Computer Systems Design Lab

Instructor: Mostafa Taha

SYSC4805 Project Final Report

Team Egyptian Blue, Section L3 Group 5

Ahmed Ali, 101181126

Duncan MacLeod, 101160585

Tauheed Alamgir, 101194927

Brendan Kennedy, 101203359

Report Start Date: November 15th, 2024

Report Due Date: December 6th, 2024

1.0 Final Report

1.0 Updated Project Proposal

1.0 Project Charter

1.1 Team Name

Egyptian Blue

1.2 Objective

The aim of this project is to create a working prototype of a snow removal autonomous robot. The robot is tasked with removing simulated snow in the shape of small cubes. The robot will navigate around both static and dynamic obstacles in the shape of wooden boxes within an established 6 m^2 circular arena marked by a black tape. The basis of our concept is the integration of three separate sensors on Arduino DUE and Nano Every microcontroller boards, which will give the robot strong navigational capabilities and enable it to maneuver around both static and dynamic obstacles. The robot's design will meet specific criteria, such as a speed limit of 30 cm/s and a 5-minute operational window, which are crucial to avoid penalties. The plow design is equally as important, as we need to maximize our snow clearance to earn most of the points. Our start point is trial and error and make sure we are logging different types of data. This logging will not only let us monitor the speed and accuracy of our robot, but it will also be a crucial tool for analyzing sensor data, improving our design, and pinpointing possible improvement areas. By working on this project, we are hoping to gain knowledge in both embedded systems and robotics fields to apply in our future careers.

1.3 Project Deliverables

1. Project Proposal (October 18th)
    - An informational report about the requirements, scope, schedule, cost, and risk analysis.
2. Progress Report
    - Here we will highlight the challenges that have been faced during the project and the solutions that have been implemented to address them. An updated project timeline and milestones will be presented. Any changes made to the project's scope, or its requirements will be detailed comprehensively.
3. In Lecture presentation
    - The presentation will discuss team roles and individual contributions. An interactive session with the audience will be held, which will include a Q&A

segment. The highlight of the presentation will be a live demonstration showcasing the robot's functionalities.

4. Lab Demonstration
    - The demonstration will showcase the robot's capability to avoid obstacles and remove the simulated snow from the arena. The robot's final software interface and its control functionalities will be provided on the GitHub codebase.
5. Final Report
    - The Final Report milestone will start with an executive summary, providing an overview of the entire project. This will be followed by an in-depth analysis of the project's results and outcomes. Recommendations for future work and potential improvements will be discussed. The report will conclude with appendices containing detailed data, charts, and code.

Final Deliverables

- Finalized robot design
    - A fully functional robot equipped with integrated sensors, the plow attachment, and any other necessary modifications based on milestones' feedback.
- Source code:
    - Complete code base that shows the logic of all the sensors.
    - Documentation explaining the logic of the sensors.
- Final Report
    - Detailed description of the finalized robot prototype with integrated sensors and plow attachment.
    - Comprehensive documentation of the software, including code logic, execution flow, and operational procedures.
    - Sensor data interpretation, providing insights into environmental interactions, obstacle detection accuracy, and sensor reliability.

2.0 Scope

2.1 Requirements

Functional Requirements

- Upon activation, the robot's system shall start its operation sequence within the boundaries of a space of 2.5 x 2.5 meters.
- On power up, the system shall perform a full calibration of all sensors
- During movement, the system shall detect small wooden cubes with a side length of 20mm and push them towards the designated outside perimeter.
- When approaching the black path boundary, the robot shall prevent itself from crossing the line by more than 5cm. The wheels going outside the boundary by more than 5cm results in the deduction of 5 cubes.
- Upon detecting an obstacle in its path, the system shall navigate around the obstacle without causing any significant displacement.
- During operation, the robot shall maintain dimensions within 226 x 262 x 150 mm.
- The robot's speed shall stay below or at the limit of 30cm/s.
- Upon activation, the robot should clear all the wooden cubes within a duration of 5 minutes.
- In any case of a small hit with the obstacle which does not move the obstacle, results in the deduction of 5 cubes.
- In any case of a strong hit with the obstacle, which pushes the obstacle away, results in the deduction of 10 cubes.
- Any human interaction with the robot for adjustment or resets, results in the deduction of 10 cubes.
- If the robot's speed exceeds the permissible limit at any point during the test, results in the deduction of 10 cubes.

Non-Functional Requirements

- During all operational phases, the robot's speed should maintain a consistent speed not exceeding 30cm/s.
- Throughout the task, there shall be a timer to maintaining a countdown not surpassing a total of 5 minutes for task completion.
- Throughout the tasks, there should be a smooth and precise movement to effectively move around obstacles without causing displacements.

Software Requirements

- For enhancing modularity, the function handler of at least three sensors should be submitted to the designated GitHub repository.

2.2 Deliverables

2.2.1 Work Breakdown Structure (WBS)

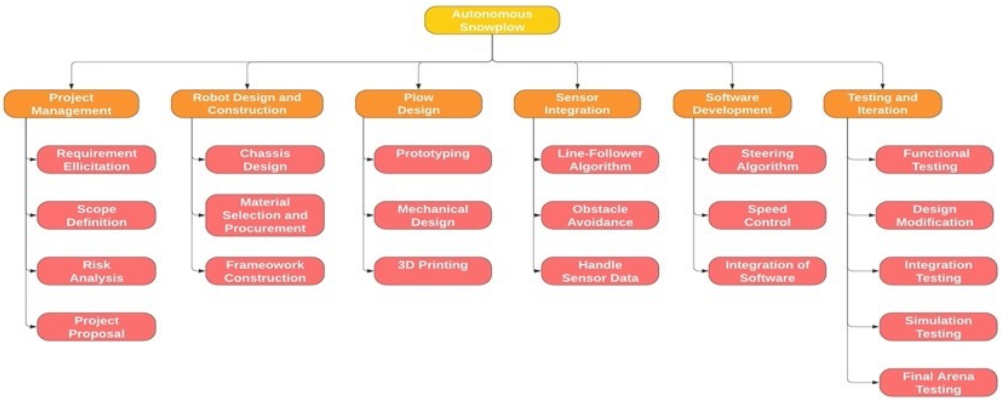


Figure 1: A list of detailed deliverables that cover all the requirements, organized in a WBS.

Update on Work Breakdown Structure for the Final Report: By the end of week 11, we have completed all tasks pertaining to project management, robot construction, plow design, sensor integration, software development and testing. For the Robot Design section, we have completed the chassis design within the given dimension constraints and selected materials. The only module we were unsuccessful in completing was the simulation testing, due to an error in motor functionality. Overall, the work breakdown structure was followed for the most part and provided a useful guide on module development throughout the project.

2.2.2 Movement Module

The Movement Module will be based on a PID control system loop. The system will have two input signals: The voltage signals driving the left and right sides of the robot. To help illustrate, a rough figure of the robot chassis is shown below.

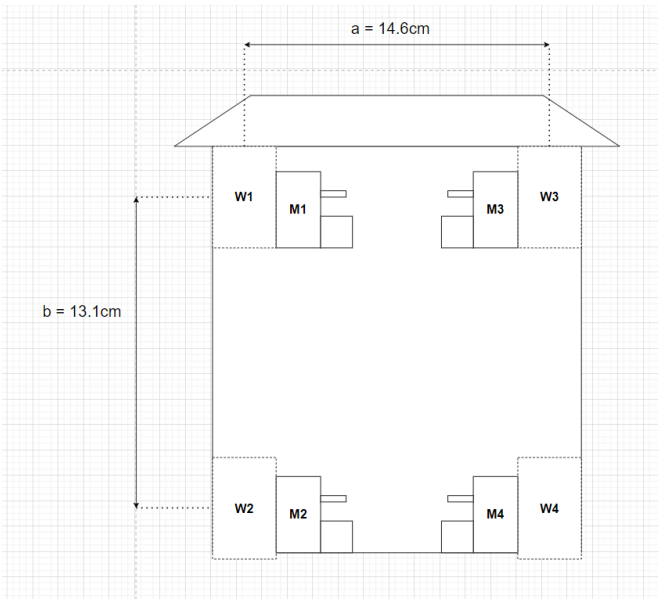


Figure 2: Diagram of rough robot chassis with labelled motors

Each motor is being driven by a voltage waveform and producing an angular velocity as a result. Given that motor pairs (M1, M2) and (M3, M4) are a fixed length apart, we simplify this problem by reducing from four independent elements to just two, one for each side. Let $vL\ (t)$ be the PWM signal driving the left motors M1, M2, and $vR\ (t)$ be the PWM signal driving the right motors M, M4. $vL\ (t)$ and $vR\ (t)$ are the inputs to the PID system. The outputs to the PID system are the left and right angular velocities $\omega L\ (t)$ and $\omega R\ (t)$. These output angular velocities can be captured by the left and right wheel encoder outputs. Having access to the output velocities is critical because it allows us to create a closed-loop system with input signals $vL\ (t)$ and $vR\ (t)$ controlling outputs $\omega L\ (t)$ and $\omega R\ (t)$.

Figure 3 below shows the general block diagram of the movement module

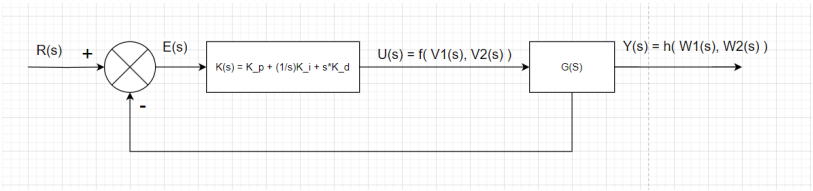


Figure 3: Block diagram of movement module

Where:

$R(s)$ is some desirable output state

$E(s)$ is the difference between the desired output, $R(s)$, and the actual output, $Y(s)$

$K(s)$ is the actual PID controller

$U(s)$ is some function of the two voltage inputs

$Y(s)$ is the output, which is some function of the two outputs

The exact form of U(s) and $Y(s)$ will depend on the task being carried out. If, for example, the Movement Module was tasked with producing straight line movement, then $Y(s)$ would simply be the difference between the two angular velocities, and the desired output would be zero.

2.2.3 Event Manager

The Event Manager will be the main interface between the sensors and the rest of the system. All relevant sensor information will be fed into the Event Manager, which will monitor incoming data and send commands to the Movement Module as needed.

2.2.4 Coordinate System

The Coordinate System will act as a bare-bones map of the environment and will assist the robot in keeping track of its position in the demo environment. The Coordinate System will feed into the Event Manager, allowing the Event Manager to cross reference between its expected surroundings, according to the Coordinate System, and its observed surroundings, according to various sensor feeds being received. Having an estimated position reading may prove useful for anticipating potential stops/redirections earlier in advance, allowing for more reliable and robust movement

2.3 Testing

2.3.1 Unit Testing

Ultrasonic Distance Sensor

- This sensor allows the robot to be able to detect obstacles and avoid them.

ToF Distance Sensor

- The ToF sensor allows the robot to detect both static and moving objects.

Analog Distance Sensor

- Test the visibility scope, reliability, and precision of each sensor for variable distances and angles. With a large enough dataset

IMU Gyroscope, and Accelerometer, and Magnetometer

- The gyroscope reading can reliably be used to determine how fast the car is turning

VMA330 Obstacle Detector

- These sensors can detect obstacles from a range of 2cm to 40cm.

Line Follower Sensor

- The line follower sensor allows the robot to keep track of the black painter's tape.

Robot Design and Construction

- Chassis Design: We will physically measure the dimensions of the chassis to verify it against the dimension of 226 x 262 x 150 mm. There should be no deviation beyond this tolerance.
- Material Selection and Procurement: Materials will be checked against the procurement list to ensure alignment with the design requirements.
- Framework Construction: Our team will examine the robot's framework for any visible structural defects and conduct a stability test under various load conditions. Framework should be stable under all conditions and free from defects.

Plow Design

- We will review the AutoCAD design and ensure it matches the physical prototype. We will also check the plow's ability to push lightweight wooden cubes simulating snow.

Sensor Integration

- Each sensor will be tested and evaluated (previous labs) to ensure the system effectively processes and responds to senor data. For instance, with the line follower algorithm, we will mark a black path and observe the robot's behavior when approaching. Each sensor should provide accurate readings and respond correctly to its designed parameters.

2.3.2 Integration Testing

Integration testing focuses on evaluating the interfaces and interactions between integrated components or systems. For our robot, the goal of integration testing is to confirm that individual modules, when combined, work as expected in unison.

- Chassis and Plow Integration: This is to confirm that the attachment mechanism between the chassis and the plow is secure. We would integrate the plow's mechanism with the robot's movement controls and test that the robot navigates and pushes lightweight wooden cubes with the plow attached. There should be no hindrance in the robot's movement.
- Sensor and Chassis Integration: With all the sensors mounted on the robot, we shall evaluate if the chassis supports them without causing interference or obstructing their functions. The robot's movement will be tested to determine if the sensors maintain their functionality and provide accurate reading consistently.
- Error and Exception Handling: We will intentionally induce errors, like blocking a sensor or overloading the robot, to see if the integrated system can identify the issue and handle it gracefully. The robot should not crash or behave erratically. Instead, it should follow predefined procedures for error handling, like stopping and signaling an error with a meaningful message.
- Functional Requirement Integration
    - Controlled Environmental Testing: In the designated testing area of 2.5 x 2.5 meters, we must make sure the area is free from external disturbances and distractions. Within this area, we will then set up specific conditions such as different cube placements and boundary markers to test the robot's behavior against these requirements.
    - Boundary Checks: We will mark the testing area with a clearly defined black path boundary (black tape) and observe the robot's behaviors when it approaches this boundary. The robot should not cross the line by more than 5cm.

Movement (Wheel Encoders + Motors):

The testing for the robot movement will include 3 stages:

1. Speed: Can the robot maintain a speed within a 1cm/s error

Test: The test for speed will be operating the robot to move at 20cm/s. The test will pass if the robot can maintain a speed of 20cm/s +- 1cm/s. If the robot's speed drops below 19cm/s or above 21cm/s, the test is a failure.

2. Direction: Can the robot maintain a straight line without veering off path

Test: To test straight line operation, a makeshift boundary (line of tape) will be set up, and the robot will be setup right next to it and will move straight forward for 2.5m. The test will pass if the robot maintains within 2cm of the tape but fails if it crosses over the tape or moves away from the tape 2cm or more.

3. Turning: Can the robot accurately turn left and right (90 degrees)

Test: To test turning, the robot will be tested at attempting a 90 degree turn in both left and right directions. The test will pass if the robot is able to turn in a range of 88 degrees to 92 degrees (98% accuracy). If the robot turns more than 92 degrees or less than 88 degrees in either direction, the test is a failure.

Navigation:

The testing for navigation will include 2 stages:

1. Mapping: Can the robot accurately map a rough grid of the area in which it clears

Test: A makeshift grid of tape will be made, and the robot will converse around the grid for 1 minute and must accurately map the border of the grid. The test will pass if the robot can accurately map the grid with 98% confidence, meaning a +- 2% margin on dimensions of grid. The test will fail if this confidence margin of 98% is not met, or the robot fails to create the map in the first place.

2. Positioning: Does the robot know it's position and orientation within this grid

Test: To test the positioning, the robot will be placed in a makeshift testing grid and will navigate to random points within the grid. The test is to have the robot communicate in Arduino its current coordinates within the grid and it/s orientation XYZ. The test will pass if the readings are approximately 95% accurate compared to the expected (observed) data. The test will fail otherwise.

Sensor Communication with Robot:

Each sensor will be calibrated within the unit testing, but will need to communicate with the robot to perform certain tasks:

1. Obstacle Detection -> Speed: Can the robot accurately detect the distance from an object and decelerate accordingly

Test: An obstacle will be set up 1 meter from the robot, and the robot will move at 20cm/s towards the obstacle. The robot must decelerate smoothly as it gets closer to the obstacle and comes to a stop about 5cm from the obstacle. The test pass/failure will be observed

as so: If the robot decelerates relatively smoothly proportional to its distance from the obstacle, and stops within a range 4.5 to 5.5 cm, the test will pass. If the deceleration is sudden or the robot drives into the obstacle, the test will fail.

2. Line Detection -> Speed: Can the robot slow down or stop if a line is detected closely in front of it.

Test: A line of tape will be set up to simulate the border. The robot will move towards the tape and the line must be detected by the line detection sensor and stop the robot. If the robot drives over and past the line, the test will fail. If the robot stops at the line, the test is successful.

3. Grid -> Navigation: Can the robot maneuver itself around its constructed grid maintaining knowledge of its position and orientation.

Test: The test for grid navigation will be the final test in our testing plan. Integrating all our previous calibration, the robot will move around an empty grid, following our constructed navigation plan. As the robot navigates it must detect lines to keep itself within bounds, must detect obstacles and avoid hitting them, and will be displaying its position and orientation to the Arduino terminal whilst navigating. The passing criterion for this test is that the robot stays in boundaries (wheels within 5cm of line), avoids obstacles (stops and turns when detected), and can cover the area in our navigation plan. The test will fail if any of these criteria are not met in tandem.

3.0 Schedule

3.1 List of Activities

Table 1: List of activities that covers all the deliverables in the WBS

	Activity	**Detailed Activities**
1	Project Management (PM)	1.1 Planning and research (PM – 01) 1.2 Defining functional requirements (PM – 02) 1.3 Resource allocation (PM – 03) 1.4 Progress Monitoring (PM – 04)
2	Robot Design and Construction (RDC)	2.1 Metal chassis design and assembly (RDC – 01) 2.2 Material selection (RDC – 02) 2.3 Arduino and motor board placement (RDC – 03) 2.4 Sensor placement (RDC – 04)
3	Plow Design (PD)	3.1 Prototyping with different shapes and materials (PD – 01) 3.2 Develop final design (PD – 02)
4	Plow Final Design (PFD)	4.1 Obtain final structure using 3D printer (PFD – 01)
5	Sensor Integration (SI)	5.1 Sensor selection (SI – 01)
6	Lane Detection (LD)	6.1 Develop line-follower algorithm (LD – 01) 6.2 Sensor function handlers for data acquisition and processing (LD – 02)
7	Obstacle Detection (OD)	7.1 Integrate time-of-flight sensor for obstacle detection (OD – 01) 7.2 Sensor function handlers for data acquisition and processing (OD – 02)
8	Software Development (SD)	8.1 Control software for running all sensor concurrently (SD – 01) 8.2 Path planning and obstacle avoidance algorithm (SD – 02) 8.3 Speed control and steering integration (SD – 03) 8.4 Software-sensor integration (SD – 04) 8.5 Testing and debugging (SD – 05)
9	Testing and Iteration (TI)	9.1 Initial function testing (TI – 01) 9.2 Iterative design modifications (TI – 02) 9.3 Integration testing (TI – 03) 9.4 Simulation testing (TI – 04) 9.5 Arena testing (TI – 05)

10	Document and Reporting (DR)	10.3 Code documentation (DR – 01) 10.2 GitHub update (DR – 02)
11	Final Demonstration (FD)	11.1 Final presentation (FD – 01) 11.2 Demonstrating snow clearing in arena (FD – 02)
12	Project Proposal (PP)	12.1 Develop project proposal (PP – 01)
13	Progress Report (PR)	13.1 Create progress report (PR – 01)
14	Final Report (FR)	14.1 Create final project report (FR – 01)
15	In Lecture Presentation (ILP)	15.1 Perform in lecture presentation (ILP – 01)

Schedule Network Diagram

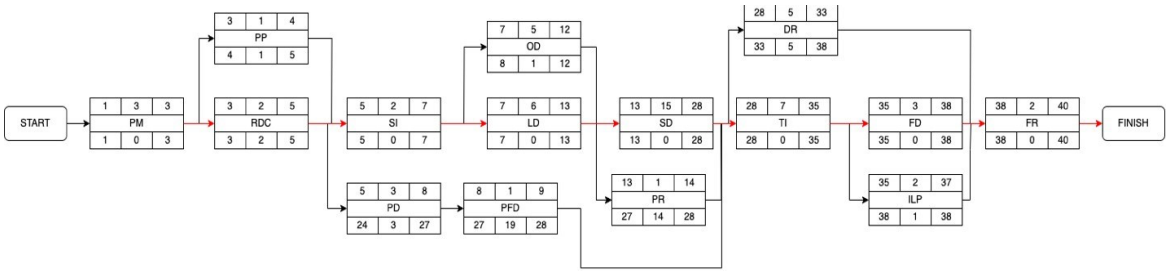


Figure 4: Schedule Network Diagram

Final Report Update on SND: The schedule network diagram was a tool created to plan the sequence of tasks and dependencies providing us a critical path for the project deadline. We attempted to closely align our time spent on each task in accordance with our initially planned schedule. However, some tasks took longer than planned to leave us with less time than planned for other tasks. Overall, we completed our tasks by our created final deadline, however with some unsatisfactory results for certain components.

Gantt Chart

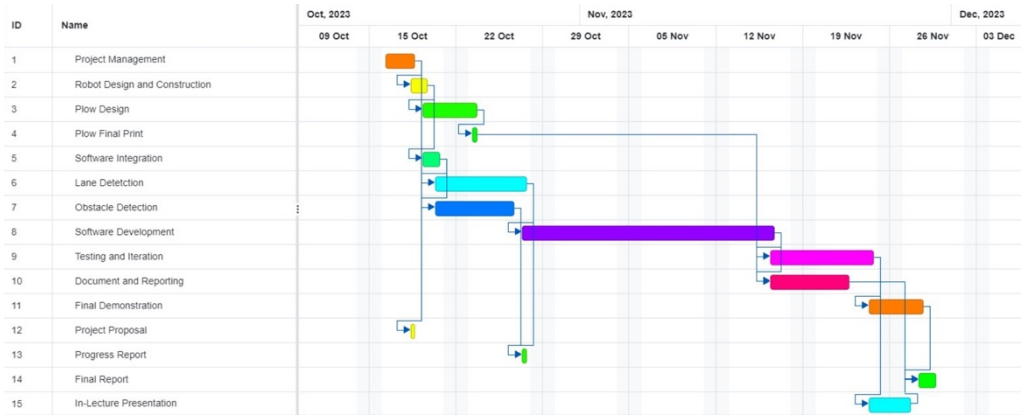


Figure 5: Gantt Chart

Final Report Update on Gantt Chart: The overall purpose of the gantt chart was to provide us with the time we should expect to develop each stage of the robot. The schedule we created was broken down into the major tasks including robot design, software development as well as our report deliverables. As expected, the longest task was software development spanning several weeks and the shorter tasks such as the reports spanned only 1-2 days. We ended up conducting tests and iterations incrementally throughout the development, so it ended up spanning several weeks rather than the 1 we planned for it. Overall, all our deliverable deadlines were successfully met as scheduled following our gantt chart.

4.0 Project Cost

Carleton University provides a kit valued at approximately $500. This kit includes various sensors and construction elements required for the robot project. These components are essential for the project's successful development and are generously provided by Carleton University to support the endeavor. In addition to the provided kit, there is an outside budget consideration for the snowplow design and implementation. The cost of this component depends on the material used, which is plastic, and the custom figure, which is 3D printed. The estimated cost for the plastic material and 3D printing can range from $20 to $50 per kilogram [2]. It's important to note that the snowplow's weight is a critical factor since a heavier plow would consume more energy for the robot's operation. Ideally, the plow's weight should maintain a ratio of 1:8 in comparison to the robot's weight. Proper cost allocation for these essential components is vital in ensuring the successful development of the project.

4.1 Cost Breakdown per Activity:

Core Movement Patterns – 4 developers, 3 weeks, 7h/week, 85h x $50/h = **$4200**

Navigation - 4 developers, 2 weeks, 9.5 h/week = 4x2x9.5 = 76 h x $50/h = **$3800**

Plow Design - 4 developers, 2 weeks, 0.75h/week = 4x2x0.75 = 6 h x $50/h = **$300**

Sensor Calibration - 4 developers, 3 weeks, 7h/week, 85h x $50/h = **$4200**

Unit Testing – 4 developers, total 7 weeks, 5h/week = 4x7x5 = 155h x $50/h = **$7750**

Integration Testing - 4 developers, total 7 weeks, 5h/week = 4x7x5 = 155h x $50/h = **$7750**

Course Kit = **$500**

Adding Costs Together - Total Cost: $28,500

4.2 Planned Value Breakdown by Week

Table 2: List of planned value for each task, broken down by week

Week	Task A PV	Task B PV	Task C PV	Task D PV	Task E PV	Task F PV	Task G PV	Total PV
1	$500	$1400	$1900	$150	$1400	$1107	$1107	$7564
2		$1400	$1900	$150	$1400	$1107	$1107	$7064
3		$1400			$1400	$1107	$1107	$5014
4						$1107	$1107	$2214

5						$1107	$1107	$2214
6						$1107	$1107	$2214
7						$1107	$1107	$2214

4.3 Cost Baseline Figure

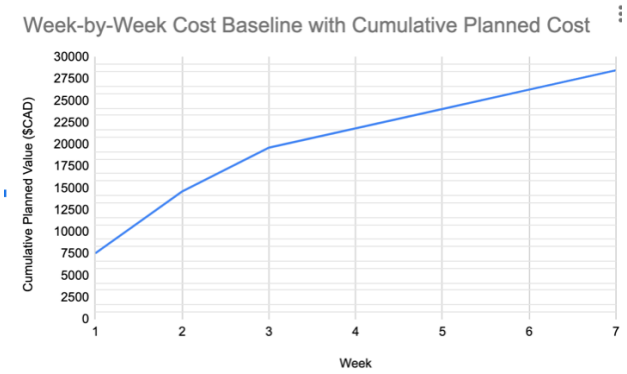


Figure 6. Cost Baseline Figure weekly breakdown

5.0 Human resources

R = Responsible A = Approver

Table 3: List of activities and their respective responsible and approvers

Activity	Ahmed Ali	Brendan Kennedy	Duncan MacLeod	Tauheed Alamgir
Project Management (PM)	R	A	/	/
Robot Design and Construction (RDC)	/	/	A	R
Plow Design (PD)	/	R	/	A
Plow Final Design (PFD)	/	A	/	R
Sensor Integration (SI)	A	/	R	/
Lane Detection (LD)	/	R	/	A
Obstacle Detection (OD)	A	/	R	A
Software Development (SD)	R	/	A	/
Testing and Iteration (TI)	A	/	R	/
Document and Reporting (DR)	/	R	A	/
Final Demonstration (FD)	R	A	/	/
Project Proposal (PP)	R	/	/	A
Progress Report (PR)	/	/	A	R
Final Report (FR)	/	A	R	/
In Lecture Presentation (ILP)	A	R	/	/

2.0 Updated Progress Report

2.0 System Architecture

2.1 Updated Architecture

2.1.1 Movement Module

Significant progress has been made towards the PID Movement Module since the Proposal submission. One change to the original design is that the PID will now use phase as the control variable instead of angular velocity. The change from angular velocity to phase is due to periodic variance in wheel encoder pulse length at steady state, which is described in more detail in Section 2.2.1.

The movement module will use a pair of PID controllers, one for each of the left-side and right-side motors. The input to the PID network will be two inputs:

Common Mode Velocity (V_{cm})

Velocity Difference (V_{diff})

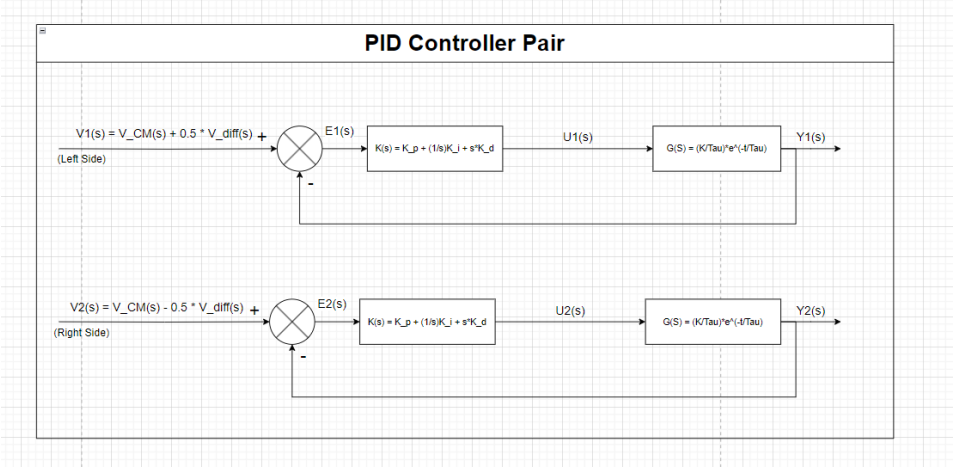


Figure 7. Block Diagram of PID Controller Pair

2.1.2 Event Manager – PID Controller Interface

While working towards implementing the Movement Module, it became apparent that some kind of intermediate submodule between the Event Manager and the PID Controllers would be required to compute the required PID inputs from the Event Manager's command. Since the Event Manager will be sending commands based on the sensory input it receives, it follows that commands from Event Manager to the Movement Module will describe some desired trajectory, and the intermediate "Command Decoder" will compute the required V_{cm} and V_{diff} controller inputs to satisfy the command from the

Event Manager command. The Command Decoder subsystem will be part of the movement module, placed in front of the movement controller. A functional diagram of this interface is given below:

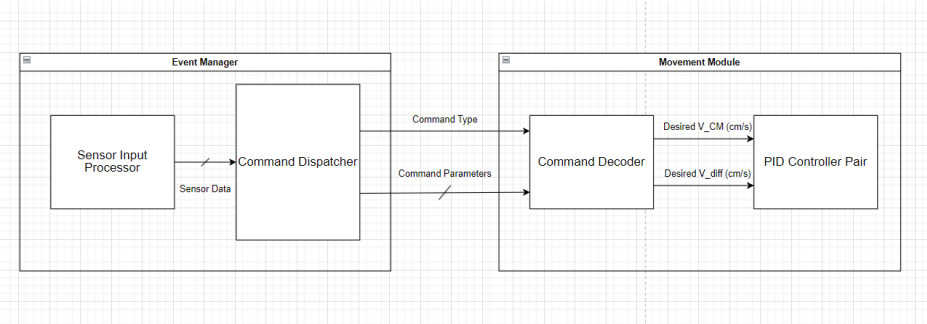


Figure 8. Functional Level Interface between Event Manager and Movement Module

2.2 PID Movement Module Progress

2.2.1 Wheel Encoder Data Analysis

A large part of the work towards the PID controller implementation has been analyzing the quality of the wheel encoder output. The wheel encoder output allows the system to measure its output velocity, which is what makes closed loop control possible. Given that the PID controller pair is controlling the system's movement, the importance of clean and reliable wheel encoder output is crucial to the PID controller's functionality.

2.2.1.1 Gathering Data

An experiment was set up using the Arduino Nano Every to test the wheel encoder output. The experiment lasted three seconds in total. The target motor was put at rest for the first second, and then driven high for two seconds. The goal of the experiment was to record each rising edge event of the wheel encoder output while the motor was running. The TCA0 peripheral kept time, it was set up so OVF interrupts occurred every 10 milliseconds (The peripheral clock was 250kHz, and TCA0.PER was set to 2500). TCB0 was set to listen to the wheel encoder output and count events. When a new event was recorded, it was timestamped and stored in a buffer. At the end of the experiment, the Arduino printed the contents of the buffer to the Serial Monitor. An example of the printed buffer contents is shown below.

```
17:02:54.377 -> Event Count = 51
17:02:54.377 -> [0] Frame = 1d; Counter = 13d
17:02:54.413 -> [1] Frame = 105d; Counter = 547d
17:02:54.440 -> [2] Frame = 112d; Counter = 351d
17:02:54.504 -> [3] Frame = 117d; Counter = 1416d
17:02:54.536 -> [4] Frame = 122d; Counter = 1345d
17:02:54.568 -> [5] Frame = 127d; Counter = 974d
17:02:54.600 -> [6] Frame = 132d; Counter = 151d
17:02:54.632 -> [7] Frame = 135d; Counter = 1535d
17:02:54.664 -> [8] Frame = 136d; Counter = 880d
17:02:54.696 -> [9] Frame = 136d; Counter = 1803d
17:02:54.728 -> [10] Frame = 141d; Counter = 382d
17:02:54.792 -> [11] Frame = 145d; Counter = 1014d
17:02:54.824 -> [12] Frame = 149d; Counter = 1141d
17:02:54.857 -> [13] Frame = 153d; Counter = 967d
17:02:54.888 -> [14] Frame = 157d; Counter = 763d
17:02:54.921 -> [15] Frame = 161d; Counter = 719d
17:02:54.955 -> [16] Frame = 161d; Counter = 1088d
17:02:54.984 -> [17] Frame = 165d; Counter = 1760d
```

Figure 9. Sample wheel encoder output data

2.2.1.2 Data Cleaning and Analysis

One immediate issue with the raw data is that it had occasional 'glitch' outputs, a couple of which can be seen in the timestamps in Figure 9. Entry 8 and Entry 9 for example supposedly occurred in the same frame, 923 clock ticks apart. At a clock frequency of 250kHz, this would suggest that just 3.69 milliseconds had passed between rising edge events. The wheel encoder gears have 10 teeth, so the time between pulses ideally would indicate a rotation of

$$\left(\frac{1}{10}\right)2\pi = \frac{\pi}{5}$$

radians. For these two consecutive events to both be legitimate, the wheels would have needed to spin at $\omega = \left(\frac{\pi}{5}\right)\left(\frac{1}{13.69 \cdot 10^{-3}}\right) = 170.18 rad/\sec$, which is far outside the range of feasibility. The cause of these glitch outputs was initially thought to be a debouncing issue. Upon further testing and analysis, the current assumption is that EMI is mainly responsible for the occasional erroneous output.

2.2.2 Deriving Motor Transfer Function

The general step response for a motor with voltage input and angular velocity output takes the following form:

$G(s) = \frac{K}{\tau s+1}$ (s-domain)

$g(t) = K\left(1 - e^{\frac{-t}{\tau}}\right)$ (time domain)

The experiment described in the above section is functionally a step response test. After manually cleaning the output to remove erroneous data, the result is an asynchronously sampled motor step response. The only unknown parameters are the static gain (K) and the time constant (τ). The static gain was easily extracted by looking at the steady state output, which converged to a constant value. Extracting the time constant is slightly more error-prone, but can be achieved by averaging multiple trials together. The extracted static gain value was found to be approximately $K = 2$, while the time constant was found to be $\tau = 0.1$ seconds.

2.2.3 Wheel Encoder Event Prediction

The first attempt to improve the wheel encoder output was to create an algorithm that predicted approximately when the next wheel encoder output should be. Now that the motor's phase impulse response is known, the idea of this prediction algorithm is that it receives a buffered feed of the battery pack voltage (which determines the magnitude of the voltage difference applied to the motor terminals when a motor is driven), and performs a discrete convolution using the already known phase impulse response. To improve computation speed, the impulse response was stored as a hard-coded lookup table (Located on github at duncan_dev/lib/motor_tf.h), sampled at 100Hz. The discrete convolution implementation is located in the same directory (See motor_tf_utils.c).

3.0 State chart and Sequence Diagrams

3.1 UML State Diagram

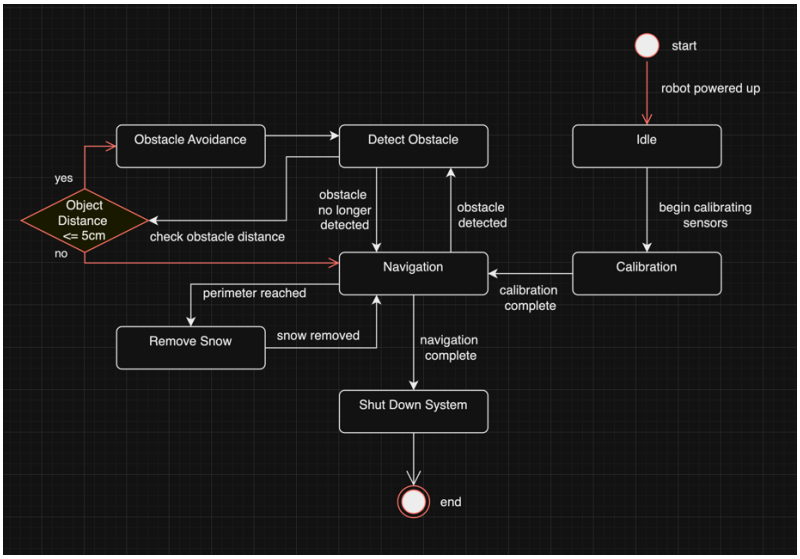


Figure 10: Diagram of UML State Diagram

3.2 UML Sequence Diagram

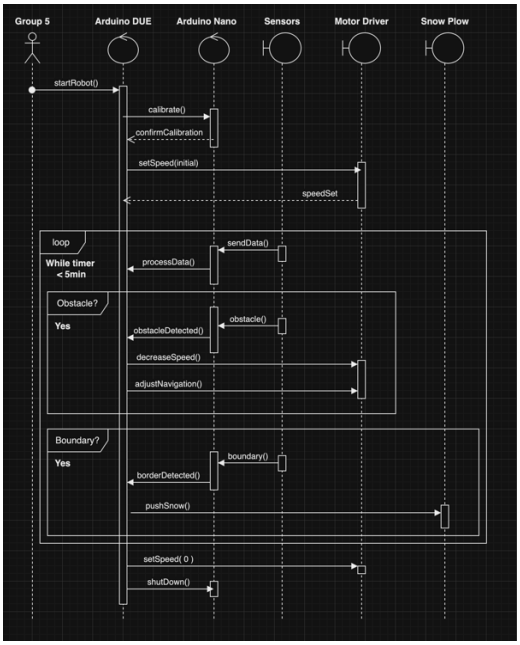


Figure 11: Diagram of UML Sequence Diagram

4.0 Watchdog Timer Utilization

In normal expected operation, the watchdog timer will be periodically cleared to prevent a system restart. The watchdog timer is intended to be an emergency recovery feature in case the system goes into an error state where it is not processing events normally. In such a situation, the system will fail to clear the watchdog timer, which will cause a system reset, allowing the system to regain functionality.

5.0 Planned Value Analysis

5.1 Cost Breakdown by Activity

A: Course Kit = **$500**

B: Core Movement Patterns – 4 developers, 3 weeks, 7h/week, 85h x $50/h = **$4200**

C: Navigation - 4 developers, 2 weeks, 9.5 h/week = 4x2x9.5 = 76 h x $50/h = **$3800**

D: Plow Design - 4 developers, 2 weeks, 0.75h/week = 4x2x0.75 = 6 h x $50/h = **$300**

E: Sensor Calibration - 4 developers, 3 weeks, 7h/week, 85h x $50/h = **$4200**

F: Unit Testing – 4 developers, total 7 weeks, 5h/week = 4x7x5 = 155h x $50/h = **$7750**

G: Integration Testing - 4 developers, total 7 weeks, 5h/week = 4x7x5 = 155h x $50/h = **$7750**

Adding Costs Together - Forecasted Budget: $28,500

Timeline for Project Completion: End of Week 7 --> November 29th

5.2 Weekly Planned Value

Table 4: List of tasks by their planned values, broken down by week

Week	Task A PV	Task B PV	Task C PV	Task D PV	Task E PV	Task F PV	Task G PV	Total PV
1	$500	$1400	$1900	$150	$1400	$1107	$1107	$7564
2		$1400	$1900	$150	$1400	$1107	$1107	$7064
3		$1400			$1400	$1107	$1107	$5014
4						$1107	$1107	$2214
5						$1107	$1107	$2214
6						$1107	$1107	$2214
7						$1107	$1107	$2214

Analyzing the Planned Value table above, it is seen that in the original budget plan, the work was not delegated evenly week to week according to the weight of the budget. This is obvious as week 1 had a planned value of $7564, yet weeks 4-7 only had a planned value of $2214. However, this was only initially the breakdown of costs by week. The actual weekly timeline of these deliverables up until the end of week 5 has taken a far different

route so far, which is explored in section 5.3 below. The following cost baseline figure can be shown for the initial budget analysis:

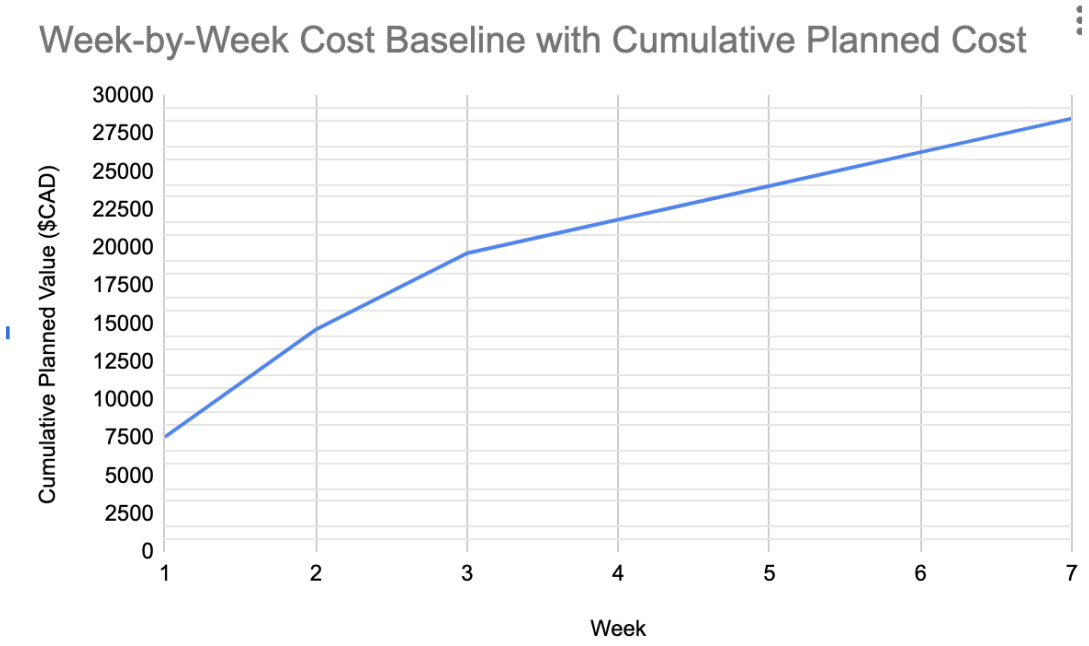


Figure 12: Cost baseline breakdown by week

5.3 Weekly Actual Cost

Table 5: List of actual cost components, broken down by week

Week	Task A AC	Task B AC	Task C AC	Task D AC	Task E AC	Task F AC	Task G AC	Total AC
1	$500	$1000	$1200					$2700
2		$1000	$1200	$100				$2350
3		$1000	$1200	$100	$1400			$3750
4		$1000	$1200	$100	$1400	$2000		$5750
5		$1000	$1200		$1400	$2000		$5600
6		$1000	$1200				$4000	$6200
7							$4000	$4000

Looking at the Actual Cost table above, an outline of realized spending is shown up until currently, which is the end of week 7. The table shows that contrary to the planned value table, the spending is increasing rather than decreasing as the weeks go on. This is due to the increase in development time on the project. When looking at the individual tasks, task B and C have a larger actual cost than the planned value, which is due to those tasks taking longer than the initial 2–3-week timeline. Also, task D and E have an actual cost equal to the planned value. For task F (Unit Testing), the start date was week 4

however the hours of development time per week have almost doubled. And looking at task G (Integration Testing), we conducted integration testing in the final 2 weeks of development, slightly exceeding budget on that task.

5.4 Final Planned Value Analysis: Planned Value vs. Actual Cost

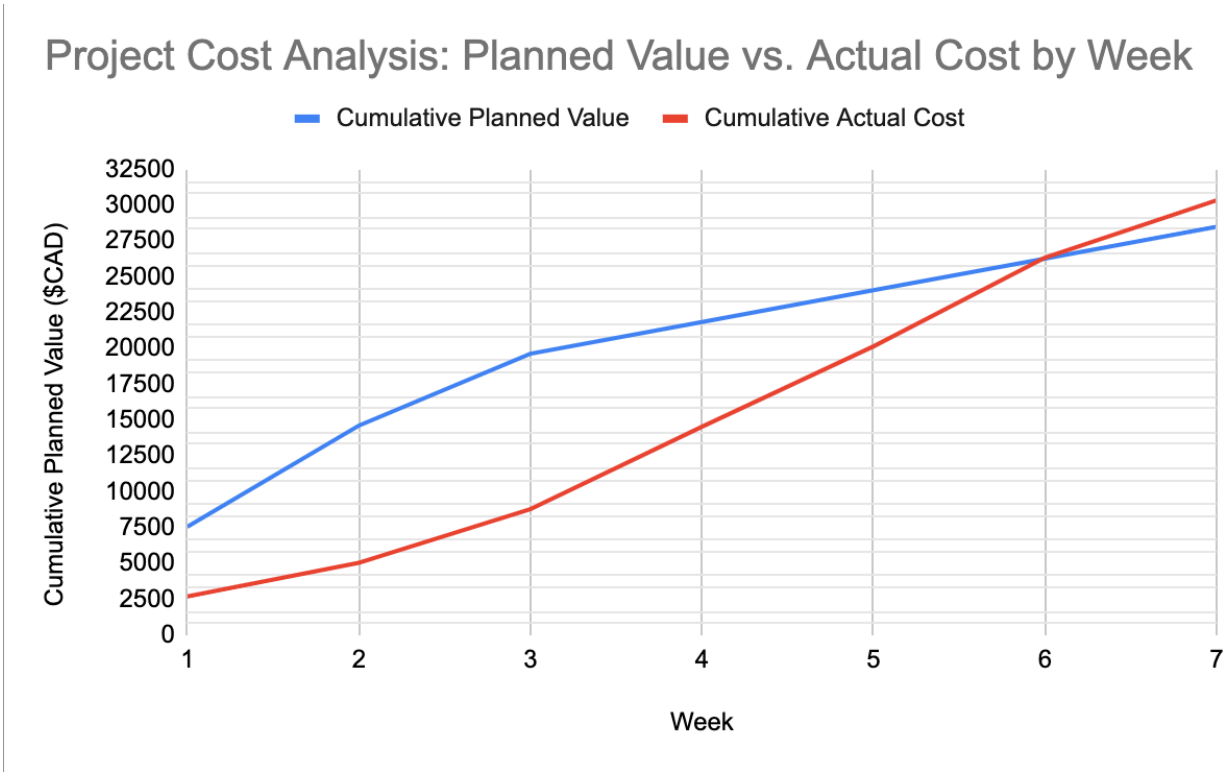


Figure 13: Planned value vs. Actual cost by week

Looking at the chart above, the actual cost overtook the planned cost in the last 2 weeks, which shows that we are exceeding the budget. However, this was crucial development time spent on integration testing which was required in order to test the functionality of component interactions in the robot. The following cumulative figures up until the end of week 7 can be observed below.

Planned Value for all 7 weeks:	**$28500**
Actual Cost for all 7 weeks:	**$30350**

Looking at the numbers above, we are $1850 above budget at the end of our project. As predicted in week 5, due to the fact that we had not initiated the integration testing yet, we ended up exceeding the planned value in the last 2 weeks of development. In week 6 we were exactly on pace and on budget, but unfortunately due to a final week sprint, we exceeded the project budget. Overall, we ended up finishing the project on time, however we were slightly over budget, due to the large increase in development time spent on integration testing in the final 2 weeks.

3.0 Team Member Contributions

3.1 Brendan

Brendan's main contribution was conducting and managing the unit tests and integration tests, as well as aiding in the development of code logic for the ultrasonic distance sensor and the line follower sensor. The unit tests were created in the preliminary stages of the project to determine individual component functionality in isolation. And the integration tests were created to examine the interaction between multiple robot components. The logic determined for the ultrasonic distance sensor was that the robot would operate at full speed until <20cm from an object, where it would slow down to an eventual stop at 10cm from the object. He was also responsible for the cost breakdown and planned value analysis for project cost management, which is explained in detail in the planned value analysis section.

3.2 Ahmed

Ahmed's primary contributions encompassed the design, implementation, and integration of critical hardware and software components for the robot's functionality, excluding the PID controller for the movement module. He was responsible for the calibration and unit testing of all sensors, including the ToF sensors, ultrasonic distance sensors, line follower sensors, and the IMU. This involved developing and fine-tuning sensor calibration code to ensure accurate and consistent readings across various operating conditions.

Ahmed also played a pivotal role in designing the 2D array map logic for the robot's navigation system. The map, quantized into 5 cm blocks, enabled the robot to track unexplored, cleared, obstacle, and boundary regions in real time. He implemented algorithms to dynamically update the map based on sensor inputs, including marking obstacles detected by ToF and ultrasonic sensors and filling boundary rows or columns upon line detection.

Another significant contribution was the development of the UART communication protocol between the Arduino Due and Nano Every. This protocol ensured reliable data exchange, enabling the Event Manager on the Due to send movement commands and receive sensor data from the Nano. Ahmed's implementation included robust error handling to prevent data loss or corruption during communication.

On the hardware side, Ahmed was instrumental in assembling the electrical components. He integrated pull-up resistors for I2C communication, a constant 3.3V power supply from

the buck converter, and a logic level shifter to ensure compatibility between 5V and 3.3V components. His efforts in connecting and powering the sensors and microcontrollers ensured stable and reliable hardware operation.

Ahmed’s contributions formed the backbone of the robot's sensory perception, mapping, communication, and hardware integration, which were essential for the system's overall functionality and success.

3.3 Duncan

Duncan’s main contributions were on the Nano Every library code, the Movement Module, and the hardware circuitry. Many hours went into the Movement Module design, which proved to have many logistical challenges, many resulting from a) the asynchronous motor output feedback (via the wheel encoders) and b) Very non-ideal conditions created by the motors. The Arduino Nano Every codebase was largely designed to be modular support package for the Movement Module, consisting of peripheral configuration files for TCA0, TCB, and ADC, core building blocks of the Movement Module namely the PID Controller and motor related structs/functionality, and additional support structures such as ring buffers and flexible data types.

3.4 Tauheed

My primary contribution to the team consists of three main tasks starting off with the VMA330 IR Obstacle Avoidance Sensor Module. My main objective was to assess if it was an efficient sensor module for our obstacle detection system but after completing testing on it, I had determined we were not getting reliable enough readings due to which we had not implemented it on our snowplow robot. My second task was working on the Line Follower Sensor with which I had completed testing on both the sensors and deduced that one of our sensors occasionally would fail to accurately detect black or white surfaces. The last task was designing a snowplow attachment to be mounted in front of the robot through AutoCAD modeling. The goal with snowplow was to ensure none of the snow cubes would end up under the robot and get it stuck during the demo.

4.0 Final System Architecture

4.1 System Overview

The system design remains similar to the system described in the progress report. The Arduino Due runs the Event Manager, which is the entry point to the system. The main duties of the Event Manager are to process sensor data and send commands to the Movement Module (running on the Arduino Nano Every) as needed. The Line Follower and the IMU are connected to the Nano Every, rather than the Event Manager on the Due. The reason for having the Line Follower feed to Arduino Nano Every is to have the Line Follower as close to the movement module as possible. Figure 14 below provides an outline of the system architecture.

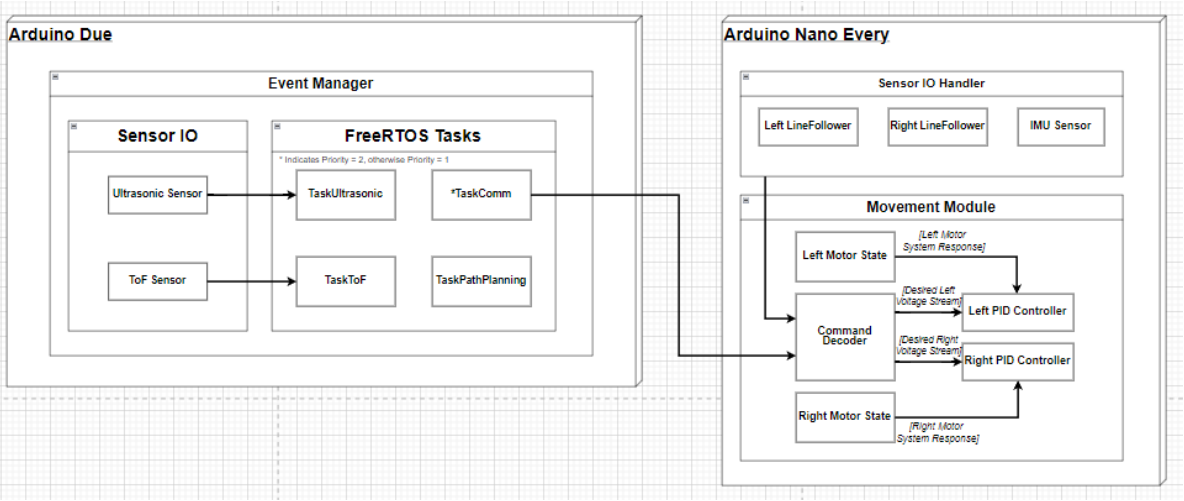


Figure 14: Diagram showing the system overview of the arduino due and nano every

4.2 Event Manager

The Event Manager serves as the central hub for decision-making and control within the system architecture. Running on the Arduino Due, its primary function is to aggregate sensor data, interpret system states, and issue appropriate commands to the Movement Module.

Design Approach

The Event Manager follows a modular approach, integrating various sub-systems to ensure robust and real-time responsiveness. This design includes:

1. **Sensor Integration:** The Event Manager receives data from the ToF sensors, ultrasonic sensors, and coordinate system updates. These inputs allow it to detect obstacles, boundaries, and map the robot's surroundings dynamically.
2. **Command Translation:** Based on sensor inputs, the Event Manager translates navigation decisions into actionable commands for the Movement Module. For example, obstacle detection triggers specific turn or stop commands to prevent collisions.
3. **Error Handling:** In the event of sensor anomalies or system errors, the Event Manager is equipped with a recovery mechanism, such as triggering a system halt or recalibrating the affected components.

Operational Workflow

- **Initialization Phase:** On system startup, the Event Manager initializes all sensors and calibrates the IMU and ToF sensors. It also ensures the obstacle map is reset to its default state.
- **Real-time Control Loop:** During operation, the Event Manager continuously polls sensor data, processes events, and updates the movement trajectory. For instance, if a black boundary line is detected, the Event Manager issues a reverse and reorientation command to prevent further boundary violation.
- **Integration with Movement Module:** Communication between the Event Manager and Movement Module is handled via UART, ensuring low-latency command transmission.

This centralized approach ensures that the robot can adapt to real-world conditions effectively, leveraging sensor fusion for accurate decision-making.

4.3 Movement Module

The Movement Module, implemented on the Arduino Nano Every, is responsible for executing the commands received from the Event Manager. This module directly interfaces with the robot's actuators and the line-following sensors to maintain precise control over movement and positioning.

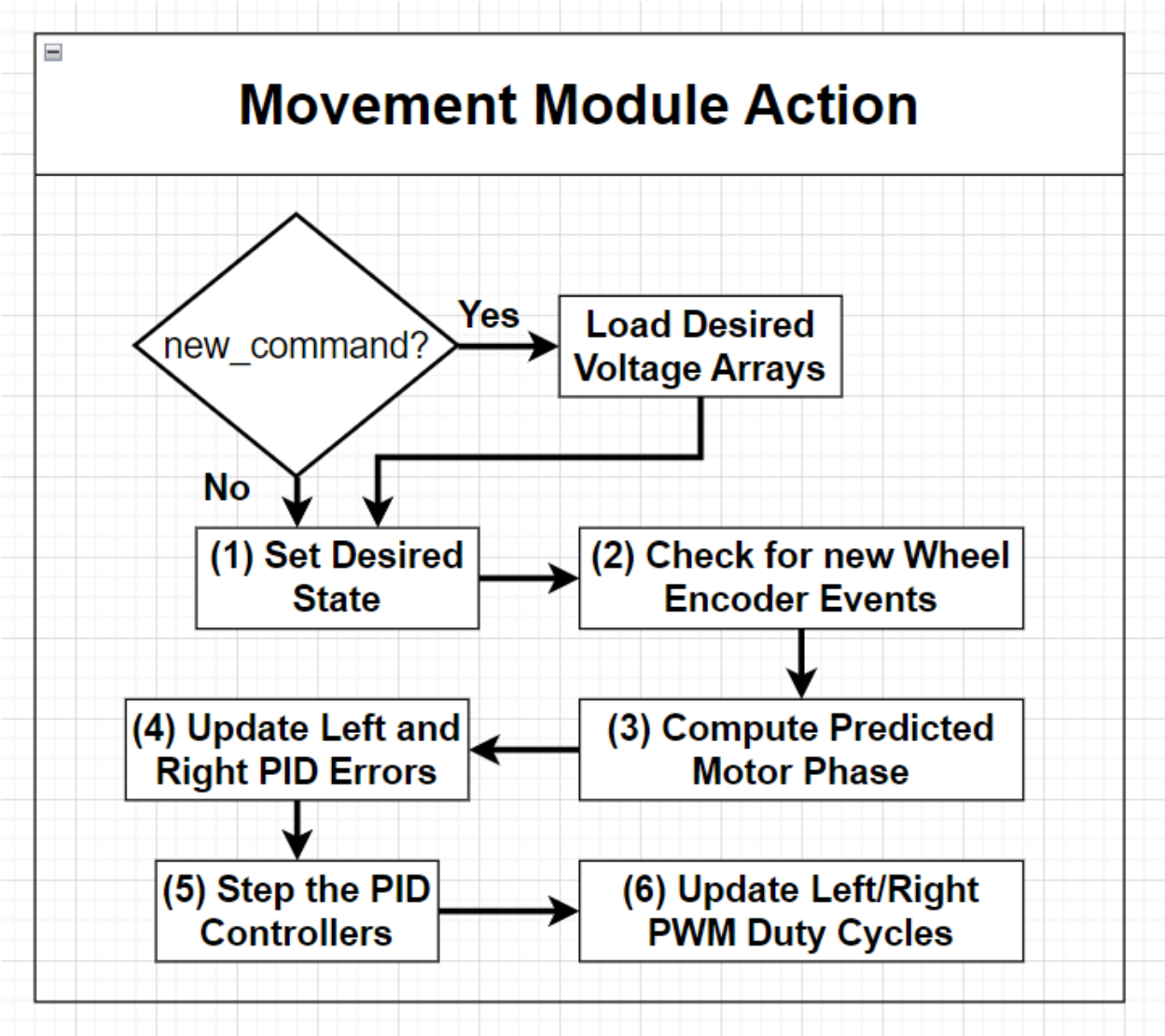


Figure 15: Diagram of task flow in movement module

The movement module's main loop executes periodically, every ten milliseconds, triggered by the TCA0 OVF interrupt. The scheduling period was chosen to balance being able to meet scheduling deadlines while still being sufficiently more responsive than the motors themselves.

4.3.1 Motor Impulse Response

The motor's impulse response characterizes how the motor responds to voltage input. The angular velocity response and the phase response were used relatively interchangeably in the design process.

The angular velocity transfer function takes the form: $G(s) = \frac{\Omega(s)}{V(s)} = \frac{Ks}{\tau s+1}$

The corresponding time domain impulse response is: $g(t) = \frac{K}{\tau} e^{\frac{-t}{\tau}}$,

Where K is the static gain of the motor and τ is the motor's time constant, describing how reactive it is to a change in input.

The motor's phase response can be trivially obtained from the angular velocity response by integrating in the time domain/multiplying by s^{-1} in the Laplace domain.

Phase Transfer Function (s-domain): $H(s) = \frac{K}{Ts+1}$

Phase Impulse Response: $h(t) = K\left(1 - e^{\frac{-t}{\tau}}\right)$

4.3.2 PID Controlled System Diagram

Figure 16 below shows the system block diagram of just the motor.

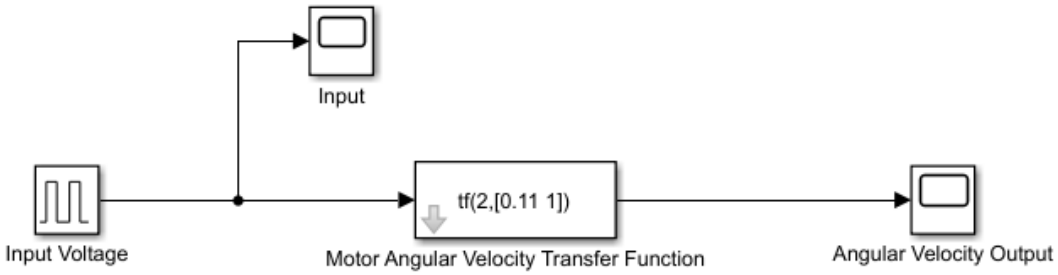


Figure 16: Diagram of the Isolated Motor Transfer Function

After adding the PID block as well as closed-loop control gives us Figure 17 below.

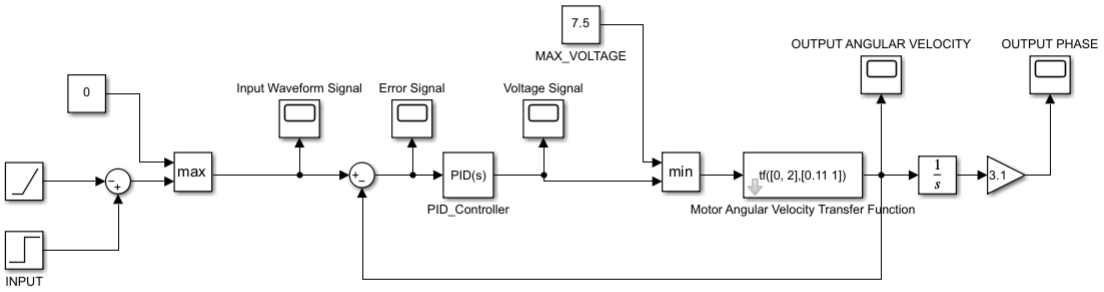


Figure 17: Full PID-Controlled Movement Module Block Diagram

The effect of the closed loop PID control is the resulting output is a smoothed yet still very responsive system output. Figure 18 below provides a side-by-side view of the system input (desired) angular velocity waveform (top left), the output angular velocity waveform (bottom left) and its integral, the total phase displacement (on the right)

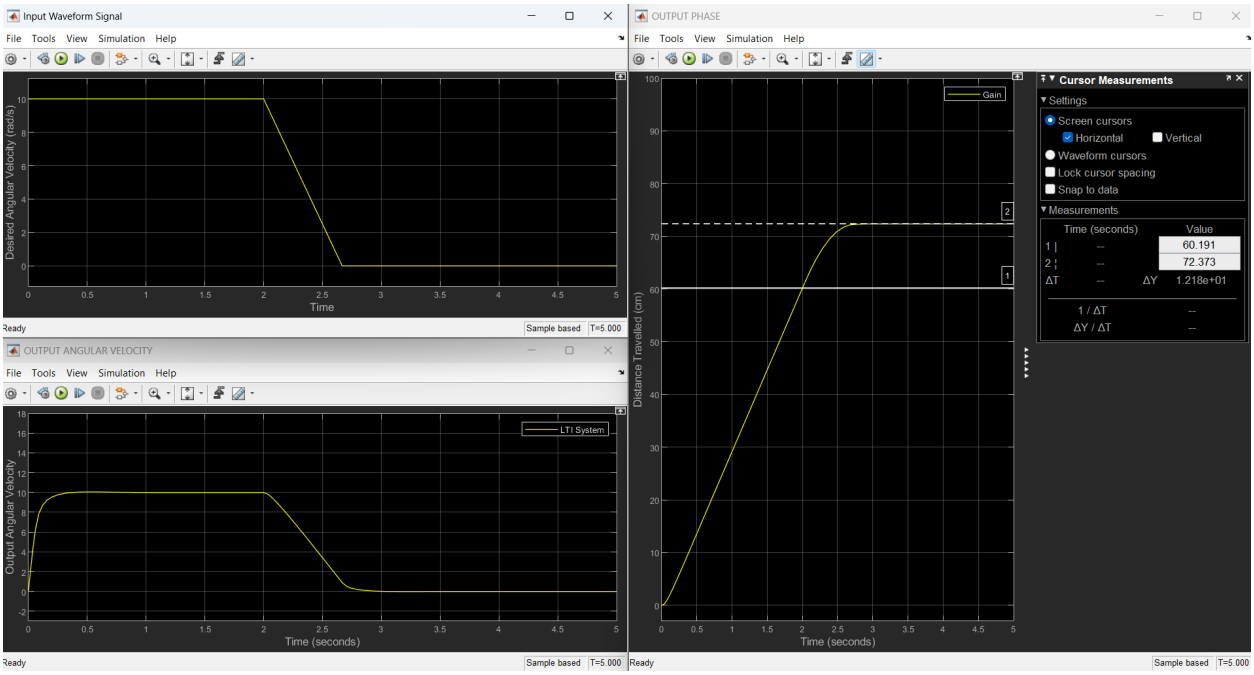
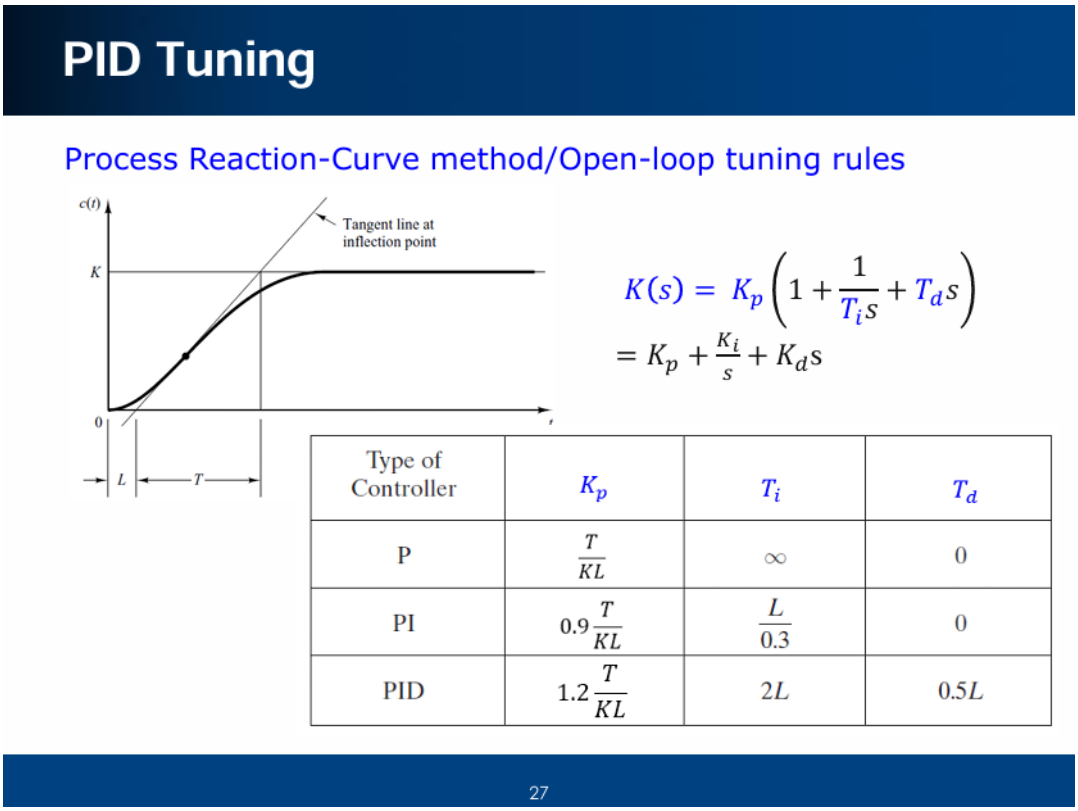


Figure 18: Simulated example of the system's output angular velocity (bottom left) and phase (right) waveforms, resulting from the top left input 'desired' waveform

4.3.3 PID Controller Tuning

The PID controllers were tuned via Ziegler-Nichols open-loop tuning rules [1, Chp 8.2, pg 568]. Zielger-Nichols tuning rules are intended to provide a reasonable 'starting point', which can then be refined. Figure 19 [5] summarizes Zielger-Nichols PID tuning rules.



Type of Controller	K_p	T_i	T_d
P	$\frac{T}{KL}$	∞	0
PI	$0.9\frac{T}{KL}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{KL}$	$2L$	$0.5L$

Figure 19: Summary of Ziegler-Nichols PID Open-Loop Tuning Method

The Ziegler-Nichols tuning convention is also implemented in our Arduino Nano Every codebase. Figure 20 shows the snippet of code defining the PID Controller struct and macro definitions for their coefficients.

```
15
16  #ifndef PID_CONTROLLER_H
17  #define PID_CONTROLLER_H
18
19  #define K 2
20  #define Tau 0.11
21
22  // Macro definitions for the PID tuning parameters
23  #define K_P(DT) 1.2 * Tau / (K * DT)
24  #define K_I(DT) 0.6 * Tau / (K * DT * DT)
25  #define K_D(DT) 0.6 * Tau / K
26
27
28  typedef struct pid_controller_struct {
29      double Kp;
30      double Ki;
31      double Kd;
32
33      double integral;
34      double proportional;
35      double derivative;
36
37      double output;
38
39      double desired_state;
40      double prev_desired_state;
41
42      double error;
43      double previous_error;
44
45  } pid_controller_t;
46
```

Figure 20: PID Controller struct, implemented in lib/pid_controller.h

4.3.4 Implementing Synchronous Updates with Asynchronous Sensor Feedback

One of the biggest challenges that presented with building movement module was that the feedback from the wheel encoders is asynchronous. There were initial attempts to work with an asynchronous movement module loop, but it became quickly apparent that the movement module needed to execute in fixed time intervals. The biggest flaw introduced by having the movement module execute on wheel encoder events was that our movement module was now reliant on the system moving to work, and its responsiveness was now directly proportional to the system's speed, which is the variable it is supposed to be controlling in the first place. It suffices to say that the movement module needs to execute in fixed time intervals to be effective and reliable.

To have the movement updating in constant time intervals, it was decided that the system would predict the velocity/phase of the motors, using the supply voltage and a statically stored array of the motor's phase and angular velocity impulse responses. Figure 21 below shows part of the header file that defined the motor's impulse response array.

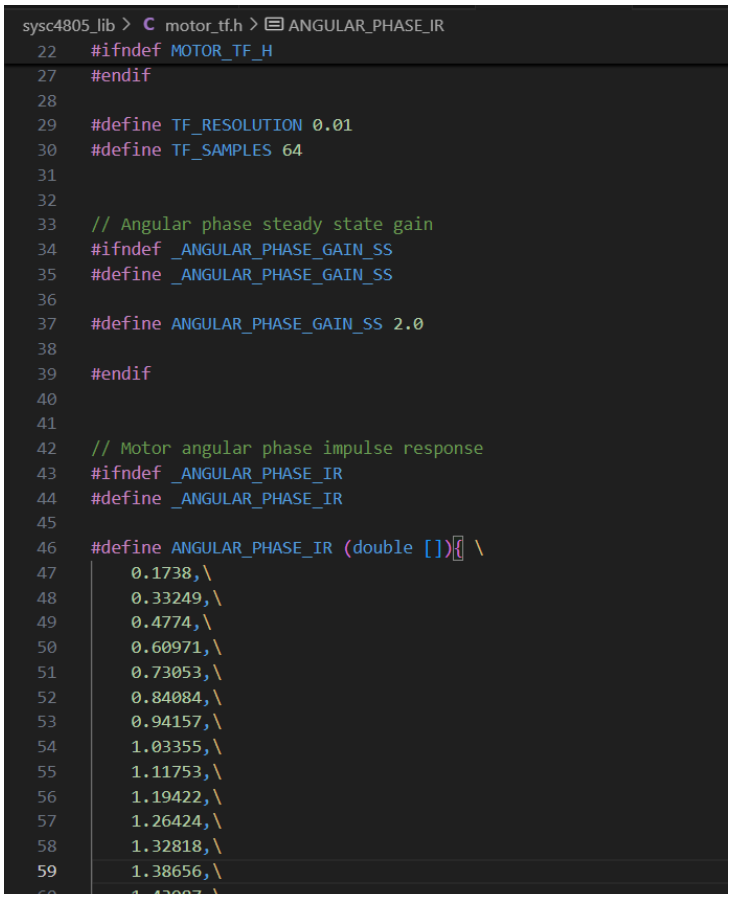


Figure 21: Screenshot of code for angular phase impulse response

Introducing the new motor response prediction algorithm added significant overhead to an already large task. The system now, in effect, has two streams of motor velocity:

1) The predicted angular velocity waveform via the impulse response and motor voltage stream
2) The measured output from the wheel encoders

With multiple sources comes the added challenge of interpolating them, which proved challenging, and we unfortunately were not able to figure it all out in time for the demo.

4.3.5 Motor Output Computation

The initial idea for calculating the motor output assumed a constant supply voltage, which was attenuated by the duty cycle. Although we figured there would be some loss in accuracy, the ease of implementation and possible computational shortcuts in specific cases made it worth looking into. Unfortunately, we did not foresee the effect the motors ended up having on the supply voltage. The motors being driven cause a transient drop in voltage that was too large to ignore, as well as adding significant noise to the system's ground potential. To measure the transient voltage drop due to the motor, the supply voltage was sampled at 100Hz and observed the effect of driving the motors. Figure 22 below shows the output log of this experiment.

```
[17] Voltage Reading = 6.17
[18] Voltage Reading = 6.17
[19] Voltage Reading = 6.17
[20] Voltage Reading = 6.18
[21] Voltage Reading = 4.99
[22] Voltage Reading = 5.24
[23] Voltage Reading = 5.37
[24] Voltage Reading = 5.53
[25] Voltage Reading = 5.60
[26] Voltage Reading = 5.62
[27] Voltage Reading = 5.69
[28] Voltage Reading = 5.69
[29] Voltage Reading = 5.72
[30] Voltage Reading = 5.73
```

Figure 22: Voltage reading output of motor sampled at a supply voltage of 100Hz

In light of the non-ideal conditions created by the motors, we decided to add functionality to sample the voltage in real time into a buffer. Since the impulse response of the motor is known, storing a buffer of voltage samples gives us the system input and the system impulse response, allowing us to compute the time domain output of the motor using discrete convolution.

Final Movement Design

The final movement module design is optimized for modularity and real-time responsiveness. Key components include:

1. **PID Control System:** The module employs a dual-PID control system to regulate the angular velocities of the left and right wheels. This ensures smooth and precise maneuvers, whether navigating straight paths or making non-sharp 110-degree turns.
2. **Direct Line Follower Integration:** The line-following logic is directly implemented on the Nano Every. This proximity reduces latency in detecting and responding to boundary lines. Data from the line sensors is processed locally, and corrections are made to the wheel speeds to keep the robot aligned.
3. **Obstacle Avoidance Handling:** The Movement Module works in tandem with the Event Manager by implementing immediate actions, such as stopping or executing a predefined turn upon receiving obstacle detection commands.

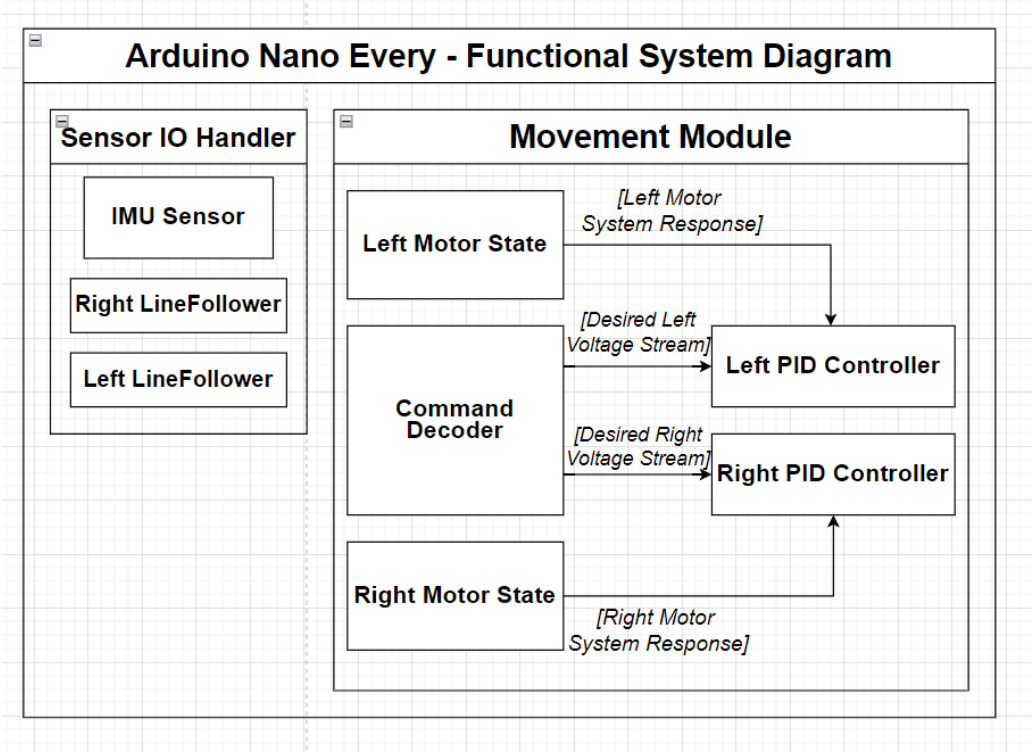


Figure 23: Functional system diagram of arduino nano every with sensor IO handling and movement module

4.0 Control Charts

This section evaluates the performance and reliability of the robot's critical functionalities, including line detection, obstacle avoidance, and speed regulation, based on the control charts. The charts depict results from multiple test iterations, providing quantitative and qualitative insights into the system's behavior and success rates.

4.1 Line Detection

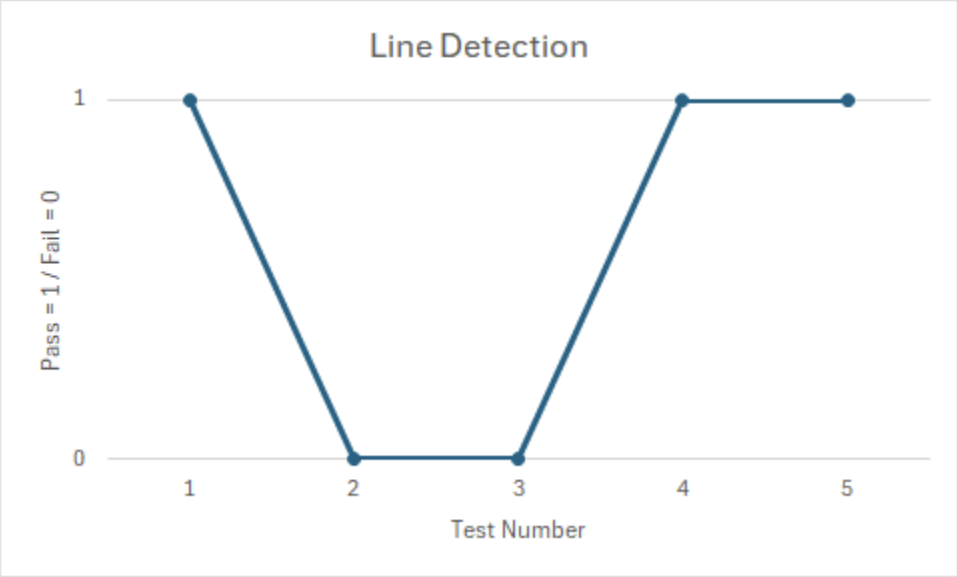


Figure 24: Control chart for the line detection test showing pass or fail for the 5 trials

The line detection performance is illustrated in the chart, which plots test outcomes over five iterations. A binary metric is used, where a pass is indicated by 1, and a failure by 0.

- **Observations:**
  The robot successfully detected the line in the first and last two tests but failed during the second and third iterations. This inconsistency could be attributed to environmental factors such as uneven lighting, sensor misalignment, or delays in the Event Manager's response time.
- **Key Insights:**
  - The overall success rate of 60% highlights room for improvement in robustness.
  - Failures often occurred when the robot approached the boundary at non-perpendicular angles, reducing the line sensor's effective detection range.

- **Proposed Improvements:**
    - Sensor recalibration to ensure consistent sensitivity across different angles and lighting conditions.
    - Implementing a confidence threshold by aggregating multiple sensor readings to filter out anomalies.
    - Adding additional line-following sensors or repositioning the existing ones to increase coverage.

4.2 Obstacle Avoidance

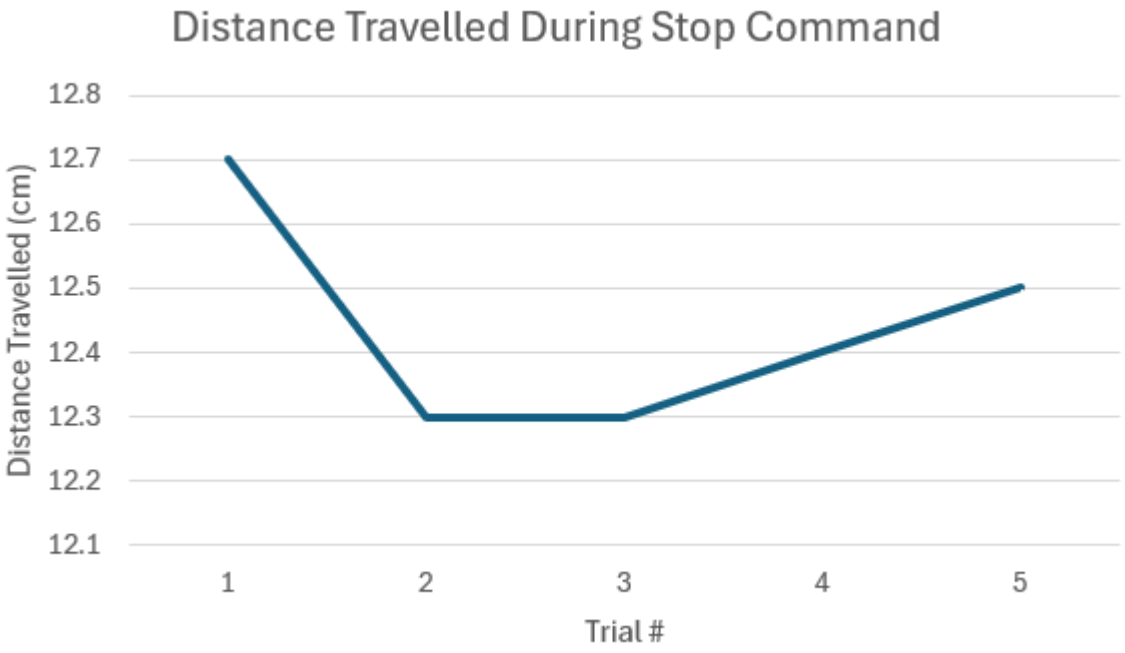


Figure 25: Control chart for the obstacle avoidance test showing distance travelled during stop command

The chart visualizes the distance traveled by the robot after receiving a stop command across five trials.

- **Observations:**
    - The stopping distance varied slightly across iterations, with the shortest distance at 12.3 cm (trials 2 and 3) and the longest at 12.7 cm (trial 1).

    - This variability could be linked to wheel inertia, surface friction, or minor delays in command execution by the Movement Module.

- **Key Insights:**
    - The robot demonstrates reasonable consistency, with deviations within a 0.4 cm range across all tests.
    - The PID controller on the Movement Module appears to regulate motor deceleration effectively.
- **Proposed Improvements:**
    - Fine-tune the PID control parameters to minimize overshoot during deceleration.
    - Conduct additional trials on various surfaces to ensure consistent performance in diverse environments.
    - Integrate preemptive braking logic when the robot approaches obstacles or boundaries.

4.3 Speed

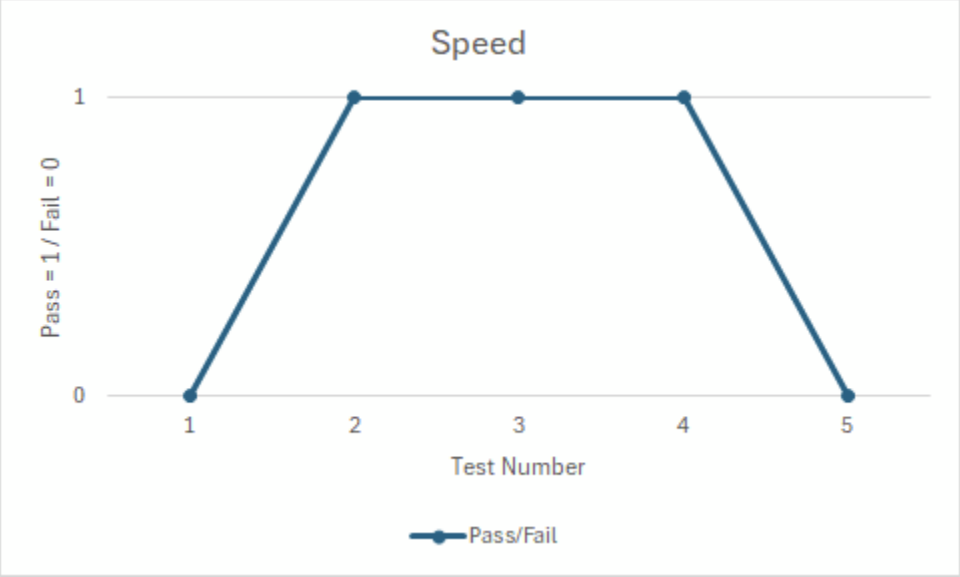


Figure 26: Control chart for speed testing showing pass or fail for the 5 trials

Figure 26 evaluates the system’s ability to maintain the target speed within acceptable margins. The tests assessed the robot's speed stability under different load and surface conditions.

- **Observations:**
    - The robot maintained its target speed successfully in tests 2 through 4. However, it failed to achieve consistent speed in tests 1 and 5, where external factors such as battery fluctuations or uneven terrain may have played a role.
    - Failures indicate instances of either under-speeding or over-speeding beyond the acceptable ±1 cm/s margin.
- **Key Insights:**
    - The speed regulation mechanism demonstrates a 60% success rate, consistent with the challenges faced in maintaining precise control.
    - The wheel encoder feedback loop effectively adjusted motor velocities in most scenarios.
- **Proposed Improvements:**
    - Implement dynamic voltage compensation to account for battery level changes.
    - Extend PID control tuning by incorporating test scenarios with varying weights and inclines.
    - Add predictive algorithms in the Event Manager to adjust speed proactively based on environmental factors.

These control charts provide critical feedback for iterative development and fine-tuning of the robot’s systems. By addressing the identified challenges and implementing the proposed improvements, the overall reliability and performance of the robot can be significantly enhanced.

Extended Testing:
Current tests were conducted under controlled conditions. Expanding the scope to include real-world scenarios, such as uneven terrains or varying lighting, will provide insights into the system's adaptability and robustness.

Redundancy and Fault Tolerance:
Introducing redundant sensors and fault-tolerant algorithms can help mitigate the impact of sporadic failures, particularly in line detection and speed regulation.

5.0 Integration & Customer Testing Results

For our integration tests, we tested multiple robot components' interactions with each other in controlled conditions. These integration tests allowed us to examine and verify the functionality of multiple components working together to achieve a specific task. The integration tests are broken down into 3 main categories, movement, navigation and sensor communication within the robot. For the movement category, the main components involved are the wheel encoders and the motor. The tests for movement will involve testing the speed, direction and turning of the robot. For the next test category, the navigation, the main components under test will be the IMU and the due/nano. These tests will be examining the robot's ability to accurately map the grid and keep track of its position and orientation within the grid. And finally for the third phase of integration testing, we will be testing the sensor communication within the robot. These tests will involve the obstacle detection sensor with the motors, the line detection sensor with the motors, and finally the IMU with the due and motors. The goals of these tests respectively are to test the robot's ability to stop successfully when obstacles or lines are detected. For customer testing, the robot was placed in acceptance test conditions in order to pass its outlined requirements. An additional test was done on the robot chassis to ensure it was still within the given size constraint of width x length x height of 226x262x150mm after our snowplow was attached to the chassis.

5.1 Movement (Wheel Encoders + Motors)

The testing for the robot movement will include 3 stages:

1. Speed: Can the robot maintain a speed within a 1cm/s error

Test: The test for speed will be operating the robot to move at 20cm/s. The test will pass if the robot can maintain a speed of 20cm/s +- 1cm/s. If the robot's speed drops below 19cm/s or above 21cm/s, the test is a failure.

2. Direction: Can the robot maintain a straight line without veering off path

Test: To test straight line operation, a makeshift boundary (line of tape) will be set up, and the robot will be setup right next to it and will move straight forward for 2.5m. The test will pass if the robot maintains within 2cm of the tape but fails if it crosses over the tape or moves away from the tape 2cm or more.

3. Turning: Can the robot accurately turn left and right (90 degrees)

Test: To test turning, the robot will be tested at attempting a 90 degree turn in both left and right directions. The test will pass if the robot is able to turn in a range of 88 degrees to 92 degrees (98% accuracy). If the robot turns more than 92 degrees or less than 88 degrees in either direction, the test is a failure.

Table 6: List of Integration tests for each test in the Movement phase

Int. Test Case ID	Integration Test Type	Components Tested	Test	Status	Result
IT-001	5.1.1 Speed	Wheel Encoders & Motors	Test 1 Above	Completed	The test was successful, the robot was able to maintain a speed of about 20cm/s without deviating over or under by 1 cm/s.
	Code for IT-001 was referenced from the Lab 3 Manual [2]				
IT-002	5.1.2 Direction	Wheel Encoders & Motors	Test 2 Above	Completed	The test was successful, the robot was able to operate in a straight line without deviating over 2cm from the tape.
	Code for IT-002 was referenced from the Lab 3 Manual [2]				
IT-003	5.1.3 Turning	Wheel Encoders & Motors	Test 3 Above	Completed	The test was successful, the robot turned within the acceptable 88 to 92 degrees threshold placed as pass criteria.
	Code for IT-003 was referenced from the Lab 3 Manual [2]				

5.2 Navigation (Mapping and Positioning)

The testing for navigation will include 2 stages:

4. Mapping: Can the robot accurately map a rough grid of the area in which it clears

Test: A makeshift grid of tape will be made, and the robot will converse around the grid for 1 minute and must accurately map the border of the grid. The test will pass if the robot can accurately map the grid with 98% confidence, meaning a +- 2% margin on dimensions of

grid. The test will fail if this confidence margin of 98% is not met, or the robot fails to create the map in the first place.

5. Positioning: Does the robot know it's position and orientation within this grid

Test: To test the positioning, the robot will be placed in a makeshift testing grid and will navigate to random points within the grid. The test is to have the robot communicate in Arduino its current coordinates within the grid and it/s orientation XYZ. The test will pass if the readings are approximately 95% accurate compared to the expected (observed) data. The test will fail otherwise.

Table 7: List of Integration tests for each test in the Navigation phase

Int. Test Case ID	Integration Test Type	Components Tested	Test	Status	Result
IT-004	5.2.1 Mapping	IMU & Due	Test 4 Above	Completed	The test was successful, the robot was able to accurately map the grid within the given constraint of +- 2%. Further development and strategy was developed to fix this prior to the acceptance testing.
	Code for IT-004 was referenced from the Lab 2 Manual [3]				
IT-005	5.2.2 Positioning	IMU & Due	Test 5 Above	Completed	The test was successful, the robot was able to utilize the gyroscope to successfully track its XYZ orientation with under 5% error as outlined in our success criteria.
	Code for IT-005 was referenced from the Lab 2 Manual [3]				

5.3 Sensor Communication with Robot

Each sensor will be calibrated within the unit testing, but will need to communicate with the robot to perform certain tasks:

6. Obstacle Detection -> Speed: Can the robot accurately detect the distance from an object and decelerate accordingly

Test: An obstacle will be set up 1 meter from the robot, and the robot will move at 20cm/s towards the obstacle. The robot must decelerate smoothly as it gets closer to the obstacle and comes to a stop about 5cm from the obstacle. The test pass/failure will be observed as so: If the robot decelerates relatively smoothly proportional to its distance from the obstacle, and stops within a range 4.5 to 5.5 cm, the test will pass. If the deceleration is sudden or the robot drives into the obstacle, the test will fail.

7. Line Detection -> Speed: Can the robot slow down or stop if a line is detected closely in front of it.

Test: A line of tape will be set up to simulate the border. The robot will move towards the tape and the line must be detected by the line detection sensor and stop the robot. If the robot drives over and past the line, the test will fail. If the robot stops at the line, and does not cross it with its back wheels, the test is successful.

Table 8: List of Integration tests for each test in the Sensor Communication phase

Int. Test Case ID	Integration Test Type	Components Tested	Test	Status	Result
IT-006	5.3.1 Obstacle Detection	Ultrasonic Distance Sensor & Motors	Test 6 Above	Completed	The test was successful, the robot stopped between the criteria threshold of 4.5cm to 5.5cm successfully.
	Code for IT-006 was referenced from the Lab 2 Manual [3]				
IT-007	5.3.2 Line Detection	Line Detection Sensor & Motors	Test 7 Above	Completed	The robot successfully stopped at the line before its back wheels crossed over the black tape, so the test passed.
	Code for IT-007 was referenced from the Lab 1 Manual [4]				

5.4 Customer Testing (Acceptance)

8. Grid -> Navigation-> Obstacle/Line Detection-> Snow Removal (Final Test): Can the robot maneuver itself around its constructed grid maintaining knowledge of its position and orientation, remove snow cubes and avoid leaving line boundaries and avoid obstacles.

Test: The test for grid navigation will be the final test in our testing plan, the customer test. Integrating all our previous calibration, the robot will move around the grid with cubes, following our constructed navigation plan. As the robot navigates it must detect lines to keep itself within bounds, must detect obstacles and avoid hitting them, and will be displaying its position and orientation to the Arduino terminal whilst navigating. The passing criterion for this test is that the robot stays in boundaries (wheels within 5cm of line), avoids obstacles (stops and turns when detected), and can cover the area in our navigation plan. Additionally, the robot will be placed under a 5-minute time limit. The customer test will fail if any of these criteria are not met in tandem.

Table 9: Customer acceptance testing results

Cust. Test Case ID	Customer Test Type	Components Tested	Test	Status	Result
CT-001	5.4.1 Final Test	IMU, Ultrasonic & ToF Sensor, Line Detection, Due, Nano and Motors (All Sensors)	Test 8 Above	Completed	The customer testing was successful days before the demo and without the addition of the PID controller. However, due to issues regarding the PID controller, the customer test failed during the Lab 11 demo. Without movement, none of the other test criteria could be evaluated during our Lab 11 Demo.

To summarize customer testing, our implementation was unsuccessful during the Lab 11 demo. This was due to many factors regarding the issue with power to the motors, as well as significant noise in sensor readings. During initial tests, our line detection, ToF and

ultrasonic distance sensors were successful in providing good readings. Even in combination with the motors in integration testing, the sensors were able to work in tandem with the robot movement.

5.5 Plow Design

Table 10: List of Integration tests for each test in the Sensor Communication phase

Int. Test Case ID	Integration Test Type	Components Tested	Test	Status	Result
IT-006	5.5.1 Robot Dimensions	Robot Chassis with Snow Plow	Within maximum size constraint of WxLxH = 226x262x150 mm	Completed	Success: Robot size WxLxH was 176x232x79.5mm

The finalized plow design is shown in Figure 27 below. As the snow cubes are 20mm and the vehicle's ground clearance is 15mm, our design would only reach ground clearance as it would still be able to clear the cubes. As there is a chance of cubes piling up during operation our team decided to increase the height to at least 25mm and keep a ground clearance of 5mm. The plow design also includes two ledges on both sides, which will protrude at an angle of approximately 30 degrees with a length of 20mm. This design choice was made as when the vehicle turns it should still be able to keep the blocks within the plows channel preventing the cubes from getting displaced from the side.

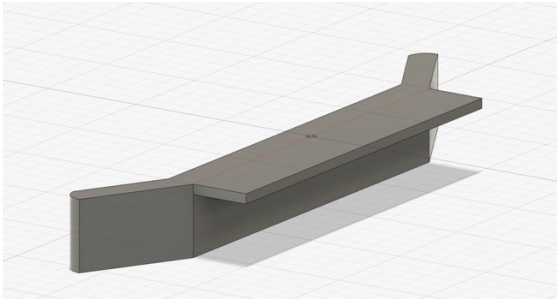


Figure 27: AutoCAD Representation of snow plow design, left diagonal view

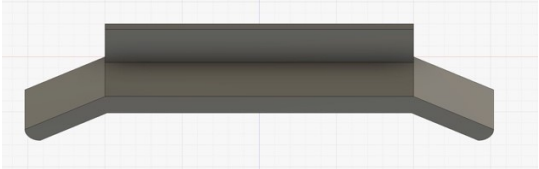


Figure 28: AutoCAD Representation of snow plow design, front view

For attaching the plow module, we had two main options, the first of which was creating two protrusions under the top section of the plow module that will insert into the vehicle's open slots on the front side of the lower deck shown in Figure 29 below. This ensures the plow will be stable, preventing any movements that might occur due to moving the cubes. The second option was to have two holes within the top section of the plow module and use zip ties to attach the module to the vehicle's open slots on the front side of the lower deck. This option was kept as 3D-printed objects tend to be brittle and may break apart under stress like in the case of our first option where when we insert the two protrusions onto the vehicle there is the risk of cracks and fractures.

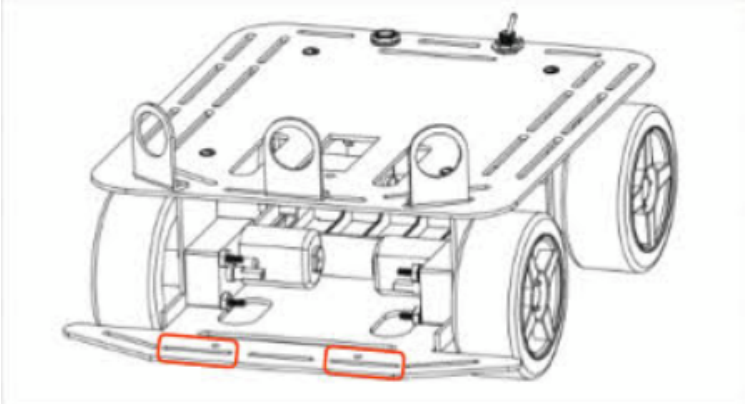


Figure 29: Open slots within the lower deck of the chassis.

6.0 Final GitHub Repository

GitHub repository link --> *https://github.com/SYSC4805/project-l3-g5.git*

References

[1] K. Ogata, *Modern Control Engineering, 5th ed*. New Jersey: Pearson Education, 2010

[2] M. Taha, *Computer Systems Design Lab (SYSC4805) Laboratory Manual*, Lab 3: µC Communications, Motor Driver Board, and Wheel Encoder, Dept. of Systems and Computer Engineering, Carleton University, Fall 2024.

[3] M. Taha, *Computer Systems Design Lab (SYSC4805) Laboratory Manual*, Lab 2: Ultrasonic, ToF Distance, and IMU Sensors, Dept. of Systems and Computer Engineering, Carleton University, Fall 2024.

[4] M. Taha, *Computer Systems Design Lab (SYSC4805) Laboratory Manual*, Lab 1: Intro, AD2, Line Detection, and Obstacle Avoidance Sensors, Dept. of Systems and Computer Engineering, Carleton University, Fall 2024.

[5] S. Chao, *Automatic Control Systems I (SYSC4505) Lecture 12: PID Controller Tuning*, Dept. of Systems and Computer Engineering, Carleton University, Winter 2024.