Computational Robotics
Warmup Project Writeup

Jack Fan

Which behaviors did you implement?

I implemented wall follower and obstacle avoidance.

For each behavior, what strategy did you use to implement the behavior?

For my wall follower, I initially tried to implement PD control but was unsuccessful(wall_follower.py). Then I had a discussion with Jimmy Wu, who provided me with a way simpler implementation, which resulted in wall_follower-2.py. In this more successful version, the neato would identify the distance between itself and the wall right in front of and then move towards the wall until a set distance. It would then turn clockwise and start moving after rotating for approximately 90°. By constantly trying to make the difference between the distance to the wall at 90+x° and 90-x° zero, the neato would move parallel to the wall.

As for obstacle avoidance, my program calculates the distance average of the front area. The neato would keep going straight until it detects that an obstacle is within a certain radius. My program then averages the distance to the front left and the front right and decides whether to go left or right. If the obstacle gets too close and hits the danger radius, the neato would back off and recalculate and decide whether it should go left or right.

For the finite state controller, which behaviors did you combine and how did you detect when to transition between behaviors?

There are three states to the wall follower. First, find distance to the wall in front of it; second, rotate clockwise 90°; finally, keep moving along the wall. The transition between the first and the second is the distance: when distance is less than a given value, the program transitions into the second state(while loop). The neato would keep rotating clockwise until the 90° and 90±x° distance to the wall are nonzero. Then it would transition into the final state where it modifies the direction such that the 90+x° and 90-x° distance are the same(extremely close), making the robot move parallel to the wall.

There are also three states to the obstacle avoidance. First, the program would check if the neato is too close to any obstacles by checking whether the average distance in front of it is less than a certain value. If so, it would back off. It would keep doing so until it's out of the "danger zone." Then it would transition into the second state where it calculates the average distance on the front left and the average distance on the front right and decide whether to go left or right, given that these two values aren't zero, meaning "infinite" distance. In the last state, when the neato is not in the "danger zone" and does not have any potential obstacles on the front right or front left, it would move forward.

How did you structure your code?

I tried to think from the stand point of the neato. I always try to figure out the logic sequence in the beginning. This was particularly important for the obstacle avoidance since there are priorities to the algorithm. I felt that for obstacle avoidance the priority was to not run into objects, hence I started with the "in-danger-zone-please-back-off." After that, there are several things to do before the neato is positive that it can move forward. As for the wall follower, due to the restriction(the neato has to start facing a wall), the neato kind of performs sequentially rather than changing its behavior logically, except for the last state where it constantly calibrates to move in parallel. In a nut shell, I structured my code such that either the priorities or the action sequence are clear.

What if any challenges did you face along the way?

My biggest challenge was, of course, that my simulator, Gazebo, didn't work for the entire duration of the project. I didn't get to play with it until after the project is due. However, I realized that it is MUCH MUCH easier to get things working in gazebo. In reality the biggest issue for me was the quality and the blindspots of the lidar. I had to algorithmically deal with the values that didn't make sense. In addition, for the obstacle avoider, my program was having a hard time identifying thin chair legs and table legs. I currently can't think of a way to implement thin pillar detection without significantly increasing computation.

One thing for the wall follower is the fact that the lidar's angle aren't accurate. For example, when I tried to find the distance at 90° using /scan/ranges[90], the distance I was getting wasn't exactly at 90°. This caused a huge problem when I tried to calibrate the neato's direction by making the distances at $90 \pm x°$ equal.

What would you do to improve your project if you had more time?

I would try and do the full PID control that I failed in the beginning for the wall follower and come up with a robust thin pillar detection for the obstacle avoidance.

Did you learn any interesting lessons for future robotic programming projects?

Yes. GAZEBO IS USEFUL; MAKE SURE IT WORKS. Test your code in reality when you feel that you have achieved something rather than finishing the entire program in simulation. Constantly adjusting to reality is a very important part of robotics. After all, robitics is a very practical and application heavy field.