SYSC 4805: Project Final Report

Autonomous Snowplow Robot

Professor: Prof. Mostafa Taha
Date: April 9th, 2021

Members:

Name	Student Number
Chukwuka Ihedimbu(ICE)	101081703
Shan Rameshkanna	101061978
Shaviyo Marasinghe	101019133

TABLE OF CONTENTS

1.0 INTRODUCTION

Team british racing green embarked on the challenge of implementing a simulation of an Autonomous robot that has the advanced ability to safely clear snow in a control environment. This project is initiated as a part of SYSC 4805 computer designs course and it follows the guidelines that are set by the course curriculum.

This report consists of the overall progress that was made in the Autonomous snowblower project from the start of the project on February 10th, 2021 to the date of submission of the progress report (April 10th, 2021).

2.0 BACKGROUND

2.1 Overall Objective

Every winter, there are about 100 people die in the United States by shovelling the snow and in a six year span of study shows that about 1647 fatalities from cardiac related injuries caused by snow shovelling [1].

The workload of the project was divided among the three members of the group and the activities were followed according to the schedule provided by the gantt chart below in figure 1.

	Task Name	Duration week days
1	Draw schematic diagram of proposed end unit	1 day
2	Design chassis	1 day
3	Design powertrain	2 days
4	Design auger and impleller assembly	2 days
5	Design chute and cap assembly	2 days
6	Install multilayer sensors	1 day
7	Program powertrain	2 days
8	Program maneuverability	2 days
9	Program collision avoidance & obstacle detection	2 days
10	Program auger and impeller assembly	2 days
11	Program chute and cap assembly	2 days
12	Program pathfinding and navigational algorithm	3 days
13	Program communication between API and client	1 day
14	Unit testing	2 days
15	Ensure operational safety, failsafe mode and manual overrid	1 day
16	Iterative and intrusive test stages of end unit	3 days
17	Ensure no harm to the environment and list potential hazard	1 days

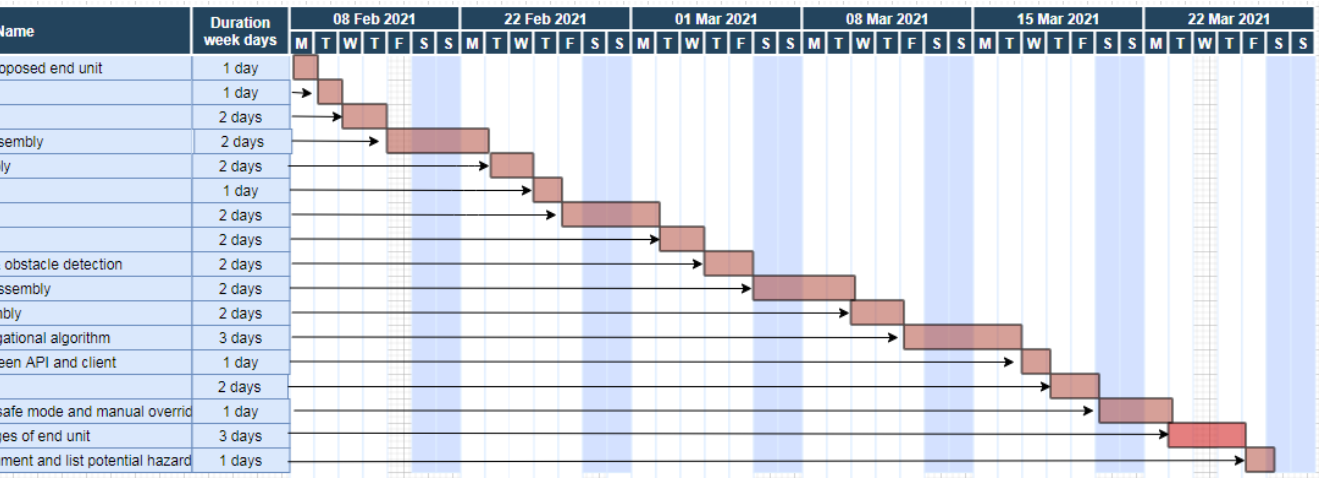

Figure 1: Gantt Chart of Project Schedule

2.2 Robot Architecture

This Autonomous robot was implemented using two main systems. The Robot system and the Remote API Server. The Robot System is where the Motors, Sensors, and physical objects make up the Robot. The Remote API server is where the background functionality of the robot will be computed. Figure 2 below outlines the overall architecture of the Robot that we have followed throughout the term.

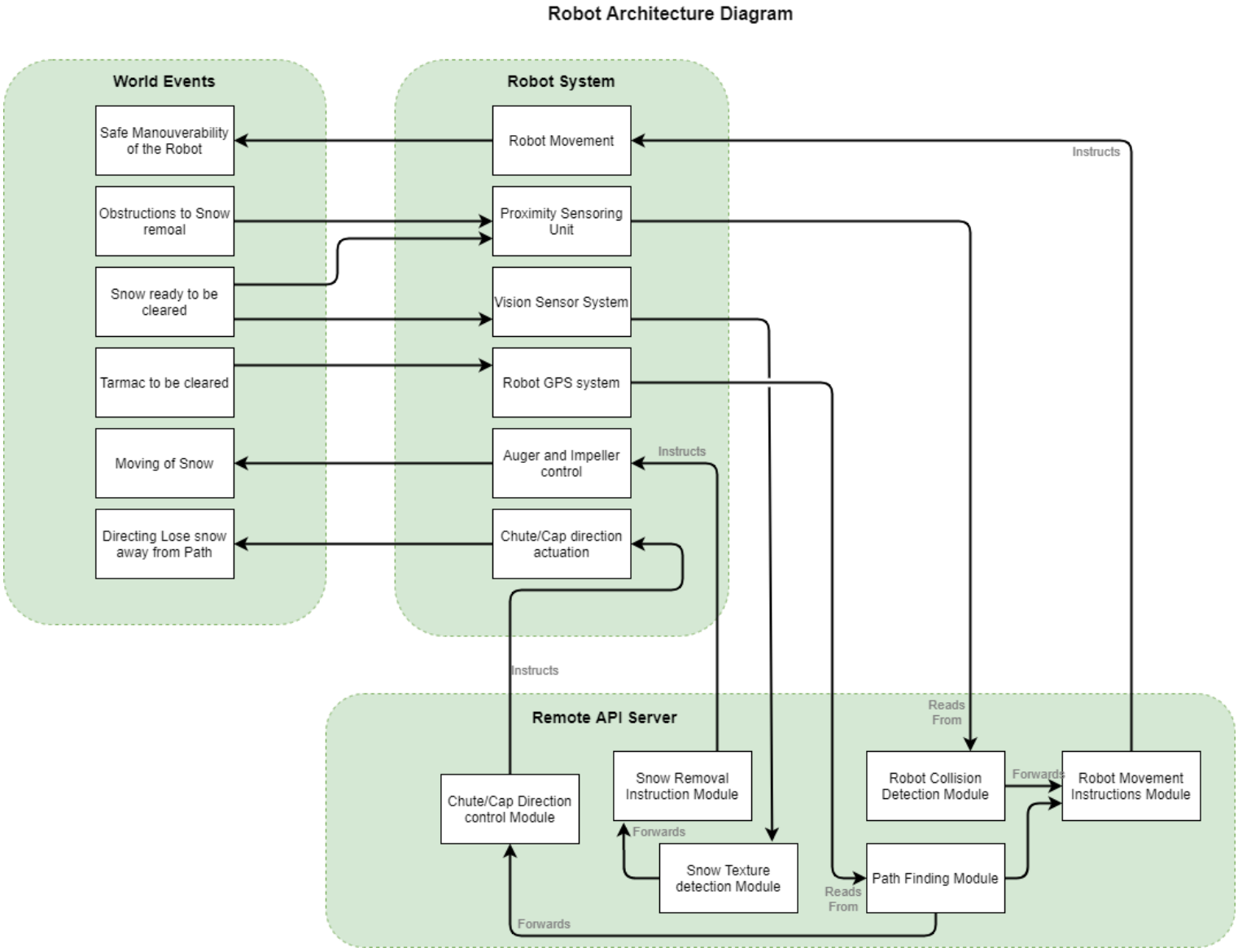

Figure 2: Robot Architecture Diagram of the Autonomous Snowblower Robot

2.2.1 Robot System

Robot System contains the components of the robot that will interact directly with the simulation environment. These components are grouped into units respective to the tasks they help to accomplish. These are the groups and the tasks that they contribute to:

Robot Movement Unit: This unit contains the two revolute joints connected to wheels that are controlling the robots's forwards and backwards movement as well as it's left or right movement.

Proximity Sensing Unit: This unit contains a group of proximity sensors around the robot to sense if any obstructions that may hinder normal operation are within the proximity of the robot. One proximity sensor is also used to pick up the object handle of the snow block in front of it.

Forward Vision Sensing Unit: This unit contains three vision sensors with one forward facing sensor acting in perspective style to pick up obstruction colours, and two in orthogonal style looking down for redundancy in tarmac detection.

Auger and Impeller Control Unit: This unit contains two revolute joints that are connected to the Auger and the Impeller mechanisms of the snow blower robot. The Auger unit picks up loose snow from the pavement and directs it to the impeller which will blow this loose snow through the chute into the direction that the cap is facing. Although due to time constraints, this unit was simulated with a proximity sensor to delete the snow object within the remote API.

Chute/Cap direction Activation Unit: This unit controls the direction that the Chute and Cap will direct the picked up loose snow to. Since auger and impeller was not implemented fully, this unit also was not. However, direction of movement parameters are implemented in the system such that in the future, the cap can be rotated to the direction of the snow instead of the direction of cleared tarmac.

2.2.2 Remote API Server

Following the layout of the Robot, the Remote API server contains different modules tasked with completing their respective tasks. The following are these modules with their tasks:

Robot Collision Detection Module: Reads the data from the Proximity sensing unit for impending collisions or obstructions that the robot will have to react appropriately to. Computes an appropriate reaction and forwards to the Robot Movement Instructions module.

Path Finding Module: Reads information from the GPS module implemented in the test environment to calculate the movement path.

Snow Texture detection Module: Reads information from the forward facing vision sensor unit for appropriate clues on the snow that is to be cleared. Snow removal instruction module

Snow removal Instruction Module: Awaits information from Snow texture detection module for weather a path of snow can be cleared.

Robot Movement Instruction Module: Awaits instructions for pathfinding and collision detection modules to provide safe movement instructions to the robot.

Chute/Cap direction Control Module: Awaits instructions from the pathfinding modules to find a suitable location to move the collected snow to. This was not fully implemented due to time constraints.

2.3 Robot State

The autonomous Snowblower robot will travel between the following states during normal modes of operation.

- Robot state of the Autonomous snowblower Robot is begun on the Stand-By state. This is where the robot and remote API connects.
- When the initiation to plough is given, Robot will move to the Move straight state. It will stay in this state until it reaches the end of its course.
- If an obstruction is detected in Move Straight state, it will move to Avoid Left or Avoid Right states depending on whether the obstruction is detected to the right of the robot or to the left respectively.
- When the end of the ploughable path is reached, the robot will move to U-turn Left or U-turn right states if the robot's heading is north or south respectively.
- Once turning is complete, the robot will move back to Move Straight state.
- If a curve is detected, the robot will move to the turning state straight from the ploughing straight state.
- If an obstruction is detected, the snow removal process will be halted and the robot will move to the avoid left or avoid right state.
- If the robot is able to maneuver around the obstacle, then it will maneuver around the obstacle.
- Once the entire grid has been ploughed, then the robot will enter complete ploughing state.
- From there it will move on to stand-by state to await further instructions.

Figure 3 below is a diagram of these states that the robot will move between during normal operation.

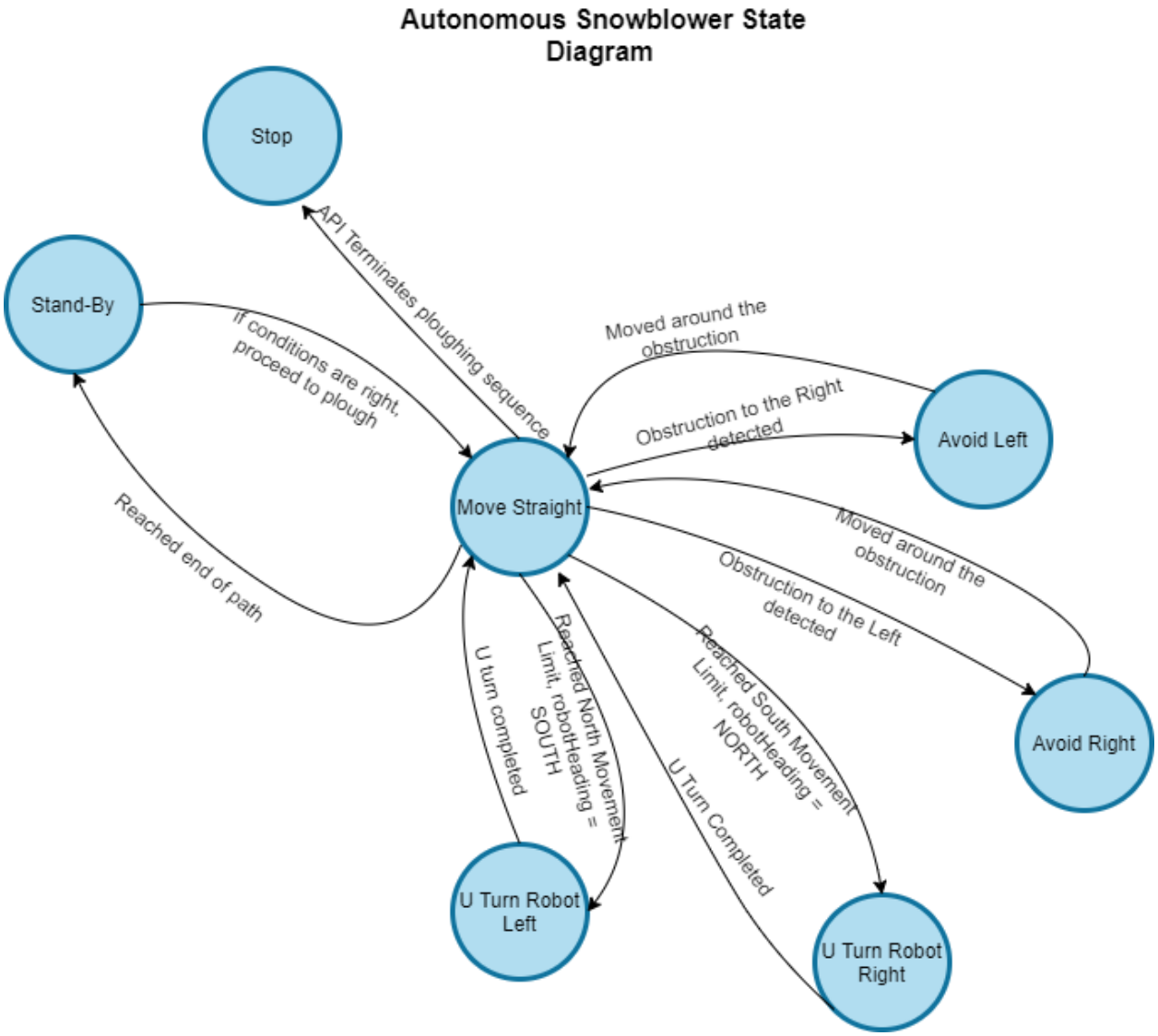

Figure 3: Robot State Diagram of Autonomous Snowblower Robot

2.4 Sequence Diagram

The sequence diagram illustrates the behaviour of the robot on inputs from the user. The robot takes two inputs from the user, a "start" command and "stop" command. The start command sends the robot into a "clearing state". In this state, the robot starts its wheel motors, as it moves forward it reads the proximity data and vision sensor data. It uses the proximity data to detect if there is an object in front. On detecting an object, it reads the vision sensor data to categorise the object as either a snow or an obstacle. If the object is an obstacle it changes its direction away from the object until it has a clear line of sight not obstructed by the obstacle. On the other hand, if the object is snow, it clears it and moves on in its routine. The other user command, "stop command", sends the robot into an "Idle state".On entering into this state, the robot stops all its motors.

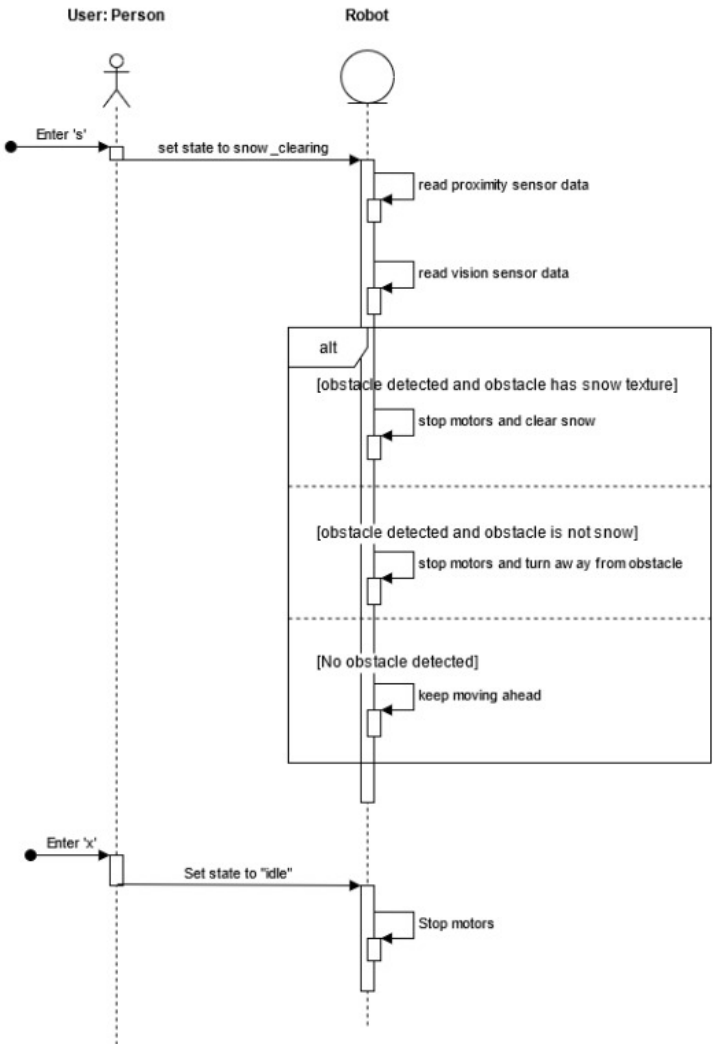

Figure 4: Sequence Diagram of the server

3.0 ACCOMPLISHMENTS

3.1 Server-Client Model Using Python Remote API

Python and multithreading: CoppeliaSIM's remote API feature was utilized during this project to make the server using Python programming language. Coppelia software comes with simulation functions and simulation constants files that can be imported into the remote API and called upon to accomplish tasks in the simulation environment. A delay was introduced with using a python based server due to cross computation on personal machines, but this challenge was overcome by reducing the operational speeds of the robot. We believe using a popular programming language like python is better in the longer scope because it provides features like multiprocessing, and multithreading that a software specific programming language like LUA cannot provide. With multithreading, it provides the server an atmosphere to address more than one Robot client at a time. Furthermore, it provides the option to add a watchdog timer to the server. Figure below shows the use of python and multithreading.

```
class myThread(threading.Thread):
    def __init__(self, threadID, name, clientID, referencePoint, referencePoint2, planeX, planeY):
        threading.Thread.__init__(self)
        self.ThreadID = threadID
        self.name = name
        self.clientID = clientID
        self.referencePoint = referencePoint
        self.referencePoint2 = referencePoint2
        self.planeX = planeX
        self.planeY = planeY

    def run(self):
        print("Starting " + self.name)
        runThread(self.name, self.clientID, self.referencePoint, self.referencePoint2, self.planeX, self.planeY)
        # Insert robot main loop here, and run multiple threads as more robots are introduced
        print("%s Exited" % self.name)

# Thread Runnable
def runThread(threadName, clientID, referencePoint, referencePoint2, planeX, planeY):
    global currentState
    keepRunning = True

    print("running " + threadName)
```

Figure 5: Foundation of Robot client that can be launched as a Thread

State Machine: Basic foundation of the Autonomous snow plough server is a state machine that changes the robot's state according to the real time obstacles and challenges the robot faces during operation. This system was implemented into the server in Python language. During the main loop of the server, it will read all

of the necessary sensors that the robot uses for operation. As real time situations take place, the robot will react to these events and change its state accordingly. This ultimately makes the entire Autonomous robot an event triggered system rather than a time triggered system. Being event triggered makes the robot quick to react to real world events and we believe as this is a real world product that can be exposed to pets, children, and property, it is important for real world events to be monitored. Figure 6 below shows all of the states of the robot, and the initial state of the robot.

```
# Robot State Variables
STOP = 0
STANDBY = 1
MOVESTRAIGHT = 2
UTURNLEFT = 3
UTURNRIGHT = 4
AVOIDLEFT = 5
AVOIDRIGHT = 6

# Robot state variable
currentState = STANDBY
```

Figure 6: All of the states of the robot as well as initial state the robot is configured to launch in

Remote operation: Remote operation ability means that the server and robot can be run on different devices. This provides much higher computing ability during operation. CoppeliaSIM is a resource heavy during operation and as the complexity of the project increased, we experienced more strain on the personal device that was running both the server and the client. Eventhough multi device operation was not something that we attempted during this project, the feature exists in the remote API functions given by CoppeliaSIM to implement this feature. Figure 7 shows how the host name can be configured to another device on the network that can run the simulation.

```
print ('Program started')
sim.simxFinish(-1) # just in case, close all opened connections
clientID=sim.simxStart('127.0.0.1',19999,True,True,5000,3) # Connect to CoppeliaSim
if clientID!=-1:
    print ('Connected to remote API server')
```

Figure 7: simxStart function provides the ability to remote the server to another device

3.2 Differential Drive & Maneuvering

Robot maneuvering is done by two revolute joints on either side of the robot that is individually operated to steer the robot left or right. Each motor's speed can be varied to steer the robot, as well as the speed difference between the two motors can be manipulated to increase or decrease the radius of the arc created by steering the robot. Robot angle alteration when heading in a straight path is provided by the control chart on figure 7 below

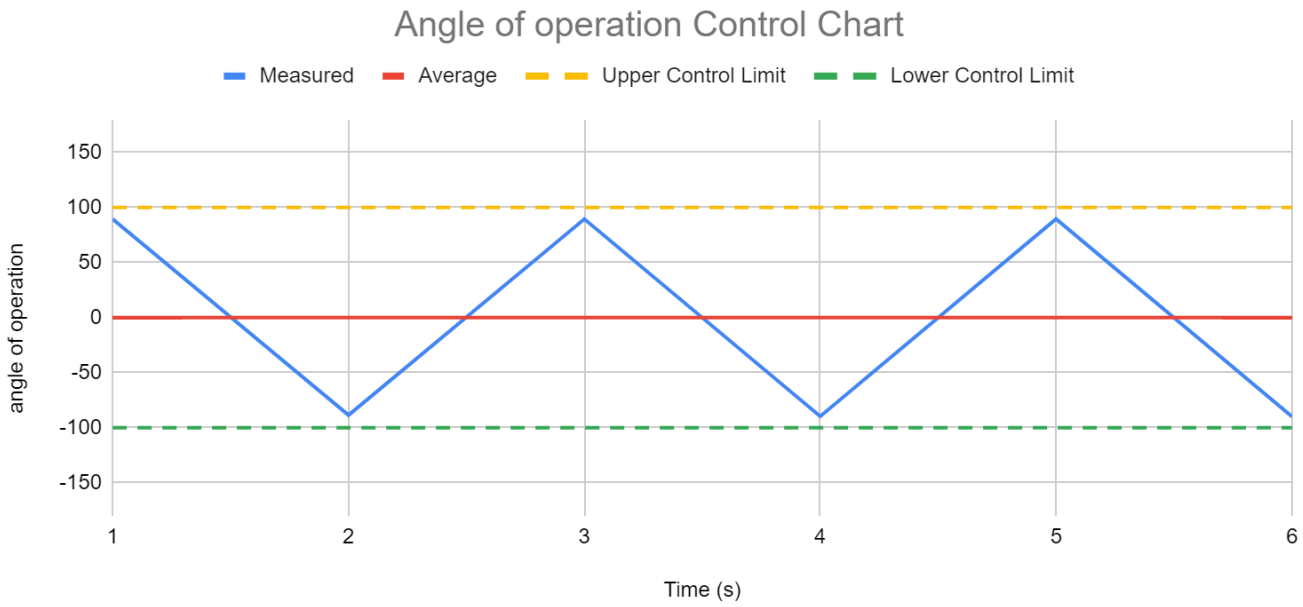

Figure 7: Detected angles every second of operation when in move straight state

3.3 Path defining & Real-time Tracking for Autonomous Navigation

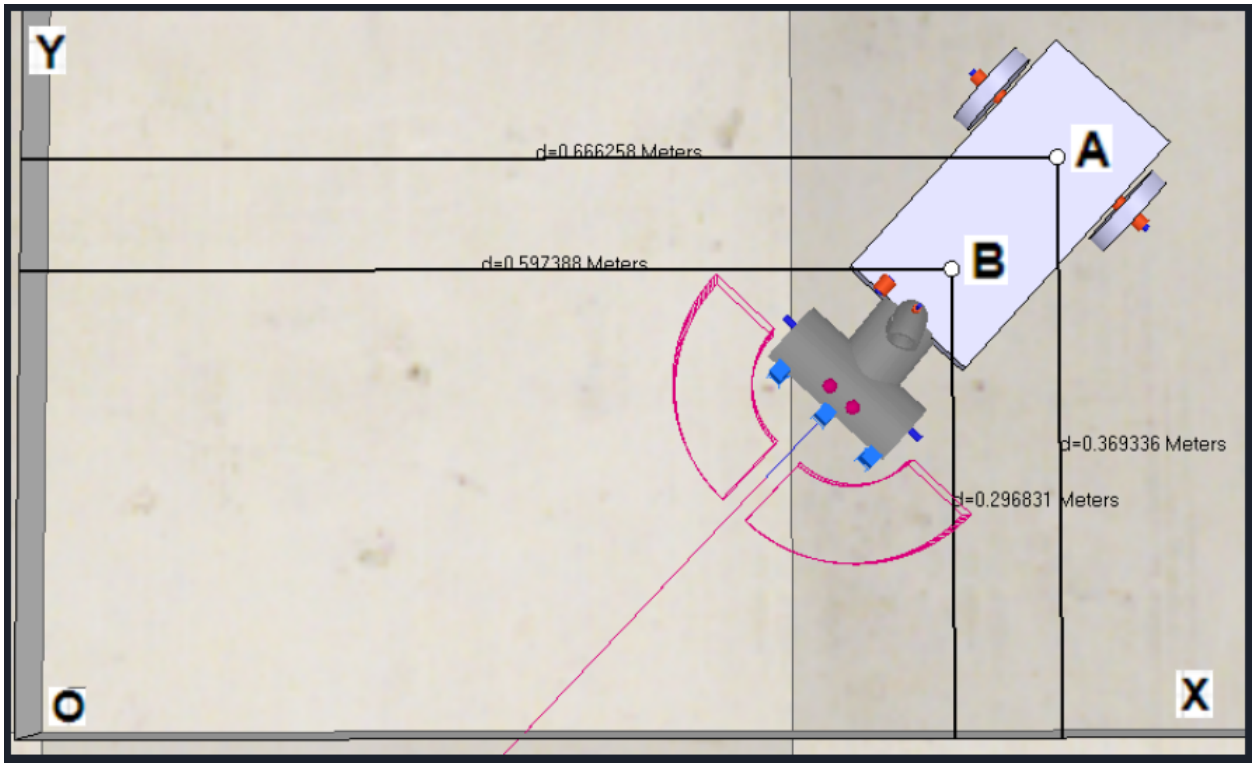

Figure 9: Determining real-time location & orientation of the robot

The team had encountered a major issue in implementing a two dimensional coordinate system as the Coppelia software package did not have a built-in GPS module with it. Therefore the team decided to apply trigonometric and geometric techniques to implement its own coordinate system in order to define the navigational path for the robot. It is crucial to have a well defined navigational path for the robot to traverse and operate within the set perimeters and to prevent it from going out of the operational zone.

As illustrated in the Figure 7 above, the team had established static X & Y coordinate planes on the Scene space and added two dynamic set points on the symmetric line of the robot. Now the instantaneous distances from the X & Y planes to those two set points on the robot are obtained in real-time through Coppelia's built-in distance handler functions.

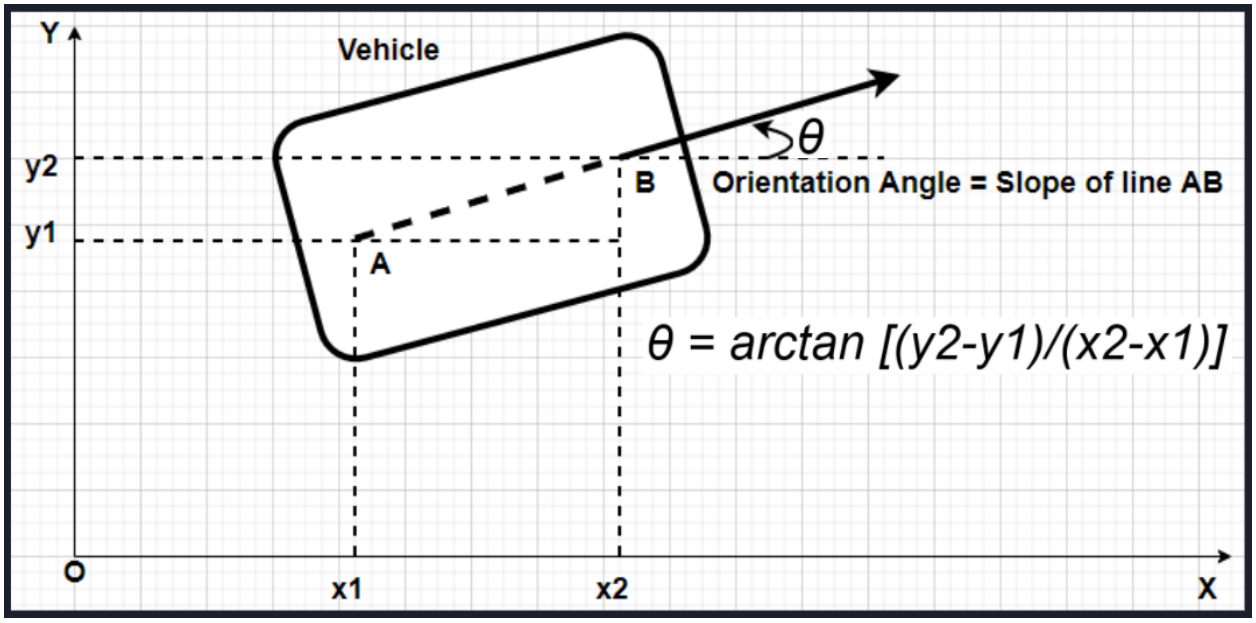

Figure 10: Determining robot's instantaneous orientation angle

As shown in Figure 8 above, the distance readings of both set points are used to calculate the orientation of the robot in real-time. The slope of the symmetric line AB on the robot is interpreted as the robot's orientation angle and is obtained by estimating the acr tangent of the ratio of Y and X coordinate differences. Now this orientation angle can be used in the Python or Lua code to localize and maneuver the robot by adjusting or actuating it's both rear wheel angular velocities iteratively. This task is tested in Lua program and finally translated and implemented in Python API server module.

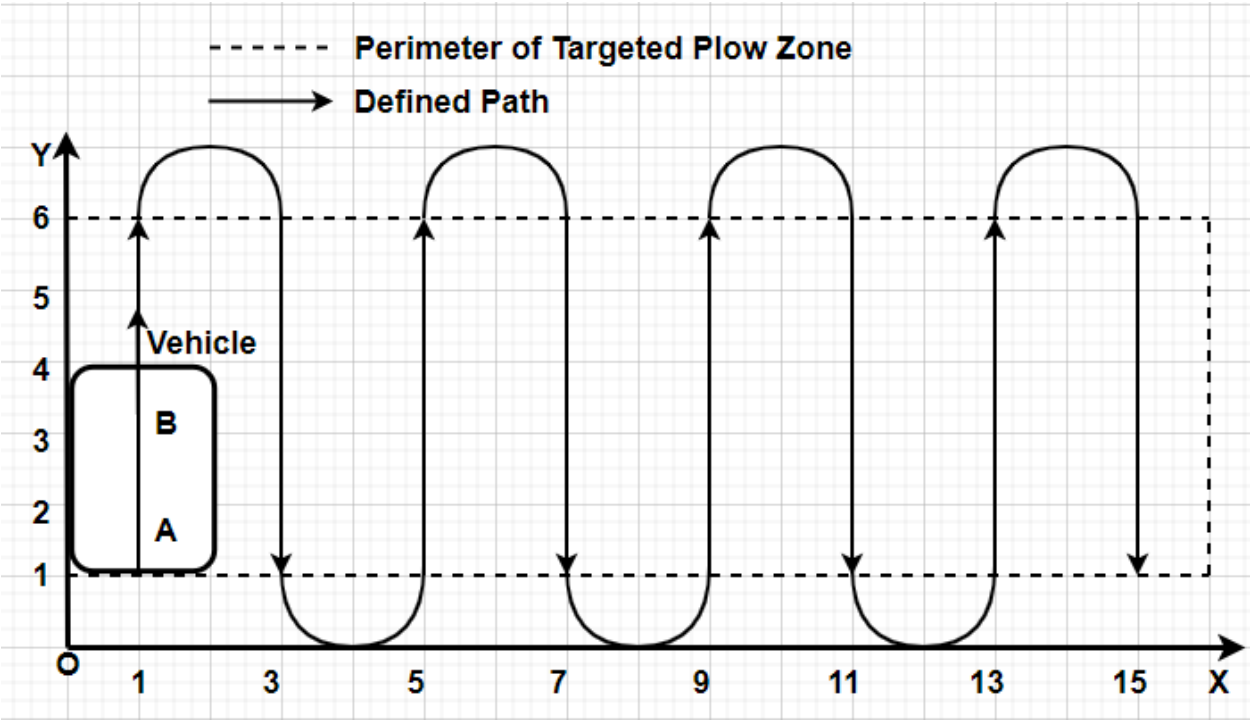

Figure 11: Coordinate plane & path traversal

A fully implemented 2D coordinate plane is shown in Figure 9 above. The width of the snow shovel is 2 units, therefore the robot will have to ensure that each of its row traversals, X increment value should be at 2 units. At every end of row traversals, the robot is commanded by the remote API server to perform positive and negative 180 degree turns while shifting its X coordinate by robot's width distance precisely so that the snow on the driveway can be ploughed thoroughly without over runnings or skippings.

3.4 Obstacle detection & Collision Avoidance

Obstacle detection of the robot is done cohesively using the forward facing vision sensor and disc shaped wide angle proximity sensors. The proximity sensors will always detect objects in its path, however it will not be triggered to change states into avoiding unless it is detected to be an obstacle by the forward facing vision sensor. When an obstacle is identified and detected, by the left or right facing proximity sensor, the robot will move to Avoid right to Avoid left states to halt snow removal and clear the obstacle.

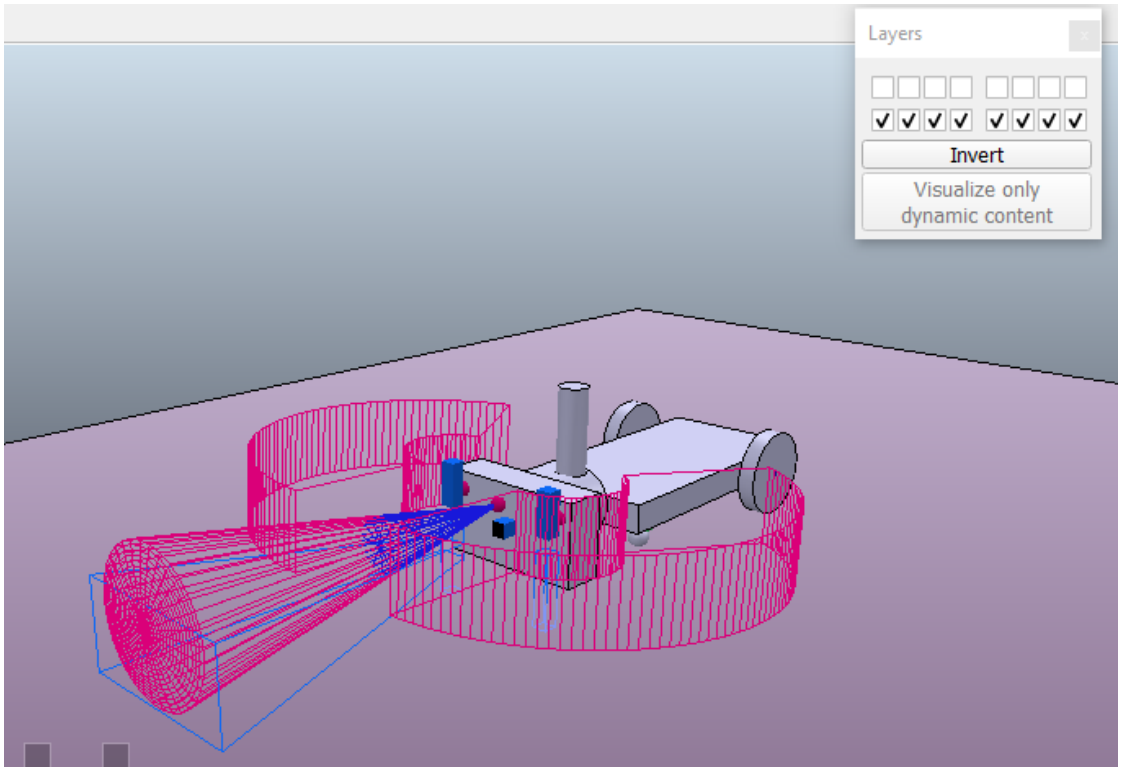

Figure 12: Left and rightward facing proximity sensors for obstacle avoidance

3.5 Snow detection & Plow activation

Snow Detection: Snow detection is done by the forward facing vision sensor. The picture received from the vision sensor's red blue and green components are extracted and an average value is calculated. This average value will be used as the average colour component of the received picture. This value is compared with a threshold value to determine whether the object in front of the vision sensor is snow or an obstacle.

Plow Activation: Auger and impeller activation was implemented in the final project mostly due to time constraints. However, the removal of the snow was simulated using a proximity sensor that picked up the object handle of whatever object was in front of the robot, and when it is detected to be snow, then that snow object's handle is returned, and the server will remove the snow block. In a real life scenario, this can be replaced by a signal to activate the snow auger. Figure 13 below shows the simulation environment that the robot proceeds to detect and plow snow in.

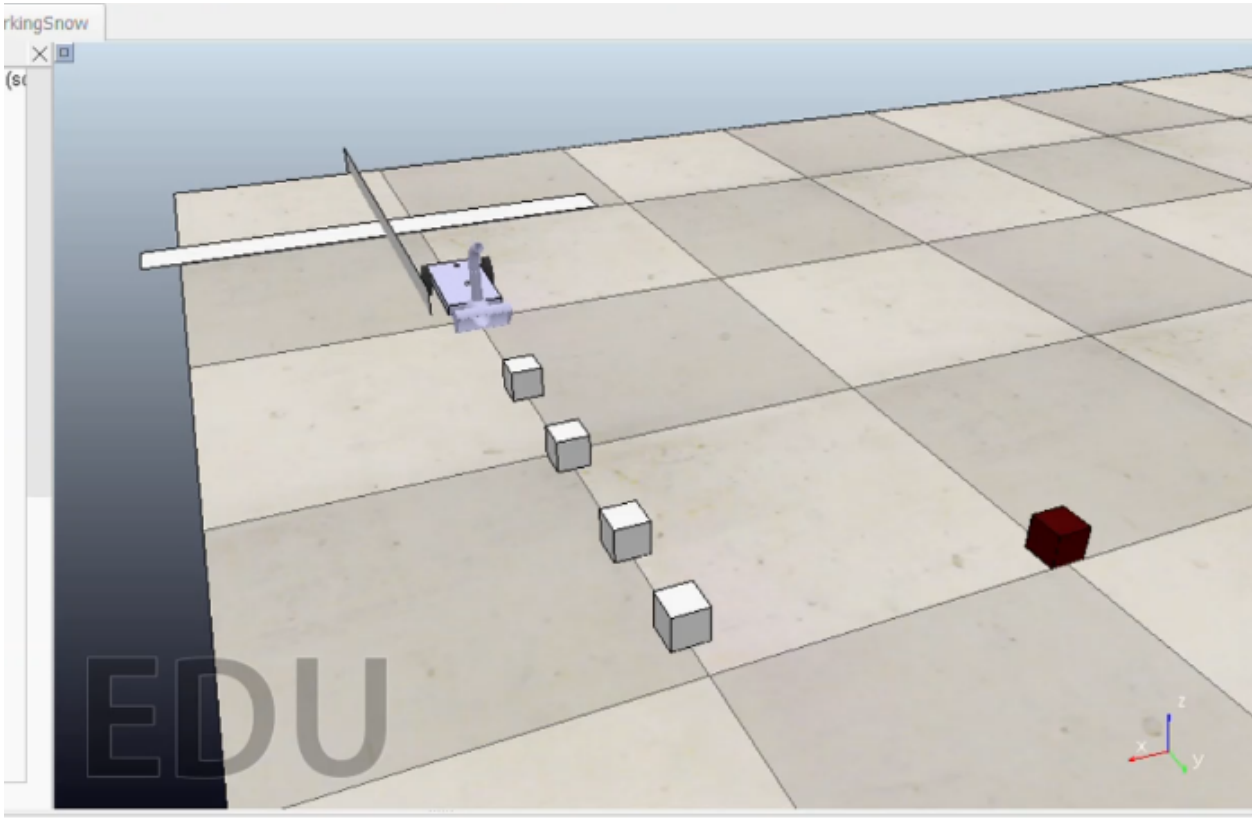

Figure 13: cuboid objects that the robot will recognise as snow blocks

3.6 Environmental & User safety

The team's prime mission has always been based on finding an engineering solution for the above stated problem to save human lives in a safe manner. There are several safety mechanisms included in the software design such as sensor redundancy, failsafe mode and manual override features. If the software prototype is built as a hardware mechanism in real world scenario, it is also possible that the final design could be fully powered and operated by an electric power bank. Electrically powered robots could be considered as environmentally friendly designs as they do not emit harmful emissions to the environment unlike the robots powered by fossil fuels.

4.0 TEAM MEMBER RESPONSIBILITIES

4.1 Project Accomplishments Responsibility Chart

Task/ Responsibility	Responsibility of	Approved by
CAD designing robot Components	Shaviyo Marasinghe	Chukwuka ihedimbu
Programming Move Straight, Stop, Standby, U Turn left and U Turn right states in Python remote API	Shaviyo Marasinghe	Chukwuka Ihedimbu
Programming avoid left and avoid Right states in Python remote API	Shan Rameshkanna	Shaviyo Marasinghe
Programming Maneuverability in Remote API	Shaviyo Marasinghe	Chukwuka Ihedimbu
Draft Sketch & Design assist	Shan Rameshkanna	Shaviyo Marasinghe
Path Finding & Navigation	Shan Rameshkanna	Shaviyo Marasinghe
Sensor Redundancy for Enhanced Safety	Shan Rameshkanna	Chukwuka Ihedimbu
Obstacle detection and Snow detection	Chukwuka Ihedimbu	Shan Rameshkanna

Table 1: Project responsibilities distribution within team

4.2 Project Final Report Responsibility Chart

Task/ Responsibility	Responsibility of
1.0 Introduction	Shaviyo Marasinghe
2.1 Overall Objective	Shan Rameshkanna
2.2 Robot Architecture	Shaviyo Marasinghe
2.3 Robot State	Shaviyo Marasinghe
2.4 Robot Sequence	Chukwuka Ihedimbu
3.1 Server Client Model	Shaviyo Marasinghe
3.2 Differential Drive and maneuvering	Shaviyo Marasinghe
3.3 Path Finding	Shan Rameshkanna
3.4 Obstacle detection	Shaviyo Marasinghe
3.5 Snow detection	Shaviyo Marasinghe
3.6 Environmental & user safety	Shan Rameshkanna
5.0 Conclusion	Shan Rameshkanna

Table 2: Report responsibilities distribution within team

5.0 Conclusion

The team has successfully designed and tested the software prototype of an autonomous snowplow. Upon approval of this software prototype, this system can be manufactured as a hardware prototype in large numbers with affordable price so this design could save hundreds of lives every winter in North America in the near future. The team strongly believes that the operational functionalities of the end model will satisfy customers' requirements that were outlined in the project proposal. Furthermore, the design is ultimately intended to operate as an environmentally friendly mechanism so there is no harm to the ecosystems during the manufacturing or operational states of theis robot.