Robot Localizer

Nick Francisci, Marissa Okoli, Ziyi Lan
Due Friday, March 3rd

The goal of our project was to tackle the robot localization problem by constructing a particle filter to determine the location of our robot in a map given an initial estimate and sensor data over time.

At a high level, our particle filter operates by repeating these steps:

1. Randomly scatter guesses (particles) about the robot's pose around an initial estimated pose
2. Compare the robot's sensor readings to what the sensors *should* read for each guess, if that guess were correct. Weight each guess according to how well the sensor readings match those hypothesized sensor readings.
3. Pick new guesses by randomly selecting an existing guess (favoring those with higher weights), then add some noise to that existing guess' heading and position
4. Estimate the robot's current position and heading as the mean across all guesses. Ideally, this would account for the weighting of the previous round of guesses. However, as of our current implementation, it does not.

Because this project was highly scaffolded, we did not make any major architectural decisions. The decisions we did make were in tuning the weighting and resampling algorithms.

First, we needed to choose a scheme for adding noise to the resampled particles. For simplicity, we chose to add noise by sampling a gaussian distribution for each attribute of the particle's pose - x, y, and heading. Though particle filters sometimes use information about the robot's motion to add a directional bias to the distributions for particle resampling, we did not see reason to do so, as the filter's performance already exceeded our expectations.

The other major design decision that paid off was to tune our weighting function using a particularly sharp drop-off, IE. to make it very aggressive in weighing not-quite-matched sensor readings poorly. In effect, this meant that only the one or two best matched particles would be used as a basis for the particle resampling. Combined with a moderate amount of noise, this allowed our robot to correct for small errors (because of the noise) but continuously lock on to only the most promising particle of the cloud.

The first challenge we faced was applying the movement of the robot to each of the particle. At first we didn't realize the robot's position and particle coordinate are in two different coordinate system. It also took us some time to make the conversion.

The other challenge we faced was finding/ tuning the weighting function. We started with making the square of the difference between the scan range point's position and the nearest map's position. Then we realized that the function is actually wrong as it should be the larger the distance is, the lower the likelihood is. Then we switched to e^-(closestDist*100). We did a lot tests to find the optimal parameter in this model.

Ideally if we had more time, we would improve the last step of our particle filter such that it took advantage of the weighting of our particle guesses in the previous step. While using the mean to estimate the robot’s pose is a useful method, we miss the chance to capitalize on the computation that we have already completed to improve our final estimate.

Our main getaway from the project is start simple and make thorough test for each step, however small it might seem, along the way. Also it’s important to prioritise some tasks over others so that the project can proceed smoothly without hindered by a particular part.