ENPM809E Final Project

Brian Bock

Summer 2020

Table of Contents

1 Project Description

The task is to drive a Turtlebot through a Gazebo maze (Figure 1) using a wall following algorithm. The robot can utilize a right hand or left hand wall following algorithm, but does not need to be capable of both. Project requirements prohibit the use of SLAM (Simultaneous Localization and Mapping), which would traditionally be used for a task like this. The robot is equipped with a rotating LiDAR sensor, which can record distances in single degree increments around the robot. The laser's range exceeds the bounds of the maze, so laser range is never an issue for this problem.

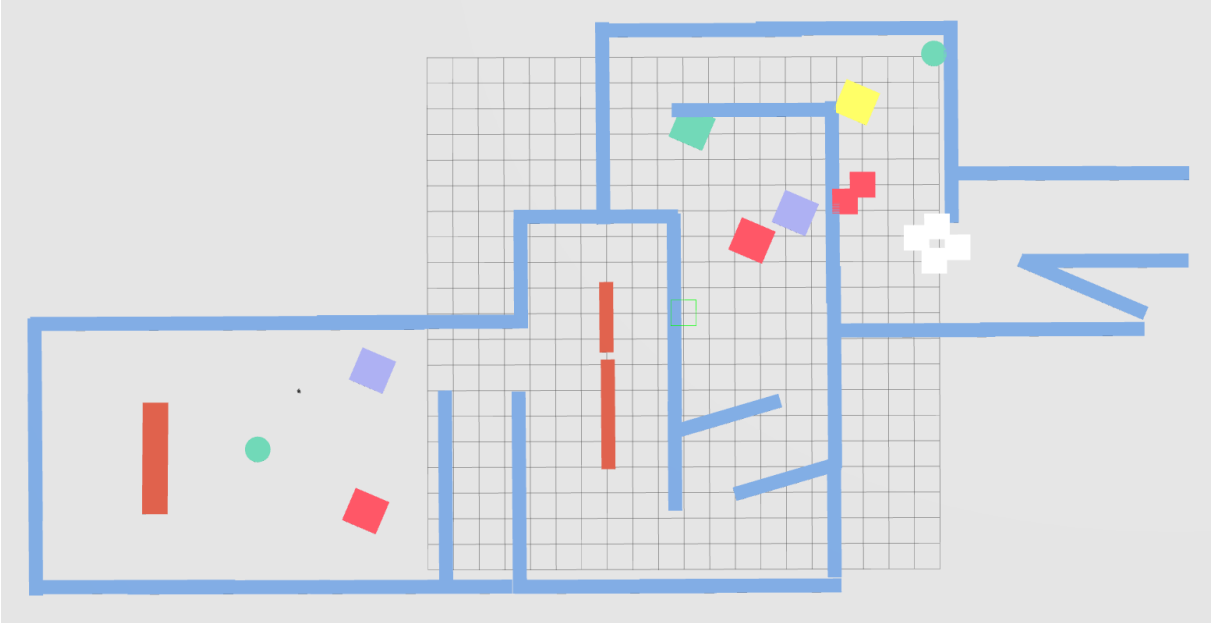

Figure 1: Orthographic view of the maze

2 My Approach and Challenges Faced

2.1 Finding the Walls

The robot must be spawnable at any arbitrary start point within the maze. The first challenge the robot faces is therefore to find and get to the nearest wall. Once the robot is at a wall, it can be begin the wall following algorithm. The robot has no pre-knowledge of the course or obstacles, and must rely on its (limited) sensors to determine where the walls are.

I begin by perform a single 360° sweep of the laser, which returns 360 distance measurements. The 0° position is directly ahead of the robot, and the angles increase (following the right hand rule convention) counterclockwise around the robot. I then cluster the measurements in groups of a set size (so far, either 5°, 10° or 15°). In most rooms (Figure 3), the cluster that represents a segment of wall *should* have the lowest standard deviation (Figure 2). To find the nearest wall, I take the clusters with the lowest deviations and from there select the cluster with the shortest average distance.

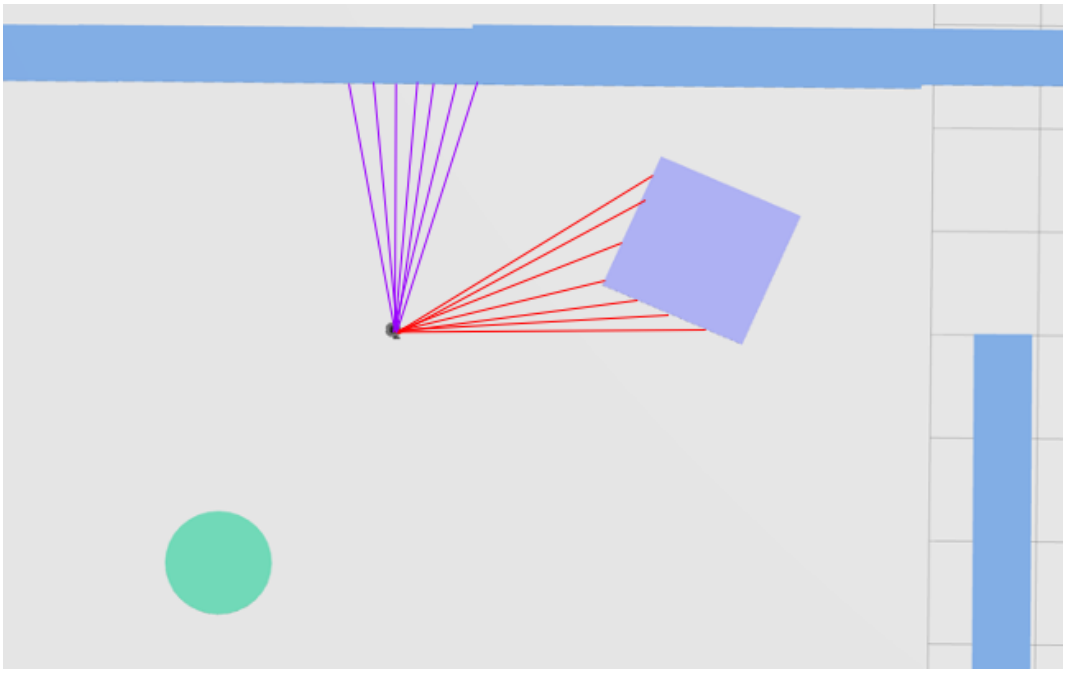

Figure 2: Graphic showing the clusters of laser angles hitting an obstacle and a wall.

I think it's helpful to subdivide the maze into a series of connected rooms (Figure 3), as some rooms present unique challenges. Within most of rooms 1, 2, 4, 5, 6, and 7, I expect that my wall finding approach should get the robot to a wall. Room 3 is trickier; it's obstacles are very wall-like. It is likely that this wall finding approach will lead the robot to the center obstacles from many start points within room 3. If the robot starts at those obstacles, it's wall following algorithm will lead it in circles around the obstacles and it will make no progress through the maze. A similar issue occurs on the left half of room 1 (nearly any point to the left of the green circular obstacle, labeled 1a).

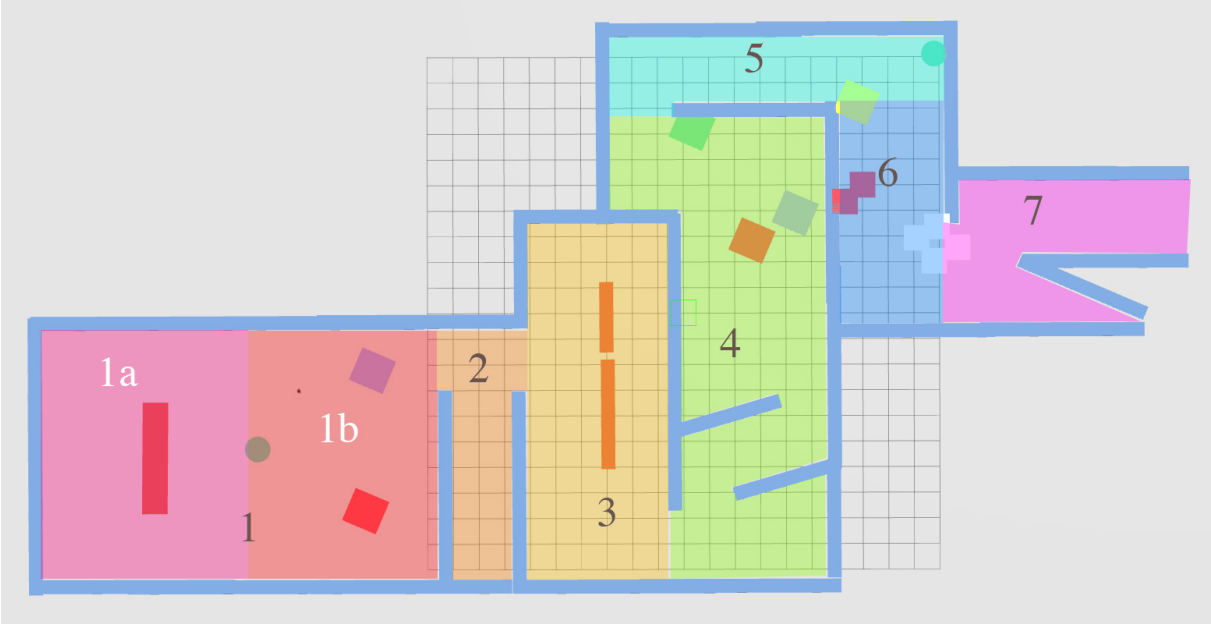

Figure 3: The maze subdivided into 'rooms'

2.2 Solving the Wall-Like Obstacle Problem

The wall-like obstacles present some clear challenges. There are regions (in rooms 1 and 3) where I'd have near equal probability of ending up at a wall or an obstacle. From a purely geometric standpoint, I don't think there is a clear way to differentiate between walls and wall-like obstacles. The difference between an obstacle and the wall behind it is not sufficient - the entrance to room 2 (from both room 1 and 3) looks the exactly same (as just one example), and I wouldn't be able to choose the walls of room 2 as legitimate walls (Figure 4). The area or length of a wall is also insufficient - the longer wall in room 3 is of comparable (or greater) length than the top wall of room 2 and other walls (especially when viewed from the robot)

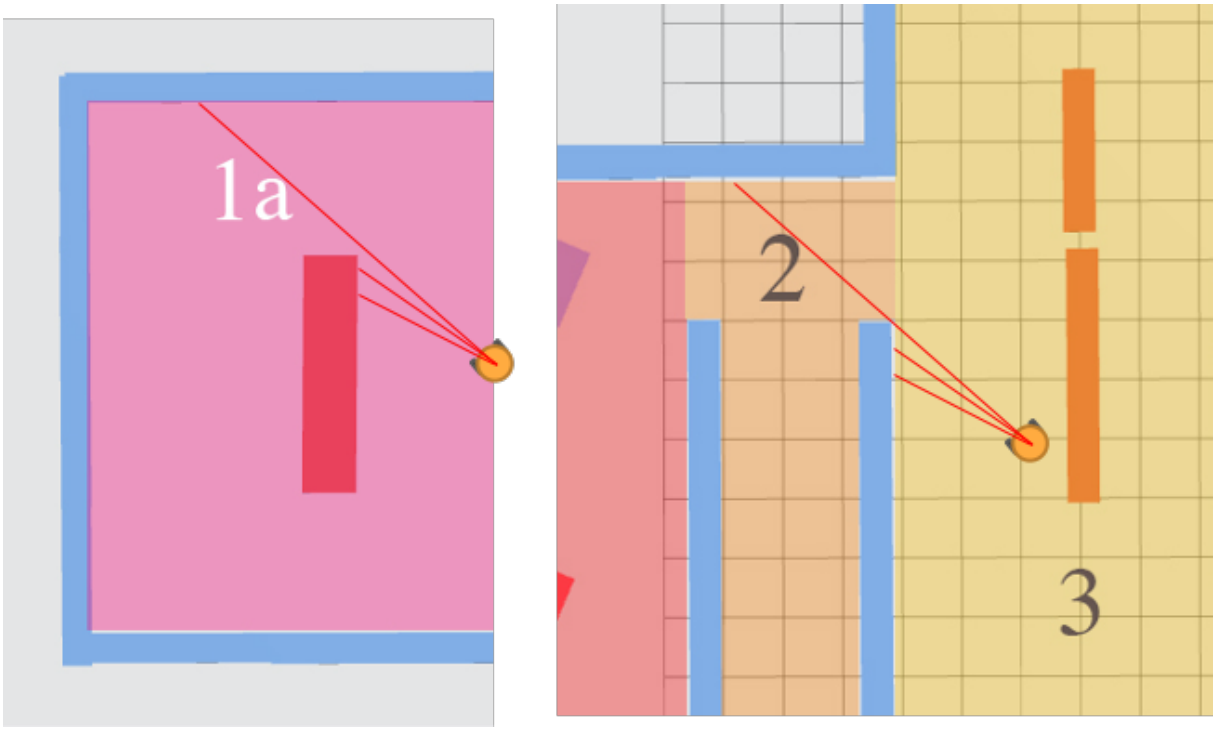

Figure 4: Diagram showing the feature similarities between an obstacle and a doorway

Without the use of SLAM (or a similar system), there isn't an obvious way to efficiently differentiate between legitimate walls and wall-like obstacles. If the obstacles had reflectivity that was measurably different than that of the walls, the intensity portion of the LaserScan message could be useful, but this is not the case.
The revised plan is therefore as follows. When the robot is spawned and performs its initial 360° scan, it must save it's location (scan_start) and the top wall contenders. It will travel to the best wall candidate and again save it's location (wall_start). The robot will follow the wall as best it can. If, at any point, the robot ends up at wall_start (or close enough to be considered the same space), it will know it has gone around an obstacle, and not around a wall. It must then navigate and return back to scan_start (which should be more or less a straight path of unobstructed travel, because that's how the robot got there). Once back at scan_start (both position and orientation), the robot must explore it's second best candidate wall. This repeats as many times as required until the robot solves the maze. The absolute worst case scenario is Figure 5, in which all readily visible wall candidates are actually obstacles. This will be the most inefficient solution, as the robot may need to circle each obstacle before it finds a wall.

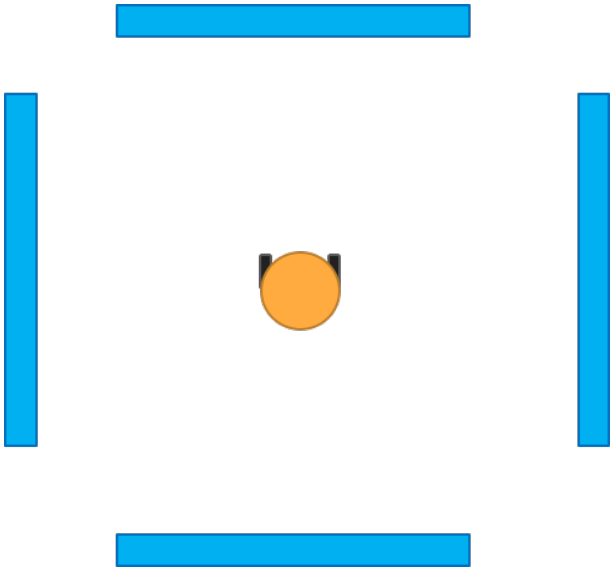

Figure 5: Worst case configuration with 4 false wall candidates

In the case where the all of the robot's best candidate walls are actually obstacles, the robot must pick a random direction and drive a short distance. This new position becomes scan_start, and the whole process repeats. Eventually, this will find the correct external wall, but it is clearly an inefficient system.

2.3 Wall Following

The wall finding function should terminate with the wall to the right of the robot. I don't want the robot to be completely against the wall - it will need some buffer room for positional/control error so it doesn't get stuck. The robot must continuously scan at several points: immediately in front of the robot [F] (0°), directly to the right [R] (270°), front right [FR] (≈ 285°), and back right [BR] (≈ 255°). While driving alongside a straight wall, the distance measurements from FR and BR should be nearly identical (nearly, to allow for some noise). If $FR > BR$, the robot has veered left of the wall, and must turn slightly right to compensate. Similarly, if $FR < BR$, the robot has veered right towards the wall (and will likely soon collide), and must turn slightly leftward to compensate (Figure 6). Everything is done from the robot's perspective. It doesn't know if it has veered from the wall or if the wall is now angled, and it doesn't matter. The corrective actions are the same.

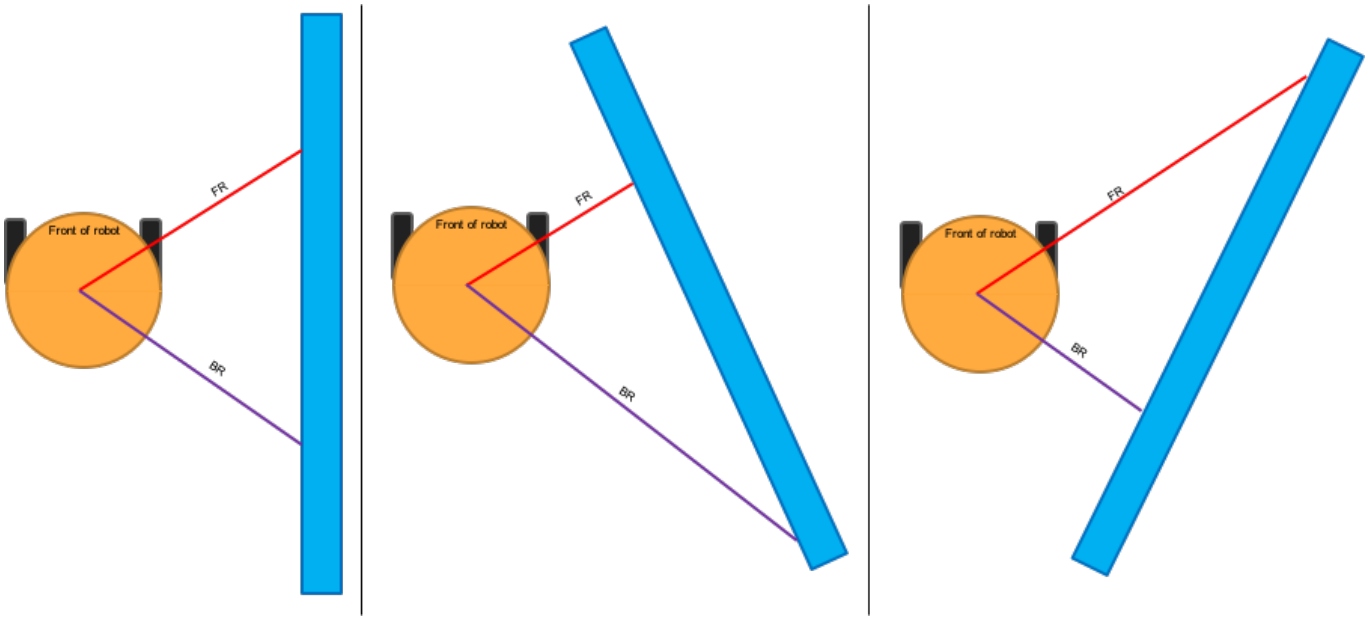

Figure 6: Diagram showing the relationship between FR and BR for three wall alignment conditions

The F check is to avoid collisions with any obstacle in the robot's path. When the F measurement drops below a threshold, the robot stops, rotates leftward (counterclockwise), drives forward a small amount, and then resumes the FR/BR checks.

2.4 Victory Condition

The robot's task is complete when it exits the maze at the right of room 7. Along the final wall of the maze should be the first time the F distance measurement returns ∞ (Figure 7).

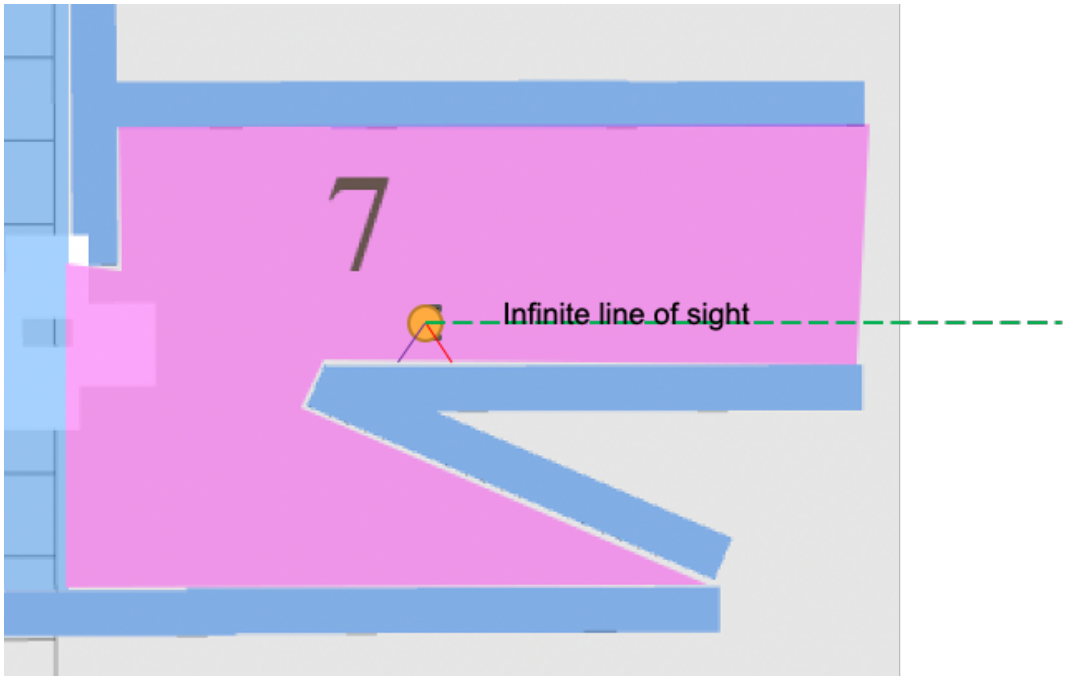

Figure 7: Diagram showing the infinite line of sight as the robot approaches the finish

When $F = \infty$, the robot knows it is approaching the goal. When FR first becomes ∞ (Figure 8), the robot must stop the FR/BR control sequence (otherwise it would round the corner and keep wall following the outside), but keep driving straight. This ∞FR condition must only be checked while $F = \infty$ (to avoid risk of an erroneous ∞ measurement) .

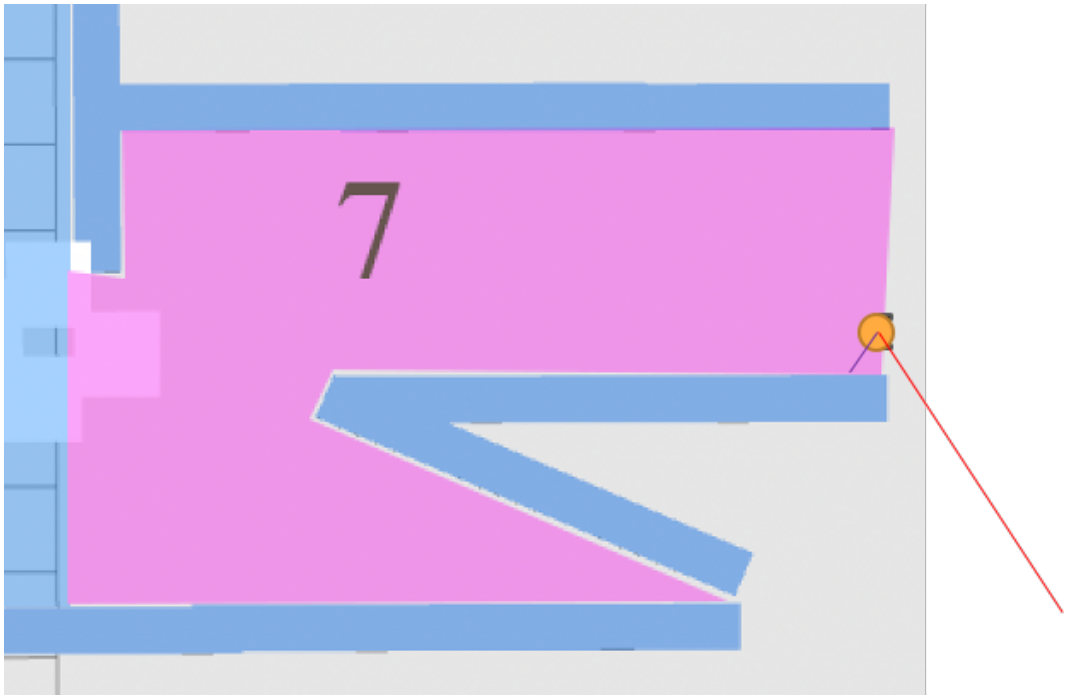

Figure 8: Diagram showing the infinite FR as the robot crosses the finish

When F, FR, and BR are all ∞, the robot has exited the maze and the program can terminate.
I attempted some involved trigonometry to determine the exact angle to correct the robot's pose by based on the measurements of FR, R, and BR, but it was difficult to integrate this approach with linear motion, and so it was abandoned in favor of a less mathematically rigorous (but much more successful) gentle nudge approach. In this system, differences between FR and BR are compensated by small increases in angular velocity in the correct direction. This works very well, but does cause the robot to be somewhat slow in it's course corrections.

3 Course Feedback

I liked that the final project is a fairly realistic scenario and task, with clear applicability to the real world. The slides were fairly detailed, organized, and easy to follow. Despite the time we spent in class on the subject, I'm still not 100% clear on the details related to ROS packages and ROS Python packages (specifically the requirements of the file structure and importing python packages, as well as the implementation of multiple publishers and subscribers in a single package). It may have been helpful to have had RWA3 to get additional experience/practice with this. However, timing wise, I was glad to only have had RWA1, RWA2, and the final project. Each (especially the later two) took much more time and effort than I had originally anticipated.