Shane Skikne
9/25/2014
CompRobo

I implemented a Python class called wallFollower, that is able to find a wall and then follow it while maintaining a set distance from the wall. The robot starts by looking for a wall by checking the laser scan for any set of consecutive angles that see an object. This object could obviously be many other things that aren't a wall, but for this first assignment, it is assumed that there are only walls to keep things simple. Once the program identifies a wall, the robot then moves perpendicularly to or from the wall until it reaches a set distance from the wall. The robot will then rotate until it's heading is parallel to the wall. The robot is then able to move forward following the wall using two measurements, the closest distance to the wall and distance to the wall at two other, opposite angles.

To get ready to actually follow the wall, I use only proportional control, because the speed at which the robot moves or rotates is proportional to how far the robot is from its desired wall distance or orientation. The system works well enough, because the robot doesn't overshoot the desired angle or distance. Once the wall is found, I use proportional control and something I would consider derivative control lite. The speed is set, while the angle the robot is travelling at is controlled to get the robot to maintain a set distance from the wall. The proportional control component of the wall following simply adjusts the angle based on how close or how far we are from wall. This worked effectively at keeping the robot from completely leaving the path, but often just made the robot weave from side to side and never quite settle on the line it was supposed to be following. To combat this issue, I also check the distance to the wall at 45 degrees and 135 degrees (90 degrees is pointing directly at the wall when the robot is travelling parallel to the wall). Comparing these two values gives us an idea of how the robot is oriented towards the wall, which I use to continually make the robot more parallel to the wall while it is correcting the distance.

One of the major challenges was transferring the code from theoretical, functional code to the simulation and then from the simulation to the real robot. A lot of assumptions I made when writing code assumed that the world would be perfect when the code was run. This was not even true when I ran the simulation and was so far from true in the real world. One large example was data sampling. When the robot was not even moving, the laser scans could still vary substantially. While the scans were usually pretty accurate, there were often some small inconsistencies. Once I tested the robot in the real world, sometimes readings just returned 0.0. When these inconsistencies are expected and handled well, they don't do much damage, but before I understood what was going on, they caused a lot of confusion. I don't have much in place to handle readings that are not 0.0, but aren't accurate, but ignoring all readings that were simply 0.0 did a lot to make the robot more effective at wall following.

My code was structured in a Python class with two types of functions besides __init__. Some of the functions handled reading the laser scan and storing the relevant data from it. The most important piece of information was always the closest point to the wall, but these functions would also determine

when the wall was actually found and store all the scan data in case the other set of functions wanted to use it. The other set of functions controlled the movement of the robot based on the data stored by the scanning functions. The main function, motionController calls the different motion functions, one by one. The motion functions help find the wall, turn the robot to a set angle, move the robot to a set distance from the wall and drive the robot as it follows the wall.

For the finite state controller, I had four separate behaviors and booleans to determine when to switch between them. The different behaviors were finding the wall, rotating the robot to a set angle, moving the robot until it was a set distance from the wall and following the wall. During the first behavior, finding the wall, the robot is scanning around looking for a wall and driving forward if it can't find one. It knows it has found a wall when it sees something at 7 consecutive angle readings.Once the wall is found, the robot then turns to face the wall before it moves to a set distance from the wall. To turn to the correct angle, the robot spins proportional to the angle reading the closest point to the wall until that point is at 0 (and the robot is facing the wall). To move to the correct distance, the robot once again moves proportionally to the difference between the desired distance to the wall and the distance to the closest piece of the wall.It knows it has reached the correct distance when the distance to the closest point is within .01 of the desired distance. The robot will then turn 90 degrees again to be facing parallel to the wall. While the robot is following the wall, it will drive at a continuous speed and change the direction to ensure it is staying a set distance from the wall.

I learned many different lessons for future projects. One of the biggest is to not write all the code at once. While this strategy is useful in all programming, it is especially relevant to robotics, because all the pieces don't always work together as you'd expect. For example, once I realized that the data from the scans was not as reliable as I had hoped, I had to edit code and make it more robust. Had I been testing continuously, I would have realized I would be hitting these issues and I could write the code better from the start. Another one of my big takeaways is that often, 80% of the functionality can be written in about 20% of the time. I could very quickly get the robot to do a pretty good job and now getting it do a very good job is where I'd need to be a lot more careful. My takeaway is not that I should always slack off in robotics, but rather that realizing how perfect each piece of a project needs to be in order for entire project to work is an important consideration.

Note: I have done some object avoiding, too, but was not at a point I wanted to hand in and I had done over 10 hours on this.