Marena Richardson
9/22/14

Warmup Project Writeup

For this project, I implemented the wall following and object avoidance behaviors. I implemented the wall following by finding the closest distance in the list of laser range finder measurements. I also found the angle of that distance relative to the neato. I then sent a velocity message with a linear component of 0.2 and a proportionally controlled angular component. The proportional control worked by taking the current angle of the wall and subtracting the closest parallel direction (90 degrees or 270 degrees) and multiplying that by an appropriate factor to generate a velocity. To implement the object avoidance, I created a list of the values of the laser range finder measurements straight ahead of the robot and took their average. When the average value straight ahead was less than 1.3, I checked the robot's right (270 degrees) and left (90 degrees) to see which one was closer to an obstacle. I set a direction variable to 1 or -1 such that the robot would always turn to avoid the obstacle straight in front of it towards the direction that had more space. I then sent a velocity message equal to this direction variable multiplied by an angular speed about the z axis.

The finite state controller was used to switch between wall following and obstacle avoidance. I mentioned this in the paragraph above, but I programmed the neato such that it would follow the wall until an obstacle appeared in front of it at a distance closer than 1.3 meters. At that point, the obstacle avoidance state would turn on and would turn off again once the object was no longer less than 1.3 meters in front of the robot.

I used a class named Wall_Follow to hold my code. (I implemented wall following first and never changed the class name). This parent class holds three methods: init, scan_received, and main. Init is the initialization method that runs whenever the class is instantiated. Scan_received is the callback for the laser scan subscriber and does all of the processing of the measurements in order to set the variables such as distance_to_wall and angle_of_wall appropriately. It also sets the finite state control variable. Main is the function that sets up the publisher and subscriber and publishes velocity messages to the neato based on the state the robot is in.

I had a little bit of trouble figuring out how to make my finite state controller run so that I wasn't overwriting my own velocity messages. I also didn't initially know that a measurement's position in the ranges array was its angle. That was a key insight. I didn't encounter too many problems during this project, however.

If I had more time, I would probably implement some sort of proportional control with the obstacle avoidance as well. It works as is, but it turns on suddenly and turns off before it has turned a full 90 degrees. It then goes forward until it is about to run into the object again at a different angle and completes the turn. This is less than ideal. I would love to go back in and smooth out those transitions so that the motion of the neato is more continuous.

In this project, I learned that it is important to be aware of how your code is changing between states. If you are sloppy about the transitions between states, you can end up overwriting your messages. I also learned that proportional control is an excellent tool for making the robot's motion appear smooth. Overall, I had fun working on this project, and I feel much more comfortable with running code from ROS packages on the neato than I did at the beginning of the assignment.