

Term 3 Project

Sofiya Flenova, Tara Kasayapanand, Ted Tang, Minqi Zhu

May 2025

1 Abstract

We present a six-wheeled ground robot designed to autonomously navigate a multi-terrain obstacle course. The robot is engineered to handle challenges including line following, stair climbing, treadmill traversal, rough terrain, bridging surface gaps, and descending via a zipline. We will be working on robust mechanical design, sensor integration, and adaptive control system for this robot.

2 Mind-map and work breakdown

Before starting the project, it was helpful to outline the main components using a mind map to identify the most effective way to distribute the work. Although all team members contributed to every section, we established a structure where each area had a designated lead responsible for overseeing and completing the task. The rest of the team provided consistent support and assistance as needed throughout the process.

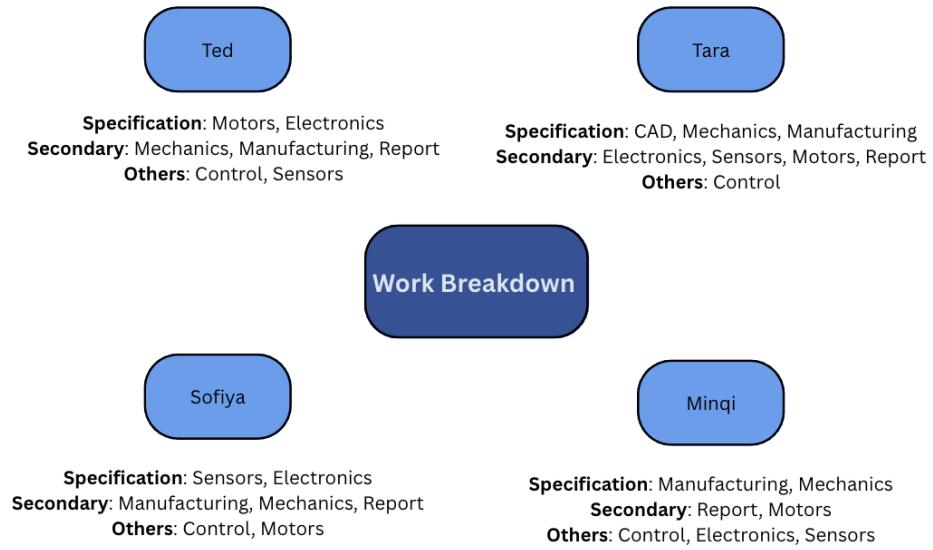


Figure 1: Work breakdown.

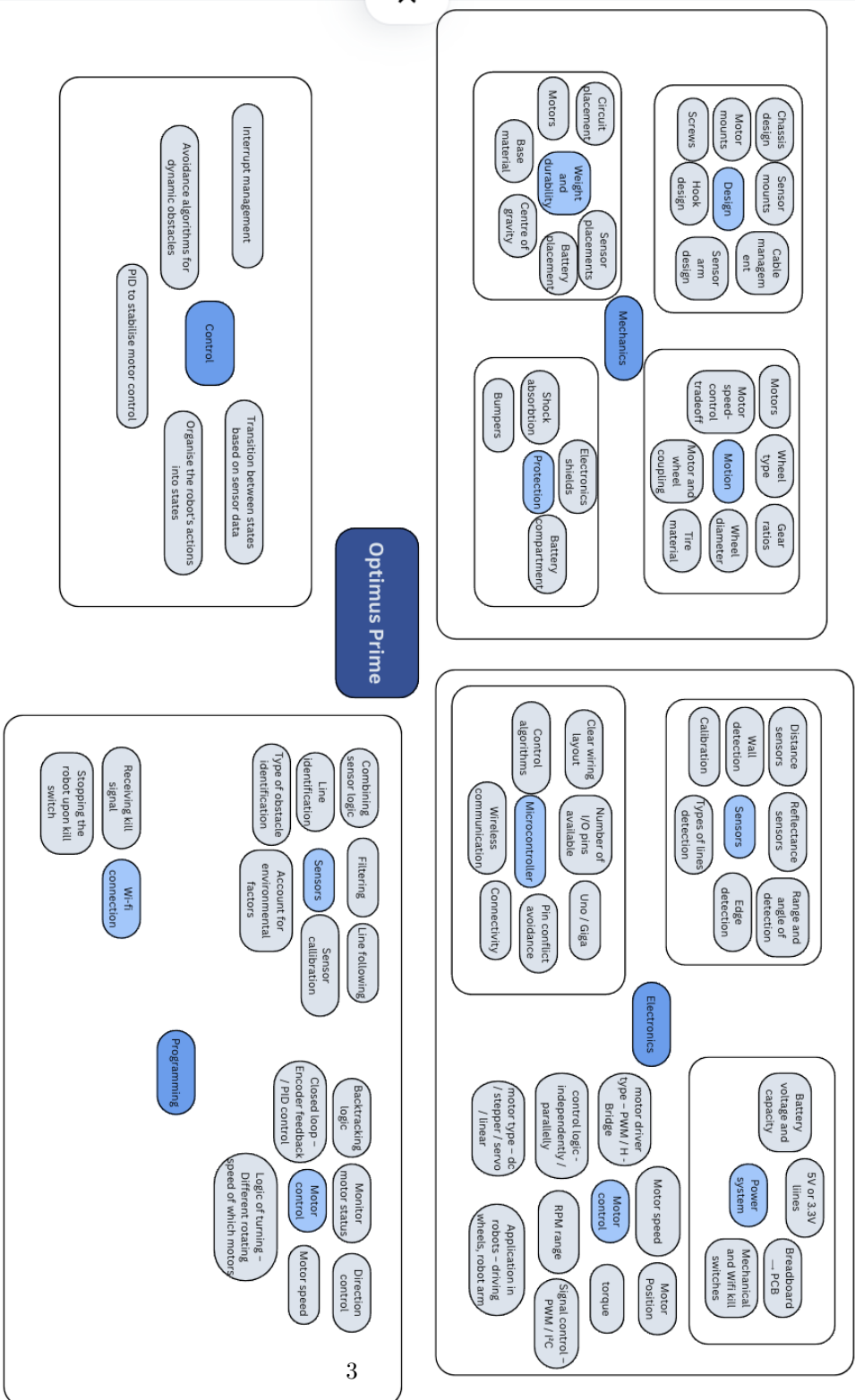


Figure 2: Mind-map.

3 Bill of Materials

Component Name	Qty	Purpose
Mechanical Bill		
Printed wheel	2	Custom thick front pair wheel
Plastic D-type wheel	4	Back four wheels
Rubber tire tracks	6	For more friction and traction
609-2RS Bearing	4	Rocking mechanism
Hook	2	Clamp onto rope for Floor is Lava and zipline
Gear 25mm OD	2	Hook mechanism
M2 10mm screw	24	Mounting motors
M2 12mm screw	2	Mounting micro servo
M2 wash	2	Mounting micro servo
M2 nut	2	Mounting micro servo
M2 self tapping screw	2	Mounting micro servo
M3 12mm screw	3	Attaching arm, gear, and servo
M3 20mm screw	1	Attaching arm to gear
M3 wash	3	Attaching arm, gear, and servo
M3 nut	5	Attaching arm, gear, and servo and adding buffer
M3 self tapping screw	4	Attaching square to secure and stabilise front frame
M4 self tapping screw	11	Fasten platform to base [2] and mounting gear of right hook [1] and mounting hook servo [6] and connecting hook to extender rod [2]
Electrical Bill		
IG220019X00015R DC Motor	6	Individual motor control
DSS-M15S Servo	1	270°servo to power hooks
FS90-C Micro servo	1	Micro servo to move reflectance sensors
Button	1	Mechanical kill switch
QTR-HD-09RC Reflectance Sensor 9x1	2	Line-following
Ultrasonic Distance Sensor (HC-SR04)	1	Wall and obstacle detection
GP2Y0A51 Distance Sensor	2	Side distance sensor
Arduino Giga R1 WiFi	1	Microcontroller
Motoron M3S55 Motor Shield	2	Individual motor control
Ansmann 10.905V Lithium-Ion Battery 3.5Ah	1	Power source

4 Manufacturing process

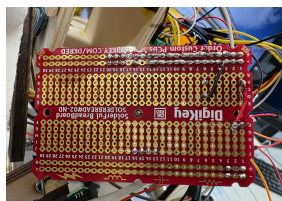
During the manufacturing, which was the area of the project that everyone worked on, we utilised numerous equipment in the lab. That included laser cutters, Prusa 3D printers, drills, screws, soldering irons, multimeters, clamps,

sandpaper, saws and many more. Since our robot was primarily constructed from wood, we also relied heavily on adhesives such as super glue, hot glue and wood glue to secure parts effectively. Below are some photographs that may give more insight into the process visually. (Figure 3)

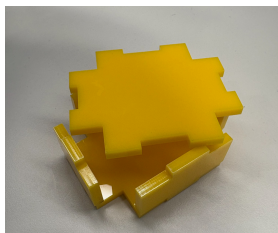
5 Approach to obstacle course

From the obstacle course we were provided, we have outlined some key elements that we had to think about in order to successfully approach each of the sections.

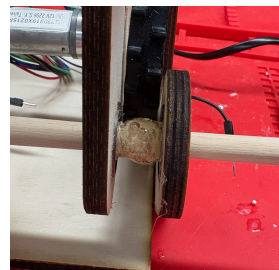
1. Line following and wall avoidance– We assessed the number of distance and reflectance sensors needed, as well as the optimal configuration for their placement to achieve accurate line tracking. Initially, we decided to position two reflectance sensors on the underside of the robot, near the front, arranged in an inverted “T”. We also decided on using three distance sensors located at the front, left, and right sides of the robot. These would ideally be mounted as high as possible to minimise the risk of misidentifying course obstacles as surrounding walls.
2. Step climbing – The main concern here was the choice of wheels and motors. The wheels would ideally be thick and large, yet as lightweight as possible. The motors, particularly the front pair, would need to be sufficiently powerful. However, we also had to consider the limitations of the available power supply. Moreover, more powerful motors typically come with increased weight, which we aimed to minimise to preserve overall mobility. Additionally, we had to consider the fact that if the reflective sensors were too close to the ground, there would be a chance they would bump into the steps.
3. Rough terrain - here the requirement would have to be quite similar to those of stair climbing. Durable, lightweight, strong motors and large wheels.
4. Floor is lava - we needed to think of a way to carry the robot from one side of the gap to another. We have realised that if the robot is sufficiently long, then it will be able to overcome the opening by simply driving through it. However, we still chose to design a hook mechanism to serve as a support at the top of the structure. The hook itself would not be motorised or used to drive the robot forward. Instead, its purpose is to reduce the load on the wheels by providing additional stability.
5. The ramp and treadmill - we have realised that the primary challenge for the whole third section of the obstacle course would be overcoming resistance, requiring significant power. Therefore, with our selected motors, we anticipate that this will not be the area of greatest concern.



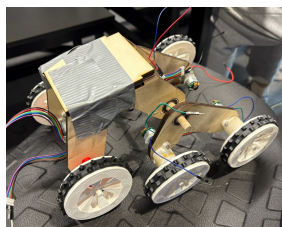
(a) Soldered PCB.



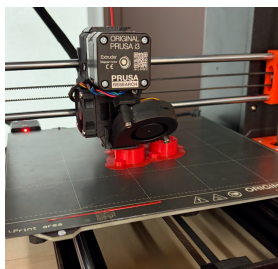
(b) Acrylic battery compartment.



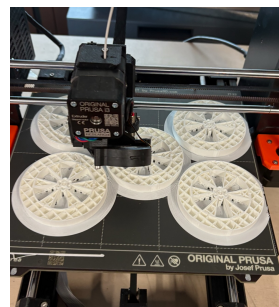
(c) Use of hot glue for robot stability.



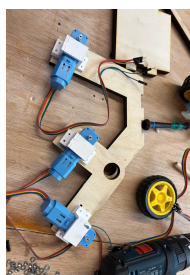
(d) Use of tape in early prototype stages.



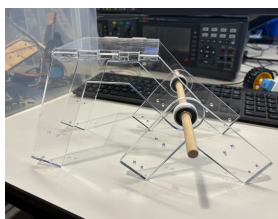
(e) Motor mount 3D printing process.



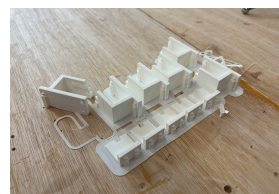
(f) Wheels 3D printing process.



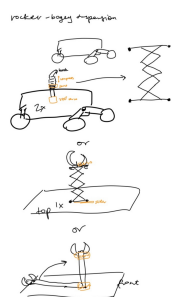
(g) First prototype chassis manufacturing process.



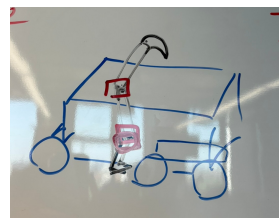
(h) Early acrylic prototype.



(i) 3D Printed early motor mounts.



(j) Hook - initial ideas.



(k) Hook - placement ideas.

Figure 3: Manufacturing and assembly progress.

5.1 Our design (Theory)

With the above factors in mind, we decided to build upon the rocker-bogie suspension system. As the design favoured for many extra-terrestrial rovers, it is able to effectively navigate rocky and uneven terrain such as the Giant's Causeway and Lunar Surface sections in the obstacle course. It is also relatively lightweight, not requiring a full body or chassis. Famously, NASA's Spirit, Opportunity, and Curiosity, and countless other space agency rovers have utilised the rocker-bogie design on Mars [3]. The main benefit of this design for us is that the wheels move independently, meaning each wheel can adjust its height according to the terrain to maintain stability. The major disadvantage that is acknowledged for this design is the stress and damage on the wheels. However, this issue does not apply to us in the same magnitude as the robot will only be traversing obstacles primarily composed of wood or smooth surfaces.

As a 6-wheel drive, and with the extra maneuverability of each wheel, the robot is able to turn in place, which is ideal for the tunnel, and also perform an arc turn [4].

The 4 arms consists of two pairs of linkages: the bogie and the rocker. The rocker is directly connected to the base, but can still move independently from them. The design functions with either the bogie or the rocker as the front. Figure 54 below has the rocker at the front, however, we decided to put the bogie at the front to ease the stair climbing section and as we did not need to descend from anything (while in contact with the ground) [5].

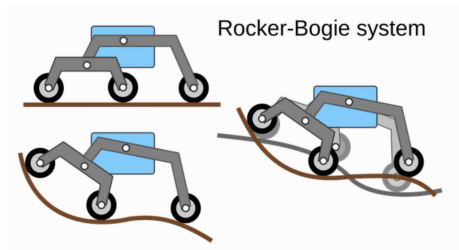


Figure 4: Rocker-Bogie Diagram [5]

6 Prototyping

6.1 Prototype 1

The aim for our first prototype was to estimate the dimensions as well as find the ideal design for the wheels. We focused on evaluating whether the spacing between wheels in our design concepts was practical to implement and suitable for the robot's intended performance.

For the base, we repurposed a scrap piece of wood, which resulted in a length that was not ideal for our intended design. However, we had a reference to base our measurements off. However, it allowed us to establish reference measurements for future iterations. To connect components, we initially used tape, which offered only limited stability. Still, that did not matter as much since we were not yet testing the prototype on the move and we weren't yet utilising the motors. The primary goal at this stage was to test our six-wheel drive concept. We used the thickest wheels available to us, though we were concerned that their diameter might be insufficient for certain obstacles. We have anticipated that in the future we might need to either find other wheels or even design our own.

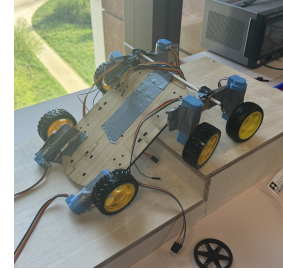


Figure 5: Prototype 1

6.2 Prototype 2

The second prototype was laser cut with 12mm wood. We designed and 3D-printed mounts for all six motors. With the mounts securely screwed in place, the motors were held firmly and remained stable during testing. The issue we came across with this prototype was the fact that there was a motor mounted in wrong orientation so the mounts hit the ground before the wheel.

Also, there was not enough space between the wheels to go up the stairs. This time we have tested the work of the motors by connecting them to a breadboard that rested on the top. The main concern for us was whether it was or was not able to climb up the stairs, which is why that was the first obstacle we started testing it on. While this prototype was able to overcome 1mm and 3mm steps with ease, it lacked sufficient power and/or the choice of wheels was suboptimal for larger obstacles. That is why in the next prototype we had to rethink the design on the first two wheels in particular, since those were the ones stalling right before the 5mm step. We also decided to change the motors to 12V after receiving feedback from both the lab assistants and teaching assistants, who noted that the DC motors we initially selected were relatively weak and might not meet the performance requirements of our design. We have also made the decision for the next prototype to be cut from thinner wood in order to reduce the weight of the structure and make it overcome obstacles more efficiently.

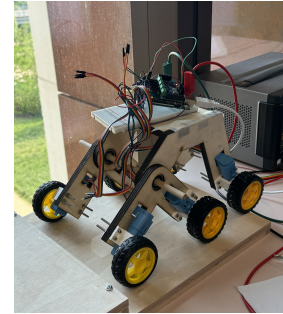


Figure 6: Prototype 2

Additionally, for the Giant's Causeway section, we found that the robot had the potential to get caught on the sharp edges of an elevated block due to the way the frame has been cut. Since the edges of both the smaller and larger linkages are sharp with 90° angles, the issue that the frames would make contact with the terrain and not be able to overcome it with the force of the motors. This

issue is depicted below.



Figure 7: Giant's Causeway hitting sharp edges.



Figure 8: Rounded frame gliding over sharp edges.

To combat this, we decided to use rounded edges for the frame where the motor would be mounted and where it may have contact with the terrain. For the next iteration of the linkages, we will be scaling it down to account for the larger diameter of the wheels and the size prerequisites of the 30x30cm tunnel. The edges of each linkage will be rounded so it can glide across Giant's Causeway; the design will change for the new mounts to include holes for the screws and new motors; it will be laser cut with thinner plywood at 6mm instead of 12mm. With using new motors, the shaft for the ones we picked was different from the current wheels available, so 3D printing wheels with the corresponding coupling also needed to be considered.

7 Short development timeline

During the first week, we developed and tested Prototypes 1 and 2. We experimented with various wheel configurations using wheels sourced from the lab, securing them temporarily with gaffer tape. Motor mounts were installed to allow for better mechanical stability. Sensor integration began, and we confirmed that the line sensors could successfully detect lines and differentiate between various types of course surfaces. All three distance sensors were confirmed to be functioning, although they had not yet been tested within the actual wall environment. On the software side, we wrote and tested basic movement code, including motor control for moving forward, adjusting speed, and executing turns.

In the second week, we upgraded the motors and installed new motor mounts to improve reliability and performance. We also 3D-printed custom wheels tailored to our robot's requirements. With sensor placement becoming a priority, we began designing a mount for the servo that would control the reflectance sensors. We started conceptualising the hook-raising mechanism and the design

of the hook itself for the zipline descent. On the electronics side, both the mechanical and UDP-based kill switches were implemented and tested. We also completed and tested code that integrated both sensor readings and motor control logic.

During Week 3, we 3D-printed mounts for both servos—one for the reflectance sensor arm and one for the hook-raising mechanism. The hooks themselves were also printed, and we connected the gear system that allows them to lift in sync. The reflectance sensors were assembled onto a laser-cut board and attached to the newly built arm. We also revised the wheel configuration again, reverting to lab-provided wheels for improved traction and leaving only the front two as our original 3D-printed versions. Our Wi-Fi connection was thoroughly tested and finalised using the lab network, ensuring stable UDP communication. During trial runs, we soldered a custom board, connected a mechanical button, and successfully wired all five sensors and six motors. However, there was a faulty solder on the mechanical button, which forced us to use a disconnected spare. Additionally, the robot was not yet fully mechanically sound at that stage.

8 Theory and Mechanical Design

8.1 Wheels Movement

8.1.1 Mathematical Modeling

Our car is equipped with six motors, one for each wheel. We use two motor shields with six outputs to control each motors independently. These motors assist with both straight-line movement and directional changes by varying the rotation speed between the left and right wheels.

When the car is moving in a straight line, all of the wheels rotate in the same direction. The following parameters are defined:

- Wheel radius: r
- Wheels angular velocity: $\omega_{FL}, \omega_{FR}, \omega_{ML}, \omega_{MR}, \omega_{BL}, \omega_{BR}$
- Linear velocity of the robot: v
- Total distance traveled: d
- Travel time: t
- Distance between left wheels and right wheels: L

The ideal average linear velocity of the whole system is:

$$v = \frac{r \cdot \omega_{FL} + r \cdot \omega_{FR} + r \cdot \omega_{ML} + r \cdot \omega_{MR} + r \cdot \omega_{BL} + r \cdot \omega_{BR}}{6} \quad (1)$$

The total distance traveled is:

$$d = v \cdot t = \frac{r(\omega_{FL} + \omega_{FR} + \omega_{ML} + \omega_{MR} + \omega_{BL} + \omega_{BR})}{6} \cdot t \quad (2)$$

During turning, the two middle wheels on each side remain stationary to reduce slippage and improve stability. Only the front and rear wheels on both sides contribute to the turning motion. The turning angle is generated by the difference in rotation speeds between the left and right side wheels (front and rear only).

The angular velocity of the robot:

$$\omega_{car} = \frac{r(\omega_{FR} + \omega_{BR} - \omega_{FL} - \omega_{BL})}{2 \cdot L} \quad (3)$$

The turning radius of the robot path:

$$R = \frac{v}{\omega_{car}} = \frac{L(\omega_{FL} + \omega_{FR} + \omega_{ML} + \omega_{MR} + \omega_{BL} + \omega_{BR})}{3(\omega_{FR} + \omega_{BR} - \omega_{FL} - \omega_{BL})} \quad (4)$$

8.1.2 Design

In designing the wheels, a few key deliberations were taken into account. First, the coupling had to fit the shaft of the IG220019X00015R motors, and given this was to be 3D printed, a relatively sizeable 0.5mm tolerance in the diameter was given to the design. Secondly, it did not seem reasonable to add tread patterns to the outside "tires" in the design which would have added a significant amount of time in the post-processing of the 3D printing process to remove supports, etc. Instead of treads, we found existing rubber tracks with decent slip resistance and elasticity which was suitable for tires of 75-80mm diameter. This determines the diameter of the wheels, being 80mm in the final design, which was larger than the ones used to prototype. Third, we wanted the width of the tires to be sufficiently wide for stability but not too wide allowing the robot to have space when turning tight corners, especially in the tunnel. The weight of each wheel would also need to be as light as possible while balancing durability and robustness. In the end, the wheels had a diameter of 80mm, a width of 20mm, and an infill of 10% making each wheel 26g.



Figure 9: 3D printed wheel with tires.

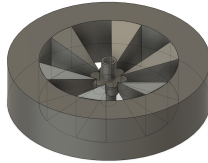


Figure 10: Wheel CAD.

Due to the rocking aspect of the smaller linkage, the path of the middle pair of wheels would intersect with the back pair. We found this issue when in prototyping with the readily available wheels that were 18mm in width. To combat this, we offset the

front linkage using a buffer on each side so the wheels would not collide, however, this meant that the overall width of the robot was increased by a significant amount. Thus, we decided to use thinner, readily available wheels for the four back wheels to reduce the buffer space needed and, therefore, the width of the robot. This excludes the front two that utilise the 3D printed design which are kept wide for stability since they do not interfere with the other wheels.

8.1.3 Visualisation of Mechanisms

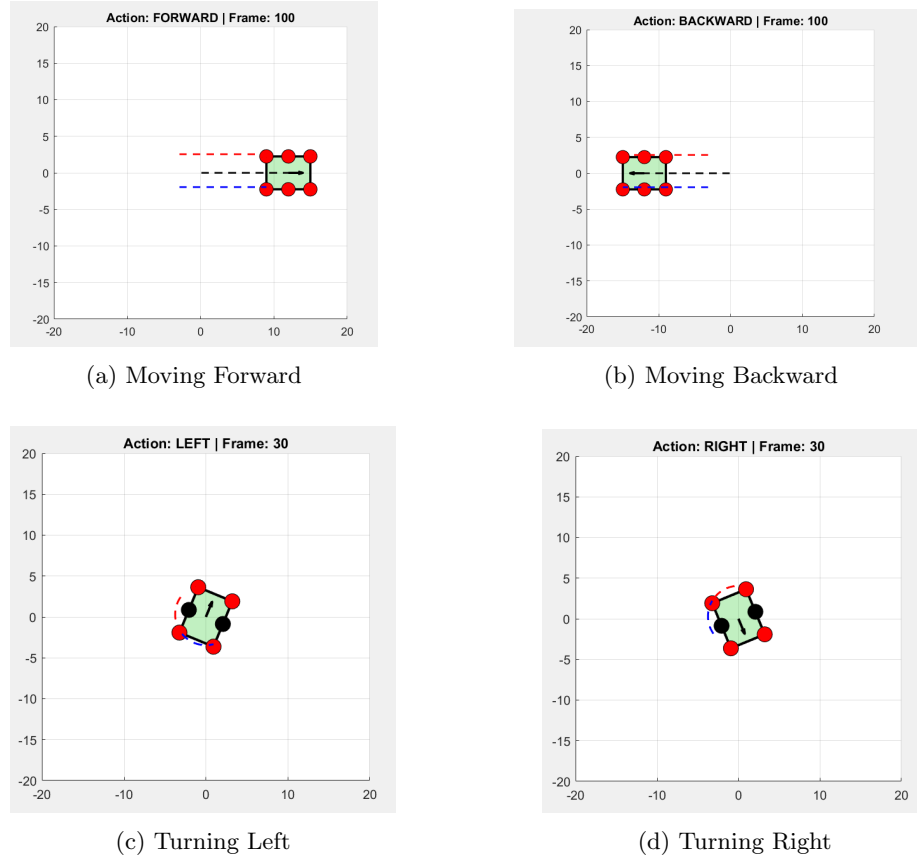


Figure 11: Visualisation of Wheel's Movement

Above are Matlab plots demonstrate the kinematic behavior of six-wheeled robot with independently controlled wheels under four different motions: forward, backward, left turn and right turn. Wheel positions are marked, where rotating wheels shown in red and stationary wheels in black during turns. Tra-

jectories of the front left and right wheels are traced as dashed blue and red lines to visualise the wheel paths and the robot's center trajectory is also plotted in dashed black.

8.2 Hook

8.2.1 Mathematical Modeling

For the "Floor is Lava" task, we designed a dual-gear mechanical claw system, which is connected to two hooks to support the robot's body and keep it at the same horizontal level while moving across the lava. The DSS-M15S servo is used to rotate the gear with an arm linking to the hook attached. The gear will then cause an opposing rotation to an adjacent gear, allowing a closing action to occur. This is visualised in the next subsection. The servo rests on a platform designed to elevate this subsystem to a suitable height in order to clamp onto the rod. Both the servo is mounted onto this platform and the adjacent gear that is not directly connected to the servo. The mount and hook designs are seen below.

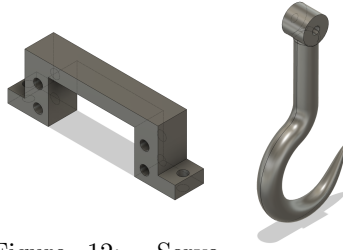


Figure 12: Servo mount CAD.

Figure 13: Hook CAD

The front wheels will leave the surface first, and the robot will be driven by the back wheels. Then, when the back wheels lift off, the front wheels will drive the robot. This ensures there are always wheels touching the ground to keep moving our robot forward to cross the lava.

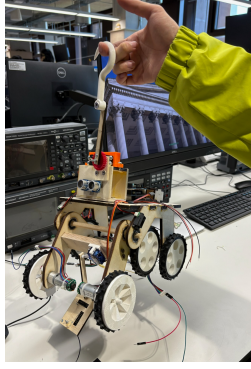


Figure 14: Hook holding the whole robot weight.

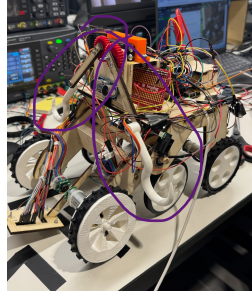


Figure 15: Hooks down.

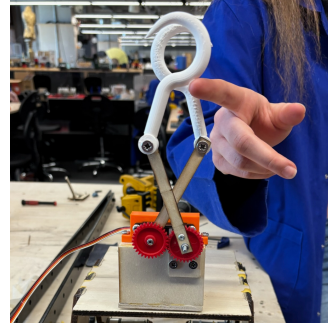


Figure 16: Hook mechanism close-up.

The trajectory of the hook is dependent on the angle rotated by the servo. The following parameters are defined:

- Gear radius: r
- Length of connecting link: L
- Rotation angle of the first gear: α
- Rotation angle of the second gear: β
- Radius of the hook arc: R_{hook}
- Center coordinates of gears: O_1, O_2

The driving gear and driven gear mesh together with equal radii, so they rotate with equal angular speeds but in opposite directions:

$$\alpha = -\beta \quad (5)$$

Where the transmission ratio:

$$i = \frac{\omega_{\text{driving}}}{\omega_{\text{driven}}} = -1 \quad (6)$$

The positions of the hook tips at the ends of the two rigid links are:

$$P_1 = O_1 + L \begin{bmatrix} \cos \beta \\ \sin \beta \end{bmatrix} \quad (7)$$

$$P_2 = O_2 + L \begin{bmatrix} \cos \alpha \\ \sin \alpha \end{bmatrix} = O_2 + L \begin{bmatrix} \cos(-\beta) \\ \sin(-\beta) \end{bmatrix} = O_2 + L \begin{bmatrix} \cos \beta \\ -\sin \beta \end{bmatrix} \quad (8)$$

The hooks' orientation angles are:

$$\theta_{\text{hook1}} = \beta + \frac{\pi}{2} \quad (9)$$

$$\theta_{\text{hook2}} = \alpha - \frac{\pi}{2} = -\beta - \frac{\pi}{2} \quad (10)$$

The new centers of hook arcs C1 and C2 can be represented as:

$$C_1 = P_1 - r_{\text{hook}} \begin{bmatrix} \cos \theta_{\text{hook1}} \\ \sin \theta_{\text{hook1}} \end{bmatrix} \quad (11)$$

$$C_2 = P_2 - r_{\text{hook}} \begin{bmatrix} \cos \theta_{\text{hook2}} \\ \sin \theta_{\text{hook2}} \end{bmatrix} \quad (12)$$

8.2.2 Visualisation of Mechanisms

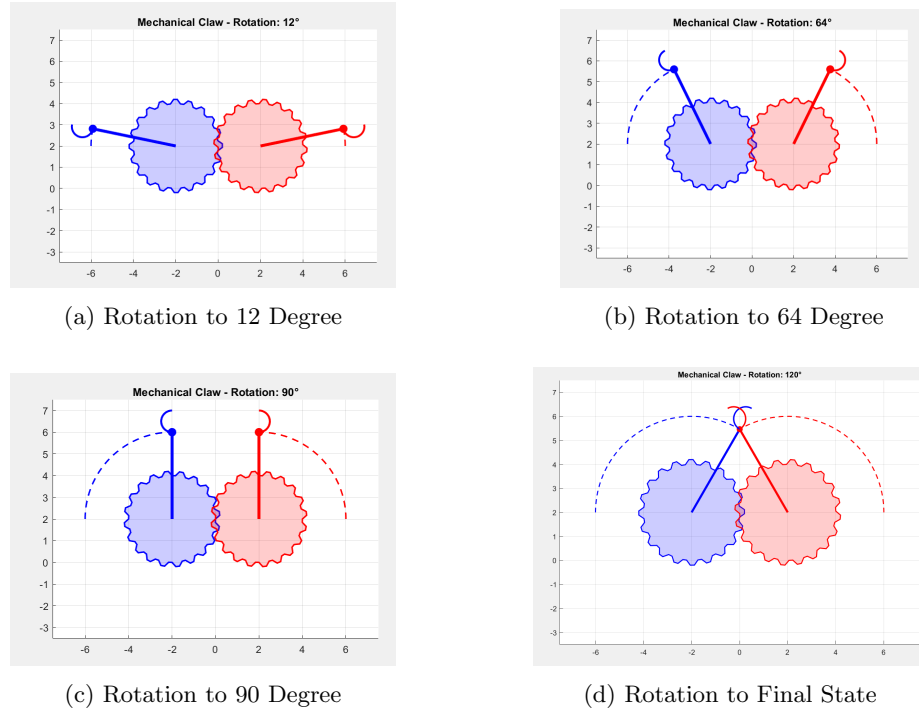


Figure 17: Visualisation of Dual-wheel Hook System

Above are Matlab plots demonstrate the kinematic behavior of mechanical hooks driven by two intermeshing gears and connected linkages. As the gears rotate in opposite directions, the arms pivot, causing the claw tips to open and close to be able to hook a bar above.

8.3 Conceptual Design and Schematics

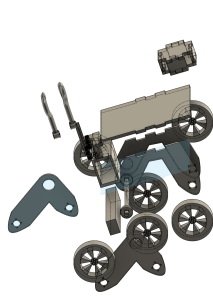


Figure 18: Exploded view of full assembly

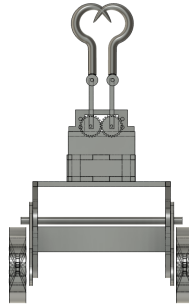


Figure 19: Front view of full body assembly

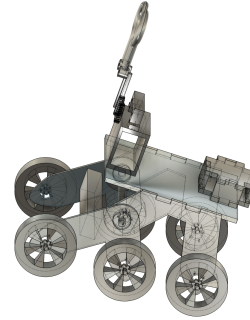


Figure 20: Side view of full body assembly

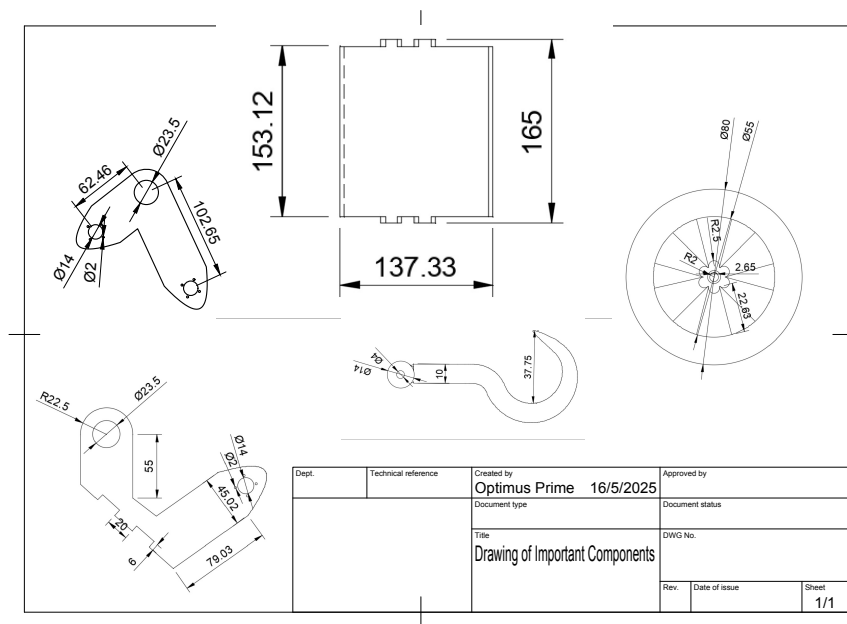


Figure 21: Drawing of important components with variables and parameters

9 Electronic Components

9.1 Schematic Diagram

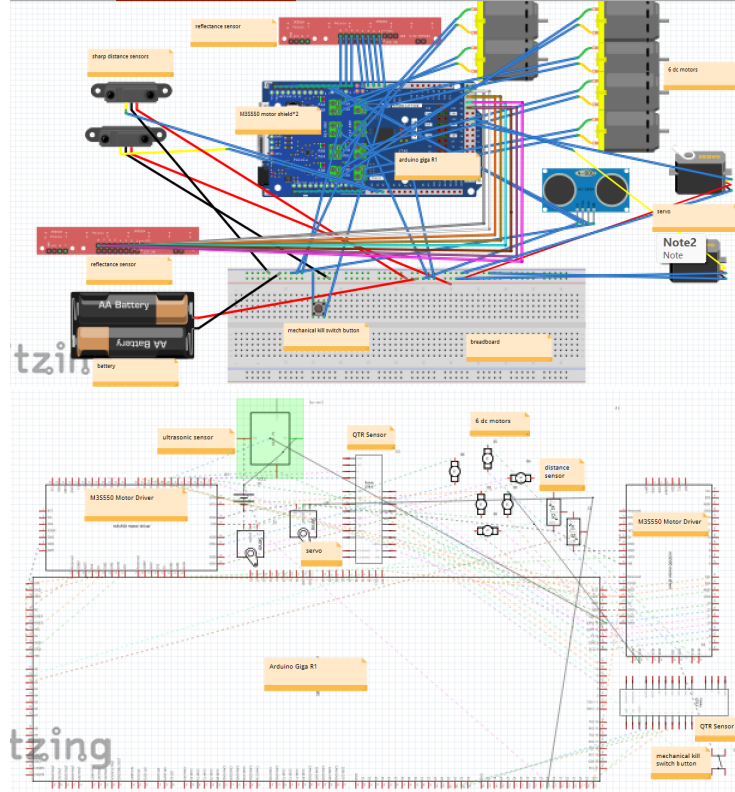


Figure 22: Electronic Schematic Diagram

Above are the labelled schematic diagrams illustrating all the hardware components and the wiring connections for each pin used in our project.

9.2 Motors

9.2.1 DC Motors

We are using two Motoron M3S550 triple motor controller shields to drive six motors. These shields are stacked together on an Arduino Giga R1 board. Although they share the same I^2C bus, each shield is assigned a unique I^2C address, where the lower shield is set to 0x10 and the upper shield to 0x14.

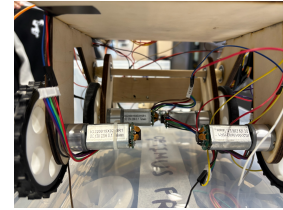


Figure 23: Motors mounted inside.

This configuration allows the microcontroller to control each shield independently via the SCL and SDA lines. With three outputs per shield, we can independently control all six IG220019X00015R DC motors.

As previously mentioned, all motors will be assigned the same PWM value during forward or backward motion. However, for turning direction, the two middle motors will remain stationary while the left and right wheel sets operate at different speeds to create a turning angle. The MCU will continuously receive data from all sensors and makes real-time decisions based on this input. Once a decision is made, control signals are sent from the MCU to the motor shields via the I^2C bus. The motor shields then drive the motors to execute forward, backward, left-turn, or right-turn movements.

The motor can be mounted with simple M2 screws through the linkage into the motor, so no mount is necessary. Below are the simulated control signals and waveform plots showing the communication from the MCU to the motors.

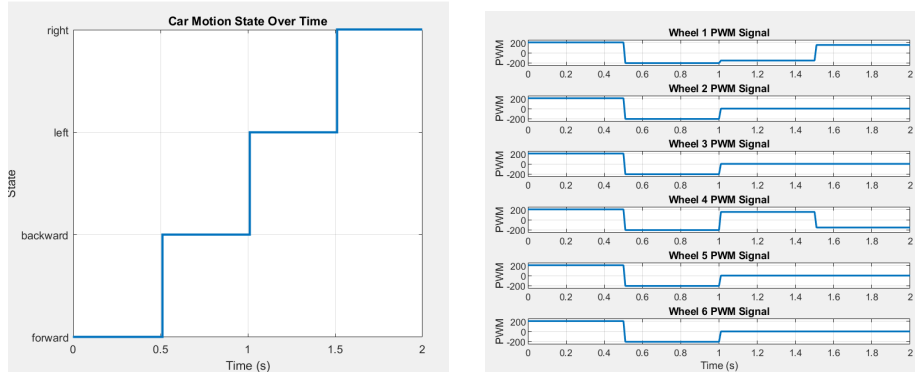


Figure 24: Signals and Waveforms from MCU to motors

9.2.2 Servo

In addition to the DC motors, we use a DSS-M15S servo to drive the gear in the previously mentioned dual-gear hook system. It can precisely rotate between 0 and 270 degree, which is an ideal choice since we do not need the servo to be continuously spinning. Instead, we need accurate angular positioning to hook onto the bar above the robot.

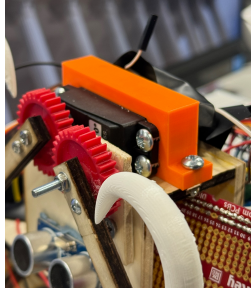


Figure 25: Testing reflectance sensors with "fork" detection.

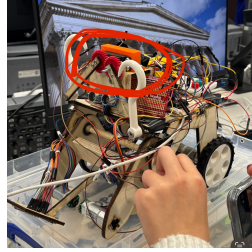


Figure 26: Ultrasonic sensor mounted at the front.

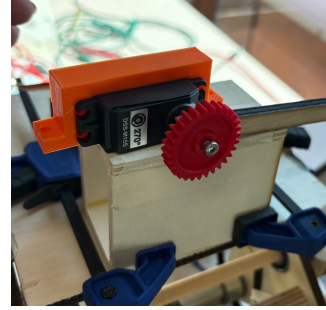


Figure 27: Reflectance sensor arm mechanism on the robot.

9.3 Sensors

As we have mentioned, we positioned two reflectance sensors in an inverted "T" shape near the front underside of the robot, and three distance sensors at the front, left, and right sides, mounted as high as possible to minimise the risk of misidentifying obstacles as walls.

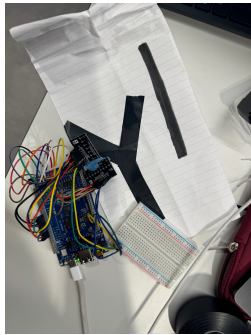


Figure 28: Testing reflectance sensors with "fork" detection.

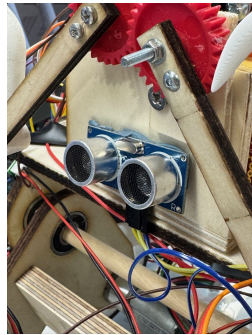


Figure 29: Ultrasonic sensor mounted at the front.

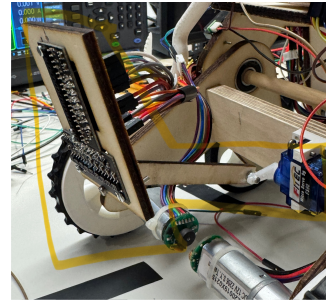


Figure 30: Reflectance sensor arm mechanism on the robot.

We have started off with code to test whether the sensor logic is able to identify different parts of the line-following obstacle such as: turns (90 degrees, acute and smooth), forks, dead-ends, double lines, dotted lines, obstacle ahead, course-ending. Additionally, we tested whether our robot correctly distinguishes between obstacles ahead and walls on the right, left, or both sides using the distance sensors. We also assessed whether the robot can determine the appropriate response to these challenges. For the distance sensors we have made a decision to use two Sharp distance sensors for the sides of the robot and one ultrasonic sensor for the front. The reason for that was the fact that the front of the robot

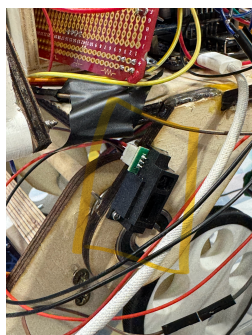


Figure 31: Distance sensor placed on the side.

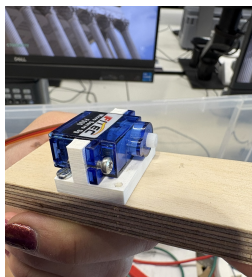


Figure 32: Arm servo mount.

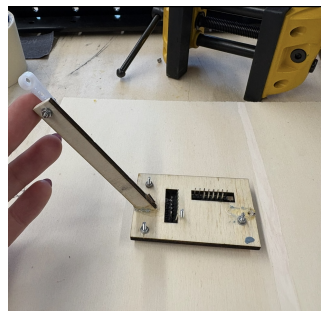


Figure 33: Sensor arm de-attached.

was long, and the 2-15cm range on the Sharp sensors would not work. There was no space to place the front sensor at the very front of the robot (somewhere where the wheels end) so it had to be attached to the front of the base. That meant that the sensor was around 10cm away from the end of the wheels. This would have introduced problems of the sensor detecting the wall too late, and robot simply bumping into the wall ahead. The ultrasonic sensor was mounted to the elevated platform at the front of the robot. This platform was initially purposed for the servo mechanism for the hooks, however, it turned out to be the perfect place to fix the ultrasonic sensor to. The Sharp distance sensors were drilled on to the side of the robot. We also made sure to place them as close to the front as possible, in order to avoid the potential problem to it detecting the wall too late and not having any space to actually turn.

To ensure the reflectance sensors are positioned optimally close to the ground for accurate line detection, while also allowing for flexibility when navigating uneven terrain or transitions between surfaces, we decided to implement a mechanical arm mechanism. This arm, operated by a micro servo motor, enables the sensors to be actively lowered during standard line-following operations and raised when the robot encounters sections where ground clearance is required. This design protects the sensors from potential damage. The micro servo itself was attached to the front of the robot, in between the two front wheels. The mount for the servo was also 3D-printed. As a result, for our first trial run, we managed to get the sensors to read the values from all 5 sensors successfully and display them in Arduino Serial Monitor. (Figure 35)

We have also plotted the values we got from each of the sensors with the help from Matlab. On the diagrams below you can see the plotted results for reflectance sensors (all 18), Sharp distance sensor and ultrasonic sensor.

For the reflectance sensor, this graph shows the movement from white space onto a line. (Figure 36) That is why you can see some lines gradually drop from

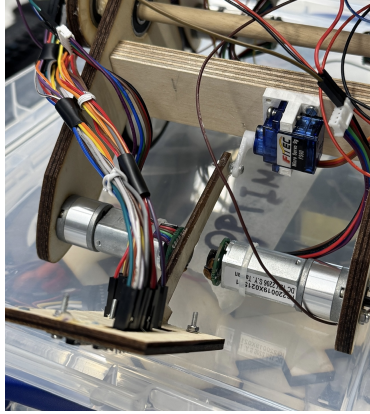


Figure 34: The view of the front of the robot - wiring assembly (used electrical tape).

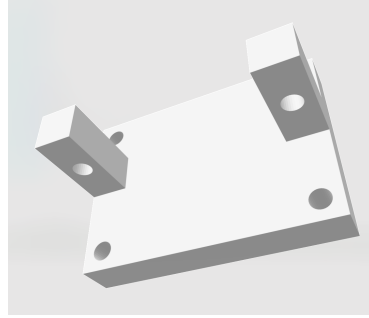


Figure 35: Arm servo mount.

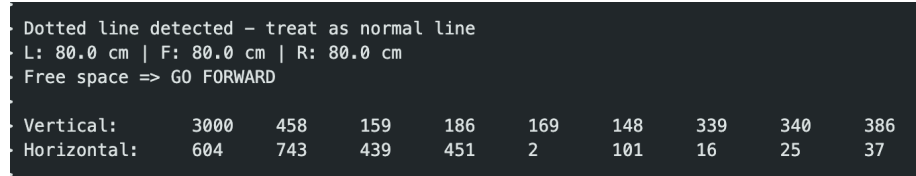


Figure 36: Sensor readings.

top to bottom (from white to black) one by one. Then, the thicker lines indicated 3 middle sensors on the second board going into the black line simultaneously. The 6 other are remaining on the top, since they stay on the white surface. The data has quite a lot of noise, which is something that we might need to approach later on in the development.

For the Sharp distance sensor, (Figure 37) which operates reliably within the 2–15 cm range, we implemented a hard-coded condition in the code: when the measured distance exceeds 15 cm, the output is forced to display a value of 80. This serves as an indicator that the object is beyond the sensor’s effective range. As a result, the diagram shows noticeable spikes to 80 — these represent moments when we were intentionally testing the 15 cm threshold. Outside of these spikes, the data remains relatively stable, with minimal noise. In the rest of the readings, we moved a hand gradually closer to and farther from the sensor to simulate normal usage.

As for the ultrasonic sensor, (Figure 38) its operating range is significantly larger than that of the Sharp sensor, so we did not implement any forced output conditions. Instead, we simply moved a hand gradually closer to and farther from the sensor to observe how the distance readings responded. The results were as expected, showing smooth variations corresponding to the hand’s posi-

tion.

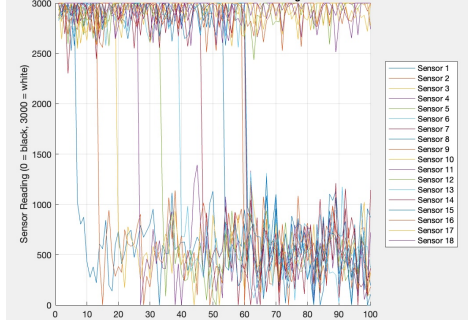


Figure 37: Reflectance sensors readings.

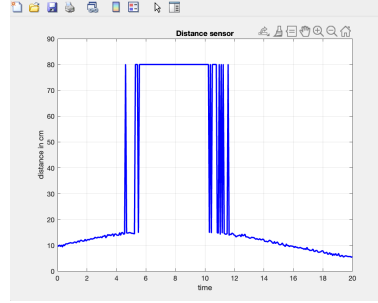


Figure 38: Sharp distance sensor readings.

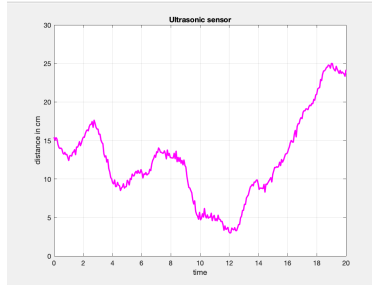


Figure 39: Ultrasonic readings.

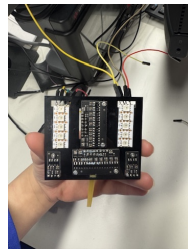


Figure 40: Reflectance sensor board with vertical, horizontal, left, and right sensors and two strips of LEDs

9.4 Kill Switches

We successfully implemented both mechanical and Wi-Fi kill switches, enabling the robot to automatically stop or start its motors upon receiving a signal. The mechanical kill switch required a slight press-and-hold action to activate, which introduced a minor delay in response time. To test the Wi-Fi UDP

kill switch, we used Wireshark to monitor network traffic and verify that the "KILL" message was sent to the specified port. The test messages were written in Python and transmitted over the network to trigger the shutdown of the motors. Initially, we used a personal hotspot for testing, but encountered issues

```
16:00:05.346 -> Connecting to Wi-Fi...
16:00:05.346 ->
16:00:05.346 -> WiFi connected
16:00:05.346 -> IP address: 192.168.0.248
16:00:27.769 -> Button pressed!
16:00:27.769 -> Robot started via button
16:00:29.915 -> Button pressed!
16:00:29.915 -> Robot stopped via button
16:00:41.246 -> Button pressed!
```

Figure 41: Wifi connection and button press results.

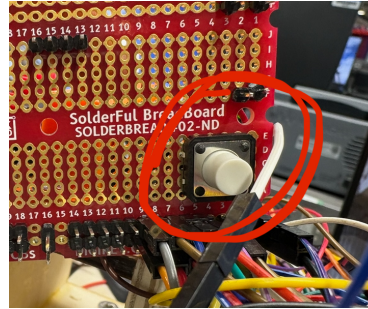


Figure 42: Button soldered.

```
16:02:31.574 -> Received UDP packet: Stop
16:02:32.592 -> Received UDP packet: Stop
16:02:36.867 -> Received UDP packet: D:,S:100,K:j, n, shift_r
16:02:36.965 -> Received UDP packet: D:F,S:100,K:j, n, shift_r
16:02:37.097 -> Received UDP packet: D:F,S:100,K:n, r, shift_r
16:02:37.196 -> Received UDP packet: D:F,S:100,K:r, shift_r
16:02:37.294 -> Received UDP packet: D:F,S:100,K:j, n, r, shift_r, t
```

Figure 43: Faulty UDP packets transmitted.

due to iPhone restrictions, which prevent devices from communicating with each other on a hotspot network. Despite this, during the official trial runs, the robot successfully received the "Stop" message and stopped the motors immediately. The mechanical button was then used to restart the motors. Moreover, during the test trials, we were assessed on whether the robot would respond to any unintended messages received over Wi-Fi. The results showed that when any message other than the designated "Stop" command was transmitted over the network, the motors remained unaffected. This confirmed that the robot was correctly programmed to respond only to the specific stop command.

10 Trial run prototype

In the photo below you can see the fully assembled prototype of the whole robot with all the sensors attached and all the electronics organised. The robot looks messy and we did have to use a lot of electric tape to locate the wires on the base, so that they do not interfere with the wheels. In future prototypes, we will pay special attention to the wiring, colour coding, and possibly use cable ties to fix the wires in place.

The front two wheels had the tendency to pop off after some rotating. This

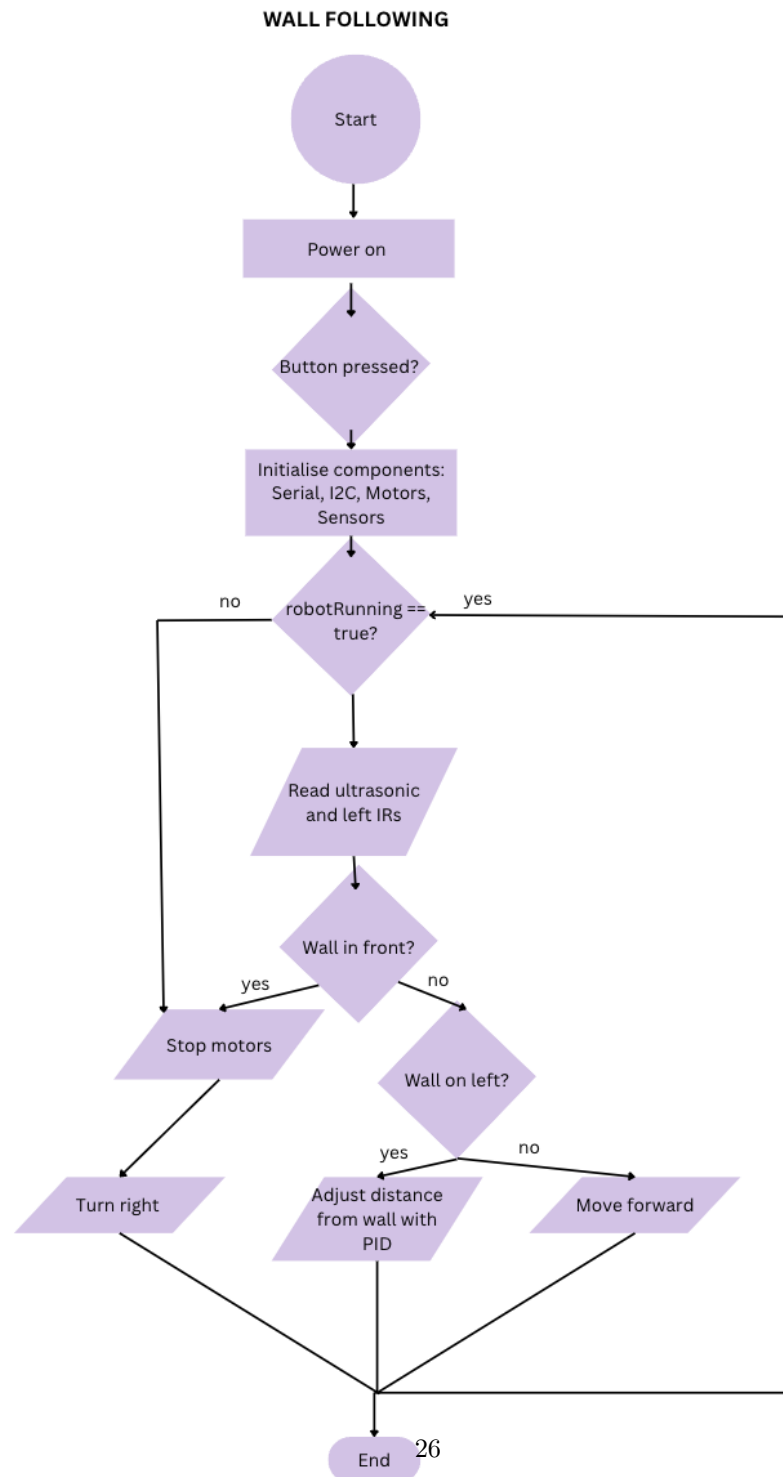
is due to the shaft eroding the thin plastic coupling of the wheel. We will need to reinforce the coupling with thicker walls and reprint these. A future issue may also arise with the motors suspended in the middle of the robot. Some freely rolling belt or coupling to reinforce the parallel motor pairs may need to be added. We will also need to integrate all of our sensor code into one file. Overall, a few key changes need to be made to ensure that the robot has the highest chance of success over the obstacles as possible.

11 Programming

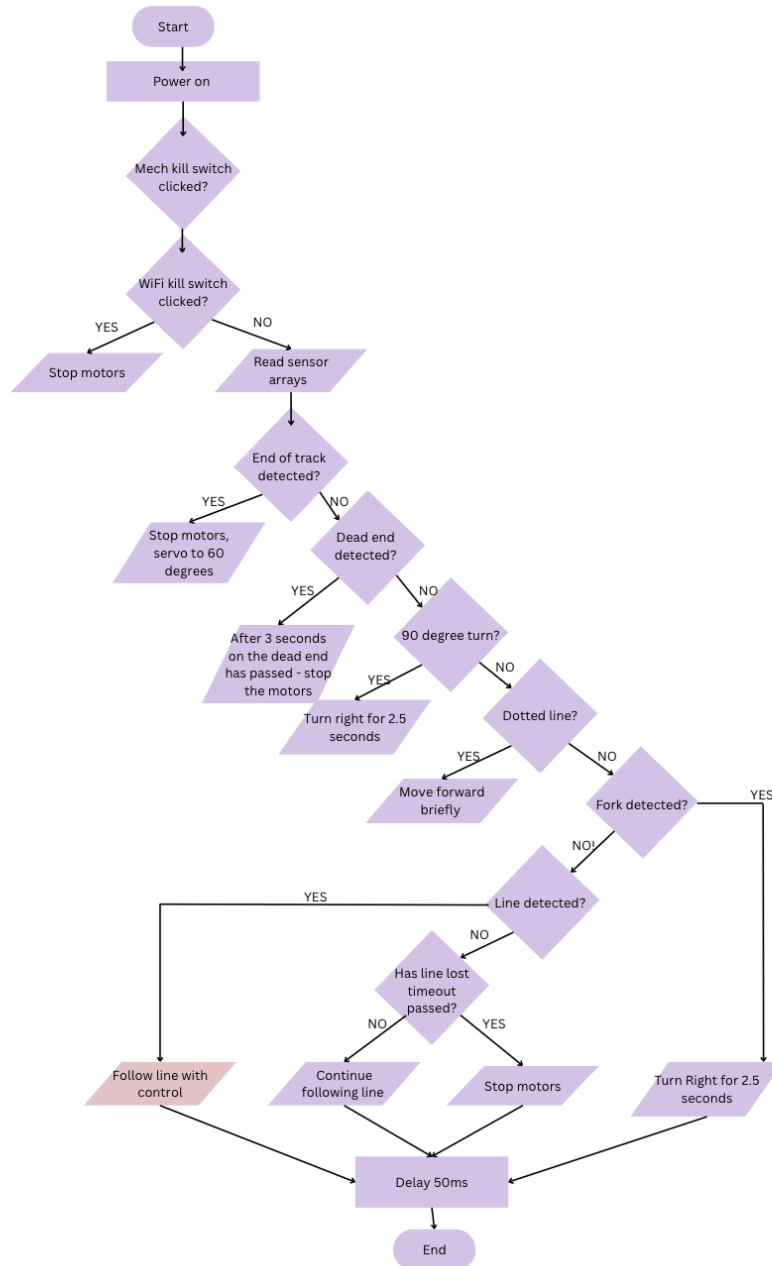
11.1 GitHub Repository

<https://github.com/tara-kas/term3robot.git>

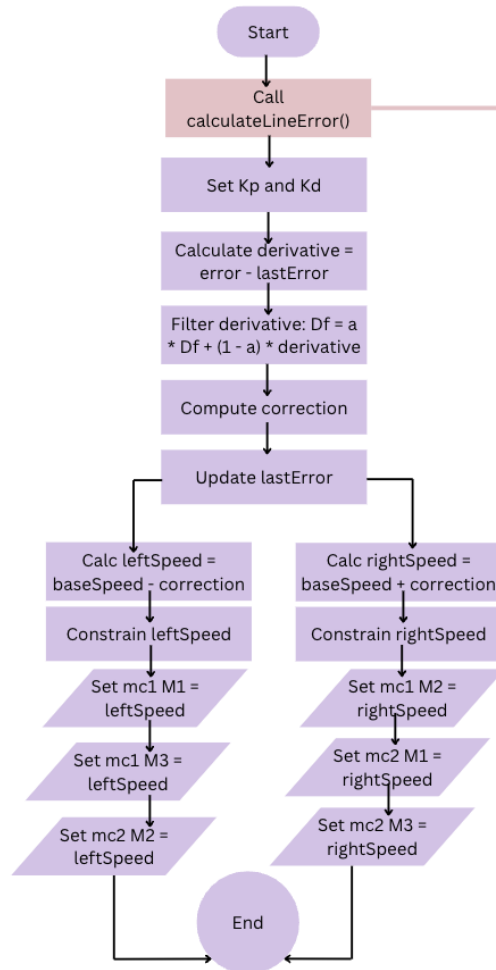
11.2 Flowcharts



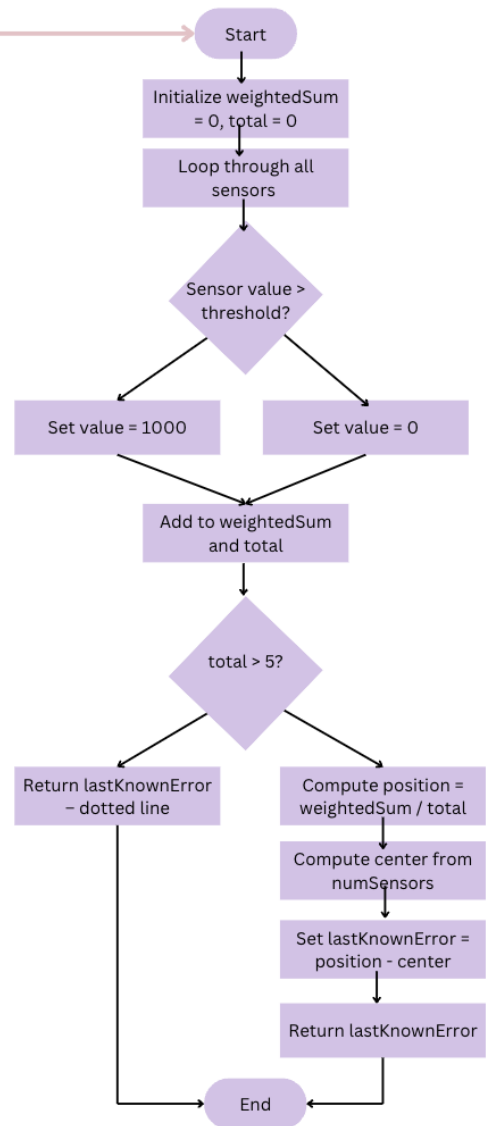
LINE FOLLOWING



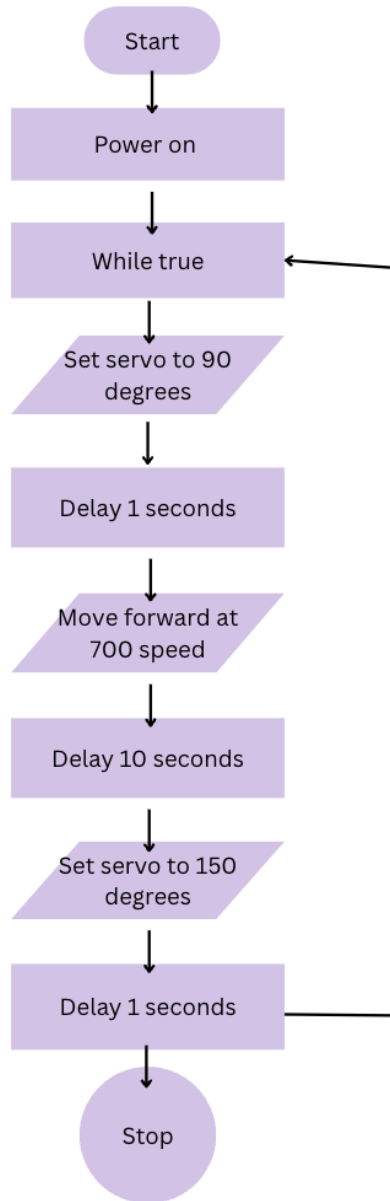
FOLOW LINE WITH CONTROL FUNCTION



CALCULATE LINE ERROR FUNCTION



FLOOR IS LAVA



12 Algorithms Performance

12.1 Algorithms That Worked as Expected

12.1.1 Motor Control Logic

The motor control code for the 6-wheel robot using two Motoron I2C motor controllers worked as expected, which successfully tested a sequence of movement behaviours, including going forward, backward, turning right and turning left on the spot. The behaviour of each motion function matched the expected real-world movement of the robot.

```
void moveForward(int speed) {
    mc1.setSpeed(1, speed); // Front left
    mc1.setSpeed(2, speed); // Front right
    mc1.setSpeed(3, speed); // Middle left
    mc2.setSpeed(1, speed); // Middle right
    mc2.setSpeed(2, speed); // Left back
    mc2.setSpeed(3, speed); // Right back
}

void moveBackward(int speed) {
    mc1.setSpeed(1, -speed);
    mc1.setSpeed(2, -speed);
    mc1.setSpeed(3, -speed);
    mc2.setSpeed(1, -speed);
    mc2.setSpeed(2, -speed);
    mc2.setSpeed(3, -speed);
}

void turnLeft(int speed) {
    mc1.setSpeed(1, -speed); // Front left
    mc1.setSpeed(2, speed); // Front right
    mc1.setSpeed(3, -speed); // Middle left

    mc2.setSpeed(1, speed); // Middle right
    mc2.setSpeed(2, -speed); // Left back
    mc2.setSpeed(3, speed); // Right back
}

// In-place right turn: right wheels backward, left wheels forward
void turnRight(int speed) {
    mc1.setSpeed(1, speed); // Front left
    mc1.setSpeed(2, -speed); // Front right
    mc1.setSpeed(3, speed); // Middle left

    mc2.setSpeed(1, -speed); // Middle right
    mc2.setSpeed(2, speed); // Left back
    mc2.setSpeed(3, -speed); // Right back
}

void stopMotors() {
    mc1.setSpeed(1, 0);
    mc1.setSpeed(2, 0);
    mc1.setSpeed(3, 0);
    mc2.setSpeed(1, 0);
    mc2.setSpeed(2, 0);
    mc2.setSpeed(3, 0);
}
```

Figure 44: Motor Test

Each motion is performed without any errors. There were no I2C communication failures and both controllers responded reliably. We set all wheels to positive speed for forward movement, reversing all speed to negative motion. And we used the difference in rotation speed to generate left and right turn.

Potential enhancement if we have time in the future:

- Adding an inertial measurement unit (IMU): Right now our robot has limitations towards turning accuracy because the turning is based on fixed `delay()` and speed, leading to possible over/under-rotation. Besides, our robot may turn unexpectedly, even if it wants to go straight initially but encounters small obstacles on the ground. We can use gyroscope data to track actual rotation angle and stop turning precisely when target angle is reached. In addition, we can use accelerometer and gyro fusion to correct the heading in real time.

```
// === Main Loop ===  
void loop() {  
  moveForward(400);  
  delay(3000);  
  
  stopMotors();  
  delay(3000);  
  
  moveBackward(400);  
  delay(3000);  
  
  stopMotors();  
  delay(3000);  
  
  turnRight(400);  
  delay(3000);  
  
  turnLeft(400);  
  delay(3000);  
}
```

Figure 45: Motor Limitation

12.1.2 Mechanical Kill Switch

Our code implements a debounced button toggle mechanism to start and stop the robot. The logic functioned exactly as intended during testing.

We used `millis()` effectively filtered out signal noise and false triggers from the button press. We had real-time feedback via `Serial.println()` confirmed state changes, which made debugging and observation easy.

```

int reading = digitalRead(buttonPin);

if (reading != lastButtonReading) {
    lastDebounceTime = millis();
}

if ((millis() - lastDebounceTime) > debounceDelay) {
    // The button state has truly changed
    if (reading != lastDebounceState) {
        lastDebounceState = reading;

        if (reading == LOW) { // Button pressed
            robotRunning = !robotRunning;
            if (robotRunning) {
                Serial.println("Robot started");
                startRobot();
            } else {
                Serial.println("Robot stopped");
                stopRobot();
            }
        }
    }
}

```

Figure 46: Mechanical Switch

One press of the button reliably started the robot and a second press stopped it.

Potential enhancement if we have time in the future:

- Switch to a capacitive touch sensor or use a more reliable tactile switch: right now we have to press to button hard and long to make it work, it varies in actuation force or bounce.

12.1.3 WiFi Kill Switch

The checkKillSwitch() function successfully listens for UDP packets on a designated port(2390) and parses incoming messages. The logic for recognizing a specific shutdown command ("Stop") and toggling the killSwitch flag worked correctly.

```

const char* ssid = "PhaseSpaceNetwork_2.4G";
const char* password = "8igMacNet";

WiFiUDP udp;
unsigned int localUdpPort = 2390;
char incomingPacket[255];
bool killSwitch = false;

```

Figure 47: Wifi Kill Switch

Potential Enhancement:

- Add logic to accept a "Resume" command to reactivate the robot: we can only restart our robot through mechanical kill switch now. Once the wifi kill switch is activated, it stays that way until a reset or power cycle.

12.1.4 Sensor Test Code

The sensor test functions well. The implementation covered a comprehensive set of sensor inputs and provided clear feedback in real-time via Serial.println() outputs.

Both the ultrasonic and IR sensors reliably provided consistent distance measurements. The logic in checkWallsAndAct() correctly interpreted different obstacle scenarios and printed appropriate decisions like "TURN RIGHT" or "BACKTRACK"

```
// read distance ad ultrasonic sensor values
distanceCMleft = readIRDistance(leftIRPin);
distanceCMright = readIRDistance(rightIRPin);
distanceCM = readUltrasonicDistance();

// if wall is closer than set threshold values
bool wallLeft = distanceCMleft < 7;
bool wallRight = distanceCMright < 7;
bool wallFront = distanceCM < 17;
```

Figure 48: Distance Sensor

The method readSensorArray() correctly discharged and timed sensor pins to capture reflectance data and successfully identified dead ends, forks, curves and dotted lines.

```
Dotted line detected - treat as normal line
L: 80.0 cm | F: 80.0 cm | R: 80.0 cm
Free space => GO FORWARD

Vertical:    3000  458   159   186   169   148   339   340   386
Horizontal:  604   743   439   451    2   101   16    25    37
```

Figure 49: Output of Data

```

void readSensorArray(const int pins[], unsigned int values[]) {
    unsigned long timeout = 3000;

    // set all pins to output and drive to high
    for (int i = 0; i < numSensors; i++) {
        pinMode(pins[i], OUTPUT);
        digitalWrite(pins[i], HIGH);
    }

    // wait!
    delayMicroseconds(10);

    // set pins to input
    for (int i = 0; i < numSensors; i++) {
        pinMode(pins[i], INPUT);
    }

    // measure discharge time
    unsigned long startTime = micros();
    for (int i = 0; i < numSensors; i++) {
        values[i] = timeout; // initialise to max timeout
    }
}

```

Figure 50: readSensorArray

Potential Enhancement:

- Applying smoothing or filtering to reduce jitter: Voltage-to-distance calculations are nonlinear and can be sensitive to minor fluctuations.

12.1.5 Dragon Lava Pit

Our code combines a servo positioning with forward motion of a six-wheel robot. We set the servo to an angle of 90 degrees, which is the position when the hook is clamped onto the rod above the lava pit. Then the robot drives forward at speed 700 for 7 seconds to go across the lava. It then set the angle of servo to 200 degree to release the hook off the rod.

```

void loop() {
    dssServo.write(90);
    delay(1000);
    MoveForward(700);
    delay(7000);
    dssServo.write(200);
    delay(1000);
}

```

Figure 51: Dragon Lava Pit

Potential enhancement:

- Add wall-following functionality for the dragon lava pit to automatically detect the hook-on and hook-off positions: Currently, we manually measure the distance and adjust the delay time and speed needed to cross the lava pit. However, this only works reliably when crossing individually. We can improve this by adding control logic: when the wall is detected, we hook on; when wall tracking is lost, we hook off. This is feasible because there are walls on the sides of the lava pit that can be used for detection.

12.1.6 Wall Following

We use ultrasonic obstacle avoidance, which stops and turns the robot if an obstacle is detected within 17 cm. To maintain wall-parallel alignment, we use front and back IR sensors on the left side. Additionally, we have servo control that initializes the servo angle to 60 degrees to raise the reflectance sensor, as line-following is no longer needed. This prevents damage to the reflectance sensor when navigating stairs or large crossways.

```
// if wall is closer than set threshold values
bool wallLeft = distanceCMleft < 7;
bool wallRight = distanceCMright < 7;
bool wallFront = distanceCM < 17;

// no motor functions since it's just sensor test
if (wallFront) { // if a wall is in front, decide whether to turn right or left or go back
  if (wallLeft && !wallRight) { // if a wall is left but not right -> turn right
    Serial.println("Obstacle ahead + left wall => TURN RIGHT");
  } else if (wallRight && !wallLeft) { // if a wall is right but not left -> turn left
    Serial.println("Obstacle ahead + right wall => TURN LEFT");
  } else if (wallLeft && wallRight) { // if a wall is right and left -> go back
    Serial.println("Walls on all sides => STOP or BACKTRACK");
  } else { // same as above
    Serial.println("Obstacle ahead, space on sides => BACKTRACK");
  }
} else {
  if (wallLeft && wallRight) {
    Serial.println("Between walls, no front obstacle => GO FORWARD");
  } else {
    Serial.println("Free space => GO FORWARD");
  }
}
```

Figure 52: Wall Following

Potential Enhancement:

- Add moving average or low-pass filtering to IR readings: IR sensor noise or environmental reflection will cause slight PID instability.

- Use IMU or ultrasonic to dynamically end turns: We right now just hard-coded turn duration which is not adaptive.

12.2 Algorithms That Did Not Work as Expected

12.2.1 Line Following

It is designed to follow a black line on a surface using an array of IR sensors. It can detect dead ends, the end of the track, dotted lines, and 90-degree sharp turns. It is supposed to use PID control to follow the line under normal conditions, excluding special cases. It performed pretty well before but fail to do so on the day of trial run. However, it sometimes turns right automatically even though the serial monitor indicates it should go forward.

```
// line following control
int calculateLineError() {
    long weightedSum = 0;
    int total = 0;

    for (int i = 0; i < 9; i++) { // 9 for the reflectance sensor arrays
        int value = sensorValuesH[i] > threshold ? 1000 : 0;
        weightedSum += (long)value * (i * 1000);
        total += value;
    }

    if (total < 5) {
        // no black detected - possibly a gap (dotted line)
        // return the last known direction to keep moving smoothly
        return lastKnownError;
    }

    int position = weightedSum / total;
    int center = ((numSensors - 1) * 1000) / 2;
    lastKnownError = position - center;

    return lastKnownError; // return the error
}
```

Figure 53: Error Calculation

```

void followLineWithControl(int baseSpeed) {
    int error = calculateLineError();

    // experimental values through testing
    float Kp = 0.4;
    float Kd = 0.2;
    float Ki = 0.15;

    static float integral = 0;
    integral += error;
    int derivative = error - lastError;
    D_filtered = D_alpha * D_filtered + (1.0 - D_alpha) * derivative;

    // pd implementation
    int correction = Kp * error + Kd * D_filtered;
    //int correction = Kp * error;
    lastError = error;

    int leftSpeed = baseSpeed - correction;
    int rightSpeed = baseSpeed + correction;

    leftSpeed = constrain(leftSpeed, -400, 400);
    rightSpeed = constrain(rightSpeed, -400, 400);
}

```

Figure 54: PID Control for Line Following

Reason why it did not work:

On the final day, we reuploaded the sensor test code to our board and discovered that two vertical sensors and one horizontal sensor were broken and not working. The vertical sensors don't significantly affect our robot's performance because we only count the number of vertical sensors detecting black and set the threshold to 5, instead of requiring all of them to detect black, to identify special cases such as end of track. However, the horizontal sensor is very important because it contributes to the weighted sum used in our error and correction calculations. The broken sensor is the sixth one from the left, positioned near the middle, so its weighting is very high. Since it is broken, it always detects black, which adds a large value (3000 multiplied by its weight) to the total error. This causes a conflict: even if the left sensors detect the line and indicate the robot should turn left, the sixth sensor signals the board to turn right. That is why the robot suddenly stopped working properly.

Dotted line detected - treat as normal line									
L: 80.0 cm F: 80.0 cm R: 80.0 cm									
Free space => GO FORWARD									
Vertical:	3000	458	159	186	169	148	339	340	386
Horizontal:	604	743	439	451	2	101	16	25	37

Figure 55: Improper Performance of the Reflectance Sensor(the sixth sensor which has a red circle on the figure constantly gave us the value of 3000)

How to improve:

What we can do is move the horizontal reflectance sensor array slightly to the right. Instead of using all 9 horizontal sensors, we will only use 5 sensors. We will reset the center sensor to be the third one instead of the fifth. This way, the broken sixth sensor will no longer be part of the line-following process, allowing our robot to perform as expected.

What we can also do is ignore the fourth sensor symmetrically, using only seven horizontal sensors instead: the 1st, 2nd, 3rd, 5th, 7th, 8th, and 9th. This configuration will create gaps between the 3rd and 5th sensors, and between the 5th and 7th sensors. Unfortunately, the overshoot might be somewhat significant since our robot will only adjust its direction when the 3rd and 5th sensors detect the black line, potentially causing the robot to oscillate during the process. However, it should still be able to maintain line tracking.

13 Discussion and Conclusion

The biggest takeaway from this project is test, test, test. We learnt so much by testing our robot, making many iterations and prototypes, tweaking small changes like frame angles, base configurations, and wheel sizes.

We worked very well as a team, assisting each other when we were stuck on something. We also learnt that a second perspective on something is integral to building and consolidating a product.

We faced many challenges from wiring issues and current limits from the power supply, to changing our motors three times, we adapted to each issue and came out with a better iteration each time. Our mechanical design became more and more stable and durable with each prototype, with the robot easily overcoming the stairs, ramps, and Giant's Causeway.

However, we encountered a reoccurring issue with the wheels. The rubber treads around our tires would consistently come off, getting caught on the wheels or the motors. They would also coming off completely, making the robot uneven and result in floating wheels or less traction. We decided not to glue these onto the wheel for reusability and in the case that we wanted to swap out our wheels at any point. One of the main advantages we had with our design was the modularity. We were able to change wheel sizes easily to test different combinations, detach the reflectance sensor arm to experiment with different sensor placements, and explore the different types of hook mechanisms. For the hook, we changed from a two claw design that sweeps from the sides up to a scissor lift that allowed Bumblebee to be much more compact.

One of our major concerns was the size of the robot, being approximately 27cm in width and 29cm in height with the scissor lift retracted. The length was less concerning as it was integral in overcoming the lava pit. The size was naturally restrained by the entrance to the tunnel at 30x30cm (width by height). However, in the end, the size of the robot did not turn out to be an obstacle, as it could still complete sharp turns and stay on the track.

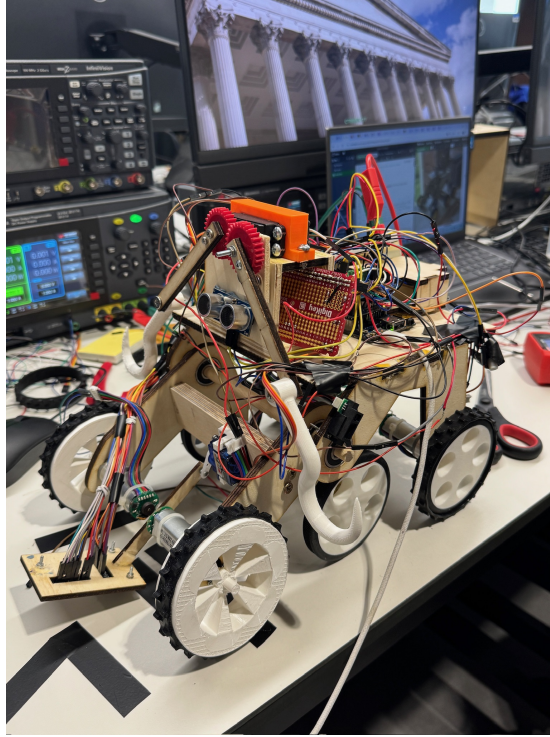


Figure 56: Assembled prototype

Overall, the past 6 weeks have been a very challenging but enjoyable experience filled with wires, laser cutters, and sensors. We are proud of the final product and how Bumblebee performed on the race track, and the progress we have made along the way.

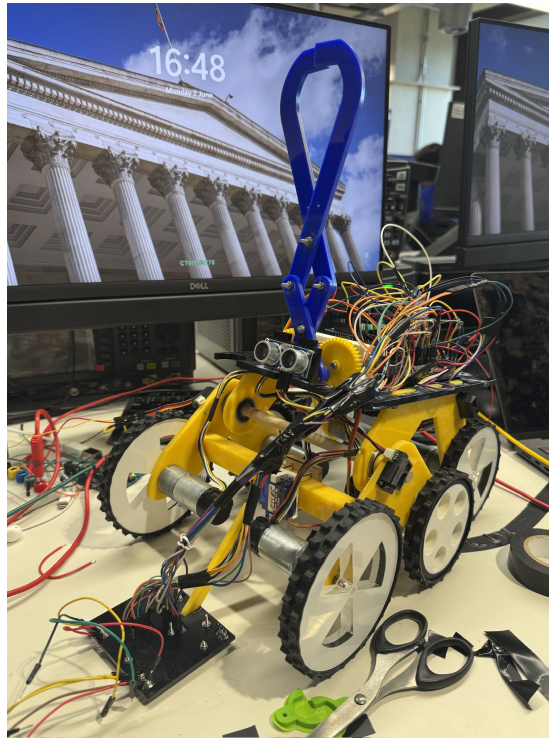


Figure 57: Final design of Bumblebee

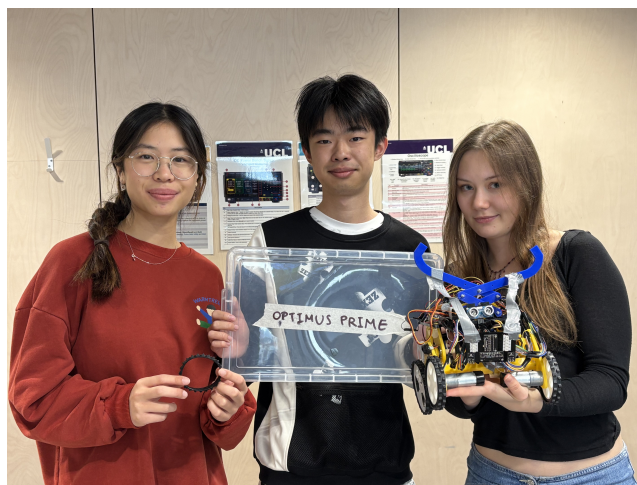


Figure 58: Team photo with final robot

14 References

References

- [1] YouTube. (2017). *6-Wheel Robot with Climbing Mechanism*. Retrieved from <https://www.youtube.com/watch?v=90d0E53R8PE>
- [2] Digilent. (n.d.). *IG22 Motor Gearbox Datasheet*. Retrieved from https://digilent.com/reference/_media/motor_gearbox/290-006_ig220019x00015r_ds.pdf
- [3] Chaudhari, U. H. (2023). *The Power of Rocker-Bogie Suspension System*. International Journal of Novel Research and Development. Retrieved from <https://www.ijnrd.org/papers/IJNRD2306140.pdf>
- [4] Harrington, B. D., Voorhees, C. (2004). *The Challenges of Designing the Rocker-Bogie Suspension for the Mars Exploration Rover*. 37th Aerospace Mechanisms Symposium. Retrieved from <https://ntrs.nasa.gov/api/citations/20040084284/downloads/20040084284.pdf>
- [5] SunFounder. (n.d.). *Understanding and Making Rocker-Bogie System*. Retrieved from https://docs.sunfounder.com/projects/galaxy-rvr/en/latest/lesson2_rocker_bogie.html