Samuel Breck, Gavin Bains, Bryce Check

CSCI 445, Fall 2018  
December 09, 2018

**Final Project Writeup**

# **Algorithm Design**

**Localization of the Robot and Navigating to Goal**

In order to localize the robot, we made use of the particle filtering algorithm to first get an estimation of our location on the map before we navigated to the goal location. We were able to make use of our implementation of the particle filtering algorithm from lab9 in order to quickly localize the robot. We used 1200 particles, 0.10 translation variance and 0.20 rotational variance for the odometry, and 0.1 measurement variance for the sonar for our final particle filter.

We modified the autonomous localization by making use of a simple turn-left condition when the robot approached a wall and was within a 0.50 M distance. Otherwise the robot would originally continue to move forward and constantly do the measuring part of the particle filtering algorithm. We noticed that there were points on the map that would cause our localization algorithm to either take a long time or to never finish at all because the robot would just turn in circles and the form two or more clusters of particles. After some experimenting, we found that if the robot ever did the move forward-measure action 3 times in a row, them the robot would look 90 degrees left, take a particle filter measurement and then 90 degrees right and measure again. This was able to get the robot out of more positions, but not quite all of the positions on the map. Once there is a 0.05 variance or less within the x and y coordinates of all the particle filters, we finished localization and moved onto navigation towards the goal.

In order to navigate towards the goal, we made use of the RRT algorithm that we implemented in lab11. The tricky part of this integration was similar to that of lab11 in that we had to transform the points that we estimate from the particle filter into the points that the RRT uses by transforming x by x \* 100 and y by taking the abs(3 – y) \* 100. We also had to update the PIDTheta controller, which we learned how to use for error-correction and to realize when we are close enough to a goal, with calculated discrepancies between the first and second waypoints returned from the RRT. We also had to implement a function called smooth\_waypoints() that removes the first couple of waypoints due to issues skipping waypoints in the beginning due to error in the initial location. After smoothing the waypoints, we then calculated the angle the robot needed to turn in order to face towards the first waypoint. Once the robot rotated at the localized point, we were able to plug it in as the starting point and the point near the arm as the end point.

**Robot Arm, Cup to Shelf**

In order to integrate this part of the project, we made use of PA2’s implementation of inverse kinematics. As with the rest of the lab code we worked with, there are slight modifications to the actual design of the inverse kinematics. For example, between the pickup of the cup and actually dropping it off of the shelf, there are two different coordinate planes we are working with a x-y and a z-x. When picking up the cup, the y is on the right side of the x, which means we have to transform the coordinates to work with a negative x. However, when we lift up the cup, we have to switch planes to the second coordinate plane with the z-x axis. In addition to this, when we actually place the cup on the shelf, we have to work with all three axes when trying to put the cup on the shelf. Unfortunately we were not fully able to account for variations of the robots position perpendicular to the robotic arms “reaching vector.”

**Algorithm Output**

Localization and Path Finding Inputs

Here is a list of the different gains / thresholds that we had to manipulate in order to achieve the best possible performance.

# PID Inputs

self.pidTheta = pid\_controller.PIDController(200, 0, 100, [-10, 10], [-50, 50], is\_angle=True)

# Particle Filter Inputs

self.pf = particle\_filter.ParticleFilter(self.particle\_map, 1200, 0.10, 0.20, 0.1)

# Turning Left Threshodl

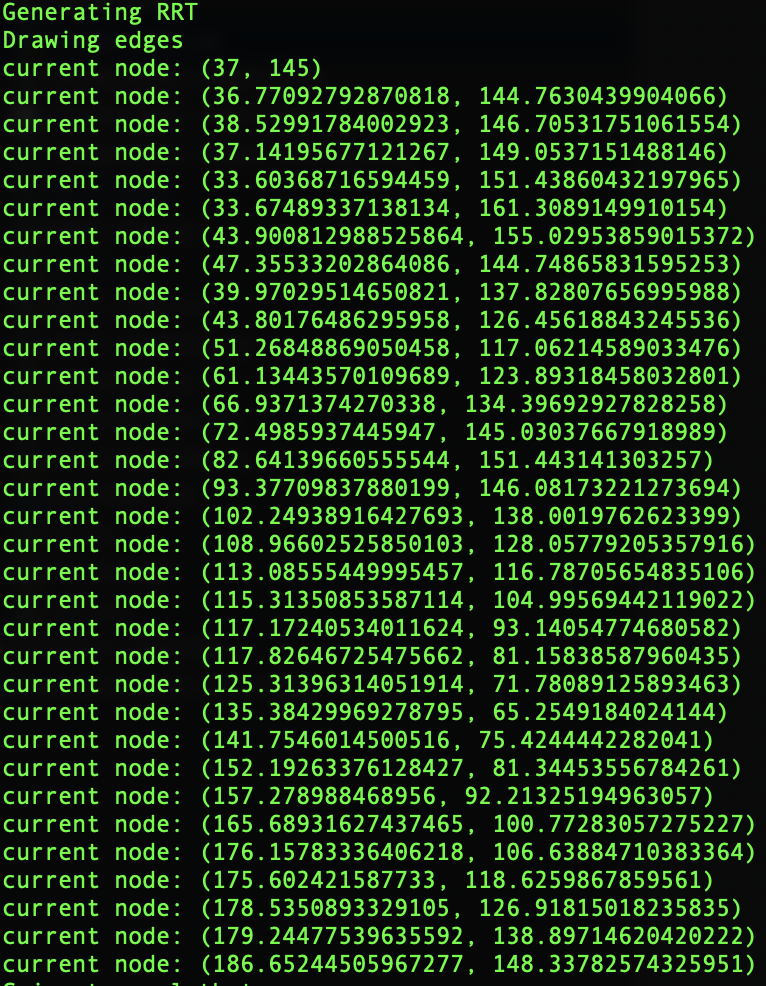
if self.sonar.get\_distance() < 0.45:

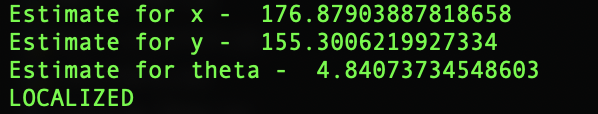
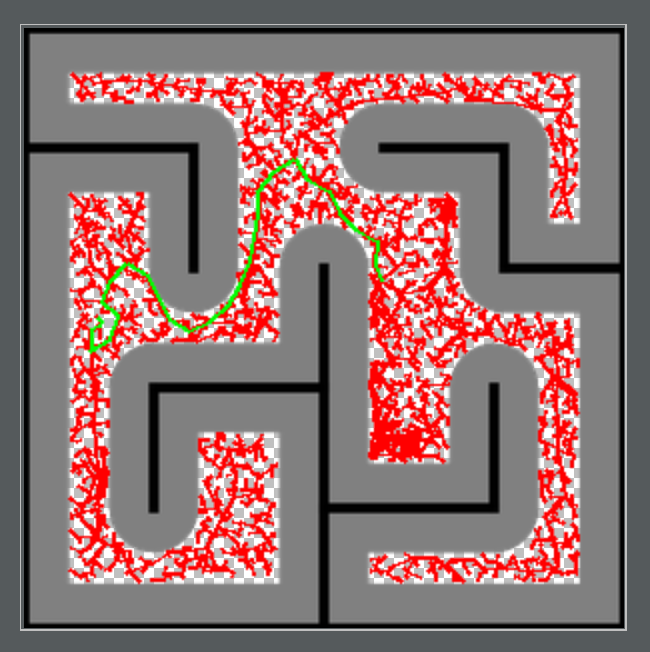
# Localization Threshold

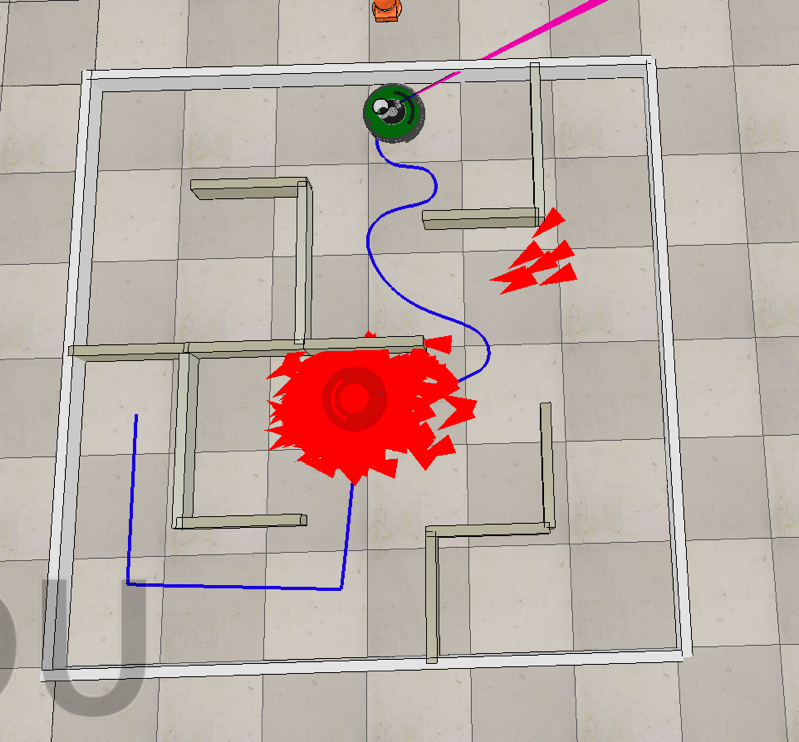
if x\_variance <= 0.05 and y\_variance <= 0.05 and theta\_coord <= 0.05:

# Path following Threshold

if abs(goal\_x - self.odometry.x) <= .1 and abs(goal\_y - self.odometry.y) <= .1:

Localization and Path Finding Outputs





Robot Arm, Cup to Shelf Inputs

# **Why?**

The reason why we chose to go with our implementation for the robot localization and the navigation had to do a lot with trial and error during the process of finding out the different gains and thresholds. There were a lot of different trial runs that we conducted in order to fine tune for the localization of the robot. In addition to this, we tried a bunch of different smoothing techniques for the RRT and decided that it was easiest to tighten the threshold and just get rid of the first couple of points. We tried eliminating every other point, checking the surroundings for obstacles, and a combination of the two. A lot of the process of design for localization and navigation processes ultimately involved a lot of trial and error.

Other important design decisions lay in understanding the role of the variance of the translational and rotational odometry as well as the measurement variance of the sonar. Once we tightened the sonar and rotational variance, the particle filter was better able to guess its orientation, that way once it took a sonar and made a particle filter estimate, the results were much closer to where the robot actually was. This additionally helped to speed up the localization portion of our navigation algorithm.