CompRobo Warmup Project

Rachel Boy

Overview

I implemented two behaviors: obstacle avoidance and wall following. The wall following code could successfully follow a wall, maintaining a specified distance to a specified degree of accuracy, and navigate most right angle corners (it could always navigate exterior corners, and succeeded at interior corners for a very specific range of distances). The obstacle avoidance behavior successfully avoided running in to almost all objects - it only managed to avoid small chair legs about half of the time, I believe because they did not appear on the LIDAR. The wall following code implemented finite state control to handle both maintaining distance from the wall and following the wall.

The Algorithms

Wall Following

To follow a wall at a specified distance, with a specified margin of error, I used finite state control as shown below, with transitions specified in the caption.

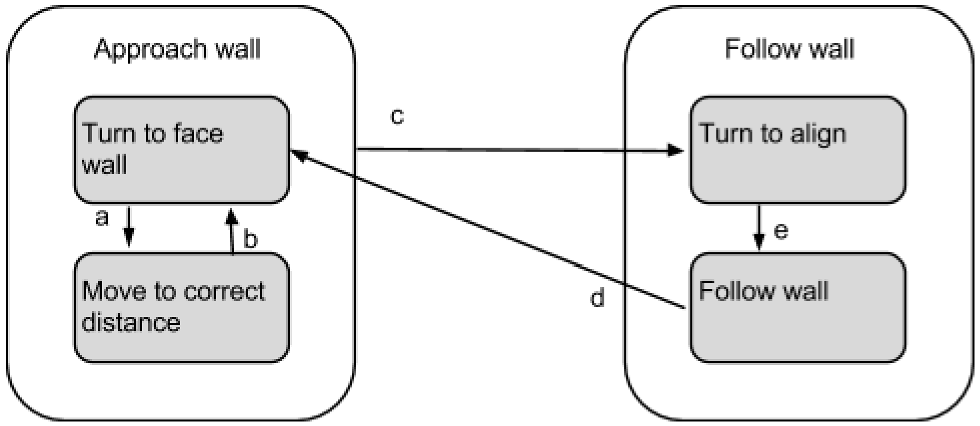

Figure 1: logical control for wall following

a) When the nearest object within 5 degrees of directly in front of the robot
b) When the nearest object is not within 5 degrees of directly in front of the robot
c) When the nearest object is at the specified distance (plus or minus a small margin of error) from the robot
d) When the nearest object is not at the specified distance (plus or minus a moderate margin of error) from the robot
e) When the nearest object is at 90 degrees to the robot

Each of the behaviors involves fairly simple proportional control.

To turn to face the wall, the robot compares the angle to the nearest object to 0 degrees, then uses proportional control on the rotation of the robot to turn in place to face the wall.

To move to the correct distance, the robot compares the distance to the nearest object and the desired distance, the uses proportional control on the forward/backward motion to achieve the desired distance.

To turn to align, the robot uses exactly the same logic as facing the wall, but aims for 90 degrees.

To follow the wall, the robot compares the angle to the nearest object to 90 degrees, and uses this for proportional control on both the speed and the turning to stay aligned to the wall.

Object Avoidance

To avoid obstacles, I look at all of the lidar data from directly in front of the robot, and transform it to cartesian coordinates. If there is anything within a set distance in front of the robot on its left side, it turns right in place until there's nothing there anymore. If there's anything in front of the robot on its right side (but not on the left side), it turns until its front is clear again. If its path is clear, it goes straight.

Code Structure

Both behaviors were entirely encapsulated in a single object. The publisher, subscriber, and any variables shared across multiple functions are attributes of this object. The callback function for the subscriber to the scan topic is a method of the parent object that determines the desired speed and turn rate based on the latest information. The parent object also has a main loop which publishes commands to the robot at a controlled rate. The code can be read in obstacleavoid.py and wallfollow.py.

Challenges, Lessons Learned, and Next Steps

Debugging with the actual robot was a challenge, because it was hard to figure out what was sensor error and what was logic error - the simulator helped a lot with this (except for when I left it running while testing on the actual robot. Then it was very, very bad.)

It turned out that small objects (like chair legs) were very hard to see with the LIDAR - using the camera might be necessary for that. Also, while the LIDAr is only slightly off the center, it is still a fairly large distance on the scale of the robot, and correcting for it can be important to some algorithms.

There are a huge number of ways I'd like to improve my project. A short list includes:

- Wall follow: Transform the LIDAR data to the center of the robot (to better handle interior corners)
- Wall follow: integrate the maintaining distance with the wall following so that it doesn't have to totally change states to stay at an appropriate distance
- Wall follow: (Possibly) work out some sort of check for continuity so that wall follow actually follows walls, rather than whatever objects happen to be around (whether this would actually be an improvement depends on how you frame the problem)
- Wall follow: I really wanted to switch this to more elegant finite state control with smach, but the opportunity cost of time spent coding got higher than the benefits from being cooler...

- Obstacle Avoid: Keep track of the odometry frame and have the robot actually navigate towards a given point in direction when there’s no object in front of it, rather than just going straight until the next object.
- Obstacle Avoid: Actively identify objects through clustering and plan paths to avoid them, rather than simply not running into objects it sees