Particle Filter

Sean Carter and Jeff Pflueger

March 2017

1 Project Overview

The goal of this project was to be able to use a Neato's laser scanner to find its location in a map. To do this, we designed our own particle filtering algorithm to generate guesses, and then refine those guesses based on updated data.

2 Basic Code Structure

All of the teams were provided with code that allowed for basic interaction with the map that was provided. The code was already capable of identifying the distance between a point and the nearest obstacle, so long as we provided a map. It also contained most of the utility functions that we needed to visualize our algorithm - publishing the cloud of particles, transforming between coordinate frames, and basic error handling. Our contribution was to create an algorithm that would:

- Accept an initial guess of the Neato's location
- Use that location to generate a particle cloud, identifying possible locations of the robot
- Update particles with Odometry data
- Re-weigh and Re-sample particles with laser scan data

3 Generating Particles

The user can identify roughly where the robot is by publishing a location and orientation to a rostopic. We generate a large number of objects representing new locations and rotations (the particles) by generating new locations from a Gaussian distribution centered on the initial guess. An example of how this looks can be seen in Figure 1 on the following page

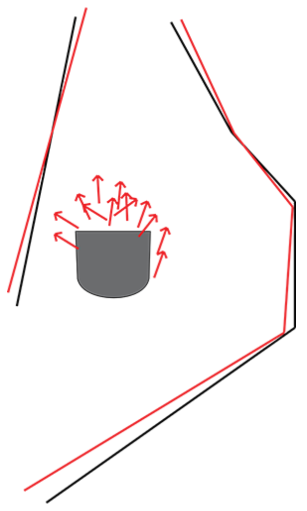

Figure 1: Example of particles generated by particle filter

4 Updating with Odometry

Filtering particles is too computationally intensive to be constantly running, so the code is structured such that it only runs once the robot has moved far enough from an initial position. We calculate the distance that the robot has moved, and the amount that it has rotated.

The most important part of this is making sure that the particle moves realistically. Our initial update would take the change in x and y locations, add some noise to it, and update every particle's location with it. Since the robot only moves forwards, this meant that particles facing the wrong way would be modeled as moving sideways. Instead, they should have caused the particle field to spread out, reflecting uncertainty in our estimate of the robot's orientation.

There were other problems. If the robot moves and rotates at the same time, a naive approach assuming that the robot rotates, then moves, results in the particle cloud becoming off center and scattering. This was our second approach, and its flaws prevented us from tuning the particle filter itself.

Our final iteration assumed that the robot rotated enough to point at its final location (based on odometry), moved forwards, and then rotated again (to its final orientation, also based on odometry). The effect of this change is that a particle starting out in the correct position is much more likely to be in the correct position after it copies the robot's movement. This significantly increased the effect of our localization algorithm.

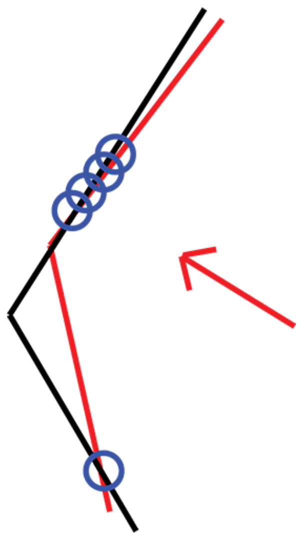

Figure 2: Example of projection of LIDAR data onto particle

5 Localization

To localize the robot, we loop through every particle. The LIDAR scan is projected on top of the particle, and we assume the orientation of the robot is that of the particle. We then check how many locations on the scan are within a threshold distance from the walls of the map. A diagram of this can be seen in Figure 2.

We then add up the total number of hits, and set that to the particle's weight. The weights are normalized and the position of the particles are averaged together based on their weights. The robot position is set to this weighted average. The particles are re-sampled, and the new particles are given positions based on the weights of the previous set of particles.

6 Decisions and Challenges

One of the major design decisions we made on this project was to use the "Hits" method of weighting particles rather that try to calculate error based on how far away the particles were from the wall. In hindsight, this made our lives way easier. We ran into many problems with the robot Odometry that we had to work out geometry for. Simplifying the weighting of particles allowed us time to focus on getting the other parts of the project right. The "Hits" method, though simple, works pretty well at localizing the robot. As we mentioned earlier, we struggled to get the Odometry updates for the robot and particles correct. We spent a lot of time banging our heads against the wall, trying to figure out where our problem was, and trying out fixes for it. In the end, we had a few problems:

- We were treating the robot as if it moved and then rotated. It doesn't seem like a problem at first but if you think about it, the order of operations changes where the robot ends up. To solve this, we made the robot rotate in lign with its angle of movement, move, and then rotate to its final position.

- When updating the particles, we were updating them as if they moved on the robot's orientation instead of their own

We learned many lessons from facing these challenges particularly in debugging.

We learned that isolating the problem is half of the battle. When trying to figure out why our code wasn't localizing, our first instinct was that the function that weighed the particles was wrong. This was because we didn't have the best understanding of how to do it, and implemented our best guess. It turned out that we were looking in the wrong direction. If we had spent time making sure our Odometry updates were working, we would have had a lot more time to make weighing function more robust.

We learned that don't try to fix everything at once. Also, if one part of your model depends on another part, make sure the first is working before moving on. We often found ourselves making lots of changes at once, to many different parts of the code. This led to us not really knowing where our problems were and struggling to figure it out.

When in doubt, draw it out. We talked to Rocco about why our code wasn't working right, and he suggested we had problems with the (surprise) Odometry update. We drew out the problem, re-wrote the code based on our updates, and got a more functional piece of code out of it.

7 Future Work

If given more time to work on this project, we would try out new approaches for weighing particles. Many of our old approaches that seemed to fail might work better with a functional particle and position updater. We generated a lot of interesting (and sometimes convoluted) weighing algorithms that would be neat to try out, but alas, we ran out of time.

Another portion of our project to iterate on is the standard deviations for our random sampling. Tuning these could speed up our localization. With the tuning, comes more response to error in the environment around the robot, so examining that trade-off could be a place to go in the future.