- **Team Name:** SCS-DAM ProMad.
- **Team Members:** Jolian Wassouf, Khalil Al-Debs, Zaid Wassouf.
- **Team Coach:** Ebraheem Moalla.

Table of Contents:

1. Introduction:

In pursuit of comfort, humans have always sought easier ways of travel, one example being the self-driving car. An “autonomous” car is a vehicle capable of sensing its environment and operating without human involvement. Driven by curiosity, humanity has reached the necessary technological advancements to achieve a truly autonomous car. Thus, in our project we strive to design, then build a car that is able to move independently while making optimal decisions. Moreover, we extend this work to incorporate obstacle avoidance. In order to achieve our objectives, we developed an integrated system using a Raspberry Pi 4 Model B, which receives and processes information to make decisions before transmitting signals to execute specific commands.

2. Solution:

We followed a solution of two parts, one for each task.

1. Open Challenge:

To address the challenge, we developed a **camera-based algorithm** utilizing the **Raspberry Pi 4 Model B** alongside the **Pi Camera Model 3**. This system enables **precise linear alignment**, ensuring the robot maintains a **consistent trajectory** along the mat’s walls throughout the **four straightforward sections**. The **steering mechanism** is dynamically adjusted based on **color recognition**, allowing the robot to detect and respond to **walls** when encountering a **corner**. Additionally, the system incorporates a **lap-counting function** relying on the strips in corners, enabling the robot to track its **completed laps** before executing a **stop command**. This **integrated approach** enhances **autonomous navigation, spatial awareness, and real-time decision-making,** optimizing the robot’s overall performance. Lines of code in addition to performance videos are provided in the Links section.

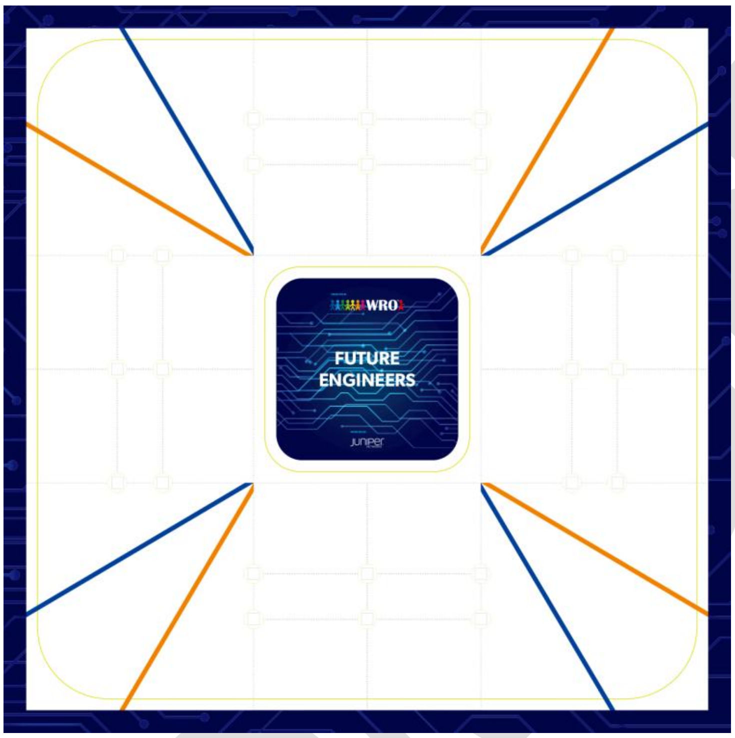


Figure 1: The Game Field Mat

To further elaborate on this mechanism, the following is a step-by-step breakdown of its functionality:

1.1. Starting The Robot:

Once the robot is placed on the **mat**, we close the robot's circuit using a simple switch to provide it with power. Then, it initiates the **program execution** with a **button press**, activating its **driving mechanism.**

1.2. Image Processing:

The **Pi Camera** captures an image of the robot's **field of view**, transmitting it to the **Raspberry Pi 4 Model B** for **processing.**

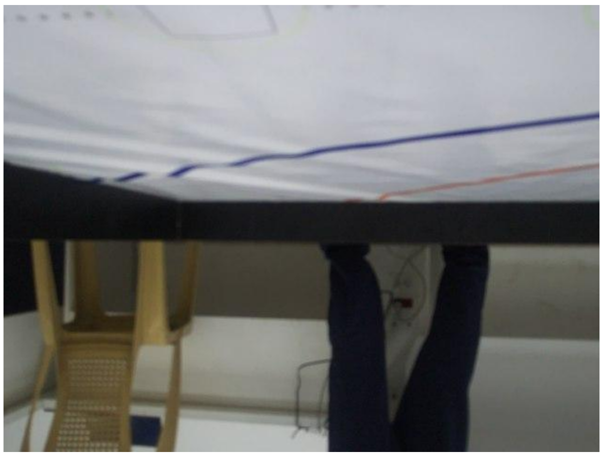


Figure 3: Raw Image

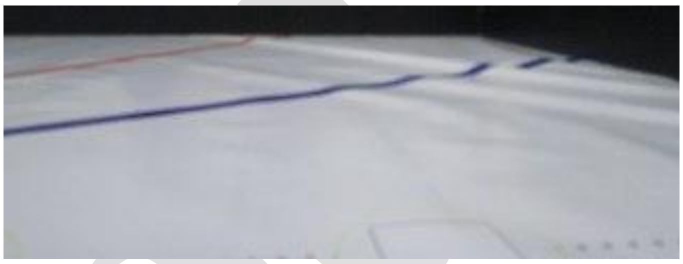


Figure 2: Processed Image

Much like **the human eye**, the **camera is positioned** to capture a **flipped image** at a **resolution of 640×480 pixels.** Upon receiving the photograph from **the PiCamera 3**, the **Raspberry Pi 4** immediately initiates **the processing sequence.** This sequence involves **flipping the image, adjusting its resolution,** and **performing color detection.**

With more details, the **Rpi4 reduces the resolution to 320×120 pixels** and **removes the top 40 rows** to isolate the area containing **the mat.** Keeping in mind that it **structures the pixel data into arrays using Python,** while **leveraging the NumPy library** for **efficient manipulation.** This technique not only facilitates **resolution reduction** but also contributes to **image inversion,** as the **pixels are reordered in reverse** to produce **the correct visual phase.**

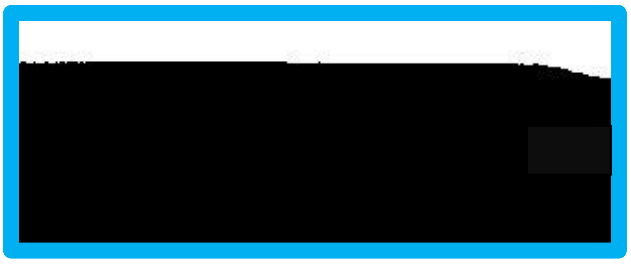


Figure 5: Walls Detection – walls are the white figures

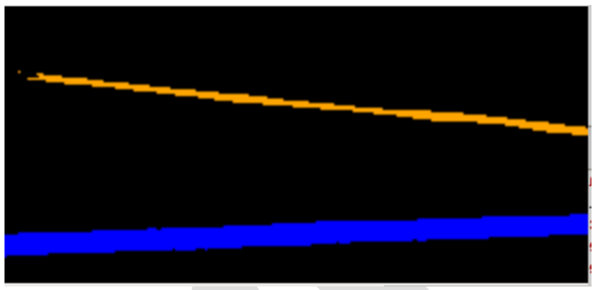


Figure 4: The Robot's View of The Lines

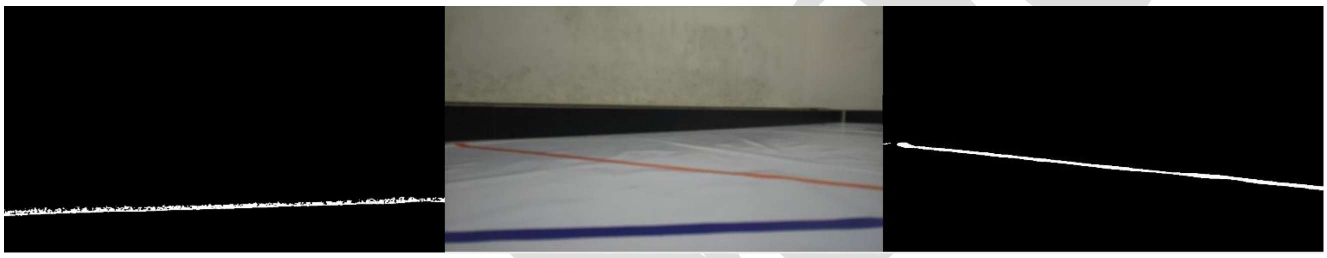


Figure 6: The Identification of Strip Colors

We arrive at the **final stage of the image processing sequence: color detection**. The **Raspberry Pi 4** is responsible for identifying a variety of colors by relying on the **HSV (Hue, Saturation, Value) color model**, which ensures **optimal detection accuracy**. Specifically, the algorithm recognizes **walls as black pixels**, while **blue and orange pixels** indicate **corner strips.**

Moreover, **color detection** is contingent upon a **threshold condition** being met. If the **pixel count** does not surpass this threshold, the robot disregards the **orange and blue corner strips**. However, if a strip is successfully detected, its **state is set to 1**, indicating **active detection.** Once the strip moves out of view, the **state transitions to 2**, initiating a **cooldown phase** before the next strip can be registered.

Additionally, the algorithm designed for **linear alignment** relies on detecting the **proportional presence of either the left or right walls.** To facilitate this, the **Rpi4 splits the image into two halves**—one representing the **left wall's black pixels**, and the other capturing the **right wall's black pixels**—allowing the system to evaluate wall alignment more accurately based on distribution and positioning.

1.3. Analysis of Given Information and Solving the Challenge:

As the **Rpi4** interprets the robot's **field of view,** it issues **optimal commands** to execute the designated task. The following outlines these commands step by step:

Step One: Upon activation—triggered by pressing the **start button**—the **DC motor** immediately begins rotating at a speed of **200 RPM.** Simultaneously, the **PiCamera** initiates **image analysis.**

Step Two: The **PiCamera** processes **30 frames per second.** For each frame, the **Rpi4** evaluates the distribution of **black pixels** across both halves of the image. It then performs **proportional calculations** to assess the robot's **alignment** relative to the surrounding **walls** and its **distance from the outer boundary.** If the robot is properly aligned, the **Rpi4** maintains the **steering wheels** in a forward position without altering the angle. However, if any **misalignment** is detected, it adjusts the **steering angle** via the **Servo motor** using a **proportional (P) controller** to restore proper orientation.

Step Three: When the robot encounters a **corner,** the **wall that the robot relies on for the proportions**—whether it is the **left or right wall**—is determined by the **color of the first strip** identified: **orange strips** signal a **clockwise turn** (left wall proportion), while **blue strips** indicate a **counter-clockwise turn** (right wall proportion).

Furthermore, these **colored strips** serve as **corner counters.** Each time the robot detects a strip, it **increments its quadrant count by one.** Once the count reaches **twelve,** the robot **terminates its operation.** Plus, to prevent the **simultaneous detection** of both strips, a **brief delay** is introduced following the observation of one.

Added Feature: An **Assembly LED** was installed to provide **visual feedback** on the robot's **readings.** It reflects key states such as **the detection of different colors.** Further details are addressed in a separate directory.

2. Obstacle Challenge:

The **solution implemented for this challenge** integrates a **color-based navigation system** to complete the **laps around the mat** and execute the **parking process.** The robot utilizes its **Pi Camera** to detect **red and green pillars,** which serve as **steering indicators.** Upon identifying a **red pillar,** the system transmits signals to the **Raspberry Pi,** prompting a **rightward steering adjustment.** Conversely, detecting a **green pillar** triggers a **leftward turn,** ensuring **precise directional control.** The **parking lot** is consistently positioned within the **starting section,** allowing the robot to **initiate its final maneuver** upon completing **three laps.** At this stage, the system scans for **magenta pixels,** identifying the **parking lot's location.** Following this step, the robot **navigates toward the designated area,** executing a **controlled parking sequence.** This **integrated approach** enhances **autonomous navigation, obstacle recognition, and precise parking alignment,** optimizing the robot's overall performance. Lines of code besides performance videos are provided as a QR code in the Links section.

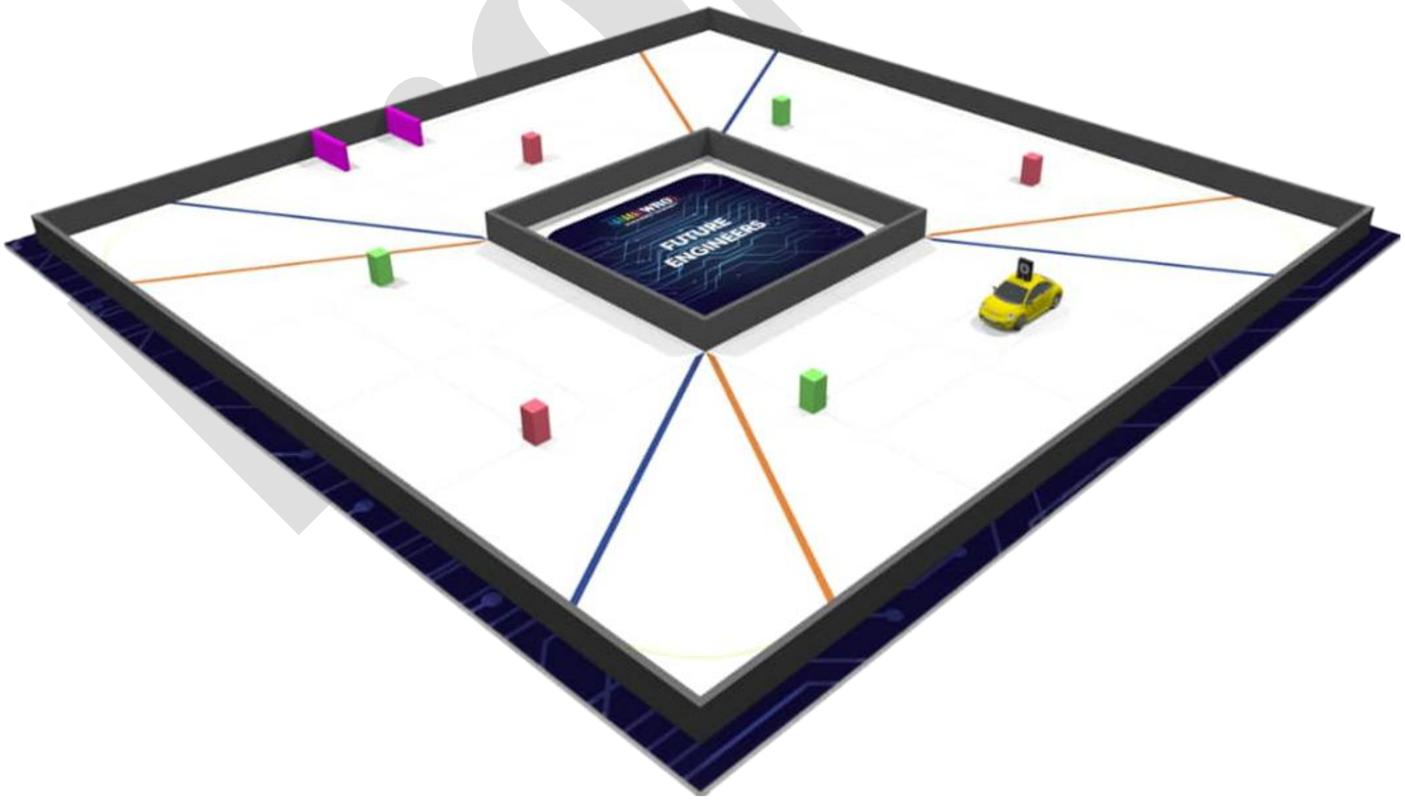


Figure 7: Obstacle Challenge Map

Let’s dive into this mechanism deeper with the following steps:

2.1. Starting The Robot:

In the obstacle challenge, the robot's mechanism is activated by first turning on the main power switch, which initiates the circuit. The program is then manually started by pressing a push button. Following this boot-up sequence, the robot's image processing begins, using the same methodology as in the open challenge. This involves capturing and analyzing visual data to guide the robot's subsequent actions.

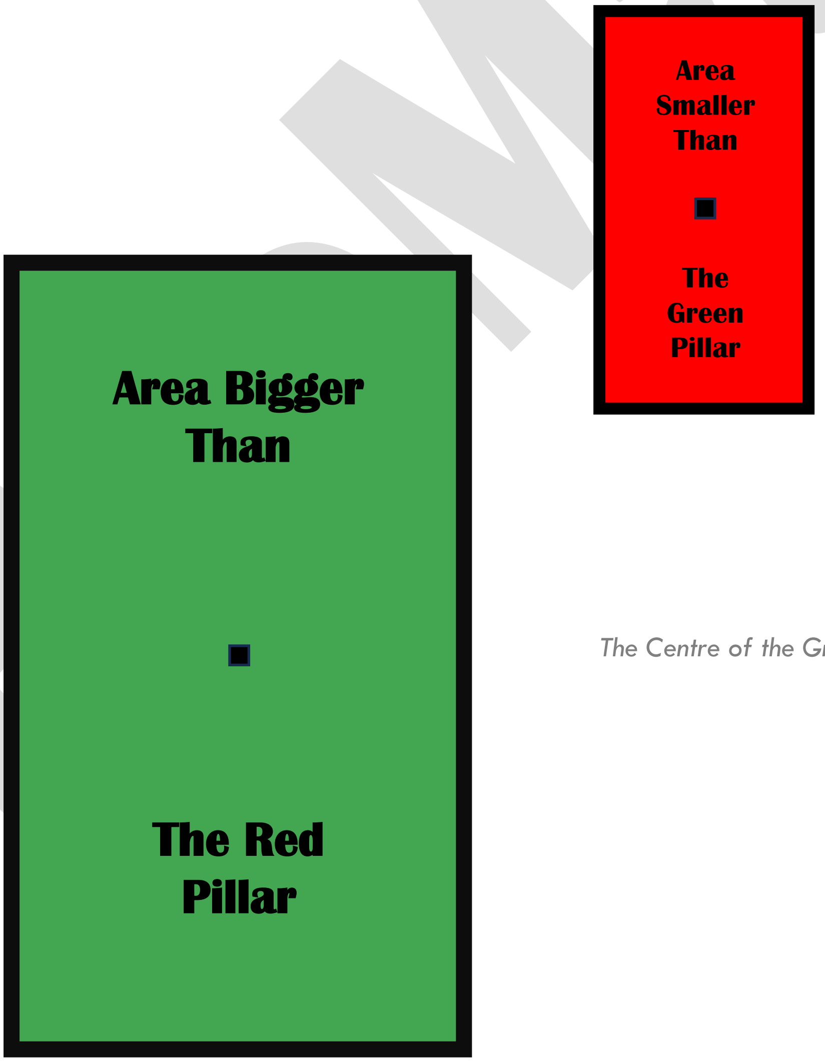
2.2. Object and Wall Detection:

Once the image is in the HSV format, the *process_hsv()* function analyzes the pixels to identify key elements:

- **Walls**: Pixels with a low **value** (brightness less than 70) are identified as walls. The code calculates the density of these pixels on both the **left and right sides** of the frame, providing an estimate of the robot's proximity to a wall. This is a **crucial input for the steering algorithm**.
- **Traffic Lights**: The script identifies red and green traffic lights based on their **hue** and **saturation** values. For instance, **red traffic lights are identified by a hue value less than 15 or greater than 175**, while **green traffic lights are identified by a hue value between 45 and 90**. These pixels are used to create separate masks for each color.
- **Parking Gate**: A purple parking gate is detected using a similar method, identifying pixels within a specific **hue, saturation,** and **value** range to create a mask.

2.3. Contour Analysis:

After the masks are generated, the script uses **cv2.findContours()** to identify contiguous shapes of each color. The **process_traffic_contours()** and **process_parking_contours()** functions then iterate through these shapes (contours) to find the largest one, which is assumed to be the target object. The **x and y coordinates** of the centroid of the largest contour are stored, along with its area and type (red or green).steps.



The Centre of the Green Pillar is Taken

2.4. Steering and Motor Control

This is where the robot acts on the visual data. The steering direction is calculated using a **proportional control (P-controller)** algorithm. The **Err** variable, or error, is calculated based on the horizontal position of the detected traffic light. A **positive error value indicates the traffic light is to the left of the center, and a negative value indicates it is to the right**. The steering direction, **dir**, is then calculated as *Err * kp*, where **kp** is a **constant gain of 0.30**. This makes the steering proportional to the error, causing the robot to turn more sharply the farther it is from the center. The **servo()** function then takes this *dir* value and converts it into a **PWM (Pulse Width Modulation)** signal to control the servo motor, which adjusts the robot's wheels. The script also includes a **fail-safe mechanism where if the wall density is too high, the robot will turn sharply away to prevent a collision.**

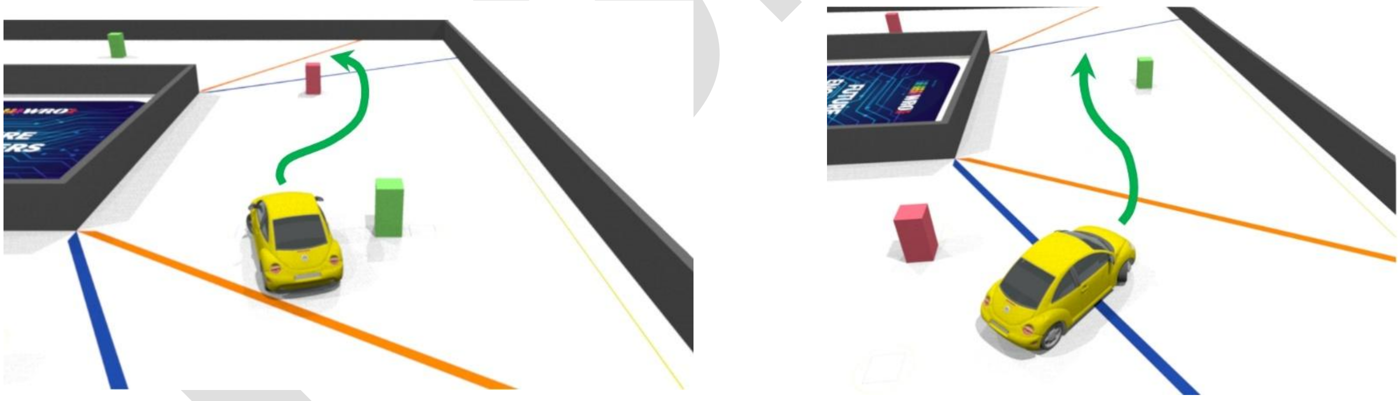


Figure 8: Obstacle Management

2.5. Mission Logic and Overrides:

The code contains a **complex state machine with various conditions that override the standard steering algorithm:**

- **Quadrant Counting:** The robot tracks its progress by counting blue and orange lines, which define different quadrants of the course.
- **Parking Sequence:** In later quadrants, the script overrides the traffic light detection and initiates a new steering routine to align with and approach the parking gate. This routine uses the **wall detection logic to align the robot parallel to the walls** before stopping when the purple parking gate is close enough. The **STOP** variable is set to *True* to halt the robot and end the program once the mission is complete.

3. Mobility Management:

To clarify how we managed the robot's mobility, we're willing to discuss several aspects:

1. The Steering Assembly:

To steer the robot properly, we used:

- **MG90s Servo Motor:**

The MG90s servo motor was specifically selected for this application due to its **compact dimensions and its capacity for exceptionally precise angle management.** Operating within a **voltage range of 4.8V to 6V,** it facilitates a **full 180° rotation span,** which is ideal for angular control tasks. Internally, its accurate motion is regulated by a **sophisticated control circuit, which, coupled with a potentiometer for positional feedback,** directs a DC motor and gear train. This regulation is achieved via **Pulse Width Modulation (PWM) signals,** where specific pulse durations within a 20ms cycle correspond to exact angles—a **1.5ms pulse for the center position, with 1ms and 2ms pulses for the opposing extremes.**

Furthermore, its performance capabilities are substantiated by a **stall torque of approximately 1.8 kg-cm at 4.8V** and a **rapid response time of 0.10–0.12 seconds per 60° of rotation.** The unit's lightweight design, high responsiveness, and seamless integration through a **simple three-wire interface (power, ground, and signal)** affirm its suitability for tasks demanding precise and dynamic motion control.

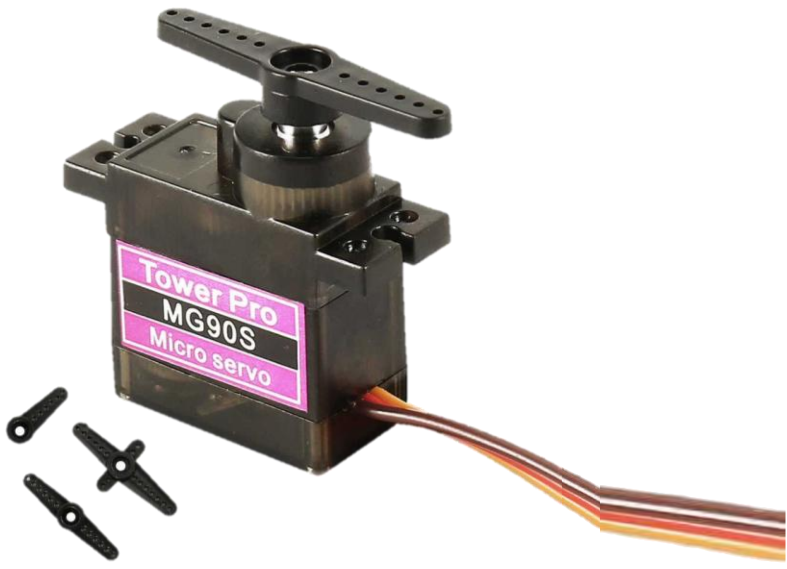


Figure 9: MG90s Mini Servo Motor

- LEGO Technic Parts:

- LEGO Technic Beam 9 – 2 were used.
- LEGO Technic Thin Beams (length not more than 4 holes) – 2 were used.
- LEGO Technic Axle Connector with Cross Insert and Beam Interface – 2 were used.
- LEGO Technic T-Beam – 1 was used
- LEGO 30.4 x 14 VR Wheels – 2 were used.
- LEGO Technic Pin with Friction Ridges 1L – 2 were used.
- LEGO Technic Pin with Friction Ridges 2L – 2 were used.
- LEGO Technic Axle with Pin with Friction Ridges 1L – 6 were used.

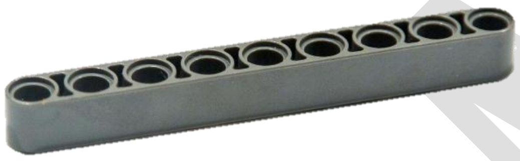


Figure 14: LEGO Technic Beam 9

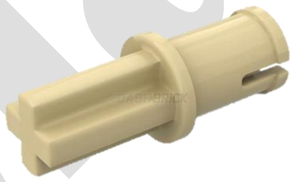


Figure 15: LEGO Technic Axle with Pin with Friction Ridges 1L

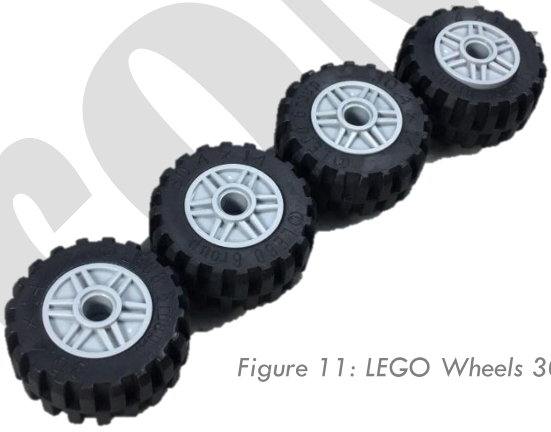


Figure 11: LEGO Wheels 30mm

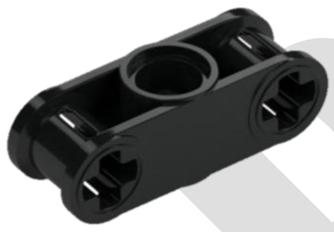


Figure 17: LEGO Technic Axle Connector with Cross Insert and Beam Interface

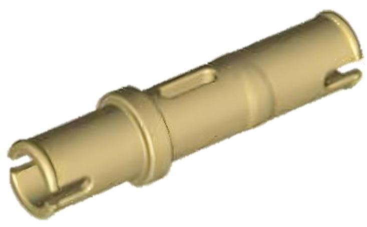


Figure 16: LEGO Technic Pin with Friction Ridges 2L

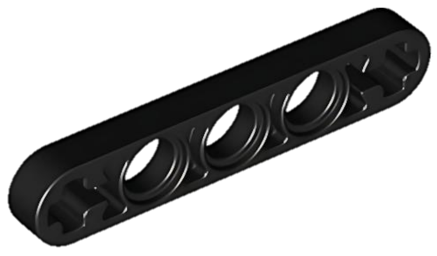


Figure 13: LEGO Technic Thin Beams

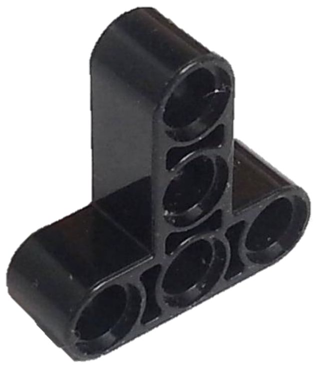


Figure 10: LEGO Technic T-Beam

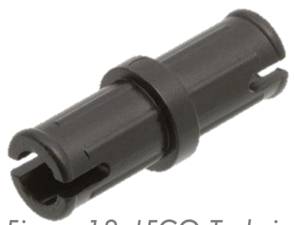


Figure 12: LEGO Technic Pin with Friction Ridges 1L

We selected LEGO parts for their compact size and structural integrity, which proved instrumental in developing the intricate steering mechanism.

We also provide a video of the robot's assembly here: Robot's Assembly.

In addition to the following photos:

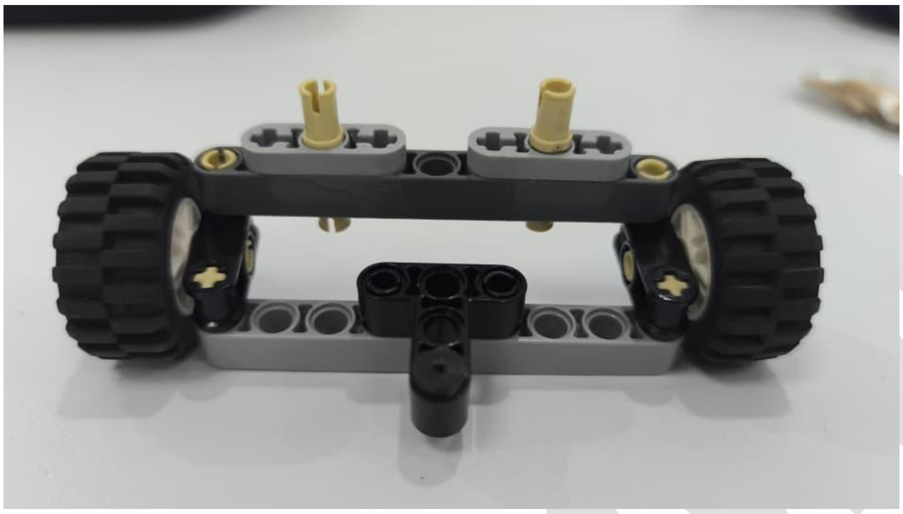


Figure 20: Steering Assembly

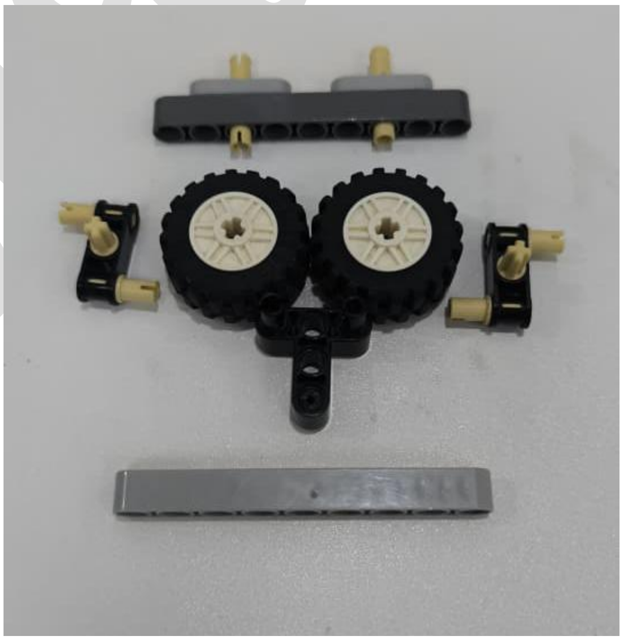


Figure 19: Steering Parts

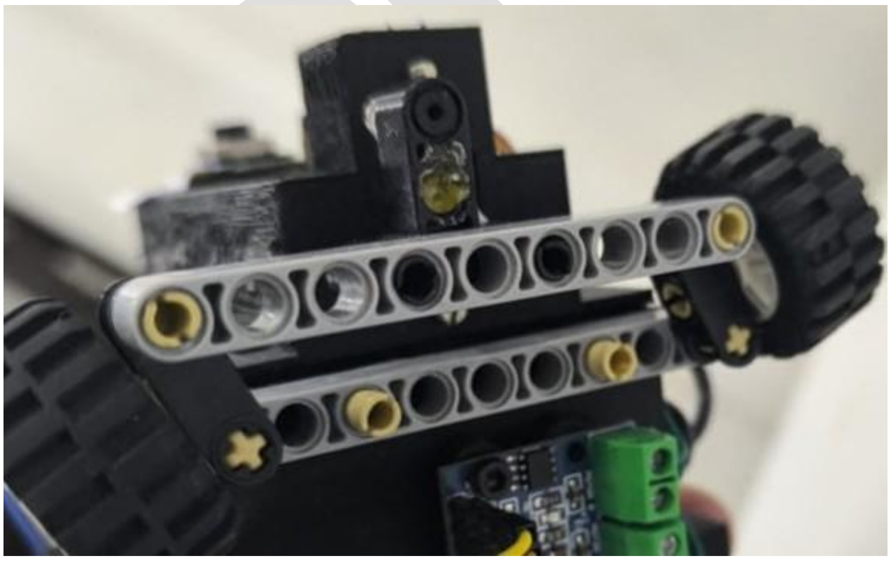


Figure 18: Steering on The Robot

- **3D Part:**

This component served as a **screw holder,** connecting the T-Beam and the servo motor, as illustrated in Figure 21.

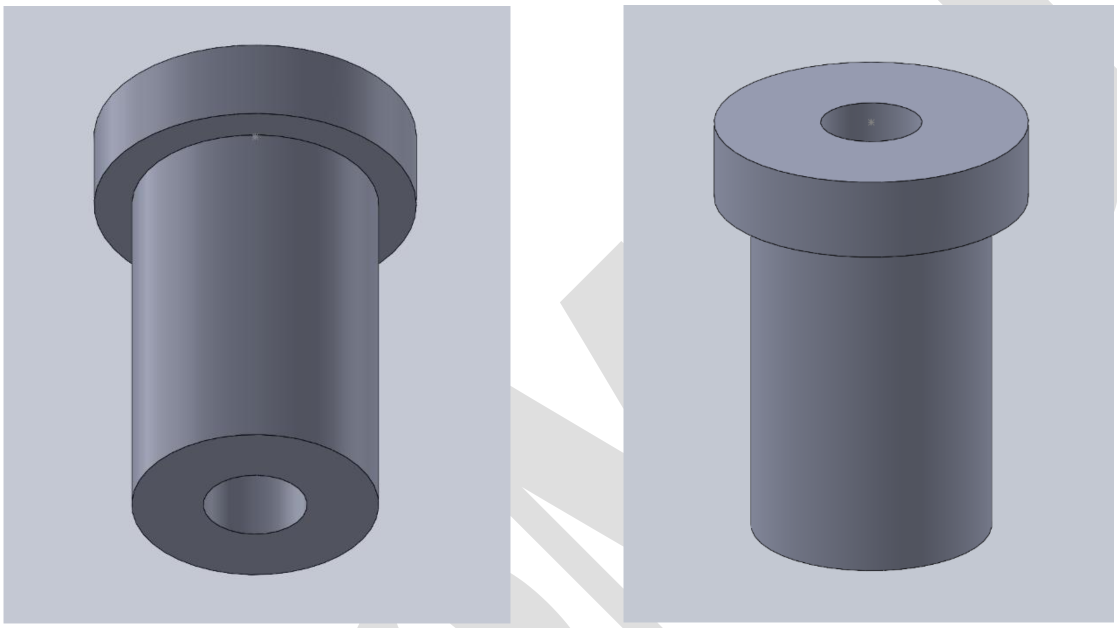


Figure 22: Servo – Tbeam Screw Connecter

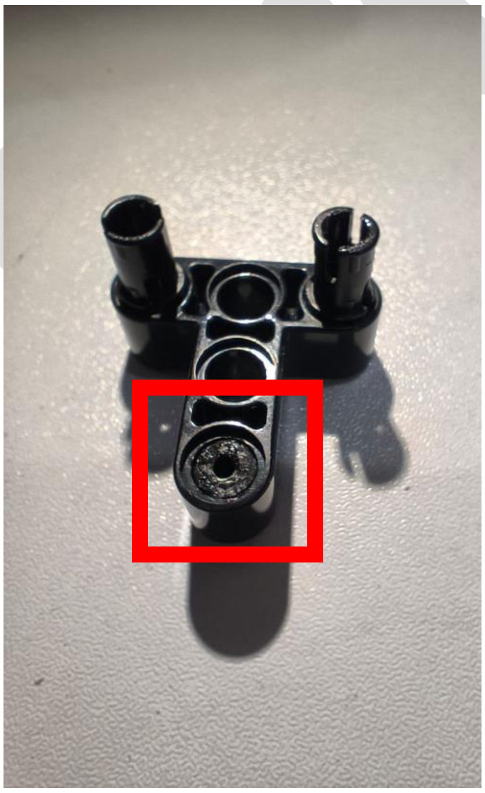


Figure 21: The Part in the T-Beam

The Steering Algorithm:

The steering logic is implemented in the *cycle()* function, which is continuously called in the main loop to provide a near real-time response. In the **open challenge**, the algorithm steers the robot by **detecting walls** on either side of the path. It calculates an error term based on the number of "wall" pixels detected and then determines the steering direction, *dir*, by multiplying this error by the proportional gain constant. The calculated *dir* value is then passed to the *servo()* function. Similarly, the **obstacle challenge** also uses wall detection but incorporates more complex logic based on the detection of traffic lights and parking gates. The error term, *Err*, is derived from the position of the detected traffic light, and the steering is adjusted accordingly. The *cycle()* function in both codes is executed repeatedly, ensuring the steering is constantly adjusted with each new frame captured by the camera.

2. The Driving Assembly:

The driving assembly has a simple idea, let's navigate it with starting with the components first:

- **GA12-N20 Geared Mini DC Motor:**

The N20 DC 12V 150RPM gear motor was an ideal choice for this project. We selected this motor specifically for **its power efficiency, compact size, and potent drive power,** all of which were crucial to our robot's design. The **motor's small dimensions were paramount, as they allowed for a streamlined and space-efficient build.** Furthermore, the **motor's high-torque output,** which delivers a rated torque of approximately 2.5 kg-cm and a stall torque reaching nearly 8 kg-cm under heavy loads, **provides the robust and responsive movement required for our system.** A durable metal gearbox further enhances longevity and reduces wear over time. This motor's compact design, impressive torque efficiency, and robust construction made it a versatile and essential component for our technical application, **ensuring reliable performance even in demanding conditions.**

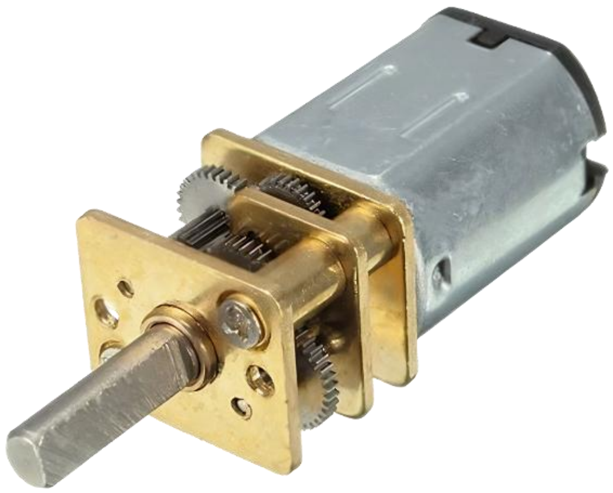


Figure 23: GA12-N20 Geared Mini DC Motor

- **L9110 Motor Driver:**

The **L9110 motor driver** is a highly efficient **PWM-based control IC** designed for **high-voltage applications**, particularly in **automotive and industrial environments**. Engineered to regulate **LED brightness and motor speed**, it operates within a **wide voltage range**, ensuring adaptability across various systems. The **L9110** supports **high-current outputs**, making it suitable for **power-intensive applications**. Its **integrated protection mechanisms**, including **overcurrent and thermal shutdown**, enhance reliability and longevity. The **PWM control** ensures **precise modulation**, optimizing energy efficiency while minimizing heat dissipation. With its **compact design** and **robust electrical specifications**, the **L9110** stands as a versatile solution for engineers seeking **stable and efficient motor control.**

In our project, it used with the DC Motor.

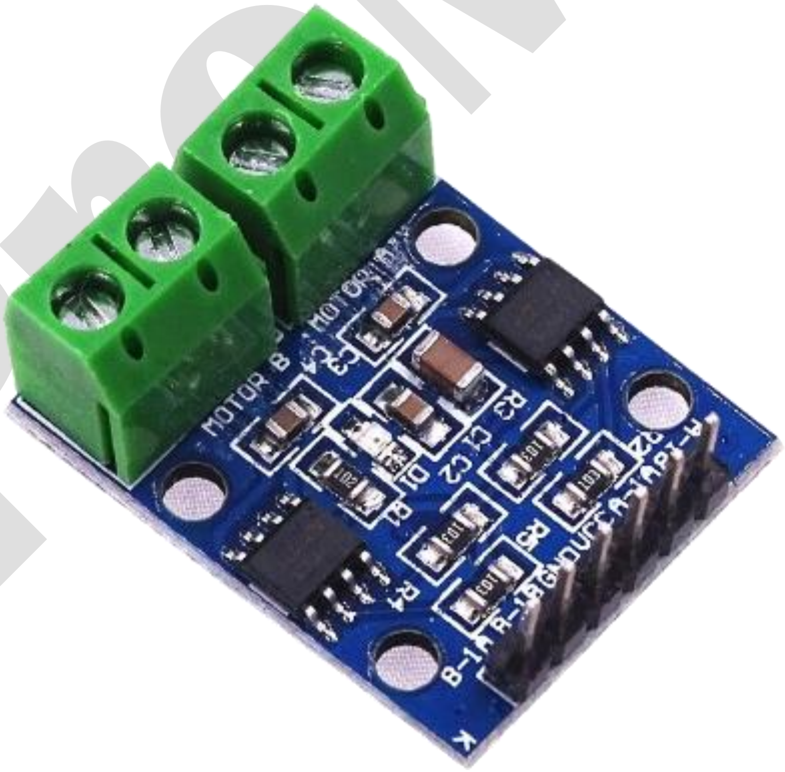


Figure 24: L9110 Motor Driver

- **LEGO Parts:**

- LEGO 30.4 x 14 VR Wheels – 2 were used.
- LEGO Technic Splined Shaft / Axle Component 5L (40mm) – 2 were used.
- LEGO Technic Differential Gear – 1 was used.
- LEGO Technic Bavel Gear 1 – 3 were used.
- LEGO Technic Bush – 4 were used.
- LEGO Technic Gear 24 tooth – one was used.

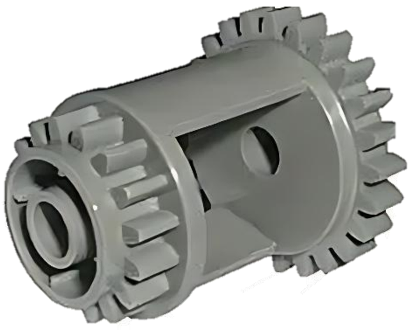


Figure 30: LEGO Technic Differential Gear

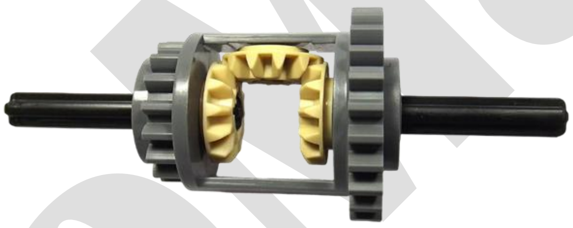


Figure 27: Differential Assembly

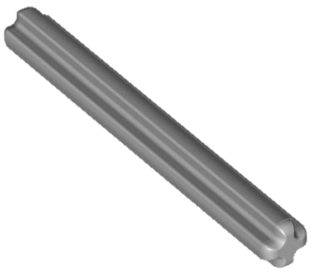


Figure 26: LEGO Technic Axle 5L

Figure 29: LEGO Technic Bavel Gear 12 tooth

Figure 25: LEGO Technic Gear 24 tooth

Figure 28: LEGO Technic Bush

- **3D Part:**

We prepared an alternative to the 24-tooth LEGO gear. A video demonstrating the assembly of the robot can be found here: Robot's Assembly.

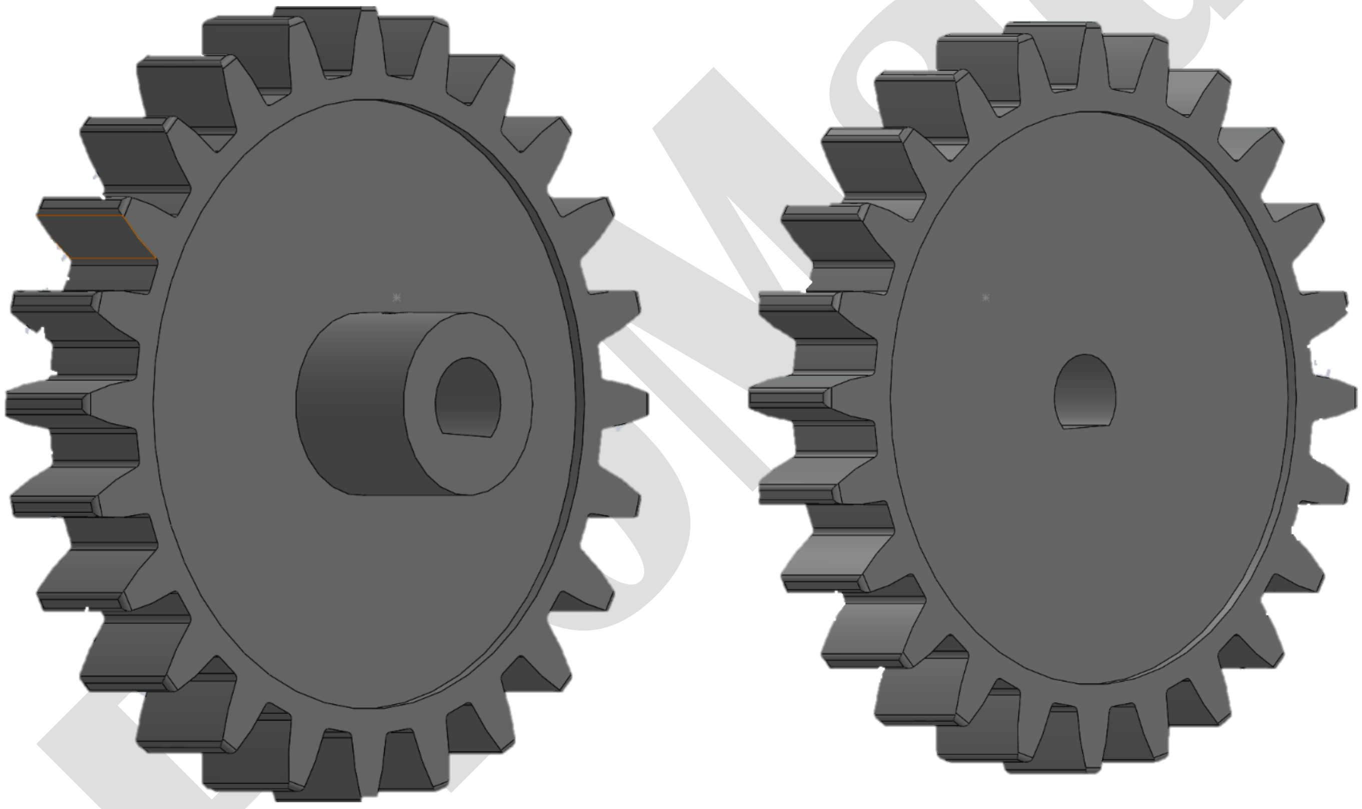


Figure 31: DC Geared Motor's Gear

3. The Robot's Body / 3D Design:

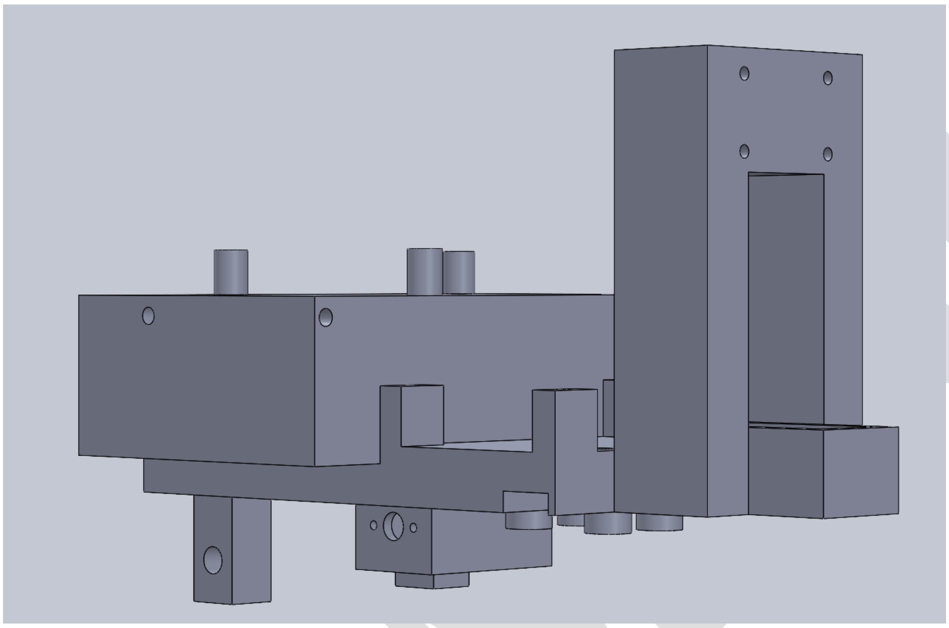


Figure 33: The Robot's Body 3D Design

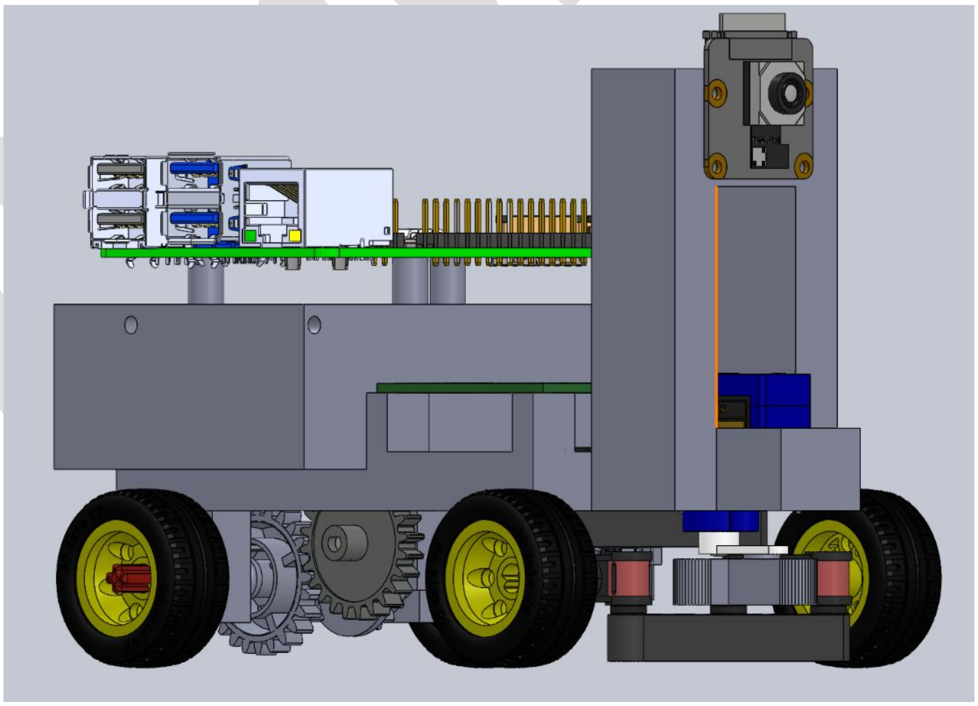


Figure 32: The Robot's 3D Design

In selecting the robot's body, our objective was to design a novel and practical model that would effectively address the requirements of the provided challenges. We, therefore, engineered a compact model that could accommodate all necessary components while maintaining a small footprint. For a detailed guide on the robot's assembly, a link to the instructional video can be found in the "Links" section.

4. The Mobility Measurement:

In order to accurately calculate the robot's velocity and power output, it is essential to first identify the following parameters:

- The Perimeter of the wheels (Pw):

{*we need the radius of the wheels: rw = 30.4mm/2 => rw ≈ 0.015m*}

$$P_w = 2\pi r_w = 2 \cdot 3.14 \cdot 0.015 \approx 0.095m$$

- The Speed of the motor (wdc):

$$\omega_{dc} = 150Rpm$$

- The Voltage and Current supplied to the motor:

$$I = 0.8A \ || \ V = 12V$$

The Velocity of the Robot:

$$v = P_w \cdot \omega_{dc} = 0.095 \cdot 150 \approx 14.31\, m \cdot min^{-1}$$

$$v \approx 0.238\, m \cdot s^{-1}$$

The Power of the Robot

$$P = V \cdot I = 12 \cdot 0.8 = 9.6\, Watt$$

4. Power and Sense Management:

We divide this category into four sections:

1. The Robot's Power Source:

1.1. 3S LiPo Battery Pack with 20A Protection Circuit:

The **3S Lithium Polymer (LiPo) battery pack** consists of **three cells in series,** delivering a **nominal voltage of 11.1V** and a **maximum of 12.6V when fully charged.** It features an integrated **20A Battery Management System (BMS)** that protects against **overcharging, deep discharging,** and **short circuits**, ensuring **safe and reliable power delivery**.

Thanks to its **lightweight pouch construction,** the LiPo pack offers **flexibility and compactness**, which are especially advantageous in **robotic applications.** Its ability to provide **high discharge rates** with minimal voltage drop makes it ideal for systems demanding **bursts of power.** Due to its **excellent energy density, thermal stability**, and **protective features,** this pack serves as the **primary power source for our robot**, enabling **efficient, high-performance operation.**

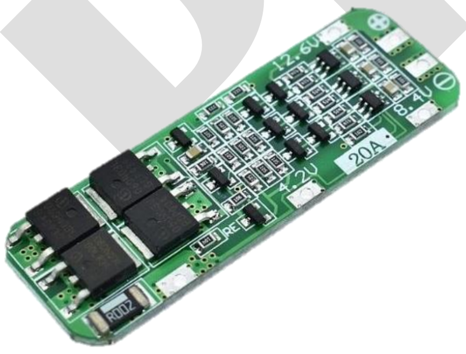


Figure 6: 20A Battery Management System (BMS)

Figure 34: LiPo Battery 2500mAh

1.2. Mini 360 DC-DC Buck Converter // 1.3. DC-DC Buck Converter Module with USB Type-A Output:

The **Mini 360 DC-DC Buck Converter** is a **compact and efficient voltage regulation module** designed to step down higher DC voltages to a lower, adjustable output. Built around the **MP2307 chip,** it operates within an **input range of 4.75V to 23V** and delivers an **output voltage adjustable between 1V and 17V.** Despite its small size—just **17mm by 11mm**—it supports a **peak current of 3A** and a **continuous current of 1.8A,** making it suitable for applications in **robotics, DIY electronics, and embedded systems.** In **our robot,** this module is specifically used to **convert the 12V supplied by the batteries into a stable 5V output,** which is essential for **powering the servo motor and the assembly LED.** Its **high conversion efficiency, thermal shutdown protection,** and **onboard potentiometer for voltage tuning** make it a **reliable choice for compact power management.**

In contrast, the **DC-DC Buck Converter Module with USB Type-A Output** is tailored for **powering USB devices directly from a higher voltage source.** Typically supporting an **input range of 6V to 24V,** it steps down the voltage to a **fixed 5V output via a standard USB Type-A port,** delivering **up to 3A of current.** This module is ideal for **charging smartphones, powering microcontrollers,** or **supplying regulated 5V to embedded systems.** In **our robot,** it serves the critical function of **converting the same 12V battery input into a suitable 5V output for the Raspberry Pi 4 Model B,** ensuring **stable operation** and **protecting the board from voltage fluctuations.** Its **plug-and-play design, built-in protection features,** and **compatibility with common USB-powered devices** make it especially useful in **automotive, solar, and portable electronics applications.**

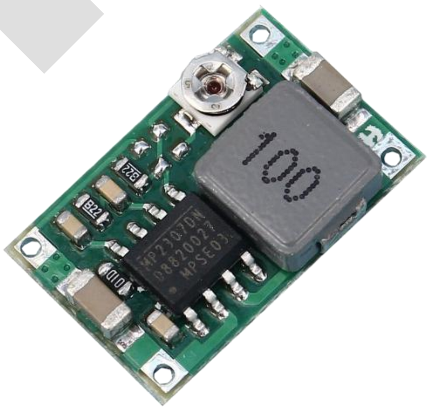


Figure 36: Mini 360 DC-DC Buck Converter

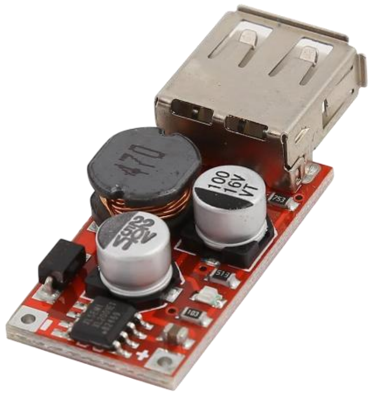


Figure 35: DC-DC Buck Converter Module with USB Type-A Output

1.4. Perfboard // 1.5. Switch:

The **Perfboard** in our robot serves as the **central hub for electrical connectivity,** linking all major components into a cohesive and efficient system. It features **two input pins** designated for the **positive and negative poles of the LiPo battery series,** which supply the primary 12V power source. Additionally, the board includes **a dedicated row of GND pins** that are directly connected to the **Raspberry Pi 4 Model B,** ensuring a stable ground reference across all subsystems. To distribute power effectively, the **perfboard** provides **six output pins for 12V** and **six output pins for 5V,** the latter being regulated through the **Mini 360 DC-DC Buck Converter.**

Furthermore, the perfboard integrates a **USB Type-A converter module,** which plays a crucial role in **delivering a regulated 5V supply to the Raspberry Pi 4 Model B.** This setup not only simplifies the wiring architecture but also enhances the reliability and safety of the robot's power distribution. By consolidating power inputs, outputs, and grounding into a single board, the perfboard ensures that all components—from servo motors to LEDs—operate smoothly and in coordination, making it an essential element of our robotic system.

And surely, we use a switch to close or open the circuit.

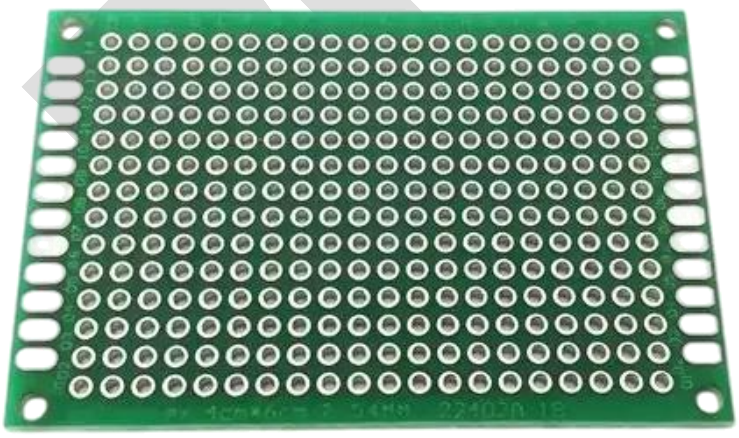


Figure 37: Perfboard

Figure 38: Switch

1.6. 470uF 25V Capacitor:

To ensure stable power delivery in our robotic system, especially during **motor startup** and **sudden load transitions,** we incorporated a **470 µF / 25 V electrolytic capacitor** across the **motor driver's supply rail.** This capacitor serves as a **local energy reservoir,** discharging rapidly when the motor demands a **surge of current.** By doing so, it prevents **voltage dips** that could disrupt the operation of critical components such as the **Raspberry Pi 4,** the **PiCamera v3,** and the **push button interface.** We selected this specific **capacitance and voltage rating** to strike a balance between **energy storage capacity** and **spatial efficiency** within the robot's enclosure. Its presence significantly reduces **transient drops** and **electrical noise,** helping us maintain **consistent system behavior** under **dynamic operating conditions**

Figure 39: 470uF 25V Capacitor

2. Main Components:

2.1. Pi Camera Module 3:

The Raspberry Pi Camera Module 3 is a state-of-the-art imaging solution designed for seamless integration with Raspberry Pi devices. Equipped with a **12MP Sony IMX708** sensor, it provides autofocus, **HDR** capability, and enhanced low-light performance, ensuring exceptional image clarity across various applications. Available in standard and wide-angle configurations, as well as **NoIR** variants for infrared imaging, it offers versatile functionality. With **full HD** video recording at **30fps**, it integrates effortlessly into Raspberry Pi's open-source camera ecosystem, making it an indispensable tool for photography, live streaming, and AI-driven image processing, thereby pushing creative and technical boundaries.

In our project, the **Pi Camera** plays a crucial role in **visual data acquisition and object recognition**. It detects **black pixels**, which represent the **walls of the mat**, ensuring accurate environmental mapping. Additionally, it identifies **orange and blue pixels**, marking the **four corner lines** for precise spatial localization. The camera further recognizes **red and green pixels**, signifying **obstacle pillars**, enabling effective navigation and collision avoidance. Moreover, it detects **magenta pixels**, which represent the **walls of the parking lot**, allowing for structured positioning within the designated area. Once the visual data is captured, the **Pi Camera transmits these signals** to the **Raspberry Pi**, where computational algorithms process the information for **real-time analysis and decision-making**. This **integrated system enhances object detection, spatial awareness, and autonomous control**, ensuring optimal performance in dynamic environments.

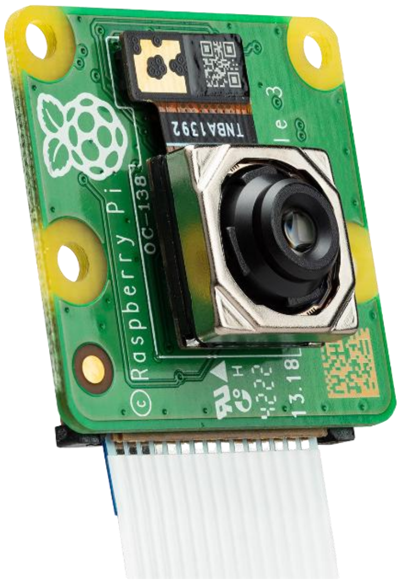


Figure 7: Raspberry Pi Camera Module 3

2.2. Raspberry Pi 4 Model B:

The Raspberry Pi 4 Model B is a high-performance single-board computer (SBC) featuring a **Broadcom BCM2711 quad-core Cortex-A72** processor clocked at **1.8GHz,** our version in this project uses 2GB of LPDDR4-3200 SDRAM. It includes dual micro-HDMI ports for dual 4K display support, Gigabit Ethernet, Bluetooth 5.0, and dual-band Wi-Fi (2.4GHz & 5.0GHz). The board offers two USB 3.0 and two USB 2.0 ports, a 40-pin GPIO header, and MIPI DSI/CSI interfaces for peripherals. The 40-pin GPIO header supports digital I/O, PWM, I2C, SPI, and UART, and it is commonly programmed using Python libraries like RPi.GPIO besides Scratch. The board is powered via USB-C (5V, 3A) and supports Power over Ethernet with an optional PoE HAT.

In our project, this **compact yet powerful device,** powered by the USB Type-A converter module, serves as the **central processing unit,** executing and managing all **computational commands** essential for system operation. It precisely coordinates the **wheel motors,** enabling smooth and controlled mobility. Additionally, it interfaces with the **camera,** facilitating image processing and environmental awareness. Beyond these core functions, the device seamlessly integrates with **various components,** ensuring efficient communication and execution of all robotic commands. Its **high-speed processing capabilities** and **multi-functional adaptability** make it a crucial element in optimizing the robot's **performance, responsiveness, and autonomous functionality.**

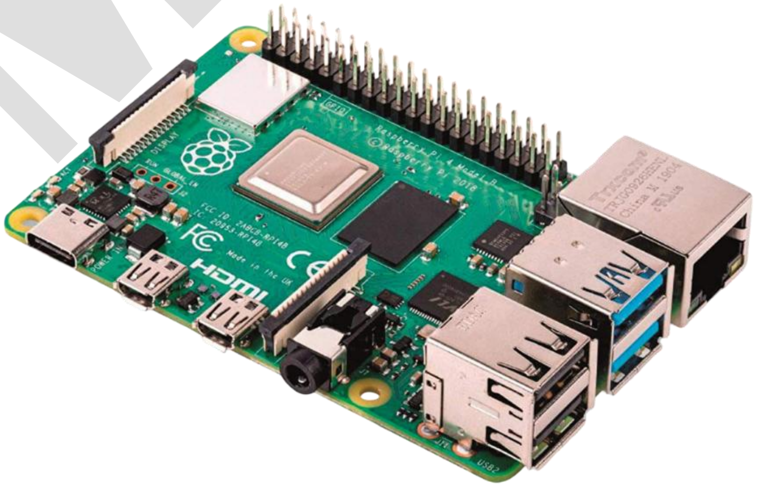


Figure 8: Raspberry Pi 4 Model B

2.3. WS2812B Assembly LED: **feedback component** // 2.4. Push Button:

The **WS2812B assembly LED** is a highly integrated, individually addressable RGB lighting solution that combines a **5050-sized SMD LED** with an embedded **control IC** in a single package. This integration allows each LED to be controlled independently via a **single-wire digital interface**, significantly simplifying circuit design and reducing the number of required microcontroller pins. Operating within a **3.3V to 5V range**, each LED can produce over **16 million colors** by modulating the intensity of its red, green, and blue channels through **8-bit PWM control**. The data signal propagates from one LED to the next, enabling seamless daisy-chaining of multiple units without complex wiring. This makes the WS2812B ideal for dynamic lighting applications, including status indicators, visual feedback systems, and aesthetic enhancements in embedded and robotic platforms.

In our robot, the **WS2812B assembly LED** functions as a dynamic **visual feedback system**, reflecting real-time sensor readings. It is powered via a regulated **5V line from the Mini 360 Buck Converter**, with ground referenced to the **Raspberry Pi 4 Model B**, and controlled through a GPIO pin using libraries like **Adafruit NeoPixel**. The LED color corresponds to environmental cues detected by the camera: when blue pixels are identified, the robot converts HSV data to RGB and displays blue; similarly, orange, red, green, and magenta are shown when their respective strips or pillars are recognized. This mapping allows quick visual confirmation of object detection and classification. Additionally, a default color is used to indicate the robot's idle or standby state, ensuring continuous status awareness during operation.

The Push Button is used to start the program.

Figure 40: Push Button

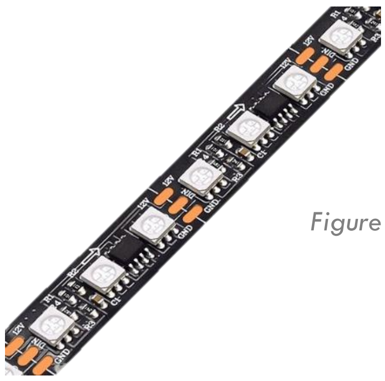


Figure 41: WS2812B Assembly LED

3. Wiring Diagrams:

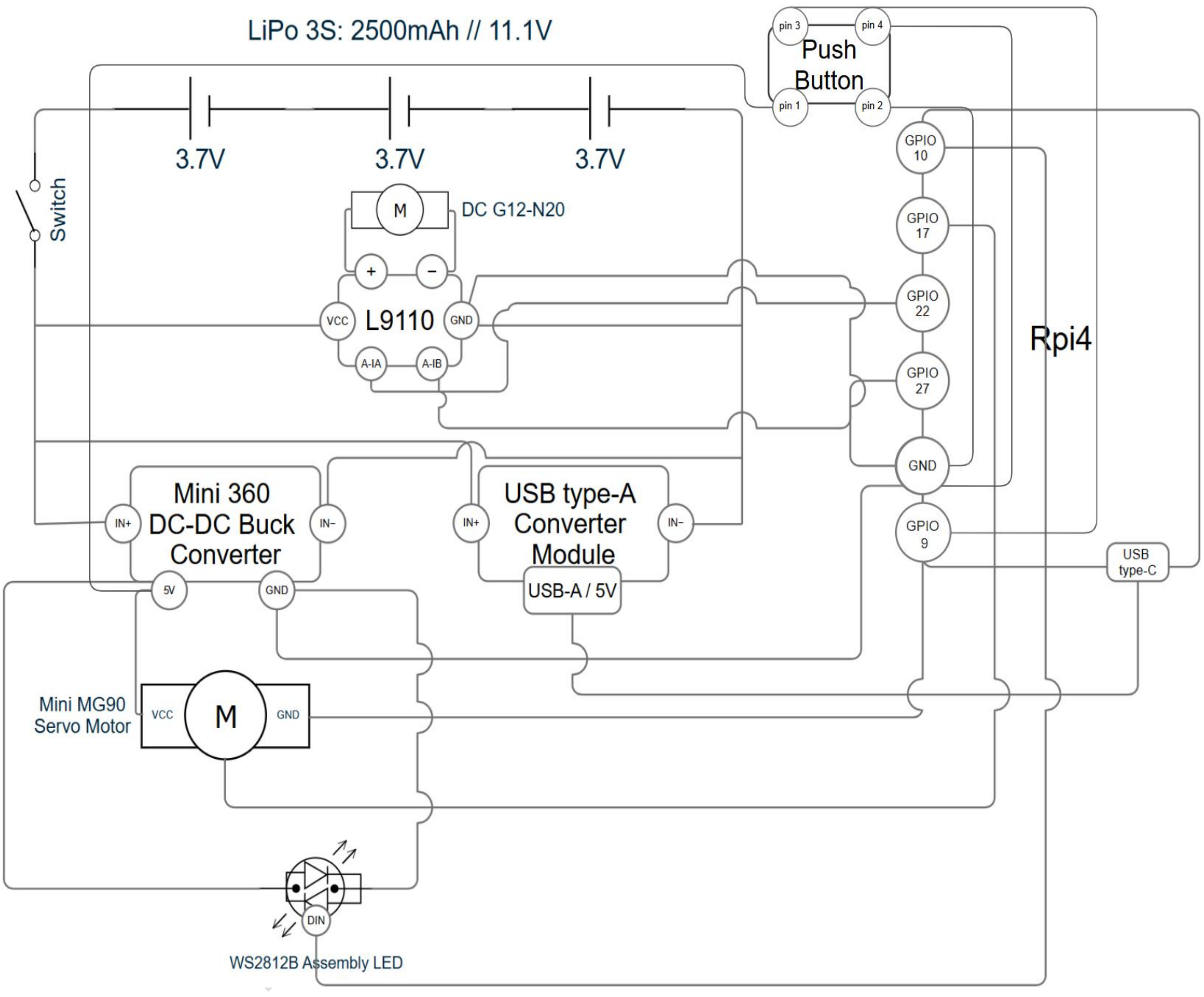


Figure 42: Schematic

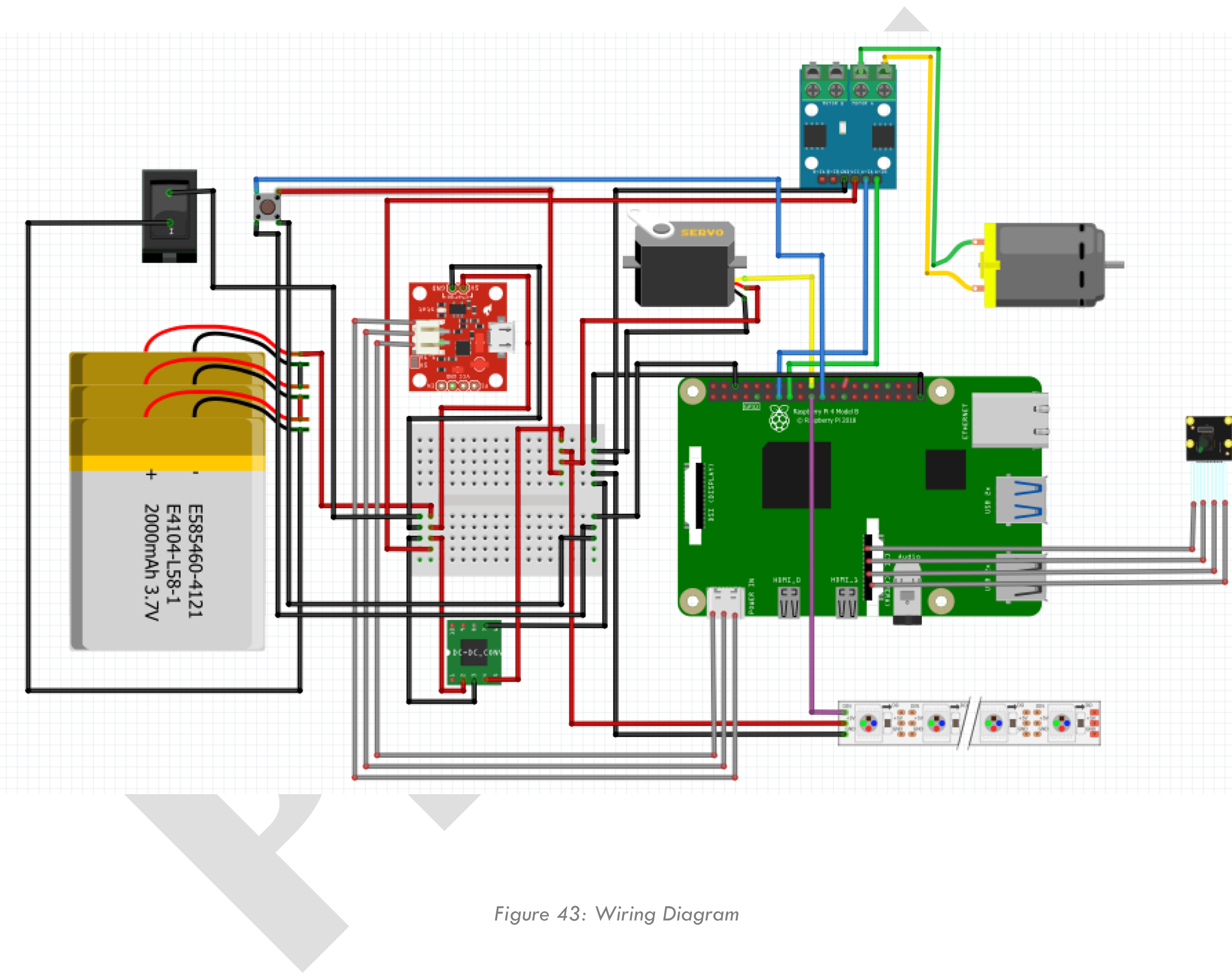


Figure 43: Wiring Diagram

4. Sensing:

4.1 Vision-Based Navigation Algorithm:

I. Image Acquisition and Pre-processing:

1. **Image Capture**: The system initializes the PiCamera and configures it to capture a 640x480 resolution image in an RGB888 format. This raw, high-resolution frame serves as the primary source of environmental data for each operational cycle.

2. **Downsampling and Flipping**: To optimize computational efficiency, the high-resolution 640x480 image is not processed directly. Instead, it undergoes a custom downsampling and vertical flipping transformation. The algorithm iterates through every second row and every second column of the bottom half of the raw frame (from row 240 to 479). This effectively reduces the image to a more manageable 320x120 resolution. The vertical flip is a result of the specific loop structure that populates the new image array from the end to the beginning, which orients the frame as if viewed from directly above and in front of the robot.

3. **Color Space Conversion**: The downsampled 320x120 RGB frame is immediately converted to the **HSV (Hue, Saturation, Value)** color space. This is a critical step because the HSV model is significantly more robust for color detection under varying lighting conditions compared to the RGB model. Hue represents the color, Saturation represents the color's intensity, and Value represents the brightness. By isolating these components, the algorithm can reliably identify colored objects regardless of shadows or bright spots.

II. Environmental Perception and Feature Extraction:

With the image properly formatted, the algorithm proceeds to analyze its content to understand the robot's surroundings. This is where the two challenges begin to show notable differences.

Core Feature Extraction (Common to Both Challenges)

Both algorithms are designed to detect three primary environmental features:

- **Wall/Boundary Detection**: The darkest areas of the image are classified as walls or boundaries. This is achieved by thresholding the **Value (V)** component of each pixel in the HSV image. Any pixel with a Value below a certain threshold (specifically, 70) is considered part of a wall. The algorithm then calculates the proportion of wall pixels on the left half of the screen versus the right half, creating left_wall and right_wall metrics. These values are fundamental for the robot's steering logic, enabling it to stay centered on the track.
- **Colored Line Detection**: The algorithm is programmed to detect two specific colored lines that serve as quadrant markers:
  - **Blue Line**: Detected by identifying pixels with a Hue between 90 and 135 and specific Saturation and Value ranges.
  - **Orange Line**: Detected by identifying pixels with a Hue between 0 and 30 and specific Saturation and Value ranges.

When the number of detected blue or orange pixels crosses a predefined threshold, the system registers the passage of a line, which is used to increment a quadrant_count. This acts as an odometer, tracking the robot's progress along the course.

Advanced Feature Extraction (Exclusive to obstacle challenge algorithm)

The **obstacle challenge algorithm** script includes a more complex perception system designed for a more challenging environment that includes traffic lights and a final parking task.

- **Traffic Light Detection**: This version of the algorithm actively searches for red and green objects, which are presumed to be traffic lights. It does this by creating binary "masks" for each color:
  - **Red Mask**: Isolates pixels with a Hue less than 15 or greater than 175.
  - **Green Mask**: Isolates pixels with a Hue between 45 and 90.

After creating these masks, it uses OpenCV's findContours function to identify distinct shapes. The script then analyzes these shapes (contours) to find the largest valid traffic light, filtering them by area and aspect ratio (vertical rectangles). This allows the robot to identify and react to traffic signals.

- **Parking Zone Detection**: In its final phase, this algorithm searches for a **purple** object, which designates the parking area. The detection mechanism is similar to that of the traffic lights, isolating pixels with a Hue between 130 and 145 and then finding the largest contour.

4.2 Camera Position:

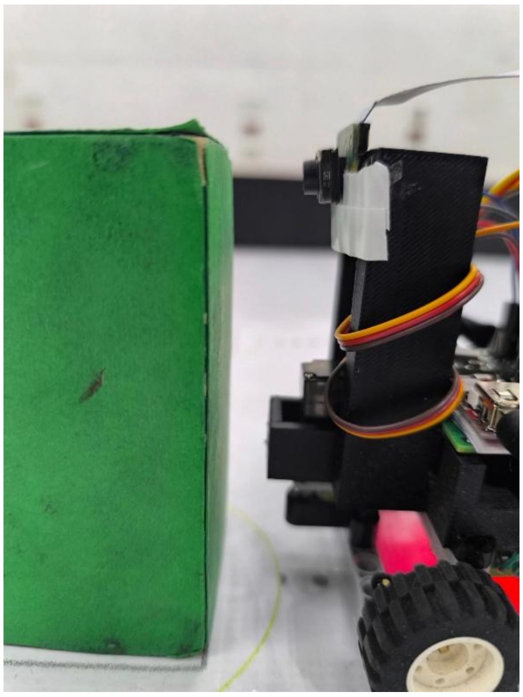


Figure 45: Camera Position

Figure 44: Camera View

CRUCIAL FOR THE ROBOT'S PERFORMANCE!!

To ensure accurate visual input, the **camera must be positioned precisely 10 centimeters above the ground from the center of the sensor,** allowing it to capture **only the designated game field**. Any elements **outside this boundary**—such as **clothing items** like **green shirts**—may be **mistakenly interpreted by the robot as part of the operational environment**. Consequently, the robot may attempt to **address non-existent tasks,** leading to **unnecessary or erroneous actions.**

To verify the correct positioning, the **camera must be placed exactly 10 centimeters from the ground,** ensuring that **no part of the pillar is visible above its top edge.** The **captured view must precisely match the reference shown in Figure 44,** confirming that the camera is aligned solely with the **intended game field.**

In the following, we aim to clarify as much as possible the importance of the camera position through some examples:

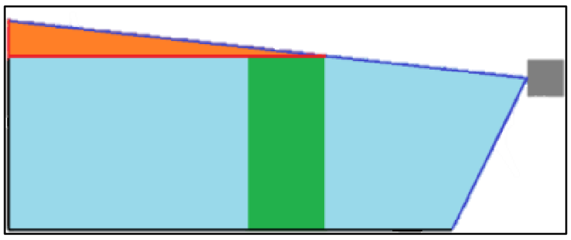


Figure 46: Camera Lower

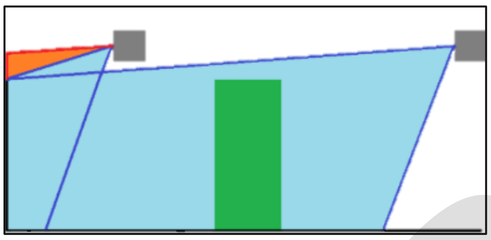


Figure 47: Camera Higher

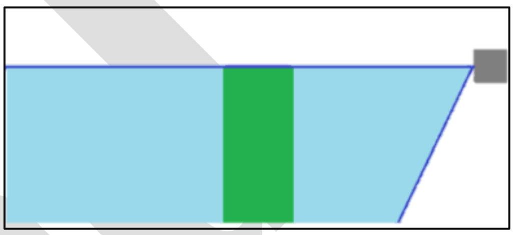


Figure 48: Correct Position

Figure 46: The camera is positioned too low, causing it to capture areas beyond the walls and outside the game field. This misalignment may result in unintended visual input.

Figure 47: The camera is positioned too high, and when it encounters the walls, it is forced to capture external elements beyond the game field, potentially leading to false interpretations.

Figure 48: The camera is correctly positioned, ensuring that only the game field is visible. With this alignment, the mission will proceed successfully and without interference.

5. The Evolution of The Robot:

5.1 Past:

In this section, we explore the earlier stages of our robot's development, including its initial design, and the key changes we made along the way.

Initial Designs:

Our primary goal was to create a robot that was both simple and user-friendly—an accessible tool for learning, yet robust enough to tackle complex challenges effectively. With that in mind, we opted for a LEGO-based structure, which successfully completed the assigned tasks. However, despite its functionality, the design presented several limitations that became apparent during testing. These shortcomings prompted us to reconsider and refine our approach, leading to significant improvements over time.

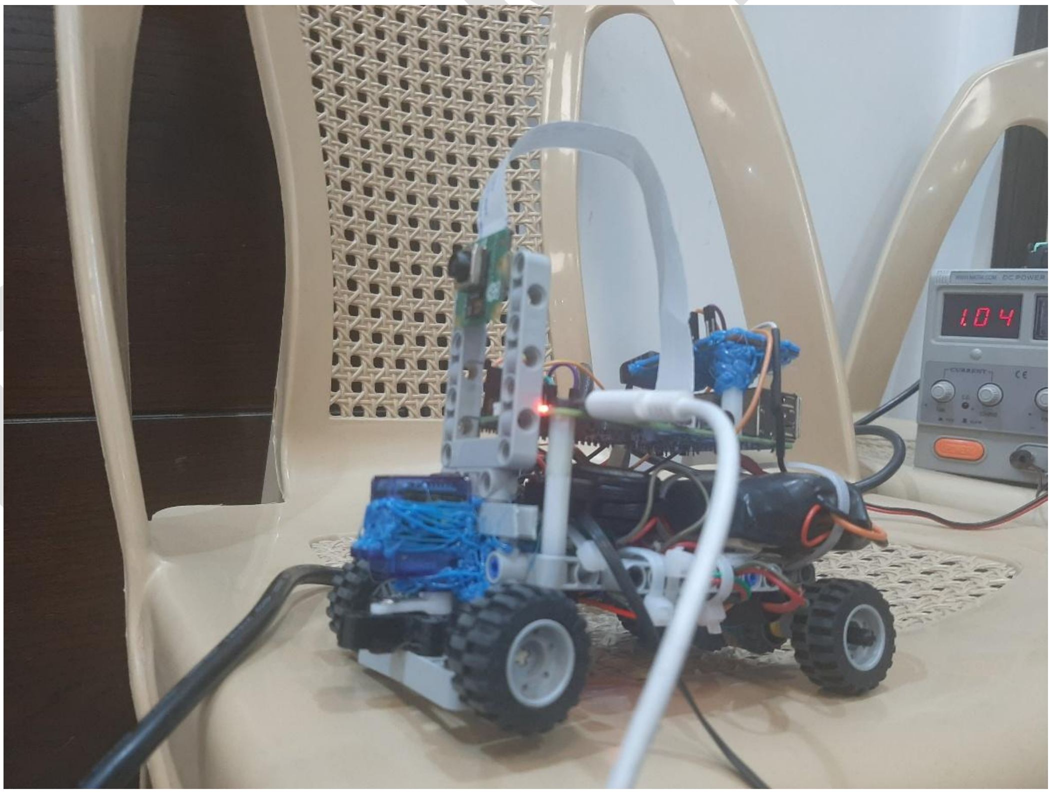
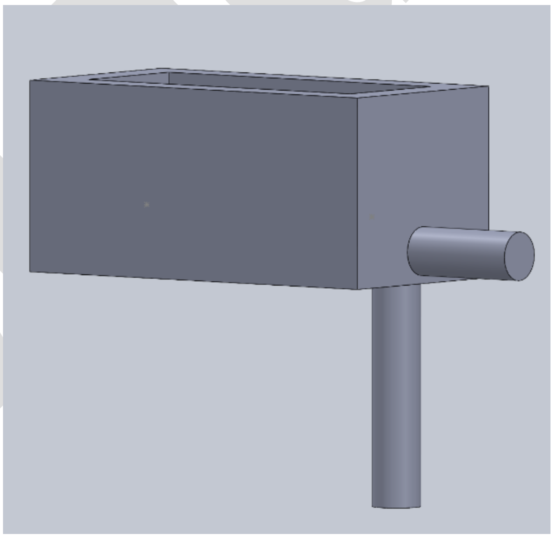


Figure 49: The LEGO Design

We used LEGO components as the main body of the robot, as demonstrated in Figure 49. However, the photo depicts our initial prototype, in which some components were temporarily stabilized using a 3D printing pen. To improve structural integrity and ensure a more reliable assembly, we later designed custom 3D-printed parts that provided better support and helped hold everything together more effectively.



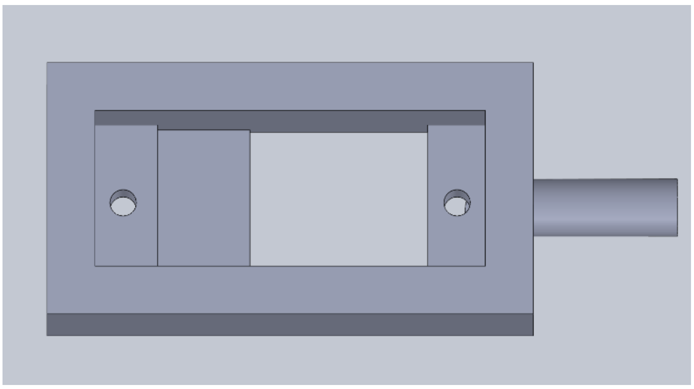


Figure 51: Servo Holder 3D Design

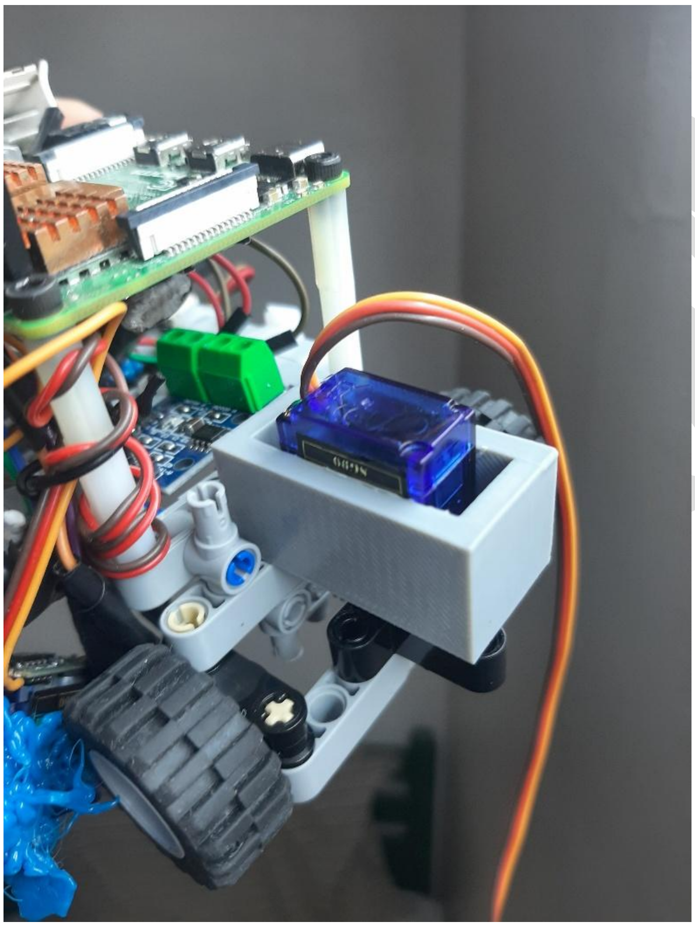


Figure 50: Servo Holder Implemented

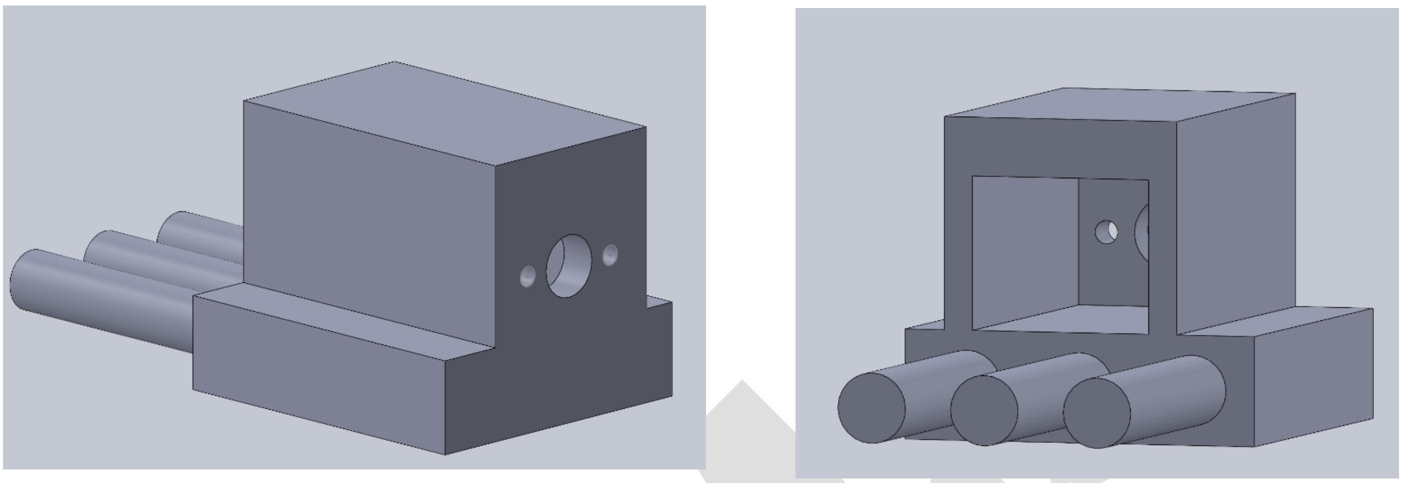


Figure 53: DC Motor Holder

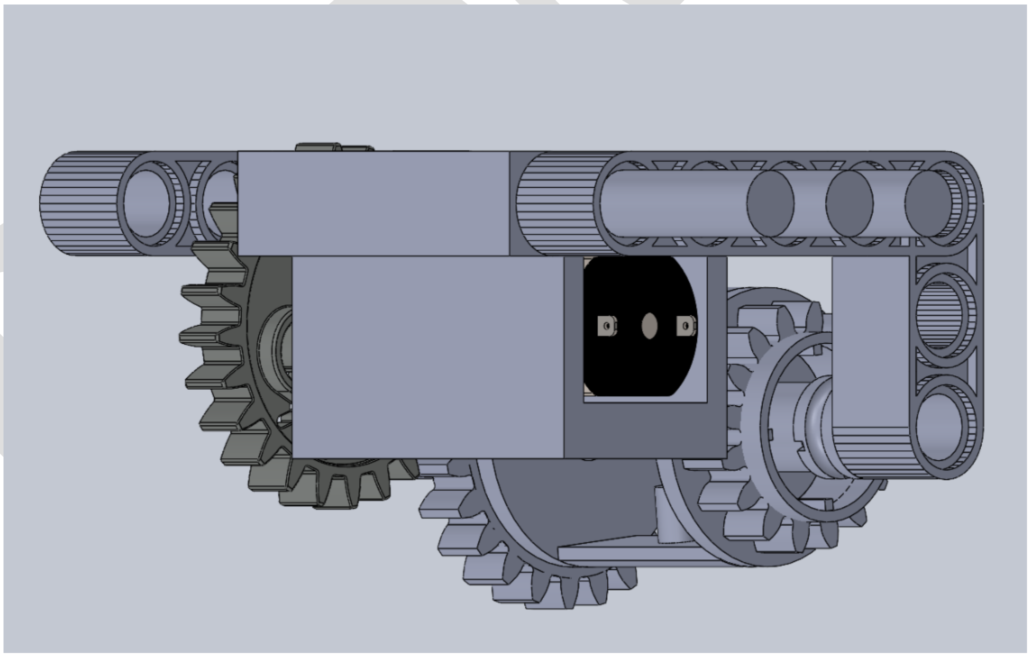


Figure 52: DC Motor Holder Implemented

Challenges with the Initial Design

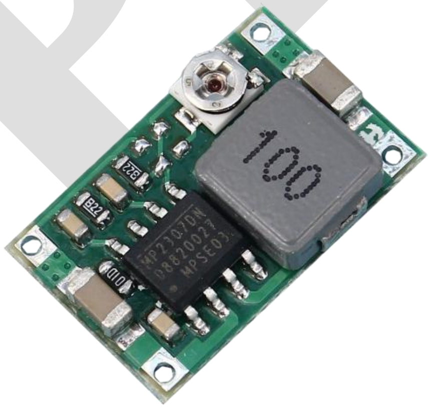
Our early prototype faced several structural limitations that hindered its overall stability and functionality:

- Many components were held together using temporary fixtures, without a reliable long-term solution.
- The available LEGO parts were slightly flexible, which conflicted with the rigidity required by our core design.
- The layout did not allow for a secure placement of the battery, making power management difficult.

In response to these issues, we explored more durable alternatives. This led to the development of our current design, which incorporates custom 3D-printed components to enhance stability and ensure a more cohesive build.

Power Supply Limitations

Moreover, at the time this design was implemented, our custom power supply circuit had not yet been developed. As a result, the structure was not equipped to accommodate or support the electrical components effectively. This limitation further highlighted the need for a more integrated and adaptable design approach.



5.2 Present:

Major Design Overhaul

Noticeably, we implemented significant changes to improve upon the previous design. A completely new body for the robot was constructed, as detailed in our documentation, offering enhanced structural integrity and better component integration. Additionally, we developed a custom circuit specifically for power management, allowing for more efficient energy distribution and improved system reliability

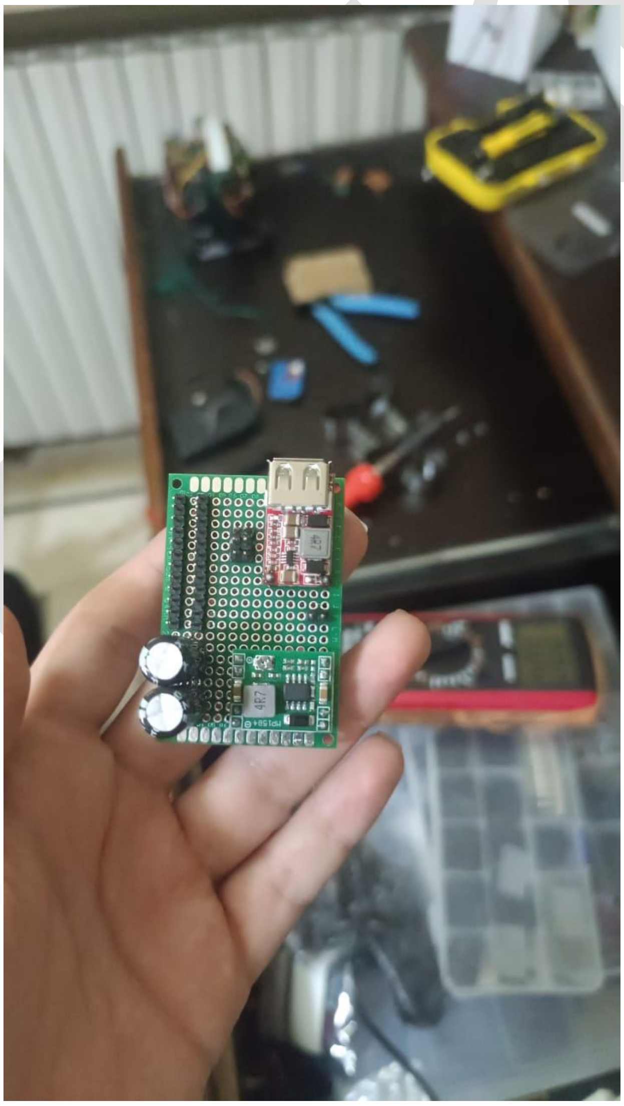


Figure 54: Our Custom Power Circuit

5.3 Future:

Future Possible Developments

We have extensively considered the future development of this design from multiple perspectives—including the body structure, the underlying algorithm, and other critical components—which will be thoroughly presented in the following section.

The Robot's Design:

We plan to modify several components of the robot in order to reduce its overall size. These adjustments include:

- **The Batteries:** One potential solution to reduce the robot's overall size is to use micro batteries, which offer compact energy storage without significantly compromising performance. If micro batteries are not feasible due to power requirements or availability, we propose repositioning the existing battery setup. Specifically, placing the batteries in a vertical orientation could help minimize the robot's width, thereby contributing to a more streamlined design.
- **The Motor Driver:** To further reduce the robot's size, we considered using more compact motor drivers. The DRV888 motor driver was a promising candidate due to its small footprint and performance capabilities. However, excessive heat generation during operation made it unsuitable for our current design. If thermal management issues can be resolved—either through improved heat dissipation techniques or alternative cooling solutions—we strongly recommend adopting the DRV888, as it would significantly contribute to downsizing the robot.

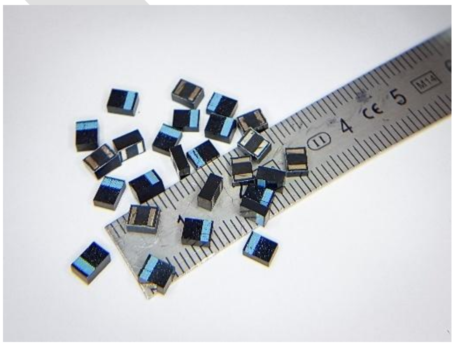


Figure 55: Micro Batteries

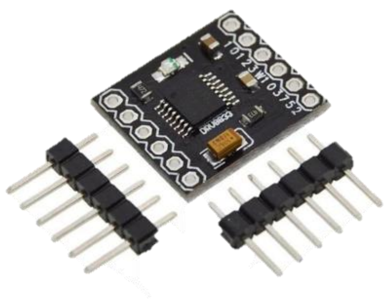


Figure 56: DRV888 Motor Driver

- **Replacing the Raspberry Pi 4 Model B:** To achieve a more compact design, we consider replacing the Raspberry Pi 4 Model B with either the Raspberry Pi Pico 2W or the Raspberry Pi Zero 2W. Among these options, the Raspberry Pi Zero 2W is particularly recommended due to its relatively fast processing capabilities and user-friendly architecture. This substitution would significantly contribute to downsizing the robot while maintaining sufficient computational performance for most tasks.

- **The Custom Perfboard:** The custom perfboard will remain an integral part of the robot's design, as it plays a crucial role in simplifying power distribution across components. However, to optimize spatial efficiency, we propose repositioning the perfboard vertically. This adjustment would help reduce the overall width of the robot without compromising functionality or accessibility.

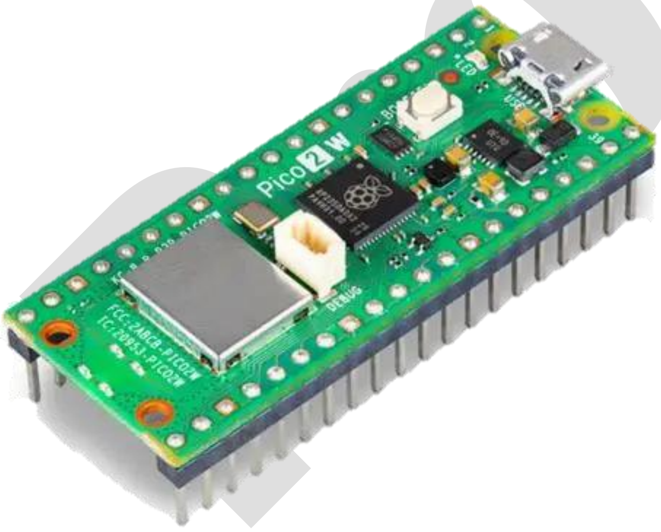


Figure 57: Rpi Pico 2W

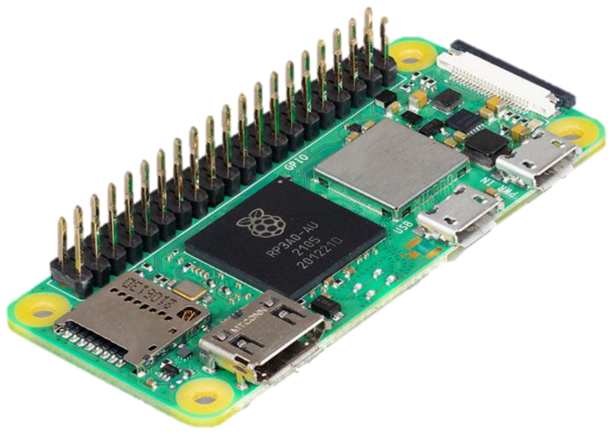
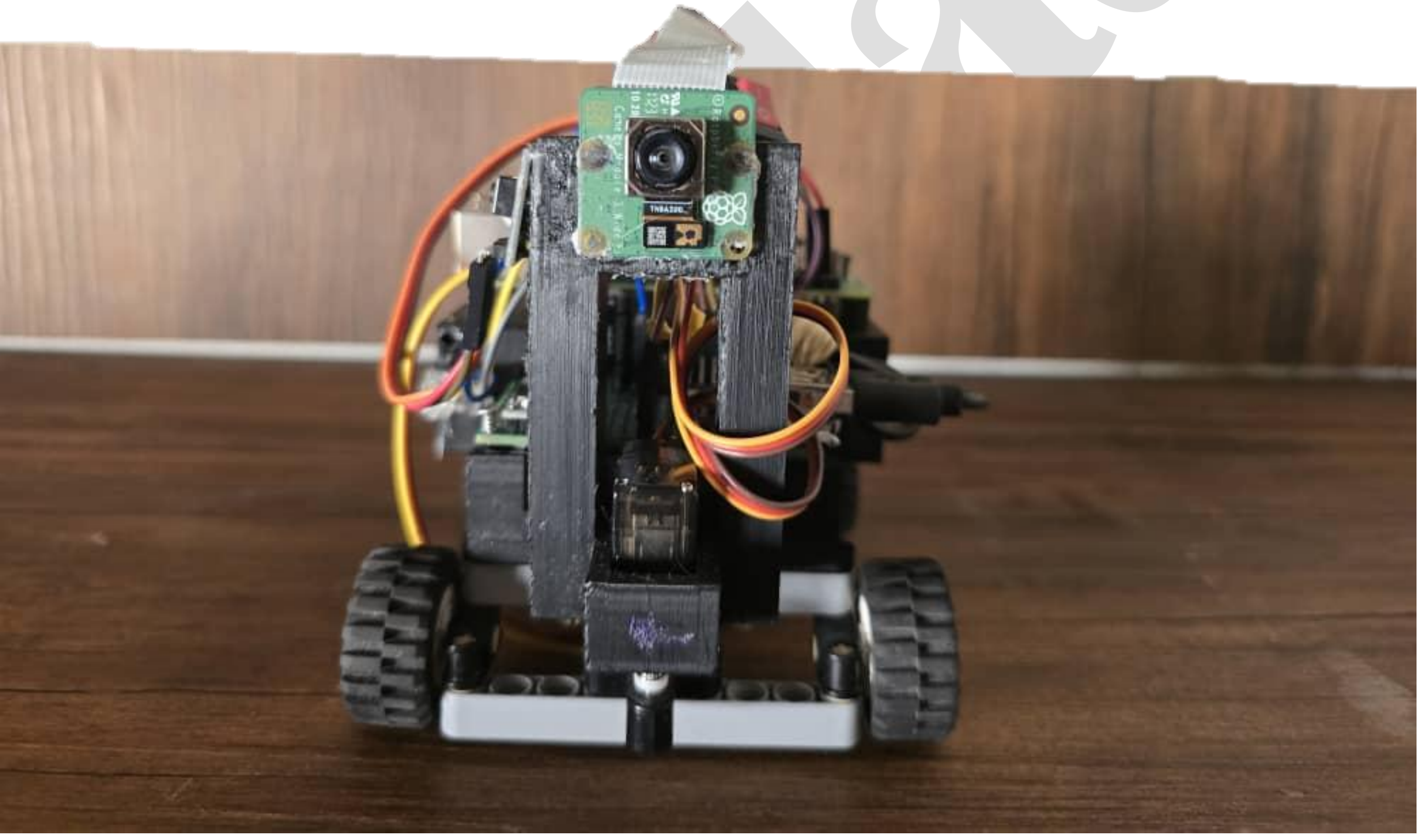


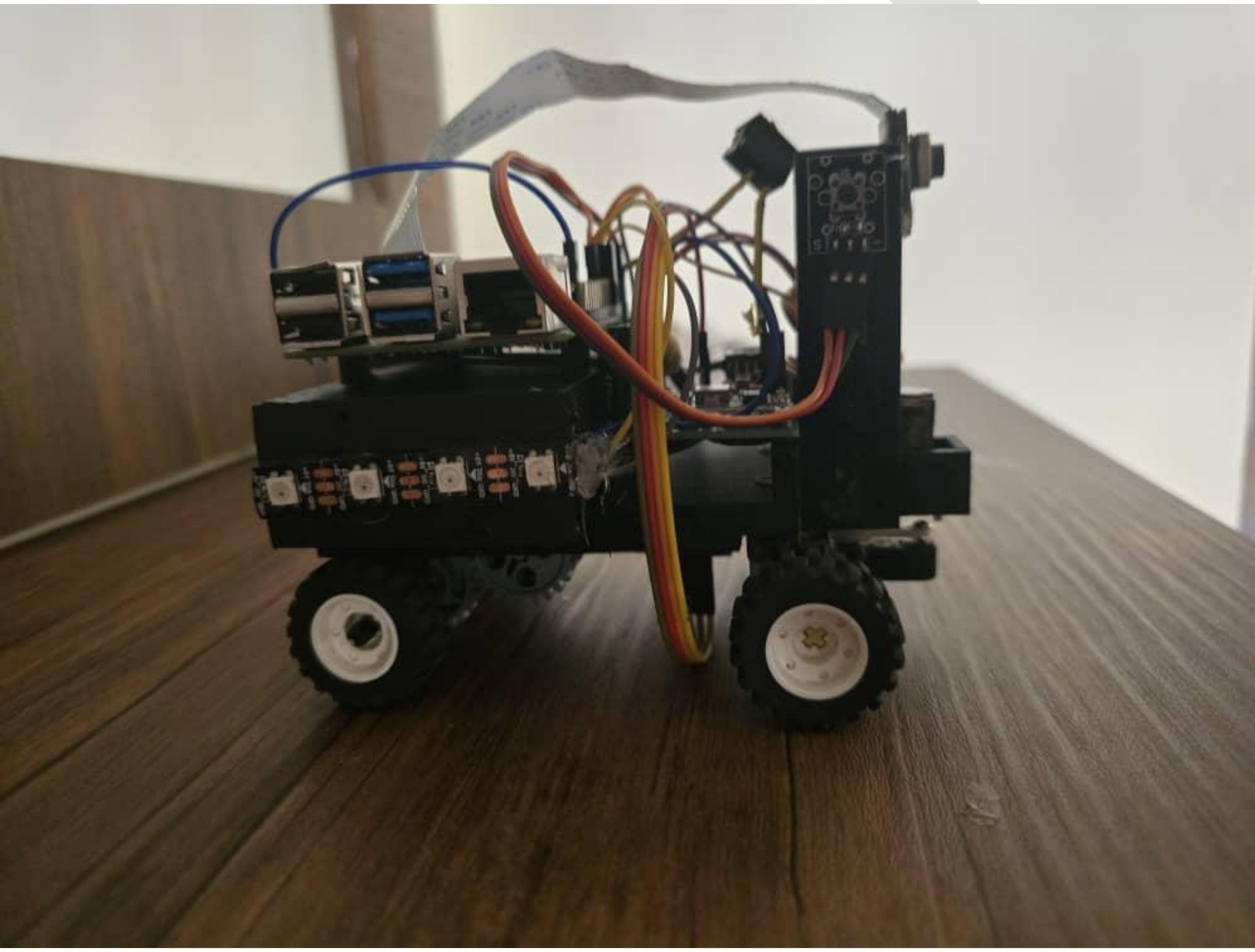
Figure 58: Rpi Zero 2w

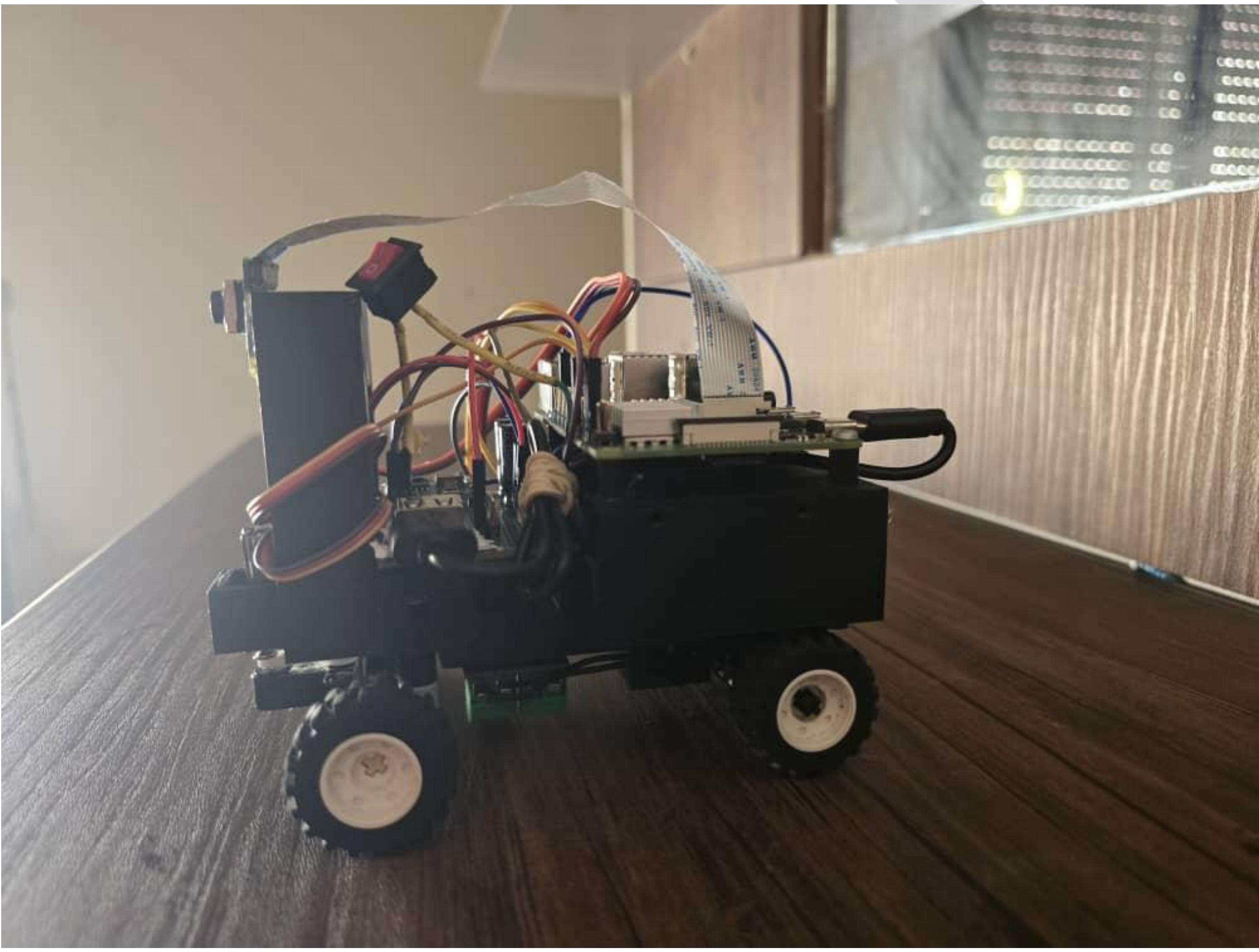
The Algorithm:

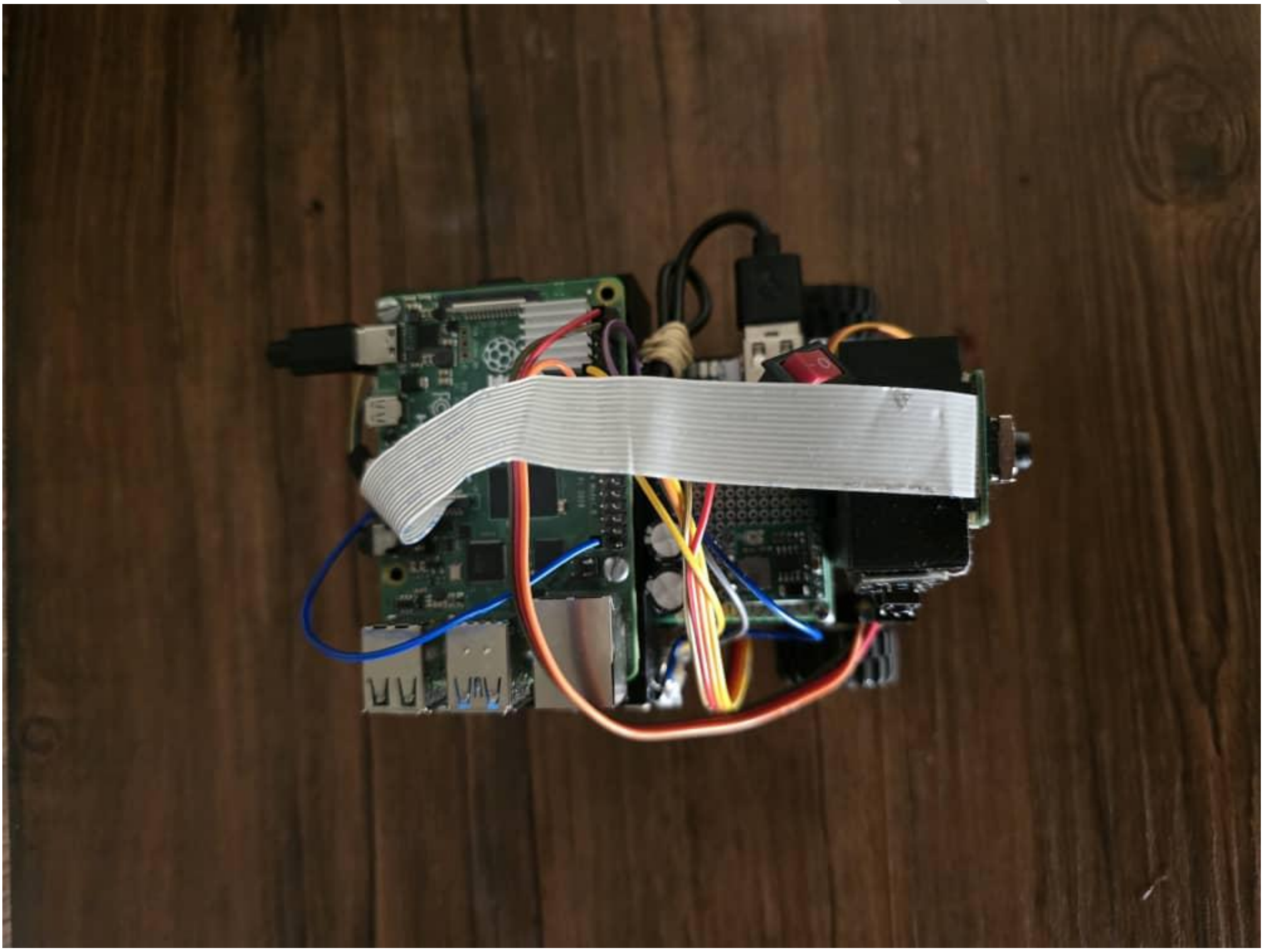
We aim to achieve faster processing and a higher frame rate, alongside the integration of a more powerful motor, in order to enhance both steering responsiveness and overall speed. Although we are still exploring additional strategies for improvement, these upgrades represent our primary focus at this stage.

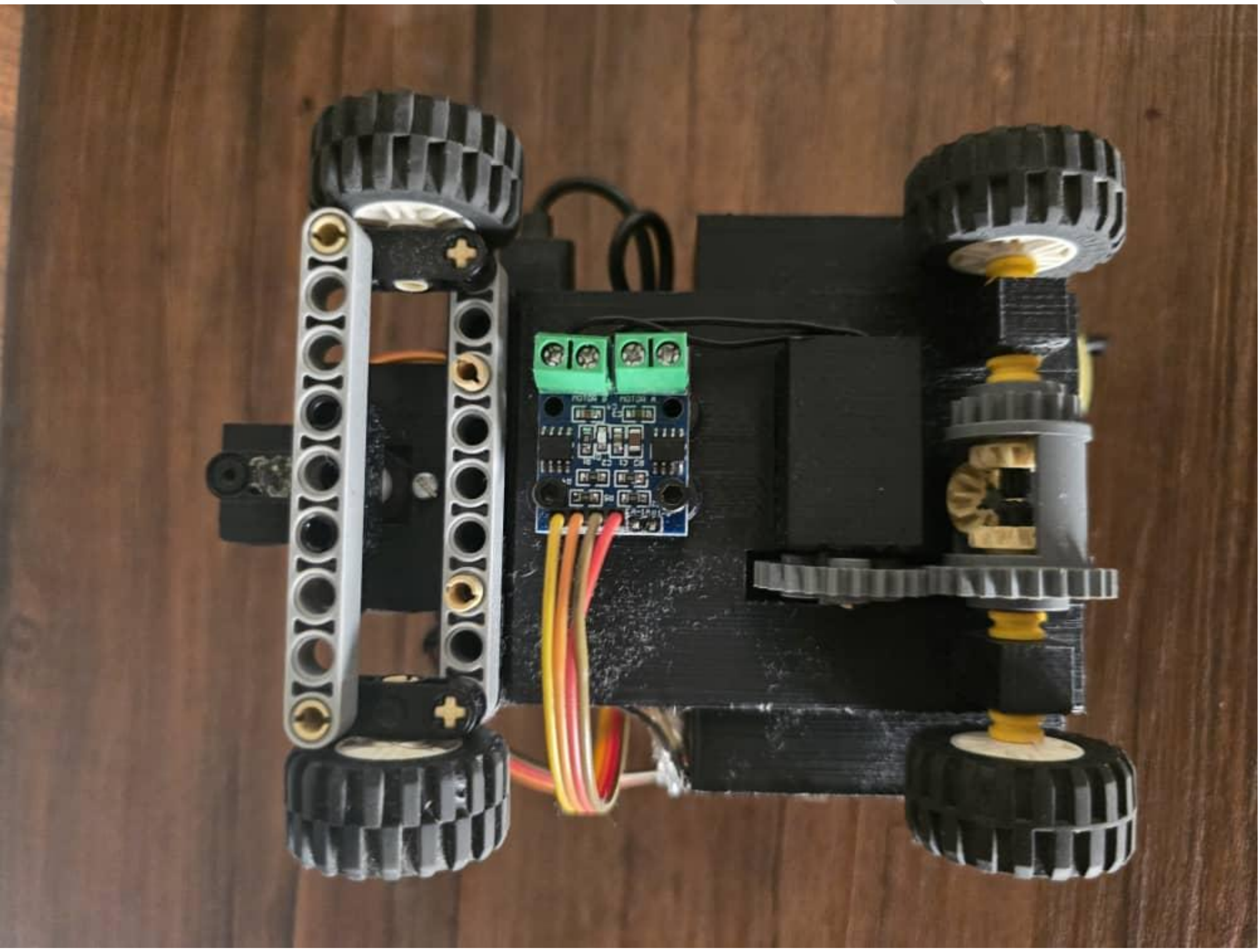
6. The Vehicle's Photos:











7. Links and Credits:

Component Links:

Raspberry Pi 4 Model B:	Amazon Link
Raspberry Pi Camera v3 Wide:	Amazon Link
MG90s Mini Servo Motor:	Amazon Link
GA12-N20 Geared Mini DC Motor 150Rpm / 12V	Amazon Link
L9110 Motor Driver:	Amazon Link
LiPo Batteries:	Amazon Link
20A BMS:	Amazon Link
Mini 360 DC-DC Buck Converter:	Amazon Link
DC-DC Buck Converter Module with USB Type-A Output:	Amazon Link
Perfboard:	Amazon Link
470uf / 25V Capacitor:	Amazon Link
LEGO Kits	Amazon Link
STL Files;	Link

Main Links:

GitHub Link QR + Link / Open Challenge / Obstacle Challenge

Open Challenge Code **Obstacle Challenge Code**

As a team, we thank the SCS for hosting and funding us. We also thank our national organizers (The DCA) for their hospitality of the WRO.

We would like to thank as well some friends who helped us along the way:

- Amr Al Ghafir
- Omar Taghlibie
- Mohammad Ward