SYSC 4805
Robot Task Project - Snow Plough

Team Laser Lemon

Final Report

April 11th, 2022

Group#: 3

Members:

Timothy Knowles
101097700

Denise Mayo
101044064

Emma Boulay
101073617

Chhavi Sujeebun
101126487

1 Project Charter

1.1 Overall Objective

The overall objective is to design an autonomous snow plough robot using *CoppeliaSim* that will clear the snow off an area enclosed by a closed path while avoiding fixed and moving obstacles.

1.2 Overall Deliverables

The core deliverable of the project is the *CoppeliaSim* model file which implements the final design for the snowplough robot. The model file contains the physical design of the plough as well as the scripts which control the robot's behaviour. The scripts implement the logic and algorithms necessary to react to sensory input and direct the robot's path to clear as much of the snow as possible within the 5-minute simulation window. Accompanying this model file, other deliverables focus on documentation and demonstration. For documentation a progress report was completed midway into project development. Also a final-report-documentation deliverable which summarised all work completed, each member's contributions and described the end product. Concerning the demonstration, there will be a demo deliverable showing the robot's clearing efficacy on the test maps, and a final project presentation to summarise all the work done while detailing the ultimate result.

2 Scope

2.1 Requirements

The project must adhere to a number of requirements and constraints. The following list serves as a written agreement between the group members on the specifications that the final deliverable will meet. The requirements are as follows:

Table 1: Requirements table for Snow Plough Robot

	Requirement
1	The robot shall be no larger than 0.5 x 0.8 x 1 metres at the start of the simulation, including the custom plough component.
2	The robot shall be no larger than 1 x 0.8 x 1 metres at any time during the simulation, including the custom plough component.
3	The robot shall have a plough component to push snow spheres outside of the highlighted perimeter.
4	The robot shall be able to differentiate between obstacles and snow.
5	The starting position of the robot shall be (x, y, z) = (0m, -6.25m, 0m).
6	The robot shall not exceed a maximum speed of 2 m/s.
7	The behaviour of the robot shall be written in Python, with support functions written in Lua as needed to integrate unsupported functionality.
8	For each sensor the robot possesses, a real-world equivalent sensor shall be documented.
9	The robot simulation time shall not exceed 5 minutes.
10	The robot shall avoid falling off the edges of the test map.
11	The robot shall be able to detect the black line of the perimeter

2.2 Work Breakdown Structure (WBS)

2.2.1 WBS Diagram

A work breakdown structure (WBS) is pictured below in Figure 1. The diagram displays a hierarchical decomposition of the entirety of the work to be completed for the project. The work has been divided into four parts; Project Management, Modelling and Design, Testing and Evaluation, and Deliverables.

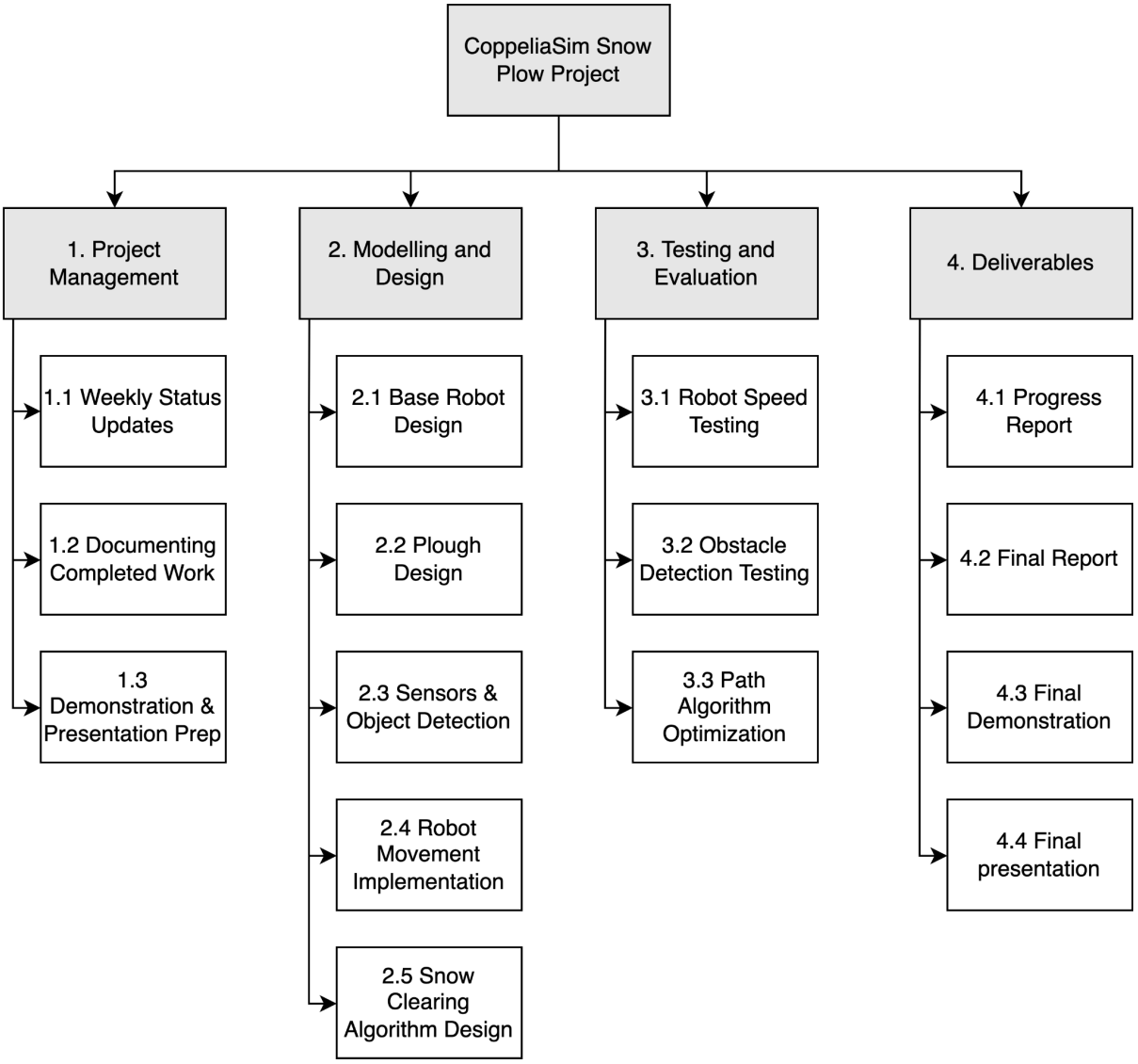

Figure 1: Work Breakdown Structure for the Snow Plough Robot Project

2.2.2 WBS Dictionary

Accompanying Figure 1 WBS, pictured below is the WBS dictionary. The dictionary (shown in Table 1) provides specific details for each terminal element, including descriptions, constraints and quality requirements.

Table 2: WBS Dictionary for the Snow Plough Robot Project

Terminal Element #	Work Description	Constraints	Quality Requirements
1.1 - Weekly Status Updates	- Consists of updating communication through Discord & follow-ups with the T.A. in weekly lab meetings	- Each member must contribute	- Team members are expected to maintain consistent communication
1.2 - Documenting Completed Work	-Document any tasks, research, algorithm completed	-Team members must follow the gantt chart and schedule network diagram to complete tasks by required deadlines	- Ideally work is documented by individual that completed it
1.3 - Demonstration & Presentation Prep	- Meetings to go through and prepare presentations and the demonstration	- Must be completed reasonably before the presentation due dates	- Clear and effectively describe work accomplished
2.1 - Base Robot Design	-Identify and implement the components required for the base robot design	- Robot cannot be larger than 1x0.8x1 metres (includes plough).	-Robot must be able to move -Robot components must remain attached to the robot body when the robot moves
2.2 - Plough Design	-Identify and Implement the components required for the plough design	- Robot cannot be larger than 1x0.8x1 metres (includes plough).	- Plough must be designed for optimal snow moving
2.3 - Sensors & Object Detection	-Identify and Implement appropriate sensors so that the snow plough robot can follow a closed path and avoid obstacles -Proximity and vision sensors are to be used	- Must be able to differentiate between obstacles and snow - Must have real world equivalent sensor - Scripts will be written in python	-Sensors must be able to detect paths, moving and fixed obstacles
2.4 - Robot Movement Implementation	- Involves having the robot properly move through the simulation environment	- Robot cannot fall off the edge of the test map - Robot cannot exceed 2 m/s	-Robot must follow a closed path
2.5 - Snow Clearing Algorithm Design	-How the robot uses the plough to move the snow out of the arena	- Robot cannot collide with obstacles	
3.1 - Robot Speed Testing		- Simulation time cannot exceed 5 minutes	
3.2 - Obstacle Detection Testing	-Test to ensure that the robot does not hit both fixed and moving obstacles	-Obstacle avoidance algorithm must be correctly implemented	-Obstacle avoidance algorithm must be designed to avoid both fixed and moving obstacles
3.3 - Path Algorithm Optimization	-Test to ensure that the robot follows a closed path	-Path finding algorithm must be correctly implemented	-Robot must be able to follow any closed path
4.1 - Progress Report	-Document the progress made in the project	- Due end of lab 6	-Team members follow the Gantt chart and complete the required tasks
4.2 - Final Report	-Document the progress made, the tests performed, the training maps tested and the challenges faced.	- Due end of lab 12	Team members follow the gantt chart and complete the required tasks, tests and training
4.3 - Final Demonstration	-Simulate and demonstrate our snow plough robot on training maps provided	- Due during last two labs	-Robot must be able to remove most amount of snow in the map while

			avoiding obstacles
4.4 - Final Presentation	Present our project	- Due during last 4 lectures	-Robot must be working and satisfying all requirements

2.3 Testing

The system shall be tested rigorously to ensure that all agreed upon requirements are met as documented in Section 2.1. The tests conducted and their results are documented below:

Table 3: Unit Testing

Test ID	Test Description	Success Criteria	Results
T.U.1	Test that the robot can detect non-moving obstacles	The robot can detect non moving obstacles ranging from 10 to 300 cm.	Pass
T.U2	Test that the robot can detect moving obstacles	The robot can detect moving obstacles ranging from 10 to 300 cm and at a speed up to 20 m/s.	Pass
T.U.3	Test the robot's obstacle avoidance algorithm	The robot can avoid collisions with obstacles.	Pass
T.U.4	Test that the robot can detect the path	The robot can detect a black line.	Pass
T.U.5	Test that the robot can move snow outside of the detected path	No snow is left inside of the designated area.	Pass

Table 4: Performance Testing

Test ID	Test Description	Success Criteria	
T.P.1	Record how the robot performs on training map 1	All snow is cleared from the designated area, without colliding with obstacles, in a maximum of 5 minutes.	Fail. ~40% snow removed
T.P.2	Record how the robot performs on training map 2	All snow is cleared from the designated area, without colliding with obstacles, in a maximum of 5 minutes.	Fail. ~40% snow removed
T.P.3	Record how the robot performs on training map 3	All snow is cleared from the designated area, without colliding with obstacles, in a maximum of 5 minutes.	Fail. ~40% snow removed

Table 5: Stress Testing

Test ID	Test Description	Success Criteria	
T.S.1	Test at what speed the robot can no longer reliably detect a moving obstacle	The robot can detect an object moving at a speed of 5 m/s	Pass

Table 6: Integration Testing

Test ID	Test Description	Success Criteria	
T.I.1	Test that the robot movement control and plough control modules work together	The plough does not impede the robot's vision or proximity sensors at any stage.	Pass

The unit tests were performed by measuring the robot module's performance on *CoppeliaSim* scenes generated by a team member. If the robot can successfully pass the tests outlined above, then it behaves as expected and should successfully remove the snow from the testing maps in the time limit.

3 Schedule

3.1 Schedule Network Diagram

The schedule network diagram for the *CoppeliaSim* Snow Plough project is depicted in Figure 2 below. The diagram is a graphical representation of the logical relationships and dependencies among the project schedule activities that are proposed in Section 4.1. The schedule network diagram uses the critical path method to estimate the minimum project duration and the schedule flexibility. In the diagram below each activity node represents an activity in the Responsibility Matrix. At the top of each activity node the early start, duration, and early finish dates are provided. At the bottom of each activity node, the late start, total float, and late finish dates are provided.

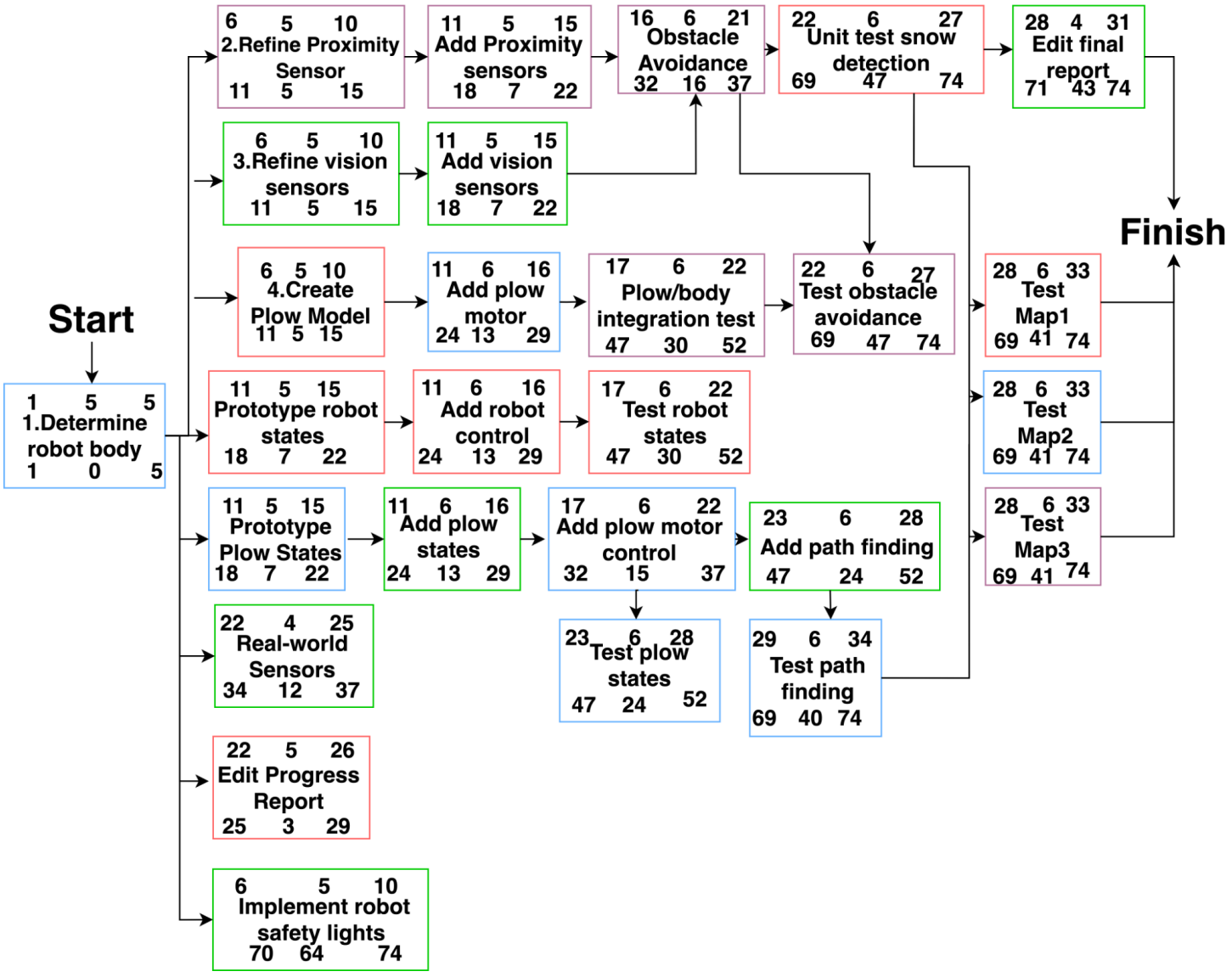

Figure 2: Schedule Network Diagram for the Snow Plough Robot Project

3.2 Gantt Chart

The Gantt chart for the project is provided in Figure 3 below. Each team member has one task assigned each week. A colour-coded legend is provided to show which team member is assigned which task. The unit tests for the robot and plough controls require their implementation to be finished before testing can start. A contingency plan is made to allow for an extra week of slack time for implementing the robot and plough control in case any unforeseen problems arise.

Gantt Chart

Activity Identifier	Activity Description	Days	Lab 4	Lab 5	Lab 6	Lab 7	Lab 8	Lab 9	Lab 10
1.1	Determine robot body	7							
1.2	Refine requirements for proximity sensor	7							
1.3	Refine requirements for vision sensors	7							
1.4	Model plough designs and choose appropriate shape	7							
1.5	Implement proximity sensors	7							
1.6	Implement vision sensors	7							
1.7	Define and prototype robot states	7							
1.8	Define and prototype plough states	7							
1.9	Compile list of unsupported remoteAPI functions in Pvthon	7							
1.10	Implement robot control API	7							
1.11	Implement plough states	7							
1.12	Implement motor for plough	7							
1.13	Format and proof-read progress report	7							
1.14	Implement plough motor API	7							
1.15	Implement obstacle avoidance algorithm	7							
1.16	Document real-world sensors	7							
1.17	Implement path finding algorithm	7							
1.18	Unit test plough states	7							
1.19	Unit test robot states	7							
1.20	Perform integration tests on body and plough	7							
1.21	Unit test path finding algorithm	7							
1.22	Unit test obstacle avoidance algorithm	7							
1.23	Implement robot safety lights	7							
1.24	Unit test snow detection algorithm	7							
1.25	Format and proof-read final report	7							
1.26	Test design on training map 1	7							
1.27	Test design on training map 2	7							
1.28	Test design on training map 3	7							

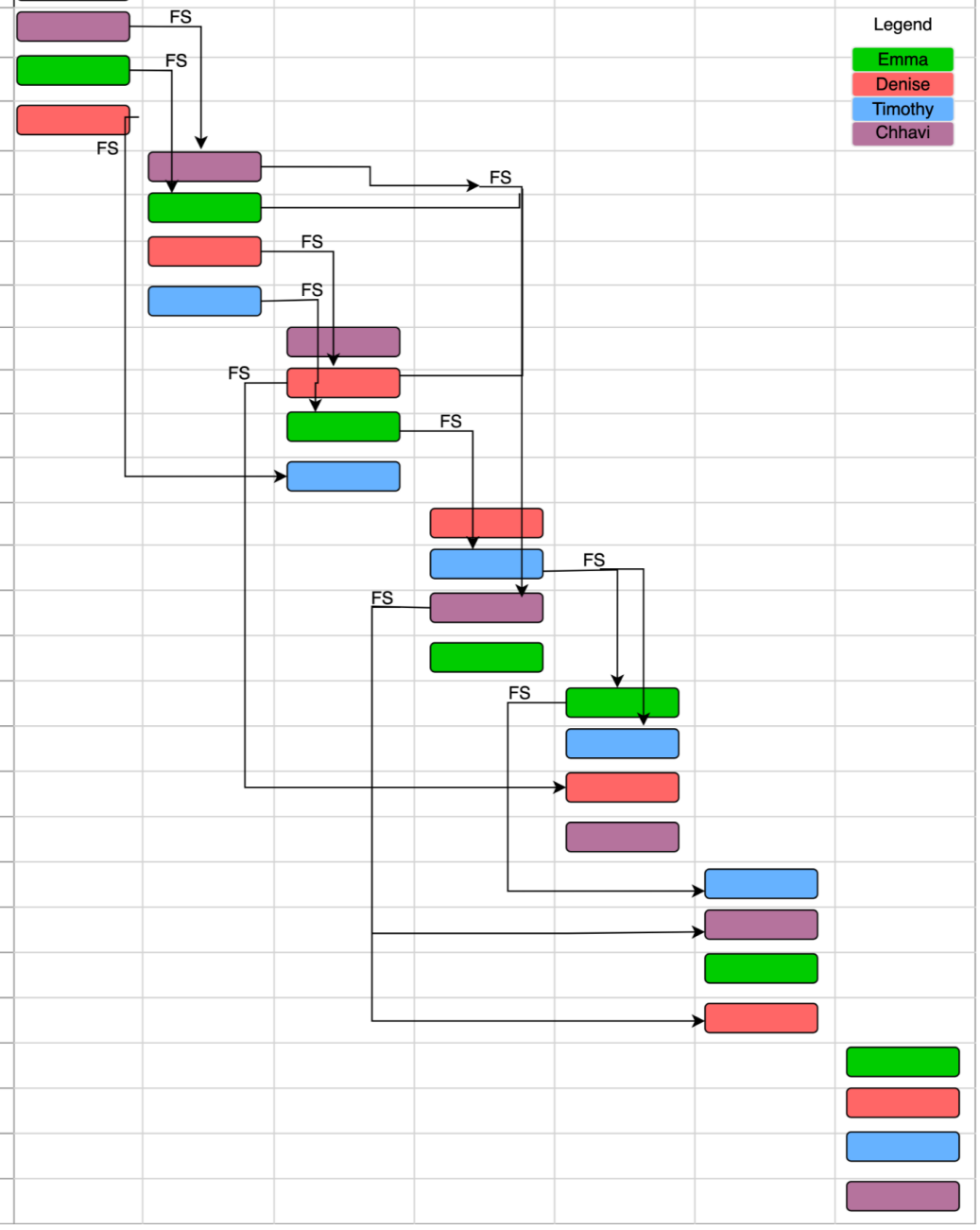

Figure 3: Gantt Chart for the Snow Plough Robot Project

4 Human Resources

4.1 Responsibility Assignment Matrix

Each member in the team was assigned 7 unique tasks that they are responsible for completing and 7 unique tasks they are responsible for reviewing. Figure 4 below shows the responsibility breakdown for the project.

Activity	Denise	Timothy	Emma	Chhavi
Determine robot body style and shape		R	QA	
Refine and document requirements for proximity sensors		QA		R
Refine and document requirements for vision sensors	QA		R	
Define and prototype robot states	R			QA
Document real-world sensor equivalents	QA		R	
Define and prototype plough states		R	QA	
Compile list of unsupported functions in Python API		QA		R
Implement proximity sensor(s)		QA		R
Implement vision sensor(s)	QA		R	
Model plough designs and choose appropriate shape	R			QA
Implement plough control motors		R	QA	
Implement safety lights	QA		R	
Implement plough states	QA		R	
Implement robot states and control API	R			QA
Implement plough control API		R	QA	
Implement general steering and movement control motor functions	QA		R	
Implement obstacle avoidance movement control functions		QA		R
Unit test robot states and control API	R			QA
Unit test plough states		R	QA	
Unit test obstacle avoidance movement control functions		QA		R
Unit test plough control API	R			QA
Unit test movement control (general steering) functions		R	QA	
Perform integration tests on plough and body		QA		R
Test design on training map 1	R			QA
Test design on training map 2		R	QA	
Test design on training map 3		QA		R
Format and proof-read progress report	R			QA
Format and proof-read final report	QA		R	

Legend	
Responsible for completing work	R
Reviewer (quality assurance check)	QA

Figure 4: Responsibility Assignment Matrix for the Snow Plough Robot Project

5 Overall Architecture

5.1 Architecture Diagram & Description

The figure below describes the overall system architecture of the design solution for the Coppeliasim Snow Plough Robot. The robot is primarily controlled by the path-finding program. The path finding program uses the Movement Control Module to control the robot wheels to move the robot around the map. The Object Detection module will inform the robot when an object is detected and the path finding algorithm can plan the most efficient route to avoid the obstacle. The Line Detection module will inform the robot when it has reached the perimeter and the path finding algorithm knows to change direction. The main path finding program uses the Plough module to open and close the plough.

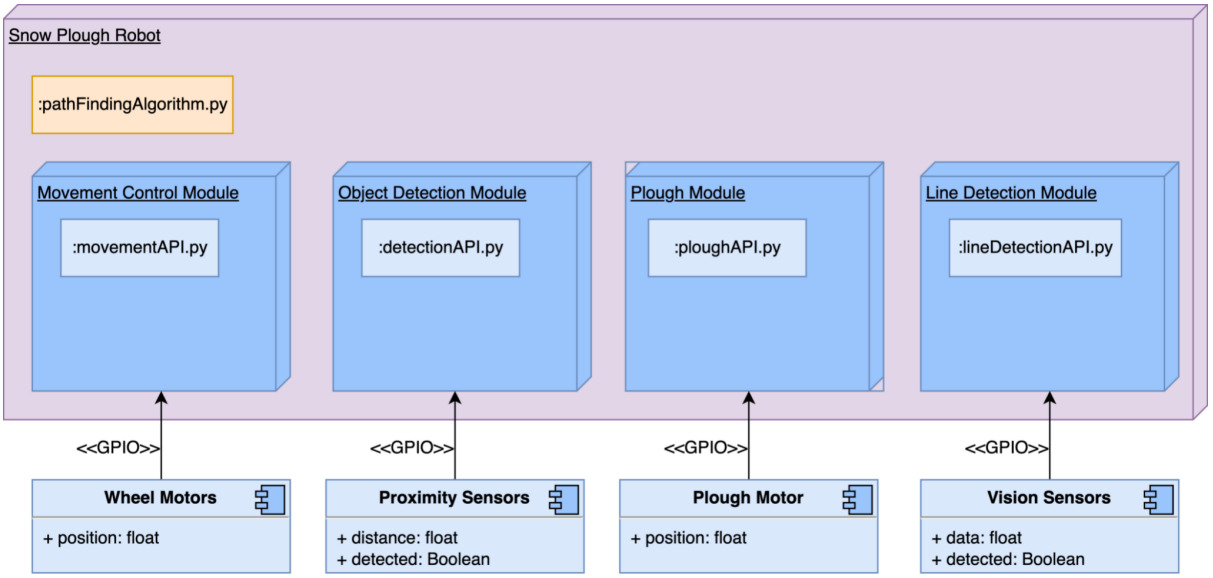

Figure 5: Overall System Architecture of the Snow Plough Robot

5.1.1 Model Design

The primary design for the snow plough robot is documented in the figures below. The initial robot has vision sensors at the front base of the robot to detect the outer perimeter and proximity sensors at the top of the robot to detect obstacles.

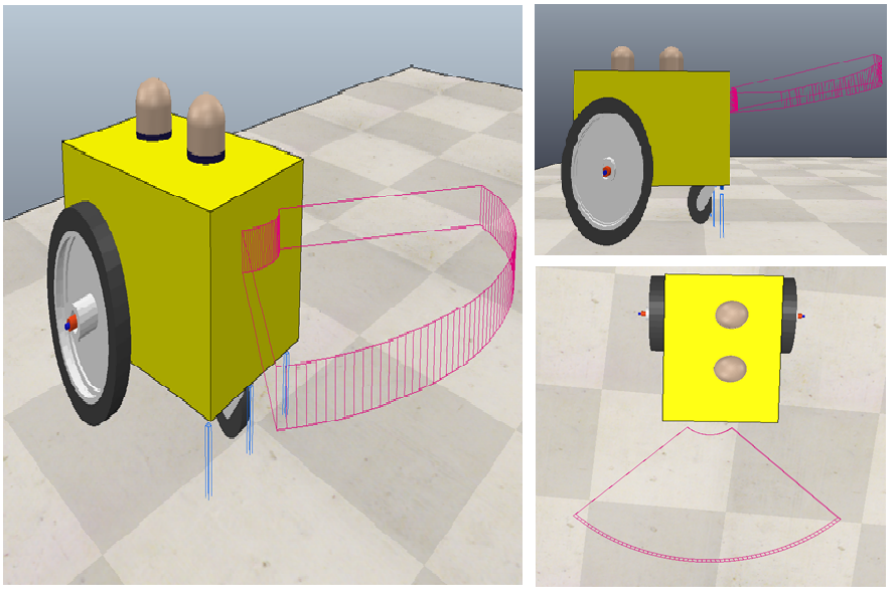

Figure 6: Initial robot body model with two rear wheels (motorised), one front castor wheel, and a front proximity sensor

5.1.2 Plough Designs

The team's initial plough designs are documented in the figures below. The team evaluated having a plough with either a single or dual hinge door system. The team's final plough design is shown in Figure 9.

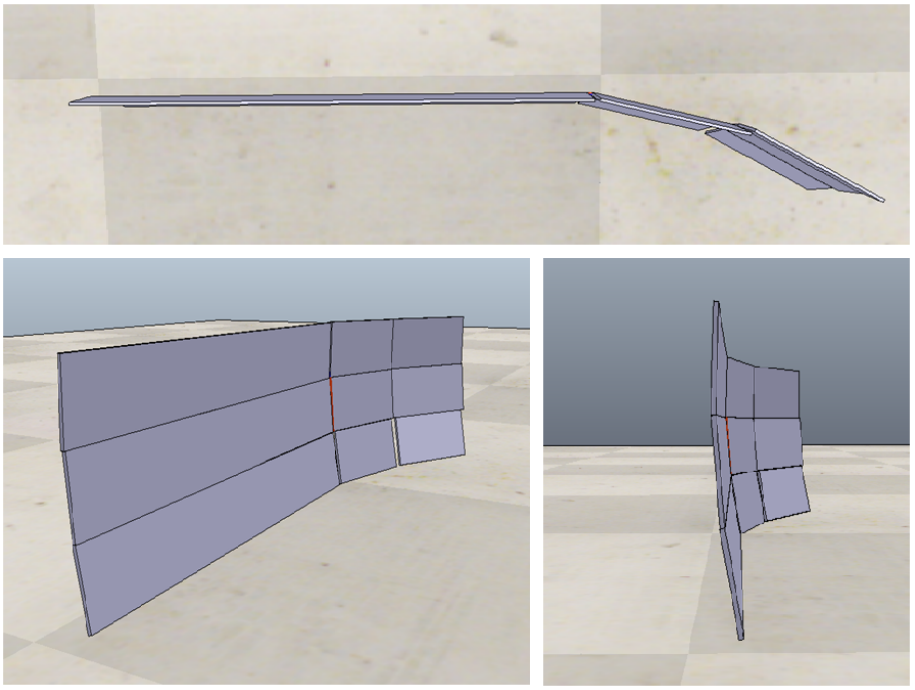

Figure 7: First draft of plough model, intended to represent the final plough design

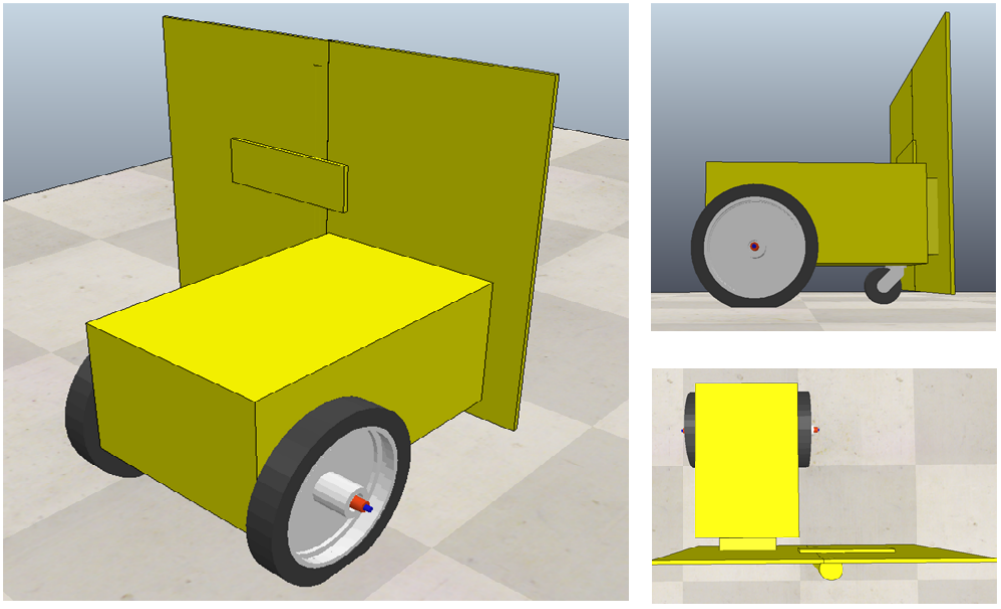

Figure 8: Robot plough design used for initial testing of folding mechanism

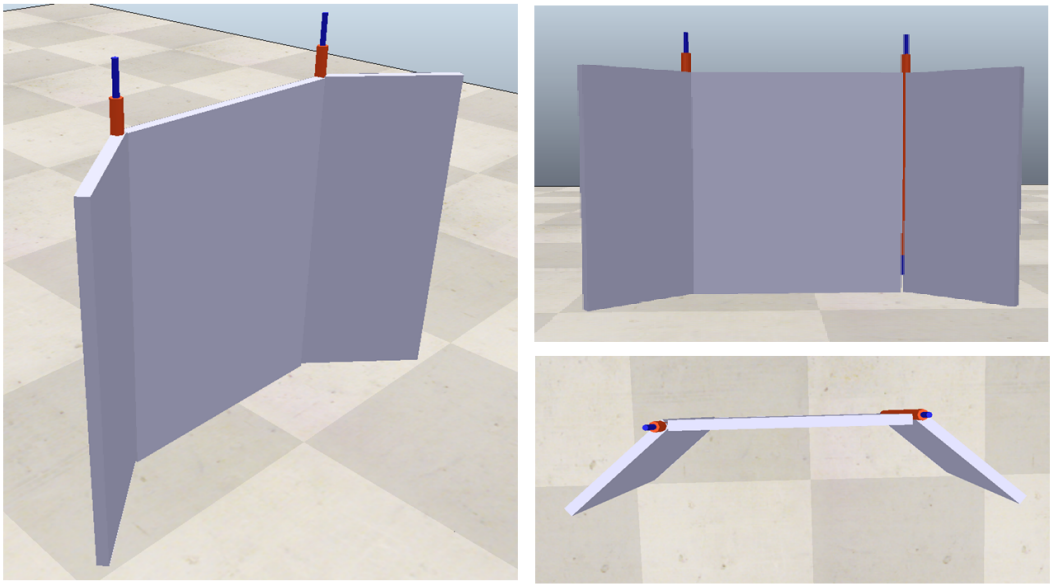

Figure 9: Alternate plough design used for initial testing and comparison against Figure 8 plough

5.1.3 Final Robot Design

The final robot design is provided in Figure Y. The team decided to change the robot body from the initial box design to the DR12 robot model provided in CoppeliaSim. During development the team ran into some issues with odd behaviour when running the simulation from the physics engine. The DR12 model dynamics are optimised to run in CoppelisSim and provide smoother robot wheel articulation.

There are two safety lights on the top of the robot body that will flash amber every 0.7 seconds. This is an important safety feature as it alerts the people in the vicinity of the robot's presence.

There are 3 vision sensors attached behind the plough at the front of the robot. These sensors alert the robot if it has crossed the black line.

There are 8 proximity sensors at the front half of the robot used to navigate around obstacles using the Braitenberg algorithm. There is a rear proximity sensor that is deployed when the robot is reversing. There is also a wide and longer range proximity sensor that covers the outside of the plough, this is used to detect objects on the left and right of the plough when the robot is turning. Finally, there is a shorter range proximity sensor that covers the width of the plough that acts as a fail-safe. If any object is detected in this region, the robot will launch an emergency stop, reverse 0.3 m, turn 45 degrees to the left or right, and then continue driving straight.

A motorised plough is attached to the front of the robot. The plough has 2 panels that are motor controlled. The plough panels can open to 135 degrees, this allows the plough to stay within the size constraints defined in the requirements.

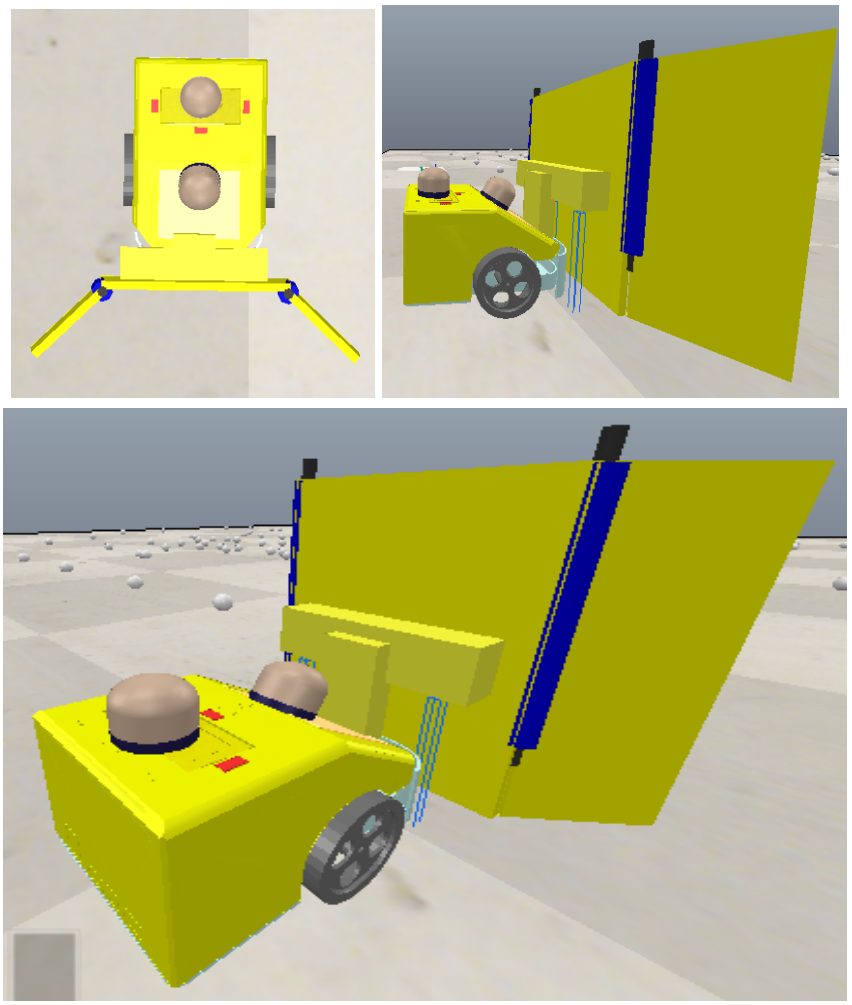

Figure 10: Final robot design

5.2 Time Triggered System

The main functions of the system are time triggered. The vision and proximity sensors will be polled in a loop to acquire the necessary information to perform the associated manoeuvre actions. A time triggered approach was chosen over an event triggered approach due to a high degree of predictability and simple implementation. However, an event triggered design consumes less power, as the micro-controller unit has more processor availability for other tasks until an event triggers an interrupt.

6 System States

The state machine governing the robot's movement and plough position is detailed below. The two state machines will interact using an additional state in the Robot Body State Chart (Figure 11).

6.1 Robot Body and Movement States Chart

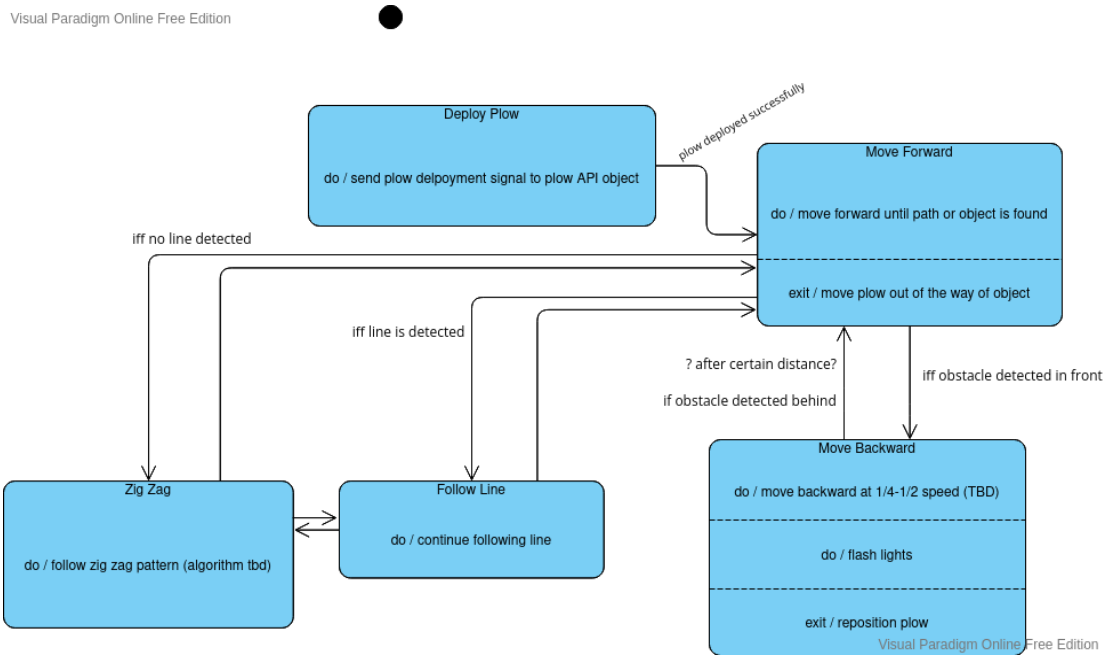

Figure 11: Robot movement state chart detailing the initial deployment of the plough, movement patterns and input transitions for obstacle avoidance

6.2 Plough States

6.2.1 Plough States Prototype

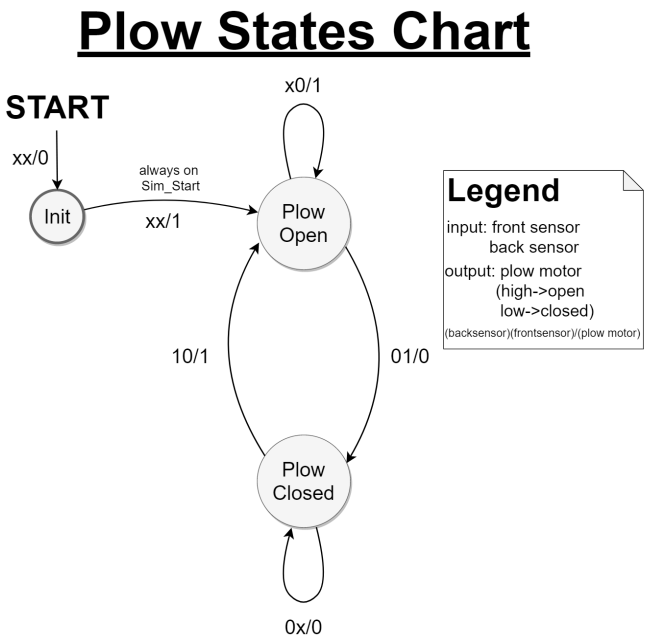

Figure 12: Plough States Chart

Figure 12 shows the operation of the plough itself. Due to the folding motion of the plough, it was determined that while avoiding an obstacle, it would be both possible and worthwhile to fold the plough section into the closed position to allow more clearance on the left side of the robot. A sample of the front and back sensor values are shown in Figure 13 using a basic test script.

```
Current state is now: InitialState
ACTIVATING PLOW MOTOR TO OPEN PLOW
Current state is now: PlowOpenState
Would you like to exit? (Y/N)n
What is new front sensor value? (0 or 1)1
What is new back sensor value? (0 or 1)0
New Sensor Input:  0 1
ACTIVATING PLOW MOTOR TO CLOSE PLOW
Current state is now: PlowClosedState
Would you like to exit? (Y/N)n
What is new front sensor value? (0 or 1)0
What is new back sensor value? (0 or 1)1
New Sensor Input:  1 0
ACTIVATING PLOW MOTOR TO OPEN PLOW
Current state is now: PlowOpenState
Would you like to exit? (Y/N)y
```

Figure 13: Plough States Prototype Script Output (including sample inputs)

6.2.2 Implemented Plough States Chart

Figure 14 shown below displays the final implemented state chart for the plough. It was decided to not have the plough close after opening at the start of the simulation, since the snow spheres impeded this movement and the closing of the plough offered little benefit in terms of obstacle avoidance. As the state chart shows, the states always progress regardless of the proximity sensor input. On initialization, the plough always moves into the 'Plow Opening' state in which the revolute motors acting as the panel hinges are activated to begin opening the plough. Once the panels have arrived at the correct orientation, the hinge motors are shut off to leave the plough in an opened state for the remainder of the simulation.

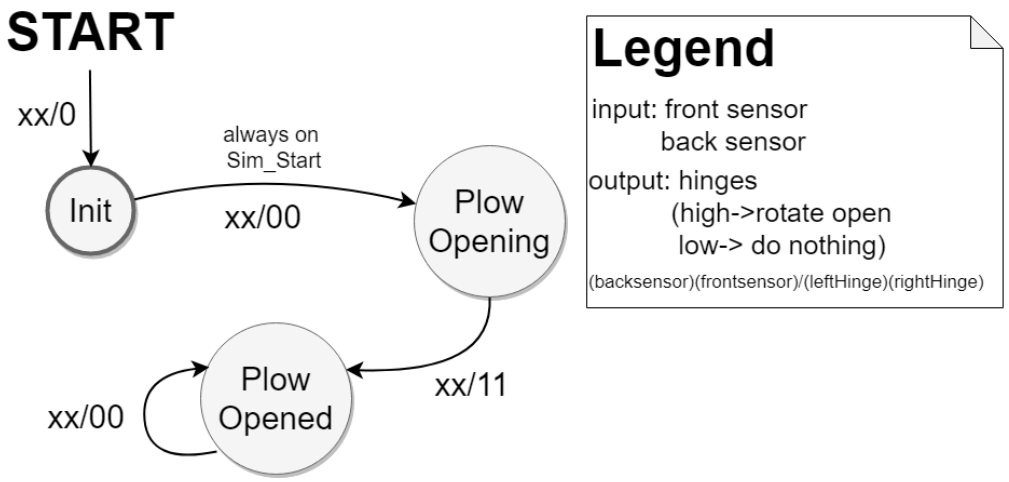

Figure 14: Plough States Chart

7 Sequence Diagrams

7.1 Robot Body Pathfinding Sequence Diagram

The path finding algorithm sequence diagram provided in Figure 15 is the first version of a pathfinding algorithm which shows the CoppeliaSim simulation environment as the client and a sample of one proximity and one vision sensor. The state of proximity or vision sensor is checked and if it is true, the velocity is adjusted in the left and right joints of the robot to make it turn left or right as required, else the velocity is adjusted in the left and right joints of the robot to make it go straight.

Table 7: Robot Pathfinding V1 Sequence Diagram Methods

Function Name	Description
Startsimulation( )	Function to start the remote api client
getObjectHandle(Proximity_sensor)	Retrieves the Proximity_sensor object
getObjectHandle(joint)	Retrieves the joint object
getObjectHandle(Vision_sensor)	Retrieves the Vision_sensor object
simxReadProximitySensor()	Reads the state of the proximity sensor
simxReadVisionSensor()	Reads the state of the vision sensor
verifydetectionstate()	Checks if the state of the sensor is true (detected an obstacle or line) or false (no detection)
setWheelVelocity(joint,value)	Adjusts the velocity in the joint of the robot using the parameter value

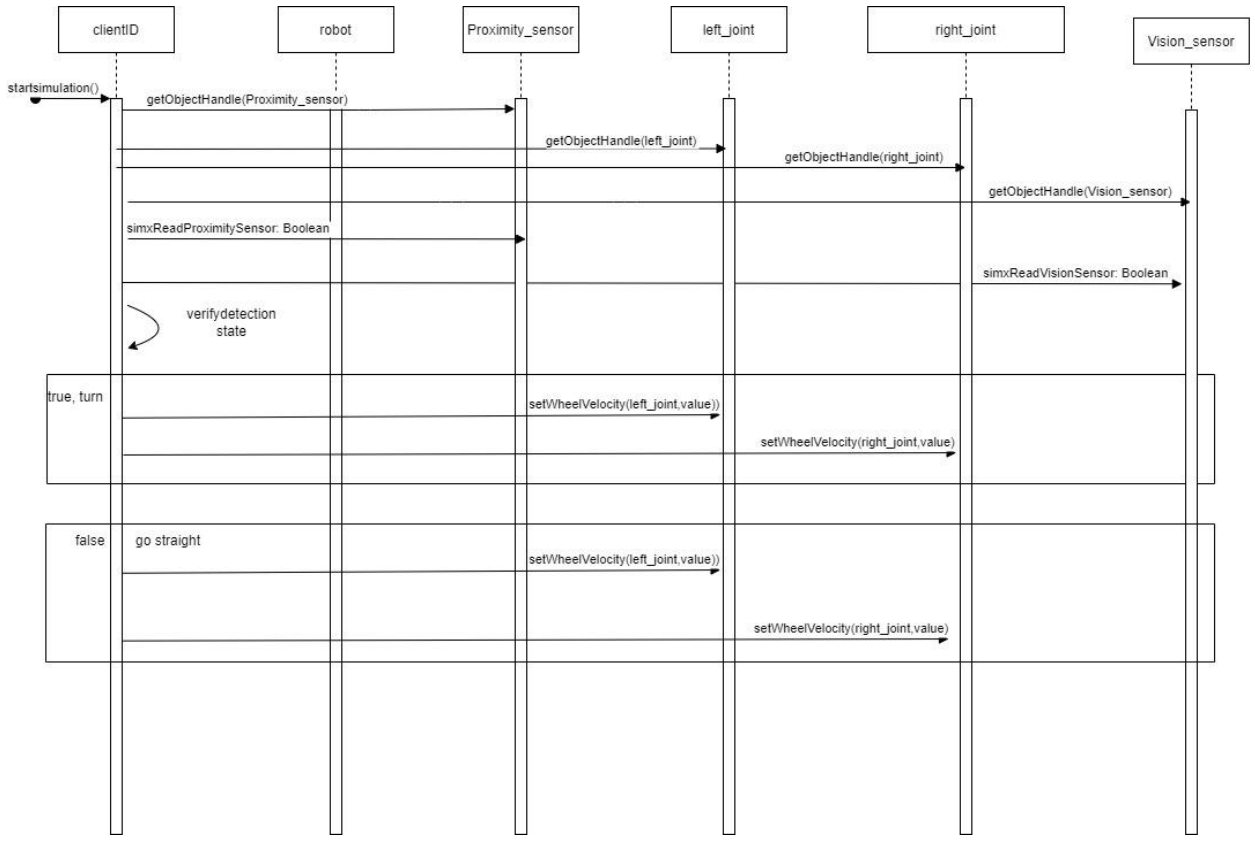

Figure 15: Robot Body Pathfinding Sequence Diagram version 1

The path finding algorithm sequence diagram provided in Figure 16 is the latest version of the pathfinding algorithm which shows the CoppeliaSim simulation environment as the client, SnowPlowRobot class responsible for controlling all the robot modules and implementing the path finding algorithm as SnowPlowRobot , WheelModule class responsible for the drive control of the robot by allowing the robot to move straight or backward and turn left or right as WheelModule, ObstacleModule class responsible for communicating with multiple proximity sensors on the robot using the Braitenberg algorithm to detect and avoid obstacles as ObstacleModule and VisionModule class responsible for communicating with multiple vision sensors on the robot and monitor which sensor detects a line as VisionModule.

When the simulation starts, all the robot modules are initialised and the robot will leave the starting area by driving straight 1 m, turn right and continue until an obstacle or a line is detected. The robot will use the Braitenberg algorithm to avoid obstacles. If the robot is already moving in the left direction, it will turn right upon detection of a line or an obstacle. If a line is detected, the robot will move outside the line to ensure that all snow is cleared in the arena, then, it will drive backwards, turn a random amount and begin driving forward again to re-enter the arena.

Table 8: Robot Pathfinding V2 Sequence Diagram Methods

Function Name	Description
SnowPlotRobot()	initialises all robot modules and prompt the robot to leave the starting area
startSimulation()	function to start the remote api client
LineDetectionApi.VisionModule(self.clientId)	initialises the VisionModule class
MovementApi.WheelModule(self.clientId)	initialises the WheelModule class
straightDist(distance,velocity)	prompts the robot to drive straight at the specified velocity for the specified distance
turnRight(deg)	turns the robot to the right by the specified deg degrees and moves the robot to the “TURNING_RIGHT” state
turnLeft(deg)	turns the robot to the left by the specified deg degrees and moves the robot to the “TURNING_LEFT” state
straight(velocity)	prompts the robot to drive straight at the specified velocity
pathFindingAlgorithm()	The main path finding algorithm, calls other methods and ensures that the robot detects lines and obstacles and avoids obstacles. The velocity and direction of the robot is also adjusted as required
backward(distance,velocity)	robot drives backward at the specified velocity for the specified distance then stops
stop()	stop the robot and moves it to the “STOP” state
checkForObstacle(motorControl, clientId, velocity)	checks if an object is detected and uses the Braitenberg algorithm to avoid obstacles
checkForObstaclePlow(clientId)	checks if an object detected is in the plow area

checkForLine()	checks if a line is detected, if a line is detected, the robot will move a bit more, then it will drive backward and turn a random angle
detectLine()	detects if a line has been crossed

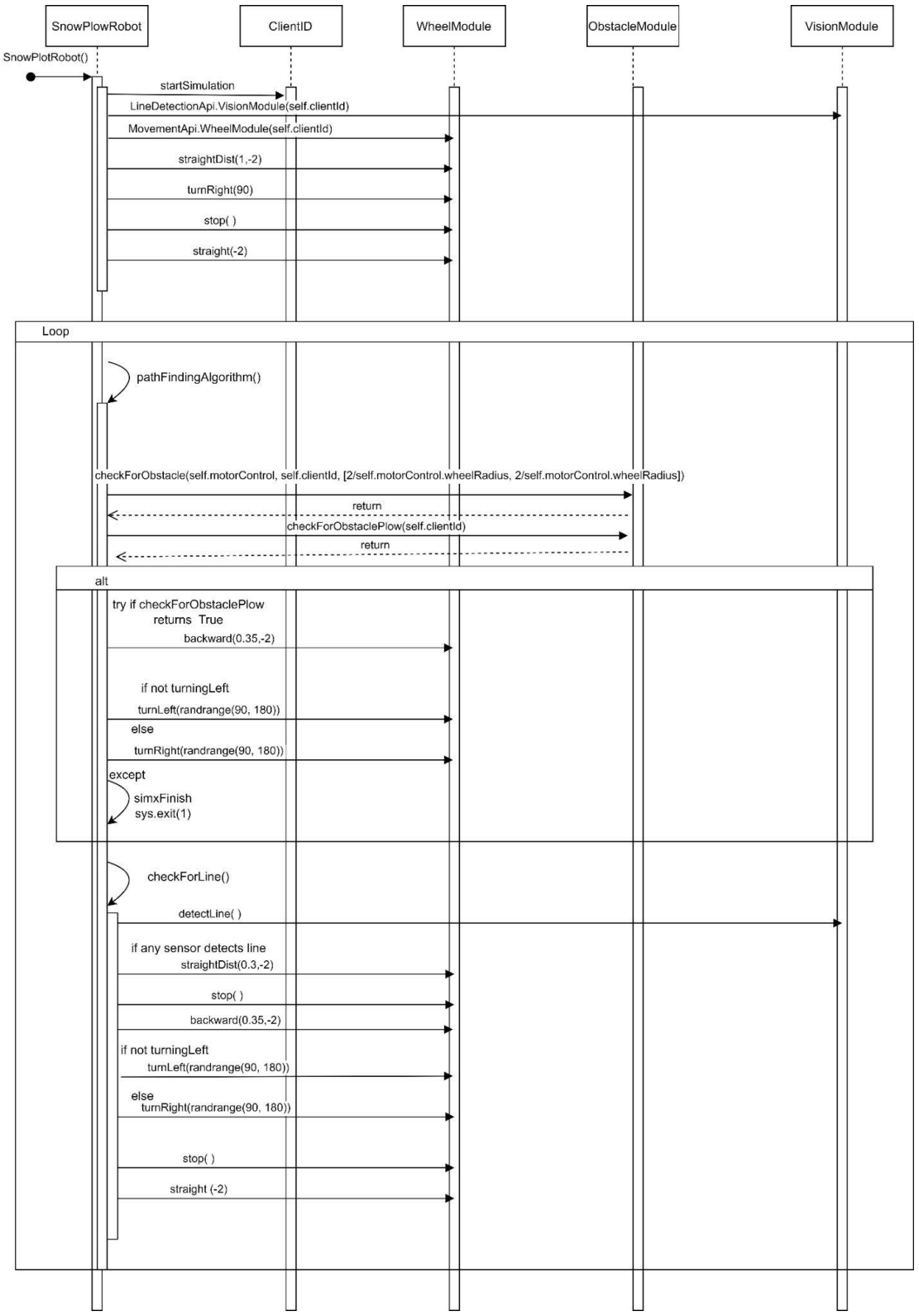

Figure 16: Robot Body Pathfinding Sequence Diagram version 2

7.2 Robot Plough Sequence Diagram

7.2.1 Robot Plough Prototype Sequence Diagram

The prototype plough sequence diagram is provided in Figure 17 below. At the beginning of the simulation the robot will activate the plough motors to open the plough. The robot will continue to poll the proximity sensors to look for obstacles. If an obstacle is detected the robot will close the plough. When the obstacle is no longer detected the robot will reopen the plough.

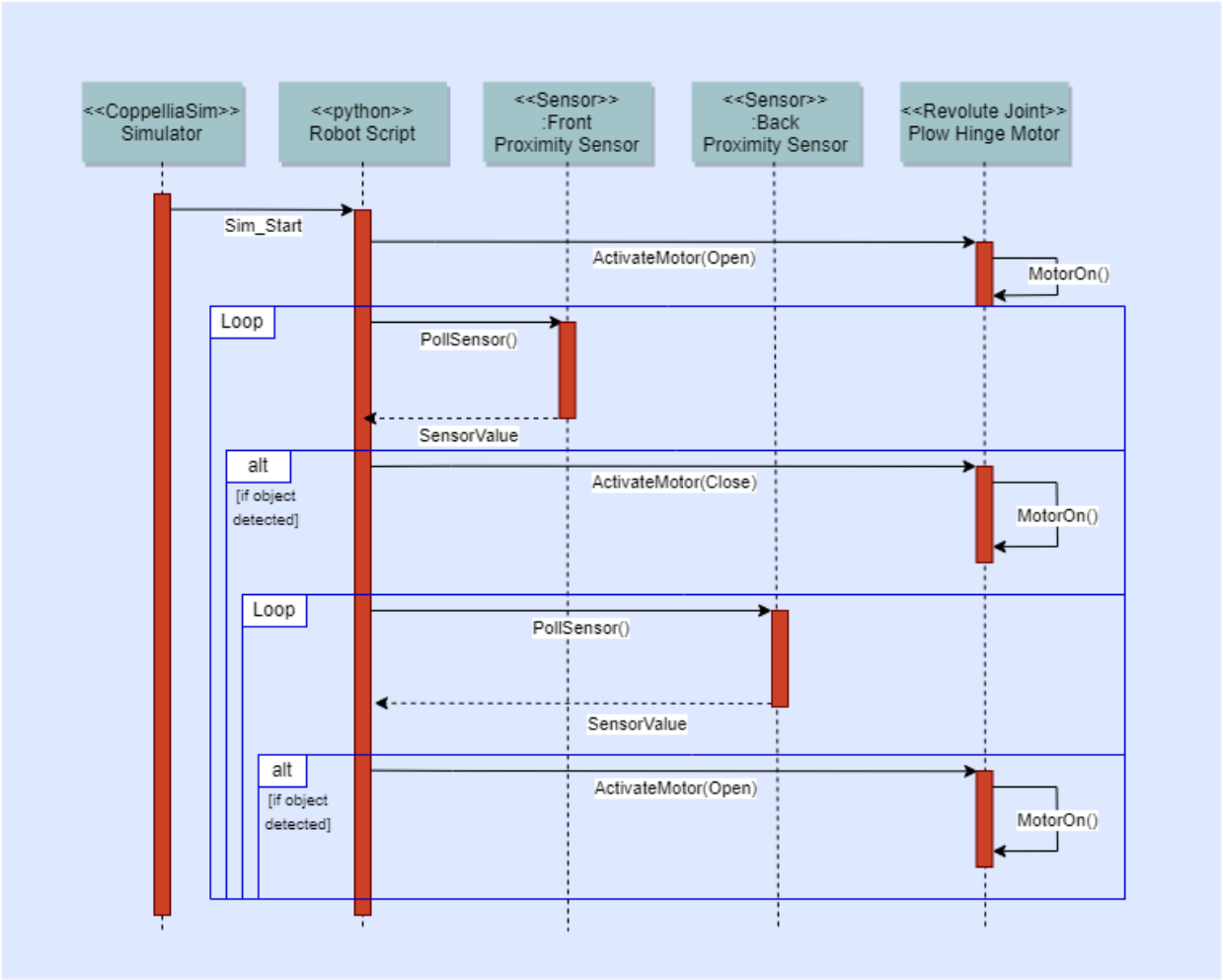

Figure 17: Prototype Robot Plough State Transition Sequence Diagram

7.2.2 Robot Plough Implemented Sequence Diagram

The implemented plough sequence diagram is provided in Figure 18 below. In testing the team encountered issues with the plough closing when the robot encountered an obstacle as snow collected in the plough would impede this movement. The sequence diagram was modified to open the plough on simulation start and not to close the plough during the simulation.

Brief descriptions of each of the implemented functions are as follows: ‘sim.simxStart()’ begins the remote API connection between the path algorithm script to the CoppeliaSim simulation. ‘plow.open()’ is a function used by the path algorithm script that sends an initial message to the simulation to use a call function script on an object’s child script. The ‘simxCallFunctionScript()’ function calls the ‘onOpen()’ function found in the child script of the plow_middle_panel object, which then sets the ‘open’ variable to True. Once the ‘open’ variable is set to true, the “sysCall_actuation()” function which is consistently looping on every simulation tick will get the current object orientation (through ‘getObjectOrientation()’) of the left and ride panels. Should the left and right panels be less than fully open, the motor on the hinges will be activated using ‘setJointTargetVelocity()’ but should the panels be fully open, the revolute motors on the hinges will be set to zero and the control boolean ‘open’ will then be set to false (meaning the plow has successfully opened).

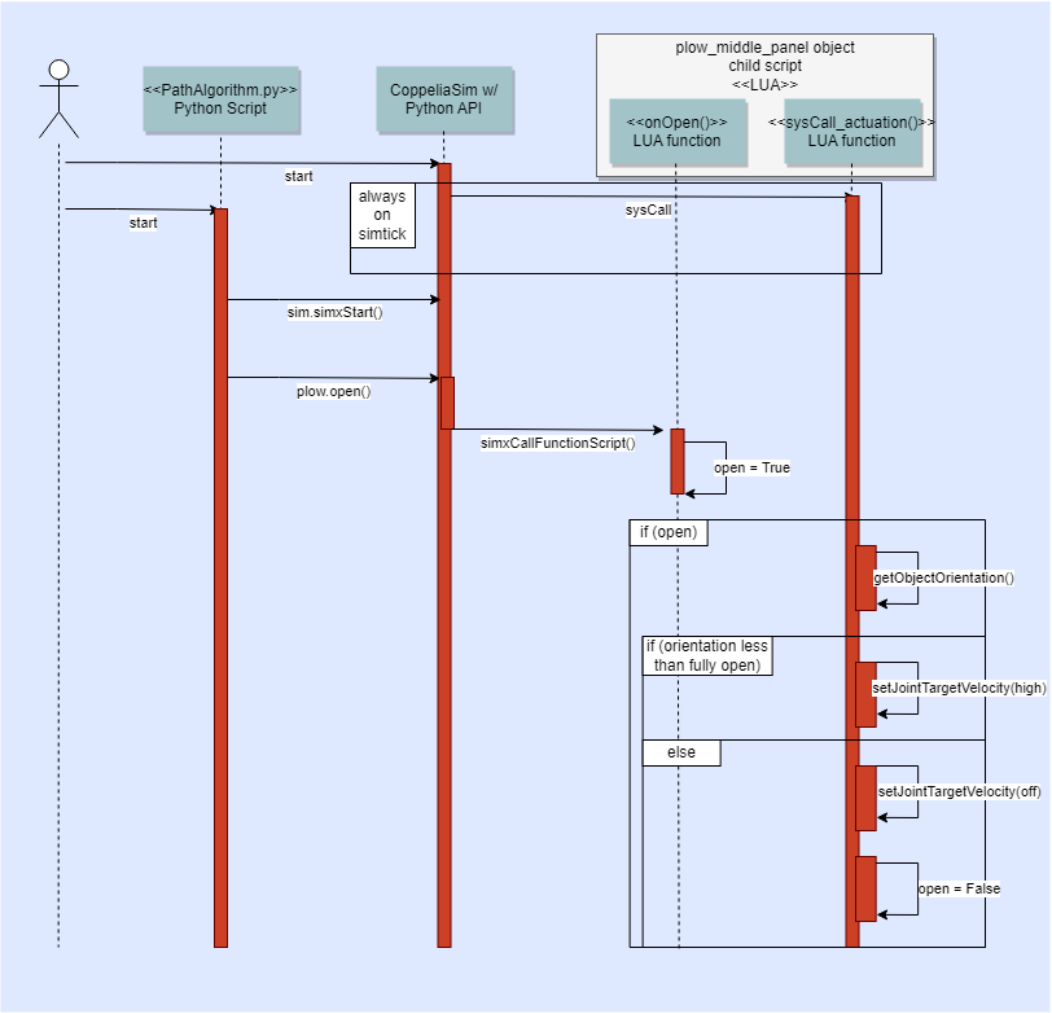

Figure 18: Prototype Robot Plough State Transition Sequence Diagram

8 Project Budget

8.1 Hardware Cost

8.1.1 Vision Sensor

An IR sensor will be used to detect the outer perimeter that is marked by a black line. The IR sensor has an IR transmitter module that emits infrared radiation, and the reflected radiation is detected by the IR receiver module. The black line will absorb the radiation and will allow the robot to detect the outer perimeter. As the robot only needs to detect the black line, as opposed to differentiating between different surface colours, a digital sensor is chosen. The sensor will also be mounted to the base of the robot body and must detect the line with a range of 0.5 mm. The Infrared Sensor Module (TCRT5000) [1] is selected as it satisfies all criteria. This sensor can be purchased from Canada Robotix for $3.19 CAD.

8.1.2 Proximity Sensor

An ultrasonic sensor will be used to detect moving and stationary obstacles. The ultrasonic sensor has a transmitter module that emits sound waves, and the reflected wave is detected by the receiver module. Using Sound navigation and Ranging (sonar) the robot can detect and avoid objects in its path allowing it to safely achieve its goal of clearing snow from designated areas. The sensor must be able to detect objects from 10 cm to 300 cm. The sensor must also have a measuring angle of 30 degrees. The HC-SR04 Ultrasonic Ranging Sensor [2] is selected as it satisfies all criteria. The sensor can be purchased from Canada Robotix for $5.09 CAD.

8.1.3 Wheel Speedometer

A mechanism to measure the speed of the wheel is necessary in order to measure the distance the robot has travelled. By multiplying the robot's wheel circumference by the number of revolutions observed the distance travelled can be calculated. By mounting a Hall-effect sensor on the frame of the robot and a strong magnet on the wheel of the robot. The robot is able to measure each wheel revolution. The AST1006-A Hall Sensor [3] is selected as it satisfies all the criteria. The sensor can be purchased from Tiny Circuit for $5.95 CAD.

8.1.3 Gyroscope

A gyroscope measures the rotational velocity of the robot. Using this, the angular position of the robot can be measured. This is deployed when the robot needs to rotate a specific amount of degrees. The MPU-6050 Gyroscope [4] is selected as it satisfies all criteria. The sensor can be purchased from SparkFun for $32.50 CAD.

The cost for all sensors required to complete this project are itemised in the following section.

8.2 Planned Value and Budget at Completion

The planned value (PV) is the authorised budget assigned to the scheduled work. Each team member is assigned an activity corresponding to a work breakdown structure component each week to complete. It is estimated that each activity takes approximately 2.5 hours to complete. Since the team has 4 members, the planned value of work for each week is 10 hours. Each team member has an hourly wage of $35/hr. The budget at completion (BAC) is the total planned value for the project. The BAC is estimated to be 70 hours, this accounts for 10 hours each week performed by the team over the span of 7 lab periods. The designated lab period is 4 hours each week, this allows each team member to have an extra 1.5 hours a week to complete their assigned activity without interfering with other school and personal commitments. This extra time is shown as the *Management Reserve* in the graph below. The project was completed on time and over budget as activities took slightly longer to complete than planned.

Table 9: Itemised Report of Planned Values

Item	Price	Quantity	Supplier
TCRT5000	$3.19	3	Canada Robotix
HC-SR04	$5.09	11	Canada Robotix
AST1006-A	$5.95	2	Tiny Circuit
MPU-6050	$32.50	1	SparkFun
Robot Body	$3000	1	Transformix
Team Member Salary (7 weeks)	$612.50	4	Carleton University
Total	$5,559.96		

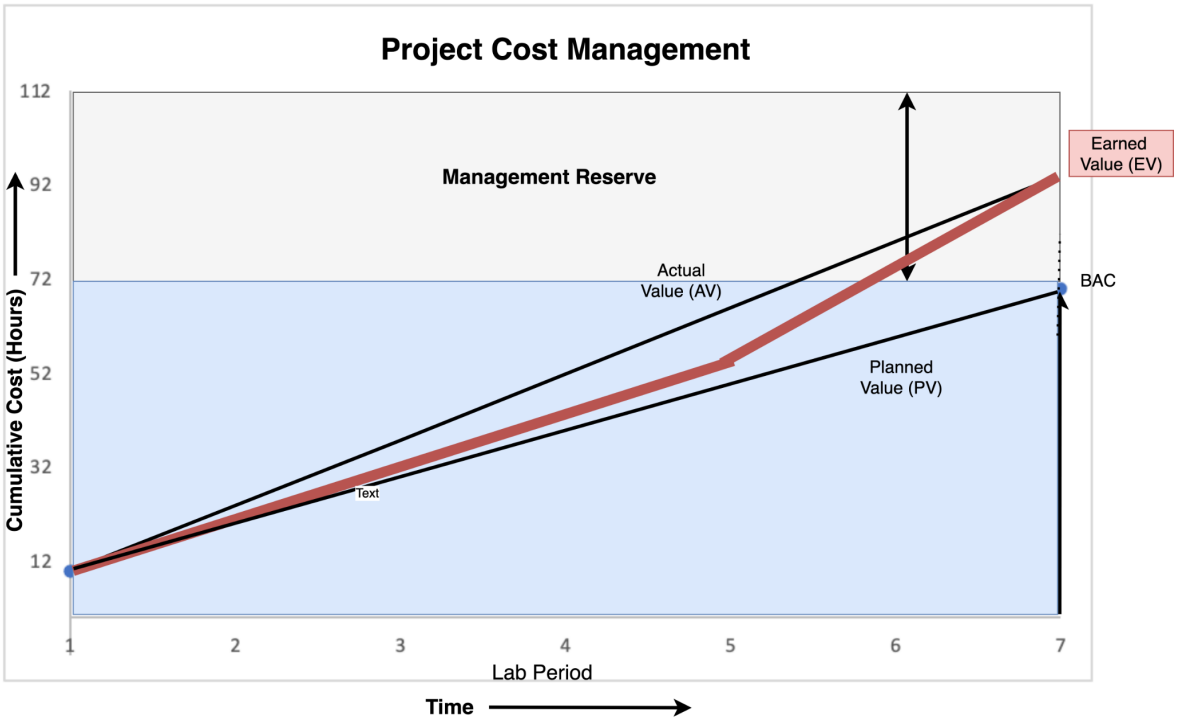

Figure 19: Project Cost Management Diagram

9 Control Charts

Control charts were used to evaluate the robot's ability to meet the project requirements documented in Section 2.1. Each functional requirement was tested 5 times and the data was documented in the following control charts. The control line is the average value of the data collected. The upper limit is placed 3 standard deviations above the control line. The lower limit is placed 3 standard deviations below the control line.

9.1 Requirement 1 Control Charts

The control chart shown below in Figure 20 displays the result of meeting requirement 1 in the x-direction at the start of the simulation for each of the 5 trials. The x-direction was unchanged for each trial (at the simulation start), therefore it clearly shows how the actual size of the robot in the x-direction was always below the upper limit (and at the exact same value) for each trial.

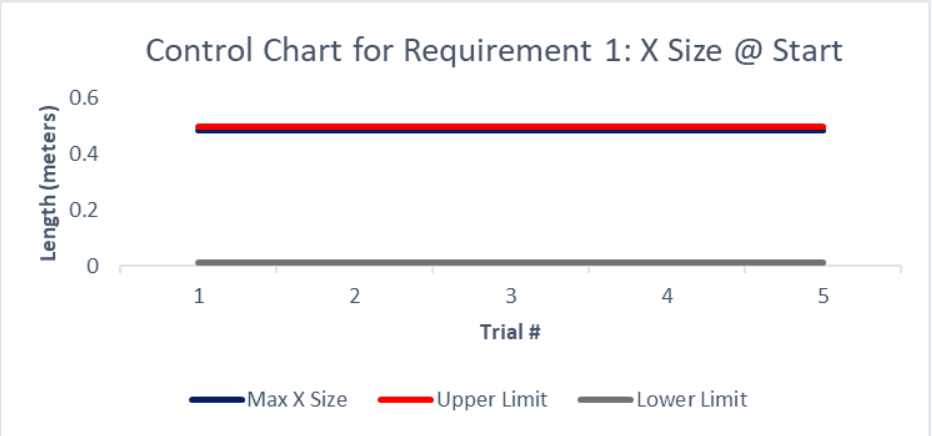

Figure 20: Control Chart for Robot Meeting Dimensioning Constraint in X-direction at Start of Simulation

The control chart shown below in Figure 21 displays the result of meeting requirement 1 in the y direction at the start of the simulation for each of the 5 trials. The y-direction was unchanged for each trial (at the simulation start), therefore it clearly shows how the actual size of the robot in the y-direction was always below the upper limit (and at the exact same value) for each trial.

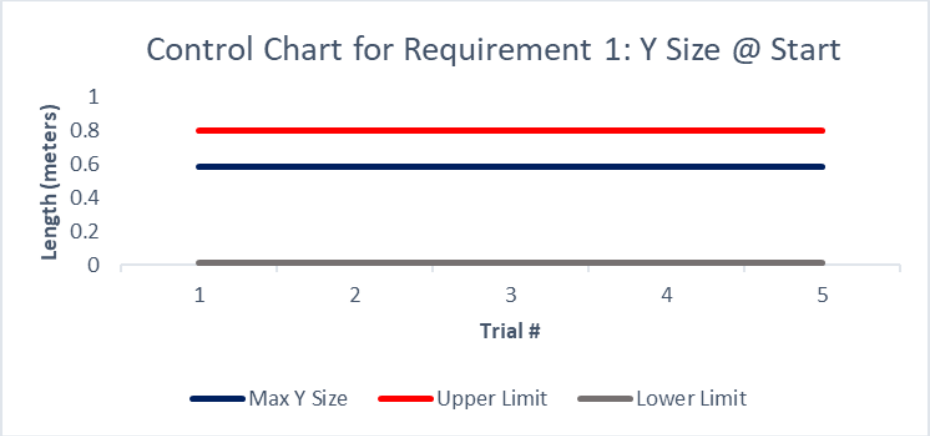

Figure 21 Control Chart for Robot Meeting Dimensioning Constraint in Y-direction at Start of Simulation

The control chart shown below in Figure 22 displays the result of meeting requirement 1 in the z direction at the start of the simulation for each of the 5 trials. The z-direction was unchanged for each trial (at the simulation start), therefore it clearly shows how the actual size of the robot in the z-direction was always below the upper limit (and at the exact same value) for each trial.

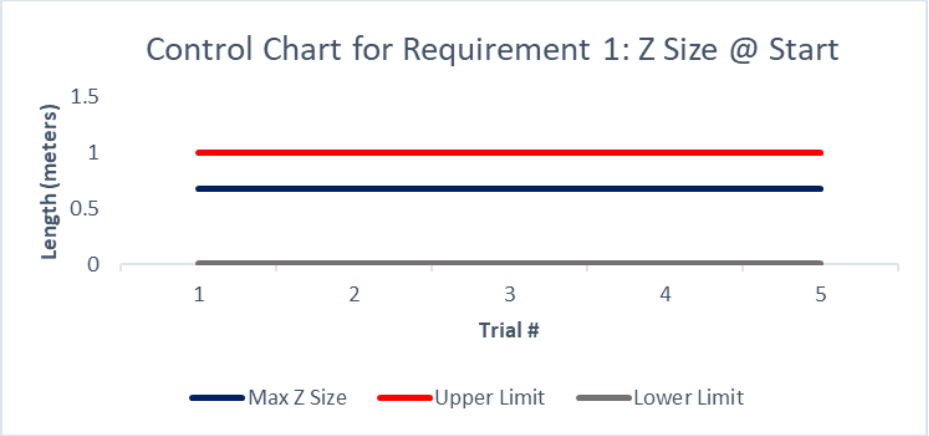

Figure 22: Control Chart for Robot Meeting Dimensioning Constraint in Z-direction at Start of Simulation

The above control charts demonstrate that the robot satisfies requirement 1, as the robot stays within the size constraint for the start of the simulation.

9.2 Requirement 2 Control Charts

The following 3 control charts document the maximum length, width, and height of the robot observed during simulation. The height of the robot is not expected to change and should always remain 0.7m as shown in Figure 25. The robot's width and length will expand during the simulation when the plough opens. The maximum length and width expected is 0.77m and 0.79m, respectively. This satisfies requirement 2.

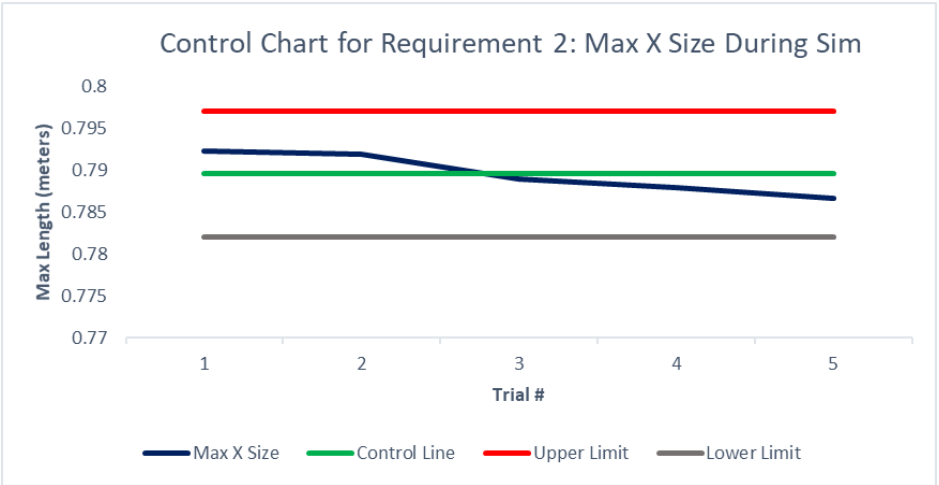

Figure 23: Control Chart for maximum width during simulation

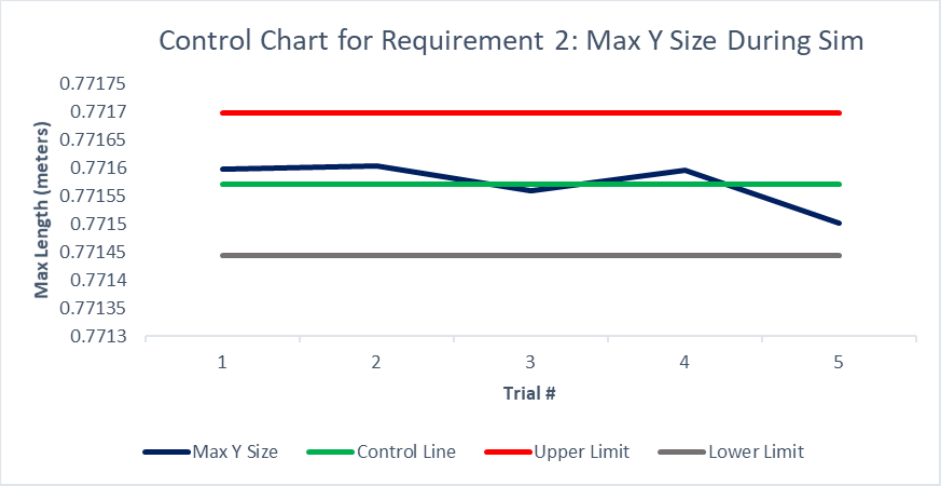

Figure 24: Control Chart for Maximum length during simulation

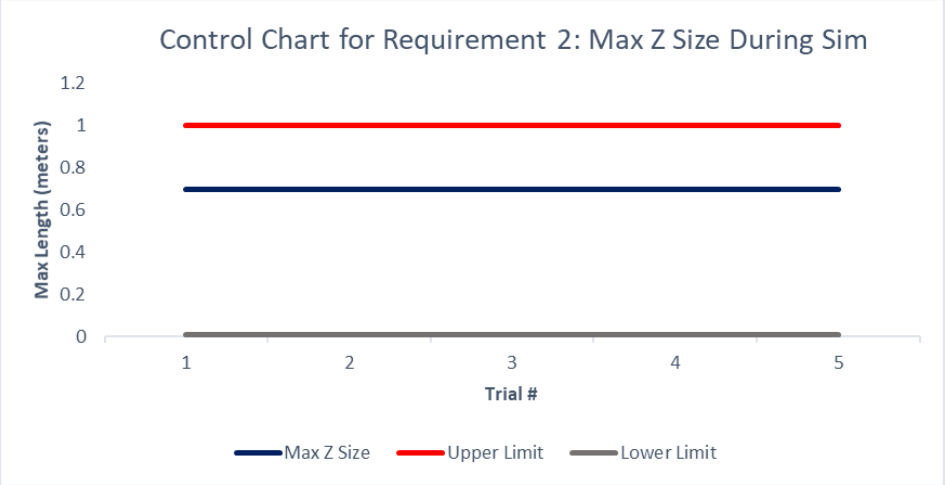

Figure 25: Control Chart for maximum height during simulation

9.3 Requirement-3 Control Chart

The CoppliaSim snow plough simulation was run 5 times on Training Map 1 to gather data on how many snowballs the robot could successfully remove from the arena. As shown in the control chart below, the robot is expected to remove 53-231 snowballs from the arena when operating normally.

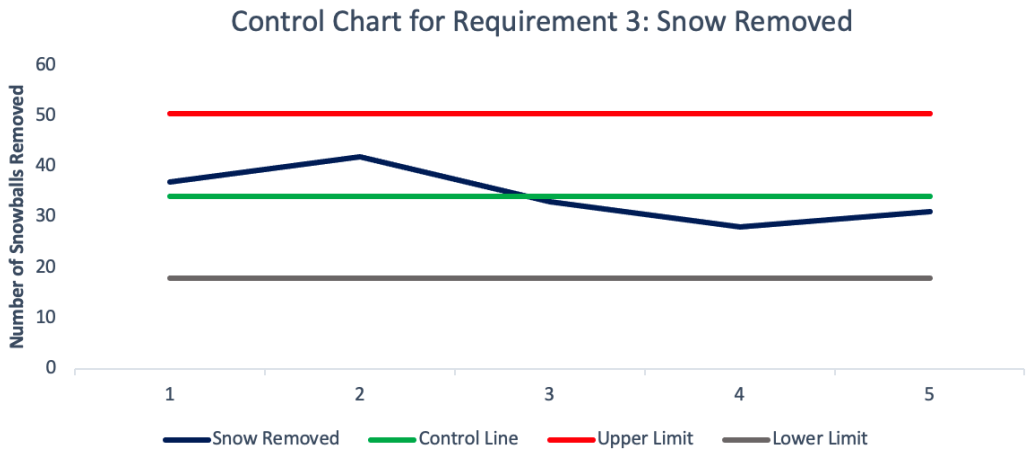

Figure 26: Control Chart for Requirement 3 (Snow Removed from Arena)

Table 10: Data for Requirement 3 (Snow Removed from Arena) Control Chart

Trial	Snow Removed	Control Line	Upper Limit	Lower Limit	Average	Standard Deviation
1	112	142.6	231.55898	53.64102069	142.6	29.652993
2	137	142.6	231.55898	53.64102069		
3	155	142.6	231.55898	53.64102069		
4	122	142.6	231.55898	53.64102069		
5	187	142.6	231.55898	53.64102069		

9.4 Requirement-4 Control Chart

The CoppeliaSim snow plough simulation was run 5 times on Training Map 2 to gather data on how many times the robot could successfully detect and avoid obstacles in the map. As shown in the control chart below, the robot is expected to detect and avoid obstacles 18 - 32 times in the training map when operating normally.

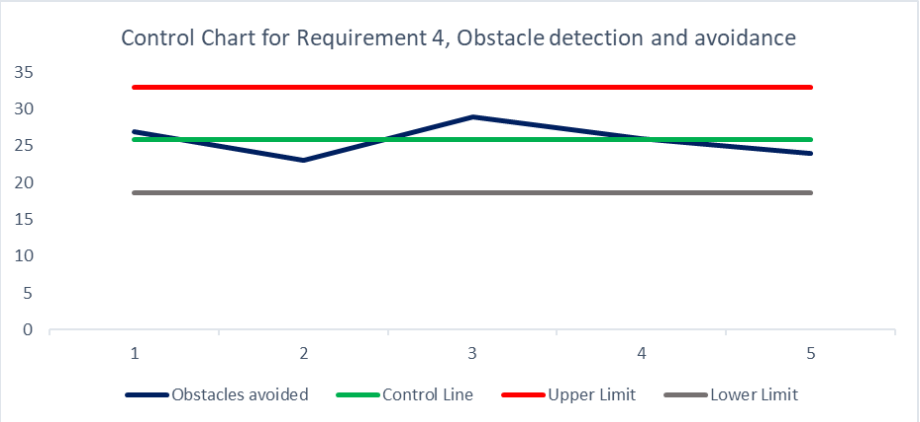

Figure 27: Control Chart for Requirement 4 (Obstacles detected and avoided)

Table 11: Data for Requirement 4 (Obstacles detected and avoided) Control Chart

Trial	Obstacles avoided	Control Line	Upper Limit	Lower Limit	Average	Standard Deviation
1	27	25.8	32.96240	18.63759817	25.8	2.387467
2	23	25.8	32.96240	18.63759817		
3	29	25.8	32.96240	18.63759817		
4	26	25.8	32.96240	18.63759817		
5	24	25.8	32.96240	18.63759817		

9.5 Requirement-6 Control Chart

The CoppeliaSim plough simulation was run 5 times on Training Map 1 to gather data about the maximum speeds of the robot. The maximum acceptable robot speed, as described in the requirements, is 2 m/s. The average maximum observed robot speed is 1.97 m/s, therefore the robot is deemed to pass this requirement.

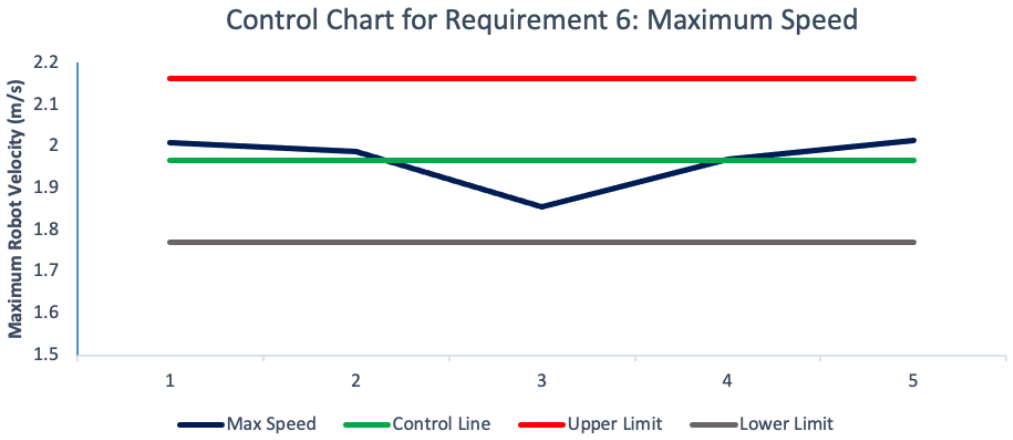

Figure 28: Control Chart for Requirement 6 (Maximum Robot Speed)

Table 12: Data for Requirement 6 (Maximum robot speed) Control Chart

Trial	Maximum Speed (m/s)	Control Line	Upper Limit	Lower Limit	Average	Standard Deviation
1	2.008	1.967	2.163	1.771	1.967	0.065
2	1.989	1.967	2.163	1.771		
3	1.855	1.967	2.163	1.771		
4	1.969	1.967	2.163	1.771		
5	2.015	1.967	2.163	1.771		

9.6 Requirement-11 Control Chart

The CoppliaSim snow plough simulation was run 5 times on Training Map 1 to gather data on the total number lines detected during the simulation. As shown in the control chart below, the robot is expected to detect the line 18-50 times during a simulation when operating normally.

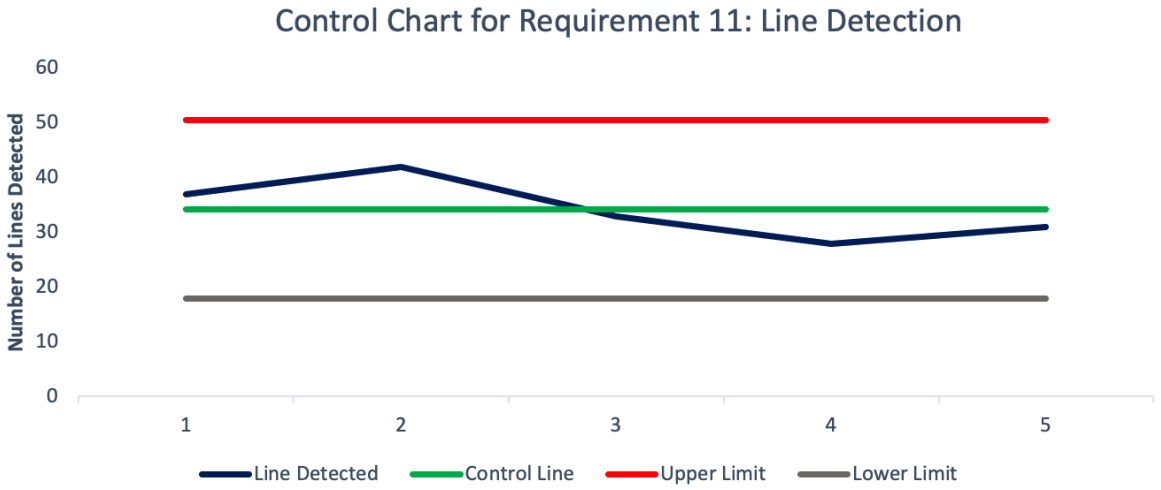

Figure 29: Control Chart for Requirement 11 (Black Perimeter Line Detected)

Table 13: Data for Requirement 11 (Black Perimeter Line Detected) Control Chart

Trial	Lines Detected	Control Line	Upper Limit	Lower Limit	Average	Standard Deviation
1	37	34.2	50.54931	17.85068809	34.2	5.449771
2	42	34.2	50.54931	17.85068809		
3	33	34.2	50.54931	17.85068809		
4	28	34.2	50.54931	17.85068809		
5	31	34.2	50.54931	17.85068809		

10 Testing and Results

The unit testing used to test the robot's main functions before beginning simulation testing on the training maps was done in Python using its built-in `unittest` framework. Each function was assigned a unit test class containing several methods to test various functions. Because most of these functions relied upon the remote API connection, each test was performed individually using various situations according to what conditions were being tested.
For example, the unit tests concerning the vision sensors used to detect the black line defining the bounds of the given area, a scene was constructed such that three scenarios were possible; the robot was positioned on the line, the robot was positioned off the line, and one vision sensor was on top of the line while the others were off the line. For each of the scenarios, the robot was manually moved before another test was run. Though it is not the most efficient way to perform the unit testing, it does provide control over the environment and the conditions of each test. Each case has its own failure conditions, tested using the `assert` statements present in the unit test code. Each assert statement corresponded to a specific requirement or testing case defined in the project proposal.

10.1 Training Map 1 Results

The scene features two trees and two outside walls with a thin wall in the centre of the map. It contained 363 snowballs on average. There are no moving obstacles or non-right angles.

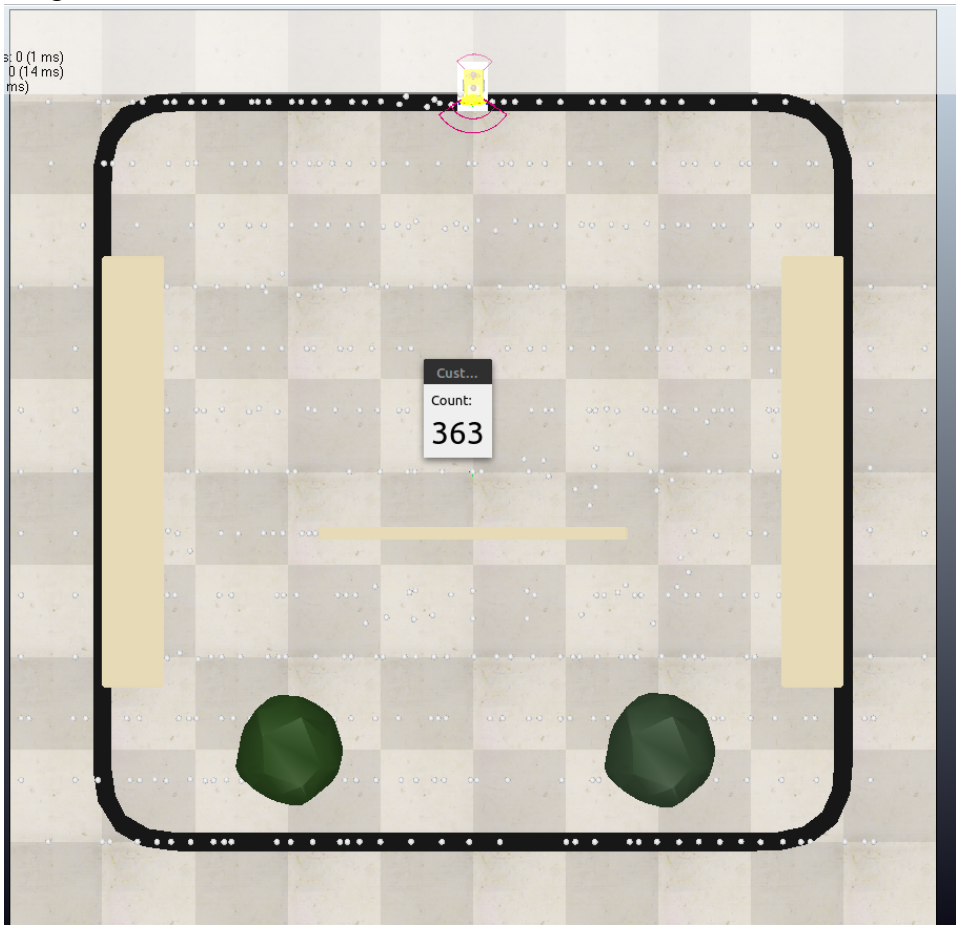

Figure 30: Starting position for robot in Training Map 1 simulation

Table 14: Simulation Results for Training Map1

Trial	Snow Removed	Lines Detected	Objects Avoided	Object Collisions
1	138	23	25	0
2	132	18	15	0
3	131	19	22	0
Average	133.7	20.0	20.7	0.0

After all 3 trials, it became evident that the algorithm's pseudo-random nature was inhibiting the robot's ability to traverse the entire map. Figure 31 shows the end of the third trial. The yellow boxes indicate the areas where the robot spent more than 15 consecutive seconds. This shows a significant issue of the robot getting stuck in corners and near obstacles that box the robot in on 2 or more sides.

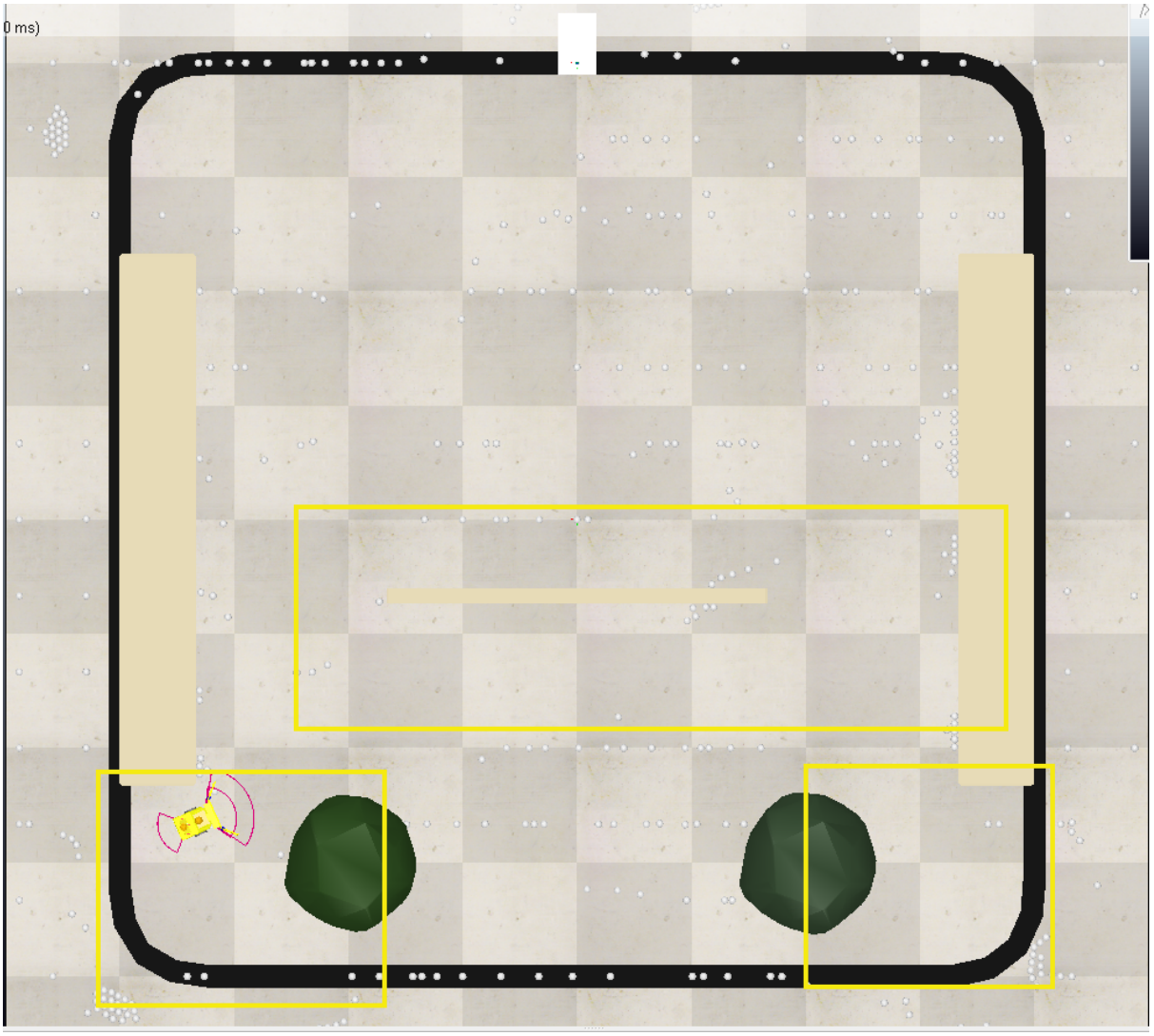

Figure 31: Results after Trial 3 for robot in Training Map 1 simulation

10.2 Training Map 2 Results

The scene at the start of the simulation for Training Map 2, along with the scene at the end of each 5-minute trial are pictured below in Figure 32. The training map featured 3 cuboid obstacles along with two tree obstacles.

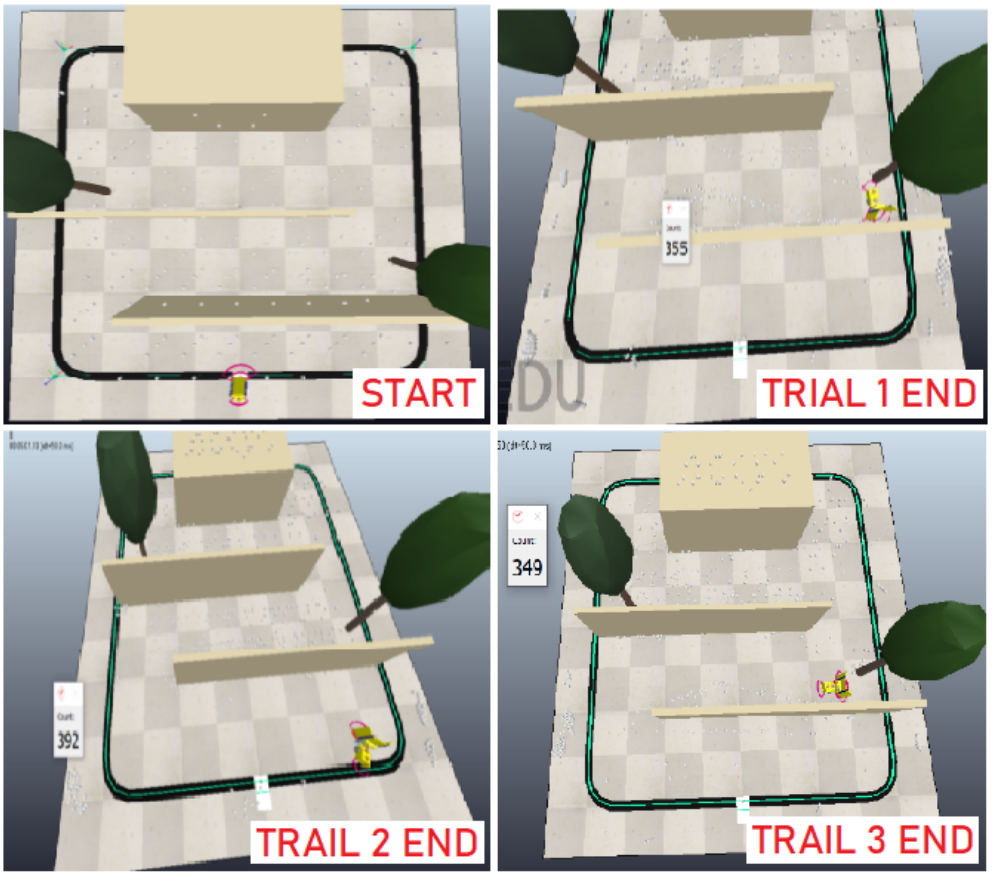

Figure 32: Sim Start of Training Map 2 Pictured Along with the End Result of All 3 Trials

The results for each simulation trial for training map 2 are summarised below in Table 15. Trial 3 garnered the most snow removed, with the least amount of lines detected as well as tying for the least amount of object collisions. Trial 2 fared the worst comparatively, with only 68 snow spheres removed, and detected the lines 25 times, meaning that the random choice of direction after encountering a line often resulted in almost immediately running into the line again.

Table 15: Simulation Results for Training Map2

Trial	Snow Removed	Lines Detected	Objects Avoided	Object Collisions
1	105	19	8	12
2	68	25	11	10
3	111	16	10	10
Average	94.67	20	9.67	10.67

Qualitative analysis of how the robot snow plough performed on training map 2 will be done with reference to the figure and its associated numbering displayed below in Figure 33. In all 3 trials, the robot snow plough failed to reach the upper half of the map, which is marked with ‘1’ in the figure below. The random nature of

selecting a direction to proceed in after detecting a line made it very difficult to traverse the full breadth of the map while simultaneously avoiding obstacles. The key challenge for our robot is distinguished in the figure below at '2'. The combination at '2' of the tree obstacle close to the cuboid wall which intersected with the contouring line posed an immense issue for the robot. When the robot made its way to this area, it would often become stuck, essentially thrashing between avoiding the tree, avoiding the wall and trying to get back into the outlined area, and thus not productively moving/clearing at all. Before the robot reached this problem area, it effectively cleared a good amount of snow which is depicted by the areas marked '3' in the figure below. These areas show the snow properly banked outside of the to-be-cleared area.

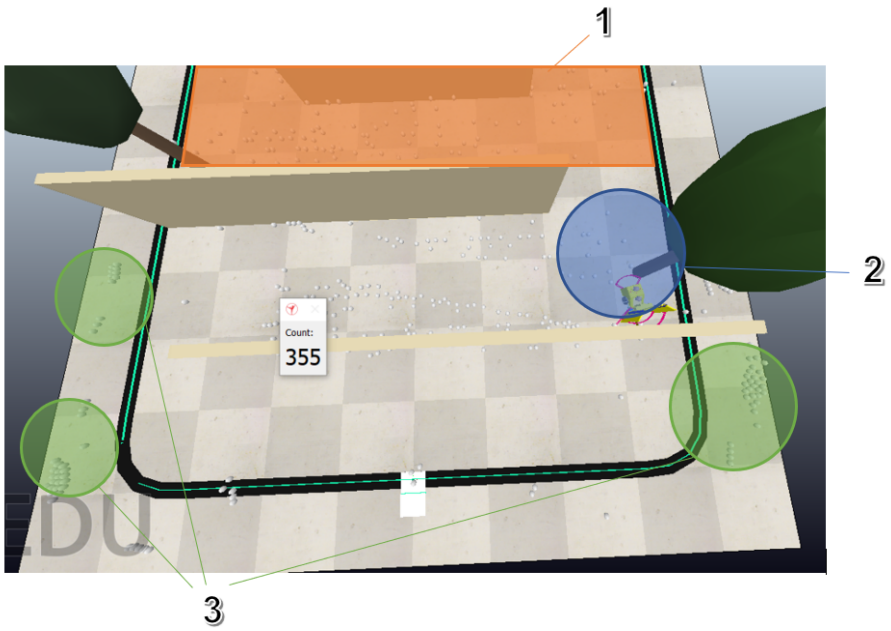

Figure 33: Qualitative Analysis of Robot Performance on Training Map 2

10.3 Training Map 3 Results

Before starting the simulation, the training map contained 492 snowballs. There are one moving obstacle and two non-moving obstacles.

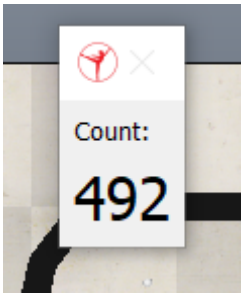

Figure 34: Initial snow count on Training Map 3

The statistics below represent the snow removed and the number of times lines are detected, objects are avoided and the robot collides with an object for each simulation trial.
Trial 1 had the second highest amount of snow removed with a count of 169 snowballs, detected lines 16 times, detected and avoided obstacles 23 times and

has no collision. Trial 2 had the least amount of snow removed with a count of 151 snowballs, detected lines 28 times, detected and avoided objects 17 times and had 1 collision with the moving obstacle. Trial 3 had the highest amount of snow removed with a count of 176 snowballs, detected lines 19 times, detected and avoided objects 17 times and had the highest number of collisions with a count of 3 collisions.

Table 16: Simulation Results for Training Map3

Trial	Snow Removed	Lines Detected	Objects Avoided	Object Collisions
1	169	16	23	0
2	151	28	17	1
3	176	19	17	3
Average	165.3	21.0	19.0	1.3

TRIAL 1 END

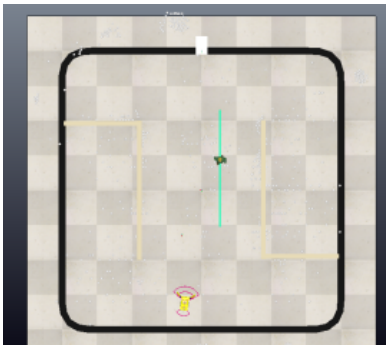

Figure 35: 323 Snowballs remaining after Trial 1 on Map 3

TRIAL 2 END

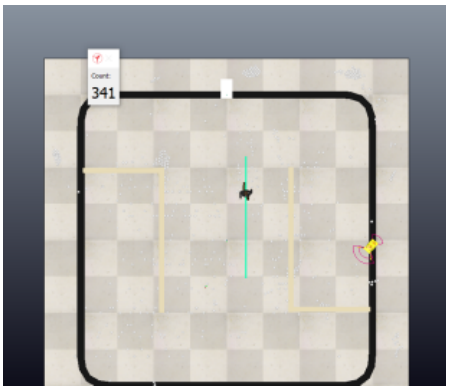

Figure 36: 341 snow remaining after Trial 2 on Training Map 3

TRIAL 3 END

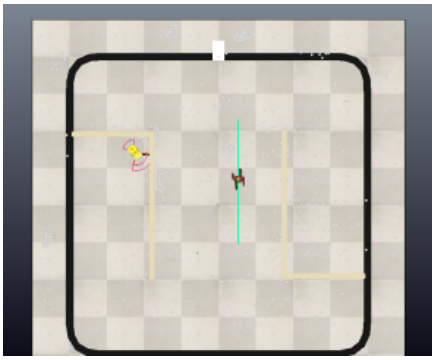

Figure 37: 316 snow remaining after Trial 3 on Training Map 3

Qualitative analysis of how the robot snow plough performs on training map is done with reference to the figure below and its associated numbering displayed in the Figure 38.
In all 3 trials, the robot was stuck for a while in the sections marked ‘1’. It would detect the line, turn a random angle and return within the boundaries. The random nature of selecting a direction to proceed in after detecting a line made the robot perform the same action repeatedly even if the snow in that section was already cleared.Since the robot would easily perform the same action repeatedly in sections ‘1’, it barely reached the lower sections of the training map. At section ‘2’, there were times when the robot failed to detect and avoid the moving obstacle and had a collision.

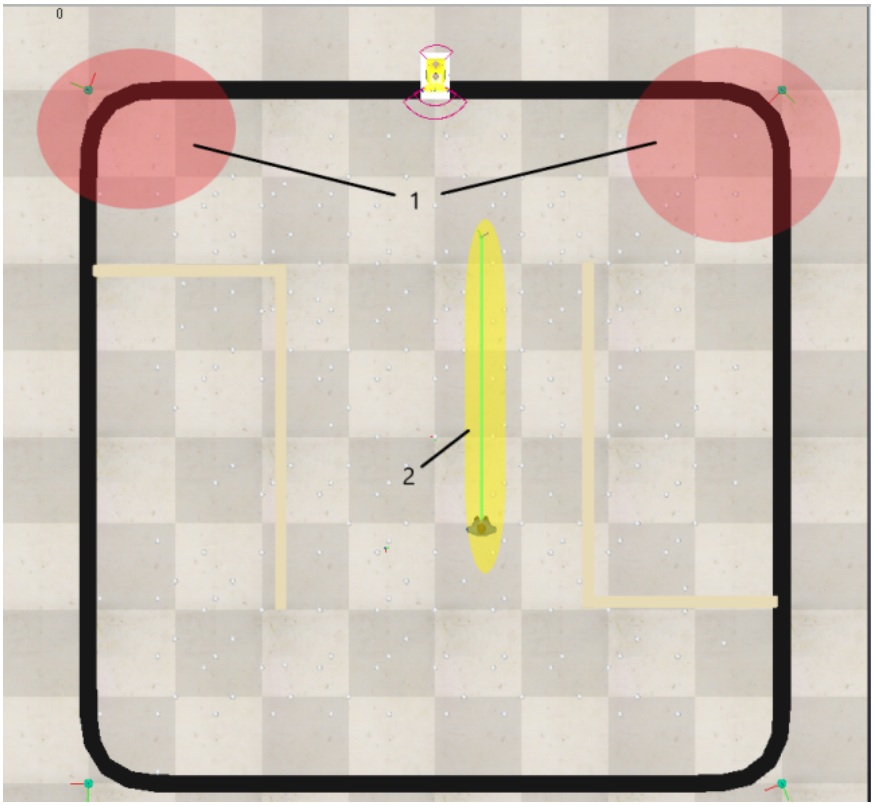

Figure 38 : Qualitative Analysis of Robot performance in Training Map 3

10.4 Training Map 4 Results

The performance of the robot on Training Map 4 was analysed and recorded over 3 trials. A summary of the quantitative results are provided in Table x below. There are 488 snow balls in Map 4 and the robot can remove approximately 40% of the snow in a 5 minute simulation.

Table 17: Simulation Results for Training Map4

Trial	Snow Removed	Lines Detected	Objects Avoided	Object Collisions
1	205	22	19	0
2	162	25	19	0
3	181	23	22	0
Average	182.7	23.3	20.0	0.0

The training map contains 4 thin rectangular boxes on different angles. The robot was able to avoid all obstacles, however it had difficulty with navigating the thin obstacles without losing many of the snow balls collected in the plough. The robot has 8 proximity sensors that are utilised in the Braitenberg algorithm implementation to avoid obstacles. However, if the robot approaches an obstacle and the proximity sensors for the Braitenberg algorithm fail to detect the obstacle there is a back-up sensor that spans the area of the plough, this is shown in Figure 39. If this sensor detects an object the robot will perform an emergency stop, reverse 0.3 m, then turn 45 degrees to the left or right. Since the corners of the boxes are very thin, if the robot is heading directly for a corner it will usually need to deploy this fail-safe mechanism because the object is in the gap between the proximity sensors range. When it does this, the plough loses all the snow balls collected.

To improve this limitation of the robot, it is recommended to add more sensors for the Braitenberg algorithm to account for thin objects.

Figure 39 : Robot fail-safe proximity sensor being deployed due to thin obstacle

11 References

Data Sheets
[1] Vishay Semiconductors, “Reflective Optical Sensor with Transistor Output”, TCRT5000 datasheet, Aug 2009.
[2] Elec Freaks, “Ultrasonic Ranging Module HC - SR04”, HC-SR04 datasheet, Nov. 2011.
[3] Tiny Circuits, “3 V Hall-Effect Linear Sensor with I2C Output”, AST1006-A datasheet, 2019.
[4] SparkFun, “MPU-6000 and MPU-6050 Product Specification”, MPU-6500 datasheet, Aug 2013.

12 Appendix

12.1 GitHub Repository Link

https://github.com/SYSC4805-Winter2022/sysc4805-project-group-3-l2-laser-lemon

12.2 Completed Activities Per Student (As Located on GitHub)

Activity ID	Activity Name	File Name	Completed By:
1.1	Define Robot Body	project_robotbody_v1.ttt	Timothy Knowles
1.2	Refine requirements for proximity sensor	project_robotbody_with_proximitysensor_v2.ttt	Chhavi Sujeebun
1.3	Refine Requirements for Vision Sensor	project_robotbody_with_proximityandvisionsensor_v3.ttt	Emma Boulay
1.4	Model plough in CoppeliaSim	PloughInitialModel.ttt	Denise Mayo
1.5	Implement proximity sensor	project_robotbody_with_proximitysensor_v2.ttt	Chhavi Sujeebun
1.6	Implement Vision Sensor	lineDetectionApi.py	Emma Boulay
1.7	Prototyping movement states and main state machine	robot_body_states.py, robot_body_statemach.py, RobotBodyStateDiagramv1.png	Denise Mayo
1.8	Define and Prototype Plough States	plow_states_chart.pdf plow_states_prototype.py	Timothy Knowles
1.10	Implement robot control API	movementApi.py	Emma Boulay
1.11	Implement Plough States	plowApi.py	Emma Boulay
1.12	Implement Motor for Plough	plow_v2_implementationTesting.ttt plow_python_basics.py	Timothy Knowles
1.12	Implement Motor for Plough	Plow v4 working motors.ttt	Emma Boulay
1.13	Proofread and format progress report	ProgressReportL2-Team3.pdf	Denise Mayo
1.14	Implement plough motor API	plowApi.py	Emma Boulay
1.15	Obstacle avoidance algorithm in python	Obstacle_avoidance_proximitysensor.py Training_Map_1_proximitysensor_implemented.ttt	Chhavi Sujeebun
1.15	Implement obstacle avoidance algorithm	obstacleAvoidanceApi.py	Emma Boulay
1.16	Document Real-World Sensors	Hardware Cost.pdf	Emma Boulay
1.17	Implement path finding algorithm	pathAlgorithm.py	Emma Boulay
1.19	Write and perform unit tests	unitTests.py, GeneralTesting.ods, RobotStatesTesting.ods, unitTests.xlsx	Denise Mayo
1.20	Perform integration testing on plough and body	L2G3-SnowPlowModel.ttm	Emma Boulay

1.23	Implement Robot Safety Lights	safetyLight.ttt	Emma Boulay
1.26	Perform testing on the testing map 1 and record results.	`Testing_Map1.ttt`	Denise Mayo
1.28	Proofread and format final report	FinalReport-L2G3.pdf	Emma Boulay

12.3 Breakdown of Team Contribution

Table 18: Code related contributions

Code Contribution	Description	Contributor
Files used for Demo		
PathAlgorithm.py	Responsible for controlling all the robot modules (Vision, Obstacle Avoidance, Plow, Motors) and implementing the pathing algorithm.	Emma Boulay + Denise Mayo
ObstacleAvoidanceApi.py	Responsible for communicating with the proximity sensors on the robot. It also utilises the Braitenberg algorithm to detect and avoid obstacles.	Emma Boulay
LineDetectionApi.py	Responsible for communicating with the vision sensors. It provides functionality to check which vision sensors detect a line.	Emma Boulay
MovementApi.py	Responsible for the drive control of the robot. It provides functionality for driving straight, backwards, and turning left or right.	Emma Boulay
PlowApi.py	Responsible for communicating with the plow module housed on the robot. It provides functionality to open and close the robot plow.	Emma Boulay
PlowControl.lua	Handles the plow opening and closing operations.	Emma Boulay
RobotMain.lua	Starts the Remote API Server. Provides lua functions for setting the wheel velocity and the Braitenberg algorithm.	Emma Boulay
SafetyLight.lua	Responsible for flashing the amber safety lights every 0.7 seconds.	Emma Boulay
Testing		
unitTests.py	Unit testing for main robot functions.	Denise Mayo

Table 19: Final Report related contributions

Contributor	Report Sections
Emma Boulay	• 1.1, 2.1, 2.3, 3.2, 5.1, 5.1.3, 5.2, 8.1, 8.2, 9.3, 9.6, 10.4
Denise Mayo	• 9.5, 10, 10.1
Chhavi Sujeebun	• 3.1,7.1,9.4,10.3
Timothy Knowles	• 5.1.1, 5.1.2, 6.2.1, 6.2.2, 7.2.1, 7.2.2, 9.1, 9.2, 10.2