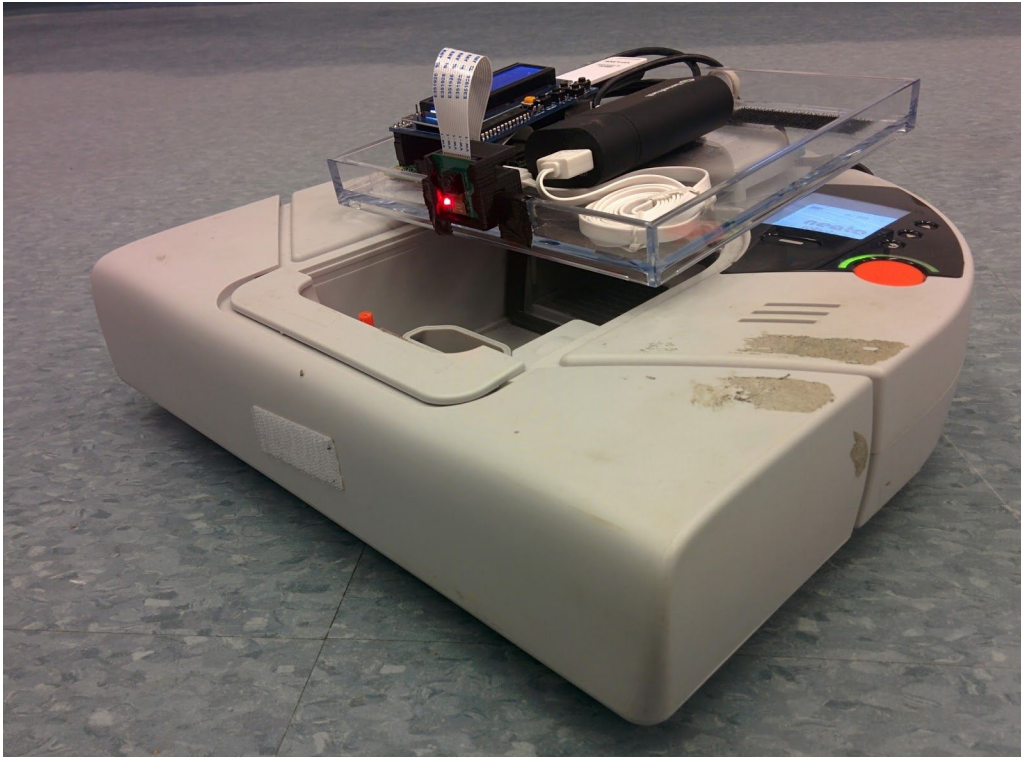# Robot Localization Project

Computational Robotics 17
Arpan Rau, Danny Wolf

Project Overview

   The robot localization project involved the construction of a particle filtering robot localization algorithm. The particle filter begins by initializing a cloud of random particles. It then weights those particles based on laser scan data to determine which particles have the highest likelihood of repersenting the robot's location. It updates the robot's pose based on those weights, and creates a new sample of particles for the next timestep in the filter. As the filter runs, it updates the pose of existing particles with odometry data.

Initialize Particle Cloud

- High Level
  - Takes in initial pose guess from RViz
  - Generates particles in normal distribution about intial pose guess
- Decisions
  - Decided upon a tight normal distribution about initial point as opposed to distribution over the map in order to get the filter to converge as rapidly as possible (versus being robust to bad initial guesses), as humans generally can guess initial pose fairly accurately

Update Particles with Odometry

- High Level
  - Computes delta between last odom and current odom
  - Adds some random error for sensor variation
  - Moves existing particles by delta
- Breakdown
  - Error normally distributed
  - Normal distribution scaled by delta as most encoder error will be proportional to distance traveled
- Decisions
  - Added encoder error here to vary particles randomly to approximate sensor error
  - Encoder error allows filter to have some slow drifting variance on particle position that is independent of resample rate (As encoder drift is dependent on distance traveled, not time, we should have some distance – dependent variance)

Update Particles with Laser

- High Level
  - Performs required coordinate transforms to get projected scan into map
  - Compares closest objects to compute likelyhood for a given particle
- Decisions
  - Fought computation time hard on this one. Ended up only sampling a few of the 360 laserscan points at each coordinate to get our filter's runtime to a reasonable number

Update Robot Pose

- High Level
  - Uses highest weight (most likely) particle to guess as to where robot pose is
- Decisions
  - Decision to use the heighest weight particle as opposed to the average of the distribution was driven by fear of multimodal distributions skewing averages to be in completely the wrong location

- Considered some kind of cluster detection, but decided not nessecary after testing showed that the highest weight particle was fairly accurate

Resample Particles

- High Level
  - Chooses based on weights from set of old particles to make new set of particles
  - Varies new particles based on normally distributed error scaled by tunable constants
- Decisions
  - Considered performing cluster detection and distributing about the centers of detected clusters (which would have minimized the effect of outlier particles) but decided against it as it seemed computationally expensive and we were already having runtime issues
  - Decided to resample based on simple normal distribution of error from weighted chosen points for computational efficiency

Normalize Particles

- High Level
  - Normalizes particle weights by grabbing weights, summing, and dividing array of weights by sum
- Decisions
  - Used numpy arrays (as opposed to a python for loop) for computational efficiency

Overall Code Structure

Our code structure overall followed the template but differed in one signifcant way. We mostly ignored the particle and particle_cloud classes, instead using computationally faster numpy arrays to store and work with our particle cloud. We translated into the template's particle_cloud and particle objects in order to use the template functions to publish our particle pose array.

Challenges

Team:

 The largest and most frustrating problem we encountered was that our likelihood model took an extremely long time to run, meaning that the filter did not update often enough to converge. We tried some tricks to reduce this computation time and brought it down from times that were utterly unworkable to merely difficult, but neither we nor the course ninjas could understand why our code was running seemingly much slower than it should have.

In a possibly related issue (possibly due to transforms publishing too slowly), Arpan's rviz refused to display anything that was not in the fixed frame. This made it very difficult to tune the filter to converge quickly.

Arpan:

Personally, I struggled with unit testing chunks of code that depended on other chunks that hadn't yet been written. For example, I could not test resample without the likelihood model in place and valid particle weights. In the future, I'll structure the way I and my team work through a project with what pieces are needed to test other pieces in mind.

Going Forward

If we had more time on this project,we'd try to use some sort of cluster-detection to make sure that outlier points did not affect the resampling process. Currently, points that are clearly outliers are occasionally chosen to be points that are resampled, and not choosing those would probably significantly speed up the convergence of our filter.

We'd also work on either finding the root cause of our runtime issues, or implementing some sort of intelligent likelihood model to avoid brute forcing through multiple laser scan points for particle (for example, we could stop working through a the laser scan points of a particle if we realize that it is very clearly nowhere near the right location).

We'd also like to find way to be robust to bad initial guesses - or, better yet, no initial pose guess from a human user. It should be possible to either initialize a ton of particles across the map at the start of a run when the robot is stationary and narrow further when a good guess to position is established.

Lessons Learned

Arpan:

This was the first python that I've had where computation time was a driver, so I took away a lot of good tricks for cutting down computation time. Numpy itself is generally a very well optimized way to do math in python (especially as opposed to iterating through lists with for loops).

I also took away a fair amount of development tips and best practices from working with Danny, who's spent a significant amount of time programming in industry. I realized that I'd only been scratching the surface of Sublime text's functionality , and that Sublime will do things like flag things that aren't PEP 8. I also learned some more git best practices and got much better at merging conflicts as Danny and I worked on the same script at once.

Github
https://github.com/wolfd/robot_localization_2017