Final Report

Autonomous Snow Plow

SYSC 4805 A - L3
Group 14 - **Celeste**

Faculty of Engineering and Design
Carleton University

Dina Younes 101231907
George Papoutsis 101235880
Jennifer Ogidi-Gbegbaje 101209061
Kyle Mathias 101266139

Professor: Dr. Abdullah Kadri

Date submitted: 04/12/2025

Table of Contents

1.0 Project Charter

1.1 Overall Objectives

The intent of this project is to design and implement an autonomous snowplow that can efficiently clear a designated area enclosed by a black path while operating fully without human intervention. The robot must detect and avoid both fixed and moving obstacles, including other robots, navigate the area precisely without going beyond the set perimeter, and push artificial snow, represented by 20mm wooden cubes, outside the highlighted perimeter in a controlled manner. The plow mechanism is designed to extend in width and length within the boundaries of 50mm and 30mm, respectively, enabling controlled movement of snow cubes based on the robot's position. The robot plow is designed to be light weight to prevent overloading and maintain optimal performance within the robot's size constraints. The robot must start its operation from a corner inside the perimeter, aligned with one edge, and complete the task within a maximum of five minutes while maintaining a safe speed not exceeding 30cm/s. By integrating reliable obstacle detection, autonomous navigation, and an effective snow-clearing mechanism, the project demonstrates the development of a reliable, intelligent system capable of completing its task efficiently, and fully autonomously, minimizing errors or penalties, and meeting all the specified requirements, measurements, and operational constraints.

1.2 Overall Deliverables

This project has the following deliverables:

1.2.1 Project Proposal Report

A report stating the group's proposed solution to the problem at hand including but not limited to a plan for testing and validation, a proposed timeline for meeting all deliverable deadlines, a cost analysis, and a list of activities required to complete all deliverables.

1.2.2 Project Progress Report

A report stating the group's progress midway through the semester. This report also states what still needs to be done for the project, and retrospective of what went well or wrong to this point in the project.

1.2.3 Project Final Report

A report showcasing the group's work at the end of the semester. This report contains content from both the project proposal, and the progress report. Additionally, a retrospective of what went wrong, what went well, and what could be improved upon is included in this report.

1.2.4 Project Presentation

A live demonstration of the group's robot in action. This deliverable showcases everything that the group has worked on over the course of the semester. During the lab time, the group will demonstrate what their robot is capable of by having the robot push wooden blocks out of a fixed space.

1.3 Revisions to Project Proposal

Table 1: Revised Table Plan of Project Proposal

Original Proposal	**Final Version**	**Reason for Change**
Wheel Encoders, IMU Sensors, Ultrasonic Sensors, Line Follower Sensors	Ultrasonic Sensors and Line Follower Sensors	The robot was already operating below the speed of 30cm/sec, so implementing additional speed-control mechanisms would have added unnecessary complexity.
3D printing plow design	Use of cardboard	It was also unnecessary to make a 3D printed plow, as a cardboard plow provided the same functionality for testing and met project requirements.

Design Finalization

- The autonomous snowplow robot was constructed with all hardware components properly wired and integrated with the software system. The plow mechanism was mounted at the front of the chassis to enable snow-clearing operation.

Updated Diagrams

- Hardware Architecture section 2.1.1
- Interaction Model section 2.1.3
- Work Breakdown Structure (WBS) diagram section 3.2

1.4 Team Member Contributions

Table 2: Team Member Contributions

Team Member Names	**Member Contributions**
Jennifer	Hardware setup, project document review, some wiring of the system.
Dina	Wiring of the system, hardware set up, software testing, plow design

George	Plow design, wiring of the system, and software development for the robot (this software was later integrated with the RTOS)
Kyle	RTOS integration, sensor testing, integration testing

2.0 System Design and Architecture

2.1 Overall Architecture

2.1.1 Hardware Architecture

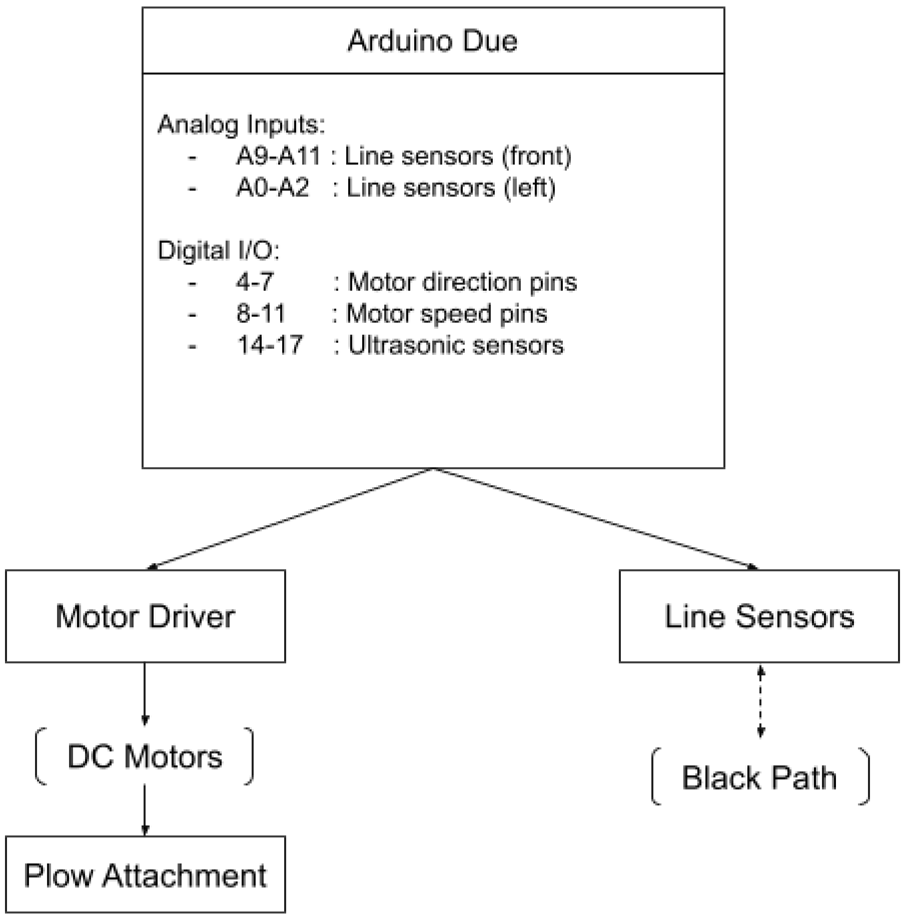

Arduino Due
Analog Inputs:
- A9-A11 : Line sensors (front)
- A0-A2 : Line sensors (left)
Digital I/O:
- 4-7 : Motor direction pins
- 8-11 : Motor speed pins
- 14-17 : Ultrasonic sensors
Motor Driver
Line Sensors
DC Motors
Black Path
Plow Attachment

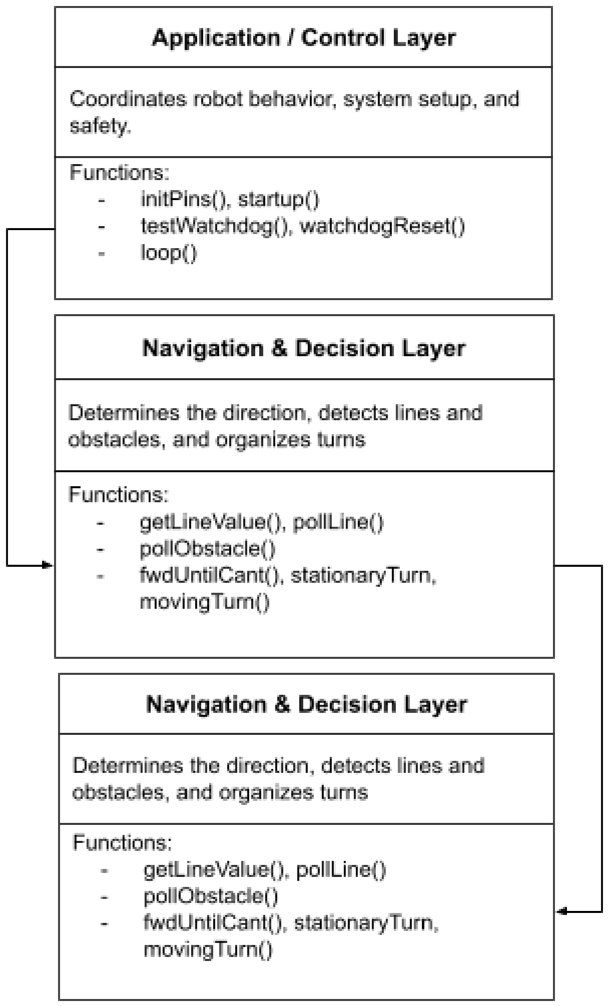
Figure 1: Hardware (Structural) Architecture of the Autonomous Snowplow Robot

Table 3: Components Description and Functionality

Component	**Function**	**Connected To**
Arduino Due	Central controller - runs the logic in .ino files	All sensors and actuators
Line Follower Sensors	Detects black path for navigation	Analog pins A9-A11 (front), A0-A2 (left)

Ultrasonic Sensors	Detects obstacles	Trigger/Echo pins: 14-17
Motor Driver	Drives 4 DC motors	Control pins 4-11
Plow Blade	Attached to motor to push snow cubes	Front of chassis
Power Source (Battery)	Powers Arduino + motors	Power input and motor driver

2.1.2 Software Architecture



Application / Control Layer
Coordinates robot behavior, system setup, and safety.
Functions:
- initPins(), startup()
- testWatchdog(), watchdogReset()
- loop()
Navigation & Decision Layer
Determines the direction, detects lines and obstacles, and organizes turns
Functions:
- getLineValue(), pollLine()
- pollObstacle()
- fwdUntilCant(), stationaryTurn, movingTurn()
Navigation & Decision Layer
Determines the direction, detects lines and obstacles, and organizes turns
Functions:
- getLineValue(), pollLine()
- pollObstacle()
- fwdUntilCant(), stationaryTurn, movingTurn()

Figure 2: Software (Functional) Architecture of the Autonomous Snowplow Robot

Table 4: Components Description and Functionality

Function Category	**Key Functions**	**Purpose**
Initialization	initPins(), startup(), waitForSerial()	Sets up all pins, serial communication, and calibration of components before operation.
Line Detection	getLineValue(), pollLine()	Aids the detection of the black tape using line sensors to

		ensure it stays within the enclosed path.
Obstacle Detection	pollObstacle()	Uses ultrasonic sensors to detect moving objects and obstacles.
Motion Control	setSpeed(), setDirection(), moveForward(), moveReverse(), stationaryTurn(), movingTurn()	Coordinates the movement of the DC motors for precise movement and turning.
Calibration	getThreshold()	Ensures line sensing adapts to lighting or surface differences.
Watchdog	testWatchdog(), watchdogReset()	Prevents software hangs
Decision Logic	fwduntilCant(), stationatyTurn(), movingTurn()	Determines when to stop, turn, reverse based on the code and gathered sensor data
Main Execution Loop	loop()	Integrates all behaviors: forward movement, obstacle detection, and turns.

2.1.3 Interaction Model

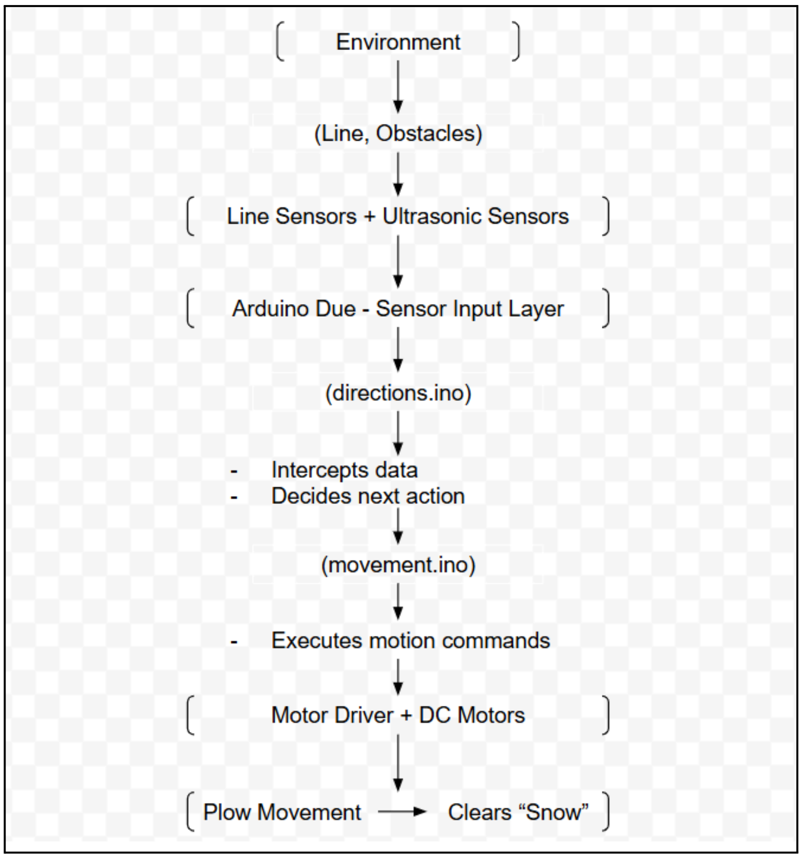


Figure 3: Behavioral (Interaction) Architecture of the Autonomous Snowplow Robot

2.2 System Design

2.2.1 State Chart

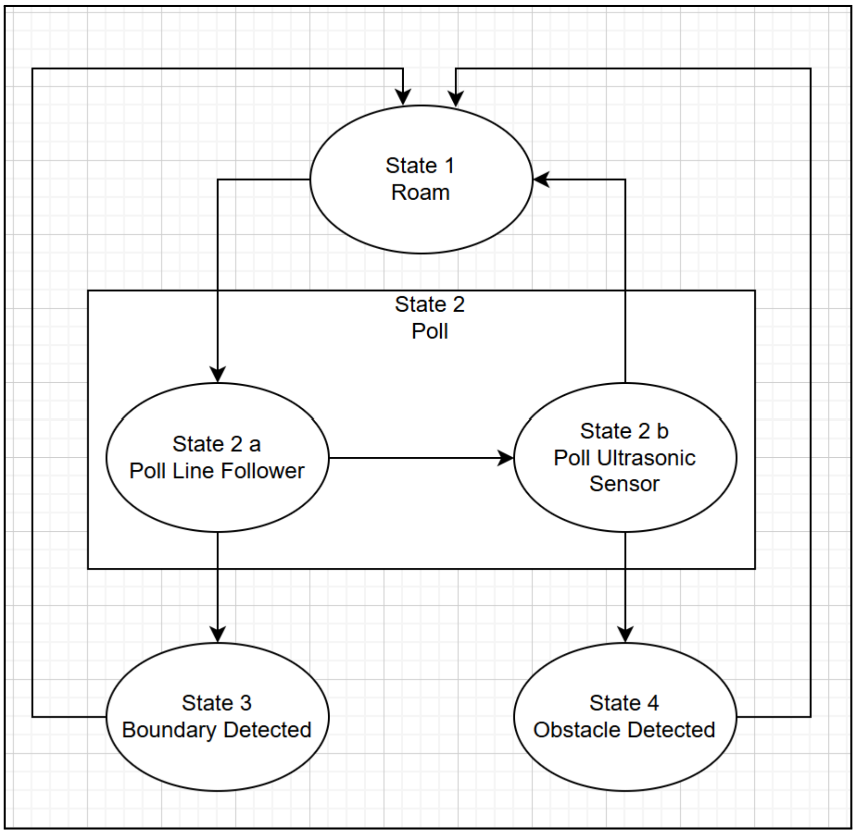


Figure 4: State chart diagram showing the different states the system enters

State 1: Roam
This state serves as the main state of the system. In this state, the robot moves forward at a predefined speed. Periodically, the system exists in this state, and enters **State 2: Poll**.

State 2: Poll
This state acts as a "general" state which includes 2 sub states. These 2 substates are for polling the different sensors currently on the robot.

State 2 a: Poll Line Follower
This state is where the system gets the current value of the Line Follower component. If the line follower returns a value indicating that the robot has reached one of the black lines at the edge of the area, the system enters **State 3: Boundary Detected.** If no line is detected, the system moves to **State 2 b: Poll Ultrasonic Sensor.**

State 2 b: Poll Ultrasonic Sensor
In this state, the system checks the current value of the Ultrasonic Sensor component. If the Ultrasonic Sensors return a value indicating that the robot is on course to hit an object, the

system will enter **State 4: Obstacle Detected.** If the Ultrasonic Sensors do not report an obstacle, the system will return to **State 1: Roam.**

State 3: Boundary Detected
In this state, the system has detected one of the black lines at the edge of the area. In this scenario, the robot will move back a small, predefined distance, and then turn a random angle between 45, and 135 degrees. This randomness has been implemented with the intention of allowing the robot to move towards the middle of the area instead of only traversing the perimeter of the area. After this procedure has completed, the system will return to **State 1: Roam.**

State 4: Obstacle Detected
In this state, the system has detected an obstacle. In this scenario, the robot does one of the following:

1. If the right Ultrasonic Sensor detected the obstacle, the robot will reverse a small, predefined distance, and then turn to the left 45 degrees
2. If the left Ultrasonic Sensor detected the obstacle, the robot will reverse a small, predefined distance, and then turn to the right 45 degrees

It is also important to note that if both Ultrasonic Sensors detect the obstacle, the system will still reverse a small, predefined distance, and then it defaults to turning right.
After this scenario has completed, the system will return to **State 1: Roam.**

2.2.2 Sequence Diagram

The following sequence diagram depicts the component-component interactions involved in the previously shown state chart.

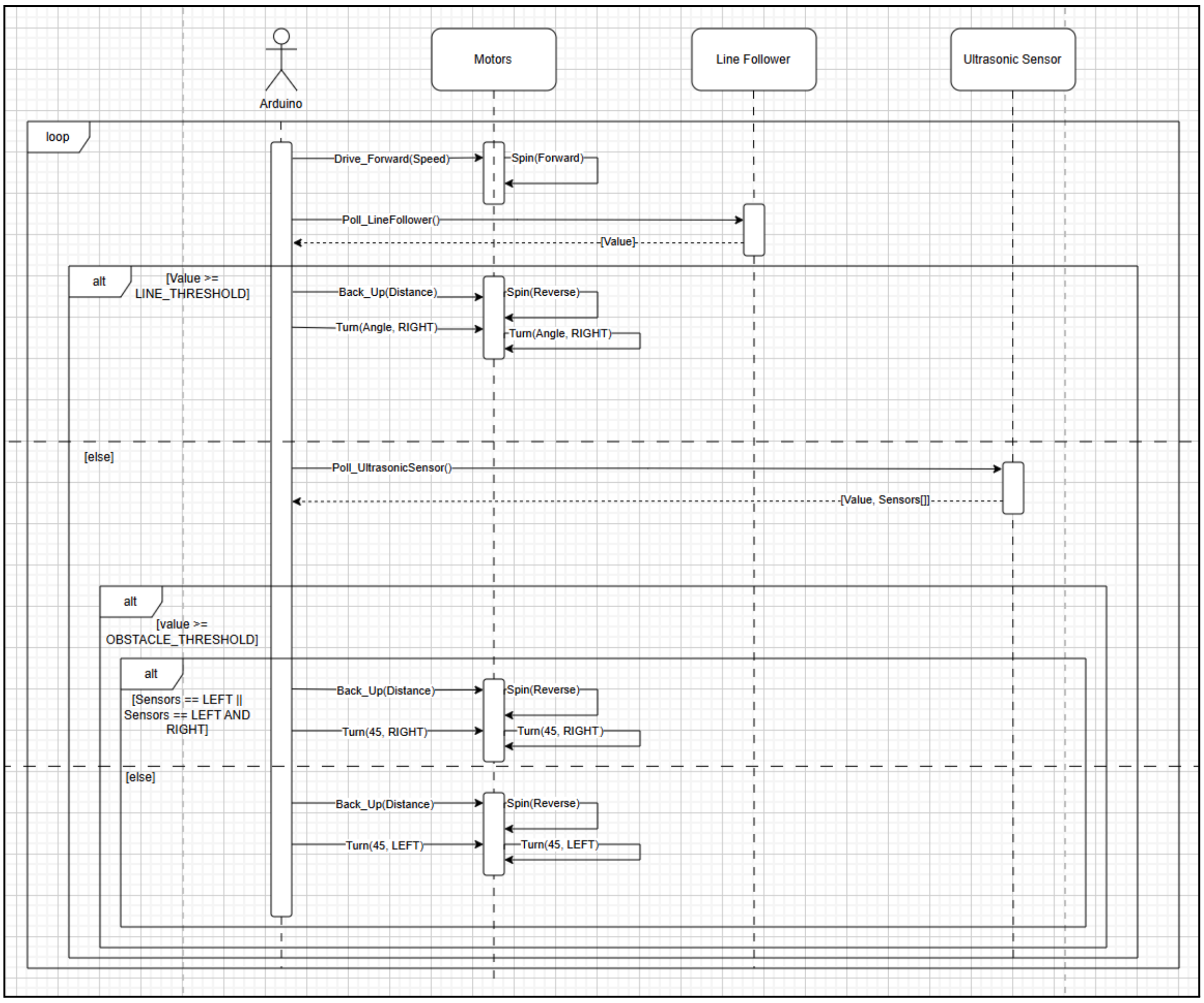


Figure 5: Sequence diagram illustrating communication between components

3.0 Scope

3.1 Project Requirements

3.1.1 Functional Requirements

R1. The robot should move through paths without hitting obstacles.
R2. The robot plow should push snow cubes (20mm wooden cubes) when detected.
R3. The robot shall change direction in the occurrence of an obstacle without leaving the black perimeter.

3.1.2 Non-functional Requirements

Performance
R4. The robot shall clear the snow cubes within 5 minutes.
R5. The robot shall operate at a speed not exceeding 30cm/s and shall accelerate gradually.

Interoperability
R6. The plow should be able to extend up to a maximum of 50mm in width and 30mm in length depending on the robot's position.

Capacity
R7. The robot plow should be able to handle multiple snow cubes of side length 20mm simultaneously.

3.1.3 Constraints

R8. The robot shall only start from a corner within the designated perimeter, aligned with one edge.
R9. The robot shall remain within the black path boundary throughout operation.
R10. The robot shall operate independently without any human interference or adjustments.

3.2 Project Deliverables

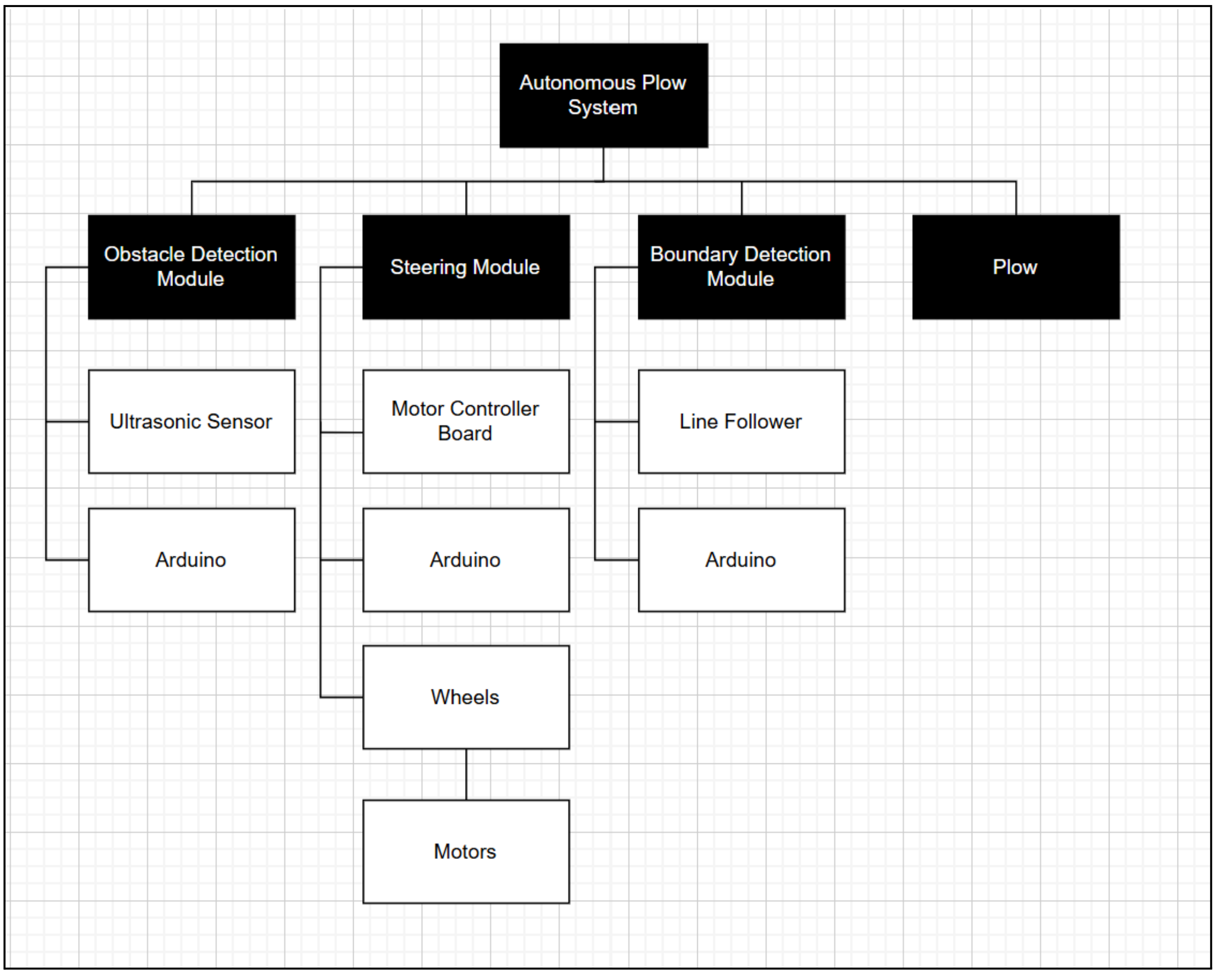


Figure 6: WBS breaking down all deliverables of the system.

3.3 Testing Plan

3.3.1 Unit Testing

Table 5: Unit testing plan

UT ID	**Test Description**	**Component(s) under test**	**Test Method**	**Expected Result**	**Requirement(s) covered**
UT-01	Boundary detection	Line Follower sensor/navigation system	Move robot along edges and attempt to cross perimeter	The robot stays within 5cm of the boundary; penalties applied if exceeded.	R3, R9
UT-02	Obstacle	Ultrasonic	Placed fixed	The robot	R1, R3

	avoidance	sensors	and moving obstacles in path	detects and avoids collisions; penalties are applied based on impact severity.	
UT-03	Snow plow functionality	Plow mechanism/ motor controllers	Place wooden snow cubes in path and activate plow	Snow cubes are pushed outside the perimeter without exceeding maximum allowed size.	R2, R6, R7
UT-04	Speed control	Motor controllers	Command robot to move at max speed	Robot speed remains <=30cm/s; penalty if exceeded.	R5
UT-05	Start position alignment	Navigation system	Place robot in starting corner	The robot starts aligned with the perimeter edge and clears snow cubes without violating project specifications.	R8
UT-06	Maximum operation time	Full robot system	Measure total snow clearing time	Task is completed within 5 minutes	R4
UT-07	Robot size compliance	Physical robot chassis/plow	Measure robot and plow dimensions	Robot and plow dimensions remain within allowed limits.	R6
UT-08	Plow control precision	Plow mechanism	Push wooden snow cubes to a target area	Snow cubes are pushed to the intended area without scattering outside target range.	R2, R7

UT-09	Handling moving obstacles	Sensors (ultrasonic), navigation system	Introduce moving obstacles during operation	The robot detects and avoids moving obstacles without collisions.	R1, R3
UT-10	Sensor reliability	Line Follower sensor	Vary lighting and add reflective surfaces	Sensors detect obstacles, boundaries, and wooden snow cubes accurately	R1, R2, R3
UT-11	Autonomy	Full robot system	Run the robot for 10-20 minutes with no manual resets	No human intervention required	R10

3.3.1.1 Obstacle Detection Module

A potential unit test for the obstacle detection module is as follows:

- Place an object at the limit of detection for the sensors (distance collected from guided lab)
- Upload the script for detecting object to the Arduino
- Ensure that the object detection module is able to detect the object whenever it is at any distance equal to or closer to the maximum distance
- Ensure that the object detection module is not able to detect the object whenever it is at any distance further than the maximum distance.

The robot would pass this test if the sensor only detects when it should be detecting an obstacle. If the sensor detects an obstacle when there is no obstacle present, or if it does not detect an obstacle when there is one present, it would fail.

3.3.1.2 Steering Module

A potential unit test for the steering module is as follows:

Moving forwards/ backwards:

- Select a pre-determined distance
- Upload the script that should move this distance
- Observe if the robot is able to move this distance both going forwards, and backwards

If the robot is able to accurately move the predetermined distance (within 5 mm) in both directions, the robot passes this test case. If it cannot, the robot fails this test case.

Turning to a specific angle:

- Set up a track with walls that the robot is supposed to not knock over if it makes the turn successfully
- Upload the script that should make the appropriate turn
- Observe if the robot is able to make the turn without knocking over any of the barriers

If the robot is able to make the turn with minimal change in the position of the barriers, the robot passes this test case, if there is significant change in the position of the barriers (i.e., the robot knocked over the barriers), the robot fails this test case.
Turn on the spot:

- Place the robot in a small enclosed space with not much space around the robot
- Upload the script that should allow the robot to turn on the spot
- Observe if the robot is able to turn on the spot in the small space

If the robot is able to complete the turn in the small space without getting stuck, the robot passes this test case. If the robot cannot do this, the robot fails this test case.

3.3.1.3 Boundary Detection Module

A potential unit test for the boundary detection module is as follows:
Detecting a boundary:

- Set up a blank, white surface
- Run the boundary detection module over the surface and ensure that it does not detect a boundary. Adjust thresholds if needed
- Change level of lighting on the surface and repeat the test

If the robot detects no black line when there is a black line present, the robot fails this test case. If the robot successfully detects that there is a black line present, the robot passes this test case.

Not detecting a boundary:

- Set up a white surface with black lines on it
- Run the boundary detection module over the surface and ensure that it is able to detect a boundary only when it is over one of the black lines. Adjust thresholds if needed
- Adjust lighting on the surface and repeat the test

If the robot detects that there is a black line when there is no black line present, the robot fails this test case. If the robot successfully detects no black line when there is no black line present, the robot passes this test case.

3.3.1.4 Plow

The plow was designed with cardboard and electrical tape, with a flat edge preventing the robot from accidentally going over the blocks, which would in turn have triggered the line follower sensors into thinking a line was present. It also has two angled edges on the sides in order to prevent the cubes from rolling off the sides, thereby allowing the plow to effectively capture cubes and push them outside, before reversing to release them.

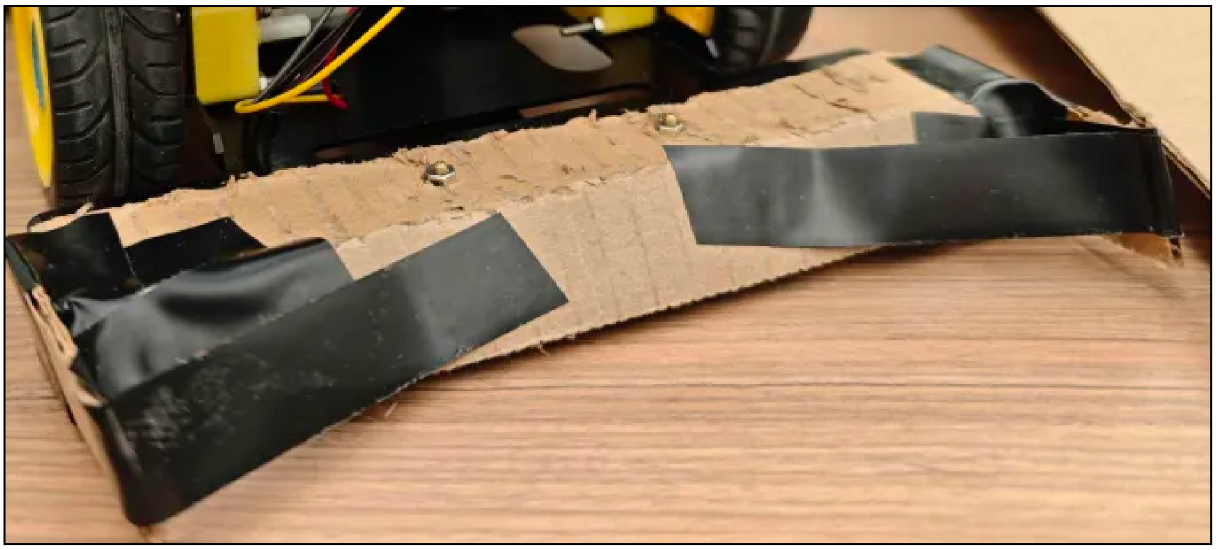
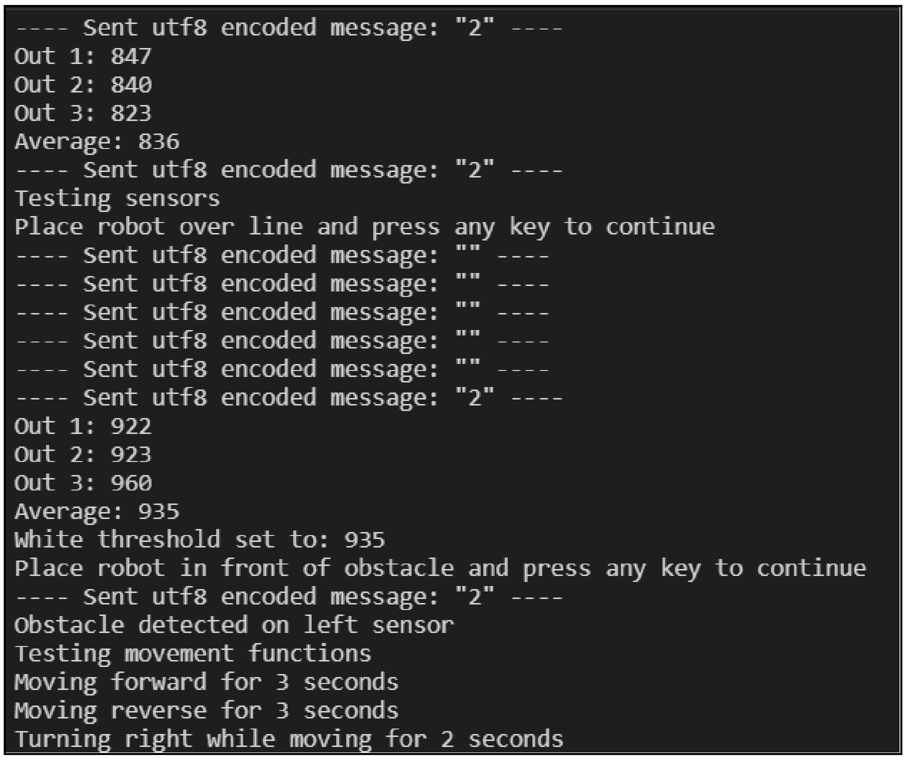


Figure 7: Snowplow with angled edges

3.3.1.5 Unit Test Results

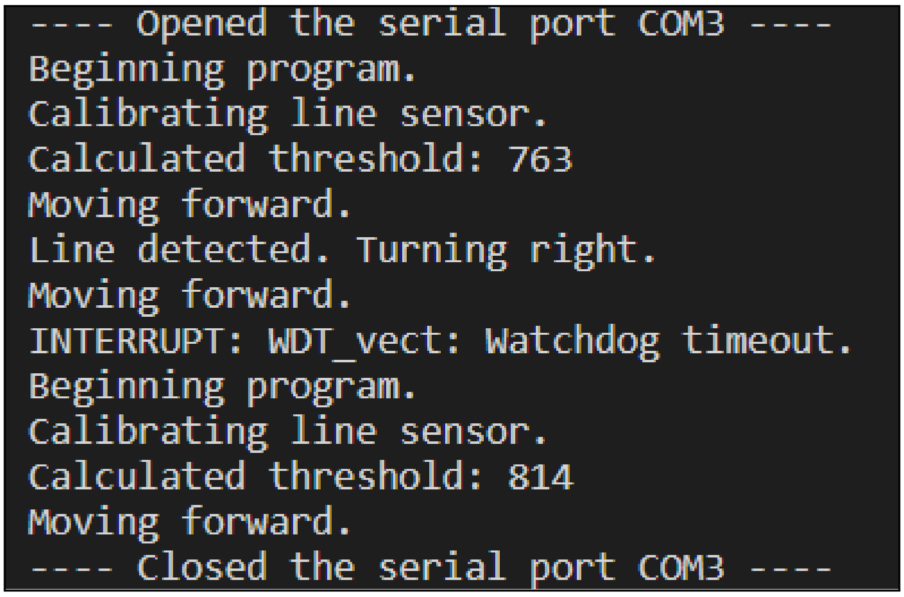


```
---- Sent utf8 encoded message: "2" ----
Out 1: 847
Out 2: 840
Out 3: 823
Average: 836
---- Sent utf8 encoded message: "2" ----
Testing sensors
Place robot over line and press any key to continue
---- Sent utf8 encoded message: "" ----
---- Sent utf8 encoded message: "" ----
---- Sent utf8 encoded message: "" ----
---- Sent utf8 encoded message: "" ----
---- Sent utf8 encoded message: "" ----
---- Sent utf8 encoded message: "2" ----
Out 1: 922
Out 2: 923
Out 3: 960
Average: 935
White threshold set to: 935
Place robot in front of obstacle and press any key to continue
---- Sent utf8 encoded message: "2" ----
Obstacle detected on left sensor
Testing movement functions
Moving forward for 3 seconds
Moving reverse for 3 seconds
Turning right while moving for 2 seconds
```

Figure 8: Summary of Unit test results

The unit tests cover all the functionality of the robot, verifying that the system works correctly and that all components are integrated and validated.

3.3.1.6 Watchdog Timer Demonstration



```
---- Opened the serial port COM3 ----
Beginning program.
Calibrating line sensor.
Calculated threshold: 763
Moving forward.
Line detected. Turning right.
Moving forward.
INTERRUPT: WDT_vect: Watchdog timeout.
Beginning program.
Calibrating line sensor.
Calculated threshold: 814
Moving forward.
---- Closed the serial port COM3 ----
```

Figure 9: Watchdog timer integration into program

The watchdog timer was enabled in the microcontroller to reset the program during a fault such as an infinite loop, which was verified with a while (true) loop and an interrupt from the WDT_vect ISR.

3.3.2 Integration Testing

For integration testing, the team proposes connecting all previously mentioned modules to the robot, and ensuring that all previously mentioned functionality is still working as intended. Additionally, the team proposes to run all tests together and ensure that the robot is still able to function as expected.

Below is a table with various combination of previously mentioned Unit tests:

Table 6: Integration testing plan

IT ID	Test Description	Component(s) under test	Test Method	Expected Result	Requirement(s) covered
IT-01	Boundary detection and speed control	Line Follower sensor/navigation system/ Motor controllers	Command the robot to move at its maximum allowed speed (<=30cm/s)	The robot stays within 5cm of the boundary; penalties applied if exceeded. Robot speed remains <=30cm/s; penalty if exceeded.	R3, R5, R9
IT-02	Obstacle	Ultrasonic	Placed fixed	The robot detects	R1, R3, R5

	avoidance and Speed control	sensors	and moving obstacles in path, and commanded the robot to move at max speed.	obstacles and safely reroutes before collision; penalties are applied based on impact severity. Robot speed remains <=30cm/s; penalty if exceeded.	
IT-03	Snow plow functionalit y and Speed control	Plow mechanism/ motor controllers/Motor controllers	Place wooden snow cubes in the path and activate the plow. Additionally, command the robot to move at max speed.	Snow cubes are pushed outside the perimeter without exceeding maximum allowed size. Robot speed remains <=30cm/s; penalty if exceeded.	R2, R5, R6, R7
IT-04	Plow control precision and speed control	Plow mechanism, Motor controllers	Push wooden snow cubes to a target area and command the robot to move at max speed	Snow cubes are pushed to the intended area without scattering outside target range. Robot speed remains <=30cm/s; penalty if exceeded.	R2, R5, R7
IT-05	Handling moving obstacles and speed control	Sensors (ultrasonic), navigation system, Motor controllers	Introduce moving obstacles during operation and command the robot to move at max speed	The robot detects and avoids moving obstacles without collisions. Robot speed remains <=30cm/s; penalty if exceeded.	R1, R3, R5
IT-06	Uninterrupt ed autonomous operation	All subsystems	Run the robot autonomously for 5 minutes with random obstacles and wooden snow cubes	The robot completes the task without manual resets	R8, R10

Common System Constraints:

R3 (change in direction upon obstacle), R5 (max speed <= 30cm/s), and R9 (autonomous operation) apply to all motion-based tests as core performance requirements.

3.3.2.1 Results of Integration Testing (Lab 9, and Lab 10)

The team performed integration testing by running our unit test scripts while observing the various components integrated together physically on the robot. For example, to test the integration between robot movement, and line detection, the team ran the unit test for the line follower while having the robot move. The team then observed the values from the unit test script, while also observing that the robot still moved as expected. A similar procedure was followed for the ultrasonic sensors. The main difference between the integration test for the ultrasonic sensors compared to the line follower sensor was the inclusion of moving obstacle testing. Since the ultrasonic sensors are used for obstacle detection, they also had to be used for testing the moving obstacles. This was done by placing the box we were provided at random positions around the robot, and moving it every time the robot detected the obstacle.

The procedure followed for the plow component was slightly different simply because the plow itself is a physical component with no code associated with it. To handle this, the team ran the same unit tests as with the other integration tests, but now had the plow mounted on the front of the robot. The team then ensured using the serial monitor that all system functions were still performing as intended while also observing the robot in real life.

3.3.2.2 Customer testing (Lab 11 demo)

After completing the integration testing in the previous labs, the team was ready for customer testing in the final lab demo. The team considers the customer testing at the lab demo to be a success as we were able to clear quite a few blocks with minimal issues. The main take away for the team from the demo was the obstacle detection. The team observed a few times in our demo that a blind spot in our sensor vision resulted in quite a few collisions with obstacles. These collisions mainly happened in one scenario depicted in Figure 9.

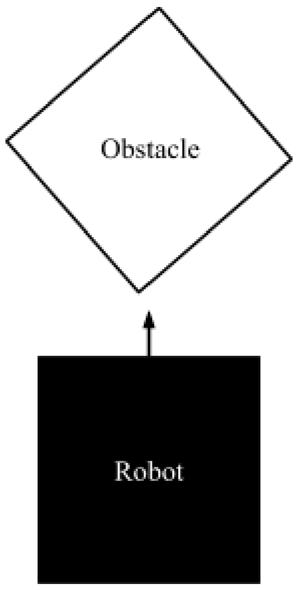


Figure 10: Scenario which caused collisions in the Lab 11 demo

The reason this scenario caused collisions is because the ultrasonic sensors positioned on the corners of the robot were not successful in detecting the obstacle before the robot made contact with the corner of the obstacle. The team had tested in previous labs this scenario and saw that it was able to detect the obstacle, however the size of the obstacle, and the specific angle the robot approached from created a scenario where the robot collided. There are a few ways that the team could mitigate this issue looking back. The first, and probably the best solution would be to add a third ultrasonic sensor in the middle of the robot to remove this blind spot entirely. Other solutions could be to angle the ultrasonic sensors inwards to remove the blind spot in the middle, though this solution may create new blind spots on the corners. The following chart outlines the results of the demo.

Table 7: Demo results

Cubes pushed out	78
Obstacles hit	9
Lines missed	2
Resets	2
Total penalties	13
Total score	65

4.0 Challenges and Limitations

During prototype testing, several performance and hardware limitations were observed.

- **Speed and Coverage Efficiency:** The robot's current movement speed is insufficient for completing the cleaning task within the 5-minute evaluation window. Because it follows no defined navigation pattern, some areas may be left uncleaned or revisited unnecessarily.
- **Sensor Blind Spots:** The obstacle-detection system occasionally fails to register smaller or transparent objects such as water bottles. This results in collisions or the robot driving directly through obstacles it is meant to avoid.
- **Hardware Constraints:** The current power system and wiring introduce reliability issues. The battery pack is difficult to remove and replace, wiring connections are fragile and prone to disconnection, and the low-voltage batteries discharge quickly—limiting operating time and overall consistency.

These issues collectively impact the robot's ability to perform its intended cleaning function efficiently and safely.

4.1 Future Improvements and Mitigation Plan

To address these challenges, the team plans several corrective actions and design optimizations.

- **Speed Optimization and Path Planning:** Implementing a systematic cleaning algorithm (e.g., serpentine or spiral traversal) will ensure consistent area coverage.

The team will also explore higher-speed motor control or lighter chassis components to improve movement efficiency.

- **Sensor Calibration and Placement:** The team will further evaluate whether the observed detection issue is primarily due to very thin obstacles, as initial testing suggests larger objects are reliably detected. Additional testing and sensor positioning adjustments will be performed to confirm performance consistency across expected object sizes.

5.0 Schedule

5.1 Project Activities

The following activities cover all project deliverables, assuming the base robot platform (locomotion and motor control) is already available. Each activity is assigned a unique ID and is linked to the Work Breakdown Structure (WBS).

1. **Snowplow Mechanism**
   A5: Design plow attachment to fit within extension limits (≤50 mm width, ≤30 mm length).
   A6: Fabricate and mount plow onto chassis.
   A7: Test snow displacement efficiency with wooden cubes.

2. **Obstacle Detection and Avoidance**
   A8: Select appropriate sensors (IR/Ultrasonic/LiDAR) for fixed and moving obstacles.
   A9: Develop sensor function handlers (atomic, robust, report errors).
   A10: Unit test obstacle detection under static and dynamic conditions.
   A11: Integrate sensors with motor control for real-time avoidance.

3. **Boundary and Navigation**
   A12: Implement line-following algorithm using black tape boundary sensors.
   A13: Calibrate navigation to start from corner position.
   A14: Test boundary handling, ensuring robot stays inside perimeter.

4. **System Integration**
   A15: Integrate locomotion with boundary sensors (motors).
   A16: Integrate obstacle detection with navigation system.
   A17: Conduct iterative system testing after each integration step.
   A18: Perform full system test (5-minute operation with snow + obstacles).

5. **Project Management and Testing**
   A19: Document weekly testing results and report progress to team/TA.
   A20: Upload code and function handlers to GitHub repository.

A21: Prepare final demo-ready version of snowplow robot.
A22: Conduct trial runs to verify success criteria before Lab 11 demo.

5.2 Schedule Network Diagram

This network connects all activities by logical dependency and lays them on the 6-week window you provided. It follows the course proposal rubric (Schedule → activities, network diagram, Gantt) and the project description's design→integration→demo cadence.

1. **Predecessors already completed (context)**
   Hardware Testing (done in Sept):
   HT1 Distance Sensors (Sep 10–16), HT2 Object Detection (Sep 17–23), HT3 Motor Control (Sep 24–30).
   These provide inputs to Hardware System Design and later Implementation.

2. **Precedence relationships (DAG)**

Table 8: DAG

Project Scope Definition (gate for design) – Oct 1–3
- S1 Functional Scope, S2 Requirements Gathering, S3 System Boundaries → all must finish before design starts.
  Dependencies: (None) → S1, S2, S3 → D1, D2, D3

Project Planning (Design streams) – Oct 6–14
- D1 Snowplow Design (Oct 6–8) depends on S1–S3.
- D2 Hardware System Design (Oct 6–14) depends on S1–S3 and draws on HT1–HT3 results.
- D3 Software Architecture Design (Oct 8–14) depends on S1–S3.

Early Integration Check – Oct 15–17
- I0 Integration Testing (checkpoint) depends on D2 & D3 (verifies early interfaces).

Implementation – Oct 20 → Nov 5
- I1 Motor Control (Oct 20–22) depends on D2 (hardware design).
- I2 Sensor Integration (Oct 20–22) depends on D2 & D3 (hardware + software designs).
- I3 Navigation System (Oct 23–Nov 5) depends on D3 and benefits from I1 & I2 availability.
- I4 Error Handling (Oct 29–Nov 3) depends on I2 and proceeds in parallel with later I3.

System Test & Close-out – Nov 6–11
- T1 System/Acceptance Testing (Nov 6–11) depends on I3 & I4 and the I0 checkpoint.

Ongoing/parallel: Documentation & repo updates occur throughout D-phase → T-phase to satisfy deliverable readiness in the proposal workflow.

3. **6-week calendar placement**

Table 9: 6-Week Calendar Placement

Week	Dates	Planned work (with predecessors)
W1	**Oct 1–3**	**S1, S2, S3** (scope/reqs/boundaries).
	Oct 4–5	Buffer / prep for design.
W2	**Oct 6–8**	**D1** Snowplow Design (← S1–S3).
	Oct 6–14	**D2** Hardware System Design (← S1–S3, HT1–HT3).
	Oct 8–14	**D3** Software Architecture (← S1–S3).
W3	**Oct 15–17**	**I0** Integration checkpoint (interfaces from **D2 & D3**).
	Oct 18–19	Buffer / fix items uncovered by I0.
W4	**Oct 20–22**	**I1** Motor Control (← D2), **I2** Sensor Integration (← D2 & D3).
	Oct 23–25	**I3** Navigation System starts (← D3; uses I1/I2 components as available).
W5	**Oct 26–Nov 1**	**I3** continues; **I4** Error Handling (Oct 29–Nov 3) runs parallel (← I2).
W6	**Nov 2–5**	**I3** wraps; stabilize for testing.
	Nov 6–11	**T1** System/Acceptance Testing (← I3 & I4 & I0).

4. **Parallelism & critical path**
   - Parallel streams: D1/D2/D3 (design), then I1/I2 parallel; I4 runs in parallel with late I3.
   - Likely critical path: S1–S3 → D2/D3 → I0 → I2 → I3 → T1. Any slippage in D2/D3, I2, I3, or the I0 checkpoint will compress the Nov 6–11 test window.

5.3 Gantt Chart

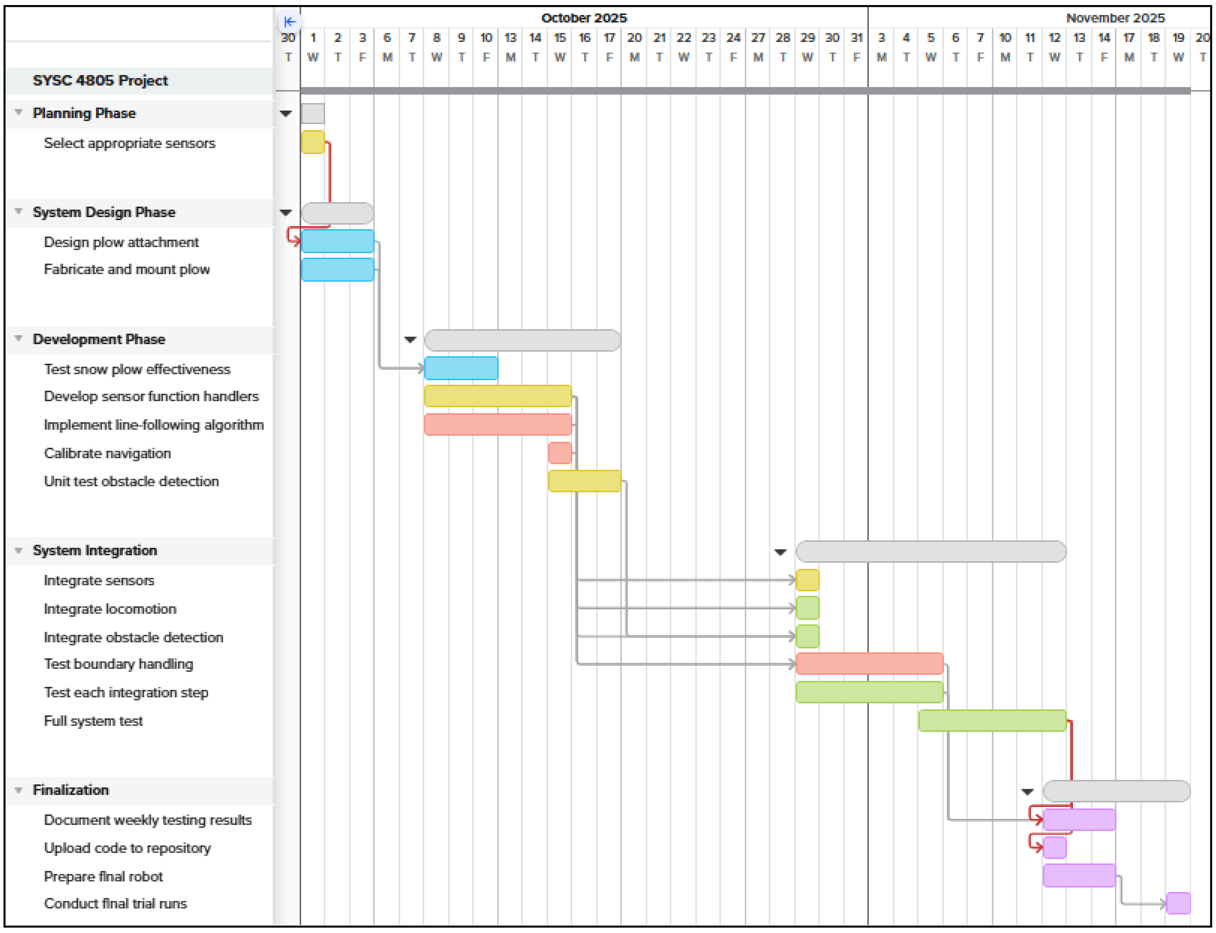


Figure 11: Gantt Chart of Project Milestones

6.0 Cost

This section details a cost breakdown of the entire project, including labor, equipment, and contingencies.

Table 10: Weekly cost breakdown

Week	Date	Planned Hours	Planned Cost ($)	Actual Hours	Actual Cost ($)	Cost Details
1	Sep 10	8	1,600	4	800	Assembling robot (8 hrs)
2	Sep 17	8	1,600	7	1,400	Testing sensors (8 hrs)
3	Sep 24	8	1,600	7	1,400	Testing motors and sensor integration (12 hrs)
4	Oct 1	8	1,600	7	1,400	Selecting appropriate sensors (1 hr)

						Fabricating and assembling plow (1 hrs)
5	Oct 8	10	2,000	8	1,600	Test snow plow effectiveness (1 hr) Develop sensor functions (5 hrs) Implement line following algorithm (4 hrs)
6	Oct 15	4	800	4	800	Calibrate navigation (1 hrs) Unit test obstacle detection (3 hrs)
7	Oct 22	0	0	0	0	N/A
8	Oct 29	14	2,800	13	2,600	Integrate sensors (1 hrs) Integrate locomotion (2 hrs) Integrate obstacle detection (2 hrs) Test boundary handling (4 hrs) Test each integration step (5 hrs)
9	Nov 5	3	600	3	600	Full system test (16 hrs)
10	Nov 12	3	600	4	800	Document testing results (3 hrs) Upload code to repo (1 hrs)
11	Nov 19	2	400	2	800	Prepare final robot (2 hrs)
12	Nov 26	2	400	1	400	Conduct final trial runs (2 hrs)
Total by:	Nov 26	70	14,000	60	12,000	

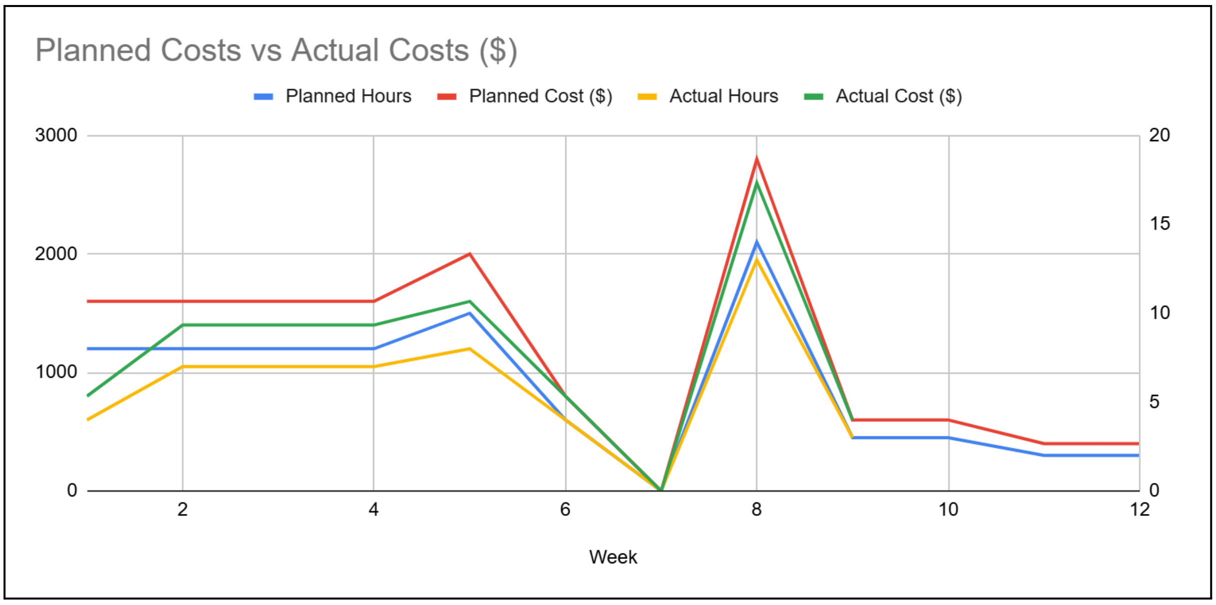


Figure 12: Planned value analysis figure

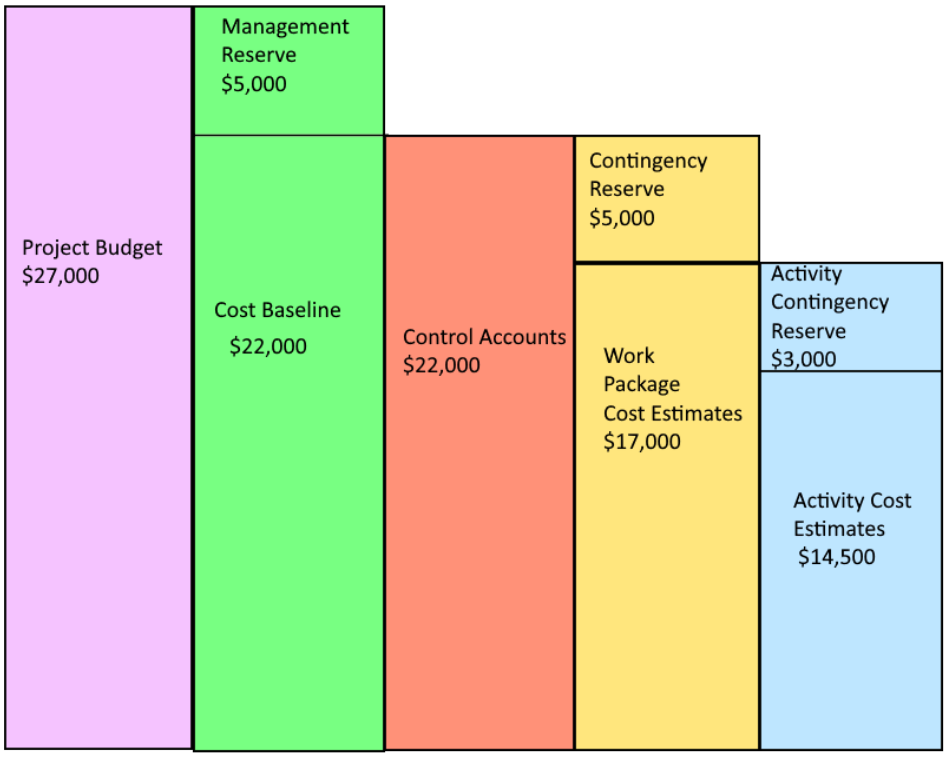

Project Budget $27,000
Management Reserve $5,000
Cost Baseline $22,000
Control Accounts $22,000
Contingency Reserve $5,000
Work Package Cost Estimates $17,000
Activity Contingency Reserve $3,000
Activity Cost Estimates $14,500

Figure 13: Cost Baseline Figure

The project was by Nov. 26 under budget, with estimated value being under the planned value of $14,000. With the estimate at completion (EAC) being roughly $12,000, the project was completed under budget. The cost estimates presented above were derived using a combination of industry referenced data. Specific cost components including Integration & Mapping, Programming & Commissioning, Batteries & Charging, and Training for Operations & Maintenance Staff were referenced industry standard data [2].

7.0 Human Resources

Below is a table summarizing the roles and responsibilities of each member.

Table 11: Responsibility Assignment Matrix Table

ID	Activity (from your schedule)	Responsible (R)	Approver (A)
A1	Design plow attachment	Kyle	Dina
A2	Fabricate & mount plow	Kyle	George
A3	Test plow effectiveness	Kyle, Dina	Jennifer
A4	Configure obstacle sensors	Jennifer	Kyle
A5	Develop sensor function handlers	Jennifer	George
A6	Test obstacle detection (static/dynamic)	George, Dina	Jennifer
A7	Integrate obstacle sensing with navigation	Dina	Dina
A8	Implement line-following (boundary detection)	Jennifer	Dina
A9	Calibrate navigation start (corner)	Dina	Jennifer
A10	Test boundary handling (stay inside perimeter)	George	Kyle
A11	Integrate nav + sensors with motor control	George	Dina
A12	Integrate plow into system	Kyle	Jennifer
A13	Iterative system testing (post-integration)	Dina	George
A14	Full system test (5-min run)	George	Kyle
A15	Document unit/integration results	Jennifer	Dina
A16	Upload code/handlers/tests to GitHub	Jennifer	Kyle
A17	Trial runs vs. success criteria	Dina	George
A18	Final demo preparation/execution	Dina, George, Kyle, Jenifer	TA/ Professor

8.0 Control Charts and Quality Management

8.1 Control Charts

8.1.1 Sensor Metrics

The following chart outlines the metrics of the 2 ultrasonic sensors and the line follower sensor readings, all averaged over the week and their deviations from expected results were graphed. As development progressed, there was a clear stabilizing trend as improvements were made to fix issues such as unstable voltage, incorrect timings, and faulty sensor readings.

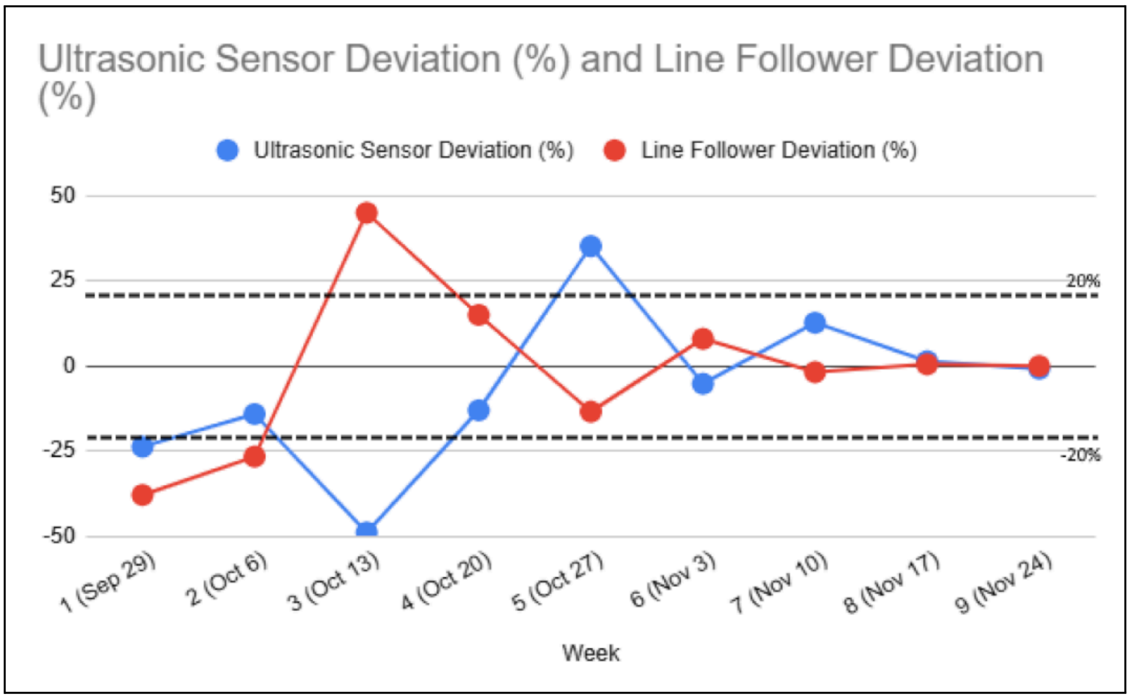


Figure 14: Control chart for sensor metrics over time

8.1.2 Movement Metrics

The following figure outlines the metrics of the movement of the robot, specifically how much time it took to stop after detecting either an obstacle or a line in milliseconds, as well as the deviation of its turning angle from the expected angle in degrees. Through trial and error to find the right timing values, the results quickly stabilized, with stopping speed stabilizing around 500ms to ensure that the robot could stop on time, but not so soon that it wouldn't push the cubes out. The turning angle deviation also stabilized getting close to 0, but due to it being a timing based metric, completely negating the angle deviation was impossible due to the unpredictability of the motors under the quickly depleting voltage of the batteries.

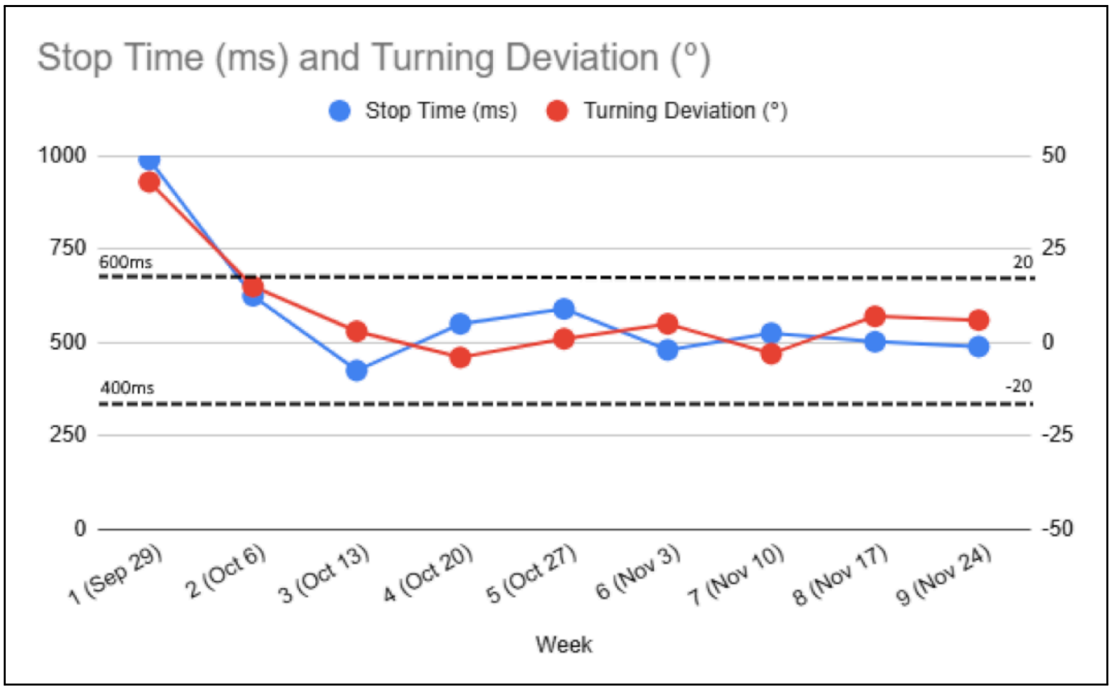


Figure 15: Control chart for movement metrics over time

8.1.3 Snowplow Metrics

The following chart outlines the performance of the robot over the development process, calculated by using the number of cubes it successfully pushed out as a baseline and subtracting the number of penalties to determine a score that would be ideally as close to 0 as possible. As improvements were made to the robot over the development process, this score progressed closer and closer to 0, never actually passing it.As nearly, all the runs resulted in at least a few cubes being left in the enclosure after 5 minutes.

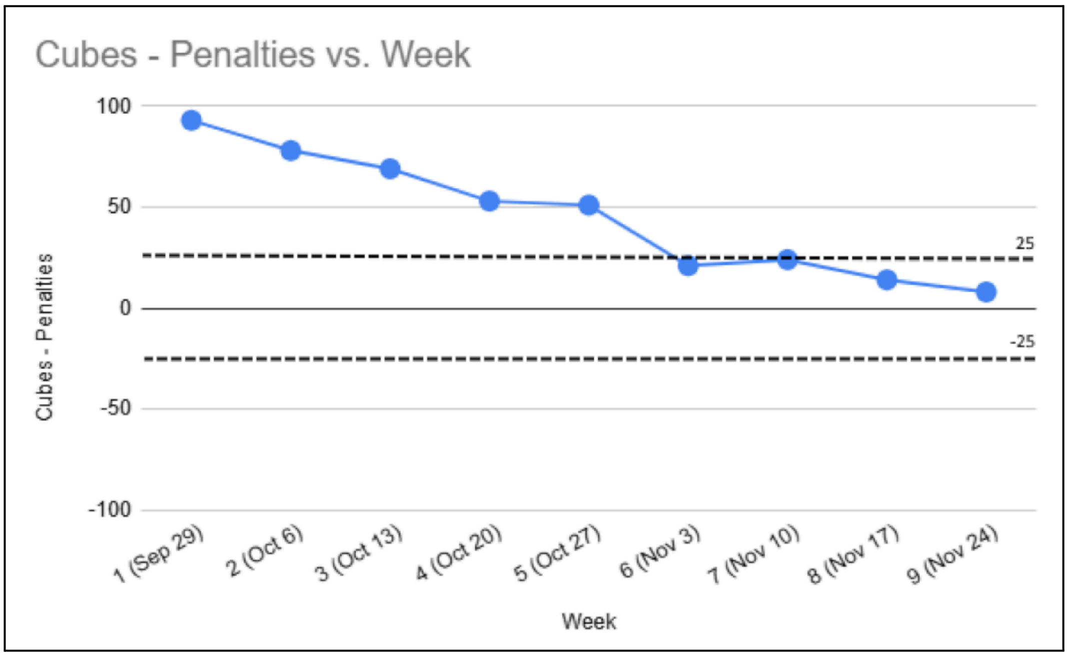


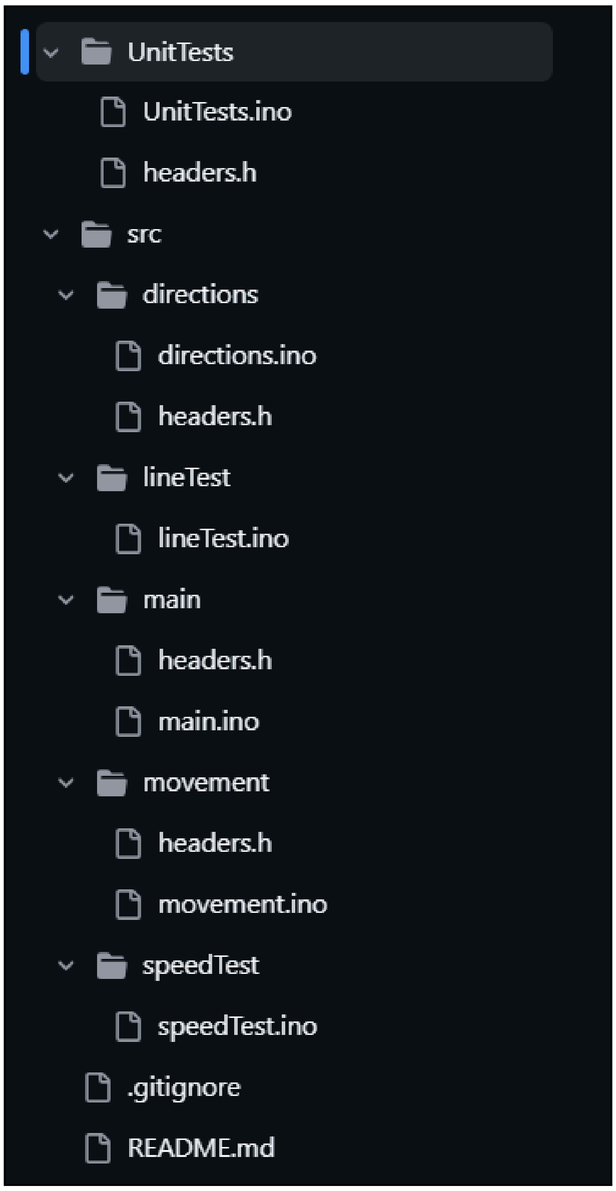
Figure 16: Control chart for snowplow metrics over time

9.0 Final Code and Repository Documentation

9.1 Github Repository Link

The repository containing the fully completed implementation and all unit tests can be found at the following link: https://github.com/SYSC4805/project-l3-g14-celeste. In the repo there is a README file detailing the structure of the repository.

9.2 Summary of Code Structure



```
UnitTests
    UnitTests.ino
    headers.h
src
    directions
        directions.ino
        headers.h
    lineTest
        lineTest.ino
    main
        headers.h
        main.ino
    movement
        headers.h
        movement.ino
    speedTest
        speedTest.ino
    .gitignore
    README.md
```

Figure 17: GitHub Code Structure

9.2.1 Folder Overview

- UnitTests: This consists of functions used to test individual components of the system.

- src: This consists of folders that represent the different parts of the system and their main functionalities. It includes the **directions**, **lineTest**, **main**, **movement**, and **speedTest** folders.
- directions: This folder implements how the DC motors directions for the autonomous snowplow are controlled and integrated.
- lineTest: This folder consists of functions responsible for detecting a black line on a white surface.
- main: This folder implements the logic for the integrated system—combining motor control, line detection, obstacle detection, and overall robot behavior.
- movement: This folder contains functions that control the robots movements patterns.
- speedTest: This defines the speed used by the autonomous snowplow .It includes 90, 130, 255.

9.2.1 Main Program Files(s) Overview

main.py

Purpose:

Implements the core logic of the autonomous snowplow, including motor control, line detection, obstacle detection, and task scheduling.

Key Functions:

- setSpeed():
  - This sets the speed for each pin of the DC motor using the **analogWrite()** function.
- setDirection():
  - This sets the motor direction for each pin of the DC motors using the **digitalWrite()** function.
- getLineValue():
  - Computes the line sensor value by averaging readings from its three pins using **analogRead()**.
- pollLine():
  - This uses the **getLineValue()** function. It uses the value obtained to compare against the white-surface threshold. If the obtained value exceeds the white threshold, a black line is detected.
- pollObstacle():
  - This uses the ultrasonic sensor to detect obstacles. It calls functions such as **digitalWrite()**, **delayMicroseconds()**, **delayMicroseconds()**, **pulseIn()**.
- getThreshold():
  - This function initializes the value for detecting a white floor.
- stationaryTurn():
  - This determines the motor behavior required to turn the robot while remaining in place.
- moveReverse():
  - This controls the reverse direction of the autonomous snowplow.
- initPins():

    - This function initializes the pins of the line sensor, motor, and ultrasonic sensor.
- startup():
    - This function configures the system by setting up the pins, speed, threshold and timer of the system.
- taskPollSensors():
    - This function is used for polling the sensors of the system, including line and obstacle sensors.
- taskRobotLogic():
    - This function is used for controlling the main logic and behavior of the robot system based on sensor inputs.
- setup():
    - This basically calls the **startup()**, creates tasks using **xTaskCreate()**, and starts the scheduler with **vTaskStartScheduler()**. It also uses Semaphores and Mutex to manage task execution.
- loop():
    - This continuously runs the polling loop to keep the system operating.

9.3 Instructions for Running the System

To run the robot system, the following sequence is executed:

- RTOS Task Creation
  At startup, the system creates two FreeRTOS tasks:
    - Sensor Task: Continuously polls the ultrasonic and line-follower sensor.
    - Logic Task: It implements the robots state machine and determines the robots actions based on global sensor readings.
- Scheduler Initialization
    - After tasks have been created, the FreeRTOS scheduler gets started. The scheduler manages and runs the tasks continuously throughout operation.
- State Machine Operation
  The robots logic tasks uses a three-state finite state machine:
    1. Forward
    2. Reverse
    3. Turning

  The autonomous robot snowplow transitions between these states whenever an obstacle or boundary line is detected.
- Running the Code in the IDE
  To compile and upload the program:
    - Ensure the folder name matches the main file name (e.g., **main/main.ino**)
    - Ensure the headers.h file is located in the same folder as **main.ino**.
    - The main file includes the header file using **#include "headers.h"**.

REFERENCES

[1] Matt Campbell, “Apparently you can set a world record in autonomous snow plowing now,” *CarsGuide,* Aug. 23, 2018. [Online]. Available: https://www.carsguide.com.au/oversteer/apparently-you-can-set-a-world-record-in-autonomous-snow-plowing-now-70547 . [Accessed: Oct. 14, 2025].

[2] standardbots, “How much do robots cost? 2025 price breakdown”, STANDARDBOTS, Aug. 7, 2025. [Online]. Available: https://standardbots.com/blog/how-much-do-robots-cost. [Accessed: Nov. 6, 2025].