# SYSC 4805 Computer Systems Design Lab

# Final Report

# Team Fandango

**Team 1 L1**

**Mostafa Taha**

**Flynn Graham**

**Laith Abdelrazeq**

**Faris Abo Mazeed**

**December 9th, 2022**

# Table of Contents

# 1.0 Project Charter

## 1.1 Overall Objective

The objective of this project is to develop a solution to the snow plow problem posed by Professor Taha. We are to build and develop a robot to clear a testing area of snow (represented by golf balls). The robot should clear this snow while avoiding all obstacles in the testing area. The robot should ensure it passes through the entire available testing area and clears as many if not all golf balls as possible. A variety of sensors will be used to complete a full sweep of the testing area while avoiding the mobile and stationary obstacles. The robot will need to be outfitted with these sensors and modified accordingly with a plow to aid in completing the objective. The solution will always stay within the given testing area while moving the balls out.

To accomplish this, we set forth a clear set of development tasks to be accomplished that will allow us to achieve this goal. These development activities can be found in sections 2.2 and 5.1.

## 1.2 Overall Deliverables

- Project Proposal (October 14th, 2022)
    - (Project Charter, Scope, Schedule, Cost, Human Resources)
- Progress Report (November 11th, 2022)
    - (System Architecture, Statechart, Sequence Diagram, Value Analysis Figure)
- Group Presentation (November 21st, 2022)
    - (Introduce, Explain Background, Demonstrate)
    - (13 Min Presentation, 2 Min Question/Answer)
- Lab Demonstration (December 8th, 2022)
    - (Remove Maximum Number Of Snowballs Off The Perimeter Using Autonomous Snow Plow)
    - (10 Min Demo, 5 Min Code Review)
- Final Report (December 9th, 2022)
    - (Control Charts, System/Costumer Testing Results, Working System Code)

# 2.0 Scope

## 2.1 List of Requirements

1. The robot should not exceed a speed of 30 cm/sec.
2. The robot should clear snow/balls that are encountered along its path.
3. When an obstacle is detected, the robot shall adjust its path accordingly to avoid collision.
4. When the area outline is detected, the robot shall change direction to stay inside the specified arena zone (black tape zone).
5. If the robot is stuck in a loop, the robot should use the watchdog timer to reset within 8 seconds.
6. If the robot is stuck between 2 or more obstacles or boundaries, the robot should trace back to get unstuck.
7. The robot should provide enough power for all sensors to operate.

## 2.2 List of Activities

There are 5 main categories of activities that we have defined for our development. The vehicle body category represents the physical additions we must make to our robot chassis. The protocols define how we will design the overall movement decisions of our system. The movements themselves are a category as they must be programmed. The sensors category is responsible for programming the sensors in a way that we can use them to accomplish the requirements. Lastly, the testing category is of course responsible for conducting our end-to-end tests.

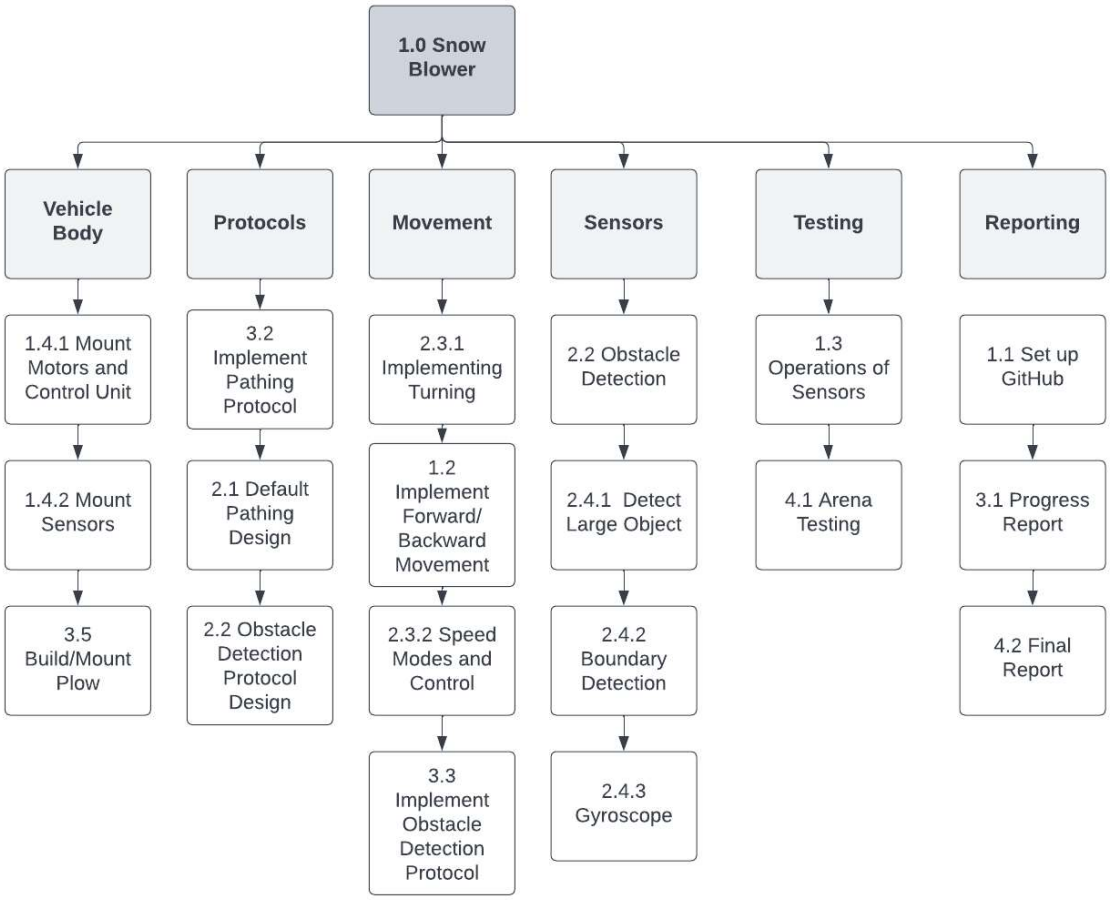


Figure 1: List of activities organized in a WBS

## 2.3 Testing Plan

The testing of our solution shall be done in a variety of stages, individual unit tests will be conducted on the sensors and motor functionality to ensure the base functions work as expected. Once those are conducted, integration tests will combine aspects together to ensure they can work in unison. Finally, an end-to-end test will be done to observe the functionality of the entire system against its objective.

### 2.3.1 Unit Tests

A variety of unit tests will have to be run to ensure the basic functionality of each component, these include:

- Testing each distance sensor at a variety of ranges. The success criteria will be if the sensor's reading is accurate to the measured distance within ± 2cm
- Testing each motor to ensure it can move backward and forward. Each motor will be run, it will pass the test if the motor is observed to run.

### 2.3.2 Integration Tests

Integration tests will be conducted in order to gauge individual pieces' abilities to begin functioning the way they are intended, these include

- Testing a variety of movements made by the robot
    - 90-degree turn (left and right)
    - 180-degree turn
    - Straight line driving
    - Hard stopping
- Testing the detection with the sensors
    - Testing detection of the outer barrier
    - Testing detection of obstacles

The success criteria of these tests will be if the robot can complete the specified action. For the movement tests, if the robot successfully performs the action, it will pass the test. For the sensor tests, we will put obstacles in front of it and take them away, if it detects the object while in front of it, and no longer when the obstacle moves, then the tests pass.

### 2.3.3 End-to-End Test

Final tests should be implemented to observe the robot completing its desired functionality, these include:

- Avoiding large stationery and mobile objects
- Moving golf balls out of the area
- Staying within the boundaries of a specified area

The success criteria of these are judged based on the final demo, the success is not a simple pass or fail, but more so a sliding scale.

# 3.0 Schedule

In this section the rough schedule for how we will develop our solution can be seen.

## 3.1 Schedule Network Diagram

Below is the schedule network diagram. See section 5.1 figure 1 for which activity corresponds to what activity number.

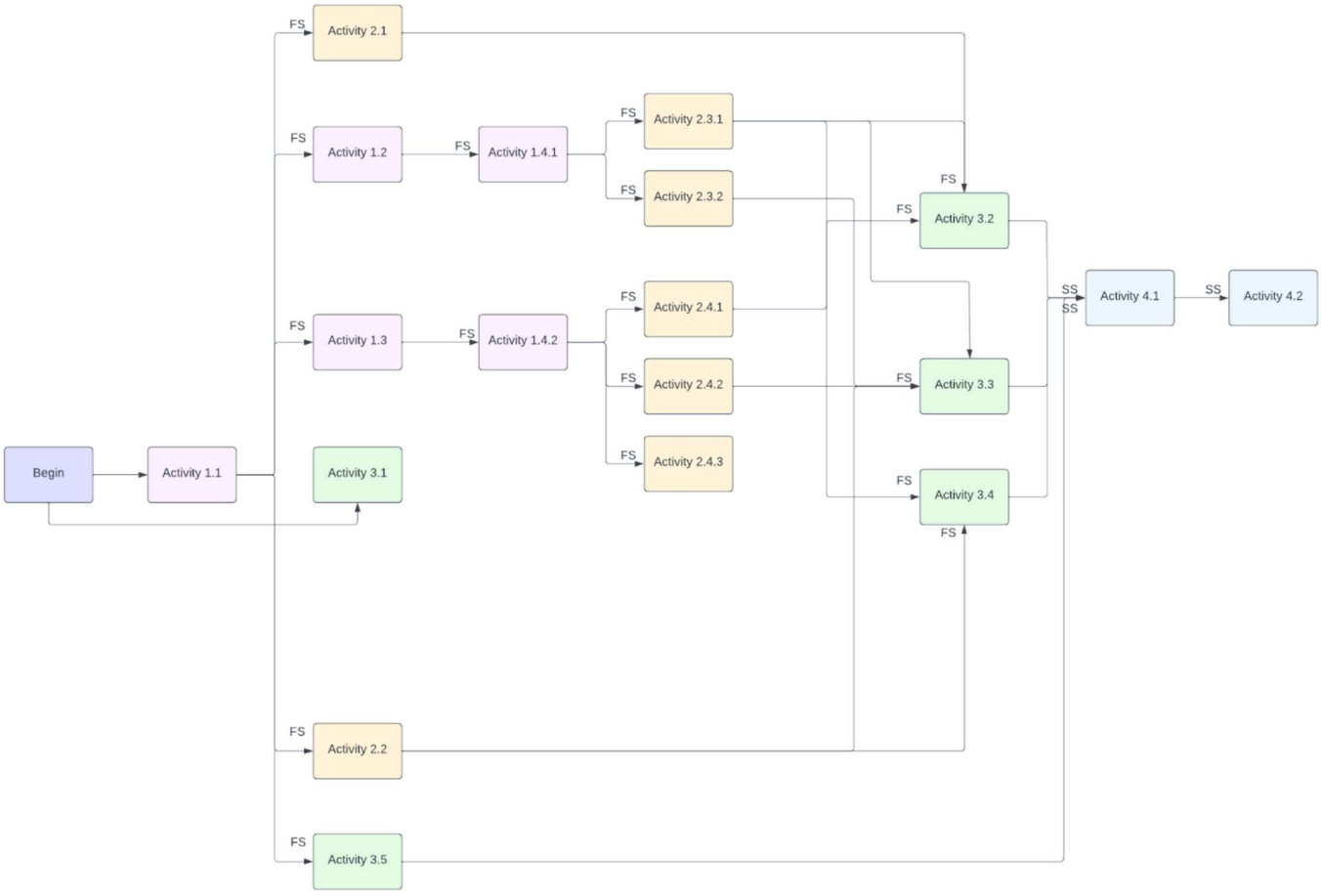


Figure 2: Schedule network diagram showing how each activity is connected

## 3.2 Gantt Chart


Figure 3: Gantt chart to show the approximate time required for each activity

# 4.0 Cost

## 4.1 Cost Description:

With 5 weeks of actual work, for the first 3 weeks, each member will work the 4 hours of the lab and 2 hours extra. For the last 2 weeks, each member will work the 4-hours of the lab and 4-hours extra. Each member is getting paid $50/hour.

## 4.2 Calculation of costs

Starting October 20th - November 24th, 2022

1. We had 6 weeks of working sessions (including reading week), for a group of three working members.
2. Weeks 1,3,4, each consisting of 18 hours, and weeks 5, and 6, each consisting of 24 hours.
3. Week 1,3,4: 6 hours/member x 3 members = 18 hours x 3 weeks = 54 hours for the group total.
4. Week 5, 6: 8 hours/member x 3 members = 24 hours x 2 weeks = 48 hours for the group total.
5. Weeks (1, 2, 3, 4, 5, 6) total group hours = 54 hours + 48 hours = 102 total hours for 3 members.
6. 102 total hours for 3 members = 102/3 = 34 total hours/member.
7. Each member's pay is $50/hour: Total Pay = 34 hours x $50 = $1700/member.
8. Total pay on the project: $1700 x 3 members = $5100.

## 4.3 Cost Baseline Table

Table 1: Approximate costs for our project

| Week | Total Hours For Group | Total Accumulative Cost |
|---|---|---|
| 1 (October 20) | 18 (lab-4hrs, extra-2hrs) | $900 |
| 2 (Reading Week) | 0 | $900 |
| 3 (November 3rd) | 18 (lab-4hrs, extra-2hrs) | $1800 |
| 4 (November 10th) | 18 (lab-4hrs, extra-2hrs) | $2700 |
| 5 (November 17th) | 24 (lab-4hrs, extra-4hrs) | $3900 |
| **6 (November 24th)** | **24 (lab-4hrs, extra-4hrs)** | **$5100** |

**Weekly Cost Value Formula =** (Hourly Rate) x (total # of hours per week for all members)

## 4.4 Cost Baseline Graph

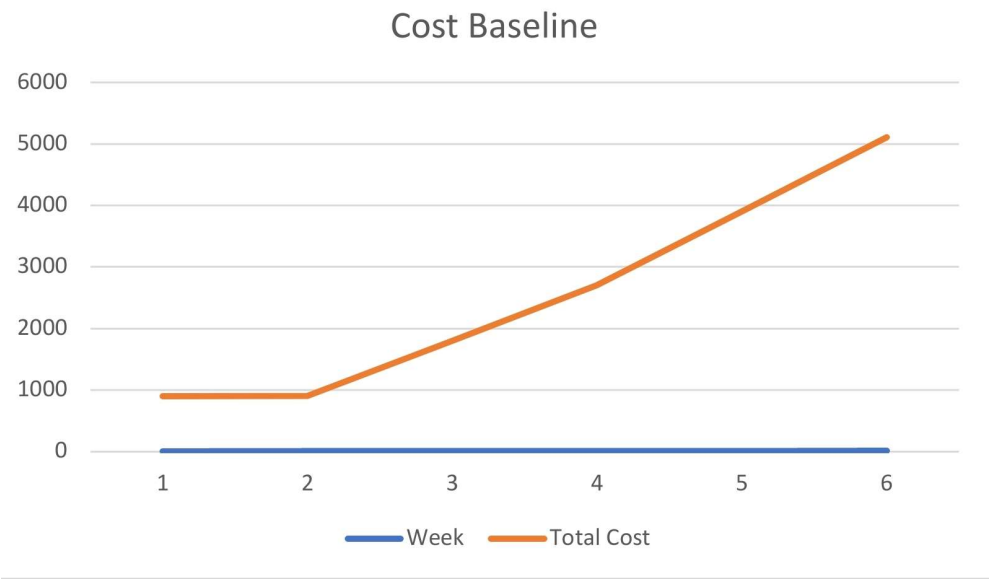


Figure 4: Costs organized into a line chart with time in weeks on the x-axis and cost in dollars on the y-axis

## 4.5 Planned Value Analysis Figure

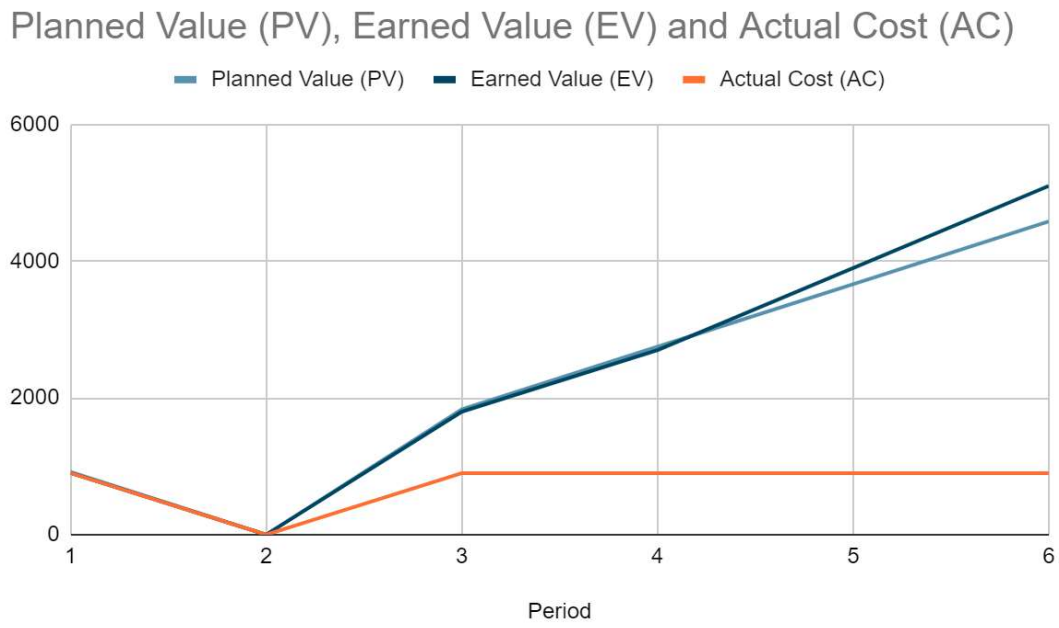


Figure 5: Planned Value Analysis Figure with time in weeks on the x-axis and cost in dollars on y-axis

Table 2: Planned Value Analysis Table

| Period | Planned Value (PV) | Earned Value (EV) | Actual Cost (AC) |
|---|---|---|---|
| 1 | 916 | 900 | 900 |
| 2 | 0 | 0 | 0 |
| 3 | 1832 | 1800 | 900 |
| 4 | 2748 | 2700 | 900 |
| 5 | 3664 | 3900 | 900 |
| 6 | 4580 | 5100 | 900 |

***Planned Value (PV) -*** The approved, time-phased budget allotted for carrying out the planned task

***PV = 5500***

***Earned Value (EV) -*** Sometimes referred to as Budgeted Cost of Work Performed (BCWP), is the portion of the assignment that is actually finished.

***PV = % Complete (Planned) x Task budget***

***PV = 75% x 5500 = 4125***

***Actual Cost (AC):*** It is sometimes referred to as Actual Cost of Work Performed (ACWP) and represents the true cost of the task, including every price required to perform it despite any difficulties. Included in it ought to be labour, supplies, tools, contractors, and other fixed costs.

***Total Actual Cost = 5100***

## 5.0 Responsibilities

With there being a variety of activities to complete, they must be distributed amongst the group members so that there is approximately 1 activity per group member per week. Below the assignment for these activities can be seen.

### 5.1 Human Resources:

Table 3: Responsibility assignment matrix

| Activity Number | Activity | Faris | Flynn | Laith |
|---|---|---|---|---|
| 1.1 | Set up the GitHub | **R** | **A** | |
| 1.2 | Testing the operation of the motors (Forwards and backward) | | **R** | **A** |
| 1.3 | Testing the operation of chosen sensors | **A** | | **R** |
| 1.4.1 | Mounting motor controller with motors | **R** | **A** | |
| 1.4.2 | Mounting sensors on car | | **R** | **A** |
| 2.1 | Default pathing of the final operation (Motion plan) | **A** | | **R** |
| 2.2 | Obstacle detection protocol | **R** | **A** | |
| 2.3.1 | Program turns (90 degrees right and left, 180 degrees, slant left and right) | | **R** | **A** |
| 2.3.2 | Program movement (Straight at a variety of speeds, hard stopping) | **A** | | **R** |
| 2.4.1 | Detect large (not golf ball) object | **R** | **A** | |
| 2.4.2 | Detect boundary Sensor | | **R** | **A** |
| 2.4.3 | Program gyroscope to create an internal map | **A** | | **R** |
| 3.1 | Progress Report | **R** | **R** | **R** |
| 3.2 | Implement Pathing | **R** | **A** | |
| 3.3 | Implement Obstacle Detection | | **R** | **A** |
| 3.4 | Implement Boundary Detection | **A** | | **R** |
| 3.5 | Building a plow | **R** | **A** | |
| 4.1 | Arena Test | **R** | **R** | **R** |
| 4.2 | Final Report | **R** | **R** | **R** |

## 6.0 Overall System Architecture

The overall architecture of the system can be broken down into several pieces: The sensors used, the movement of the robot and the motors. The Sensors poll information about the surroundings of the robot and send it to the Arduino. The Arduino then uses the information it receives from the sensors to decide what action the robot should perform. The behaviour of the motors will be dictated by the action/movement the Arduino deems appropriate. The overall architecture can be visualized in the simple diagram below.

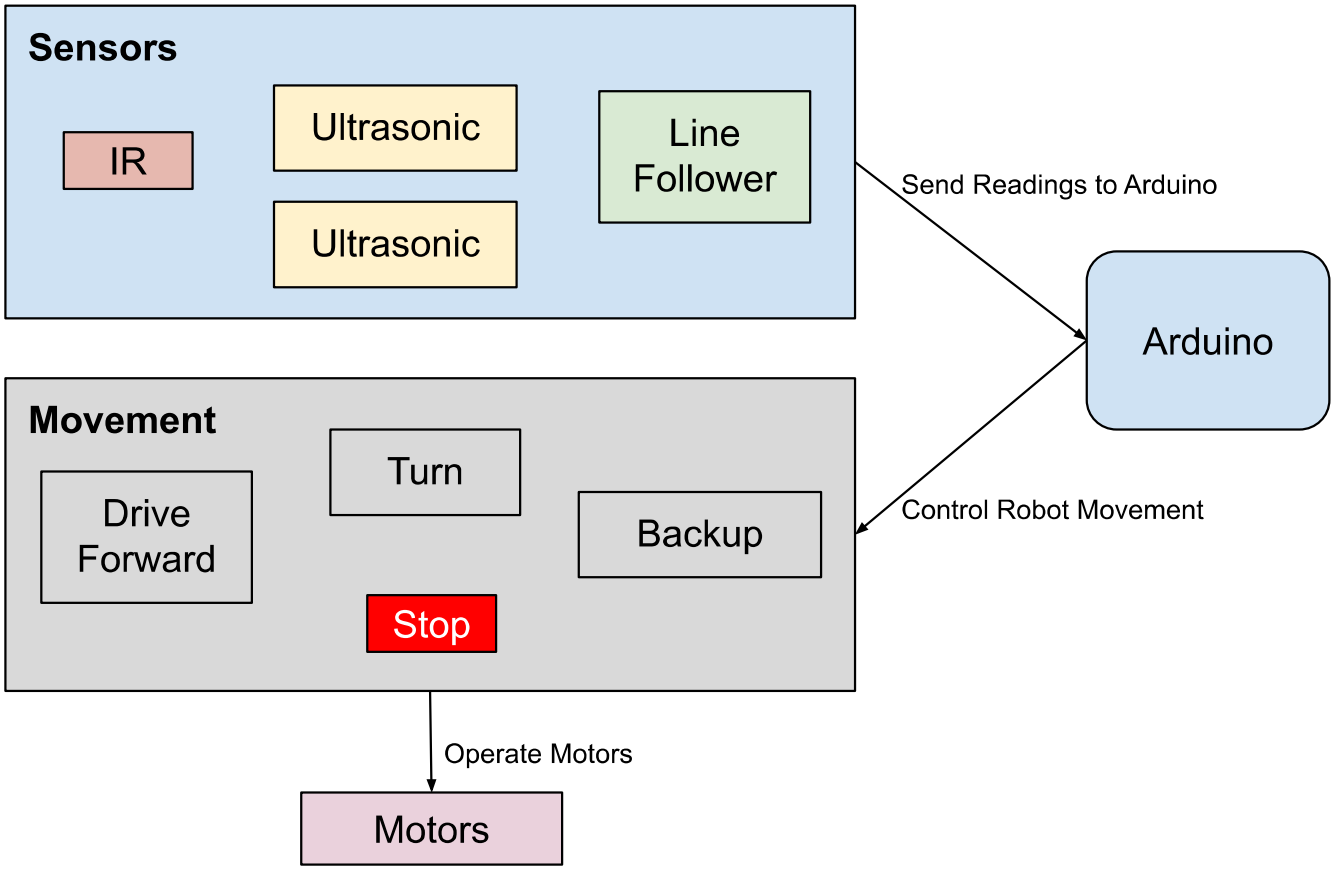


Figure 6: System Architecture **(Note: This was added based on feedback from the progress report)**

### 6.1 Sensors

The sensors we are employing in our design are two ultrasonics, mounted in the front left and front right, both angled slightly outward by approximately 30 degrees. Then we are using the IR sensor right in the middle front of the vehicle. Below the front of the car, we are also using one lane follower sensor. See the images below for reference:

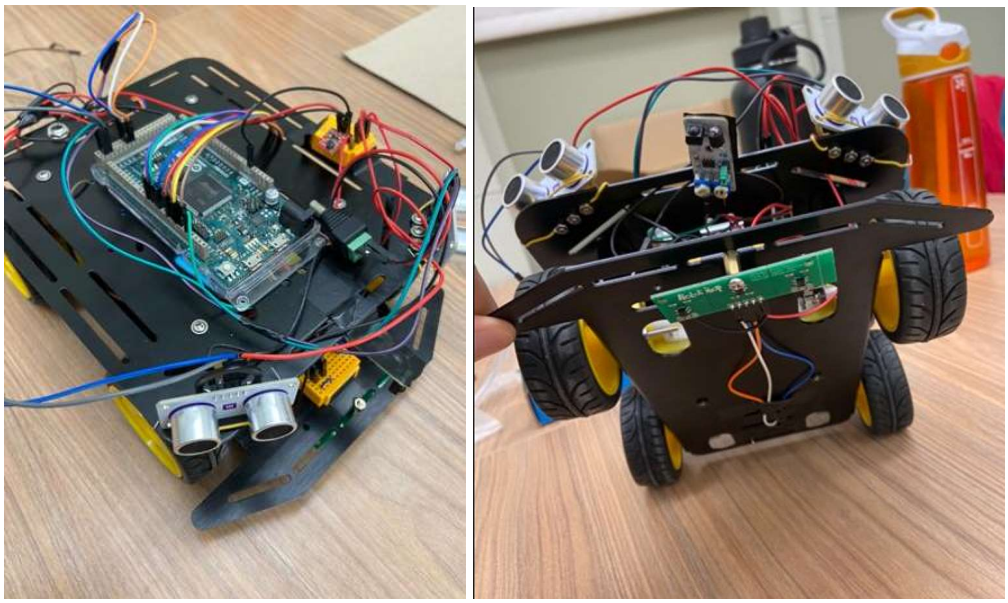

Figure 7: Images of our current robot implementation

### 6.2 Robot Movement

The movement of the robot has been defined by a variety of functions. The main movement operations we are employing are

- Driving Forward
    - 4 wheel drive with all motors going forward
- Driving Backward
    - 4 wheel drive with all motors going backward
- Stopping
    - All wheels shut off
- Turning (Left and Right)
    - Both wheels of one side driving forward, the other two driving backwards. This causes the robot to turn in place.

These operations are defined very simply. One important thing to note is the signals we are sending to our motors have a relatively low-duty cycle. This is because we believe that driving slowly allows our sensors and main loop more time to react and adjust to obstacles than if driving fast. Since the demo allows 5 minutes, we believe there is no rush, and thus are driving slowly.

### 6.3 Main Driving Algorithm

The main driving algorithm that will be implemented is as follows. The robot will drive forward until the sensors detect something close by. The way we do this for the ultrasonic sensors is by giving each sensor a threshold. The threshold values are still being tested but the two thresholds we have are labelled minDistance and stopDistance. The minDistance is the first threshold we pass, if our reading from an ultrasonic is less than this minimum distance, an object is close by. If we detect an object close, we will begin turning away from that object until the distance is greater than minDistance. However, if the sensor detects that an object is less than our defined stopping distance, we are too close to an object to properly turn. So in this case, we must drive backward a tiny bit before we begin turning. We will repeat this backward and then turn cycle until the ultrasonic distance value is no longer below the threshold.

The ultrasonics do most of the work in this regard, but we also employ a front IR sensor that aids in our detection. The IR sensor returns low if an object is within its programmed threshold. In our case, it is approximately 5 cm. If the IR sensor is low, it means that we have detected an object that is way too close. In this case, we employ the same measures as when an ultrasonic detects something within its stop distance. It will back up, and then make a decision to turn either left or right based on which ultrasonic reading is higher (meaning we have more room that way). See the state diagram below for a visual representation of this algorithm.

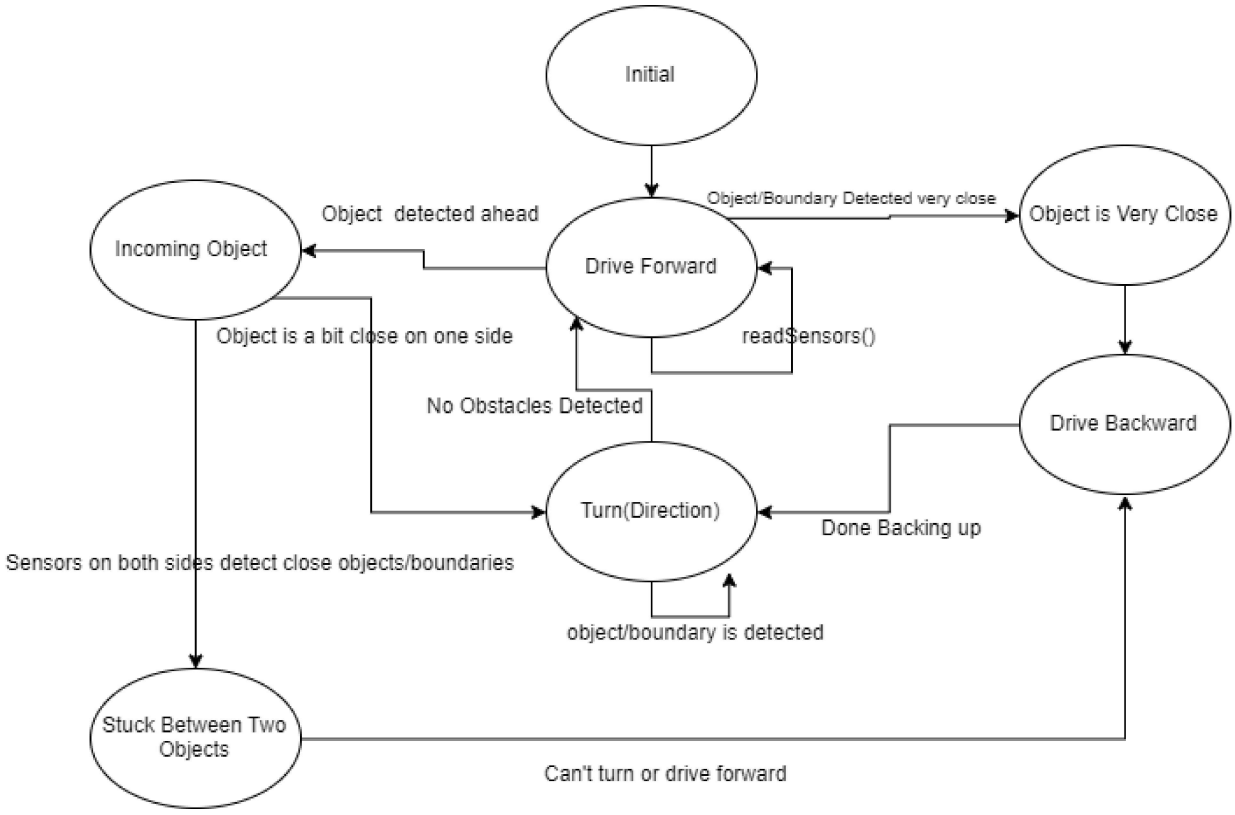


Figure 8: State Diagram **(Note: updated based on feedback from progress report)**

### 6.3.1 Watchdog Timer

This watchdog timer simply resets at the end of our main loop. It is in place if something executes infinitely for some reason, and we need to abort the system.

| | |
|---|---|
| Setup | 92 //Setting up WATCHDOG timer<br>93 delay(3000);<br>94 wdt_enable(WDTO_8S); |
| Implementation | 138 //Watchdog timer reset<br>139 wdt_reset();<br>140 //infinity loop to hang MCU<br>141 while(1){} |

Figure 9: Watchdog timer code snippet. **(Note: this was added to fix feedback from the progress report)**

The watchdog timer is designed to reset the Arduino board if we get stuck in an infinite loop for a specific time. After the test and trial, the team decided to set the time limit of the watchdog timer to 8 seconds. This was found to be the best value that gives the robot enough time to try to recover on its own, we found that if the robot was stuck for longer than 7 seconds, the robot is never able to recover, so it would be best to let the watchdog timer intervene and reset the robot.

### 6.3.3 Sequence Diagram

The robot starts in the main loop. In the loop, the robot normally drives forward while also calling a function called sesnorRead() that polls the values of all the sensors. Based on the readings of the sensors, the state of the robot changes. If an object along the path is detected but is not very close, the robot will go to the close object state and avoid the object by turning slightly. Once the object that was detected is not in range anymore, the robot goes back to driving forward in the loop. Another possible state is a very close object is detected. To deal with this situation, the robot executes a special avoidance function called hardAvoid(). Once it is done, and the object is avoided, the robot goes back to driving forward. The last situational state is the Boundary/line detected. This will initiate a function called avoidLine() which avoids a line by trying to turn or back up until the line is out of the way. Once this is done, the robot goes back to driving forward.

Below you can find the sequence diagram of how the whole system will interact:

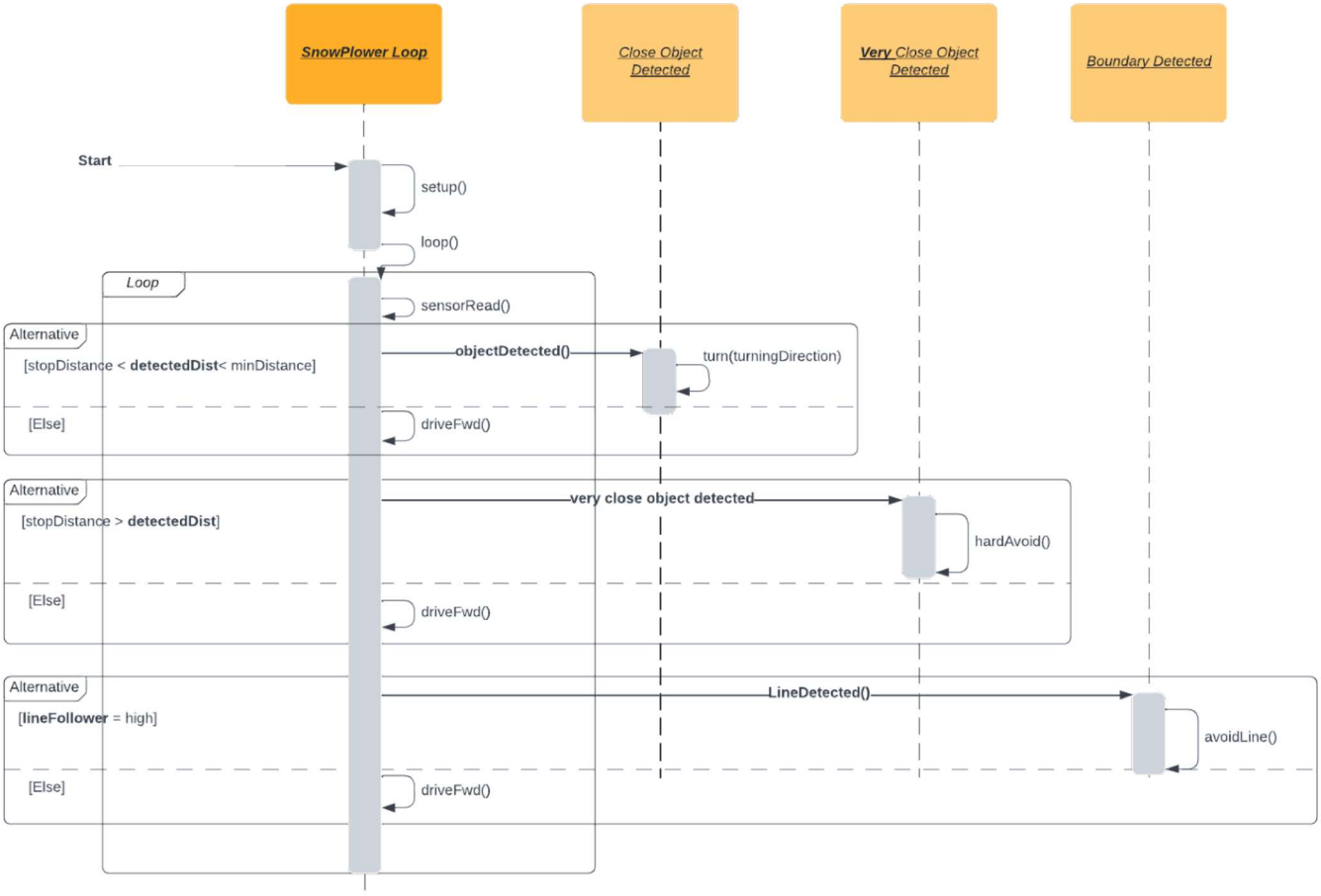


Figure 10: Sequence Diagram

# 7.0 Control Charts

## 7.1 Robot should not exceed a speed of 30 cm/sec.

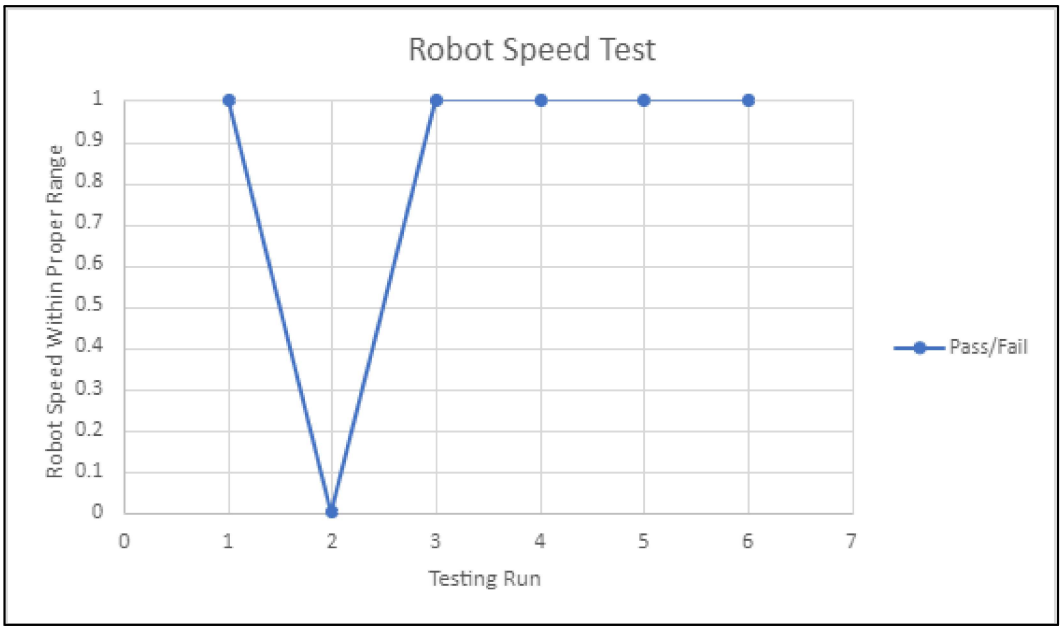


Figure 11: Testing the robot's speed. 1 represents speed under 30cm/sec. 0 represents speeds over 30cm/sec.

## 7.2 The robot should clear balls encountered along its path.

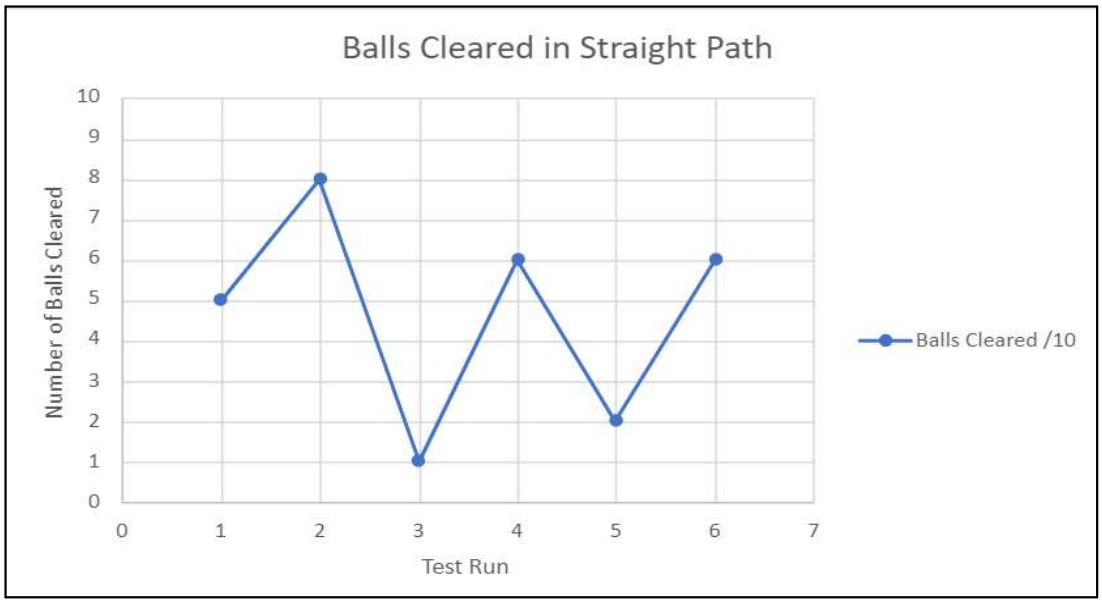


Figure 12: Test results for clearing snowballs along the robot's path. The graph shows how many out of the 10 balls placed along the path were removed.

### 7.3 Avoid Collision by Stopping When Detecting Obstacles.

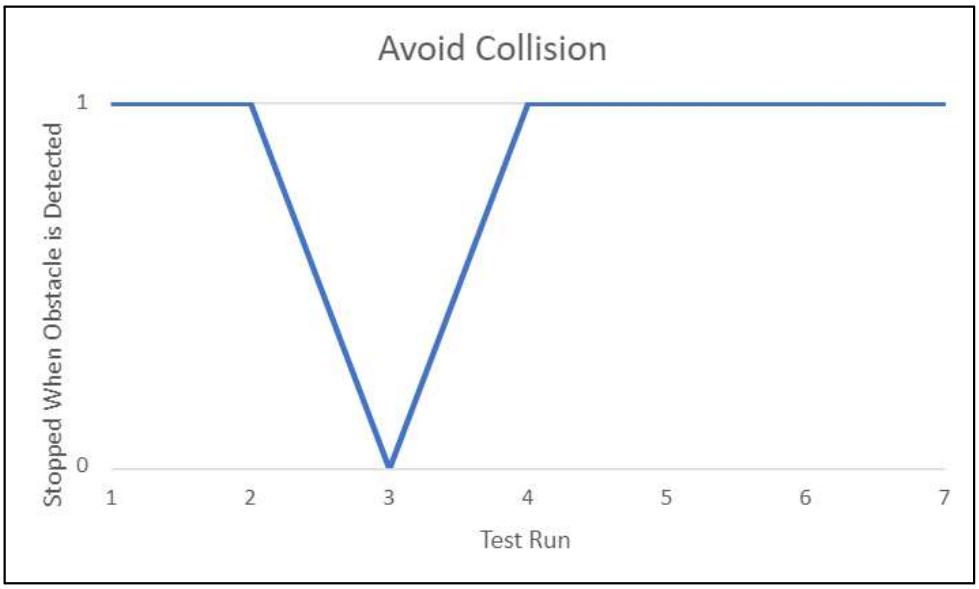


Figure 13: Test results for stopping when an obstacle is detected. 1 represents successful stops. 0 represents failing to stop (collision)

### 7.4 Change Direction When Black Line is Detected

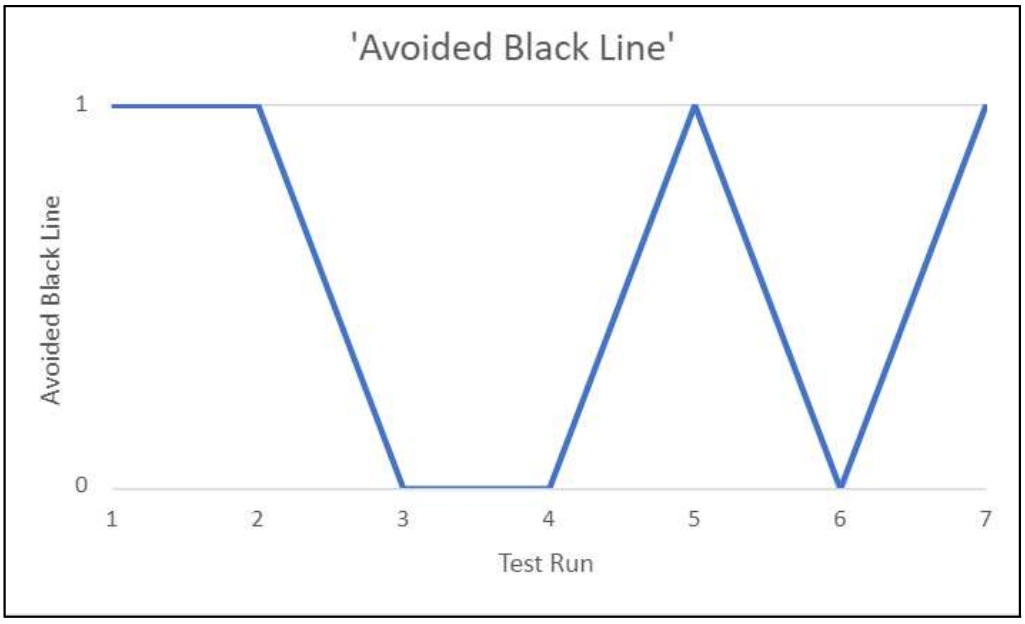


Figure 14: Test results for avoiding lines. 1 represents successful line avoidance. 0 represents failing to avoid a black line.

### 7.5 Reset When Stuck in a Loop

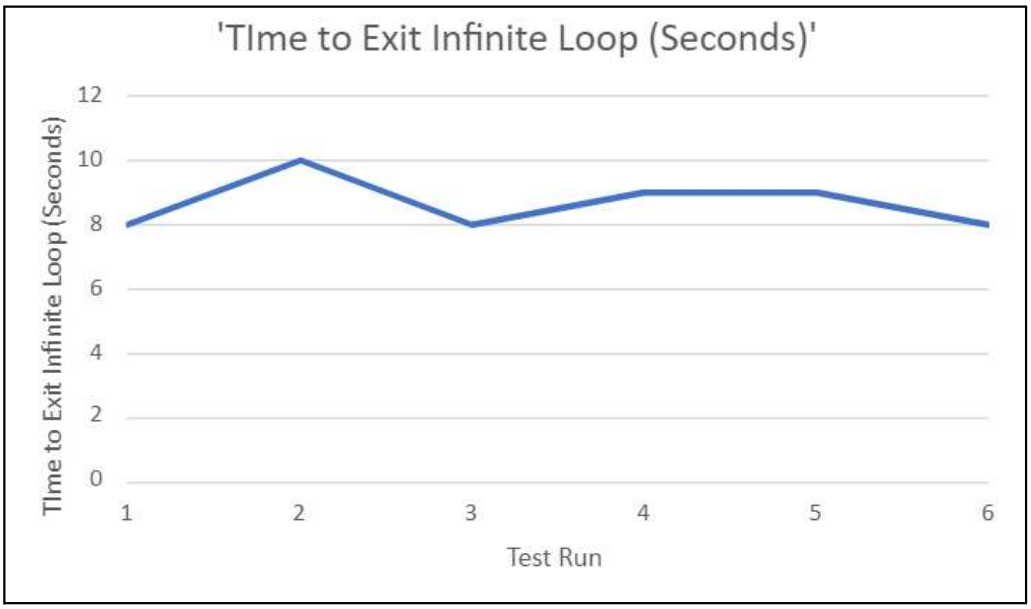


Figure 15: Test results for exiting an infinite loop situation. This graph shows the time it takes for the robot to reset when an infinite loop is detected.

### 7.6 Get Unstuck from When in Between Two Obstacles

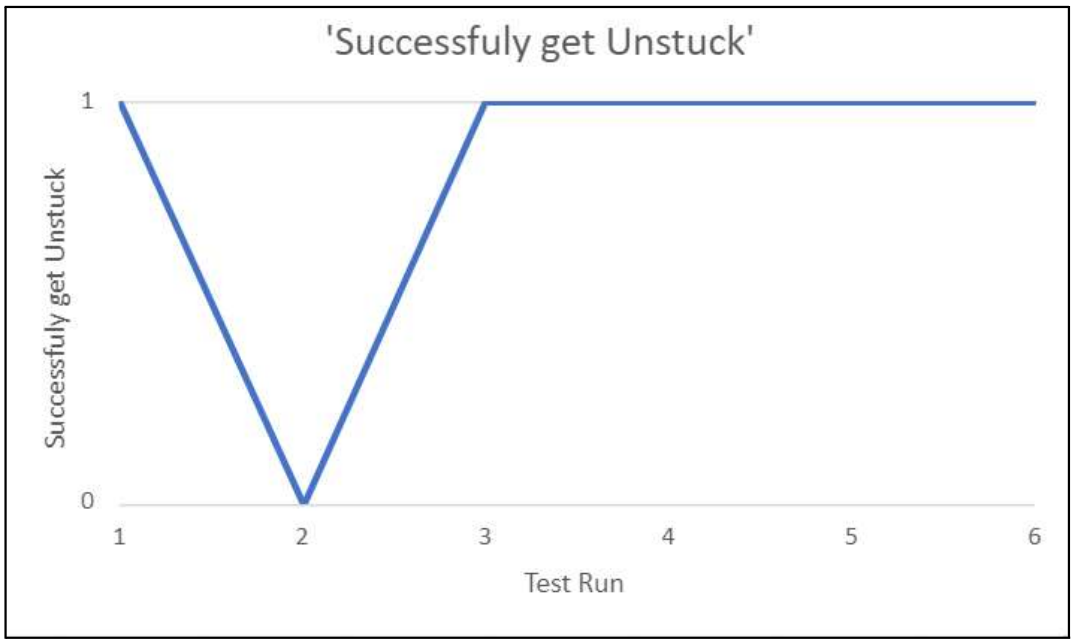


Figure 16: Test results for the robot's ability to get itself unstuck when in between 2 obstacles. 1 represents a successful attempt to get unstuck. 0 represents failed attempts.

### 7.7 Voltage Supply to Sensors

Table 4: Test results for voltage outputted to sensors. The ideal voltage desired is 3.3 V.

| Test Run | Voltage Inputted to Sensors Reading (V) |
|---|---|
| 1 | 3.3 |
| 2 | 3.27 |
| 3 | 3.24 |
| 4 | 3.27 |
| 5 | 3.25 |
| 6 | 3.27 |

## 8.0 Arena System Testing

### 8.1 Individual Testing

Table 5: Arena test results for lab 11.

| Trial | Balls Cleared From Arena | Obstacles Hit | Crossed the Boundary | Duration (Minutes) |
|---|---|---|---|---|
| 1 | 12 | 3 | 2 | 05:21 |
| 2 | 9 | 2 | 4 | 05:01 |
| 3 | 15 | 0 | 4 | 05:00 |
| 4 | 32 | 1 | 0 | 05:00 |
| 5 | 28 | 0 | 0 | 05:00 |
| 6 | 38 | 1 | 0 | 05:05 |
| 7 | 31 | 0 | 0 | 05:00 |

### 8.2 Final Demo

Table 6: Arena test results for lab 12.

| Trial | Balls Cleared From Arena | Obstacles Hit | Crossed the Boundary | Duration (Minutes) |
|---|---|---|---|---|
| 1 | 13 | 1 | 2 | 05:00 |
| 2 | 19 | 1 | 1 | 05:00 |

## 9.0 Github

Below is the link to our project repo:

https://github.com/LaithRazeq/SYSC4805