Warmup Project Writeup

Claire Beery and Jeff Pflueger

January 2017

1 Introduction

This is the Warmup Project documentation for Claire Beery and Jeff Pflueger. The aim of this project was to get used to using ROS, writing ROS nodes, and using the Neato robot platform. The following are descriptions of our implementation of robot functions. All code can be found at `https://github.com/cebeery/warmup_project_2017`.

2 Wall-Follow

Wall Following seemed like a very complex and challenging task to start. We ended up breaking it into two distinct steps: finding the wall and following the wall. The robot divides these tasks into several states, shown in Figure 1:

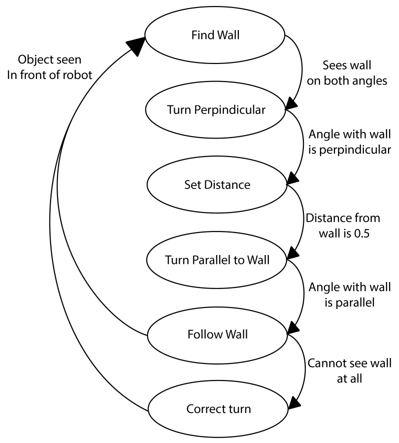

Figure 1: State diagram for the Wall Following node.

- $FindWall$: The robot looks for the wall at two LIDAR angles, ±10. It moves forward until it detects the wall on one or both angles. If only a single angle is seen it rotates toward the side of that angle until it detects an object at both angles. Then, using the Law of Cosines, the robot calculates its approximate angle to the wall and turns with a proportionate speed to that angle. The geometry is shown in Figure 2 on the next page. When its angle becomes approximately 0 (is within ± 1 deg of 0), it is declared perpendicular to the wall. The robot then moves into the $SetDistance$ state.

- $SetDistance$: The robot then moves with a bang-bang controller. It moves forward if too far and backward if too close. When it is within a certain threshold of 0.5 meters, the robot transitions into the $TurnParallel$ state.
- $TurnParallel$: The robot then turns for 3 seconds to the left and then uses two angles at 270 ± 10 and the same method as Finding to orient itself parallel to the wall. When the robot is parallel, it transitions into the $FollowWall$ state.
- $FollowWall$: The robot then moves forward, maintaining its angle with the wall, until it sees something in front of it. It then moves back to the $Finding$ state and starts over to orient itself to the new wall.
- *CorrectDistance* : The robot turns 90 clockwise. This is in case the robot stops seeing the wall at a corner. It will then move into the $FindWall$ state and start the process over.

As time went on, we realized that there were a lot of edge (heh, get it?) cases to wall following. What does the robot do at corners? How does the robot deal with it when the wall suddenly goes away? We implemented the *CorrectDistance* state to combat these problems. The $FollowWall$ state will trigger *CorrectDistance* if it stops seeing the wall to its right. This will turn the robot in an attempt to re-orient itself with the wall.

3 Person-Follow

The objective of the Person Follow node is for the robot to recognize a body in front of it, and follow that bodies movement. Figure 3 on page 4 shows the region the robot checks for readings. It then averages the locations of the readings and represents them with the marker COM (Center Of Mass). The robot uses two proportional controllers to approach the COM; One for the distance between the robot and COM, One for the angle between the robot and COM. Early on in the development, we had problems with the robot seeing the wall and not being able to escape running into it. We soft-solved this problem by limiting the distance the robot can detect values for the COM.

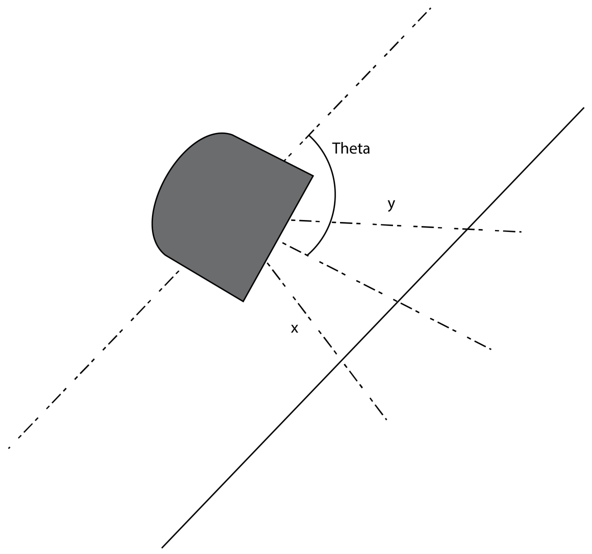

Figure 2: Geometry of angle finding towards wall. Objective is to minimize Theta

4 Obstacle Avoidance

The Obstacle Avoid node attempts to do just that: avoid obstacles. To do this, it looks at a semi circle in front of it, shown in Figure 4 on page 5. Any value that the robot sees, it creates a force for based on its position relative to the robot. The node then adds up all of the forces it sees, and averages them together. The robot than bases its movement on that net force.

The obstacle avoid node separated into 2 states, Active and Passive avoidance. Transitions are detailed in Figure 5 on page 5. These names are misnomers. Active refers to when the robot takes action due to a specific edge case. Passive refers to when the robot is in its normal obstacle avoidance state.

Obstacle code is on a seperate branch of the git repository.

- *Active* : Active avoidance only occurs when the robot sees something directly in front of it. The robot immediately moves backward and turns in an effort to get unstuck.
- *Passive* : Passive avoidance takes a look at the LIDAR data in front of it and treats readings as individual forces. It averages the forces together and looks at the direction and magnitude of its vector. Then, the robot tries to move in the opposite direction.

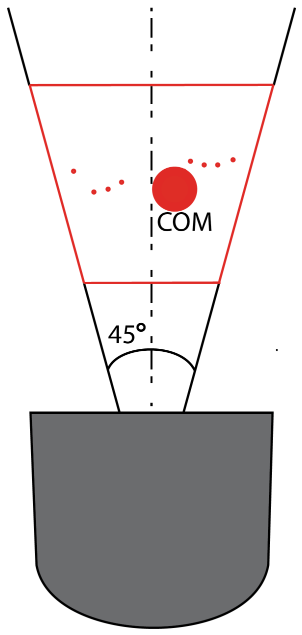

Figure 3: Bounding region where robot looks for person to follow. COM represents the center of mass of the LIDAR readings in that region

5 Finite State Machine

For our finite state machine, we decided to combine the wall-follower node and the person-follower node. It takes the finite state machine for both of the nodes, and combines them with a *Transition* state between them. The robot starts in the *FindWall* state, and transitions through to the *FollowWall* state, where it will check to see if it sees a person on its left. If the robot sees someone, it will transition into following them. If not, It will continue to follow the wall. The *FollowPerson* state is similar. The robot will follow a person around until it no longer can see anybody. At this point, it transitions into the *FindWall* state. The full state machine is seen in Figure 6 on page 6.

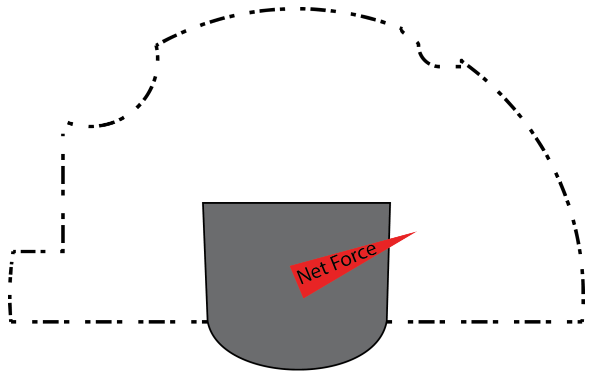

Figure 4: Diagram of how the robot looks for obstacles to avoid. Indentations are obstacles. Net Force shows the magnitude and direction the robot is compelled to move

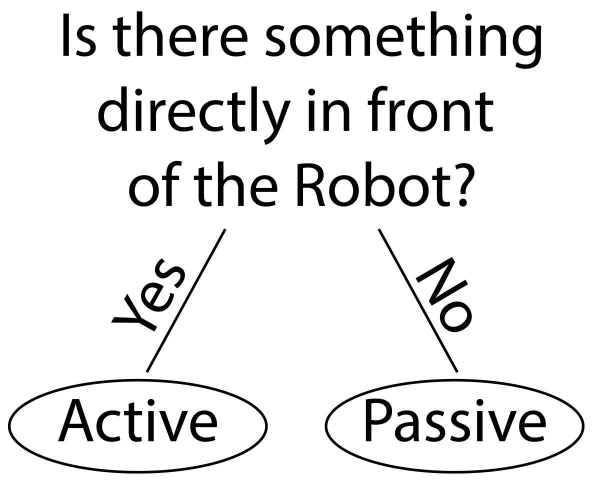

Figure 5: Details the states in the obstacle avoid node: Active and Passive

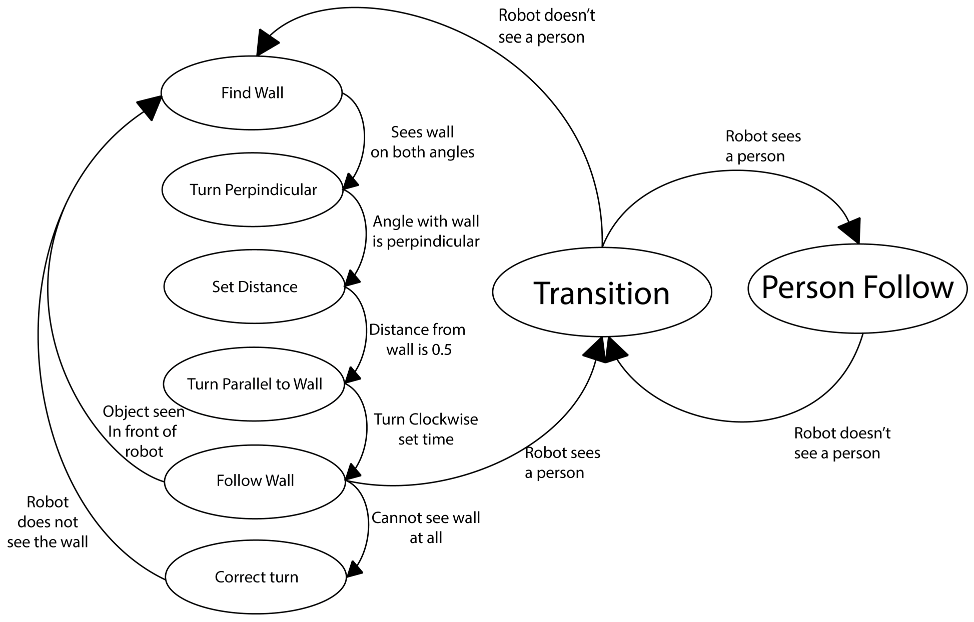

Figure 6: States and transitions of the Wall and Person Follower

6 Future Work

If we were continuing this project, we would primarily focus on improving our obstacle avoid function. At the moment, it is forced around by objects that it sees and this can run into problems when the robot sees something in front of it. Instead, searching for the largest, open space might be the better way to run the program. Instead of getting caught by objects, the robot would try to find holes it can fit through. In addition, we would work on making the person following program less prone to seeing walls. When it sees a wall, the robot gets stuck because it sees a large Center Of Mass in front of it. To fix this, we would implement movement detection as well. It would see if an object is moving by looking at the odometry data, and prioritize following those objects instead of non-moving ones.