PID – A Closest range Approach

Lilian Lamb

A proportional–integral–derivative (PID) controller is a commonly used controller in robotics. The controller has three aspects: proportional, integral, and derivative. The proportional aspect takes the error function and multiplies by the proportional component (kp). The integral aspects take the integration of the error function and multiplies by the integral component (ki). The derivative aspect takes the derivative of the error function and then multiplies by the derivative component (kd). All aspects are then summed together and used to control the robot's movement. In this assignment, I chose to utilize a PD controller (PID without the integral aspect) to control the rotational component of the robot while keeping the linear constant.

On boot, the robot started to search for the closest obstacle (either a wall or an obstacle in the middle of the environment). The robot then moves forward at a rate of 1 until there is an obstacle within range. Once within range the robot rotates until it is perpendicular to the obstacle. Now that the robot is in the proper position, the robot switches modes and starts the wall following. The switch is the first part of the program that can be improved. For some reason, once the robot starts wall following, it does a 180-degree rotation to face the opposite direction. The robot does this even when the rotation is switched during the boot up sequence. This rotation does not affect the program's goals, nor the ability to wall follow, but ideally the robot would not do this.

The rest of the wall follow algorithm is simple. It is controlled by a PD controller where kp is 0.5 and kd is 0.2. I chose not to do a PID controller as the integral aspect caused the rotation to grow too rapidly and caused the robot to spin in a circle. I calculated the error by taking the difference between the ideal distance to the wall (2) and the distance to the closest range. The closest range was found by the same method it was found in the first assignment. I modified the code to loop through all indexes more smoothly and to get a better idea of the angle at which the closest range is. I calculated the turn rate by multiplying kp by the error and adding the value by kd multiplied by the difference of the current error by the previous error. That turn rate was provided to the robot as the angular velocity alone with a constant 0.2 linear velocity.

The program has its positives and negatives while traversing the cave single world on stage. Firstly, the robot was able to follow wall consistently and stay close to the wall without losing it at any step, especially when another object came into sensor range. This success can be seen in Figure 1-6 that were all taken in a single run. However, the program

had several issues. The first being that the robot tended to oscillate during the wall follow which could be solved by fine tuning the controller or properly integrating the integral aspect into the controller. Even with the oscillation, the robot succeeded in wall following the entire time, but was not as smooth as it could have been.

The second issue is that since the robot does not keep track of all obstacles, only the closest range, the robot can switch obstacles that it is following. If the robot is wall following and then comes into range of an obstacle that is closer, it will then start looping around the obstacle; however, at any point of looping the obstacle the robot comes into range with the wall and the wall is the closest it will go back to wall following. Similarly, with sharp corners, the robot can get stuck in a loop and start spinning in a circle. This issue could be easily mitigated by implementing a technique used in potential fields. The program could keep track of the turn rate and if it exceeds a certain value could implement a negative force and push the robot out of the corner. The robot could then reenter a similar system as in boot (find the wall and turn perpendicular) then reenter wall follow.

Whil the current approach does not implement either method, both methods could be implemented to improve the operations and the ability to wall follow. Another aspect that could be implemented in the future, that is a current limitation, is the ability to follow in any direction. Currently the program can only wall follow to the left. While the project's goal is to follow a wall, being able to follow a wall in only one direction is limiting and inefficient. The program is able to succeed and properly wall follow, but clearly has its limitations. With a few small edits and additional testing, the robot could be improved upon and be able to succeed in wall following in any environment.

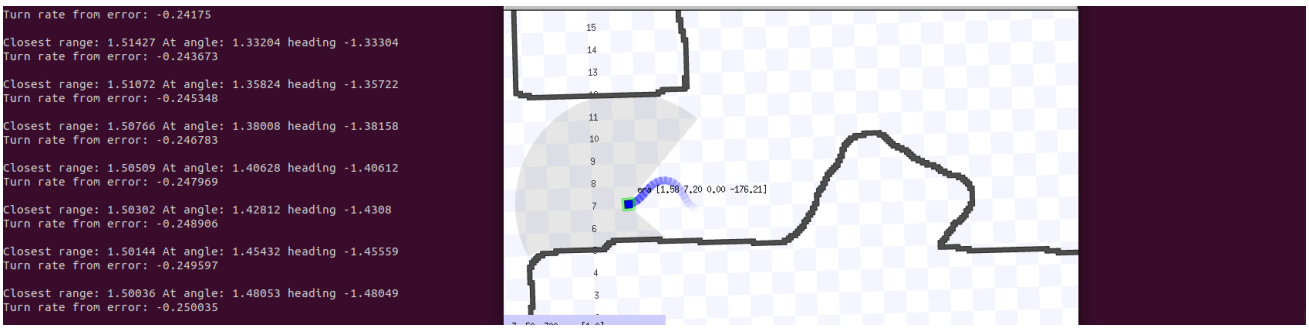

Figure 1: robot starting wall follow after detecting object

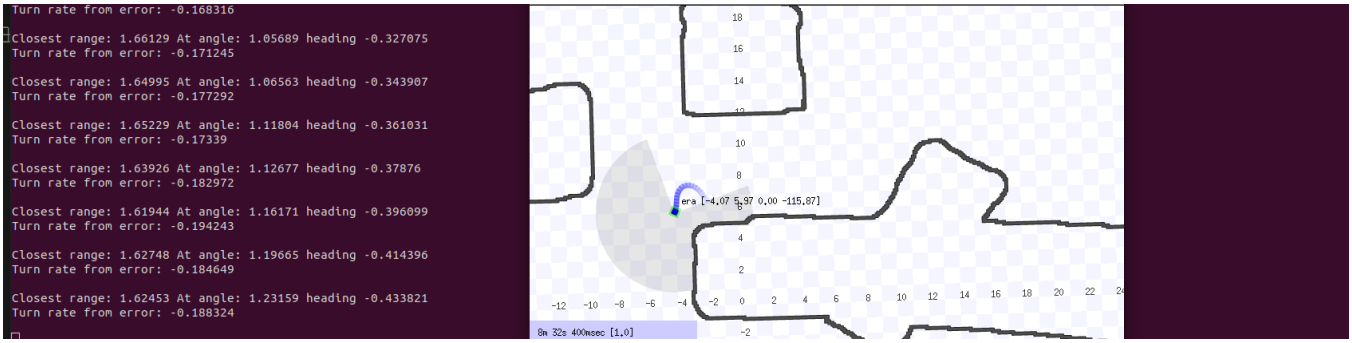

Figure 2: Robot turning during wall follow

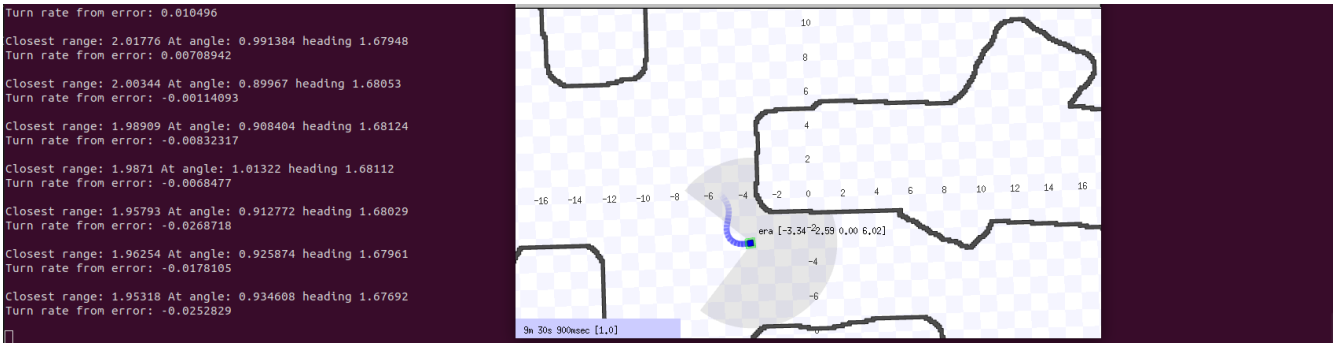

Figure 3: Robot taking second turn

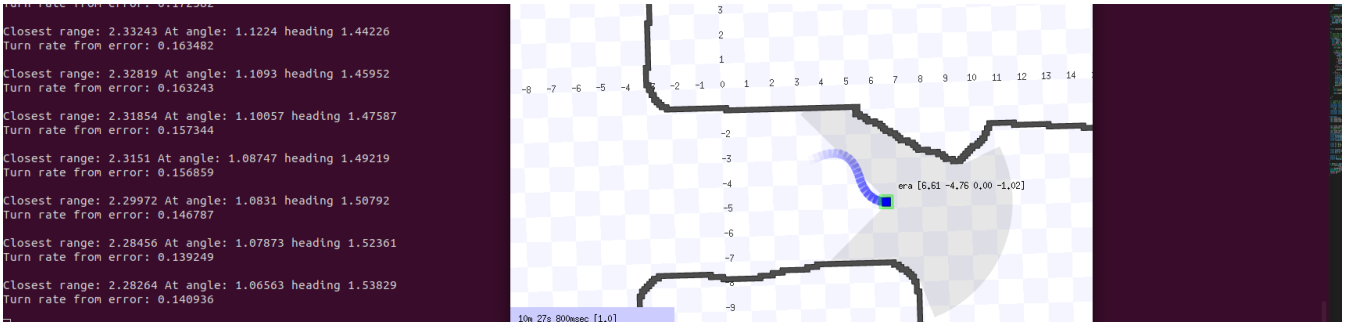

Figure 4: Robot continuing to wall follow while in range of another obstacle

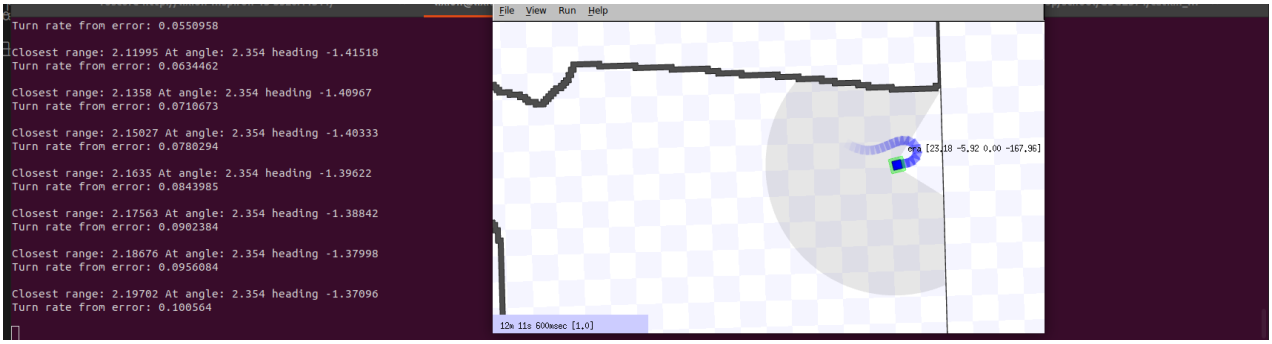

Figure 5: Robot wall following during right angle corner

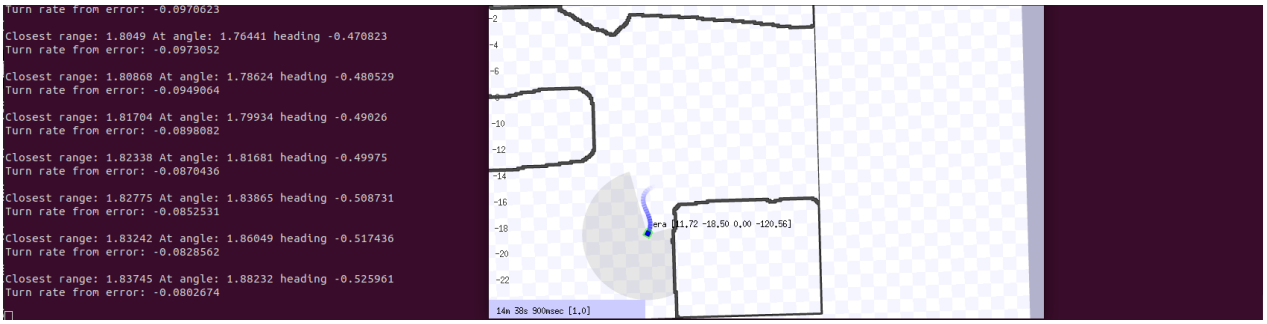

Figure 6: Robot properly wall following since overturn during corner