ME 598-A
Introduction to Robotics
Fall 2022

Simulating a Mobile Robot

[Group-R8]
[12-12-2022]

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

The report has been prepared by:

1. Rakesh Gummapu

2. Sharvil Singh

3. Shiv Bhatt

Abstract:

In this lab, we will model a differential drive mobile robot using the Robotics Playground in MATLAB and Simulink. To simulate a differential drive robot in the Robotics Playground App, we use the MATLAB Simulink tool. We focused on the robot's motor control in this lab while modeling the function of recognizing obstacles and exploring solutions when the robot encounters obstacles.

Table of Contents

Introduction:

With the fast growth of robotics today, the precision of robots is increasing. When deciding how to manage the robot, we may consider elements such as speed and traveling distance. However, the robot can only directly control each wheel's motor speed and direction. In an ideal world, we may believe that the information represented by the two is essentially the same. In the real world, however, mistakes accumulate, resulting in a discrepancy between what the robot/program believes is happening and what really occurred. When numerous forces (such as friction) are mixed in the actual world, the data provided by the motor encoder frequently does not match the predicted data. This type of mistake is unavoidable.

What we need to do is estimate the error and adjust it ahead of time. Taking this experiment further, we can get a lot of daily applications, such as sweeping robots, reasonable estimation of expected operating speed without GPS, and so on.

Theory & Experimental Procedure:

The primary goal of this section is to become acquainted with the overall operation of the Robotics Playground App. First, we must re-save it under a different name. This helps to avoid setting conflicts. We can change the "area" where the robot moves and interacts with the obstacle environment by "checking" it. Then, as shown, connect the two constants to each motor independently and run the model by clicking the Run (play) button.

Part 1: Open-Loop Robot Motion

A.) Open and modify an existing Robotics Playground Model

In this part we have inspected the playground area and made several changes through the console and observed the result. The modifications made are listed below: -

1.) Modified the area dimension to length 10m and width 12m

2.) Increased the distance between the wheels from 0.5m(default) to 0.7m

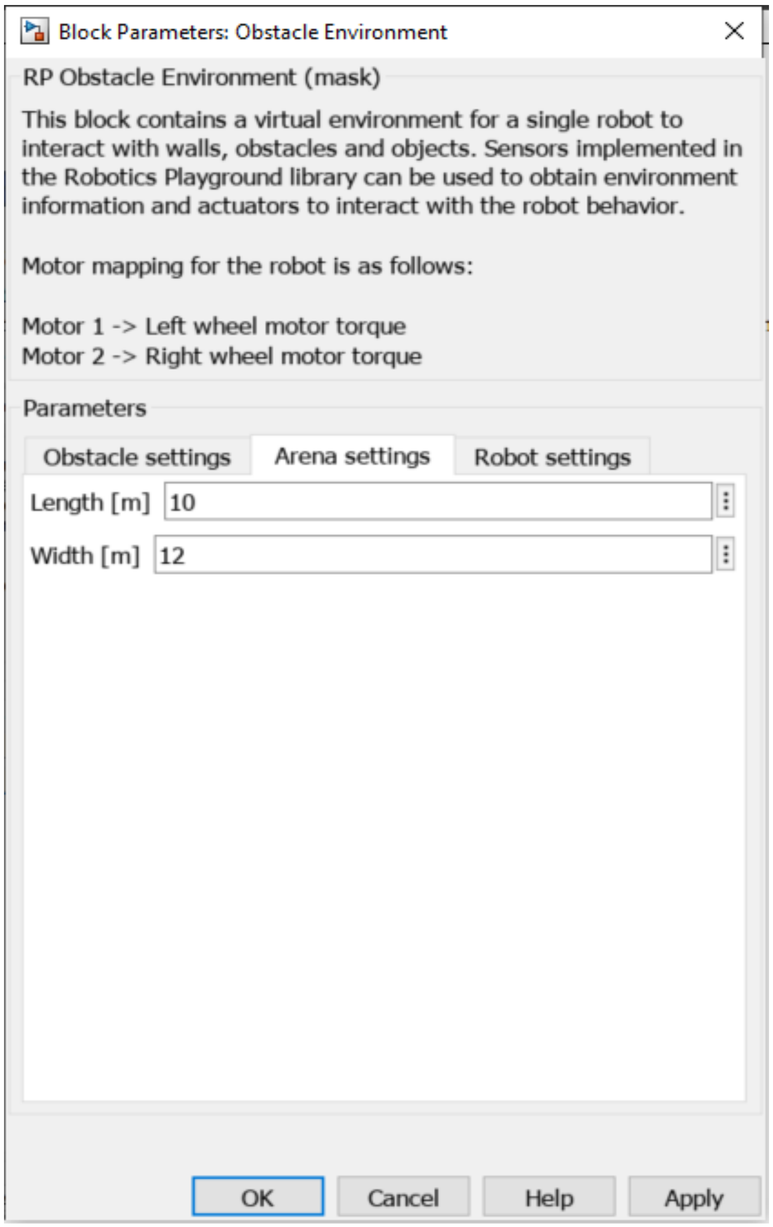
3.) Increased the wheel radius from 0.1m(default) to 0.2m

4.) Changed the RGB color combination to [0.5 0 0]

5.) Changed the Initial Position form [-1.4 0] to [-2 0]

6.) Increased the number of obstacles from 3(default) to 6 and placed them in different positions.

Below are the snaps depicting the above listed changed made through console: -



Block Parameters: Obstacle Environment

RP Obstacle Environment (mask)

This block contains a virtual environment for a single robot to interact with walls, obstacles and objects. Sensors implemented in the Robotics Playground library can be used to obtain environment information and actuators to interact with the robot behavior.

Motor mapping for the robot is as follows:

Motor 1 -> Left wheel motor torque
Motor 2 -> Right wheel motor torque

Parameters

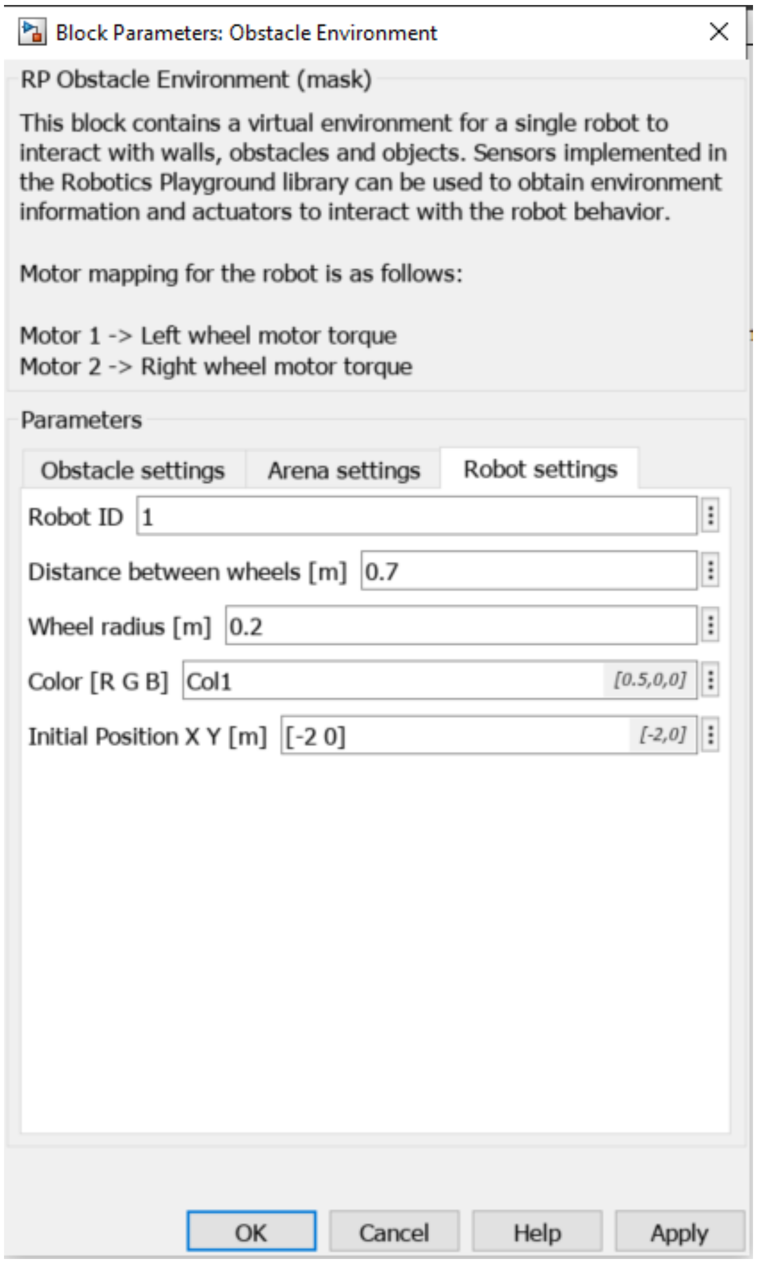
Obstacle settings | Arena settings | Robot settings

Length [m] 10

Width [m] 12

Arena Modification

Figure 1



Block Parameters: Obstacle Environment

RP Obstacle Environment (mask)

This block contains a virtual environment for a single robot to interact with walls, obstacles and objects. Sensors implemented in the Robotics Playground library can be used to obtain environment information and actuators to interact with the robot behavior.

Motor mapping for the robot is as follows:

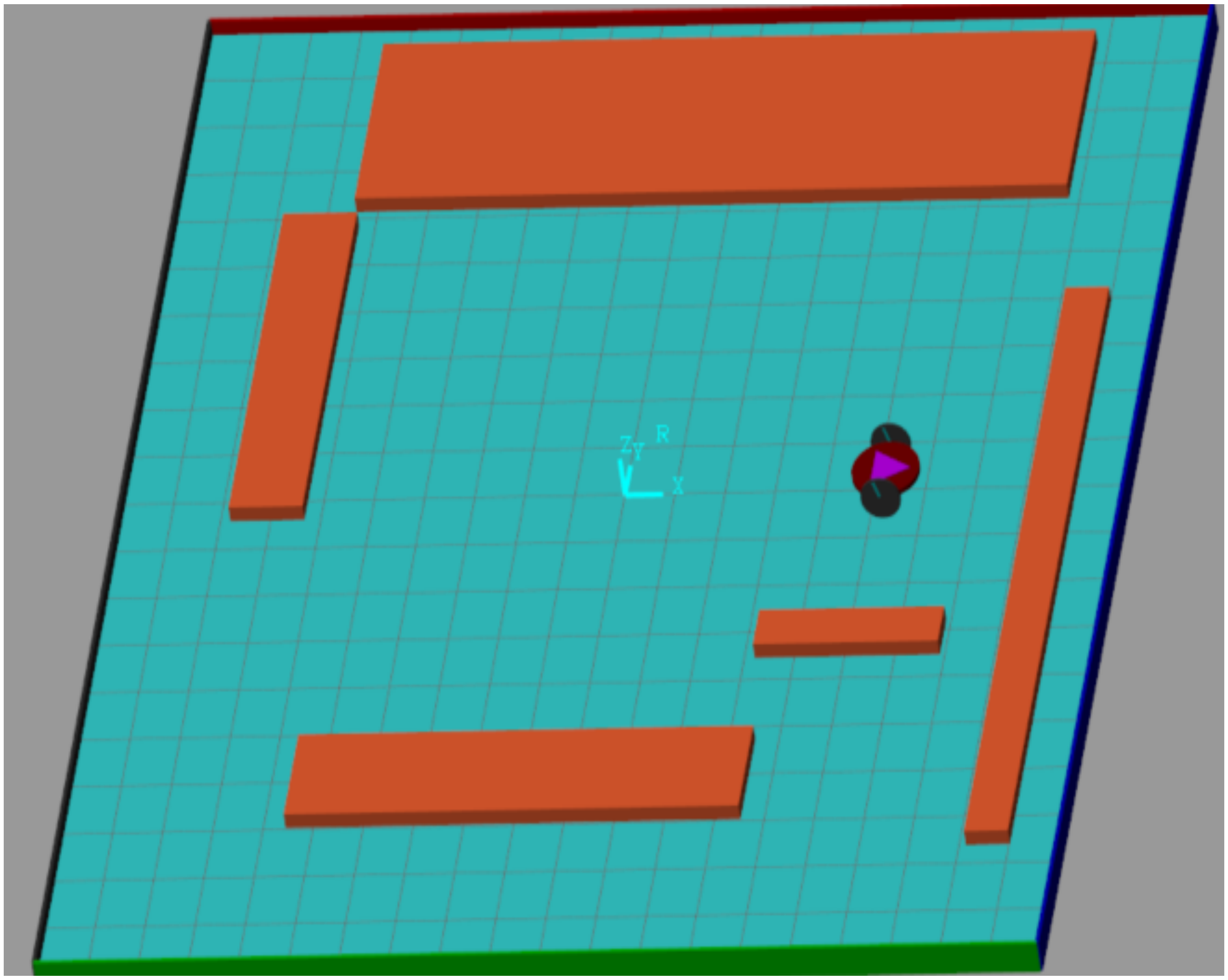
Motor 1 -> Left wheel motor torque
Motor 2 -> Right wheel motor torque

Parameters

Robot ID	1
Distance between wheels [m]	0.7
Wheel radius [m]	0.2
Color [R G B]	Col1 [0.5,0,0]
Initial Position X Y [m]	[-2 0] [-2,0]

Robot Modification

Figure 2



Obstacle setting in Robot Playground

Figure 3

B.) Setting Up Motor Speeds

In the Robot playground, we are instructed to with a Move Forward Start file. This file gives us access to the Simulink circuit using which we can modify values of the Right and the Left wheel motors of the robot and can examine different kinds of motions.

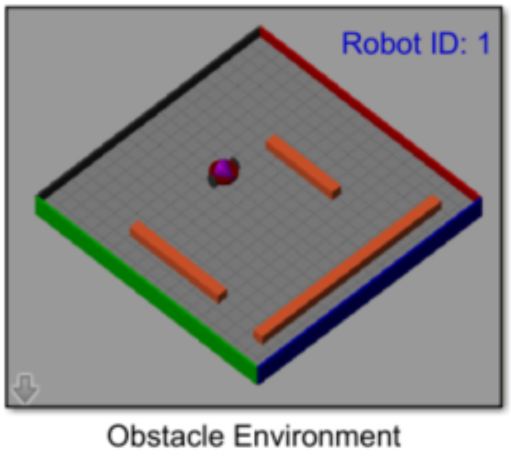
The steps we followed are: -

1.) We connect both the constant blocks and with the respective motors by dragging and dropping the connecting from constant to the respective motor.

2.) We double click on the constant block and enter the value in the pop-up window (In this case we assign the value 100 to both the blocks)

(Note – the values we provide in the blocks need to be in the range of -127 to +127)

Below are the attached snaps for the above process: -



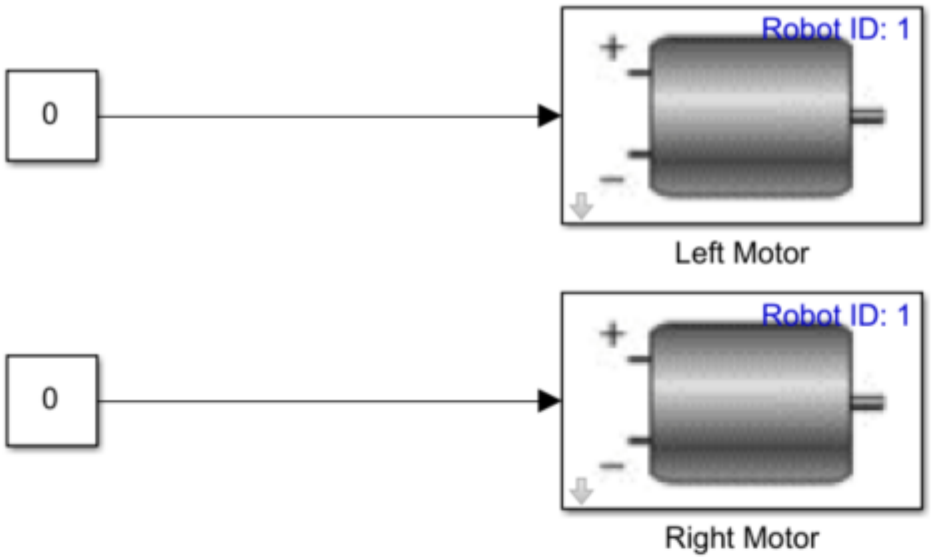
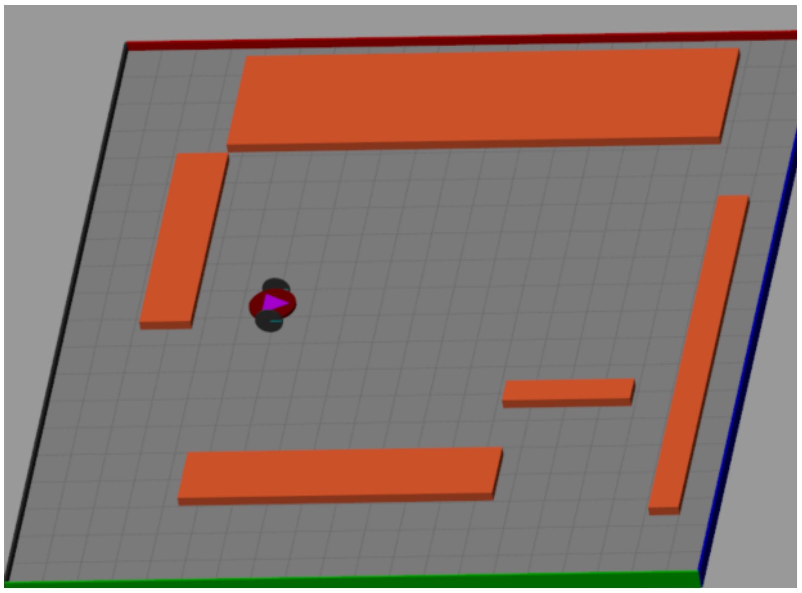


Figure 4

C.) Various motions are examined

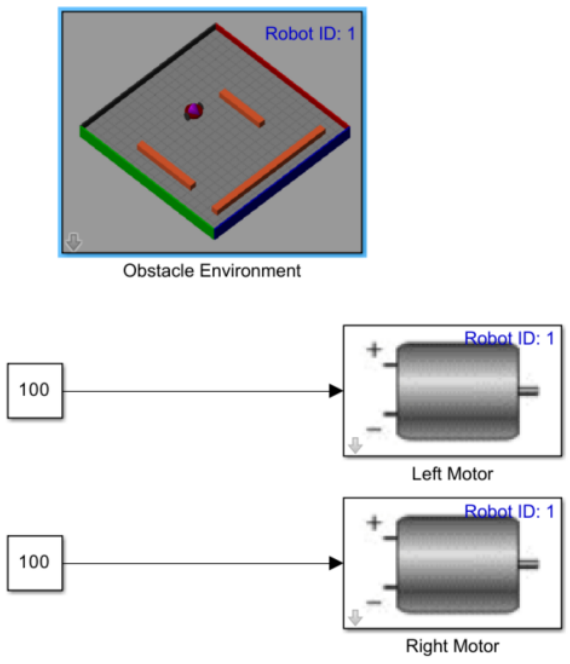
In this section we have examined different motion we can perform by changing the values of the motor value constants.

We have seen that by modifying these values we can demonstrate forward, backward, pivoting and arching motion through a robot.



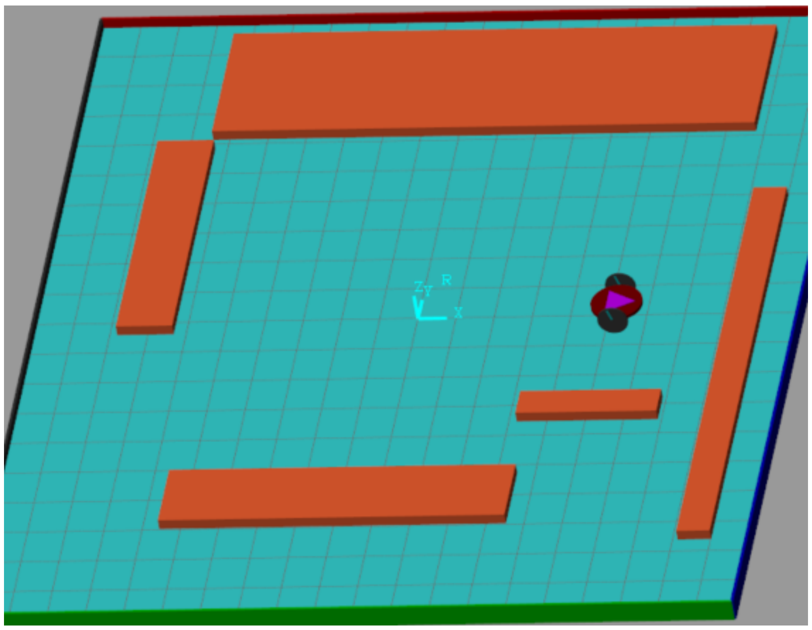
Initial Robot Position

Figure 5



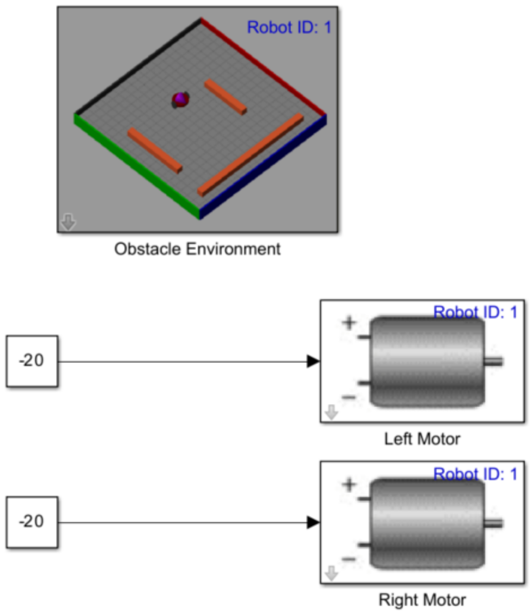
Forward Motion Parameters

Figure 6



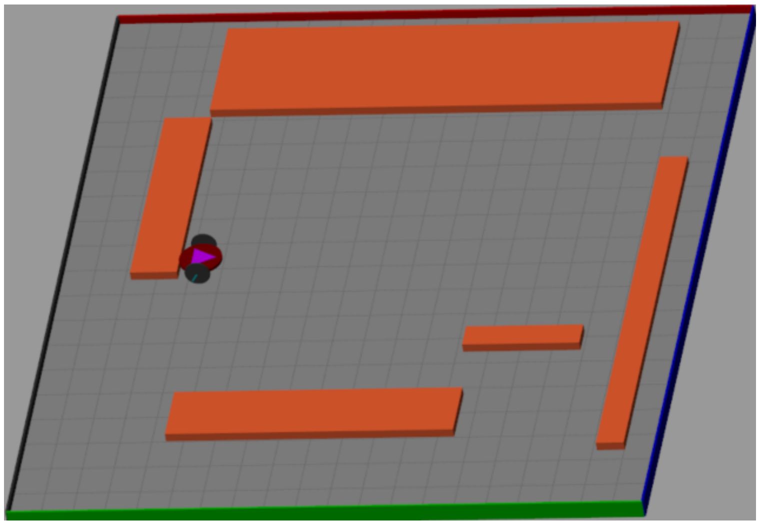
Forward Motion Final Position

Figure 7



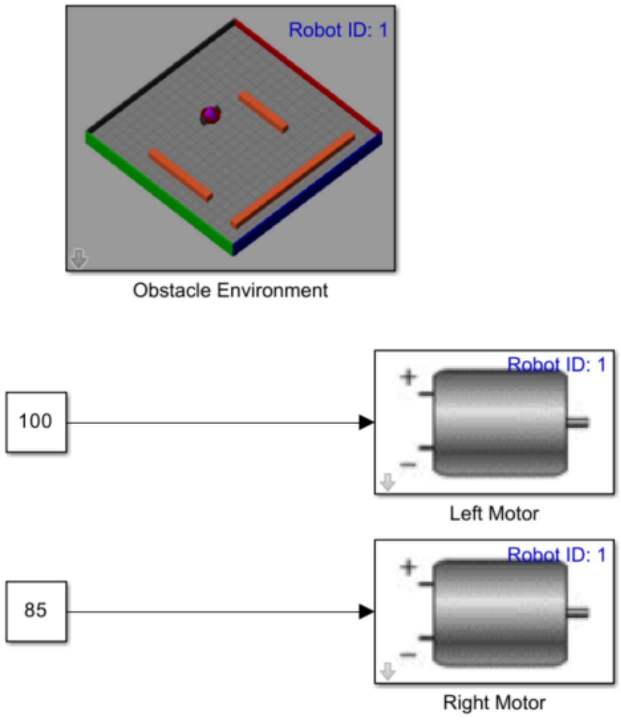
Reverse Motion Parameters

Figure 8



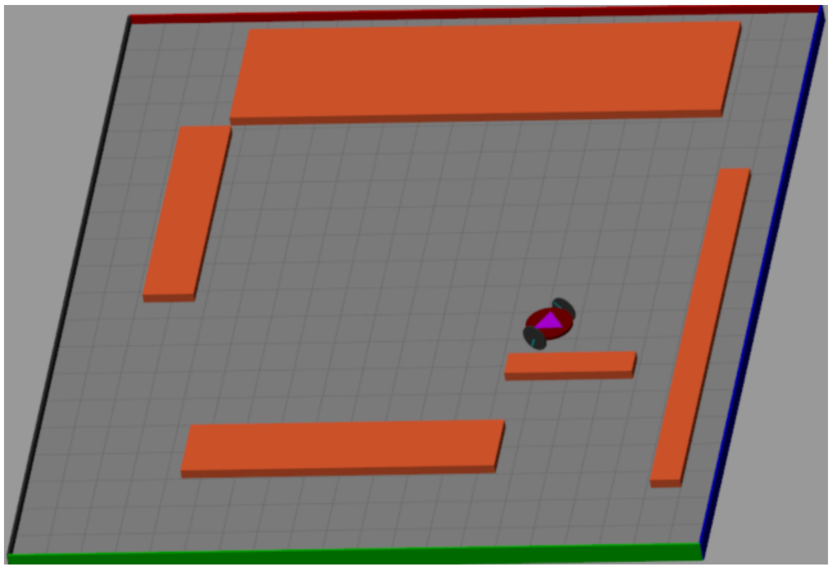
Reverse Motion Final Position

Figure 9



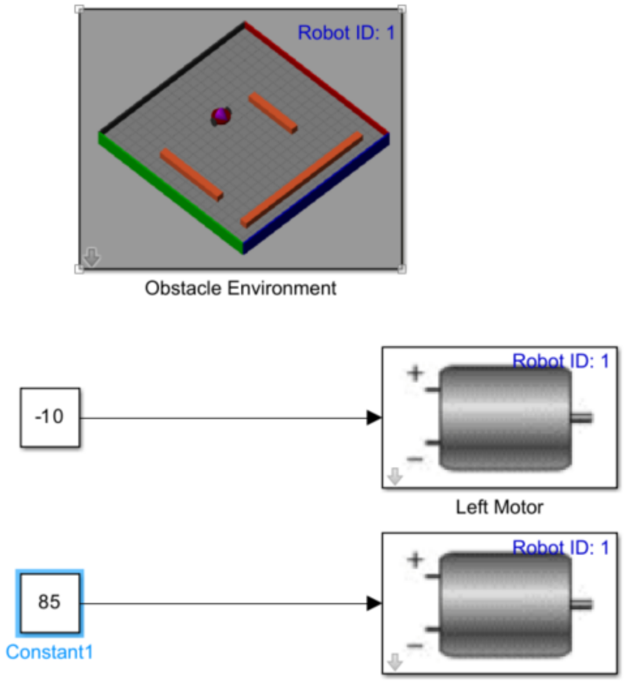
Arcing Motion Parameters

Figure 10



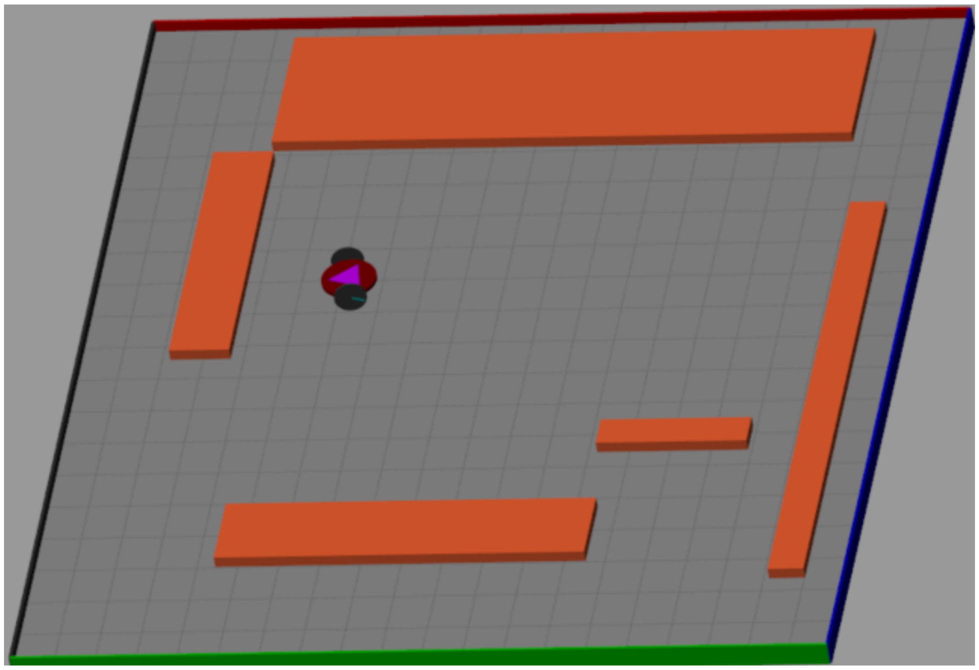
Arcing Motion Final Position

Figure 11



Pivot Motion Parameters

Figure 12



Pivot Motion Final Position

Figure 13

D.) Utilities of the Above motions: -

Forward: This motion is used to make the robot move in the forward direction. To achieve this kind of setting we need to give positive values to both the constant blocks.

Reverse: This kind of motion is used to make the robot move in the backward direction. To achieve this kind of motion we need to provide negative values to both the constant blocks.

Arcing: This kind of motion can be using we want to turn the robot to a certain angle. This can be achieved if we put different positive values or different negative values in both the constant blocks.

Pivoting: This kind of motion can be achieved if we want robot to make a complete turn along a pivot or want to robot to move in circular path. This can be achieved if we put a positive value in one constant block and negative value in the other constant value as per our requirement.

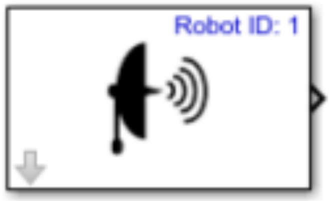
Part 2: Using Distance Sensors:

First, define the Distance Sensor block. When we set the Distance Sensor block on the virtual robot, we can receive data about the robot's closeness to other items in the playground, and we can also alter the various parameters of the sensor to match our needs. To avoid collisions, this data can be utilized to adjust the robot's direction. The Switch block is then introduced. This block is equivalent to an IF statement in that it redirects Simulink model execution based on the block input signals and the given switching criteria.

This is to avoid collisions by utilizing distance sensor data. Essentially, if the distance sensor indicates that the robot is within a particular distance of the item, the switch block will adjust the value of the wheel motor to shift the robot's direction and avoid collisions. You can add a "fcn" block, which is a blank function built with written code, to further modify the robot's behavior. This function block can be used in place of the previously used Switch block. This "fcn" block can also be adjusted to shift the robot's direction gradually based on the distance between the robot and the wall. This allows the robot to follow the perimeter of the playground.

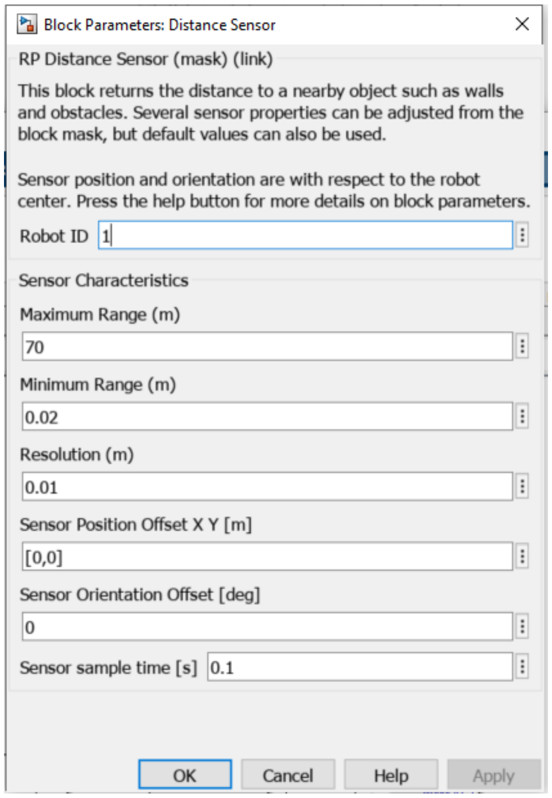
A.)Distance sensors basics

This sensor is used to know the amount of distance covered by the robot in order to know how much distant the robot is from the obstacle.



Distance senor block

Figure 14


Block Parameters: Distance Sensor

RP Distance Sensor (mask) (link)

This block returns the distance to a nearby object such as walls and obstacles. Several sensor properties can be adjusted from the block mask, but default values can also be used.

Sensor position and orientation are with respect to the robot center. Press the help button for more details on block parameters.

Robot ID: 1

Sensor Characteristics

Maximum Range (m): 70

Minimum Range (m): 0.02

Resolution (m): 0.01

Sensor Position Offset X Y [m]: [0,0]

Sensor Orientation Offset [deg]: 0

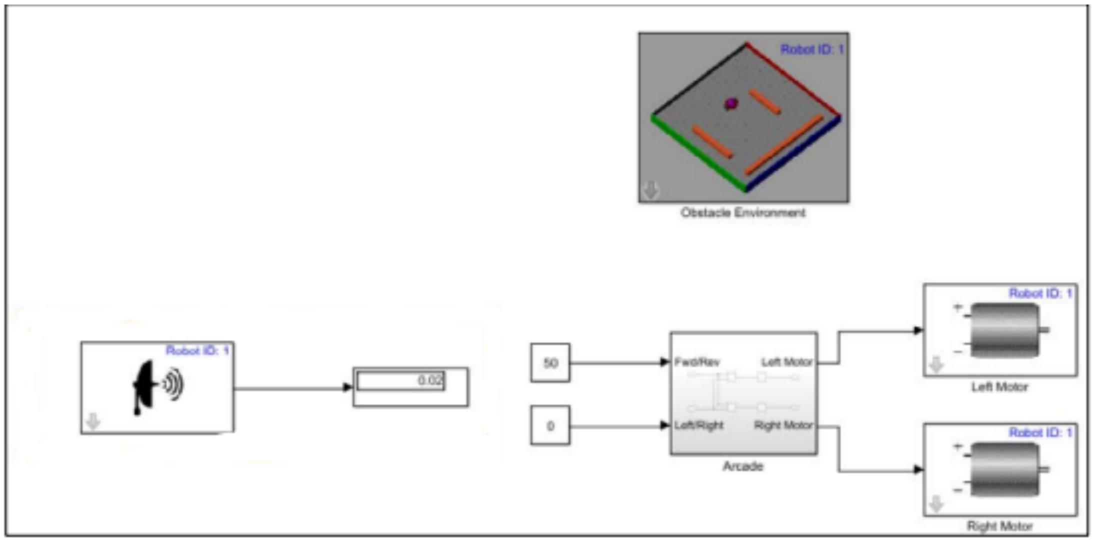
Sensor sample time [s]: 0.1

Block Parameter for Distance Sensor

Figure 15

For our robot, we put a switch block. The switch parameters are determined by the distance sensor data. In the absence of a trigger condition, the robot continues to move ahead until the distance sensor output falls below or equals a specific threshold. When this condition is fulfilled, the Switch module comes to a complete stop in order to prevent colliding with the wall.

B.) Wall Avoidance

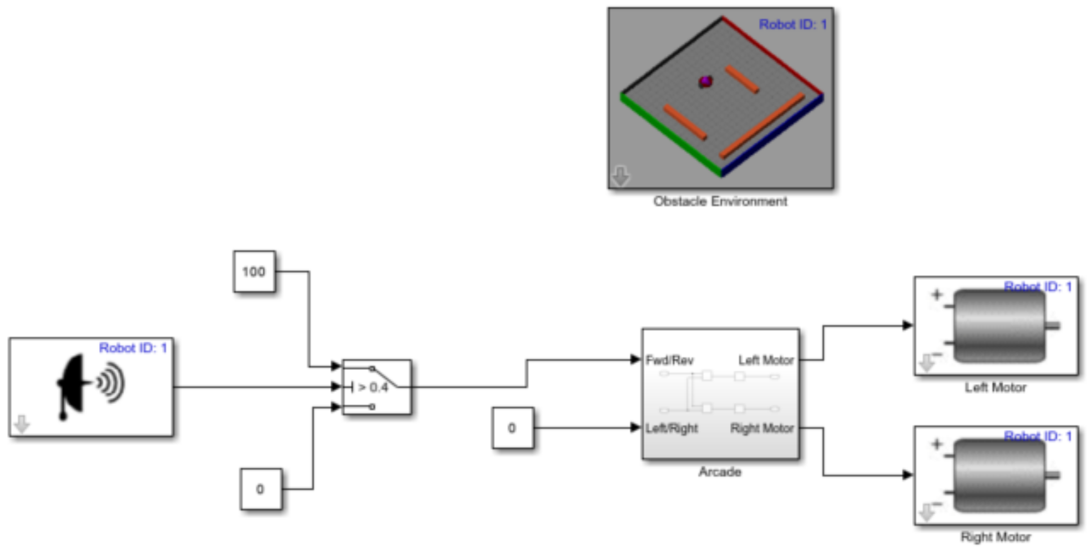


Sensor Circuit for Wall Avoidance

Figure 16

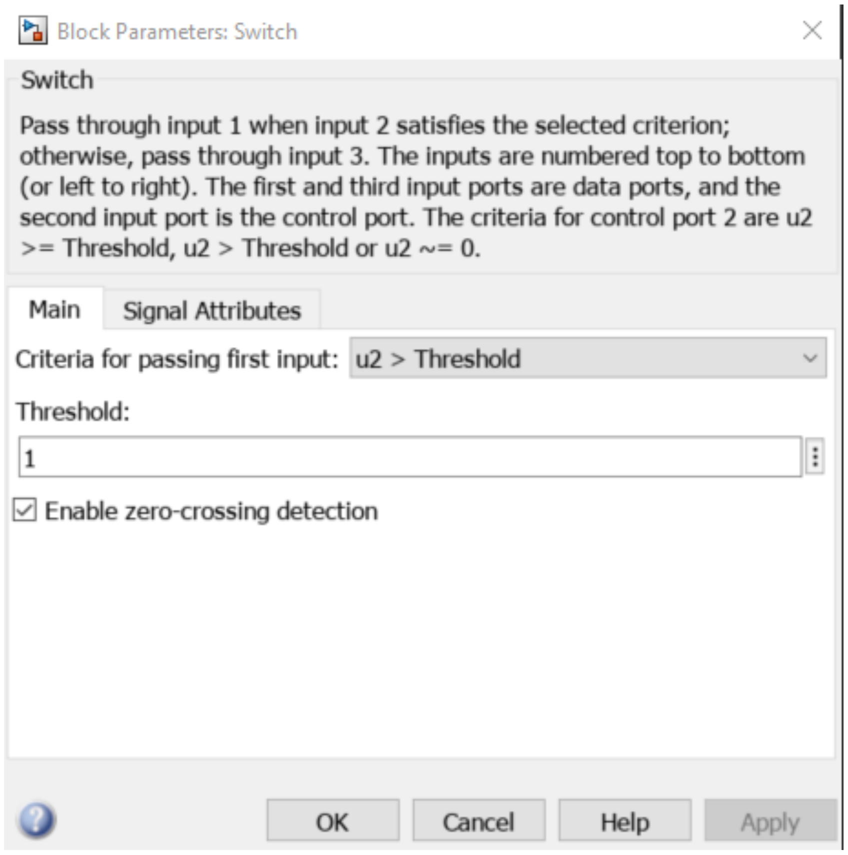
Then we created two switch blocks as well as the coding that was utilized to draw the robot movement. We've made the following changes. If the sensor reads a value greater than 1.5, the robot will continue to move forward. If the sensor reads a number less than 1.5 but greater than 1.25, the robot turns right without stopping, allowing it to complete the arc turn. If the sensor reading falls below 1.25, the robot will come to a halt and twist to avoid colliding with the wall.

C.) Using Switch Block for conditional Operations



Simulink Switch Block Within the Simulink model for Wall Avoidance

Figure 17



Switch

Pass through input 1 when input 2 satisfies the selected criterion; otherwise, pass through input 3. The inputs are numbered top to bottom (or left to right). The first and third input ports are data ports, and the second input port is the control port. The criteria for control port 2 are u2 >= Threshold, u2 > Threshold or u2 ~= 0.

Criteria for passing first input: u2 > Threshold

Threshold:

1

☑ Enable zero-crossing detection

Block Parameters for Switch Block Configuration

Figure 18

D.) Wall Tracking

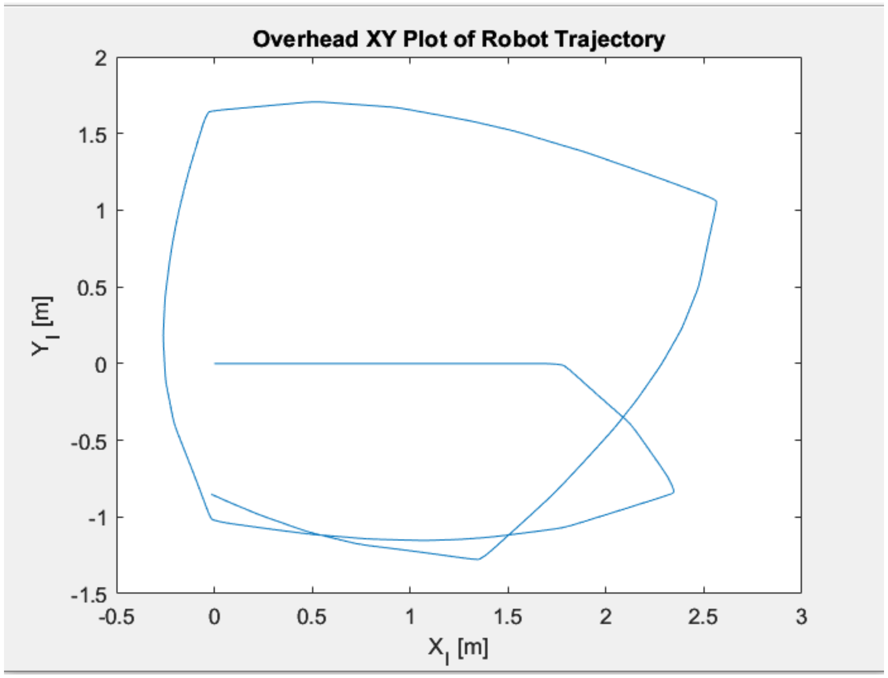


Figure 19

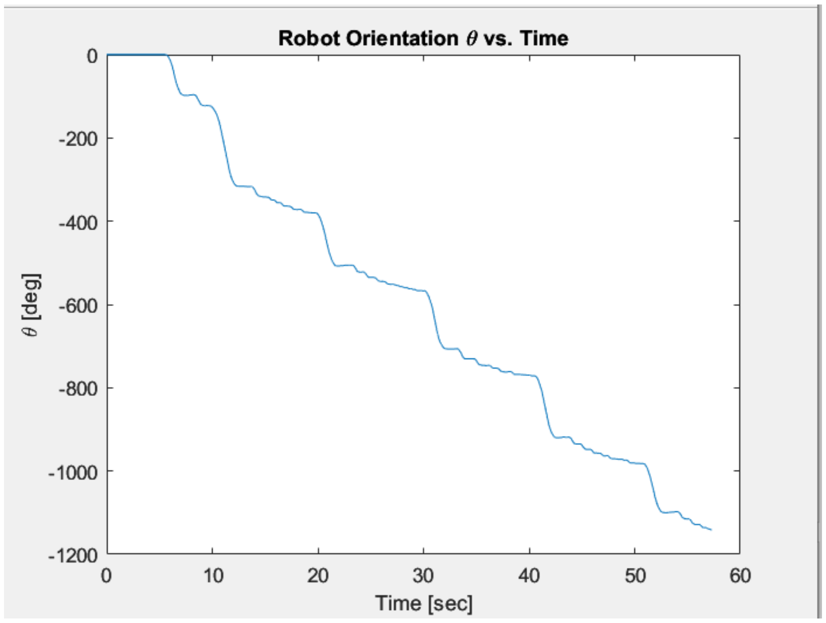


Figure 20

Finally, we modified the model to use a fcn block directly attached to the motor rather than a switch block connected via another block. The fcn block is coded as a simplified version of the preceding section's two switch blocks, and the turning speed setting is simplified. The robot will not move at the same pace as in the previous model unless the value read by the sensor is less than 1.5. When a value less than 1.5 is read, the left motor remains unaltered, but the right motor begins to move in the opposite direction at half speed.

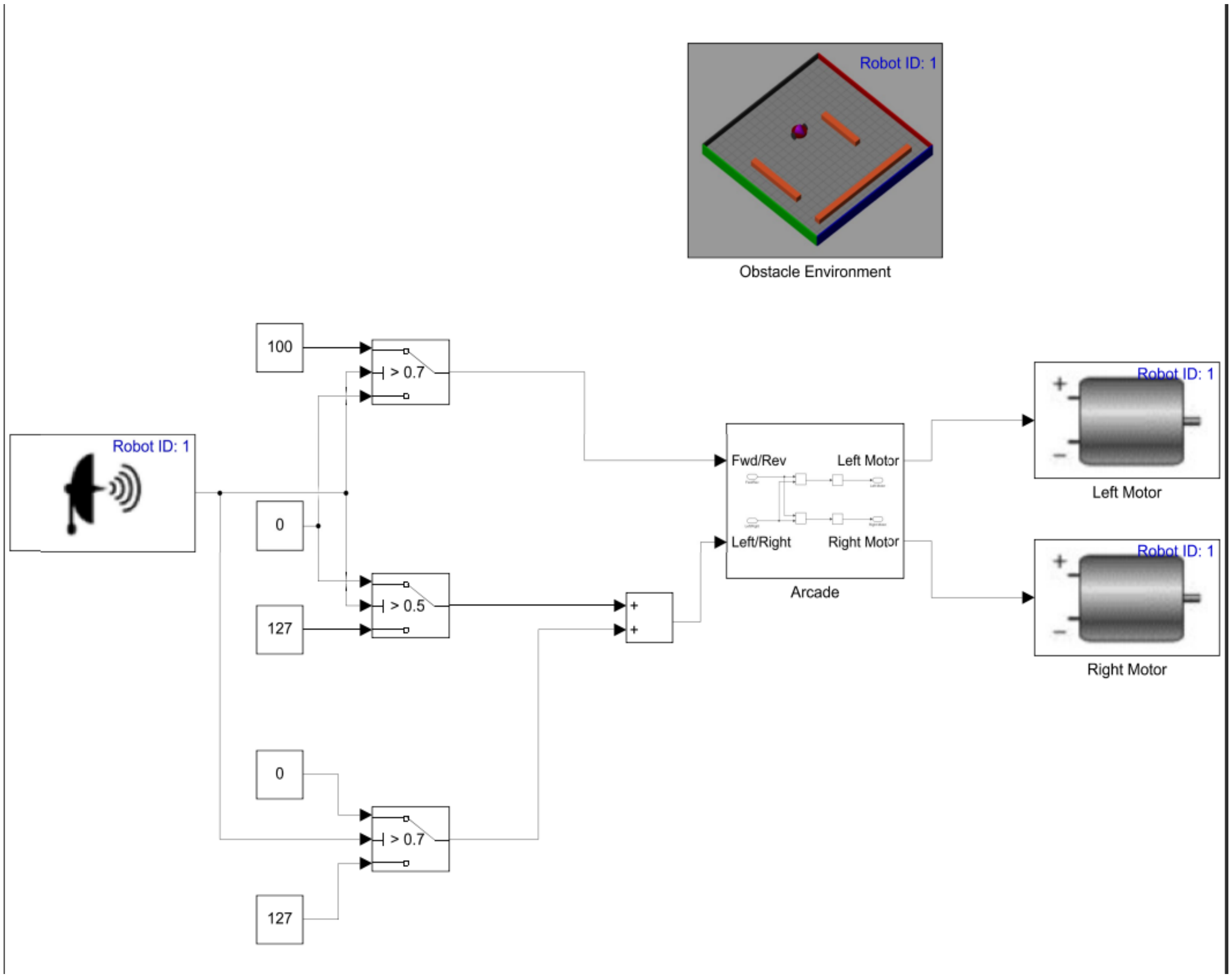
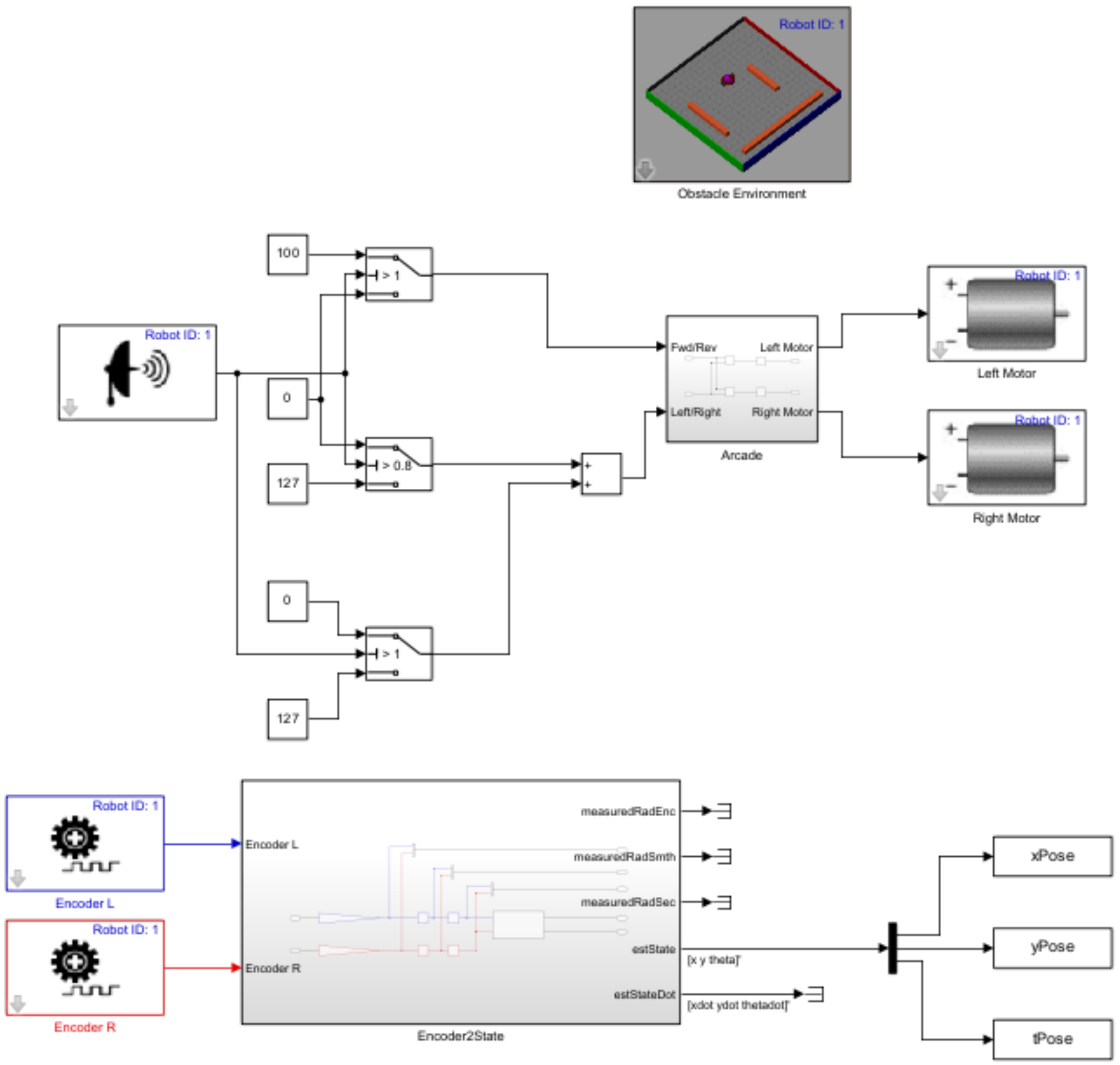


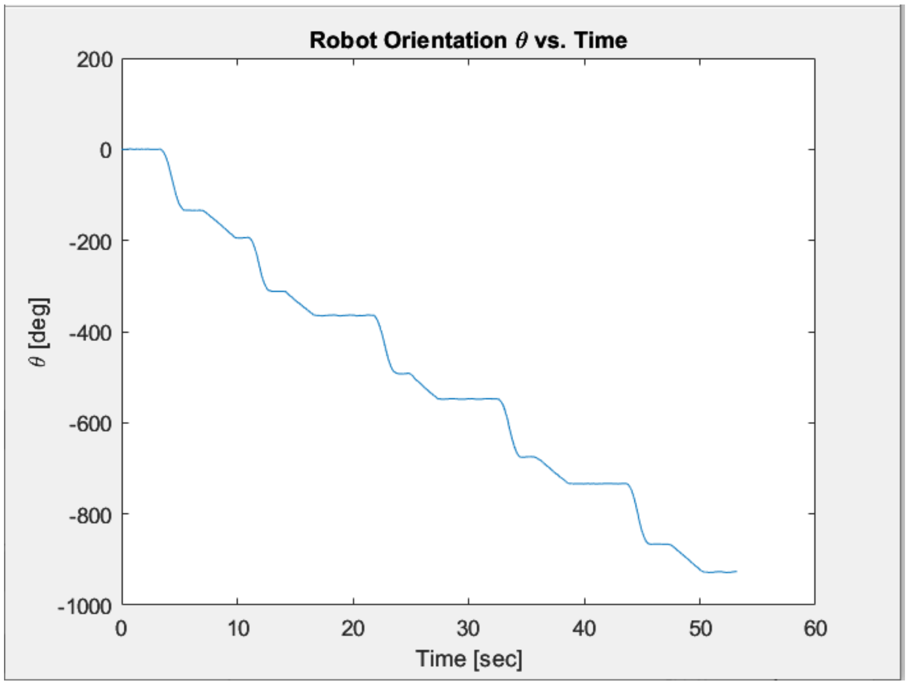
Figure 21



Wall Avoidance Starting Circuit Configuration

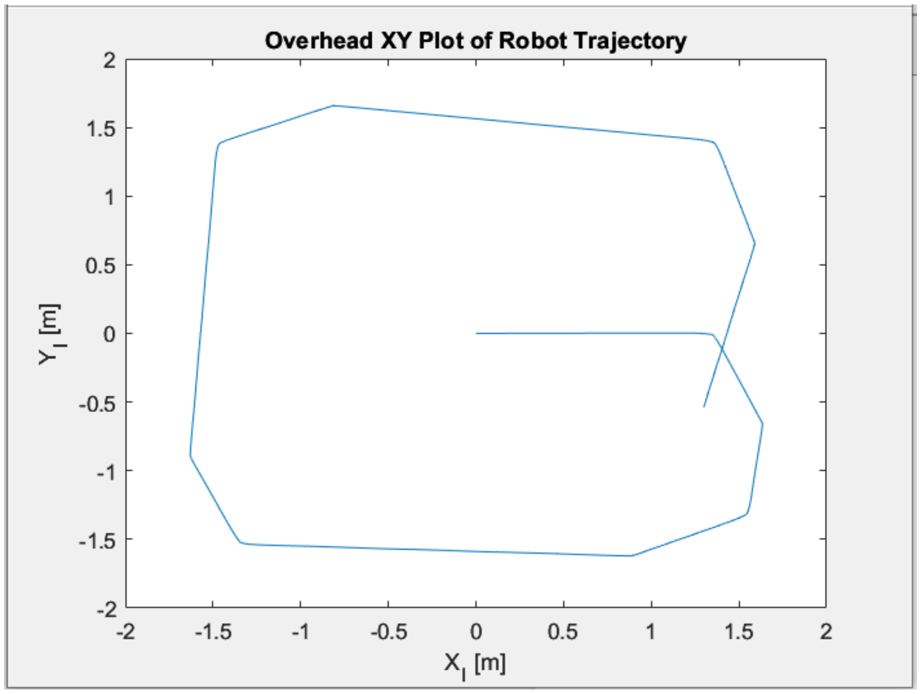
Figure 22

E.) Using MATLAB Code within Simulink Blocks



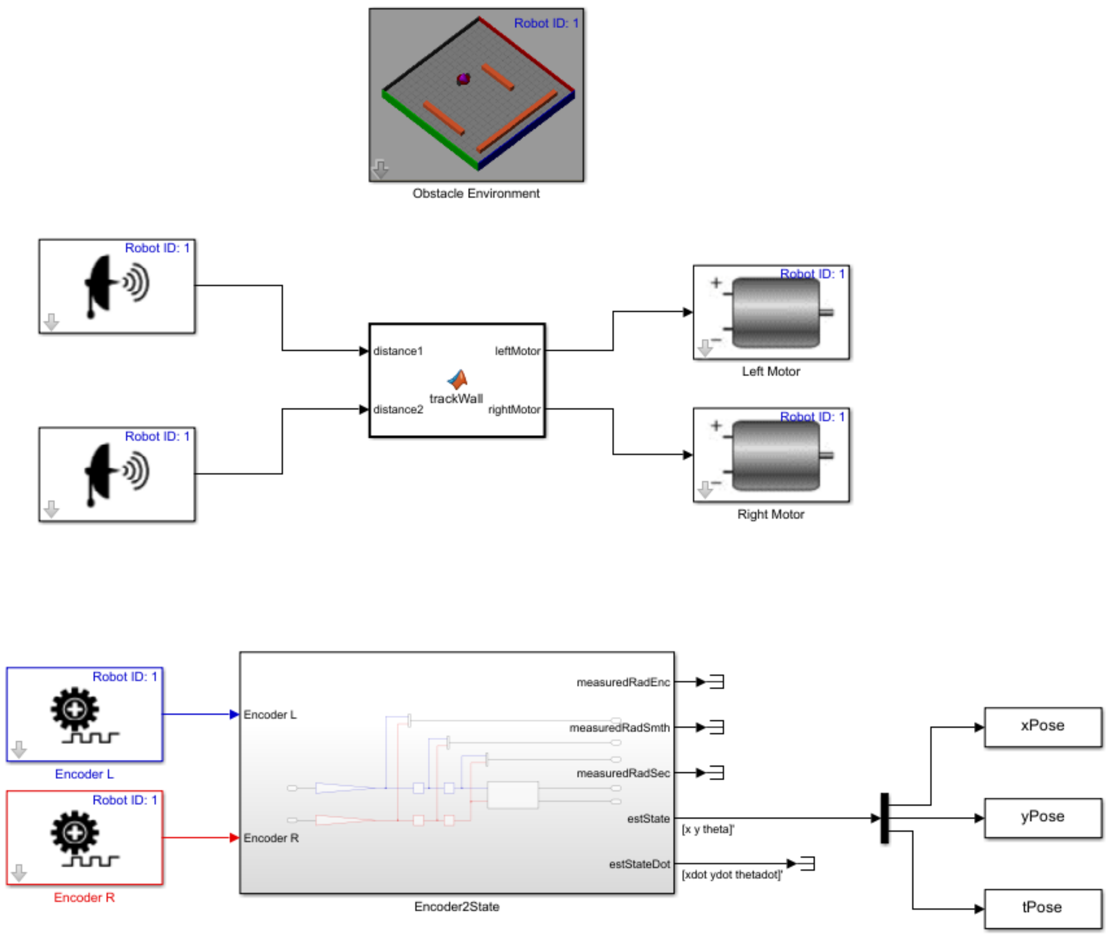
Robot Orientation Vs Time Plot

Figure 23



Final Overhead plot of Robot Trajectory

Figure 24



MATLAB Function start

Figure 25

Results:

We were able to successfully run the bot in all directions by changing the input speed given to the left and right motors. We also gave in values, greater than 127, to input which were truncated to 127.

We learned to implement the distance sensor and then, using the switch block, use it to stop the bot from colliding with the wall.

Finally, we completed the wall tracking assignment by using both, the switch blocks and the function file. Switch blocks took 61 secs to complete the task while the function file took 53 secs. The overhead xy plot of function file was much better than that of switch block.

Discussion:

In this experiment, we merely used the distance sensor data to determine whether there are any barriers ahead. This significantly reduces our overall task load and simplifies the task. Because in real trials, we require more sensor data to accomplish the entire task, which is considerably more complicated than completing this work using simulink.

Likewise, obstacle avoidance operations will become increasingly complex. When we come across a hurdle in the experiment, we merely need to give simple directions like turning left, turning right, or returning. We may also need to consider the specific torque and whether there are obstacles on the left and right sides of the robot in the laboratory or make specific obstacle avoidance judgments based on global planning.

Furthermore, when conducting this experiment, we sensed that it was part of the robot's navigation realization. The robot detects the obstruction and proceeds. However, if we look at it from another perspective, we can use this experiment as part of the mapping. Instead of recording the robot's trajectory, this program can record the location information of barriers and navigable zones relative to the robot. As a result, we only have a fragmentary map. Can we relate this map to worldwide directions and coordinates if we install a global positioning device, such as GPS, on the robot? Of course, finishing the map is not so straightforward. Every time the robot runs, it may continually rectify map faults and update the map.

Part D - We tried the switch block experiment with 2 such blocks but we couldn't get a correct output. The turn command didn't turn the bot 90 degrees when less than 0.8 units away from the wall which led to it going towards the wall and colliding with it. Therefore, we included a third condition wherein it would turn slightly if it the sensor is 0.8 to 1 unit away from the wall. This not only improved the graph and performance but also completed the task in lesser time, i.e., 61 seconds.

Part E - After the previous output, we felt the need to have a reading of how far the bot is from the wall. Therefore, we introduced another sensor on the left part of the bot facing outwards. We coded in such a way that if this second sensor detected a wall just 0.1 units or less away, it would initiate a slight turn away from it. And after that it would follow the usual procedure like the one, we did in the switch block case. This improved the performance significantly. The graph was much better than that of switch block case. Also, we completed the simulation in 53 seconds, which is the fastest we have done in this lab.

Conclusion:

In this lab4, we utilize a motor to control the robot's speed and orientation. When the motor speeds are the same, the robot moves in a straight line; when the motor speeds are different, the robot turns left or right. Simultaneously, we will replicate the process of robot detection and obstacle avoidance using distance sensors and switch modules. In general, this is an experiment based on the Robotics Playground App, carried out using MATLAB's Simulink function.

Appendices:

Function file:

```
function [leftMotor, rightMotor] = trackWall(distance1, distance2)


if distance2 > 0.1
    if distance1 > 0.5
        leftMotor = 127;
        rightMotor = 127;
    elseif distance1 < 0.5 && distance1 > 0.1
        leftMotor = 127;
        rightMotor = -127;
    else
        leftMotor = 0;
        rightMotor = 0;
    end
```

```
elseif distance2 <= 0.1
    leftMotor = 25;
    rightMotor = -25;
else
    leftMotor = 0;
    rightMotor = 0;
end
```

Track Pose File:

```
% 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/pi*tPose)
title('Robot Orientation \theta vs. Time')
xlabel ('Time [sec]')
ylabel ('\theta [deg]')
```