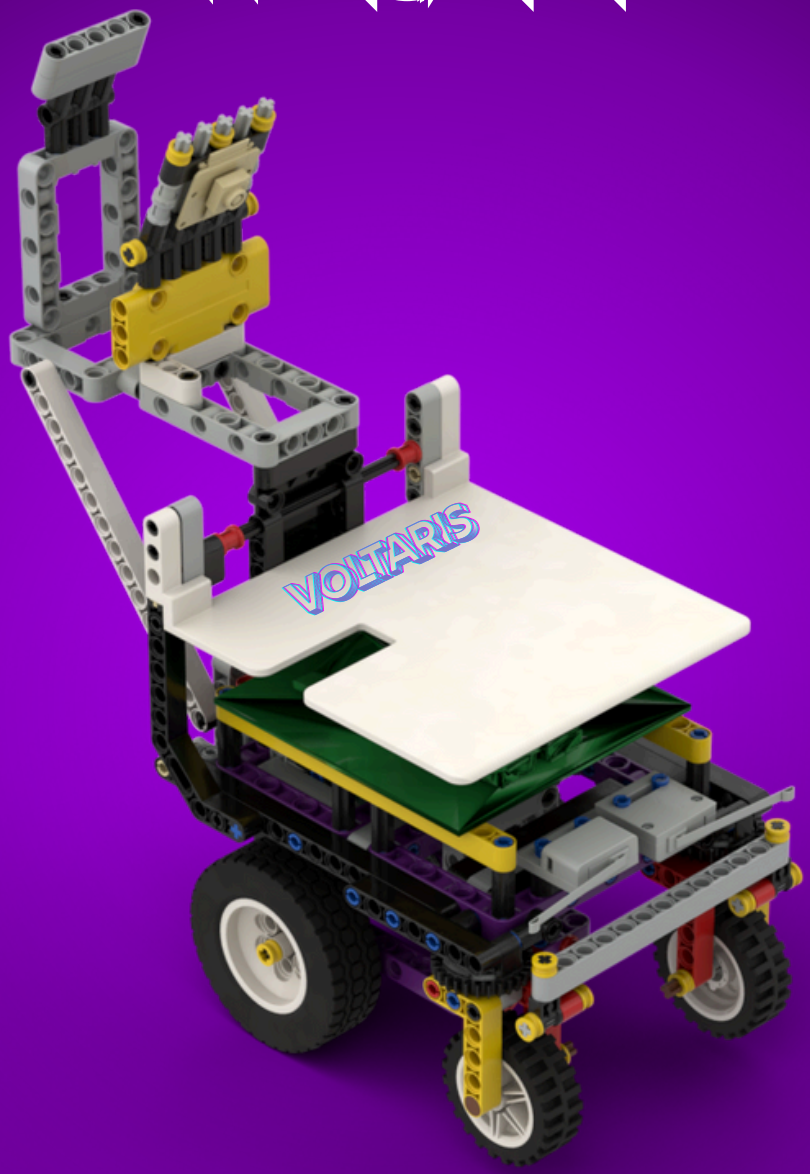
CIRCUITRON #1642



ENGINEERING NOTEBOOK 2025

WORLD ROBOT OLYMPIAD™ WRO INDIA

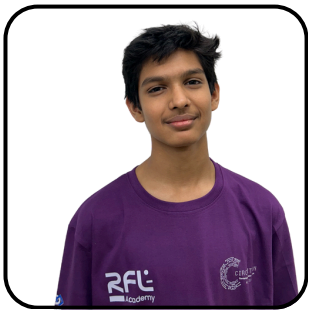
THE FUTURE OF ROBOTS

TABLE OF CONTENT

About Our Team

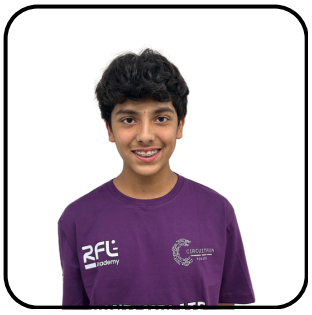
We are Team Circuitron — Contemplate, Create, Conquer — a focused group of innovators driven by a shared passion for robotics. Guided by the nature of the challenge, we strategically divided the industry, each member taking charge of a core component to engineer a unified solution. With the amazing journey we built extensive skills. We developed functional coding to drive and control our robot , Electronics in Power management and refined design through testing.

Meet The Team : Circuitron



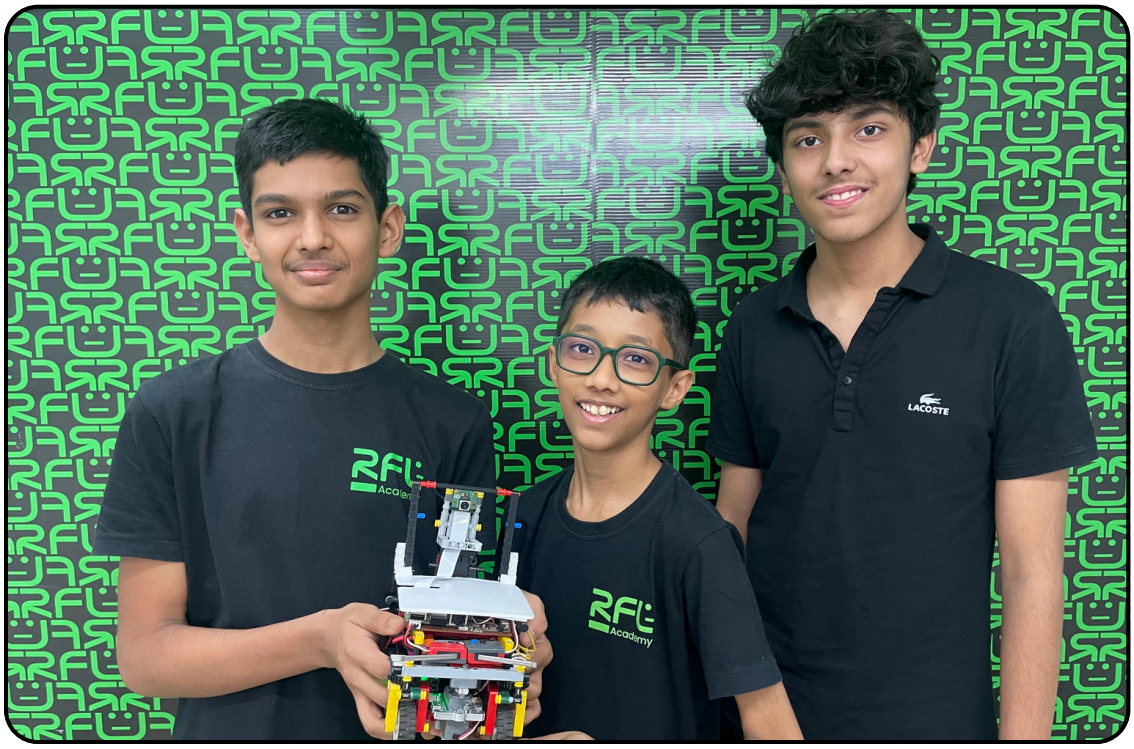
Pranay Agarwal
Lead Programmer
Grade 9

Paritra Gada
Constructor
Grade 9



Aditya Parikh
CAD specialist
Grade 9

Team Photo



Acknowledgement

Sunil Sir
Construction & Programming Mentor

Prashant Sir
CAD & Control System Mentor

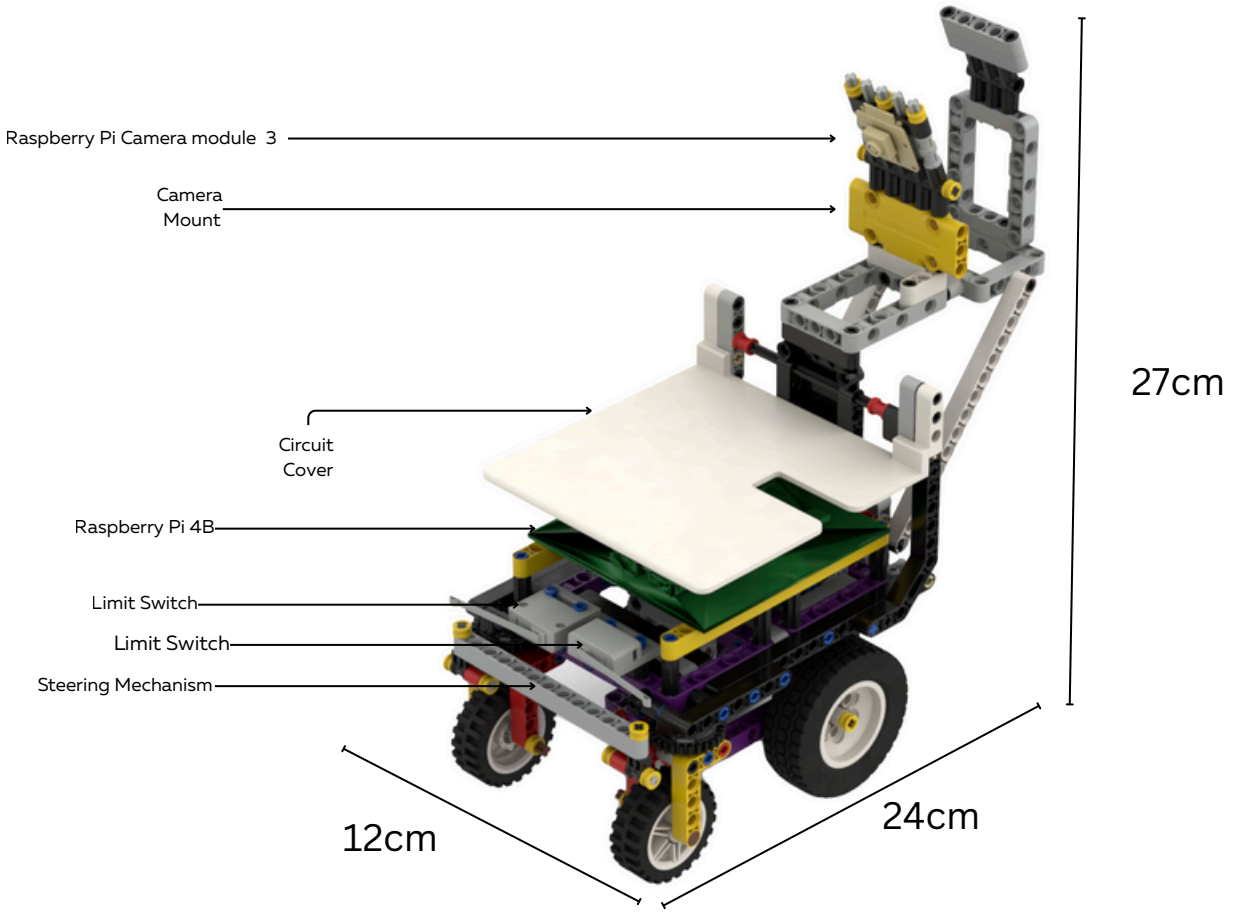
We take this opportunity to sincerely thank our mentor, Mr. Sunil Solanki for his invaluable technical guidance and consistent support throughout the course of this competition. His expertise, motivation, and encouragement played a vital role in helping us stay focused and perform to the best of our abilities. We are truly grateful for his mentorship and dedication.

We would also like to thank Mr. Prashant Kumar Chaudhari for his immense support in Electronics and the planning of mechanical design. He has guided us along the journey and has helped develop our CAD and designing skills. We are truly thankful for his immense support and guidance.

We understand the time it takes to share detailed feedback and appreciate your effort in doing so. Your continued support is invaluable, and we look forward to learning more from you.

Meet Voltaris

Our robot operates on the principle of sensor-driven perception, processing, and actuation. It begins by collecting input from the Raspberry Pi camera module , which provides information about the environment, such as obstacles, walls, and navigation.
The self-driving car challenge solves both the open and obstacle challenge with innovative design and methodology.



SWOT Analysis

Strength
Our primary **strength** lies in the redundancy and reliability that is associated with it. Incorporating IR sensors, limit switches and a custom made motor, we combine speed with precision

Weakness
The main **weakness** of our robot is its complete dependence on the Raspberry PiCam 3 for detection of all the elements.

Opportunity
An **opportunity** for our bot is to utilize the Raspberry Pi Cam 3 which is responsible for all the driving functions of our robot

Threats
One major **threat** we may face is rapid battery depletion due to the multiple sensors attached onto the robot.

Mobility Management

Our robot's mobility was designed with a focus on stability, torque balance, and precise steering.We tested multiple motor and chassis configurations before finalizing an optimized design.

DC Motor Configuration

1. N20 Brushed DC motor

The N20 DC motor is a small, lightweight brushed DC motor commonly used in miniature robots, toys, and precision devices. It is usually paired with a metal gearbox to provide usable torque while maintaining compact size. The motor is easy to drive with standard motor drivers and operates directly from low power voltage (3-12 V)

We considered the N20 motor for our vehicle because:

- It is compact and lightweight, suitable for small prototypes.
- It is cost-effective and widely available.
- With a gearbox, it may be able to deliver sufficient torque (140 g/cm) for light loads while operating at a reasonable rpm (600)

N20 Brush DC motor

2. NEO 550 brushless DC motor

The NEO 550 from REV Robotics is a lightweight, high-performance brushless DC motor designed for competitive robotics. It delivers excellent power-to-weight ratio, high efficiency, and fast response. Unlike brushed motors, it requires the SPARK MAX controller for operation, which allows precise speed and torque control. The integrated hall sensor provides reliable feedback for closed-loop control.

We considered the NEO 550 for our vehicle because:

- It provides higher rpm (11000) and torque (1.2Kg/cm) than small brushed motors .
- It is compact and lightweight, suitable for our chassis.
- Brushless design ensures longer life and efficiency.
- It integrates well with control electronics for smooth driving.

Neo 550 Brushless DC motor

3. Lego medium motor

The LEGO Medium Motor is a compact brushed DC motor with an internal rotation sensor designed for LEGO robotics systems. It provides moderate torque (815 g/cm) and speed (250 rpm), making it suitable for driving small vehicles.

We considered the LEGO Medium Motor for our vehicle because:

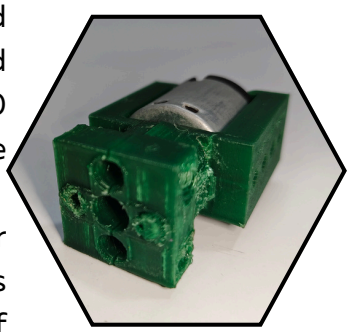
- It is easy to integrate with LEGO Technic parts and gearboxes.
- It has a built-in rotation sensor for basic feedback control.
- It is durable and beginner-friendly, suitable for prototyping.

Lego medium motor

4. D360 DC motor

The D360 DC motor is a compact brushed DC motor widely used in robotics. It is valued for its reliability, ease of control, and compatibility with custom gear systems. With an rpm of 8000 and torque of 305 g/cm, and a suitable gearbox, it would be perfect for our project.
In our project, the D360 was mounted inside a 3D-printed motor case and connected to a LEGO WeDo gearbox, which allowed us to efficiently transfer motion and achieve the required balance of speed and torque for vehicle movement.
We selected the D360 motor because:

- It is simple to integrate with motor drivers and power supplies.
- When paired with a gearbox, it provides sufficient torque for driving our vehicle.
- It is lightweight, cost-effective, and easily replaceable, making it ideal for prototyping and competition use.



D360 3D Printed Cover

Servo Motor Configuration

The 2000 Series Dual Mode Servo from REV Robotics is a versatile smart servo motor designed for robotics applications. It can operate in two modes:

1. Standard Servo Mode – controls angular position (like a traditional servo), ideal for tasks such as steering, arms, or manipulators.
2. Continuous Rotation Mode – functions like a DC gear motor with variable speed control, useful for driving wheels or other mechanisms requiring continuous motion.

Key features include:

- High speed output with precise control.
- Dual operation modes for flexible design choices.
- Durable construction with mounting compatibility for robotics frames.

2000 Series Dual Mode Servo motor

Dual Mode Functionality: Default Mode: **Precise** angular positioning (up to **300°)** via PWM

Durable Construction: Metal gears and dual ball bearings ensure **long life** and **smooth output.**

High-Speed Performance: Rotational speed up to **290 RPM** in certain models.

Wide Voltage Range: Operates between **4.8V–7.4V**; performance varies by input voltage.

Implementation

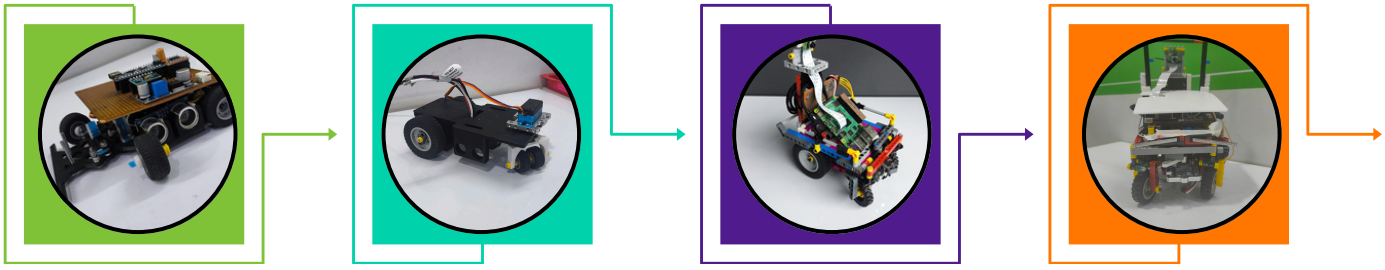
Design Efficiency

1. The target was achieved through a structured process of continuous design iterations.
2. At each stage, we carefully tested the prototype, identified performance limitations, and documented the challenges.
3. The insights gained from these evaluations guided improvements in motor selection, mechanical integration, and control strategies.
4. By incorporating feedback from repeated testing, we were able to refine the design, overcome our shortcomings, and progressively move closer to an optimized final version that met our project goals.

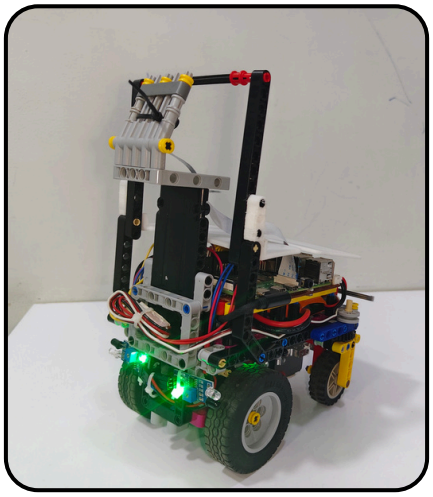
Modular robot

Efficient

Compact



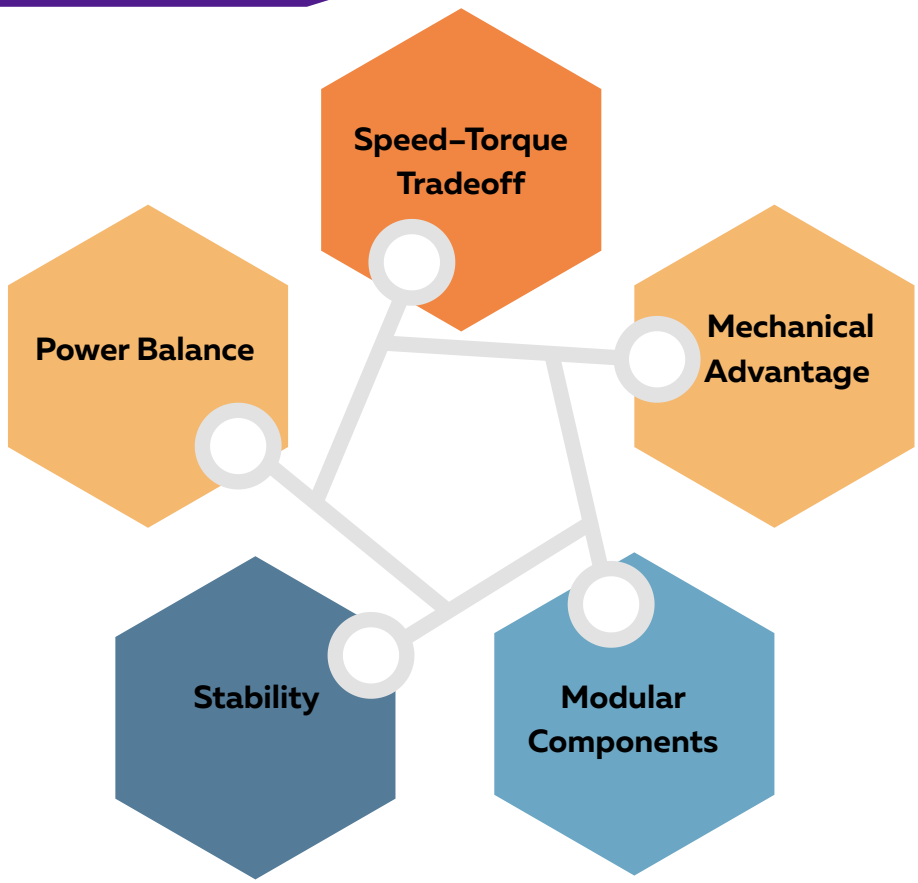
Achieving Compactness



- Compactness was achieved by optimizing internal layout, placing components like the motor vertically, and minimizing unused space within the chassis.
- We carefully selected appropriately sized motors and wheels, and routed wires efficiently.
- The use of LEGO-based prototyping in early stages also helped experiment with space-saving designs.

Voltaris Chassis Design

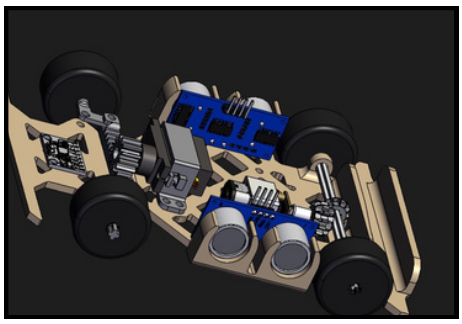
Engineering Principle

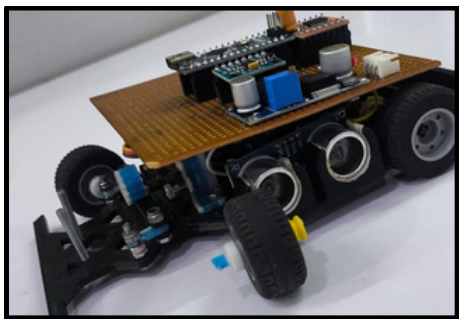


- **Speed–Torque Tradeoff**: With a motor torque of 305 g·cm and top speed of 147 cm/s, the robot balances quick movement with enough torque for maneuvering. The small frame (118×200 mm) ensures torque remains effective without excessive load.
- **Mechanical Advantage**: Compact Lego-based gearing and partial 3D-printed components optimize motor output. This setup gives the robot precise movement control while preventing power losses.
- **Power Balance:** The 7.4 V, 1300 mAh LiPo battery sustains the Raspberry Pi 4, motors, and PiCam3 along with other components to run efficiently. Careful load distribution keeps runtime practical without overburdening any single subsystem.
- **Stability:** At 750 g with a well-distributed weight, the robot maintains low center of gravity. This prevents tipping during fast turns or uneven terrain navigation.
- **Modular Components:** LEGO-based construction with 3D-printed parts allows quick redesigns and four iterative improvements.

Voltaris 1.0

- Paritra suggested to use a simple open chassis with a flat base, powered by a central DC motor and controlled through a rack-and-pinion steering system.
- The chassis provided sufficient space for mounting components while maintaining a lightweight structure.
- The electronics in our robot were kept minimal and easy to reconfigure, which allowed quick modifications during testing.





Advantages

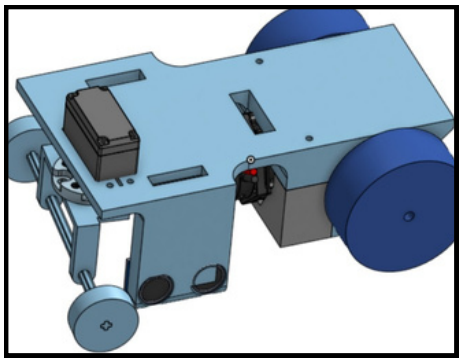
- Easy to build and modify
- Simple steering and electronics layout
- Flexible for development and testing

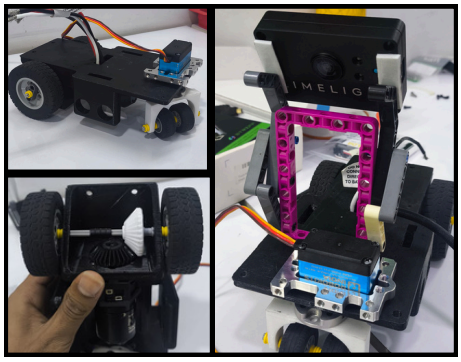
Disadvantages

- Medium structural strength
- Basic functionality
- Weight Distribution was challenging
- Turning radius was too much

Voltaris 2.0

- Pranay suggested using a Neo Motor 550 for higher speed and torque as our drive motor. A servo-based steering system replaced the rack-and-pinion, offering smoother and more precise control. A Raspberry Pi Camera module 3 was integrated for vision-based navigation, while a gearbox optimized motor output to balance speed and torque. Bevel gears were added for efficient power transfer in a compact drivetrain. This prototype marked a shift toward higher performance, better control, and computer vision integration.





Advantages

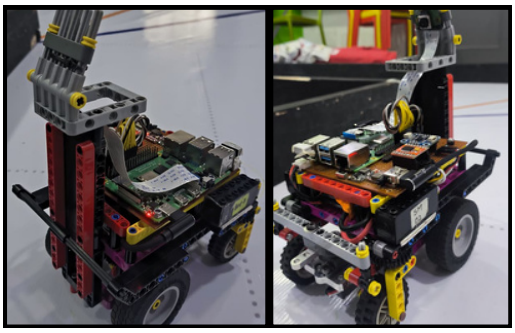
- Stronger motor
- Precise steering
- Integrated Camera
- Efficient power transmission

Disadvantages

- Heavier and bulkier
- Reduced manueverability
- More complex assembly
- Difficult to make quick decisions

Voltaris 3.0

- Prototype 3 focused on compactness, using a LEGO-based modular chassis. This was ideated by one of our teammates, Paritra.
- We integrated the Raspberry Pi Camera Wide Module 3 for improved vision.
- To save space, the motors were arranged in a vertical layout as suggested by Aditya, and a linkage steering mechanism was adopted, keeping the design compact and efficient.



Advantages

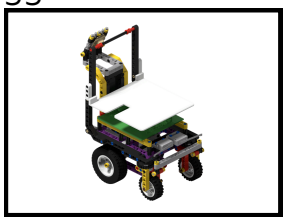
- Compact and Space efficient layout
- Modular and flexible Lego based chassis
- Clean component integration
- Simplified , maintained design
- Balanced size and functionality

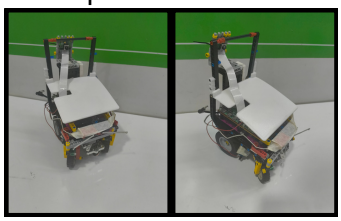
Disadvantages

- Motor was not fast enough
- Stall Torque of the motor was not high enough
- Circuit position was not stable enough

Voltaris 4.0

- Our final prototype combined the compactness of Prototype 3 and the speed of Prototype 2 to form a consistent, fast and easily maneuverable robot
- Using the D360 DC Motor along with a 22:1 gearbox, we could leverage both speed and torque
- Our teammate Pranay suggested to use Limit Switches to enable redundancy if the robot were to get stuck
- IR Sensors were added on the side for a consistent parallel parking
- Aditya suggested the idea of using a mix of 3D printed parts as well as Lego Parts





Advantages

- Faster and stronger motor
- Precise steering
- Limit Switches allow the robot to understand if it is stuck
- IR Sensors allow a more consistent and reliable parking
- Cover allows protection of circuit components

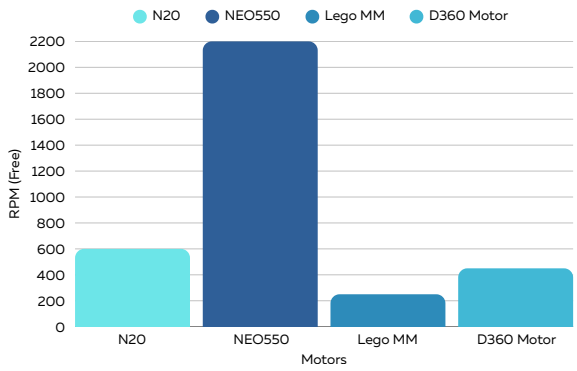
Disadvantages

- Without encoders, there may be some error that we cannot account for
- Battery depletes within just 20-25 minutes of running

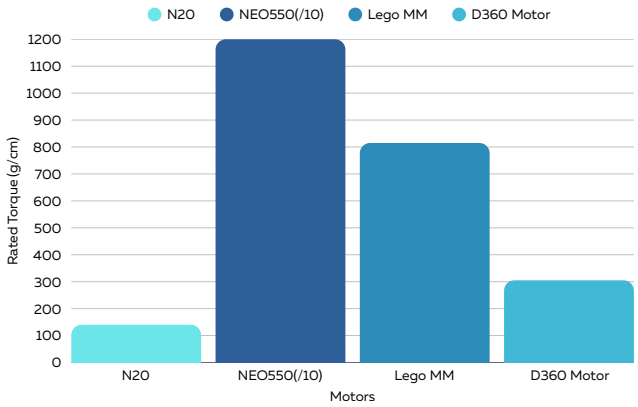
Motor Comparison With Gearbox (Speed)

Motor Comparison (Torque)

Below is the graph that represents the comparison of DC motor configuration with respect to speed and torque. Through multiple iterations, we identified that the D360 motor best meets our required balance of speed and torque for optimal performance



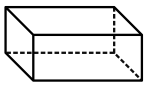
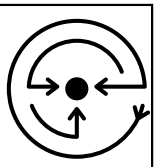
Motors Vs Speed



Motors Vs Rated Torque

Robot Comparison (Theoretical)

We made a mathematical comparison between the different iterations of our robot. Throughout the iterations, we can see a clear advantageous thread progressing onwards, showcasing the clear increasing advantage that we have gained as well progressed along our journey.

VERSION				
1	400 g	95cm/s	19.5 cm	2.875 m/s²
2	1000 g	718.3 cm/s	16.2 cm	6.63 m/s²
3	550 g	81 cm/s	12.2 cm	2.04 m/s²
4	750 g	147 cm/s	12.2 cm	4.45 m/s²

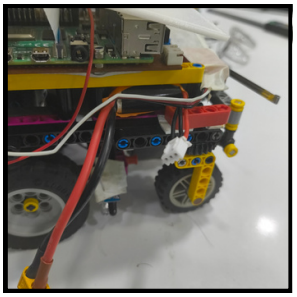
Power & Sense management

The performance of our robot mainly depends on a stable and efficient power system and voltage regulator, such as 7.4 Li-ion battery pack ,Buck converter (3 ampere), USB Output Step Down Power Charger Buck Converter

1. 7.4V Li-Ion Battery Pack

The robot is powered by a 7.4V Li-ion battery pack, built from two 18650 lithium-ion cells connected in series.
Each cell provides a nominal voltage of 3.7V, giving a combined output of 7.4V and a fully charged voltage of around 8.4V.
Li-ion batteries were chosen because they offer:

- High Energy Density – They store more energy per unit weight, allowing the robot to run longer without increasing bulk.
- Voltage Stability – They maintain a steady voltage under load, which is critical for preventing sensor resets or motor stalls.
- Rechargeability – The pack can be recharged multiple times, making it cost-effective and sustainable for repeated testing and use.



2. Buck converter (3 ampere)

- A buck converter is a DC-to-DC converter that steps down a higher input voltage to a stable, lower output voltage.
- As suggested by Aditya, it plays a key role in power management since different components operate at different voltages.
- The Li-ion battery pack provides 7.4V as the main supply, while the drive motor and servo motor requires a constant 5V. The buck converter reduces 7.4V to 5V, ensuring safe, efficient, and reliable operation of these servo motor.

Reason for selection of Buck Converter :

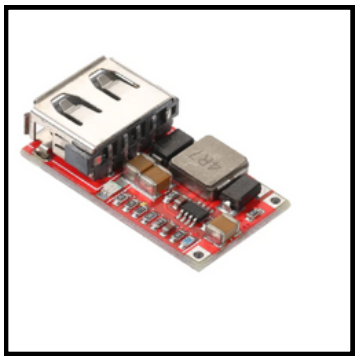
- Efficient Power Conversion – Unlike linear regulators, the buck converter minimizes energy loss by converting excess voltage into usable current instead of heat, improving overall efficiency.
- Component Protection – Sensitive electronics such as the servo require a stable 5V supply. The buck converter ensures consistent regulation, preventing voltage fluctuations that could damage these components or cause malfunctions.

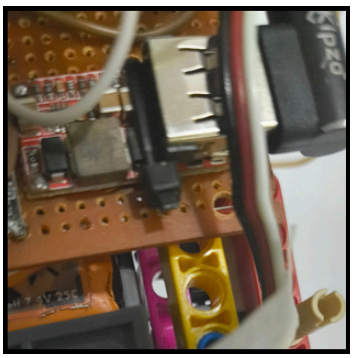
Power management

3. USB Output Step Down Power Charger Buck Converter

- Paritra further advised to use USB buck converter reduces the main battery voltage (7.4V from the Li-ion pack) down to a regulated 5V USB output.
- This 5V USB output is directly compatible with the Raspberry Pi's power input and other 5V devices.

Reason for selection :
This mini DC-DC step-down voltage converter module supports 6-18V voltage input and 5V 3A output with input polarity reverse protection and output overvoltage protection.





Controller

4. Raspberry Pi 4B Computer

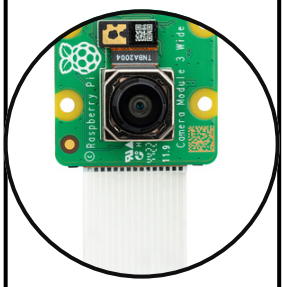
- The Raspberry Pi 4 Model B is a compact and versatile single-board computer widely used in robotics and embedded systems. It is powered by a 1.5GHz quad-core ARM Cortex-A72 processor with up to 8GB of RAM, providing desktop-class performance in a small form factor.
- It supports dual-band Wi-Fi, Bluetooth 5.0, and Gigabit Ethernet for high-speed communication, along with multiple USB ports and dual micro-HDMI outputs for 4K display support. The 40-pin GPIO header makes it highly adaptable for direct sensor, motor driver, and hardware interfacing.

Reason for selection:

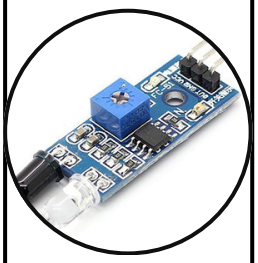
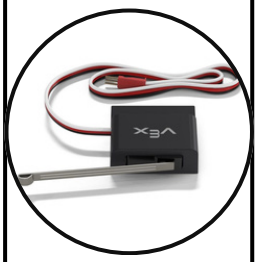
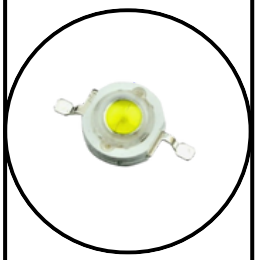
- After researching, Pranay advised us to use the Raspberry Pi 4 B for its high processing power , real time image handling capability, and connectivity enabling it to act as the central controller for coding sensor integration and Power management in our robot .

Sense management

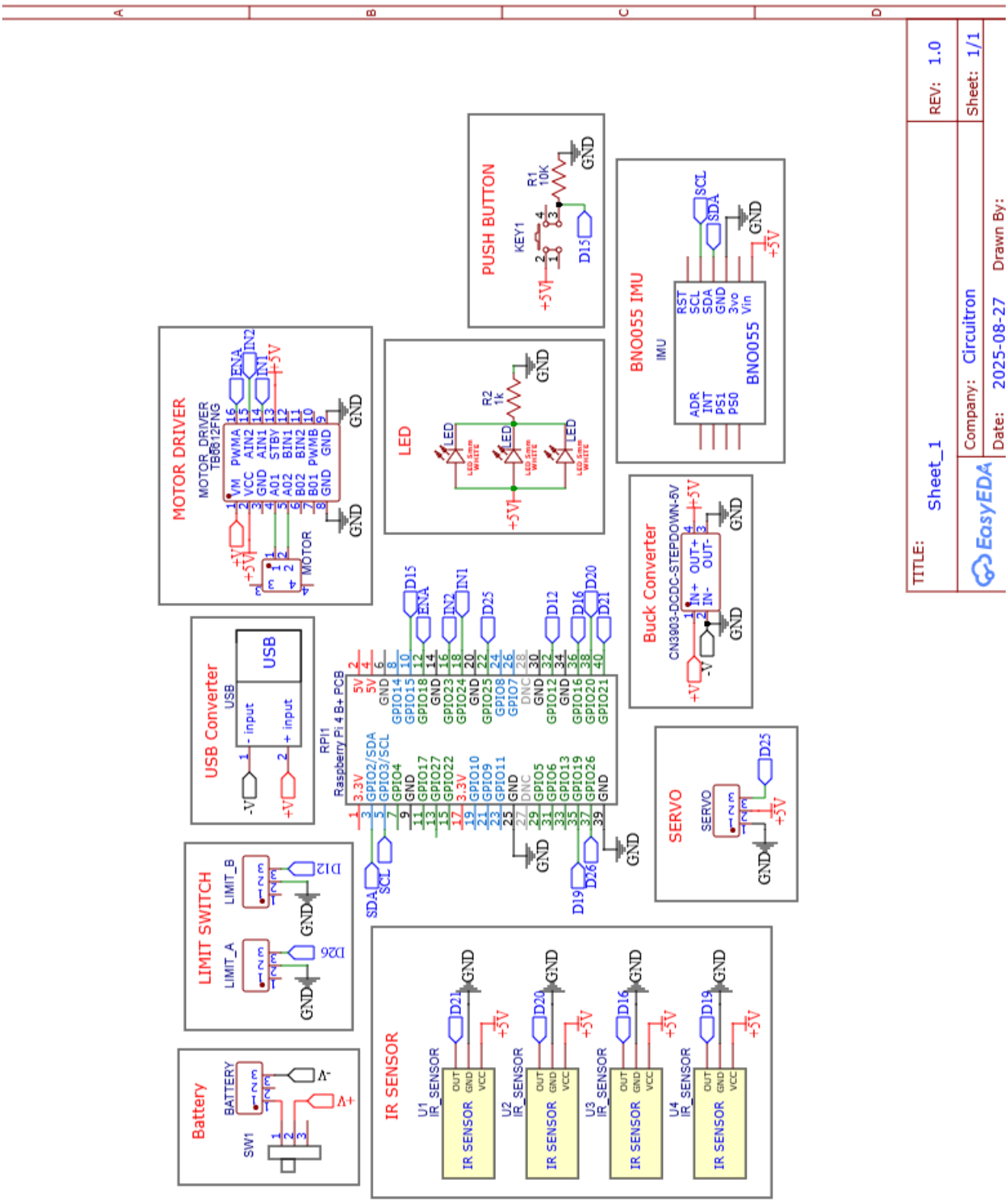
Our robot relies on a carefully chosen set of sensors that balance accuracy, efficiency, and power consumption. Each sensor was selected based on its ability to help the robot complete the challenge while keeping overall energy use low.

COMPONENT	REASON FOR SELECTING	USAGE IN ROBOT	POWER CONSUMPTION
RaspberryPi Camera Module 3 	• Capable of detecting multiple features (traffic signs, lane walls, obstacles) rapidly • Compatible with RaspberryPi 4 and can be operated via OpenCV and Picamera2 libraries • Provides a wide field of view for real-time navigation.	• Detects red and green pillars to decide left/right lane switching. • Identifies black walls for wall-following and centering. • Allows for round count by detecting the orange/ blue lines accurately	~250–300 mW (via Raspberry Pi USB)
BNO055 IMU 	• Provides orientation and stability feedback. • Helps detect sudden drifts or turns not visible through encoders alone • Communicates over 12C	• Used during parallel parking to ensure accuracy and reliability. • Eliminates the need for time based turning • Allows us to get out of the parking zone when starting our journey	~10–15 mW (via I^2C).

Sense management

COMPONENT	REASON FOR SELECTING	USAGE IN ROBOT	POWER CONSUMPTION
IR Sensor 	• Non- contact detection • Detection range can be adjusted easily according to one's requirements • Results can be seen via glowing of LEDs on the sensor	• Allows for detection of the parking zone accurately • Allows for consistent parking due to its positioning on the robot.	~25-30 mW (via GPIO pins)
Vex Limit Switch 	• Contact based detection • Long arm is triggered instantly	• Allows the robot to understand it has hit a wall/ element • The robot now incorporates a safety net, in case it ever gets stuck. • Allows for a more consistent, reliable and redundant journey	Negligible (via GPIO pins)
LED Light 	• Small and lightweight • Emits a great amount of white light	• Due to its bright light, the robot is not bothered by surrounding light conditions	~1W (via battery connection)

Schematic Diagram



Bill of Materials

Item	Cost (INR)	Link
Raspberry Pi 4b	7500	https://robu.in/product/raspberry-pi-4-model-b-with-4-gb-ram/?gad_source=1&gad_campaignid=19974686076&gbraid=0AAAADvLFWcTu9FZPNdqzwDUNqa9Azi21&gclid=CjwKCAjw2brFBhBOEiwAVJX5GP7VblF9DPel-56x3evNwvtic6E1eLzY7_RDyM-hPZEQbmFfEPm5JRoCLaEQAvD_BwE
Raspberry Pi Camera Module 3	3600	https://robu.in/product/raspberry-pi-camera-module-3-wide/?gad_source=1&gad_campaignid=19974686076&gbraid=0AAAADvLFWcTu9FZPNdqzwDUNqa9Azi21&gclid=CjwKCAjw2brFBhBOEiwAVJX5GDnaBpD_04yn6CeLhWlUEqPKC4-tGm9i9E6u7yz_vUOyDqPNrqMVkxoCElkQAvD_BwE
Vex limit switches x2	1500	https://www.vexrobotics.com/276-2174.html?srsltid=AfmBOornfvy2EzMIuj5Od1TNsaPJOtlqQhdwaAt-Cc4tFaSwrTrH18IB
IR sensors x4	88	https://robu.in/product/ir-infrared-obstacle-avoidance-sensor-module/
3D printed parts	50	
Led Light	10	https://www.amazon.in/350mA-Power-Light-Emitting-Diode/dp/B0F19BZ8YX
Lego Parts	2700	
goBILDA Super Speed Servo	3250	https://www.gobilda.com/2000-series-dual-mode-servo-25-4-super-speed/?srsltid=AfmBOorGCOd3gcuOR1l4x5KHUbaPIKePxbyfKLJCV-6AS0hanOQ28EuE
goBILDA servo mount	700	https://www.gobilda.com/1802-series-servo-frame-43mm-width-for-standard-size-servos/
Buck Convertor 5v 3A	90	https://robu.in/product/ultra-small-size-dc-dc-5v-3a-bec-power-supply-buck-step-down-module/
Buck Convertor USB	48	https://robu.in/product/dc-to-dc-6-24v-to-5v-usb-output-step-down-power-charger-buck-converter/?gad_source=1&gad_campaignid=17416544847&gbraid=0AAAADvLFWc6sX6MP_WklJqZVglNwlb7x&gclid=CjwKCAjw2brFBhBOEiwAVJX5GCMo8Uhl9GAjLUBIr1A9ekXmvEkHZ_ciunphClrgrsfBVR9zecQD0hoCuuUQAvD_BwE
IMU BNO055	1350	https://thinkrobotics.com/products/9-dof-absolute-orientation-bno055-sensor?variant=40115292405846&country=IN¤cy=INR&utm_medium=product_sync&utm_source=google&utm_content=sag_organic&utm_campaign=sag_organic&utm_source=googleads&utm_medium=cpc&gad_source=1&gad_campaignid=18239908785&gbraid=0AAAAACk3EvxjJgaRlCD3-xJw9cb8Gcrew&gclid=CjwKCAjw2brFBhBOEiwAVJX5GJcz1EdwlsERn0jNt8cm56Rf8xG1j_TJuFleHC5a4AAoLHRXX7_QsBoCdyEQAvD_BwE
32 gb micro sd card	412	https://robu.in/product/official-raspberry-pi-micro-sd-card-32gb-a2-class/?gad_source=1&gad_campaignid=19974686076&gbraid=0AAAADvLFWcTu9FZPNdqzwDUNqa9Azi21&gclid=CjwKCAjw2brFBhBOEiwAVJX5GNsnjZanxykNcnE46sY3KgWmNrqpEX8p-cZL6oaajaEx1g-D4uTYAhoCpHQQAvD_BwE
Motor Driver tb6612fng Hbridge	161	https://robu.in/product/motor-driver-tb6612fng-module-performance-ultra-small-volume-3-pi-matching-performance-ultra-l298n/
D360 base motor	200	https://www.temu.com/goods_snapshot.html?goods_id=601099518898704&title=Details&refer_page_name=bgn_verification&refer_page_id=10017_1756283670788_li07br6jp5&refer_page_sn=10017&_x_sessn_id=b0c6ec6v07
Battery	1075	https://robu.in/product/orange-1300mah-2s-25c-7-4-v-lithium-polymer-battery-pack-li-po/
Total:	22,734	

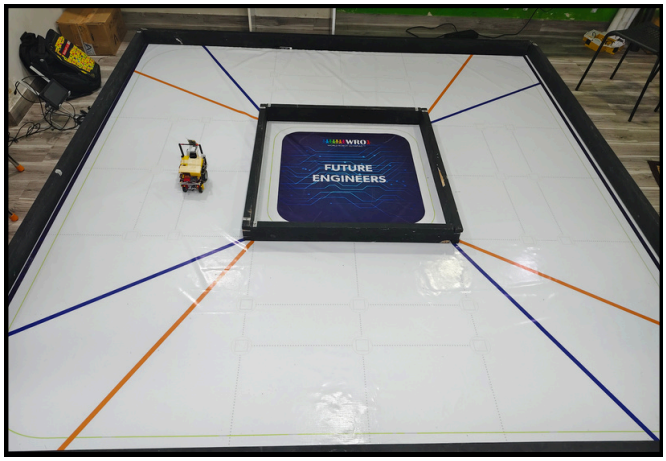
Obstacle Management

Obstacle management is the process by which a robot detects, interprets, and navigates around obstacles in its path, using sensors and control algorithms to ensure safe and efficient movement.

- Obstacle challenge
- Open challenge

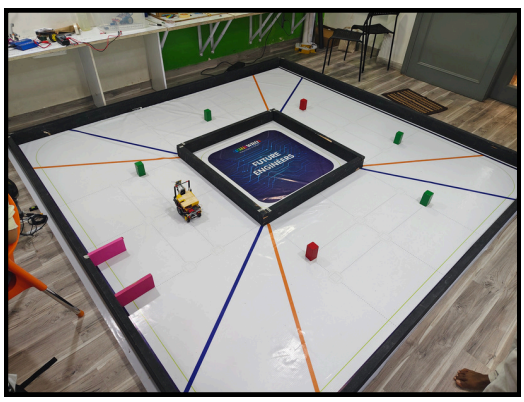
Open challenge

- The vehicle must complete three (3) laps on the track with random placements of the inside track walls and random starting positions
- The distance between the track borders will be 600 mm or 1000mm.



Obstacle challenge

- The robot must complete three laps on a track that includes randomly placed red and green traffic sign pillars serving as obstacles.
- These colored markers indicate which side of the lane the robot should keep to the right side.
- The traffic sign to keep to the right side of the lane is a red pillar.
- The traffic sign to keep to the left side of the lane is a green pillar.
- The robot should not move any of the traffic signs
- After it completes three rounds, it has to find a parking lot and has to perform parallel parking.
- The starting direction in which the car must drive on the track (clockwise or counter clockwise) will vary in different challenge rounds. The starting section of the car as well as the number and location of traffic signs are randomly defined before the round (after the check time). The following graphic shows the game field with the game objects.



Open Challenge Strategy

Initialization:

1. Starting & Line Detection

- The robot begins from the starting zone.
- Not knowing the direction, we continue to align with the wall the camera detects until a line is detected.
- Once a line is detected, on the basis of the color, the direction is decided
  - If the blue line is the first one detected, the robot sets its path to move anticlockwise around the track.
  - If the orange line is the first one detected, the robot sets its path to move clockwise.
- After the direction is set, on the basis of the error between the 2 walls, the robot tries to align itself centrally.

2. Both the orange and blue lines are used as a lap marker.

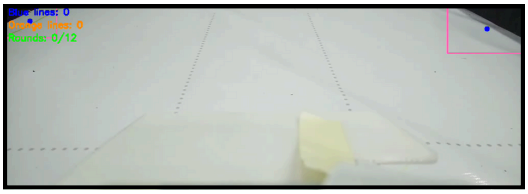
- Every time it senses any color line it adds +1 to the line counter of that color
- It is programed to detect both colors, and considers the greater count of the two.

Debounce Mechanism

- After each line detection:
  - The counter is paused for 1.5 to 2 seconds.
  - During this pause, the robot continues to wall-follow
  - This means the robot won't count the same line twice while passing over it.

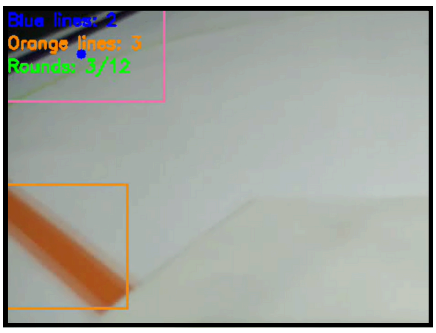
3. After completing 3 laps

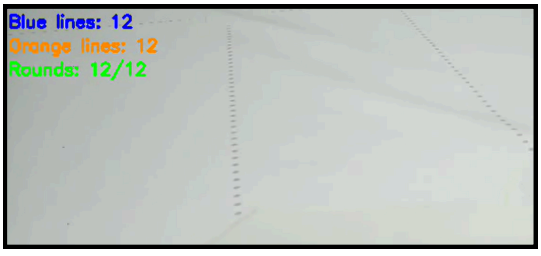
- Since the robot counts either the orange or blue lines, it requires <12 (3 laps x 4 lines)
- When the counter reaches 12:
  - The robot continues moving a little further to align correctly.
  - Then it drives forward for a fixed duration (a few seconds).
  - Finally, it stops inside the starting zone.



Initialize - Camera Image

Line Count - Camera Image





Round End - Camera Image

Pseudo Code
Open Challenge

1. IMPORTS

- Camera: picamera2
- Vision: cv2, numpy
- Timing: time, datetime
- Motor/Servo: pigpio, RPi.GPIO
- Sensors: board, adafruit_bno055

2. SETUP

- Define motor pins, servo pin, button pins
- Set motor PWM frequency
- Configure GPIO as inputs/outputs (buttons with pull-ups)
- Define motor speeds (run, slow, stop)
- Define steering limits (center, left, right)
- Define HSV ranges for key colors (black, blue, orange)
- Set camera resolution, exposure, frame rate
- Initialize camera and video recording
- Setup pigpio for motors and buttons
- Initialize IMU (BNO055) for heading tracking
- Initialize counters: line counts, rounds completed, flags (stop, over, clockwise, etc.)

3. MOTOR & SERVO CONTROL FUNCTIONS

- steer(angle): move servo within limits
- run(speed): move motors forward
- back(speed): move motors backward

4. HEADING FUNCTIONS

- Variables for heading offset, previous heading, total heading (not fully implemented)

5. VISION HELPERS

- Color masks: black, blue, orange with morphological filtering

6. MAIN FRAME PROCESSING (process_frame)

- Capture frame → convert to HSV → blur
- Create masks: black, blue, orange
- Detect:
- ・Blue/Orange lines → increment line counters with cooldown
- ・Black contours → track left/right edges of road
- Update rounds completed based on line counts
- Navigation logic:
- a) If rounds ≤ 12 → drive forward and adjust steering using black contours
- b) If rounds > 12 → stop motors
- Display debug text and save video frame

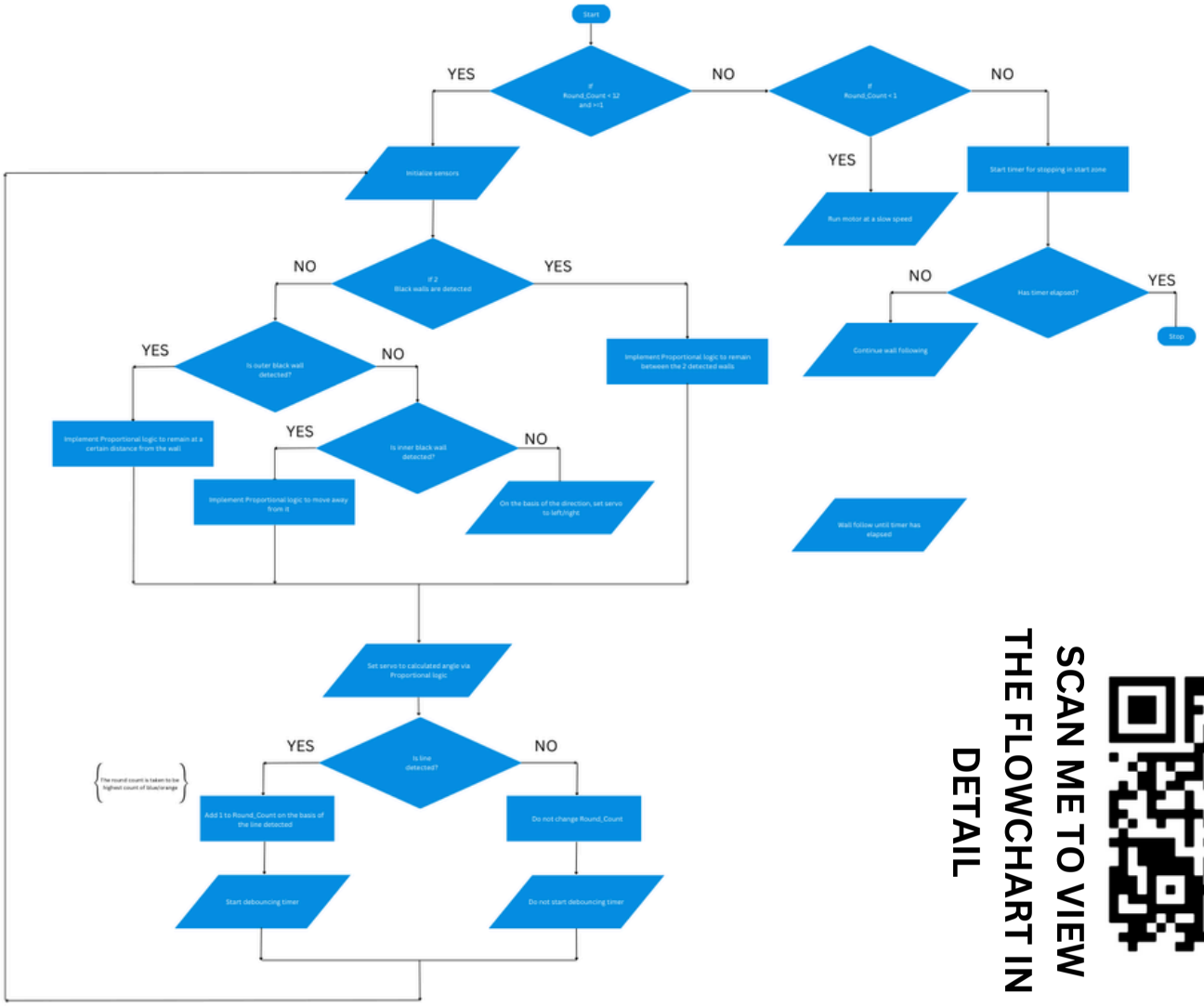
7. MAIN LOOP

- Wait for START button → set running state
- Wait for STOP button → stop robot
- If running → call process_frame()
- Else → keep motors stopped
- Exit if 'q' pressed

8. CLEANUP

- Stop motors
- Release video + camera
- Cleanup pigpio + GPIO

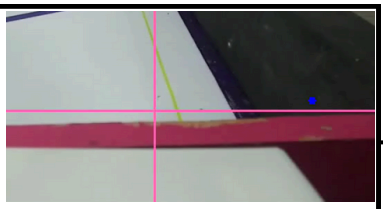
Flowchart - Open Challange



SCAN ME TO VIEW THE FLOWCHART IN DETAIL

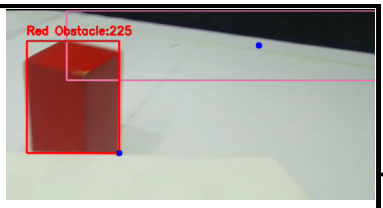
Obstacle Challenge Strategy

1. Initialization with wall detection



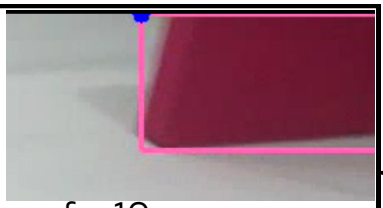
- Robot begins in the parking zone.
- It first detects the black wall:
  - If the black wall is on the right, the robot moves anticlockwise.
  - If the black wall is on the left, the robot moves clockwise.
- Based on the detected direction, the robot exits the parking zone.

2. Obstacle Handling (Red & Green Elements)



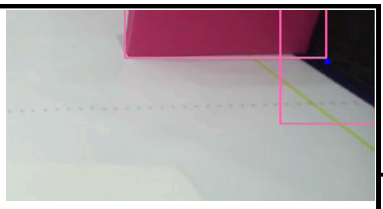
- On the path, obstacles are red and green elements.
- If one obstacle is detected:
- Red element → Robot bypasses from the right side.
- Green element → Robot bypasses from the left side.
- If two obstacles are detected, the robot follows the closer obstacle

3. Lap Counting



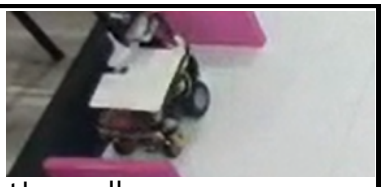
- After every lap, the purple parking zone is detected irrespective of any surrounding obstacles
- After detection, the robot cannot detect the parking zone for 10 sec.
- This process continues until the robot completes 3 +1 rounds (1 outer round)

4. Wall Following for Parking



- After 3 rounds, the robot switches to outer wall ensuring a constant distance
- It continues moving until it detects a pink wall via the IR sensors. Due to this, we are able to achieve accuracy, precision and consistency.

5. Final Parking



- Upon detecting the pink wall, the robot aligns perpendicular to it via the IMU
- It then proceeds backward until the IR sensors detects the wall.
- It then takes 2 more turns on the basis of IMU.
- Finally, the robot moves into the parking position.

Pseudo Code
Obstacle Challenge

1. IMPORTS

- Camera: picamera2
- Vision: cv2, numpy
- Timing: time, datetime
- Motor/Servo: pigpio, RPi.GPIO
- Sensors: board, adafruit_bno055

2. SETUP

- Define motor pins, servo pin, button pins
- Set motor PWM frequency
- Configure GPIO as inputs/outputs
- Define motor speeds (run, slow, stop)
- Define steering limits (center, left, right)
- Define HSV ranges for key colors (black, red, green, blue, orange, purple)
- Set camera resolution, exposure, frame rate
- Initialize camera and video recording
- Setup pigpio for motors and buttons
- Initialize IMU (BNO055) for heading tracking
- Initialize counters: line counts, rounds completed, flags (park, over, etc.)

3. MOTOR & SERVO CONTROL FUNCTIONS

- steer(angle): move servo within limits
- run(speed): move motors forward
- back(speed): move motors backward
- stop_motors(): stop motor output
- stop_robot(): stop motors + center steering

4. HEADING FUNCTIONS

- reset_heading(): reset IMU tracking
- get_heading(): read IMU, compute delta, accumulate total heading

5. MAIN FRAME PROCESSING (process_frame)

- Capture frame → convert to HSV → blur
- Create color masks (black, red, green, blue, orange, purple)
- Detect:
  - Black contours → track left/right edges of road
  - Green obstacles → avoidance
  - Red obstacles → avoidance
  - Purple target → Round Count
- Update rounds completed based on line counts
- Navigation logic:
  a) If still inside parking → perform exit maneuver
  b) If button pressed → perform manual reverse maneuver
  c) If rounds purple detected 4 times → head toward purple, then stop
  d) If obstacles (red/green) detected → steer to avoid
  e) Else → follow black contours or steer bias
- If over=True → execute parking routine
- Show debug text and save video frame

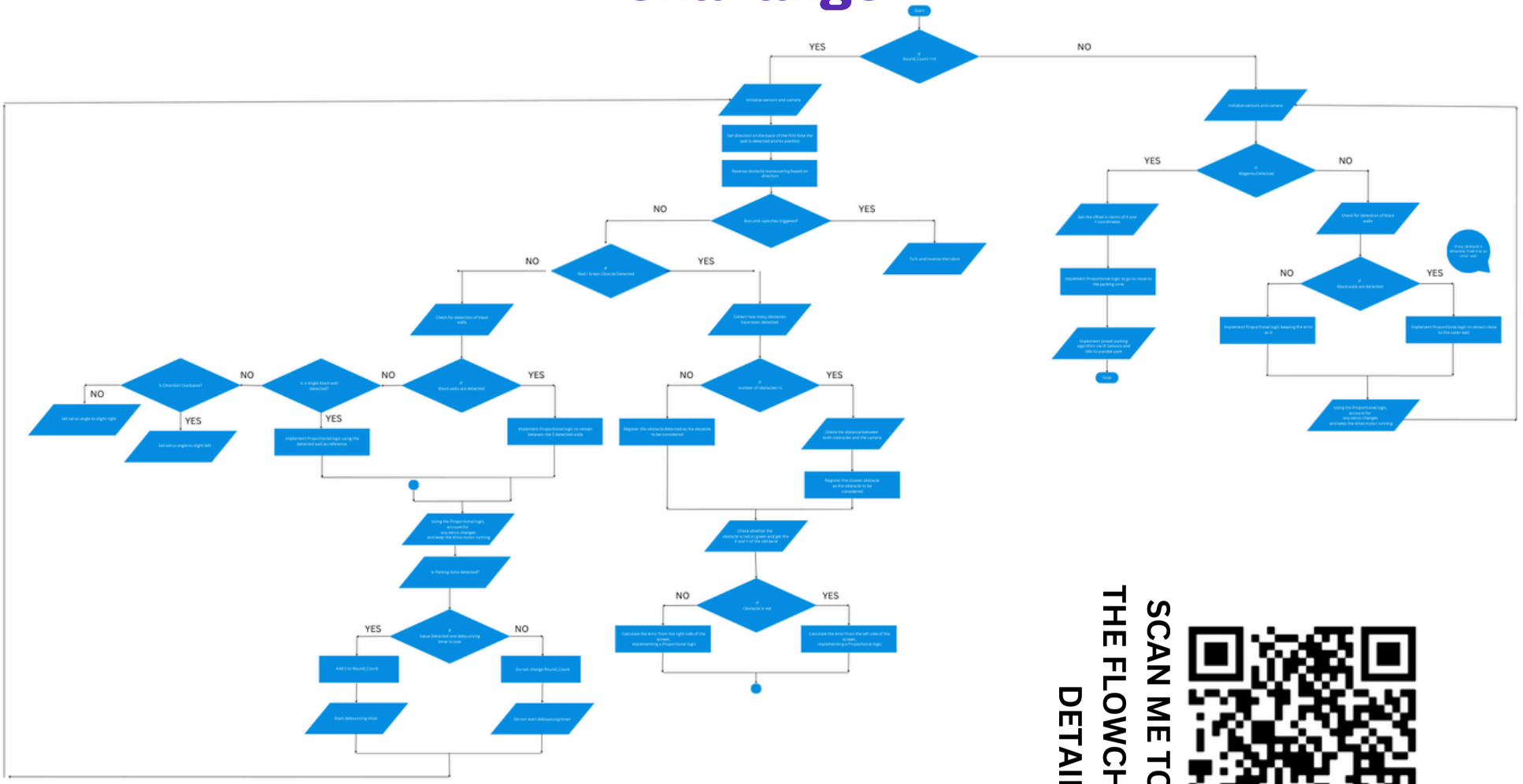
6. MAIN LOOP

- Wait for START button → set running state
- Wait for STOP button → stop and reset heading
- If running → call process_frame()
- Else → keep motors stopped
- Exit if 'q' pressed

7. CLEANUP

- stop_robot()
- Release video + camera
- Cleanup pigpio + GPIO

Flowchart - Obstacle Challange



SCAN ME TO VIEW THE FLOWCHART IN DETAIL

Code Snippets

In the below snippet, the detection logic for obstacles is shown. On the basis of predefined HSV values, detection of red is established. In order to filter it further, we set a minimum area. Since we have to go to the right of the red obstacle, we plot a point on the top right portion after which a contour is drawn around it indicating its detection.

Detection of Red Obstacles

```
red_contours, _ = cv2.findContours(red_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
red_target = None
red_y = 0
for c in red_contours:
   area = cv2.contourArea(c)
   if area > 1000: #Ensure a minimum area of detection
      x, y, w, h = cv2.boundingRect(c)
      cx = x + w #Since we have to go on the right side, we add an offset of w, the objects pixel width as seen on the screen
      cy = y + h
      cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2) #Draw a contour around it
      cv2.circle(frame, (cx, cy), 5, (255, 0, 0), -1) # Put a point on the bottom right corner
      cv2.putText(frame, f"Red Obstacle:{cy}", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
# print the cy on the screen
      if CLOCKWISE:
         if cy > red_y:
            red_y = cy
            red_target = (cx, cy)
            if rounds_completed >= 12:
               right_target = (cx,cy)
# [We take an outer left round before park. Due to this, during that we establish any obstacle as a right wall]
      else:
         if cy > red_y :
            red_y = cy
            red_target = (cx, cy)
            if rounds_completed >= 12:
               left_target = (cx,cy)
# [We take an outer right round before park. Due to this, during that we establish any obstacle as a left wall]
```

In the below snippet, the obstacle avoidance logic is shown. On the basis of the obstacle, (in this case green), the robot sets an error. The robot must deviate to the left of the green obstacle. Thus, the green obstacle must be on the right of our screen. This error is multiplied by a constant Kp, and the output causes changes in the servo position.

Detection of Red Obstacles

```
elif green_target and rounds_completed <= 12:
      run(SPEED_RUN)
      if CLOCKWISE:
         right_x, right_y = green_target
         if right_y > 100:
            steer(STEER_CENTER + ((right_x - 820) * STEER_FACTOR_GREEN_HIGH))
         else:
            steer(STEER_CENTER + ((right_x - 540) * STEER_FACTOR_GREEN_LOW))
      else:
         right_x, right_y = green_target
         if right_y > 100:
            steer(STEER_CENTER + ((right_x - 820) * STEER_FACTOR_GREEN_HIGH))
            print("820")
         else:
            print("540")
            steer(STEER_CENTER + ((right_x - 540) * STEER_FACTOR_GREEN_LOW))
```

Code Snippets

In the below snippet, the detection of the orange/blue lines is shown. The robot based on HSV values, establishes detection and filters it out by the area of the detected color. To avoid multiple detections, only after a certain time has been elapsed between prior detections can another line be detected.

Detection of Lines

```
orange_detected = False
orangeDone=0
if LINE_COUNT_MODE in ('orange', 'both'):
   contours_orange, _ = cv2.findContours(orange_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  # draw a
contour over the orange line detected
   for cnt in contours_orange:
      if cv2.contourArea(cnt) > 1000 and ParkOver:
# [only detect if getting out of the parking zone is over and the min. area for detection > 1000]
         x, y, w, h = cv2.boundingRect(cnt)
         if y + h >= LINE_DETECT_REGION_Y: # [Only within a certain distance should the contour be made]
            orange_detected = True
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 140, 255), 2)
            break
if orange_detected and (current_time - last_orange_time) > LINE_COOLDOWN:

  #only register if the line has been detected if the time between the previous detection and the current time is
#greater than the cooldown time]

   if GPIO.input(26)==GPIO.LOW or GPIO.input(12)==GPIO.LOW:
      """Since the limit switches are generally triggered at the corner, and thus due to the reverse, we count the lines
twice, we added a line_count depletion condition to avoid miscount. ]"""
      orange_line_count-=1
   orange_line_count += 1
   last_orange_time = current_time
   print(f"Orange line crossed! Count: {orange_line_count}")
```

In the below snippet, the algorithm to get out of the parking zone is shown. Using the IMU onboard the robot, we are able to easily exit the parking zone and start the round

Exit Parking

```
    if CLOCKWISE:
       if abs(get_heading()) < 20 and targ1:
 # turn1 towards the left, slightly backwards (Decrease if hitting the back wall, increase if too less)
          steer(72)
          back(85)
       elif abs(get_heading()) < 50 and targ2:
#turn2 towards the right, forwards (Increase if too less, decrease if too much)
          targ1 = False
          steer(148)
          run(85)
       elif abs(get_heading())>5 and targ3:
 # turn3 towards the left forwards (Decrease if too much, Increase if too less)
          targ2=False
          steer(72)
          run(85)
       else:
          targ3=False
          if targ3==False and targ4:
             run(0)
             steer(110)
             time.sleep(0.5)
             back(100) #come back for a fixed duration to avoid accidentally crossing over an obstacle
             time.sleep(1.1) # (teams may have to change this value)
             targ4=False
             insidePark=False
             time.sleep(0.5)
             stop_robot()
```

Into The Code

We used several libraries and modules in our code in order to achieve a smooth and reliable journey. Below, we have described their usage, function and description

- **Pigpio:** A Python-accessible GPIO library for the Raspberry Pi. This library lets the software generate precise PWM signals on the Pi's GPIO pins, which is used to control the drive motor speed (via the Spark MAX controller) and the servo steering angle. By using pigpio, the team can send high-resolution control signals to the motor controller and servo directly from Python.

- **NumPy:** A fundamental library for numerical computations in Python. NumPy is used in conjunction with OpenCV for image processing (since OpenCV images are essentially NumPy arrays). The team uses this in order to store the HSV values for color detection

- **Adafruit BNO055 Library:** A Python library that interfaces with the BNO055 IMU sensor. This library makes it straightforward to read the robot's orientation (e.g. the robot's heading/yaw angle) without handling low-level I2C communication manually. The BNO055 internally fuses accelerometer and gyroscope data to provide a stable absolute orientation reading, which the code uses for direction control.

- **Board Library:** A library that allows us to configure the 12C bus on the RPi 4, allowing for IMU communication via 12C to the board.

- **Time Library:** A library that allows us to measure the time, start timers, countdowns and allows the team to enable time based movements of the motors as well as the debouncing logic for the line detection.

- **RPi.GPIO Library:** A library that provides a straightforward path to interact with the hardware attached to the GPIO pins of the Raspberry Pi. This allows us to easily configure the pins and read the outputs provided by the various sensors connected (Limit switch and IR Sensor)

- **Cv2 Library:** A library that allows for computer vision and image processing. This library allows us to draw contours, perform HSV comparisons and allows the team to detect the obstacles, parking zone and lines with ease.

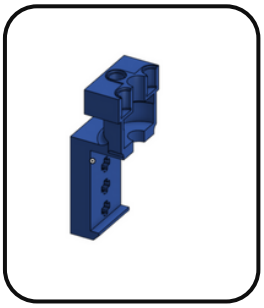
- **Picamera2 Library:** A library that allows us to easily control and communicate with the Raspberry Pi Camera module. This allows us to set the exposure, frame rate and other controllable factors.

```
from picamera2 import Picamera2
import cv2
import numpy as np
import time
import pigpio
import datetime
import board
import adafruit_bno055
import RPi.GPIO as GPIO
```

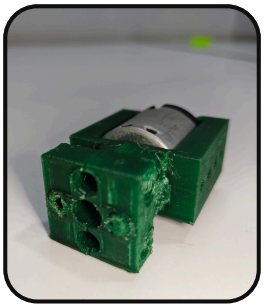
Custom Design & Development

3D printed Motor Mount

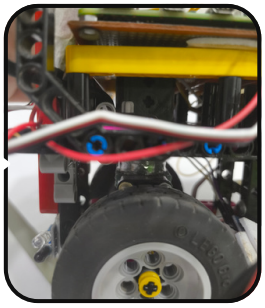
To meet the requirements of the Time Attack challenges, we have designed and customized our motor system using 3D printing material instead of relying on standard off-the-shelf solutions.



CAD Design



3D printed motor



Motor mounting on Chassis

Design Motivation

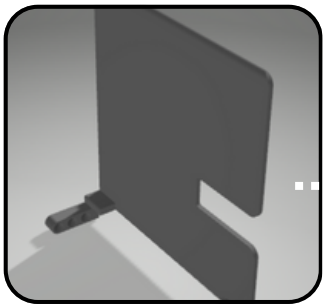
High Speed Focus: Our motor system is tuned for double the speed of a medium motor, enabling faster lap times.

Mounts and housings were designed and fabricated to integrate the motors seamlessly with our chassis, improving mechanical strength and reducing vibrations.

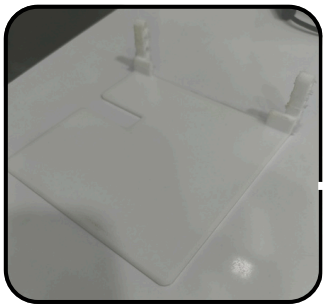
Balanced Torque: Torque output is optimized to be equivalent to a medium motor, ensuring stability during acceleration and obstacle negotiation.

3D Printed Circuit Cover

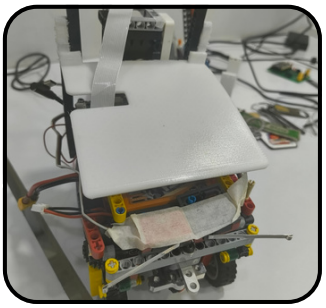
Due to the angle of the camera, the circuit components would often get detected. In order to avoid this as well as provide covering to the Zero PCB, we designed and printed a covering



CAD Design



3D printed part



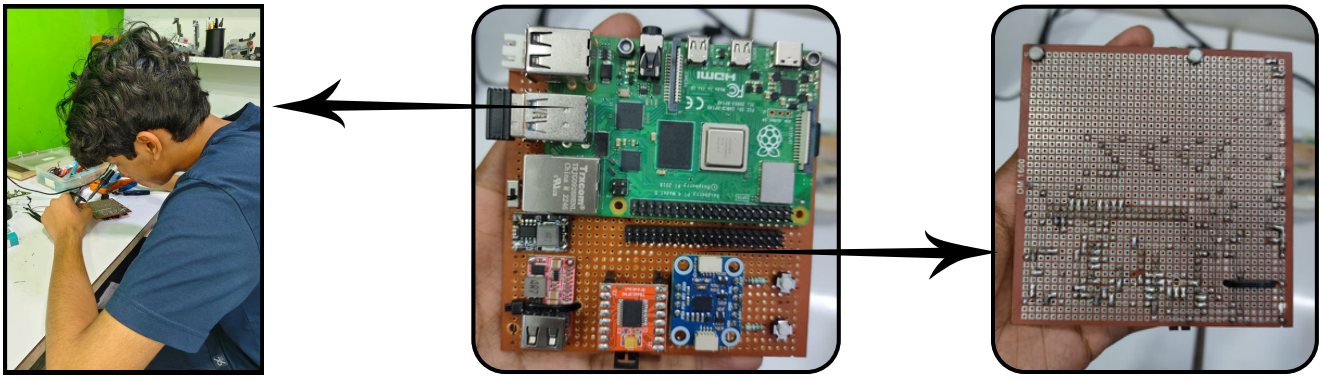
Part mounting on chassis

Custom Design & Development

Customized PCB Design

At the core of our robot, we designed a custom Zero PCB with the Raspberry Pi as the central controller. This board integrates power distribution, sensor connections, and communication lines into a single compact system, reducing wiring complexity and improving reliability. It mitigated messy wiring and helped us avoid wiring issues. Using the Cirkit Designer IDE, we were able to ensure elaborate and proper wiring.

With Circuit testing in real time to making a compact size using a zero PCB



Performance videos

Challenge Demonstration

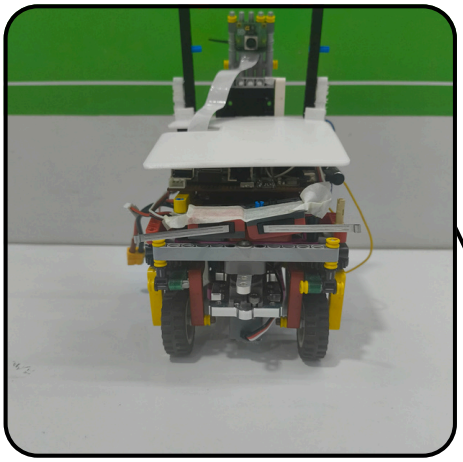
Open Challenge

This QR Code leads to a footage of our robot completing the Open Challenge. Both the direction and starting zone are randomized. The robot successfully completes 3 laps and then parks in the starting zone.

Obstacle Challenge

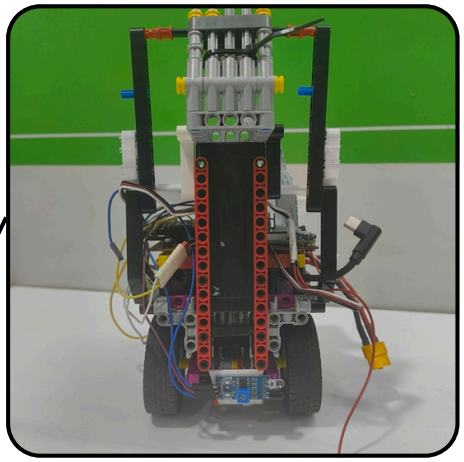
This QR Code leads to a recording of our robot completing the Obstacle Challenge. 4-8 randomized obstacles are placed along the track. If the obstacle is red, the robot deviates to the right, and if green, towards the left. The robot starts from within the parking zone whose position is also randomized. After completing 3 laps, the robot takes an additional outer lap to align perfectly with the parking zone. Then, on the basis of the IR sensors it successfully parallel parks between the 2 walls.

Voltaris Pictures

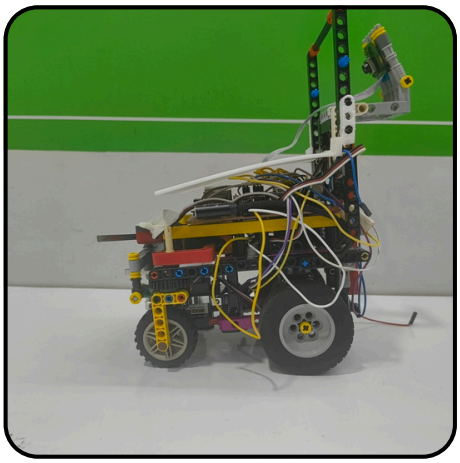


Front View

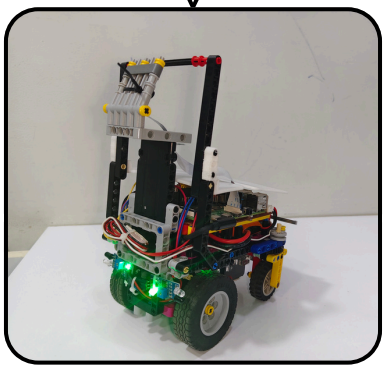
Voltaris



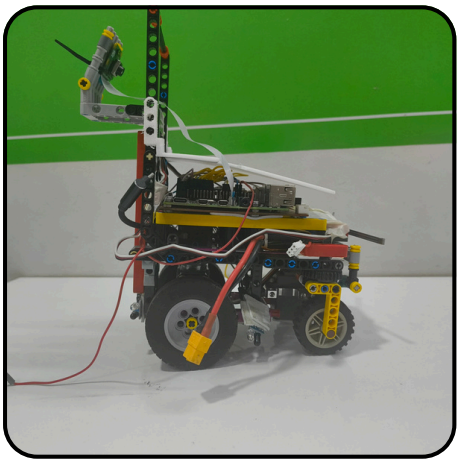
Back View



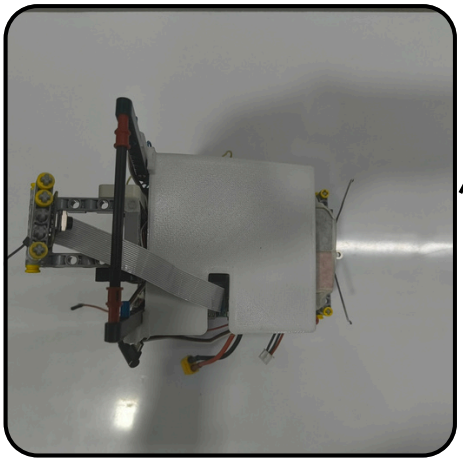
Right View



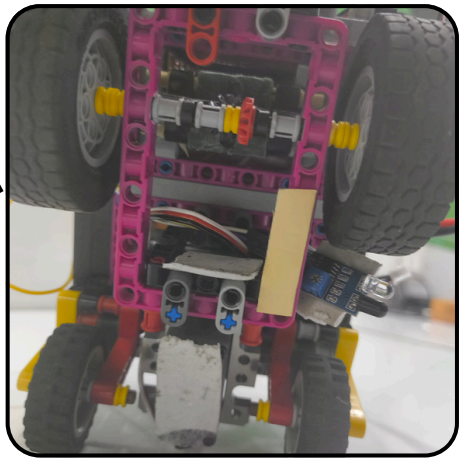
Isometric View



Left View



Top View



Bottom View

Acknowledgement to the Venerable Judges

We would like to extend our sincere gratitude to the esteemed judges for dedicating their time to thoroughly review our engineering notebook with such care and attention. We deeply appreciate your commitment to guiding us through this process and providing us with the opportunity to present our efforts and passion for technology. Your valuable insights and support have played a significant role in our growth as aspiring engineers.

Future scope

Looking forward , we aim to improve our design further by enhancing efficiency.
Our focus will be on:
Optimizing power management to extend operational runtime.

- Integrating advanced sensors and AI algorithms for more precise navigation and decision-making.

Improving modularity so the system can be easily adapted for multiple real-world applications, like in a maze solver, and autonomous
Exploring sustainable practices, including energy-efficient components and eco-friendly materials.
By continuously iterating, testing, and learning, we are developing our project into a scalable solution that can contribute to both robotics innovation and real-world problem-solving.

Furthermore, we aim to use the principles and lessons we have learnt during our journey building Voltaris, in order to build a real, self-driving car that is perfectly capable of driving autonomously despite any conditions that may hinder it.