COSC 081/281 - Principles of Robot Design and Programming (Term: Spring, Year: 2023)

Assignment: PA2 - Report

Author: Carter Kruse

Date: April 22, 2023

Implementation / Description

Overview

The purpose of this assignment is to serve as an introduction to a feedback controller for following a wall. Familiarity with ROS, publishers, and subscribers is a prerequisite. The PID controller material is modified for the purpose of this assignment.

The program is a simple ROS node that implements a PD controller that follows the right wall at a given distance, using the laser sensor. The robot is assumed to be deployed in an indoor environment, according to the world file provided for the stage. A finite state machine governs the different behaviors of the robot.

PD Class

The PD controller uses a class to perform the appropriate calculations, specifically that which is the control command. This class is based on the PID class discussed in class. The initialization function specifies the proportional and derivative gain constants. The step function returns a control command, which determines the rotation of the robot, using the standard formula (as discussed in class).

The step function is called at each sensor update, which allows the control command to be calculated as often as the sensor data is provided. To use the derivative term, the previous error is subtracted from the (current) error, which indicates that the time differential is one second. This does not cause issues (in the sense of different time differentials), as the derivative gain may be adjusted as needed.

The reset function simply resets the previous error, which is not used in the execution of the program.

Design Decisions

In developing the program, various design decisions were made, primarily to ensure the accuracy/completeness of the code. Specifically, the linear velocity, min/max scan angles, goal distance, and proportional/derivative gains had to be closely set, so that the robot would rotate or move as expected, allowing room for error.

The linear velocity was set to the maximum for the ROS Turtlebot, which is 0.22 m/s. The min/max scan angles were determined by considering that the robot should be able to detect directly in front of it, and to the right of it. There is no point in the robot detecting behind it, as that area will have already been

covered in the initial pass of the robot. Thus, we allow the robot to consider the space 45º behind from directly to the right, as we still might want to take into consideration the distance if the robot is oriented away from the wall.

The goal distance is set to 0.6 m to allow the robot to stay close enough to follow the wall, while simultaneously eliminating the possibility of the robot crashing into the wall. This value was determined through testing/experimentation: 0.5 m caused the robot to crash into the wall, whereas 0.7 m was a distance that was too far away from the wall.

After this, the proportional/derivative gain must be chosen appropriately to meet the requirements. The proportional gain is used to make sure that the robot does not impact with the obstacles (walls) when getting too close, and the derivative gain is used to smooth out the oscillations that develop when the proportional gain increases. This was the source of many of the experiments conducted, which will be explained later in this report.

Main Class

By utilizing the PD class, the main class is relatively simple. The constants are initialized at the top of the class, followed by an initialization function for the class, which is used to set the following parameters: command velocity publisher, laser scan subscriber, linear velocity, min/max scan angles, goal distance, proportional/derivative gains, PD controller, and finite state machine.

A finite state machine is used to control the operation of the robot, though there are only two states to consider: when the robot is stopped, and when it is moving (in an arc). That is, the goal is to keep the robot in constant motion, rather than make the robot stop to turn.

The *move()* and *stop()* methods are responsible for creating and publishing the Twist messages corresponding to given linear and angular velocities. The *laser_callback()* method is responsible for handling the LaserScan data, determining the minimum value according to the min/max angles specified. Using this information and the goal distance, we are able to calculate the error and set this variable as appropriate. Following the processing of the LaserScan data, the finite state machine is set to *MOVE*.

The *run()* method handles the movement of the robot, according to the finite state machine (*MOVE* or *STOP*). The frequency rate is set at a global variable, at 10 Hz. In the case where the finite state machine is *MOVE*, the PD class is used to calculate the appropriate control command, based on the error determined by the handling of the LaserScan data. This prompts the robot to adjust the angular velocity as appropriate. Otherwise, when the finite state machine is *STOP*, the robot remains stationary.

Evaluation

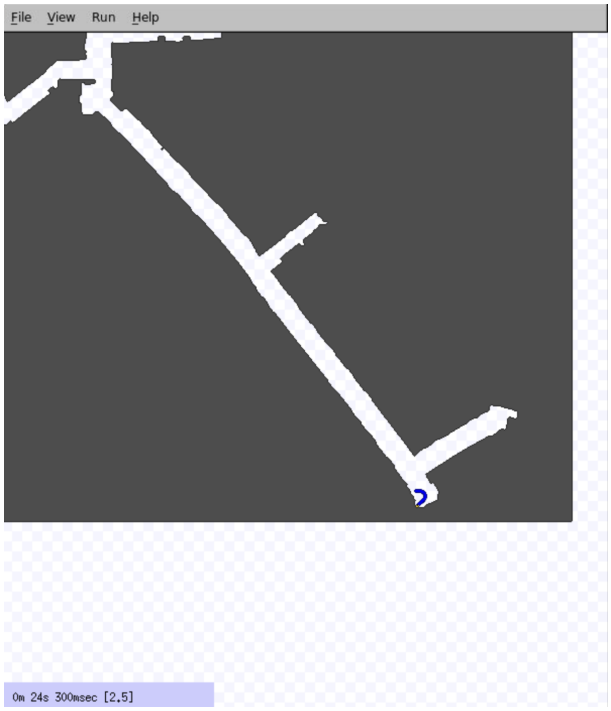
To verify the accuracy/completeness of the program, various experiments were conducted, with given parameters, highlighted as follows.

The main objective of this assignment was to test different proportional/derivative gains to determine what works. As indicated previously, the proportional gain is used to make sure that the robot does not impact with the obstacles (walls) when getting too close, and the derivative gain is used to smooth out the oscillations that develop when the proportional gain increases.

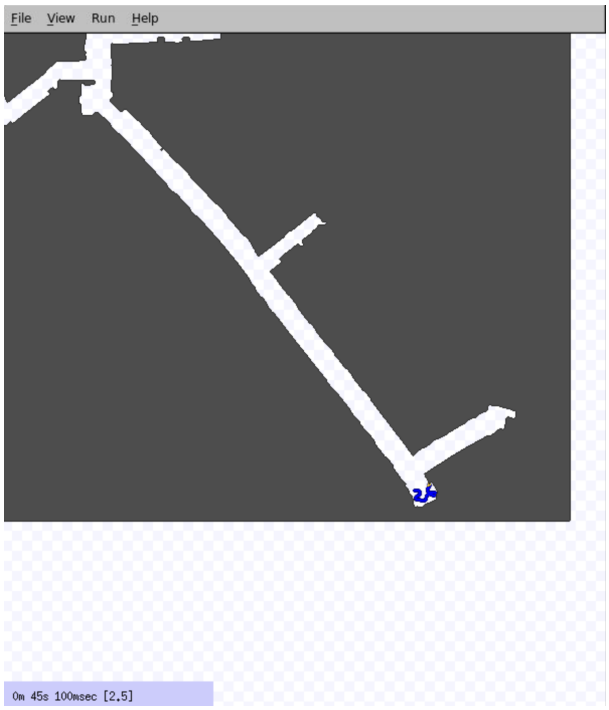
Due to the nature of the code written, left and right turns are handled gracefully. By including an angle of -135º to 0º for the FOV (field of view) of the robot (with 0º being the front of the robot), we are able to adjust the robot accordingly when a turn is deemed necessary.

The following experiments were conducted, along with screenshots of the behavior of the robot, after speeding up the simulation. The trails were enabled to display the trajectory of the robot in the past. Many of the following display the robot crashing into the walls (obstacles). Others display the robot with sharp oscillations, which is not the intended behavior of a wall-following robot.

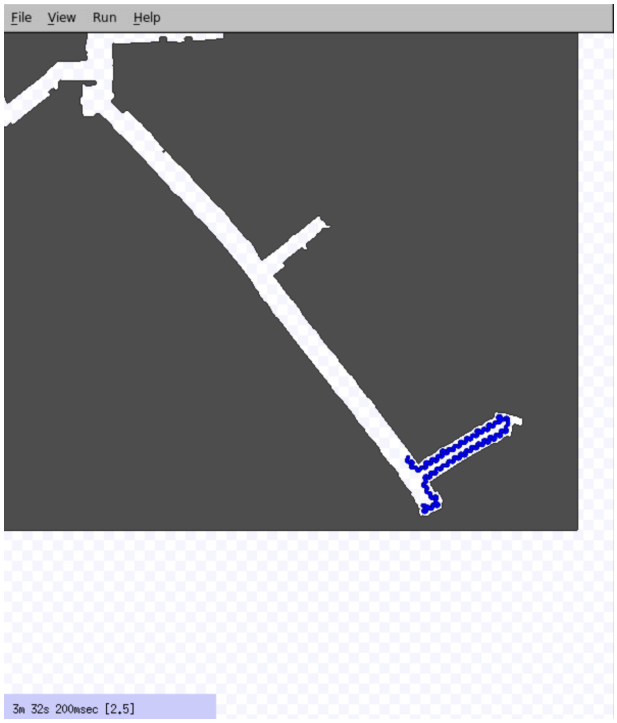
Proportional Gain: 1 — Derivative Gain: 0



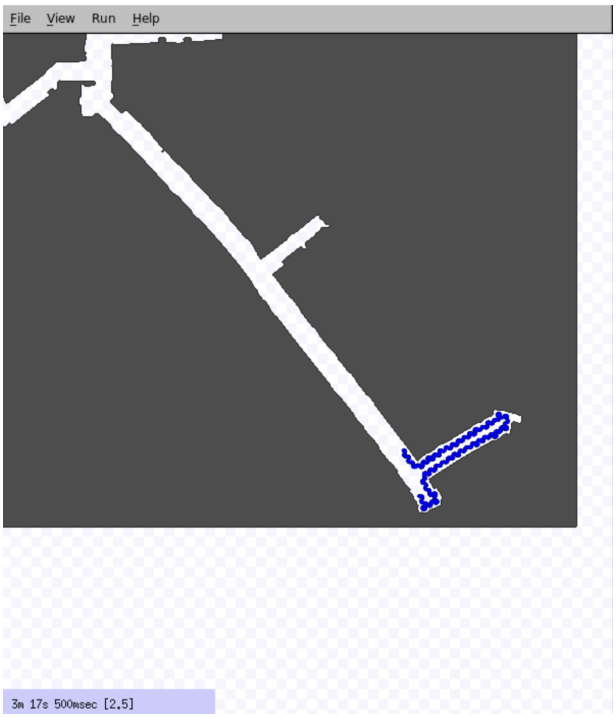
Proportional Gain: 4 — Derivative Gain: 0



Proportional Gain: 12 — Derivative Gain: 0



Proportional Gain: 12 — Derivative Gain: 4

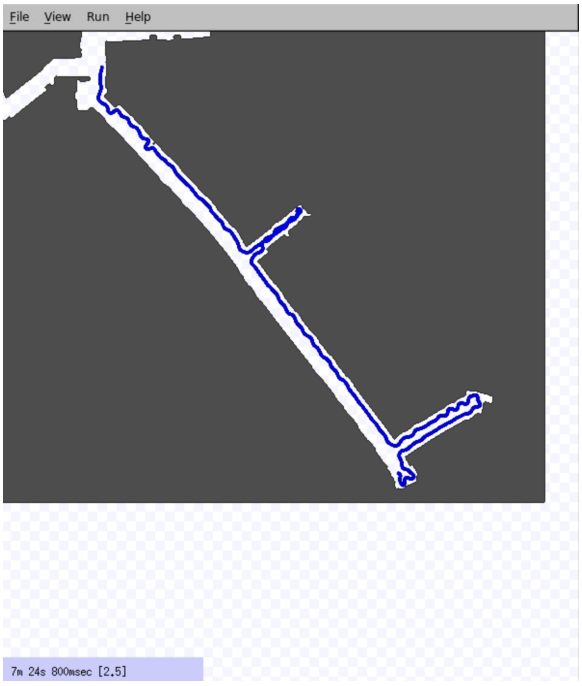


After experimentation, a reasonable result was produced, by lowering the proportional gain, while simultaneously increasing the derivative gain. This, however, did not seem to be quite what we aimed for, as there were still oscillations in the path of the robot, which created an issue when the robot approached a tight part of the maze. This is shown on the following page.

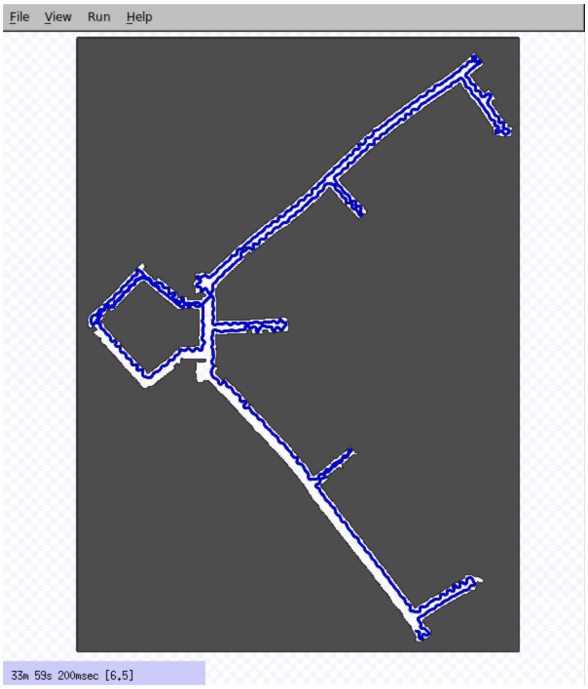
After further experimentation, it turns out that increasing the derivative gain by a significant factor produces a robot that is relatively well-functioning in the given environment. Moreover, increasing the derivative gain by such a significant factor does not seem to impact the robot colliding with any of the walls, as the proportional gain is high enough to avoid this. This is shown on the page after the following.

Proportional Gain: 4 — Derivative Gain: 22

The following displays the partial path, which was the initial testing for the proportional and derivative gains.

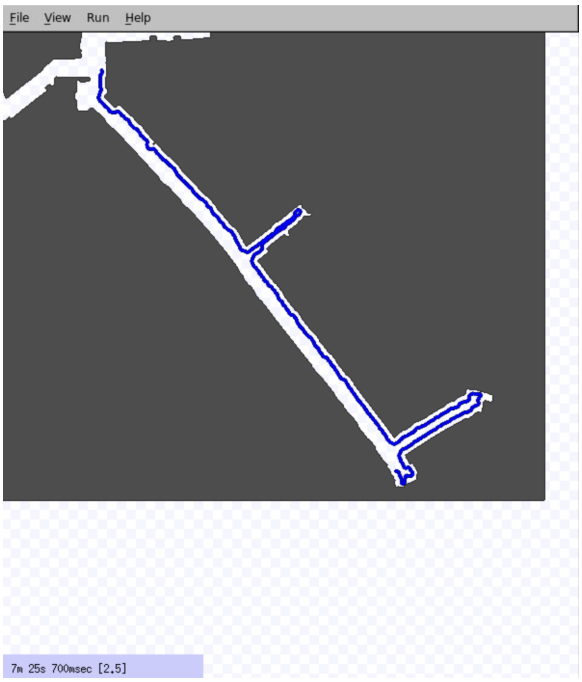


The full path is displayed below, though the execution of the program was sped up significantly to avoid wasting time. Thus, there are slightly more oscillations, due to an issue that will be addressed later.

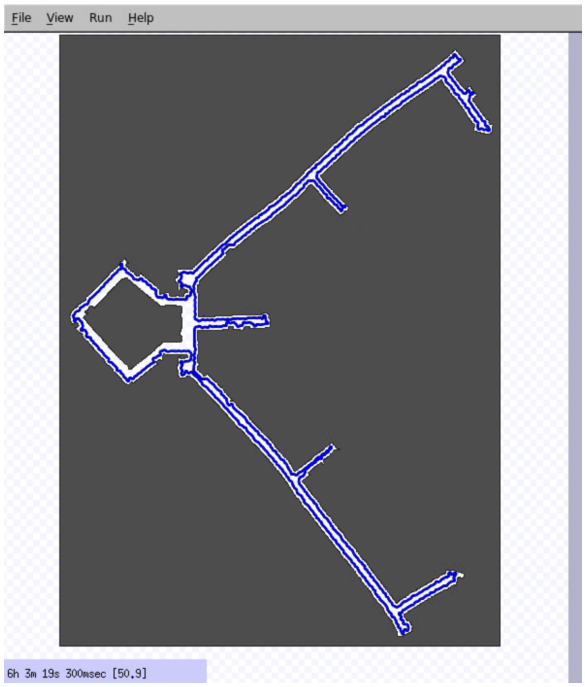


Proportional Gain: 12 — Derivative Gain: 1000

The following displays the partial path, which was the initial testing for the proportional and derivative gains.



The full path is displayed below, though the execution of the program was sped up significantly to avoid wasting time. Thus, there are slightly more oscillations, due to an issue that will be addressed later.



Conclusion

The program functions as expected, according to the specifications.

The robot does not seem to encounter difficulties, in that it does not run into issues in which it collides with obstacles, if the appropriate gains are used. Further, the robot does not become stuck, oscillating between positions or rotations.

However, it should be noted that when speeding up the simulation, there appears to be an oscillating movement of the robot. This is not the case when the simulation is run at "real time" indicating that there is likely an issue with the way the simulation is being sped up.

Credits

Alex Fick (*April 20, 2023*) → High-level discussion regarding the angles used to determine the minimum distance to the wall (on the right side).

Laser Scan Message (Documentation)
http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html