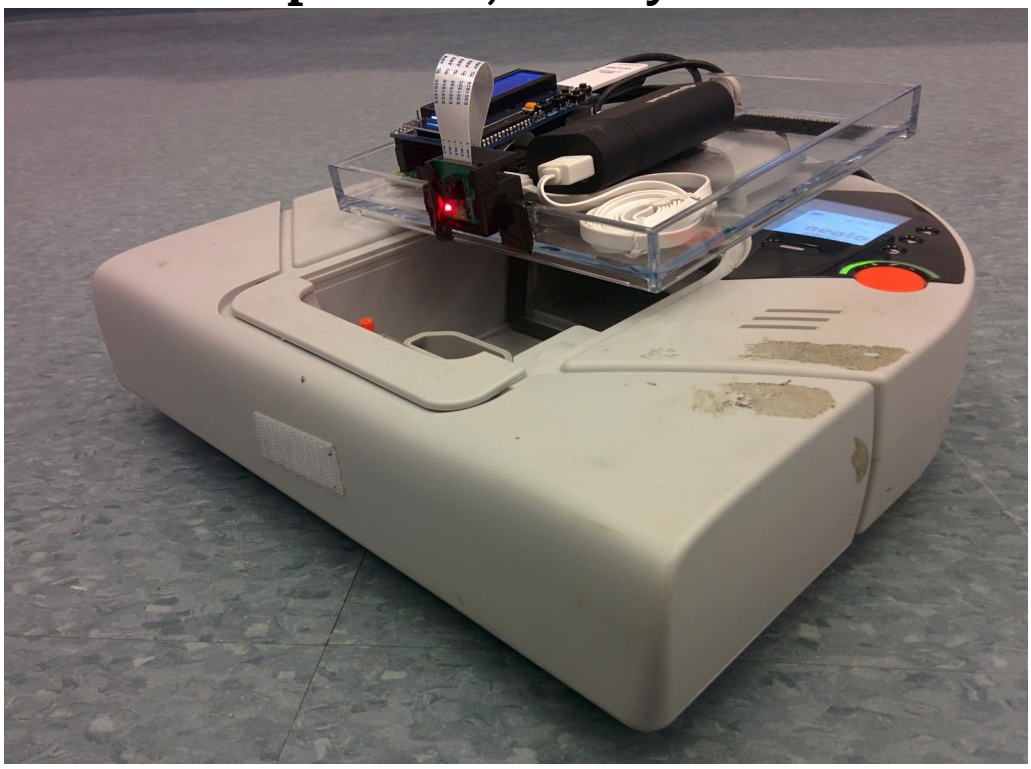
# Robot Localization Project

## 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 representing 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.

## High-Level Software Breakdown

### Initialize Particle Cloud

- High Level
    - Takes in initial pose guess from RViz
    - Generates particles in normal distribution about initial 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)
    - Ideally we would measure encoder drift and use measured values in code, here we just approximated constants that made the filter converge correctly

### Update Particles with Laser

- High Level
    - Transforms the laser scan ranges to the coordinate frames of each of the particles
    - For each laser scan point, see how close to an obstacle it is, sum and weight closeness to a number of obstacles higher
- Decisions
    - Used numpy to do transformations
    - Used a slice of the total particles (use 1/10th of the available) to save computation time
    - Throw out 0 range readings (rangefinder sees nothing) so we don't accidentally follow walls

### Update Robot Pose

- High Level
    - Uses highest weight (most likely) particle to guess as to where robot pose is
- Decisions
    - Decision to use the highest 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 necessary 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 significant 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.

We also used markers instead of a pose array to visualize the particle cloud, allowing us to visualize the particle weights as colors and helping significantly with debug.

We included several pieces of functionality just for debug, including the ability to publish the laser scan of an arbitrary particle, which was very helpful in debugging our likelyhood function.

## 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 (maybe 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 and served as yet another reminder that good visualization is key to robotics.

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.

Danny:

The biggest struggles was in the debug cycle. It was a lengthy process to wait for the bag file to start up and the OccupancyField to initialize before doing any tests. I considered finding a way to keep the OccupancyField loaded (maybe load it from a file?), but never got to that point. I used "import pdb; pdb.set_trace()" as much as I could to fix numpy bugs, as having initialized data was super useful. I also added a bunch of extra visualizations to try to hunt down bugs at the last hour, but it wasn't enough to fix them all.

## 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.

We'd probably also spend some time setting up dynamic reconfigure and tuning our constants so that the filter performs more consistently. While it does currently track the robot, it is not very robust and can fail during certain manuvers with certain wall geometries.

## 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), so learning to use it a bit better was a big plus.

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.

Lastly, I learned that using small python scripts to test things about large arrays pulled out of print debug is a great way to work out what's going on inside your code.

Danny:

I figured, given my experience in Bayes things, that this project would be easier than it actually turned out to be. I spent a lot of time fighting numpy, as I'm not that experienced with it, and I've learned a lot of cool tricks that I've forgotten about or never used before. I also had a good time trying to make the numpy/Particle interface work well, but there would be a lot more I would do if I had more time. I would have refactored a lot of the code from the ParticleFilter class into the ParticleCloud and Particle classes, especially code that deals with specific indexes of the array. I left it in a state where it would be relatively easy to refactor, but didn't do the refactor.

In the future, I will probably get started earlier on the project. Although we got an impressive amount of work done at the end, it would have been much nicer to spend the same amount of time spread out, and had more time to work through the bugs and problems with ninjas and Paul.

## Video of filter in action

https://www.youtube.com/watch?v=hFfVMKnZfdo

## Github

https://github.com/wolfd/robot_localization_2017