odometry noise

Following this, we were able to load the correct map but were unable to correctly set the initial pose of the robot relative to that map. Missing this key component limited any further testing we could do on the physical platform, but resolving these issues is a key action item as we move forward into other autonomous navigation tasks for our robot.

Conclusion/Lessons Learning (Sean)

The solution built here, comprised of the motion model used to integrate the odometry and generate particles, the sensor model used to calculate the likelihood of particles, and the particle filter that combines the two models to generate a set of poses describing the path of the robot, works effectively in simulation, though it presents challenges when implemented in situ. It provides the robot the capability to display it's location, navigate around a map, and make pose sensitive decisions based only on LIDAR data and odometry it receives while traversing it's route. Notably, this set of programs does not require the identification of landmarks to determining it's location, but rather computes the likelihood of particles based on simulated scans performed on a pre-constructed map of the location. Thanks to optimization of the sensor model and the particle filter, accurate results can be obtained in real time. The sensor model's reliance on precomputation looses some precision, but drastically decreases the time required for the evaluation of a particle. Synergistically, the particle filter's averaging of particles with the highest likelihood reintroduces higher precision, even if there is some probability the averaged pose isn't a valid location. The targeted resampling of high likelihood particles ensures high efficiency and return on time spent per particle. Though the algorithms performed well in simulation, difficulties with the application of the techniques plagued the successful function of the software when installed on the race car. It's less obvious when operating the program on the car which map the ray tracing software is making advantage of. If the wrong map is chosen for where the car is being driven, the ground truth values supplied to the sensor model will be inaccurate, and the computed probabilities will have less utility to the particle filter. Additionally, setting the initial position of the race car accurately proved to be another difficulty. The advantage of incorporating the motion model into the particle filter is that we avoid the "stolen robot" problem, but beginning the motion model without an accurate initial position requires the solution to this problem before a pose can be determined and a path started. Further refinement of techniques and operation of the race car should prevent this situation in the future. Due to an initial decision before spring break to split into two person pairs to work on segments of the project, our team found itself more adaptable to sudden changes in availability. Though a team member was unable to make the Wednesday briefing, the rest of the team was able to step up and execute the brief well.