ME 598
Introduction to Robotics

Fall 2022

Autonomous Object Sorting using a Mobile Robot

Group # R3

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

Table of Contents

Abstract

Robots are mostly used to perform a set of tasks in a predefined manner. A simple algorithm based instruction can get the robot to perform the task but it gets complicated when the task involves decision making. In order to eliminate the manual effort involved in this, the robot needs to achieve autonomicity(autonomous nature) i.e. it has to be able to decide depending on the conditions at that point. This can be introduced by adding conditional mechanisms and let the robot choose a path among the predefined ones depending on the output of the condition. With the addition and use of right sensors and their data, it can further be simplified and the robot can become autonomous. Defining these conditions in a way that all possible workflows are executed without encountering any errors is when the robot can be labeled to be autonomous. This project aims to simulate an autonomous mobile robot under some predefined conditions and perform a set of tasks without any additional input from the user during the execution. This task was achieved using Simulink and Stateflow in MATLAB. The algorithm defined to perform the task was accurate enough and the robot achieved autonomicity to sort the pucks in their respective zones. The average trial time was about 60 secs to sort both pucks and return to the origin. The team worked as a unit to think through the algorithm and put it into action. Once the algorithm was clear, everyone had their set of tasks to perform which was integrated and tested in all possible initial conditions. The performance of the model is excellent according to the team as it was able to identify pucks' color accurately and also place them in their respective zones. The model can easily be modified if there is any change in initial conditions and can be made more autonomous by scavenging the arena and sorting any number of objects it finds.

Introduction

A Mobile Robot is a type of robot that can move from one point to another point in the physical world so as to perform various tasks. Whereas Autonomous Mobile Robots (AMR) can understand and navigate throughout the environment. The AMR has the capability to navigate through any terrain without the need of any human intervention. The AMR can achieve this with the help of advanced sensors and mapping techniques that allow the AMR's to properly comprehend and know the surroundings. The main object of this project is to navigate the robot and localize its path for object sorting in a virtual environment. Basically, we are navigating our AMR to locate the objects and transport them to a predefined position (Target). Certain conditions here are that AMR should navigate the colored pucks into a similar colored target within the arena. This will be achieved by the color detection sensor present on the AMR. To meet the project objectives, an initial simulation file with preset environment and arena settings was provided which should be used as it is for the "available robot hardware.". In the arena there are 3 target positions with different colors assigned to each target. The arena origin is established at its center [0 0], and the arena walls are each 5 meters long. The robot will always start at [-1.5 0], and two pucks will always be present at [0, 1.5] and [0, -1.5]. Each of the two pucks will be either red, green, or blue in color, with only one of each color present, though the order in which the color of the pucks appear can be random.

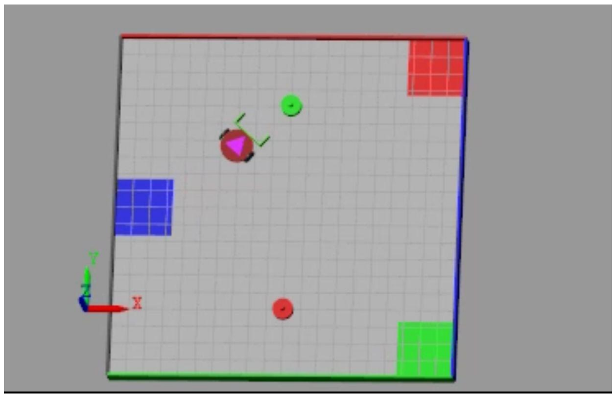


Figure 1: Overhead view of robot within the Arena

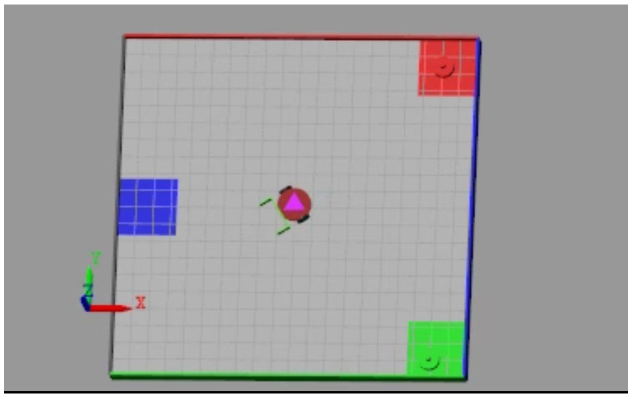


Figure 2: Successful completion of Simulation

The main objective of this project is to correctly locate the pucks and to sort them into their respective zones based on the color (Red object goes to Red zone and so on). After sorting the objects into their respective zones the robot has to return to the origin(0,0). There also is a mandatory condition that the robot shouldn't collide or scrape any walls.

Theory & Experimental Procedure

The robot's tasks can be summarized as to move around in an arena from point A to B in a repetitive manner. And since the robot only moves on the ground in a fixed environment, a 2-Dimensional coordinate system is attributed to the environment. Using basic geometry and graph theory in the algorithm, it is easy to achieve the immediate tasks and can determine what the robot should do. The robot actions can be simply divided into 4 actions, turn left or right, move forward or backward and stop. For turning left or right, a pivoting action is performed by providing the same speed but in opposite directions and for moving forward and backward, left and right motors are just provided with the same speed.

Part 1: Arena Layout and Robot Objective

The arena is 5 x 5 m in dimension with 3 colored (blue, green and red) zones present inside. It is given that:

I. The 2 pucks will always be present at [0, 1.5] and [0, -1.5].

II. The robot will start at the position [-1.5 0].
III. Two zones are at the corners of one side and the other one at the center of the opposite side. Each zone is of size ⅚m.

Since the task is to sort the two pucks, a fixed algorithm defining robot actions is defined which involves the following steps:

I. Start at the position [-1.5, 0].
II. Go to the first puck positioned at [0, 1.5].
III. Identify the color and thus the zone where the puck has to be placed.
IV. Push the puck to the designated zone while avoiding collision.
V. Go to the second puck positioned at [0, -1.5] and repeat steps (iii) and (iv).
VI. Go to the Origin of the arena at [0, 0] and stop.

Part 2: Robot Attributes

The robot has 4 sensor inputs in total:

I. The color detection camera is present on the head of the robot.
II. Three distance/proximity sensors are present on the head, left and right side of the robot.

The proximity sensor attributes are fixed and are as follows:

Maximum range = 2 m
Minimum range = 0.02 m
Resolution = 0.01

Part 3: Methods and Algorithms

To achieve the actions of the robot, certain conditions must be met and the simulation state has to be defined. These actions can be modularized into MATLAB functions as the actions are repetitive.

Turning the robot - Left/Right:

In order to proceed to the next position, the robot first aligns itself in that direction. Using graph theory, the slope of the line segment joining the robot position and the target is determined. This is achieved by using the *atan2d* function in matlab which outputs an angle in 4 quadrants. The difference between this angle and the current robot orientation tells the robot which direction it has to turn. If the angle is 0 to 180, a left turn is made and if it is negative, then the robot turns right. The robot turning stops when it is in the range of a difference of +/- 10 deg of the desired direction to account for the fast spin. So, it can correct itself again when the difference goes beyond that threshold.

Moving the robot - Forward/Backward:

Moving forward is the default action for the robot once it aligns in the target direction. It proceeds with motor gains set to 120. The robot also moves backwards with the same gains after it places the puck in its respective zone. It moves backward only till the puck is at a distance of 0.5 or more. The forward movement is carried out until the robot reaches an area around the target position as the calibration may or may not happen the exact moment it reaches the target.

Avoiding walls:

The robot has 3 distance sensors which provide the distance from the nearest wall in front, left and right directions. So, an algorithm is defined to avoid a collision with them. If there is a wall in front and none on either of the sides, then the robot moves backwards for a bit. If there is a wall on either side, then it turns in the opposite direction.

Color Detection:

When the robot approaches a puck, the distance and angle parameters are provided by the color sensor on the robot. The function only takes the input of color distances. The threshold of the sensor is low. So, when the puck is near, the distance parameter of its respective color is a number and the other distances will be *nans*. This determines the color based on which the next target position of the robot is decided.

To come up with a hard and non optimized solution, a model with just a single MATLAB function is created first which achieves the desired functionality of the robot. But this approach was complex as it would mean modifying the entire function if there are any changes in attributes. So, another approach using stateflow is followed to come up with a more modularized model. Although the output of these models is almost identical, the stateflow model is better in terms of the level of effort needed to modify them, when needed, is minimal and the understandability of the algorithm is more. Figure 3 shows the Simulink model with just a function block. Figure 4 shows another model with a stateflow chart in place of the function block. This chart is further expanded in figure 5

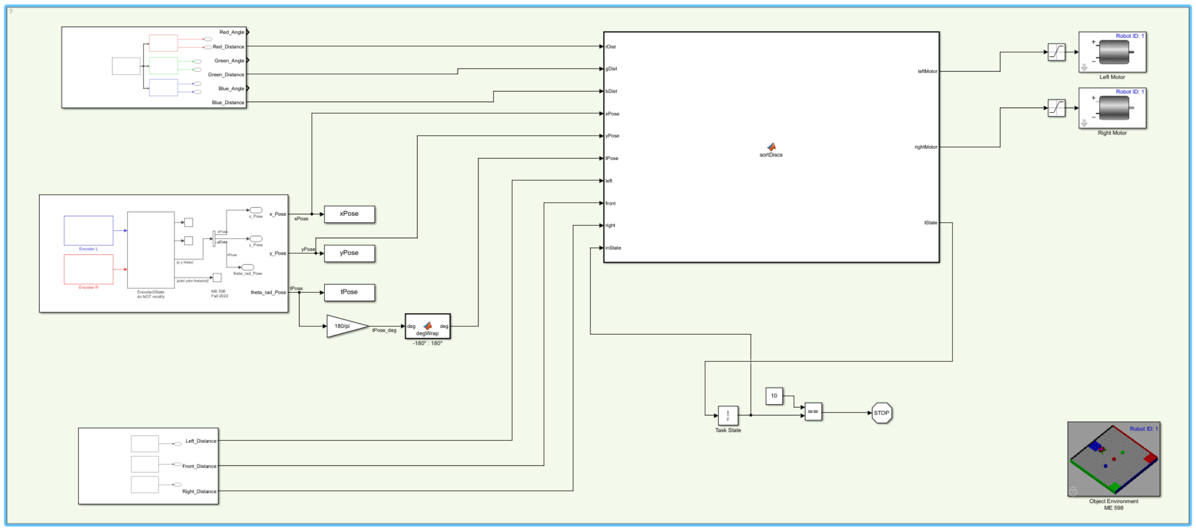


Figure 3: Snapshot of the Simulink model with a single MATLAB Function

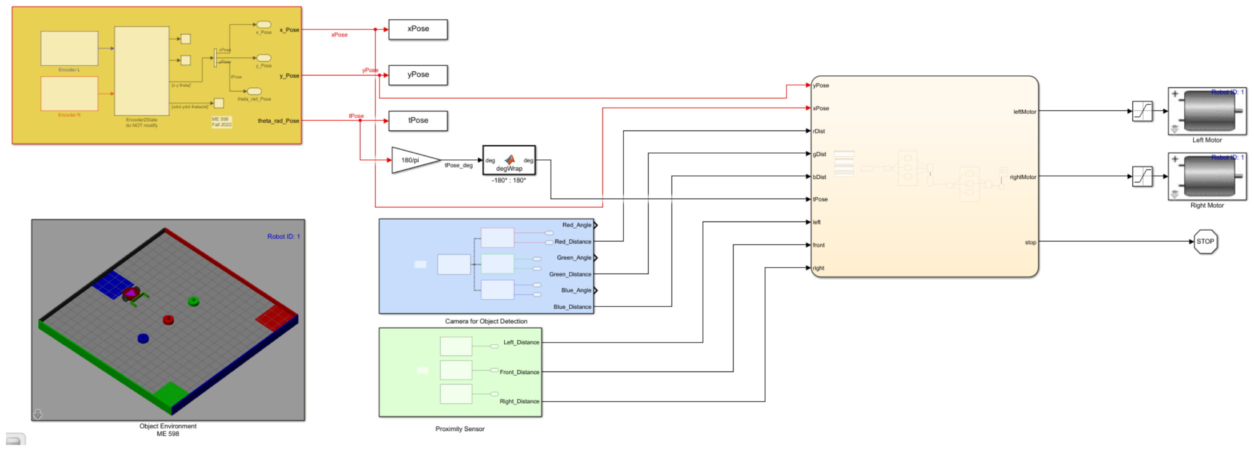


Figure 4: Snapshot of the Simulink model with a STATEFLOW Chart

The tasks are defined in part 1. To achieve each task, the robot requires the following actions, which are defined as a function.

1) **Calculate angle:** orient the robot's angle to the object. This is the desired angle.
2) **Turn Left and Turn Right:** turn the robot
3) **Move Robot:** The robot decides when to turn left, right, or go forward based on the difference between the desired angle and the current angle. This function requires both functions mentioned above.

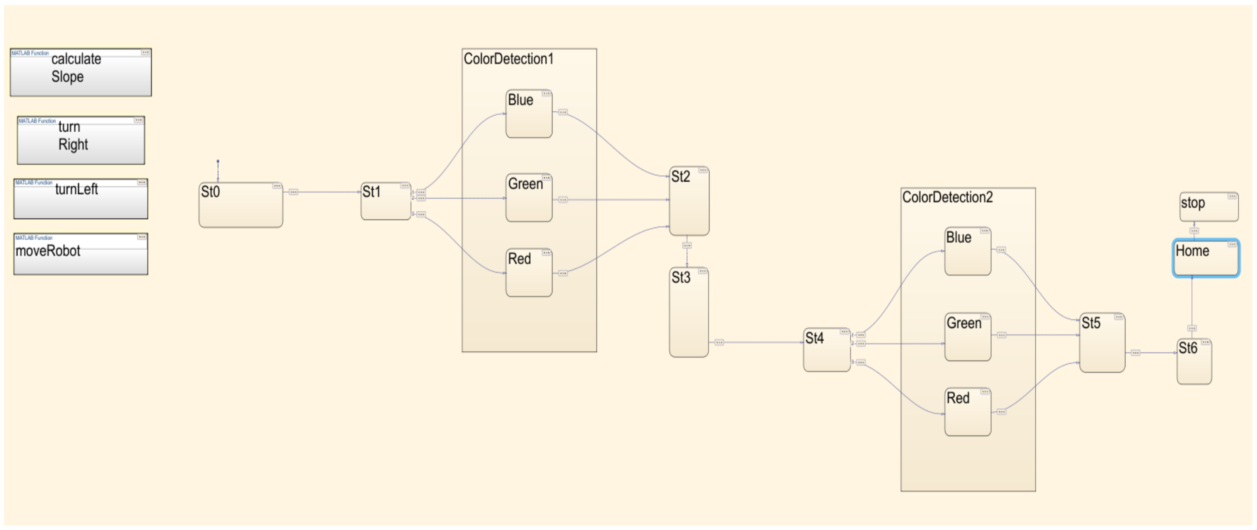


Figure 5: Snapshot of the Stateflow Chart object

After the functions are defined, the robot executes tasks, which are classified by 6 states. Details of the functions are included in the appendix.

1) **State 0:** Start from initial position and goes to the first objects location at[0, 1.5]
   Transition to state 1: -0.1 < xPose < 0.1 and 1.4 < yPose < 1.6

2) **State 1:** Detect the puck's color according to the color detector and push it to the preallocated zone: Red zone at [2,2], Blue zone at [-2,0], Green zone at [2,-2].
   Transition to state 2:
   (Red) xPose > 1.7 and yPose > 1.7
   (Blue) xPose < -1.7 and -0.3 < yPose < 0.3
   (Green) xPose > 1.7 and yPose < -1.7
3) **State 2:** Move back after placing the pucks to avoid collision
   Transition to state 3: Move away from the object about 0.5m
4) **State 3:** Proceed to the second object located at [0, -1.5]
   Transition to state 4: -0.1 < xPose < 0.1 and -1.4 < yPose < -1.6
5) **State 4:** The same as state 1
   Transition to state 5: The same as transition to state 2
6) **State 5:** the same as state 2
   Transition to state 6: The same as transition to state 3
7) **State 6:** Move to the origin at [0,0]
   Transition: -0.1 < xPose < 0.1 and -0.1 < yPose < 0.1

These states are defined within one matlab function. This function requires position, orientation, color detection, and wall distance obtained from sensors, and it outputs gains to drive both left and right motors. There is a state variable collecting state once it is transitioned and the process runs repeatedly until the last state is achieved.

The tasks can be graphically represented using the state flow. With the state flow, the system is clearly explained in that each state connects to other states through the transitions described above.

Part 4: Results and Discussion

A. Robot Trials:

In total 6 trials are conducted to check the robot's behavior for every possible initial condition. The 6 initial conditions are as follows:

Trial Number	Puck Color	Puck Color
Trial 1	Blue	Red
Trial 2	Red	Blue
Trial 3	Green	Red
Trial 4	Red	Green

Trial 5	Blue	Green
Trial 6	Green	Blue

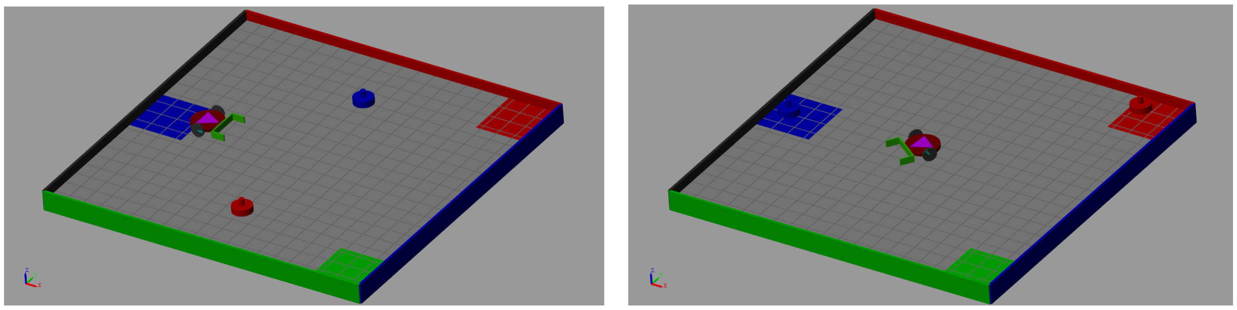
Table 1: Trials conducted with different initial conditions

The robot is successfully able to sort all three colored pucks with no collisions.

a. Red and Blue pucks

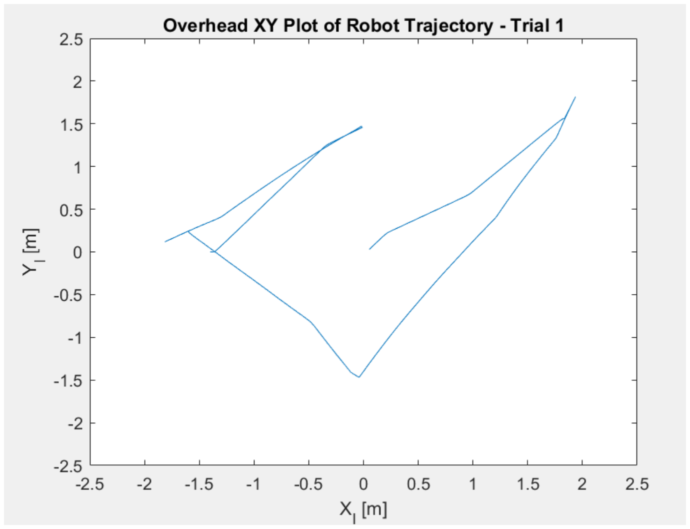
Trial 1:

In trial 1 puck 1 was blue in color and puck 2 is red in color as seen below in figure 6(a). The robot was successfully able to sort the objects into their designated zones as seen in figure 6(b) in 71.5 seconds. Additionally, there was no collision between the robot and any of the obstacles and the robot's trajectory was smooth, free of any jerky motions. Graph 1 confirms the results as the robot comes back to the home position after sorting.



(a) (b)

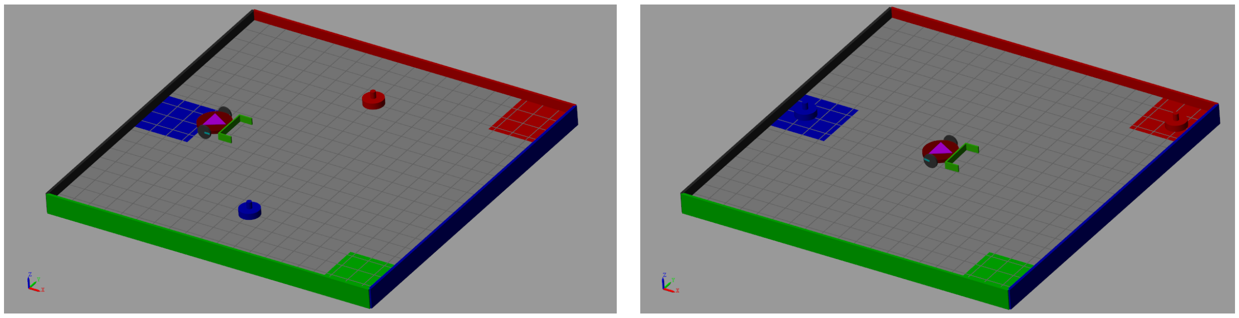
Figure 6: Arena before sorting (a) and after storing (b)the pucks in trial 1



Graph 1: Robot's trajectory for trial 1

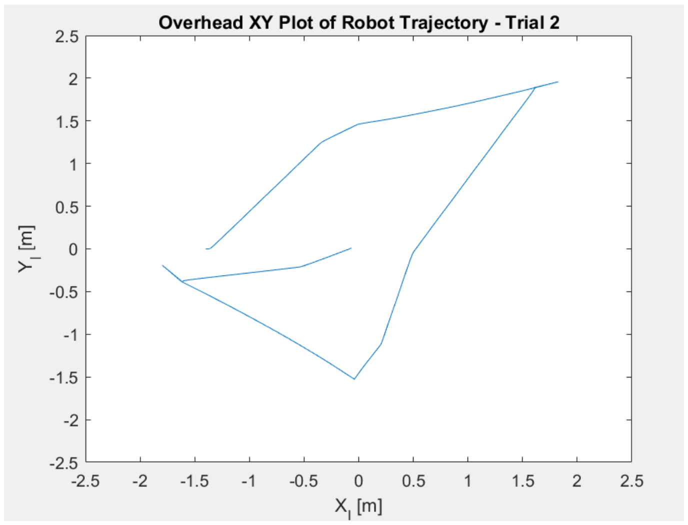
Trial 2:

In trial 2 puck 1 was red in color and puck 2 is blue in color as seen below in figure 7(a). The robot was successfully able to sort the objects into their designated zones as seen in figure 7(b) in 60.5 seconds. Additionally, there was no collision between the robot and any of the obstacles and the robot's trajectory was smooth, free of any jerky motions. Graph 2 confirms the results as the robot comes back to the home position after sorting.



(a) (b)

Figure 7: Arena before sorting (a) and after storing (b) the pucks in trial 2



Graph 2: Robot's trajectory for trial 2

b. Red and Green pucks

Trial 3:

In trial 3 puck 1 was green in color and puck 2 is red in color as seen below in figure 8(a). The robot was successfully able to sort the objects into their designated zones as seen in figure 8(b) in 72.3 seconds. Additionally, there was no collision between the robot and any of the obstacles and the robot's trajectory was smooth, free of any jerky motions. Graph 3 confirms the results as the robot comes back to the home position after sorting.

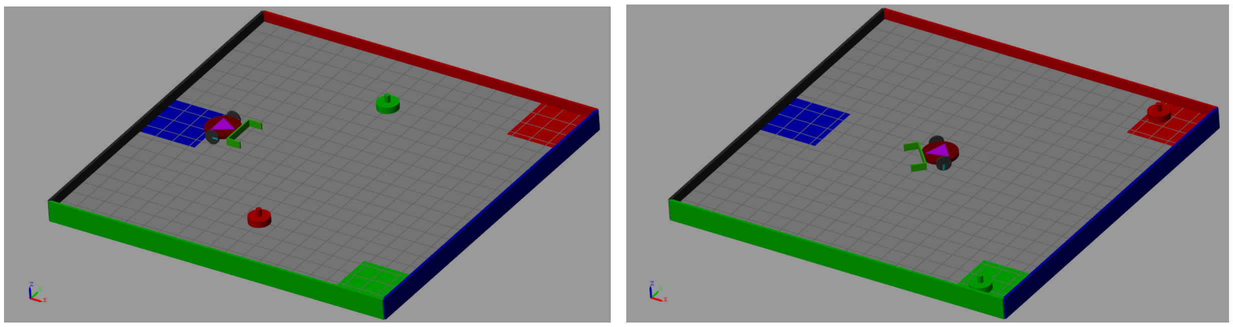
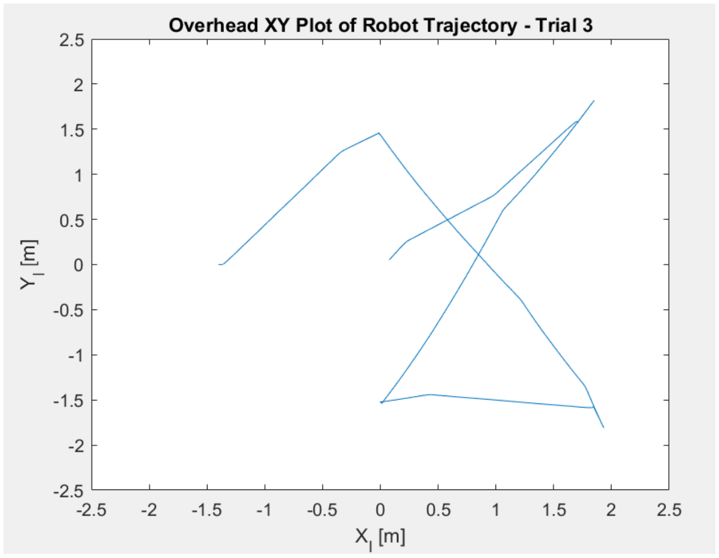

(a) (b)

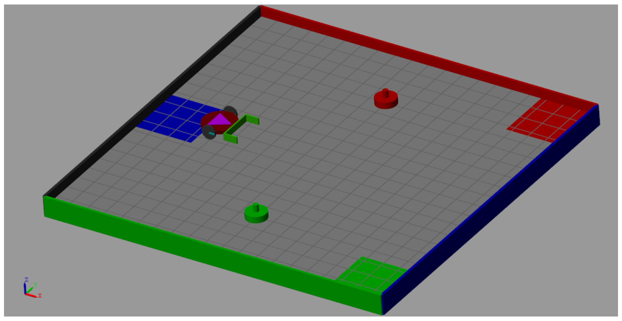
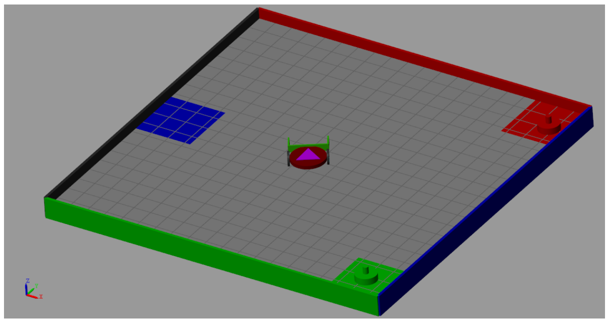
Figure 8: Arena before sorting (a) and after storing (b) the pucks in trial 3



Graph 3: Robot's trajectory for trial 3

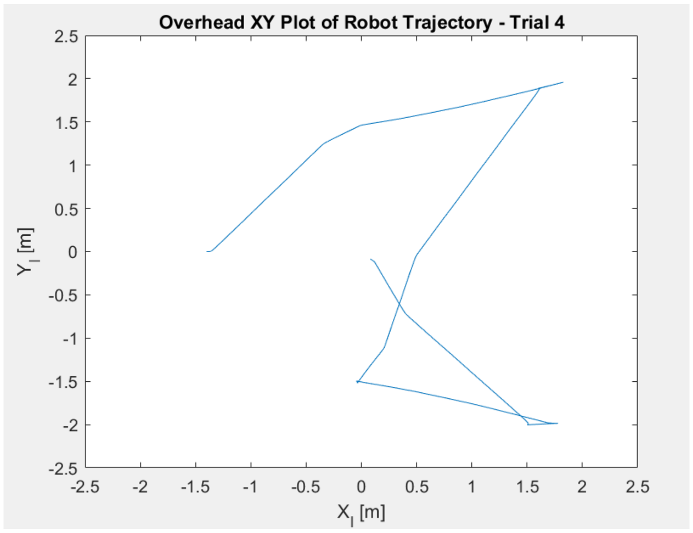
Trial 4:

In trial 4 puck 1 was red in color and puck 2 is green in color as seen below in figure 9(a). The robot was successfully able to sort the objects into their designated zones as seen in figure 9(b) in 59.5 seconds. Additionally, there was no collision between the robot and any of the obstacles and the robot's trajectory was smooth, free of any jerky motions. Graph 4 confirms the results as the robot comes back to the home position after sorting.

(a) (b)

Figure 9: Arena before sorting (a) and after storing (b) the pucks in trial 4

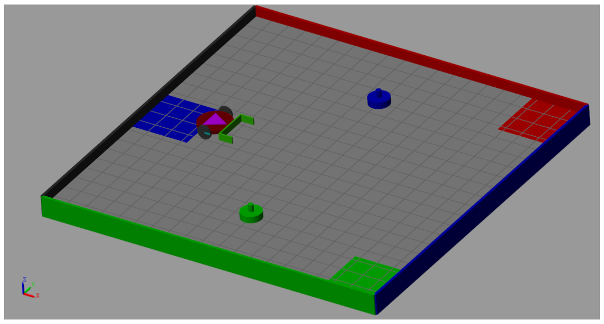
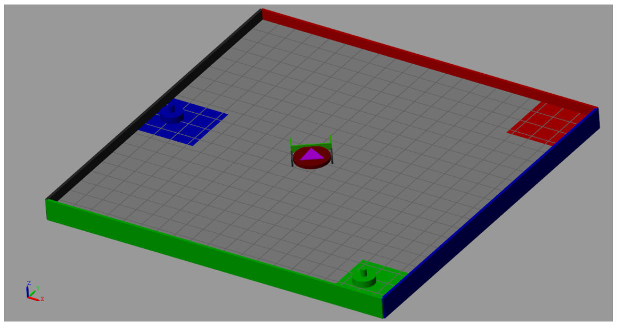


Graph 4: Robot's trajectory for trial 4

c. Blue and Green pucks

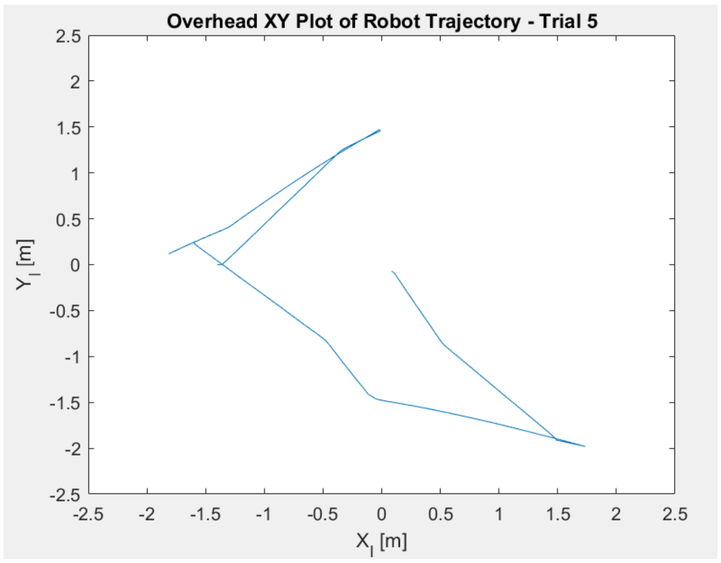
Trial 5:

In trial 5 puck 1 was blue in color and puck 2 is green in color as seen below in figure 10(a). The robot was successfully able to sort the objects into their designated zones as seen in figure 10(b) in 60.2 seconds. Additionally, there was no collision between the robot and any of the obstacles and the robot's trajectory was smooth, free of any jerky motions. Graph 5 confirms the results as the robot comes back to the home position after sorting.

(a) (b)

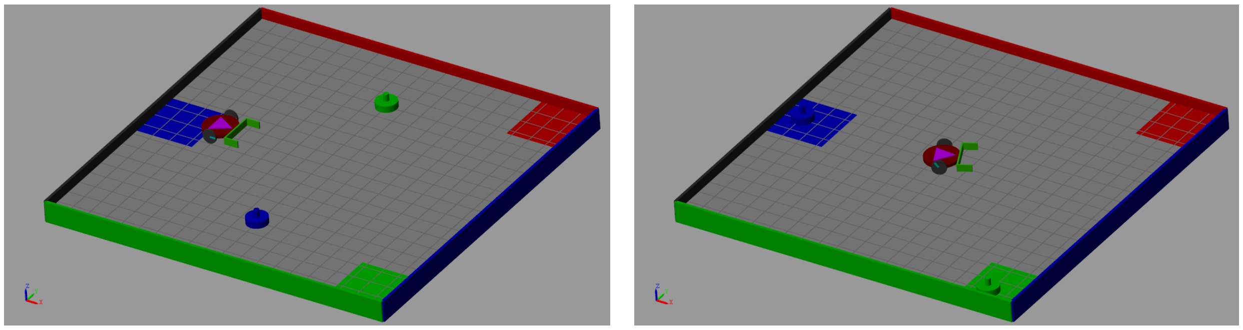
Figure 10: Arena before sorting (a) and after storing (b) the pucks in trial 5



Graph 5: Robot's trajectory for trial 5

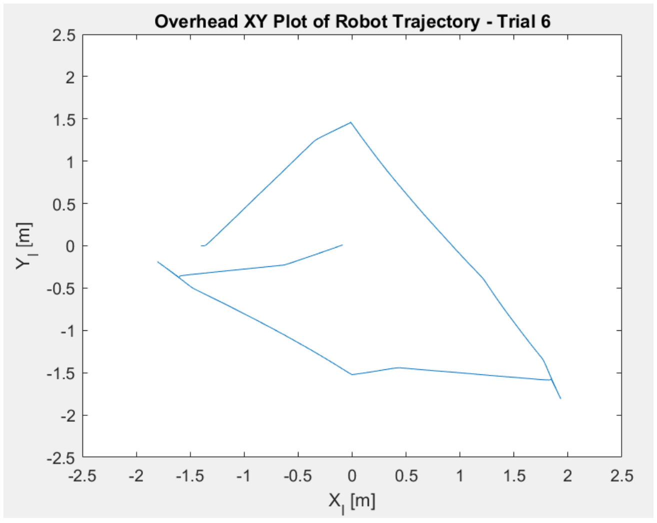
Trial 6:

In trial 6 puck 1 was green in color and puck 2 is blue in color as seen below in figure 11(a). The robot was successfully able to sort the objects into their designated zones as seen in figure 11(b) in 59.3 seconds. Additionally, there was no collision between the robot and any of the obstacles and the robot's trajectory was smooth, free of any jerky motions. Graph 6 confirms the results as the robot comes back to the home position after sorting.



(a) (b)

Figure 11: Arena before sorting (a) and after storing (b) the pucks in trial 6



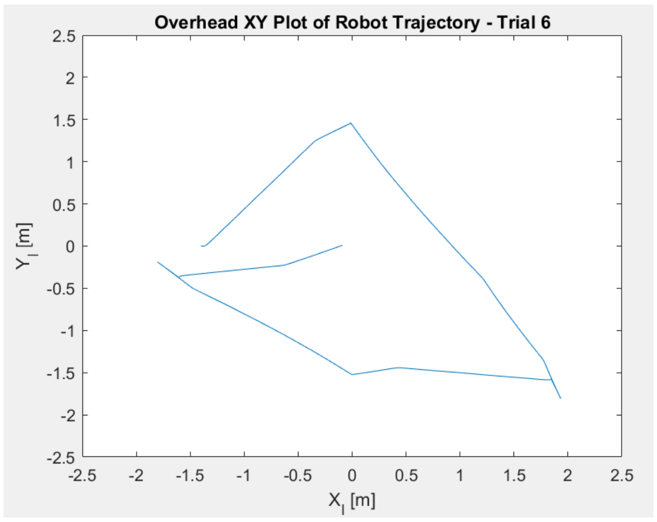
Graph 6: Robot's trajectory for trial 6

B. Accuracy and Repeatability:

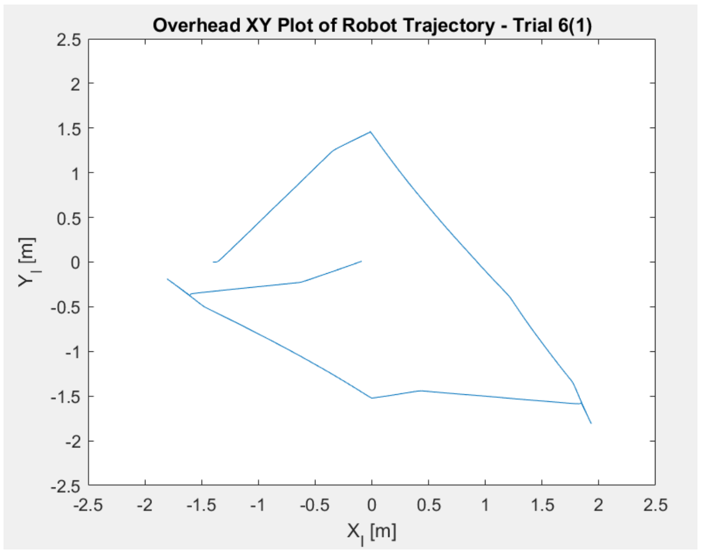
From the 6 trials presented in the above section it can be concluded that the robot is extremely accurate in:

- Identifying the color of the puck.
- Correctly identifying the place where the puck has to be placed.
- Placing the entire puck in the rightly colored part of the arena.
- Avoiding collision with the arena walls.

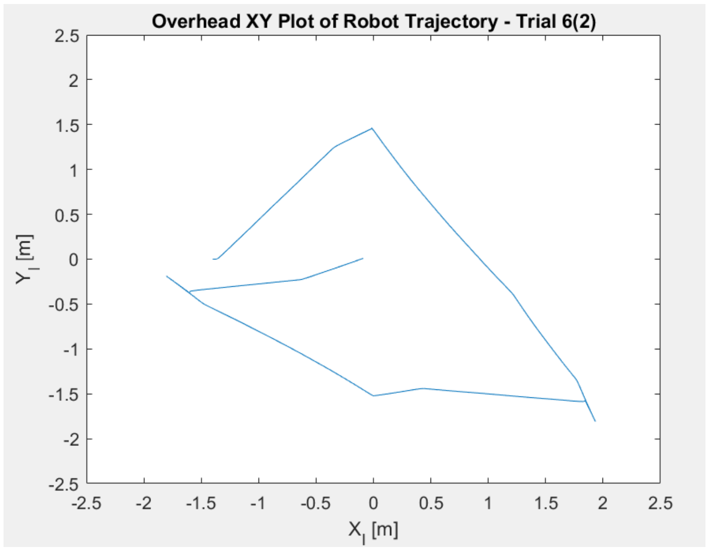
To check the repeatability of the process the simulation with green (puck 1) and blue (puck 2) is repeated three times. In each of the three trials it can be seen that the robot follows the same trajectory and successfully places the pucks in the respectively colored arena. Additionally, each trial takes exactly 59.3 seconds to complete, i.e the robot comes to a halt at position [0, 0] at the 59.3 second mark.



Graph 7: Trial 6 (repeat = 0)



Graph 8: Trial 6 (repeat =1)



Graph 9:Trial 6 (repeat = 2)

C. Video Links:

The video links of trial 3 and trial 4 using state flow and trial 3 using function block are linked below:

[Trial 4 - State Flow]:

Final Project – Autonomous Sorting, Group R3, Video 1: Shanmukha T, Peera T, Krupansh S, Sakshi P.

[Trial 3 - State Flow]:

Final Project – Autonomous Sorting, Group R3: Shanmukha T, Peera T, Krupansh S. Sakshi P.

[Trial 3 - Function Block]
[Final Project – Autonomous Sorting, Group R3: Shanmukha T, Peera T, Krupansh S, Sakshi P.]

D. Difficulties:

There were challenges when the project started as none of the teammates knew or had expertise or experience with Simulink and stateflow. And to top it off, the provided code wasn't properly working with some versions of MATLAB. Once the error was sorted out, the team was able to start working to come up with a working model. Though the logic was arrived at after a couple of sessions of brainstorming, it was difficult to model and put it to work. The easiest way was to go about the function block where the entire logic was in a single MATLAB function. But as said above, it becomes tightly coupled and might be difficult to improve/modify when needed. So, further work for a stateflow model started. Multiple tutorials were referred and trials were conducted to come up with the stateflow chart used in the final model. There were some other minor difficulties which were solved with proper coordination and communication. Once the final model was developed, it was tested with all possible initial conditions. The performance of the model was as expected and it considered all constraints and did not go out of bounds.

E. Thoughts and Remarks:

The project was a new experience for the team and it was the first time interacting with an Autonomous Mobile Robot(AMR). It gave the team an idea on how a simulation works and what kind of work needs to be done to get started and how a simulation can be used to test all possible scenarios in a faster way. Stateflow is also a nice resource and to be able to implement it in this use case is a good introduction to it. This project helped the team understand how a working model and its algorithms can be created for simulation in a way it can also be easily understood by another person without much explanation. It took the team about 50 hours of focussed work to complete the project and this report, which was another great experience to test and use our learnings from the course. The team recommends that the project be there for the future but with some more updates like adding another puck or maybe adding some obstacles in the arena although it might increase complexity.

Conclusion

This was a great experience and the project did help touch base all the learnings of the course. The team was able to use the provided resources and build on top of them to achieve the objective of the robot i.e. autonomous object sorting. Using the provided framework in simulink, we started building the model using all the sensor data from the robot as needed and figuring out the conditions that would need to be tested regularly in order to achieve the autonomicity. Color sensor data was used to identify the color of the objects the robot picks at certain positions and move them to designated zones. Stateflow was used to define the model with a robot transitioning between its states and each state has certain conditions to decide its movements. Once the task of sorting the pucks was done, the robot returns to the origin of the arena. The entire job can be completed in 58 - 73 seconds depending on the color of the pucks. There were many initial conditions assumed in the design of this algorithm and model. There might be a need to introduce a more dynamic nature for the algorithm if these conditions are to change. Furthermore, the changes can be introduced at any level, arena, environment, robot size, sensor limitations, accuracy needed and many such parameters or conditions can be changed.

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

[3] MATLAB onramp courses:
https://www.mathworks.com/help/stateflow/gs/stateflowonramp.html
https://matlabacademy.mathworks.com/details/simulink-onramp/simulink

Appendix

Function Code used in first Model

```
% function [leftMotor, rightMotor, tState] = sortDiscs(rDist, gDist, bDist, xPose, yPose, tPose,
left, front, right, inState)
%
% tState = inState;
% % leftMotor = 120;
% % rightMotor = 120;
%
%     % Calculate the required angle to turn the robot to.
%     function desiredAngle = calculateSlope(x1, y1, x2, y2)
%         desiredAngle = atan2d(y2-y1, x2-x1);
%     end
%
%     % Motor Speeds to Turn the robot to the Right
%     function [leftMotor, rightMotor] = turnRight()
%         leftMotor = 60;
%         rightMotor = -60;
%     end
```

```
%
%     % Motor Speeds to Turn the robot to the Left
%     function [leftMotor, rightMotor] = turnLeft()
%         leftMotor = -60;
%         rightMotor = 60;
%     end
%
%     % Detect Color of the disc ahead. Only called when the disc positions
%     % are reached
%     function color = detectColor()
%         if isnan(rDist) && isnan(gDist) && ~isnan(bDist)
%             color = 'blue';
%         elseif isnan(rDist) && ~isnan(gDist) && isnan(bDist)
%             color = 'green';
%         elseif ~isnan(rDist) && isnan(gDist) && isnan(bDist)
%             color = 'red';
%         else
%             color = 'none';
%         end
%     end
%
%     % Move the Disc to respective zone identified by 'color' variable.
%     % Assuming the robot has captured the disc.
%     function [leftMotor, rightMotor] = moveDiscToZone(color, nextState)
%         switch color
%             case 'red'
%                 if xPose > 1.7 && yPose > 1.7
%                     leftMotor = 0;
%                     rightMotor = 0;
%                     tState = nextState;
%                     return;
%                 end
%                 desiredAngle = calculateSlope(xPose, yPose, 2, 2);
%                 [leftMotor, rightMotor] = moveRobot(desiredAngle);
%             case 'blue'
%                 if xPose < -1.7 && yPose > -0.3 && yPose < 0.3
%                     leftMotor = 0;
%                     rightMotor = 0;
%                     tState = nextState;
%                     return;
%                 end
%                 desiredAngle = calculateSlope(xPose, yPose, -2, 0);
%                 [leftMotor, rightMotor] = moveRobot(desiredAngle);
%             case 'green'
%               if xPose > 1.7 && yPose < -1.6
%                   leftMotor = 0;
%                   rightMotor = 0;
%                   tState = nextState;
%                   return;
%               end
%               desiredAngle = calculateSlope(xPose, yPose, 2, -2);
%               [leftMotor, rightMotor] = moveRobot(desiredAngle);
%           otherwise
%               leftMotor = 120;
%               rightMotor = 120;
%       end
%
%   end
%
%   % Determine the robot motion. Turn or move forward.
%   function [leftMotor, rightMotor] = moveRobot(desiredAngle)
%       diff = tPose - desiredAngle;
%       if diff > 180
%           diff = diff - 360;
%       elseif diff < -180
%           diff = diff + 360;
%       end
%       if diff < -10
%           [leftMotor, rightMotor] = turnLeft();
%       elseif diff > 10
%           [leftMotor, rightMotor] = turnRight();
%       else
%           leftMotor = 120;
%           rightMotor = 120;
%       end
%   end
%
%   % Move robot to a position.
%   function [leftMotor, rightMotor] = moveRobotToPosition(x, y, nextState)
%       if xPose < x + 0.1 && xPose > x - 0.1 && yPose > y - 0.1 && yPose < y + 0.1
%           leftMotor = 0;
%           rightMotor = 0;
%           tState = nextState;
%           return;
%       end
%       desiredAngle = calculateSlope(xPose, yPose, x, y);
%       [leftMotor, rightMotor] = moveRobot(desiredAngle);
%   end
```

```
%
%     % Get the distance of the puck.
%     function dist = getPuckDistance()
%         if ~isnan(rDist) && isnan(gDist) && isnan(bDist)
%             dist = rDist;
%         elseif isnan(rDist) && ~isnan(gDist) && isnan(bDist)
%             dist = gDist;
%         elseif isnan(rDist) && isnan(gDist) && ~isnan(bDist)
%             dist = bDist;
%         else
%             dist = 10;
%         end
%     end
%
%     % Move robot a bit back to free the disc/puck.
%     function [leftMotor, rightMotor] = moveRobotBack(dist, nextState)
%         if dist < 0.5
%             leftMotor = -120;
%             rightMotor = -120;
%         else
%             leftMotor = 0;
%             rightMotor = 0;
%             tState = nextState;
%             return;
%         end
%     end
%
% % Based on tState, determine robot actions.
% switch tState
%     case 0
%         % Proceed to First Disc position
%         [leftMotor, rightMotor] = moveRobotToPosition(0, 1.5, 1);
%
%     case 1
%         % Take first disc to it's home position
%         color = detectColor();
%         [leftMotor, rightMotor] = moveDiscToZone(color, 2);
%
%     case 2
%         % Move a bit back to position disc in its target zone
%         dist = getPuckDistance();
%         [leftMotor, rightMotor] = moveRobotBack(dist, 3);
%
%     case 3
%         % Proceed to second disc position
%         [leftMotor, rightMotor] = moveRobotToPosition(0, -1.5, 4);
%
%     case 4
%        % Take second disc to it's home position
%         color = detectColor();
%         [leftMotor, rightMotor] = moveDiscToZone(color, 5);
%
%     case 5
%         % Move a bit back to position disc in its target zone
%         dist = getPuckDistance();
%         [leftMotor, rightMotor] = moveRobotBack(dist, 6);
%
%     case 6
%         % Move the robot back to the origin
%         [leftMotor, rightMotor] = moveRobotToPosition(0, 0, 10);
%
%     otherwise
%         leftMotor = 0;
%         rightMotor = 0;
% end
%
% % Wall avoidance algorithm. No matter where, if the wall is detected to be
% % near, the specific robot action is triggered.
% if front < 0.3
%     if isnan(left) && isnan(right)
%         leftMotor = -120;
%         rightMotor = -120;
%     elseif isnan(left) && ~isnan(right)
%         [leftMotor, rightMotor] = turnLeft();
%     elseif ~isnan(left) && isnan(right)
%         [leftMotor, rightMotor] = turnRight();
%     end
% elseif right < 0.3 && isnan(left)
%     [leftMotor, rightMotor] = turnLeft();
% elseif left < 0.3 && isnan(right)
%     [leftMotor, rightMotor] = turnRight();
%
% end
%
%
% end
```