Mike Warner
Shane Skikne
Mary Ruthven

Mobile Robotics: Scaffolded Project Level 2

We made a particle filter using the level 2 scaffolded code that was provided to us. This particle filter uses laser scans and odometry readings to locate the robot on a given map. It uses an initial guess to create a particle cloud. When the robot moves, the particle filter uses odometry readings to update the cloud moving the particles to the new locations. It then updates the particles' weights by comparing the expected laser readings and the actual readings. They're updated using bayesian theory. It uses these new weights to create a best guess for where the robot is now.

We used the structure given to us in the base code. We split up all of the classes into individual files to make the code more manageable. To approach the code, as a group we talked through all of the code that we were supposed to implement. We decided what to implement and a high level of how to implement it.

initialize_particle_cloud - gaussian distribution around last pose, this only happens at the very beginning
update_particles_with_odom - use trigonometry to move distribution according to odom data; add gaussian noise to readings
update_particles_with_laser - use bayesian to change probabilities of particle cloud based off closest distance calculation
normalize_particles - make all probabilities in cloud add to 1; summing all particle weights and divide each weight by that sum
resample_particles - take all particles and sample based on weights using the helper function
update_robot_pose - average particle distances based on weight to come up with best guess

We made test functions independent of ROS to test the individual and overall behavior of the functions we implemented. This helped us test the algorithm without having to deal with the difficulty of debugging the basics in the real world.

There is noise incorporated into three of the functions we built. The noise is a normal distribution with some standard deviation. We looked at the three standard deviations that we would have to choose. The three are for the initial particle cloud, updating the probabilities based on the laser, and updating the cloud from the odometry. We decided the standard deviation for the initial cloud would be small because we know the initial location pretty well because we control it. We experimentally measured the standard deviation of the LIDAR by placing the neato at some distance from the wall. We then took a large number of readings and looked at the standard deviation of the readings. We saw that the

standard deviation was somewhere around .005 m for any distance below 1.5 m. When the robot was more than 1.5 m away from the wall, the distribution became skewed and the standard deviation increased by an order of magnitude. Using this data, we decided that the standard deviation for the laser readings should be .005m. We did not figure out a test to find the standard deviation for the odometry. We guessed that it would be around 1 cm. All of these standard deviations are for x and y.

Our biggest challenge was that when we found that the odometry theta did not match up very well with the how much the robot was actually turning. We were getting spurious values and spent much time devoted to pinning down the bug. Eventually we discovered that it was because we were taking the mean of our x, y, and theta values independently based on their weights. Our weights were assigned to the particles based on the accuracy of their position, not orientation. Thus, we were getting a random mean theta from our particle cloud. Our solution was to set our theta to be equal to our calculated theta taken from our get_closet_obstacle_distance. Because our get_closest_obstacle_distance finds the closest obstacle from that position, we can easily calculate the theta between the obstacle and the object. This theta is then set to be our robot Pose.

We also had some trouble testing the neato in the real world. We had challenges figuring out how the neato's placement in the real world translated to our map. Specifically with where 0, 0 was where the angle 0 on the robot was. When our code was not working it was difficult to know exactly how it was failing when we did not know exactly how it was supposed to be interacting with the environment. In the future, it might be helpful to identify angle 0 on the robot right when we start testing, and draw out a rough sketch of the map marking 0, 0 to make sure that we understand the environment a bit more.

In the future if we continued working on the project, we would try to get the system working more reliably by characterizing the standard deviation of the odometry system and figuring out why the turning in the odometry system is unreliable.

One interesting takeaway we had from this project was learning to control our environment. Instead of building a map of a real room, the walls of which might be out of range of the LIDAR, contain random objects that throw off our data, and have difficult geometries to map, we simply laid down three tables against a wall to create a small box for the robot. This allowed us to quickly make a map of the simple square and also debug our robot's position very easily.

Another interesting lesson we learned was that real-world measurements is difficult. Code working in simulation on your laptop is one thing, but code on a Neato with imperfect LIDAR measurements and imperfect friction coefficients on the wheels is another.

Finally, our biggest lesson learned was related to planning. We all had trouble wrapping our heads around the Particle Filter algorithm in the beginning. We assigned functions to each other but never had

a conversation about what the algorithm is doing at a high level and, as a result, had trouble visualizing how our independent functions interact. We had a meeting a week into the project and finally discussed what was going on at a high level. Only then were we each confident enough to move forward. Next time, this meeting will happen at the very beginning of the project.