ME 598

Introduction to Robotics

Fall 2022

Simulating a Mobile Robot

Group # R3

12 December 2022

Graduate students:

"I pledge that I have abided by the Graduate Student Code of Academic Integrity."

The report has been prepared by:

1. Lab leader: Sakshi Ranjeet Pol
2. Peera Tienthong
3. Krupansh Manish Shah
4. Shanmukha Sampath Kumar Tiriveedhi

Table of Contents

Abstract

Mobile robots are robots that can move from one point to another in the physical world. These robots can be used for multiple tasks and can vary in size depending on the task. Just like internal configurations and controls, external factors can also impact the robot motion. Simulation is one of the ways a robot's behavior can be assessed in virtual environments. MATLAB provides various tools to facilitate such simulations. The simulation environment and robot are already designed and provided. This experiment is to try and test how the individual components of a robot can be manipulated to control its motion and behavior in the simulation environment using various mechanisms and try avoiding obstacles.

Introduction

A mobile robot is a machine controlled by software that uses sensors and other technology to identify its surroundings and move around its environment. Mobile robots function using a combination of sensors and physical robotic elements, such as wheels, tracks, and legs. Mobile robots are becoming increasingly popular across different business sectors. They are used to assist with work processes and even accomplish tasks that are impossible or dangerous for human workers. In this lab, we are using Simulink to train our robot on obstacle recognition, and wall avoidance with the help of the Robotic Playground app. Using the distance sensor the robot was able to avoid crashing into a wall and after running multiple simulations we were able to get a perfect trajectory plot where our robot was not colliding into the wall and nor was it taking a smaller path to avoid the wall.

Theory & Experimental Procedure

Part 1: Open Loop Robot Motion

The left and right motors are controlled by open-loop gains. By adjusting the gains, various movements include moving forward, backward and arching. To move forward, equal positive gains for both left and right motors are required. In contrast, adjusting equal negative gains will result in moving backward. Arching movement can be generated from unequal gains. To arch clockwise, adjust the gain of the right motor so that the right-motor gain is smaller than the left-motor gain. This is vice versa for arching counterclockwise. Through the combinations of these absolute value and sign changes, the desired direction is achieved. This helps in navigating the robot in different situations; such as turning left/right on a desired path or avoiding a collision. In addition, the gain size determines the motor's speed. Table 1 gives an outline of the gains/speeds supplied for each motor to achieve the desired motion.

Movement	Left Motor Speed	Right Motor Speed
Forward	100	100
Reverse	-100	-100
Pivoting	100	-100
Arcing	50	100

Table 1: Gains of motors for each movement type

Part 2: Using Distance Sensors

a) Distance Sensor basics

Apart from open-loop motion driven by 2 motors, another essential component is distance sensor. With this sensor, the mobile robot can sense the environment. In Simulink, there is a distance sensor block which will be configured to measure a length between the robot and any object facing the sensor. Inside the block, one can modify measuring range(maximum and minimum), resolution etc.

b) Wall Avoidance

The basic application of the distance sensor is to avoid wall collision. Simulink can visualize the measured distance by connecting the sensor block with a display block as illustrated in figure 3.

As seen in figure 1 the range values of the sensor are set from 10 m to 0.02 m, thus the distance sensor will give an output when the proximity of the obstacle is 10 m or less.

The area is set to be obstacle free and the robot is given a forward motion of 50 (as shown in figure 2). As the robot proceeds in the arena the output value of the distance sensor keeps decreasing and stops at 0.02 m when the robot collides with the wall.

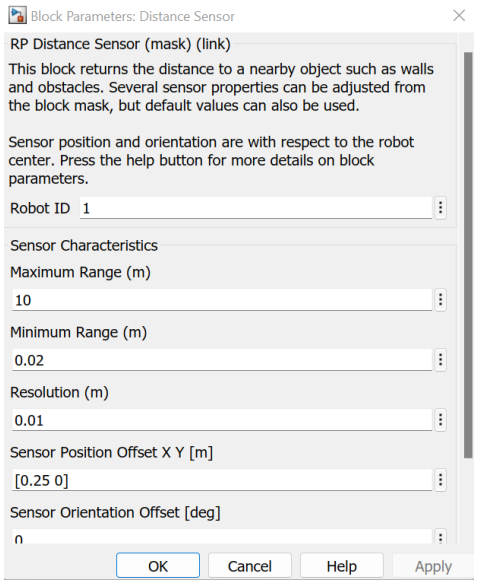

Figure 1: Distance sensor configuration

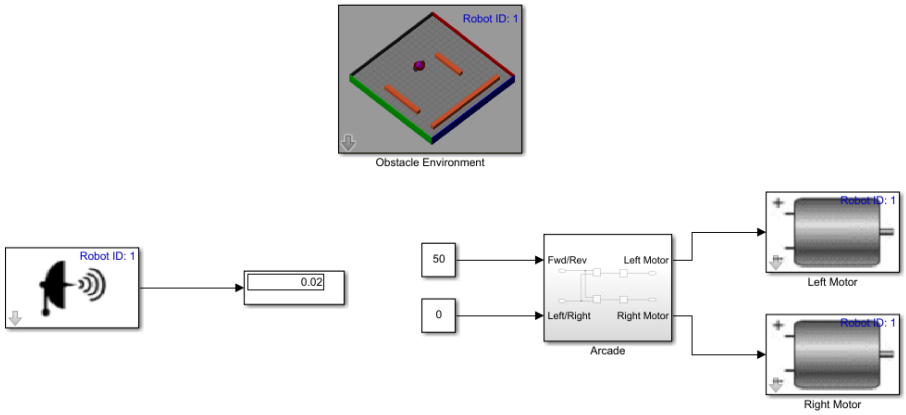

Figure 2: Motor settings and system layout

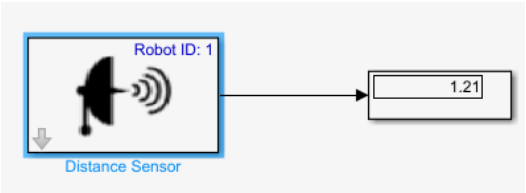

Figure 3: Sensor, Display and Switch block placed for distance measurement

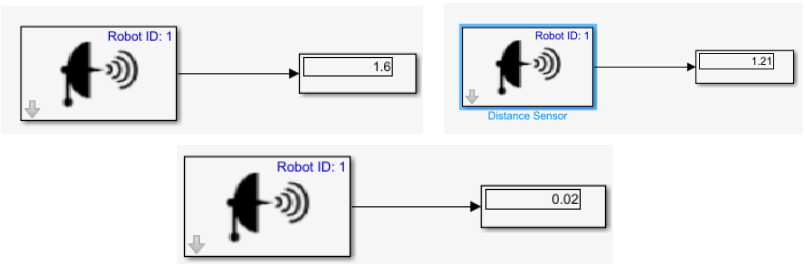

Figure 4: Different proximity values given by distance sensor.

c) Using a switch block for conditional operations

To incorporate the sensor information into the robotic system, the robot has to decide when to move, stop, or rotate by using a simple conditional statement. In Simulink, a block called 'Switch' provides this conditional operation for distance sensors as illustrated in figure 5.

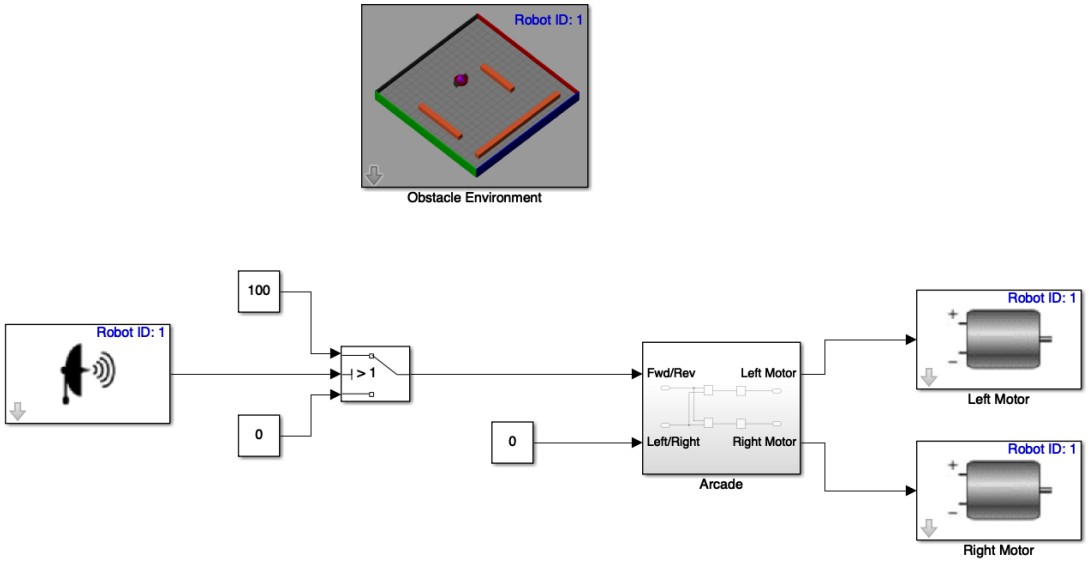

Figure 5: System layout for testing switch block

For example, the switch block can set a threshold to 1m. If the distance measured by the sensor is larger than 1m, the 100 gain is fed to both motors in order to drive forward. Otherwise, the robot comes to a complete stop(gain is 0). This helps in avoidance of collision of the robot and the wall.

d) Wall Tracking

For tracking the wall concepts from the previous subsections are used. Two switch blocks, one to control the forward/reverse motion and the other to control the turning are used. For turning, pivoting motion is preferred over arching as the robot covers lesser area to turn by the same degree, this helps in preventing collision at a later stage. Through trial and error the threshold value is set such that the robot can go as close as

possible to the wall without touching it (as seen in figure 7). Thus, the threshold value is set to 0.7 m for forward/reverse and 1.2 m for turning/pivoting. The robot is able to track the wall successfully without collision in 75 secs of simulated time. However, the robot was susceptible to slight jerks at the corners.

The robot is made to track the wall several times (as seen in figure 9). It can be noticed that the robot can continually track the wall without collision. Thus showing and confirming the repeatability of the process.

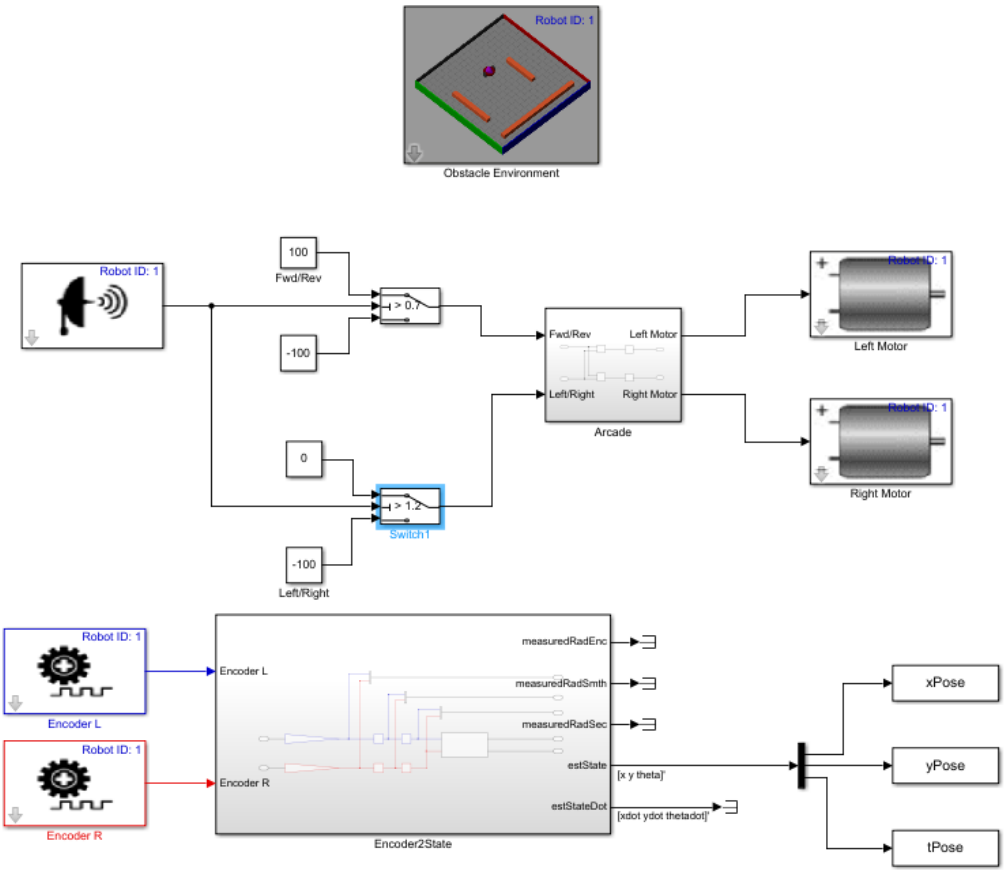

Figure 6: System layout for wall tracking

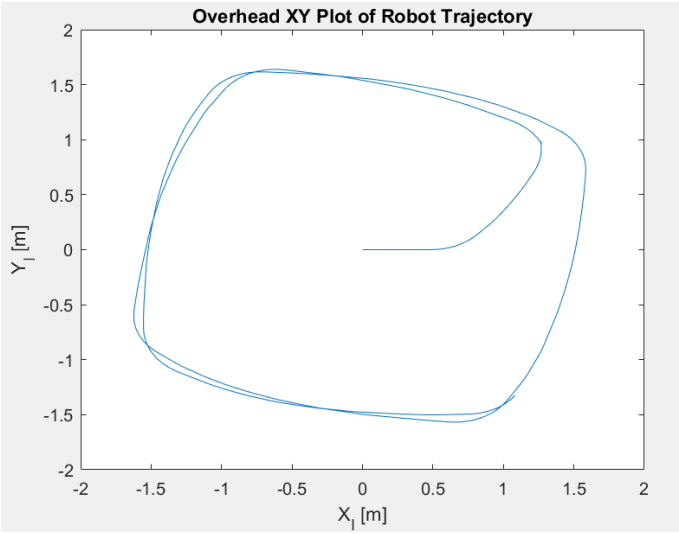

Figure 7: Robot trajectory plot on XY plane for 1 rotation

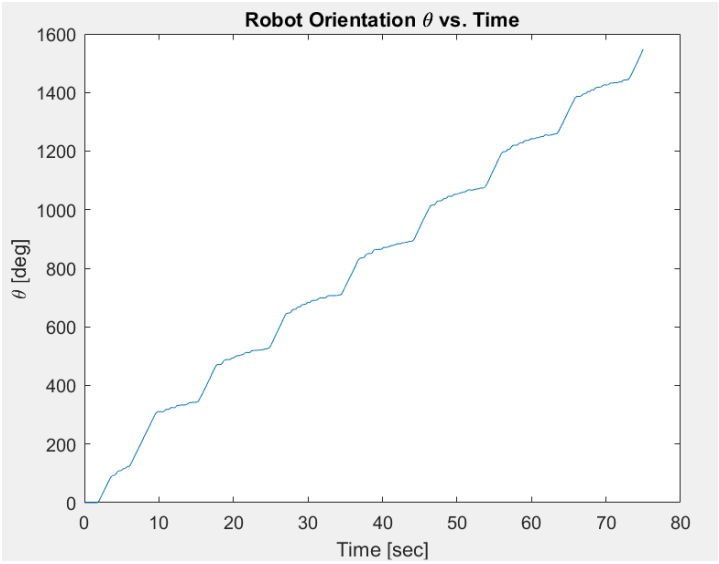

Figure 8: Robot orientation plot of the robot path during simulation

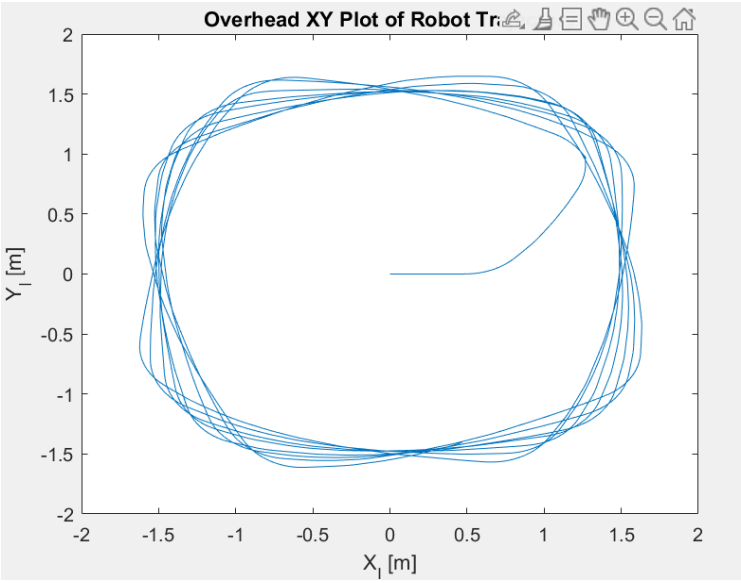

Figure 9: Robot trajectory plot on XY plane for several rotations

e) Using MATLAB code within Simulink blocks

Till now, various components in the simulink library have been used to achieve the desired motion of the robot. Now, the same behavior is tried to be achieved by writing custom code blocks or functions within Simulink. The switch and arcade blocks between the sensors and the robot are replaced with a single function block of the Simulink library. This function is modified to achieve the collective functionality of all the blocks replaced. The input variables of this function include the sensor readings. Based on these readings, the output variables of left and right rotor speeds of the robot are decided. The robot will start with a constant speed of 127 rpm on each rotor in a forward direction. If it is too close to a wall, then it stops the forward motion and starts a turning motion. If the sensor detects any obstacles the forward movement is paused and the turning/angular motion is initiated so that the robot makes a turn. This turn for now is made to be a left turn with a pivoting motion from the middle of the robot. The turn starts with a speed of 127 rpm on each wheel/rotor but in the opposite direction to make it a pivoting motion. Figure 10 shows the Simulink model for this part and the commented code of the function block is attached in the appendix. Figure 11 and figure 13 shows the overhead plot of the robot path completing the wall tracking of the environment and Figure 12 shows the robot orientation with respect to the original position. The same thresholds used in the previous part are used here as well to get the robot as close to the boundary/wall as possible. And since the motors will have similar speeds in each condition as the previous part, the robot took about 35 secs for tracking the wall completely.

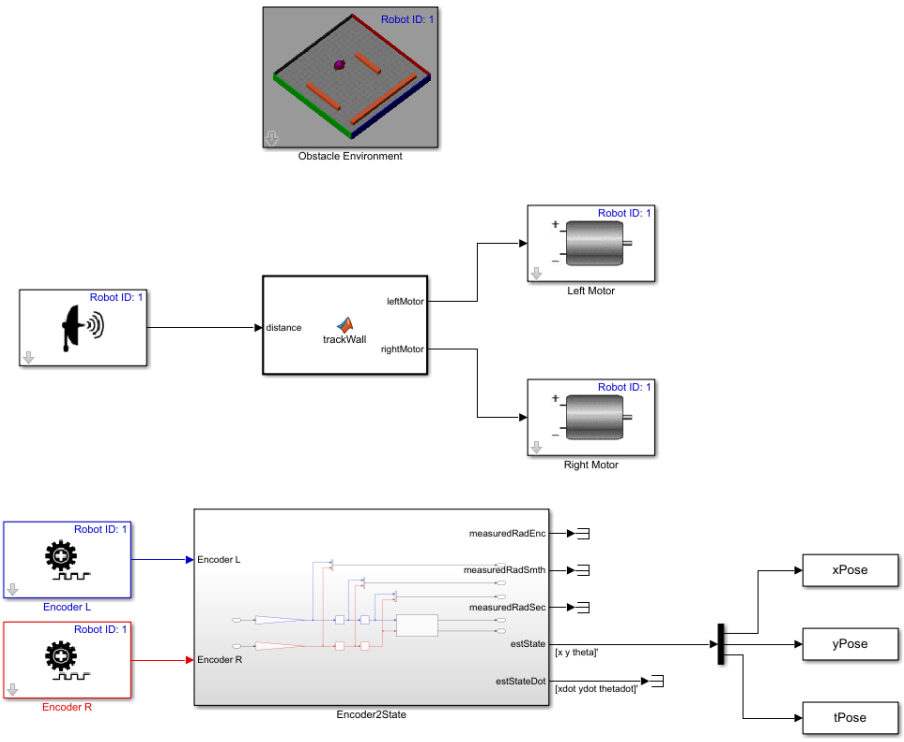

Figure 10: Mobile Robot Simulation with MATLAB function block in Simulink

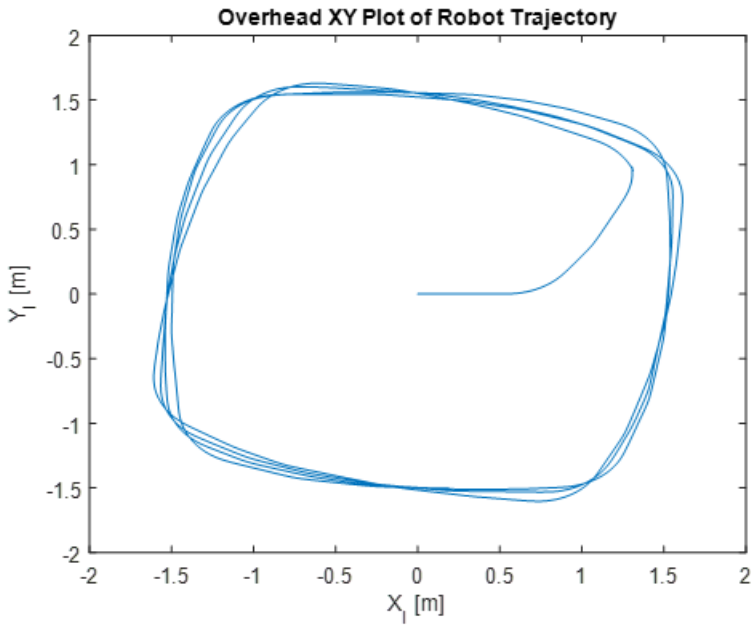

Figure 11: Overhead Plot of the robot path during simulation

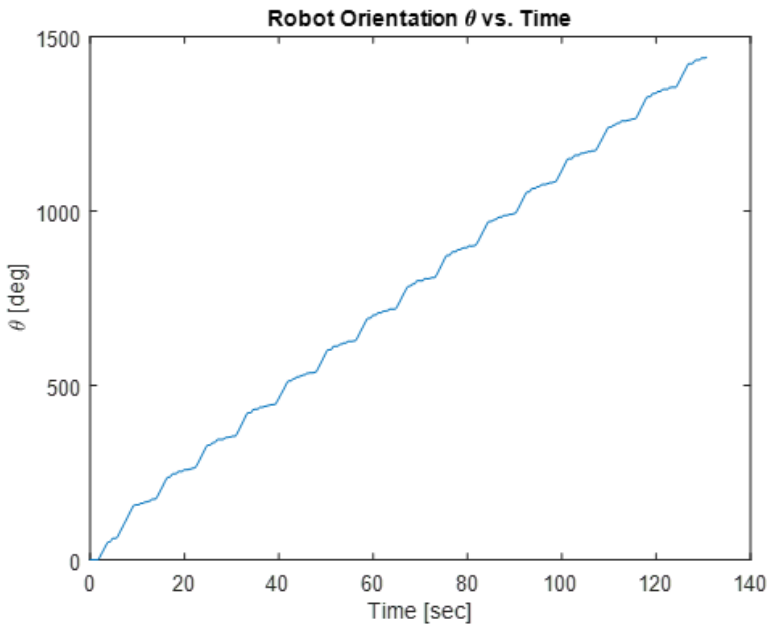

Figure12: Robot orientation plot of the robot path during simulation

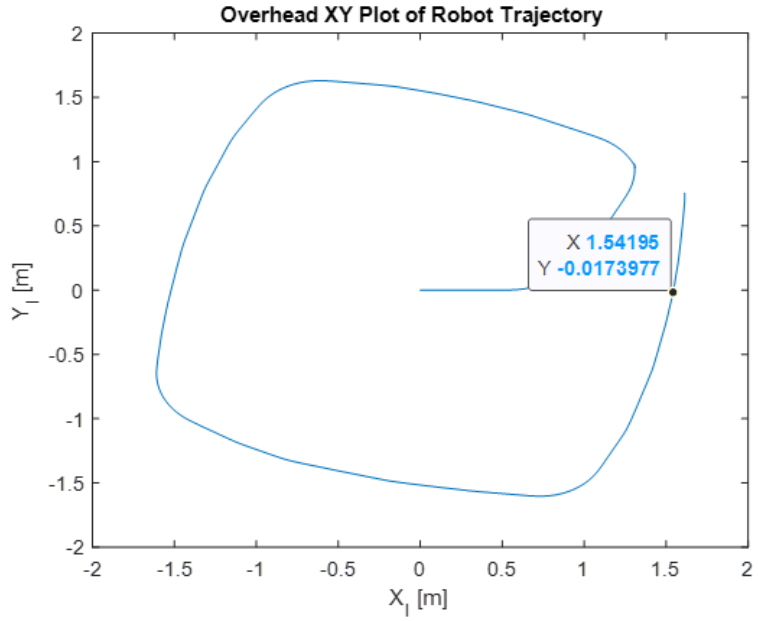

Figure 13: Overhead Plot of the robot path during simulation (1 complete round)

Part 3: Student Feedback

Overall, lab 4 successfully familiarized the group members with Simulink, and Robotics Playground app. It has created a great foundation for the final project and the members are confident to move onto the next phase that is the final project.

Some group members faced difficulty in downloading and installation of the Robotics playground app. The issue was resolved by downgrading the version of the MATLAB. Figuring out the arena layout and other related blocks did not take up a lot of time. The simulation speeds were a nice feature to figure out if the robot's path could change over a longer period of time.

Discussion

For the first part, playing around with the robot motors by adjusting the gains or speeds is a great way to understand how the simulation inputs can be configured for the robot in the environment. Achieving different types of motion was easy as it just required configuring the gains. The forward and reverse motions can be used when the robot has to achieve those motions and the pivoting and arcing motions are needed to turn the robot. But as these were constants, each simulation could only be experimented with one type of motion which is where the second part becomes interesting. Using the sensor, the motion of the robot is determined to make it autonomous in the environment. The switch blocks were used when and where a condition determines the output (similar to an IF-ELSE block in MATLAB code). The arcade was designed to add the forward/reverse and left/right turn input for the left rotor but to negate them for the right rotor The same speeds were provided for both which made it a bit difficult to achieve arcing motion. But since pivoting motion was preferred, it was easier in the function block to place the constraints/conditions to determine the speeds of the motor and when needed a left turn was achieved.

For d), the switch blocks determine which value of speed is to be supplied for longitudinal(forward/reverse) and lateral(left/right) movements and in arcade the resultant value for each rotor is supplied as described. The values of speeds for each motor is limited at (+/-) 127. So, it can't go higher or lower than that. The longitudinal motion has a threshold of 0.7 below which the forward motion stops and reverse begins. The lateral motion would be triggered when the distance is less than 1.2 and a left turn is made. So, when the robot is between 0.7 and 1.2 from the wall it has both longitudinal and lateral motion. For achieving a smoother motion, the turn has to be made for a specific amount of time when a wall is detected and before recalibrating the distance. This would require changing the simulation robot system and might require additional components like sensors on the robot. And for achieving the turn in both directions, there might be a need for additional constraints or conditions which would in turn require adding additional sensors to the robot.

For e), the entire block has to be written as a function of MATLAB and was used in the MATLAB function block of simulink. Two switch blocks, arcade and thresholding were done in a single function. The performance of this simulation was almost identical to that of the previous one in d) and so were the constraints/thresholds for longitudinal

and lateral motions. But the problem with these thresholds is that the corners of the environment are kept almost untouched i.e. the robot can't go into those areas. This can pose a problem in certain cases like a Roomba robot trying to clean corners of a room.

The provided function of TrackPose_plot.m has an error in the formula for converting the radians to degrees which has been corrected and used to get the orientation plots. (The degrees had to be reduced by a factor of 2.) Updated code has been provided in the appendix.

References

[1] Introduction to Autonomous Mobile Robots; Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza

[2] MATLAB Examples and API Reference:
https://www.mathworks.com/help/matlab/examples.html?s_tid=CRUX_topnav

Appendix

- MATLAB Code for Simulink block

```
% function [leftMotor, rightMotor] = trackWall(distance)
%
% threshold = 1.2;
% topSpeed = 127;
%
% if distance > threshold - 0.5
%     forRevMotion = topSpeed;
% else
%     forRevMotion = -topSpeed;
% end
%
% if distance < threshold
%     leftRightMotion = -topSpeed;
% else
%     leftRightMotion = 0;
% end
%
% leftMotor = forRevMotion + leftRightMotion;
% rightMotor = forRevMotion - leftRightMotion;
%
% if leftMotor > 0 && leftMotor > 127
%     leftMotor = 127;
% elseif leftMotor < 0 && leftMotor < -127
%     leftMotor = -127;
% end
%
% if rightMotor > 0 && rightMotor > 127
%     rightMotor = 127;
% elseif rightMotor < 0 && rightMotor < -127
```

```
%     rightMotor = -127;
% end
%
% end
```

- TrackPose_plot.m

```
% % Plot XY
% figure(1)
% plot(xPose,yPose)
% title('Overhead XY Plot of Robot Trajectory')
% xlabel ('X_I [m]')
% ylabel ('Y_I [m]')
%
% % Plot thea vs. time
% figure(2)
% plot(tout,360/(2*pi)*tPose)
% title('Robot Orientation \theta vs. Time')
% xlabel ('Time [sec]')
% ylabel ('\theta [deg]')
```