Table of Contents

Technical Report

Our Journey

The Alliance Tech Wizards are representing Alliance University, Bangalore, India at the World Robot Olympiad – Future Engineers 2025. This report presents our experiences, preparations, and challenges in building a robot for this category.

Our journey started with a strong interest in robotics and the belief that technology can be used to solve real problems. During the preparation, we had to rethink our ideas many times, redesign mechanical parts, and adjust our coding strategies to make the robot work as expected. Each difficulty we faced helped us gain more experience and made our design stronger.

This documentation reflects both our technical progress and the teamwork that guided us throughout this project.

The Team

We are final-year students from Alliance University, working together to design and build a system for the Future Engineers challenge.

Fig 1: Team Members with Mentor

Team Members

- **Sudhir R** — Final-year Student, Department of Electronics and Communication Engineering, Alliance University, Bangalore, India
- **Marriwada Harshavardhan** — Final-year Student, Department of Electronics and Communication Engineering, Alliance University, Bangalore, India
- **Nanditha D N** — Final-year Student, Department of Computer Science and Engineering, Alliance University, Bangalore, India

Mentor

- **Dr. Harinath Aireddy**
    - Associate Professor & Director (In-Charge) – Centre of Excellence (Additive Manufacturing)
    - Director – Centre of Excellence (Maker Space)
    - Alliance University, Bangalore, India

About the Team

The Alliance Tech Wizards share a common interest in robotics, programming, and system design. Each member contributes knowledge from their field — electronics, communication, and computer science — which allows us to approach problems from multiple perspectives.

All designing and programming work for the robot is carried out by our team members. This includes mechanical design, electronics integration, and software development. By dividing responsibilities and collaborating closely, we ensure that every stage of the project reflects our collective effort and learning.

We also have access to the Maker Space at Alliance University, which is equipped with resources such as a 3D printer**.** This has been especially valuable for prototyping mechanical parts, testing new designs quickly, and iterating until we achieve the required precision and stability.

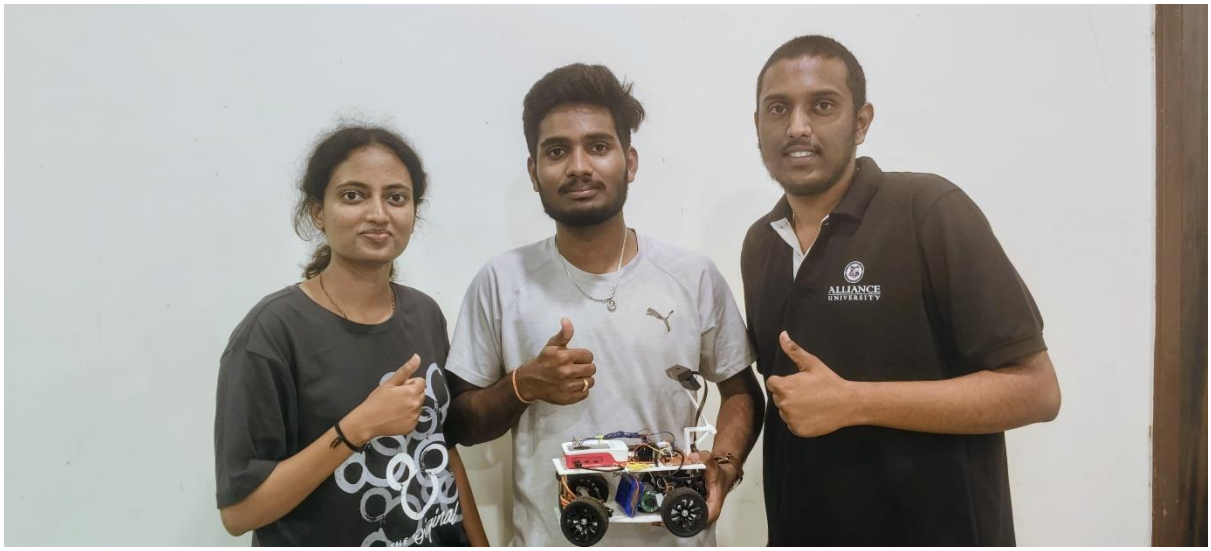


Fig 2: Funny Picture of Team Members

Motivation for Participation

Our motivation to join the Future Engineers category at WRO 2025 comes from both our past experiences and our drive to learn new skills.

We were first introduced to WRO through our mentor, Dr. Harinath Aireddy, who shared an opportunity to participate. That initial step started our journey into robotics competitions. Since then, we have participated in several events, and one of our biggest achievements was winning 2nd place at the **RoboRoarZ International 2024** in Singapore.

Over the years, we have not only continued to compete but also organized robotics-related events and workshops at our university. These activities helped us share our interest in robotics with other students and gave us the chance to inspire younger learners.

The Future Engineers category is new for us, but we see it as an opportunity to expand our knowledge. Unlike our earlier projects, here we must design and build a complete robotic vehicle, including steering, navigation, and control systems. This challenge excites us because it allows us to try ideas we have not worked on before.

With the constant support of our mentor, and our own eagerness to take on new problems, we are motivated to participate with full commitment. For us, this competition is not only about achieving results but also about learning, experimenting, and preparing ourselves for future challenges in robotics.

Problem Statement

The challenge in the Future Engineers – WRO 2025 category is based on self-driving car tasks. This season, the main focus is on Time Attack races. Unlike multi-car races, there will only be one car per attempt, and the objective is to complete several laps around the track in the shortest possible time.

The robot must drive fully autonomously, without any manual control, while adapting to both open racing tracks and tracks that include obstacles. This requires the integration of sensing, path planning, decision-making, and control in a reliable way.

Goal

Our goal is to design and develop a high-performance autonomous racing robot that can:

- Navigate the racetrack efficiently and safely.
- Handle both open tracks and tracks with obstacles.
- Use a combination of sensors, algorithms, and control systems to make real-time decisions.
- Achieving consistent lap times and overall stability throughout the race.

By the end of our preparation, we aim to build a system that not only performs well in the competition but also reflects our learning and teamwork in solving practical engineering problems.

Design Process

Before finalizing our robot's design, we went through several rounds of trial and error. Our main aim was to create a chassis that not only met our expectations but also allowed all the electronic components to fit in properly and support the required mechanisms.

At first, we designed a simple lower chassis just to test the steering mechanism and see if it matched what we had in mind. But this initial design didn't fulfill our needs — the fit for components was not right, and the setup wasn't compact enough.

Since this was our first time working on a challenge like this, we decided to study a ready-made robot available on the market as a reference. That step really helped us understand where we were lacking and gave us a clear picture of how to improve our design.

From there, we started making multiple iterations. Many of these failed because either the electronics didn't fit well, the mechanism wasn't supported, or the design became too bulky. One big challenge was the camera holder — in some versions, it captured areas outside the arena, picking up unnecessary details that disturbed the robot's functionality.

After all these refinements, we finally arrived at a design that met all our requirements. The lower chassis, upper chassis, ultrasonic sensor holder, and camera holder were all carefully designed to fit the components compactly and accurately. Every part was created with attention to dimensions, placement, and functionality, while keeping the overall robot lightweight and practical.

It was both challenging and fun to go through this process — thinking about size, mechanism, wiring space, and the exact positioning of components. Every time we made changes to the physical robot, we had to update the design to match the new electronics or mechanisms. This cycle of testing and redesigning made our final model stronger and more reliable.

Now, with this 3D-printed final design, we are ready to face the competition in both the Open Challenge and the Obstacle Challenge.

We have created this lower chassis design using Autodesk Fusion 360 software. Fusion 360 is a powerful 3D CAD (Computer-Aided Design) tool that helps in creating, simulating, and visualizing mechanical parts before actual manufacturing. It allows us to:

- Design with precision by adding accurate dimensions and constraints.
- Create 3D models that give a clear idea of how the final product will look.
- Add holes, slots, and clamps in exact positions for motors, sensors, and wheels.
- Test the design with assembly and motion simulation, reducing errors before making the real prototype.
- Save time and cost by identifying issues at the design stage itself.

Parts designed

- The robot has an upper chassis that supports the main components and sensors.
- A lower chassis forms the base and houses motors and wheels for mobility.
- A camera module holder is designed to securely mount the camera with adjustable angles for vision-based tasks.
- An ultrasonic sensor holder is created to fix the ultrasonic sensor for accurate distance measurement and obstacle detection.

1. Upper Chassis and Ultrasonic Sensor Holder Design Files

The **Upper Chassis** is designed to securely hold and organize all the key electronic components and sensors, ensuring stability, accessibility, and efficient wiring management. This part of the robot plays a critical role in keeping the overall build compact, well-balanced, and competition-ready.

Key Features of the Upper Chassis and Ultrasonic Sensor:

- **Dedicated gaps for wiring** to ensure neat routing, prevent entanglement, and allow easy maintenance.
- **Camera Holder Clamping (Primary)**: A secure mount to position the LG USB camera at an optimal angle and height for lane detection, wall monitoring, curve negotiation, and lap counting.

- **Secondary Camera Mount (Optional)**: Provision for the Raspberry Pi 5MP camera, mainly used for obstacle challenge and parking maneuvers.
- **Front & Back Pillar Rod Spots**: For supporting and stabilizing the structural frame, ensuring rigidity during high-speed runs and tight turns.
- **Raspberry Pi 5 Board Placement**: A secure slot with clearance for connectors, cooling, and easy debugging.
- **L298N Motor Driver Placement**: Positioned for efficient wiring to both motors and the Raspberry Pi.
- **ON/OFF Switch Slot**: Accessible mounting space to quickly power up or shut down the robot.
- **Zero PCB Soldering Spot**: Designed for integrating the buck converter, push button, and other minor electronics in a neat and serviceable way.

Why is this design effective now:

- Earlier prototypes included multiple ultrasonic sensor holder, but in the current refined design these have been **eliminated** to reduce weight, complexity, and power consumption.
- All perception is now handled by **camera-based sensing** (LG USB camera + Pi camera), trained to detect lane lines, walls, turns, obstacles, and parking zones. This reduces the number of mounted components while improving sensing accuracy.
- The freed-up chassis space is now used for **better airflow, wiring clearance, and weight balance**, which directly improves the bot's maneuverability and stability in curves.

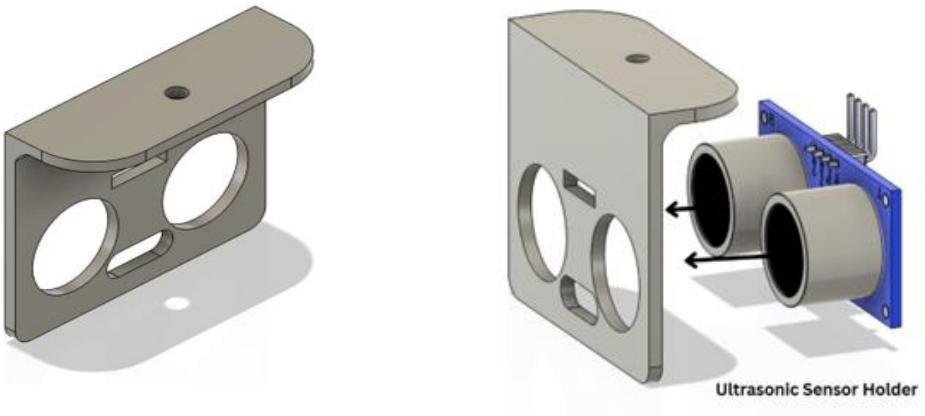


Fig 3: a) Ultrasonic sensor and holder

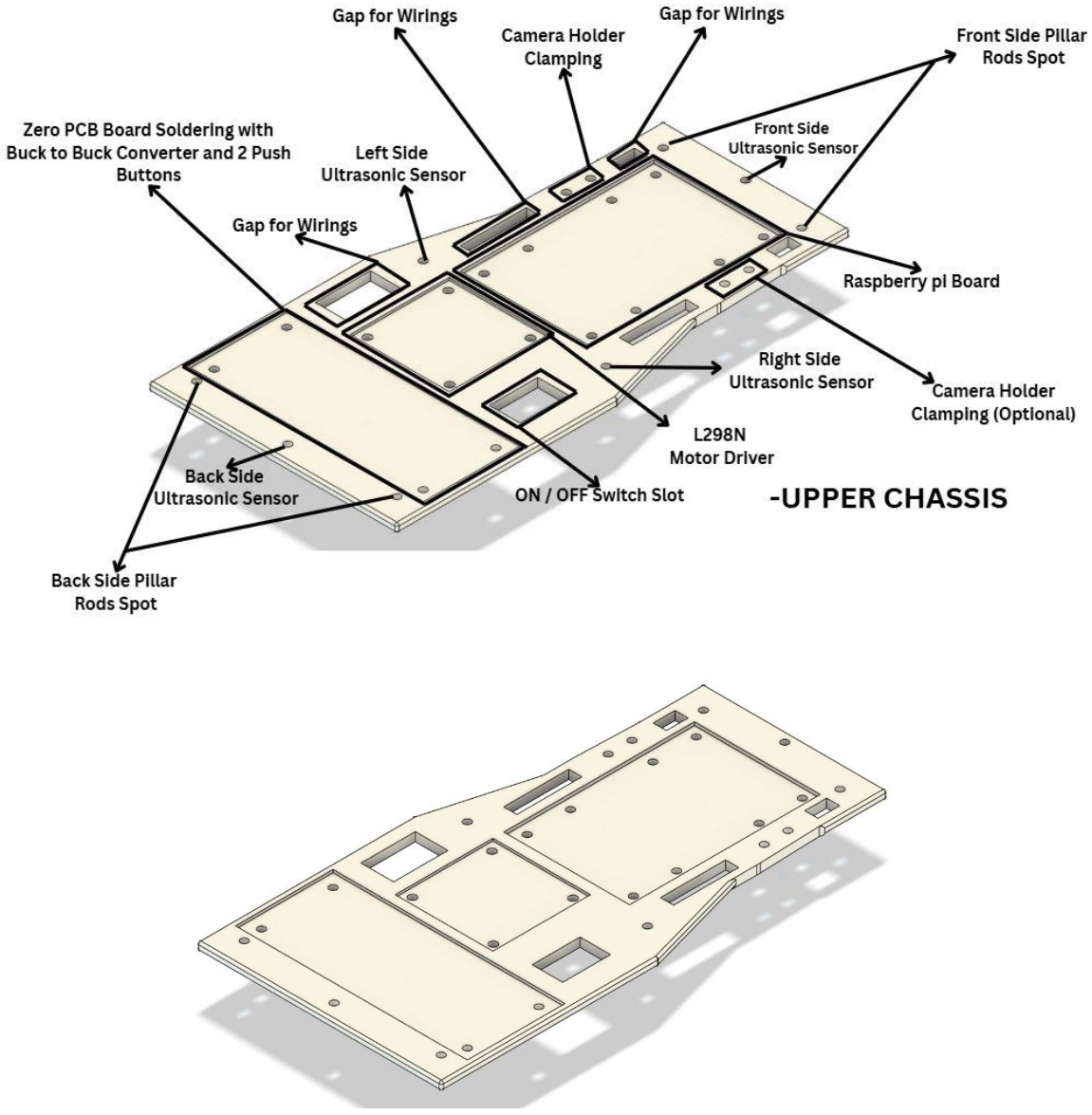


Fig 4: Upper Chassis CAD Design

2. Lower Chassis Design File

- Lower Chassis includes holding of components like
  - Left Side Rear Wheel Holder – Holds the rear wheel on the left side.
  - Right Side Rear Wheel Holder – Holds the rear wheel on the right side.
  - Right Side Front Wheel Clamp – Fixes the front wheel on the right side.

- DC Motor Clamp – Space to mount the DC motor securely.
- Servo Motor Clamp – Space to fix a servo motor.
- Place for Gears Rotation – Open slot to allow gear movement and rotation.
- Front Side Pillar Rods Spot – Points for fixing front support rods/pillars.
- Back Side Pillar Rods Spot – Points for fixing back support rods/pillars.
- Color Sensor Module Placement – Dedicated area to mount the color sensor.

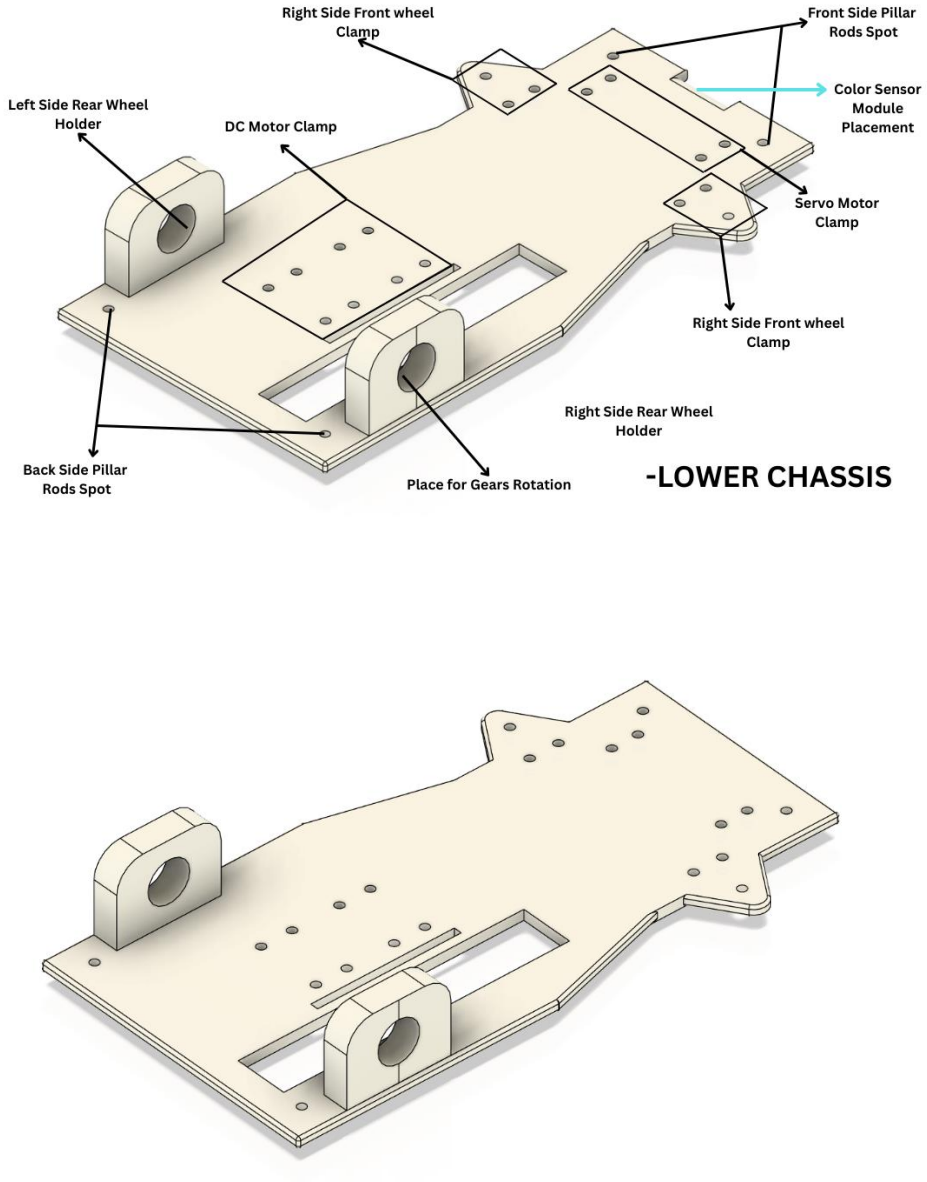


Fig 5: Lower Chassis CAD Design

3. Camera Holder

- **Purpose:** To securely mount and position a camera module onto the robot's upper chassis, allowing for stable vision input and adjustable viewing angles.

- The Camera Holder securely mounts a camera module on the robot's upper chassis.
- It has a vertical mounting plate with two long slots for adjustable camera positioning.
- Screws or nuts can fix the camera at different heights, allowing limited vertical tilt.
- The mounting plate connects to the base through a rotating hinge joint.
- This hinge allows manual tilting of the camera forward or backward.
- Enables flexible angle adjustment for tasks like object tracking, navigation, and scanning.

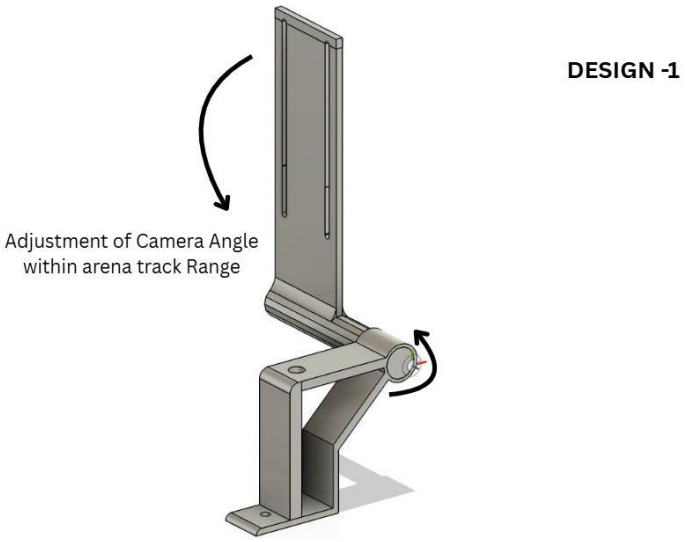


Fig 6: Camera Holder with Tilting Mechanism

The arena setup was completed using 3D-printed clamps for securing the track and specially designed joints for smooth designed joints for smoot

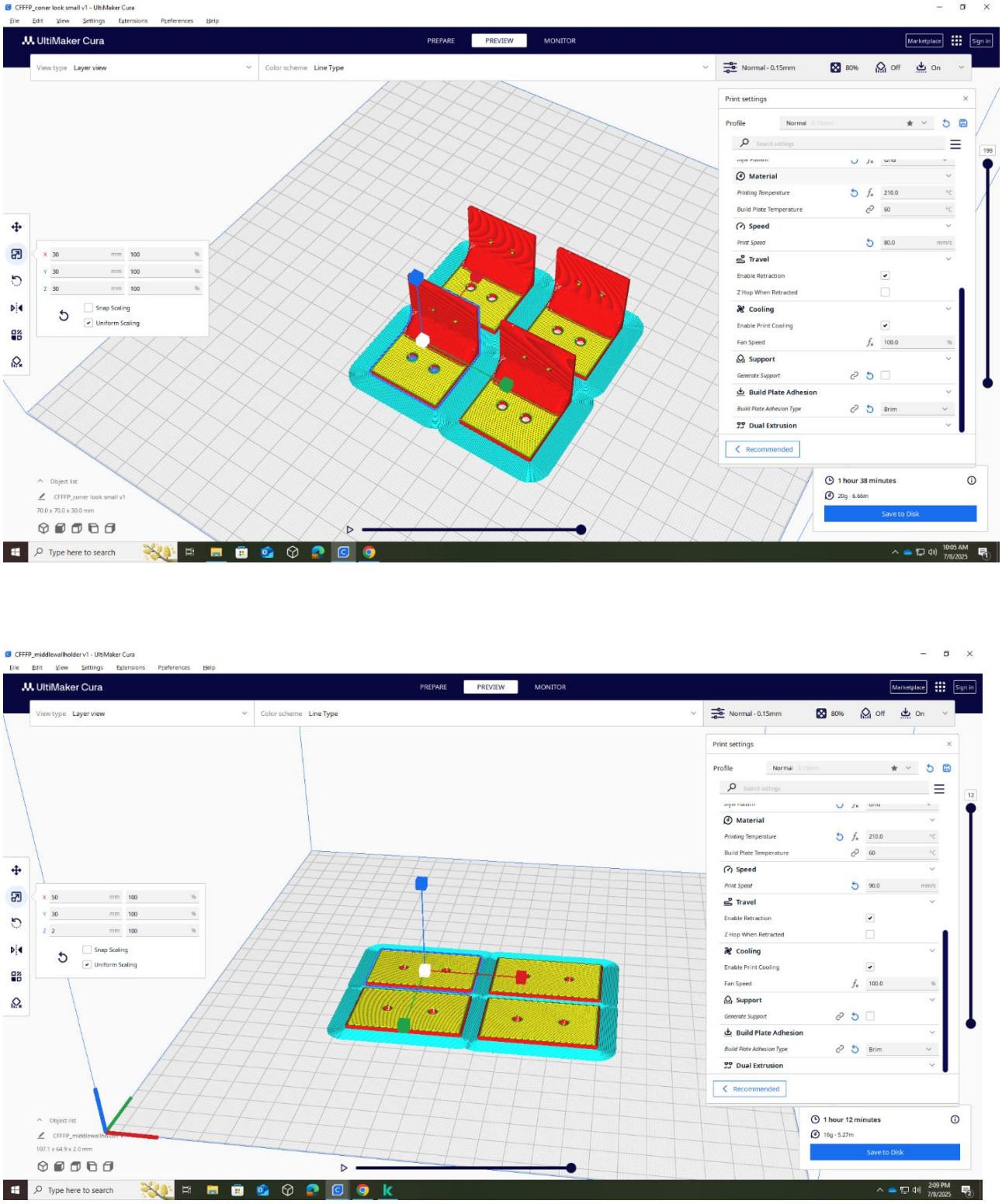


Fig 6: 3D-Printed Clamps Used for Securing the Track

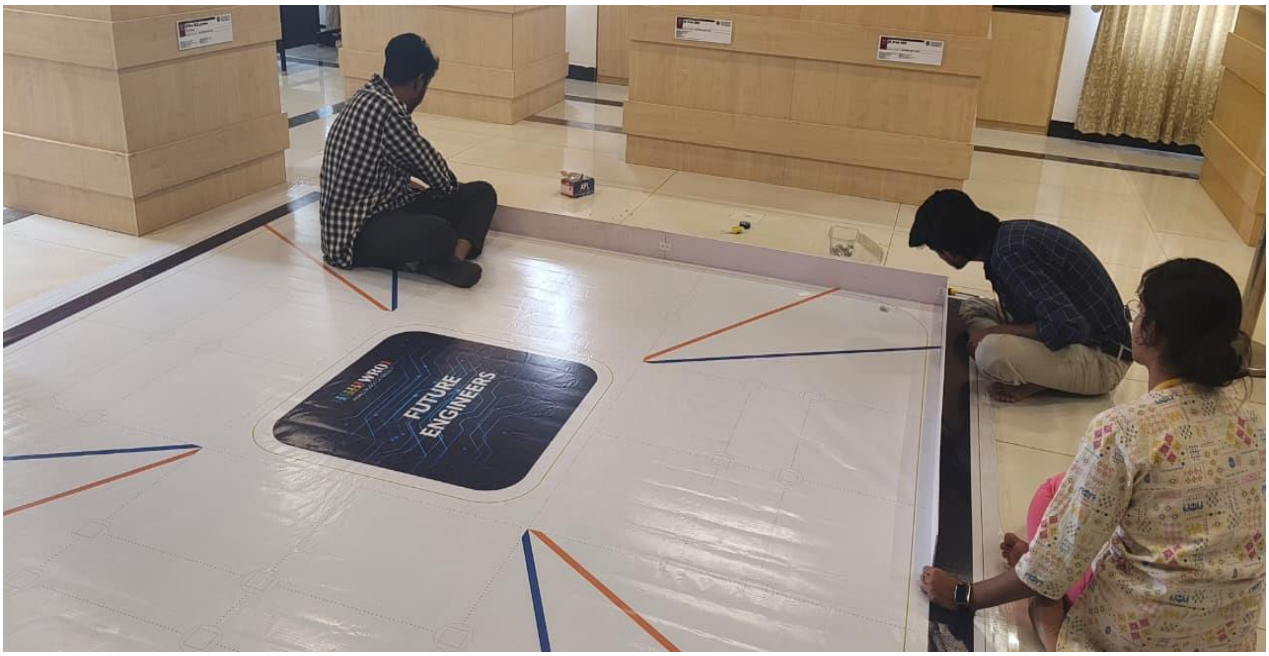


Fig 7: Assembling of the Arena Track

Manufacturing

Once the design process was finalized, the next crucial step was manufacturing the robot parts. We used a JKP 300 FDM (Fused Deposition Modeling) 3D printer for fabricating the custom components. This method was chosen because it allows quick prototyping, supports complex geometries, and gives us the freedom to update and reprint parts whenever modifications were needed.

For material, we selected PLA filament, as it provides a good balance of strength, dimensional accuracy, and ease of printing — ideal for parts that need to be both durable and lightweight.

Key Parts Manufactured

- Upper Chassis – Printed to securely hold sensors, electronic boards, and wiring, while keeping the design compact.
- Lower Chassis – Built as the structural base for motors, wheels, and batteries, ensuring stability during movement.
- Camera Holder – Manufactured to provide stable vision input with adjustable positioning, avoiding vibration and misalignment.

Printing Process and Parameters

The parts were sliced and prepared using Ultimaker Cura software, where we optimized the printing parameters to balance strength and print efficiency:

- Nozzle Temperature: 215 °C
- Bed Temperature: 60 °C
- Infill Percentage: 70%
- Infill Pattern: Grid
- Support: not enable
- Layer Height: 0.2mm
- Wall Count: 0.8mm
- Print Speed: 65 mm/sec
- Build plate Adhesion: Brim

The high infill ensured that the chassis could withstand the stress of competition, including impacts and continuous motion, while also protecting the internal electronics.

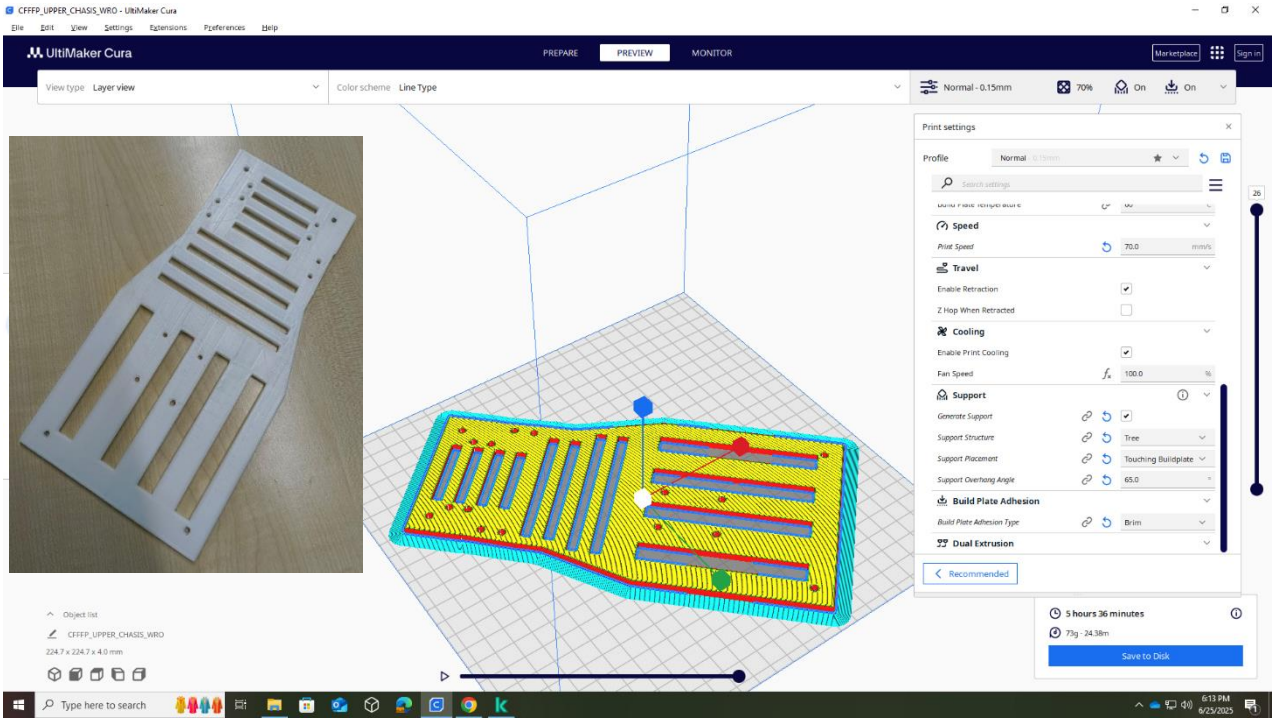


Fig 8: 3D-Printed Chassis with High Infill for Durability

Iterations During Manufacturing

Even after the designs were ready, the manufacturing stage required multiple iterations. After every print, we tested the fit of components and the robot's movement:

- In some prints, tolerances were slightly off, leading to tight fits for motors or loose holders for sensors. These were corrected by adjusting the design and reprinting.
- The camera holder design initially vibrated during motion, which interfered with vision-based tasks. Reinforcing the structure and refining the mounting angle solved this issue.
- Certain parts caused disturbances in smooth motion, so we adjusted wall thickness, infill density, and slot placements to make the chassis more rigid yet lightweight.

Importance of the Final Model

Through these construction updates, the robot became more fluent in its movement and free from unnecessary vibrations or disturbances caused by hardware limitations. The final chassis and sensor holders provided the perfect balance of strength, compactness, and functional accuracy. This robust yet lightweight structure is what now enables the robot to perform consistently in both the Open Challenge and the Obstacle Challenge.

The process of manufacturing and reprinting parts taught us the value of constant iteration — with every error corrected, the robot became stronger, more stable, and better suited for real competition.

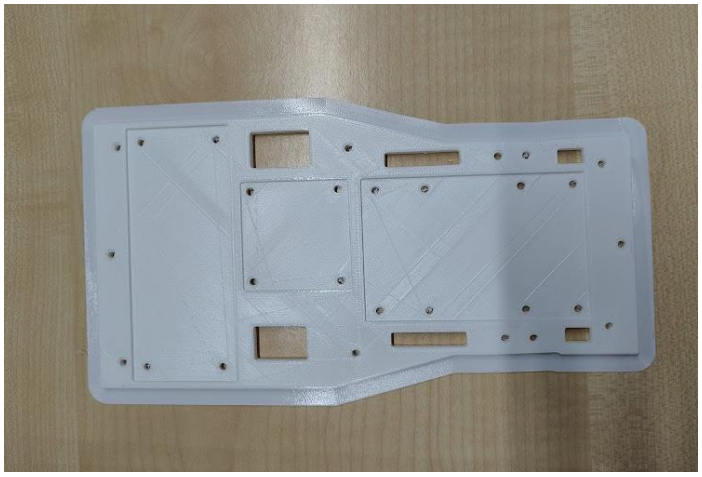


Figure 9: 3D-Printed Lower Chassis of the Robot

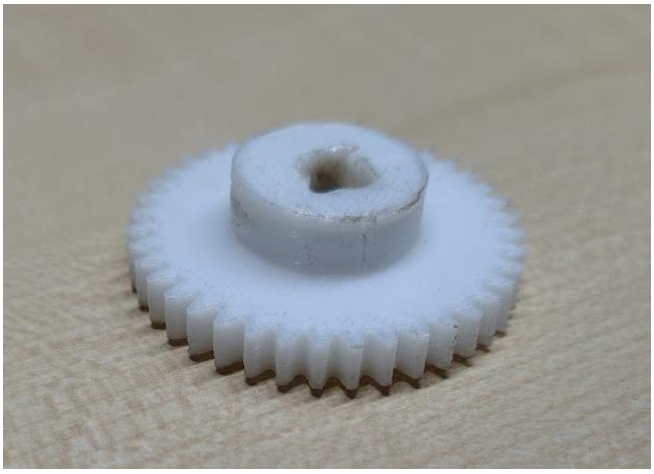
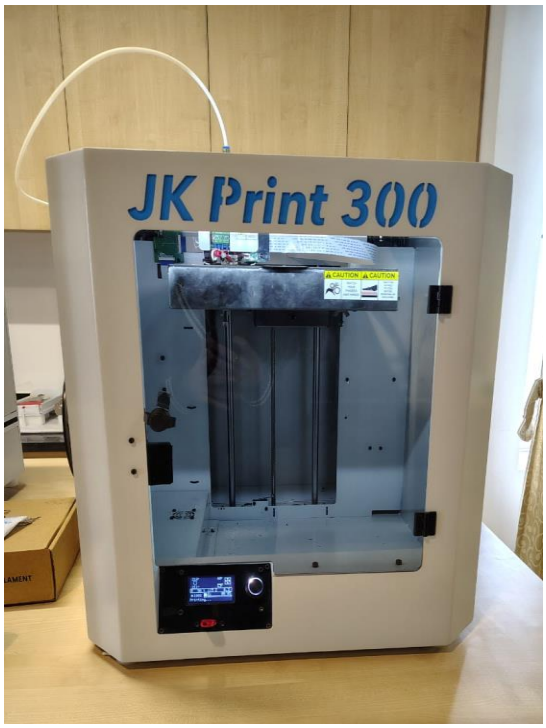


Figure 10: 3D-Printed Gear Component



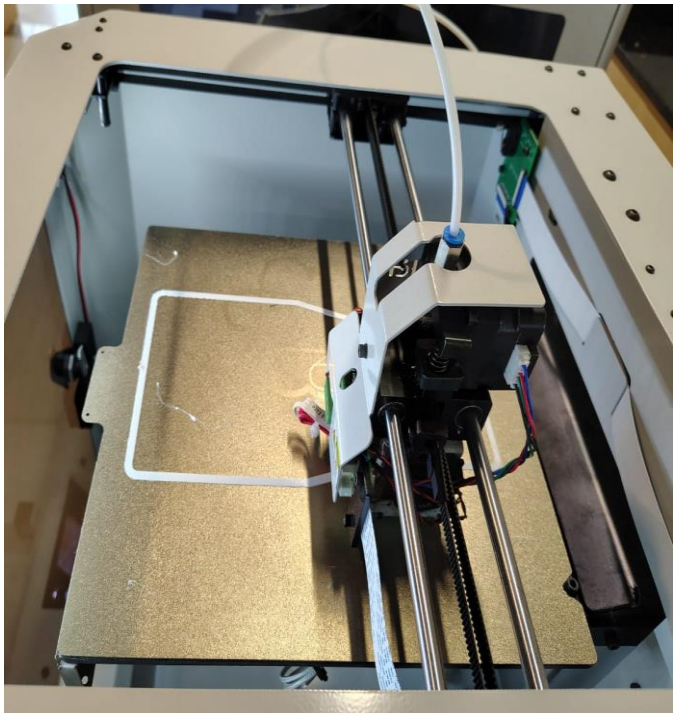


Figure 11: 3D Printing of Components Using JK Print 300

Hardware

Sl. No.	Component Name	Purpose and Use in the Robot	Components Images	Approximate Cost (₹)
1.	Raspberry Pi 5(8gb)	This is the central brain of the robot. It is a powerful single-board computer that processes data from all the sensors and makes decisions on how the robot should move and perform its tasks. It's used for computer vision, control algorithms, and overall system management.		6,000
2.	L298N Motor Driver	This component acts as an electronic switch to control the speed and direction of the DC motors. The Raspberry Pi sends low-power control signals to the motor driver, which then supplies the higher power needed to operate the motors.		90
3.	DC Motor with Encoder	Our robot uses DC motors with encoders to move forward and backward with great precision. The DC motor provides the power to turn the wheels. The attached encoder is a sensor that counts how much the motor has rotated.		700

4.	High Torque Servo Motor	The robot is a simple car-like bot with steering. The steering mechanism uses a servo motor. This servo is connected to the front wheels and provides precise steering control. This is a better method than a skid-steering system, where the wheels are controlled to move at different speeds, which is less accurate.		350
5.	Lithium-Ion Battery	This is the power source for the entire robot. Lithium-ion batteries are chosen for their high energy density, which means they can store a lot of power in a small and lightweight package, allowing the robot to operate for longer periods.		950
6.	Buck-to-Buck Converter	Also known as a step-down DC-to-DC converter, this device takes a high input voltage from the battery and efficiently steps it down to a lower, stable voltage required by components like the Raspberry Pi, which typically operates on 5V. This ensures the sensitive electronics are not damaged by the higher battery voltage.		160

7.	Red On/Off Switch	Power up the robot (ON)/ power off the robot (OFF)		50 - 100
8.	Push Buttons	A push button is used to manually start robot and reset the robot's program.		10 - 30 per button
9.	Raspberry pi Camera	The camera serves as the robot's "eyes." It captures video that are processed by the Raspberry Pi for tasks like object recognition, or identifying markers on the track. • **Limitations:** ➢ Less Wide Angle ➢ 5mp only		350
10	LG USB Camera 1080P	Gives the Higher Wide Angle and good quality Images		1300

11	wheels	65mm diameter wheel		180
12	Color sensor	Detect the color lines on arena for counting- but we used this in past now in the current design we are just using the camera for the color detection.		

Block Diagram:

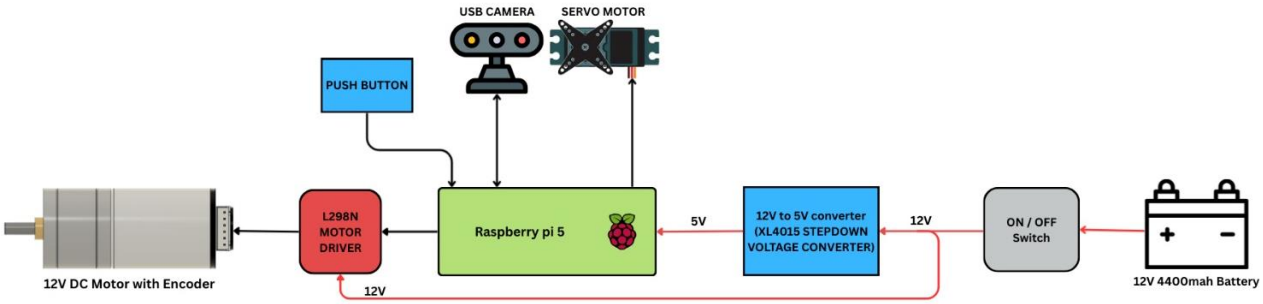


Figure 12: Block Diagram of the Robot

This is how we built the electronics for our WRO robot, which you can see in the block diagram. We chose each part carefully to make sure our robot could perform all the required tasks on the competition track.

First, we needed power supply where we used a 12V 4400mAh battery as our main power source. Since many of our components, especially the Raspberry Pi, run on 5V, we used a Buck Converter to step down the voltage. This was a critical safety step to prevent any damage to the electronics.

The core of our robot is the Raspberry Pi 5, which acts as its brain. The Raspberry Pi controls all the motors and makes decisions based on information it receives from the sensors. For movement, we used DC motors with encoders and an L298N motor driver. The driver sends power from the battery to the motors, and the encoders give the Raspberry Pi feedback on how far the robot has traveled. This feedback is essential for precise, straight-line movement.

For navigating the track, we used a variety of sensors. A USB camera and a TCS34725 RGB sensor act as the robot's "eyes," detecting lines and colors on the arena. To avoid hitting walls and other robots, we installed four ultrasonic sensors on the front, back, left, and right sides. This gave our robot a full 360-degree awareness of its surroundings. Finally, for steering, we used a servo motor to precisely turn the front wheels, which is much more efficient and accurate than just controlling the speed of the back wheels and still we are updating the robot to be more efficient in the final design.

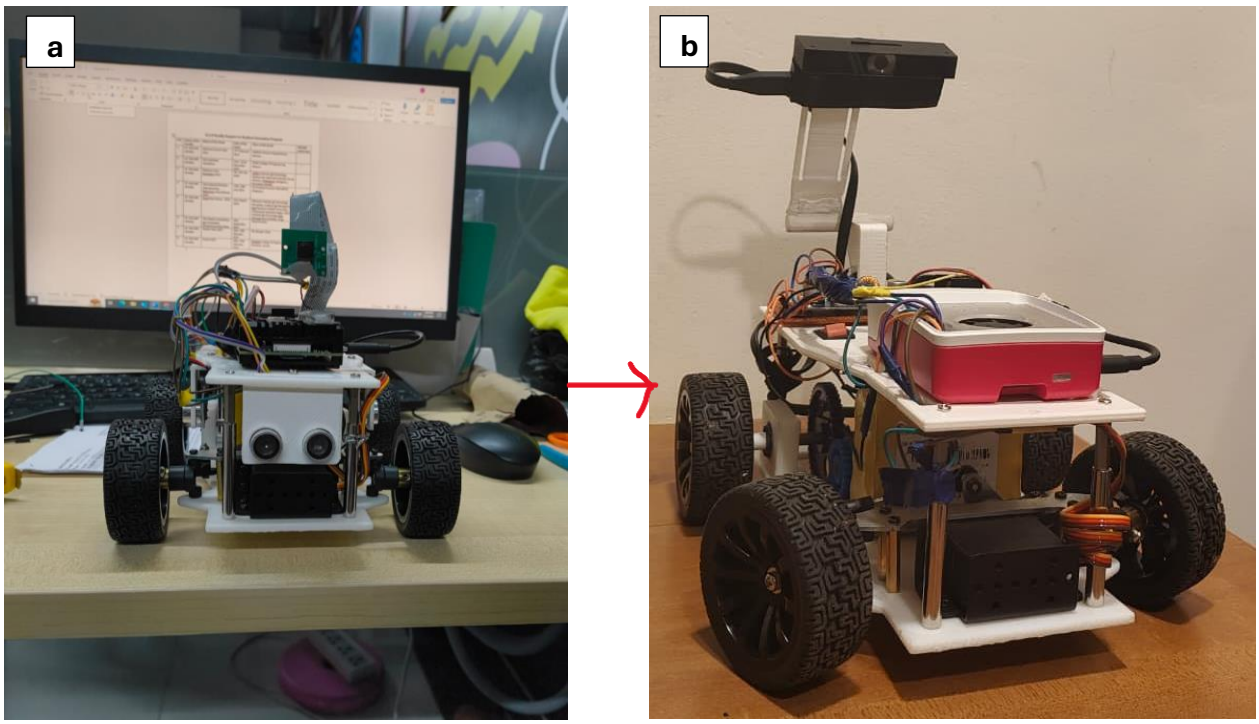
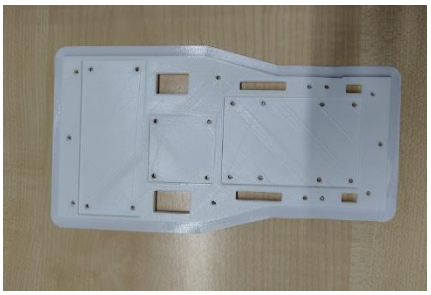
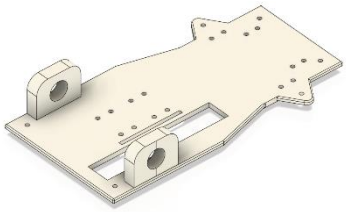
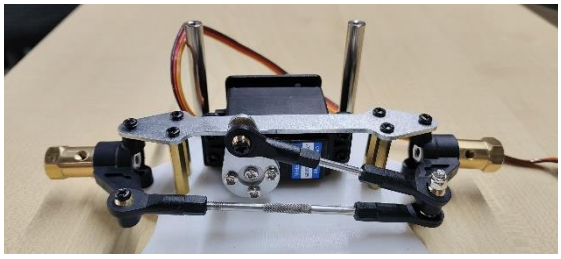
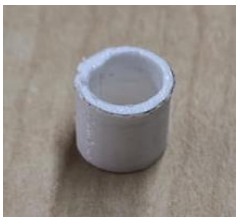
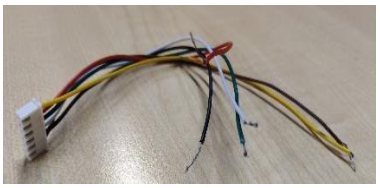
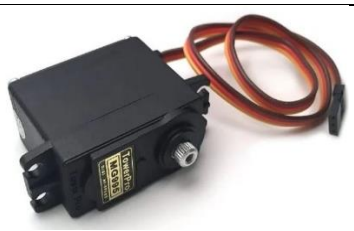


Fig 13 : (a) Bot before removing Ultrasonics sensors and few updates (b) Final Autonomous Vehicle.

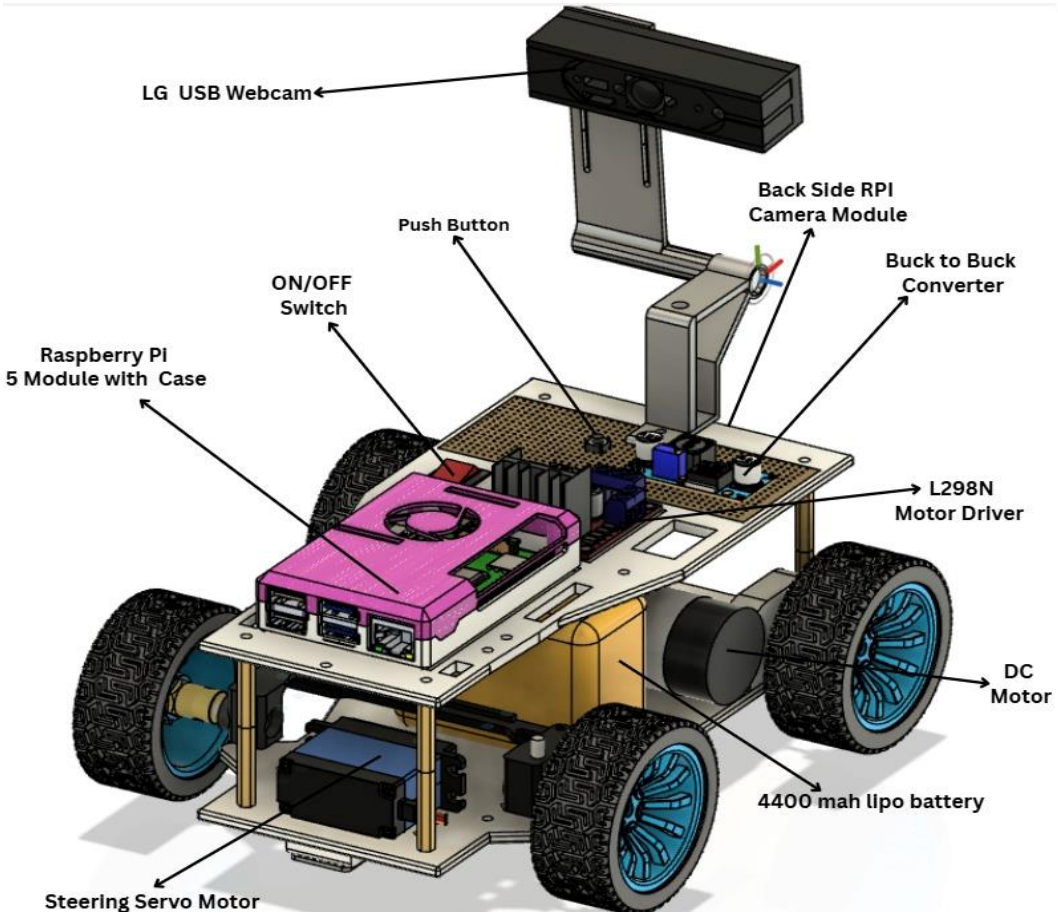
Assembly and Development

Components for Assembly of Robot		
 1. Upper Chassis	 2. Lower Chassis	 3. Ultrasonic Holder
 4. Steering Mechanism	 5. 65mm Wheel	 6. DC Motor with encoder
 7. Pillor Rods	 8. 3MM Screw	9.  3MM Nut
 10. Drive gear	 11. Pinion gear	 12. Ball bearing

 13. Motor Clamp	 14. 4MM Screw	 15. Rear Wheel Rod
 16. Wheel Supporter	 17. DC Motor Connector	 18. Li-po Battery
 19. Raspberry Pi Module	 20. Servo Motor (High Torque)	 21. Push Buttons
 22. L298N Motor Driver	 23. Buck to Buck Converter	 24. ON / OFF Switch

 25. LG PC Camera		

Image of the Complete Assembly of Components



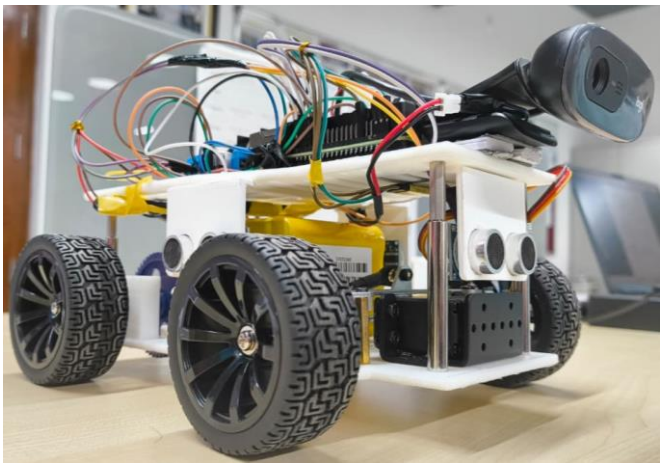
Development Stages of the Bot

- **0th Stage – Exploration and First Prototype**

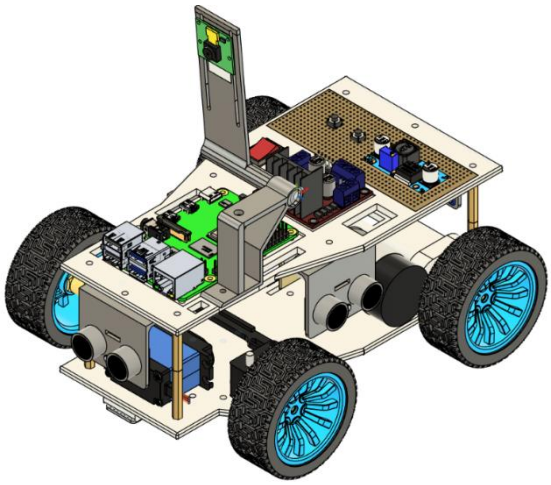
  Our journey began with designing and 3D printing the first prototype to test the **steering mechanism**. Although the performance did not fully match our expectations, this stage was an important foundation that helped us explore new concepts and identify areas for improvement.

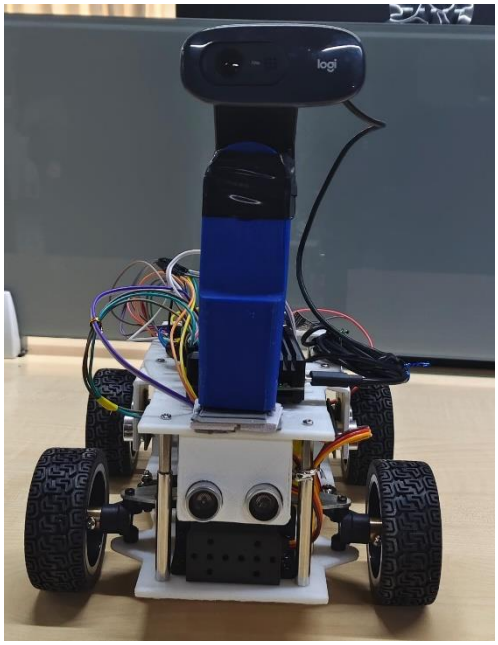
- **1st Stage – Integrating Electronics**

  Building on the first prototype, we assembled the **electronic components** onto the 3D-printed parts. This stage allowed us to validate the integration of hardware with the chassis. While the absence of a **camera holder** presented some challenges, it gave us valuable insights into design refinements needed for functionality.



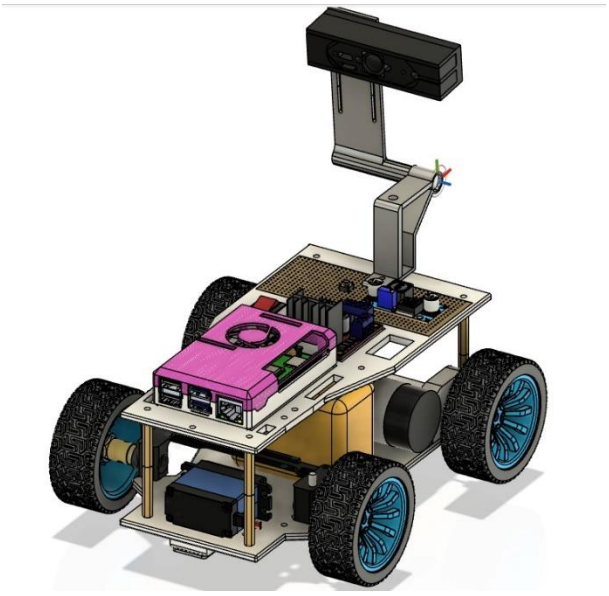
- **2nd Stage – Introducing the Camera Holder**
- To enhance stability, we designed and 3D printed a **camera holder**, which balanced the bot more effectively. This addition significantly improved performance and confirmed that iterative modifications bring the design closer to competition readiness.

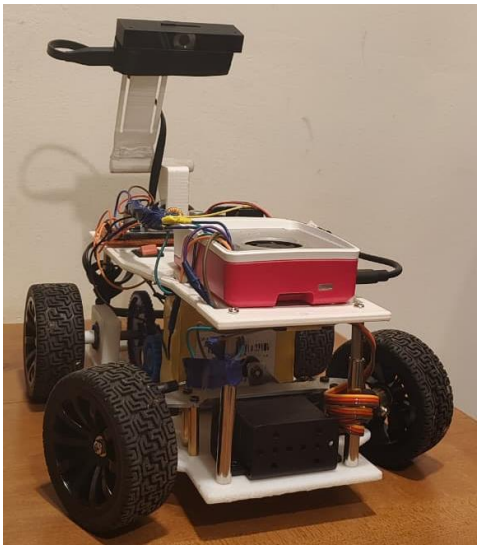




- **3rd and final working Autonomous Vechile– Upgraded Mechanism and Camera**

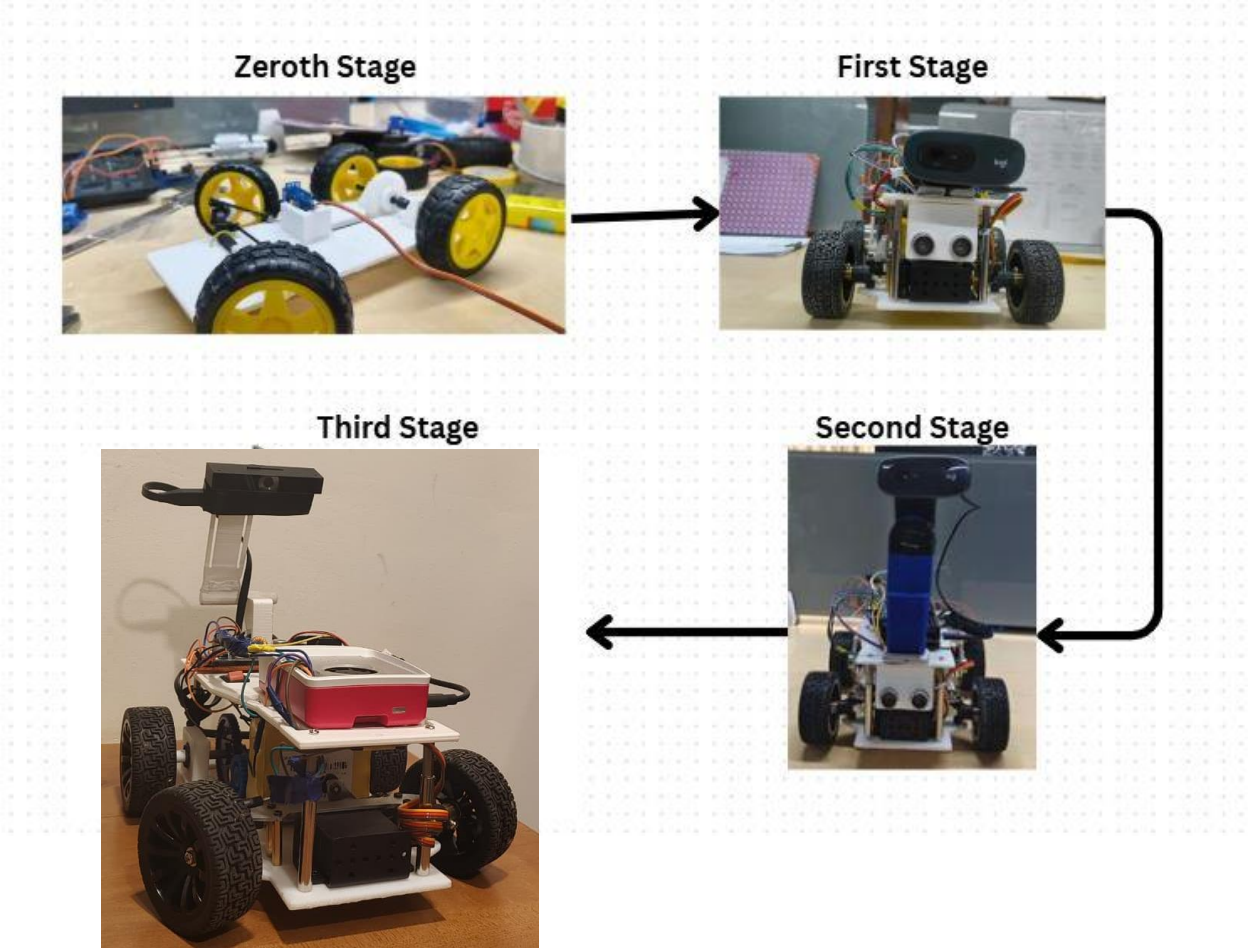
  In this stage, we introduced a **higher functionality camera** and restructured the mechanism with improved design thinking. These changes brought the bot closer to the WRO competition criteria, with better adaptability and smoother operation.





- **Final Stage – Competition-Ready Bot**

  After successive refinements, we achieved a **compact, stable, and fully functional bot**. With optimized steering, a reliable vision system, and well-integrated electronics, the bot is now prepared and trained to perform successfully in the arena. This stage reflects the culmination of our learning, innovation, and teamwork.



Software Integration

WRO2025 FE ALLIANCE TECH WIZARDS -Open Challenge

The software integration consists of three core Python files that enable a Raspberry Pi-powered robot to autonomously follow colored stripes, detect walls, make turns, and stop after completing its course. Below you'll find detailed documentation for each file.

1. masks.py

Purpose

Defines color masks for use in image processing. These are LAB color space ranges which help the robot identify different colored stripes (magenta, red, green, blue, orange) and walls (black) in the environment.

How It Works

- Each mask is a list with two NumPy arrays: the lower and upper bounds for LAB color segmentation.
- These masks are imported in other files and used to isolate specific colors in images captured by the camera.

Example Usage

Used in the find_contours function in functions.py and the main loop in main.py to detect colored stripes and walls.

File Reference

```
import numpy as np

rMagenta = [np.array([22, 150, 50], np.uint8), np.array([200, 200, 150], np.uint8)]
rRed     = [np.array([0, 150, 120], np.uint8), np.array([255, 220, 180], np.uint8)]
rGreen   = [np.array([0, 50, 0], np.uint8), np.array([255, 120, 255], np.uint8)]
rBlue    = [np.array([0, 120, 60], np.uint8), np.array([255, 160, 120], np.uint8)]
rOrange  = [np.array([0, 117, 150], np.uint8), np.array([255, 200, 200], np.uint8)]
rBlack   = [np.array([0, 105, 105], np.uint8), np.array([80, 151, 151], np.uint8)]

lotType = "light"  # 0 for dark purple, 1 for magenta (not used in main code)
```

2. functions.py

Purpose

Contains all hardware control and image processing functions. This file abstracts GPIO operations and OpenCV image processing into reusable functions.

Key Functions

- **GPIO Setup:** Configures motor and servo pins, establishes PWM channels.
- **set_motor(speed):** Controls direction and speed of DC motor using L298N.
- **set_servo_angle(angle):** Sets steering angle for the servo motor.
- **write(value):** Unified interface for controlling either steering angle or motor speed.

- **multi_write(sequence):** Executes a list of actions (angles, speeds, delays).
- **stop_car():** Safely stops the robot and resets steering.
- **move_backward_with_mirror(forward_angle, duration, speed):** Reverses with mirrored steering (for advanced maneuvers).
- **display_roi(img, ROIs, color):** Draws rectangles for specified regions of interest (ROIs) on the image, useful for debugging.
- **find_contours(img_lab, lab_range, ROI):** Segments a region of the image for a specific color and finds contours.
- **max_contour(contours, ROI):** Returns area and position of the largest contour found in an ROI.
- **display_variables(variables):** Prints debug information about main variables (e.g., detected wall areas).

How It Works

- The robot's hardware (motors/servo) are controlled indirectly by calling these functions from the main loop.
- Image processing functions enable stripe/wall detection, which informs the robot's navigation logic.

Example Usage

Used by main.py to execute movement, steering, and image analysis every frame.

File Reference

 # Example: set_motor(50) moves forward, set_servo_angle(110) steers straight.
 # Example: find_contours(img_lab, rBlack, ROI1) gets contours for black walls in left ROI.

3. main.py

Purpose

The main execution file. This script launches the robot's operational loop: waiting for button press, initializing camera and hardware, detecting stripes and walls, handling turns, counting stripes, and stopping after finishing the course.

How It Works

1. **Startup:**
   - Waits for the physical button press to start.
   - Initializes camera and sets capture parameters.
2. **ROIs (Regions of Interest):**
   - Defines five ROIs for focused color and wall detection:
     - ROI1: Bottom left (left wall)
     - ROI2: Bottom right (right wall)
     - ROI3: Center (stripe detection)
     - ROI4: Far left (orange turn trigger)
     - ROI5: Far right (blue turn trigger)
3. **Main Loop:**
   - Captures frame, processes in LAB color space.

- Detects contours for walls and stripes using masks from masks.py and functions from functions.py.
- Uses the area of detected contours to maintain wall-following via PID steering logic.
- Counts stripes (with debounce) and stops after 12 stripes.
- Handles turns when wall is lost in ROI4/ROI5 (depending on detected color).
- Displays debug info and stops on user command or course completion.

Example Usage

Run this file to start the robot after wiring and camera setup.

File Reference

```
from functions import *
from masks import rOrange, rBlack, rBlue

# Main loop:
# while True:
#     ret, img = cap.read()
#     img_lab = cv2.cvtColor(img, cv2.COLOR_BGR2Lab)
#     cListLeft = find_contours(img_lab, rBlack, ROI2)
#     ... (see full file for details)
```

Additional Notes

- **Dependencies:** Make sure to install numpy, RPi.GPIO, and opencv-python on your Raspberry Pi.
- **Hardware:** Connect L298N motor driver and a servo as per pin definitions in functions.py.
- **Camera:** Any USB webcam compatible with OpenCV should work.
- **Debugging:** Enable debug = True in main.py to visualize ROIs and variables via OpenCV window.
- **Safety:** The stop_car() function is called on exit or after finishing the course to ensure the robot halts safely.

Quick Start

1. Wire up your robot according to the pin definitions in functions.py.
2. Place all three files (masks.py, functions.py, main.py) in the same directory.
3. Run main.py on your Raspberry Pi.
4. Press the button to start the robot. Watch it follow colored stripes and walls, count stripes, make turns, and stop after the course!

functions.py — Hardware and Vision Utilities for Autonomous Robot

This file provides all the core hardware control and image processing functions for your Raspberry Pi robot car.
It abstracts GPIO operations, PWM setup, DC motor and steering servo commands, and encapsulates reusable vision functions for finding colored contours and displaying debug overlays.

Main Sections

1. GPIO and PWM Setup

- Initializes the Raspberry Pi GPIO in BCM mode.
- Sets up output pins for the L298N motor driver (IN1, IN2, ENA) and the servo (SERVO_PIN).
- Configures PWM channels for motor speed and servo position.

2. Motor Control Functions

set_motor(speed)

- Controls motor direction and speed via L298N.
- speed range: -100 (full reverse) to +100 (full forward), 0 stops.
- Used for every movement or stop command.

CENTER_ANGLE, MIN_ANGLE, MAX_ANGLE

- Constants for steering servo position.
- CENTER_ANGLE is straight ahead, MIN_ANGLE is full left, MAX_ANGLE is full right.

set_servo_angle(angle)

- Sets the servo steering angle, clamped between allowed bounds.
- Converts angle to a PWM duty cycle for accurate positioning.

3. High-Level Control Functions

write(value)

- Unified interface for controlling either steering or motor speed.
- If value is in [60,160], sets servo; if in [1000,2000], sets motor speed.

multi_write(sequence)

- Executes a sequence of actions: steering angles, speeds, or sleep delays.
- Useful for initialization or executing maneuvers.

stop_car()

- Stops all movement, centers steering, and closes OpenCV windows.
- Called on exit or emergency stop.

move_backward_with_mirror(forward_angle, duration=1, speed=50)

- Moves the car backward, steering mirrored relative to the last forward angle.
- For advanced reversing maneuvers.

4. Image Processing Functions

display_roi(img, ROIs, color)

- Draws rectangles (ROIs) on the image for visualizing detection zones.
- Used for debugging and live feedback in OpenCV window.

find_contours(img_lab, lab_range, ROI)

- Crops the specified ROI from the LAB image.
- Applies LAB color masking, erodes/dilates to reduce noise.
- Finds and returns contours matching the color mask.

max_contour(contours, ROI)

- Selects the largest contour in a region, computes area and position.
- Returns [maxArea, maxX, maxY, mCnt] for use in steering and wall detection.

display_variables(variables)

- Prints key debug information (areas, angles, detected colors, etc.) to the console.
- Overwrites previous values for real-time updates.

How This File Is Used

- **Imported in main.py** for all hardware and vision operations.
- **Keeps main loop clean:** Complex hardware logic and image analysis are abstracted away for clarity and reuse.
- **Facilitates debugging:** Functions like display_roi and display_variables enable easy real-time monitoring.

Typical Usage Example

```
from functions import *

# Set motor speed and direction
set_motor(80)  # Move forward

# Set steering angle
set_servo_angle(110)  # Straight

# Mask and find contours in a camera image
contours = find_contours(img_lab, rOrange, ROI3)
area, x, y, cnt = max_contour(contours, ROI3)

# Stop the car
stop_car()
```

Troubleshooting

- **Servo or motor not responding:** Check GPIO pin assignment and connections.
- **Vision functions not finding contours:** Ensure correct LAB mask values and ROI placement; tune masks with your LAB mask tuner tool.
- **Debugging:** Use display_roi and display_variables to monitor real-time system state.

References

- RPi.GPIO documentation
- OpenCV Python reference
- L298N Motor Driver
- Servo PWM basics

masks.py — Color Mask Definitions

This file defines color masks for use in image processing, specifically in the LAB color space. These masks allow the robot to reliably detect various colors in its environment, which is essential for stripe following, wall detection, and course logic.

What are Color Masks?

A **color mask** is a set of lower and upper bounds in a color space (here, LAB) used to isolate pixels of a specific color from an image. For example, to find all "orange" pixels, you use the rOrange mask.

Why LAB Color Space?

- **LAB** color space separates lightness (L) from color channels (A/B), making color detection more robust against lighting changes and shadows compared to RGB.
- Each mask is a pair of NumPy arrays: [lower_bound, upper_bound].

List of Masks Defined

- **rMagenta**: For magenta stripes.
- **rRed**: For red stripes.
- **rGreen**: For green stripes.
- **rBlue**: For blue stripes (used for one course path).
- **rOrange**: For orange stripes (used for the alternate course path).
- **rBlack**: For wall detection (black lines or walls).

All masks are defined as follows:

```
rMagenta = [np.array([22, 150, 50], np.uint8), np.array([200, 200, 150], np.uint8)]
rRed     = [np.array([0, 150, 120], np.uint8), np.array([255, 220, 180], np.uint8)]
rGreen   = [np.array([0, 50, 0], np.uint8), np.array([255, 120, 255], np.uint8)]
rBlue    = [np.array([0, 120, 60], np.uint8), np.array([255, 160, 120], np.uint8)]
rOrange  = [np.array([0, 117, 150], np.uint8), np.array([255, 200, 200], np.uint8)]
rBlack   = [np.array([0, 105, 105], np.uint8), np.array([80, 151, 151], np.uint8)]
```

How are Masks Used?

- **Imported** in functions.py and main.py.
- Passed into image processing functions like find_contours() to segment and find contours of the specific color within a region of interest (ROI).
- Used in stripe detection, wall following, and turn logic.

Example Usage

```
from masks import rOrange, rBlack
```

 # In an image processing function:
orange_contours = find_contours(img_lab, rOrange, ROI3)
black_contours = find_contours(img_lab, rBlack, ROI1)

- This finds all orange stripes in the main ROI and all black wall contours in the left ROI.

Additional Variable

- **lotType**: Set to "light" (purpose: for lighting condition or mode selection, but not actively used in main code).

Why Is This Important?

Correct color mask definitions are critical for reliable robot vision. - Too narrow: robot may miss stripes/walls under different lighting. - Too broad: robot may detect noise or wrong objects.

Troubleshooting

- If your robot fails to detect stripes or walls, you may need to tune these mask values for your particular lighting and camera.
- Use OpenCV tools to sample LAB values from your camera images in your actual environment.

References

- LAB Color Space
- NumPy Arrays
- OpenCV InRange Function

masks.py — Color Mask Definitions

This file defines color masks for use in image processing, specifically in the LAB color space. These masks allow the robot to reliably detect various colors in its environment, which is essential for stripe following, wall detection, and course logic.

What are Color Masks?

A **color mask** is a set of lower and upper bounds in a color space (here, LAB) used to isolate pixels of a specific color from an image. For example, to find all "orange" pixels, you use the rOrange mask.

Why LAB Color Space?

- **LAB** color space separates lightness (L) from color channels (A/B), making color detection more robust against lighting changes and shadows compared to RGB.
- Each mask is a pair of NumPy arrays: [lower_bound, upper_bound].

List of Masks Defined

- **rMagenta**: For magenta stripes.
- **rRed**: For red stripes.
- **rGreen**: For green stripes.
- **rBlue**: For blue stripes (used for one course path).
- **rOrange**: For orange stripes (used for the alternate course path).
- **rBlack**: For wall detection (black lines or walls).

All masks are defined as follows:

```
rMagenta = [np.array([22, 150, 50], np.uint8), np.array([200, 200, 150], np.uint8)]
rRed     = [np.array([0, 150, 120], np.uint8), np.array([255, 220, 180], np.uint8)]
rGreen   = [np.array([0, 50, 0], np.uint8), np.array([255, 120, 255], np.uint8)]
rBlue    = [np.array([0, 120, 60], np.uint8), np.array([255, 160, 120], np.uint8)]
rOrange  = [np.array([0, 117, 150], np.uint8), np.array([255, 200, 200], np.uint8)]
rBlack   = [np.array([0, 105, 105], np.uint8), np.array([80, 151, 151], np.uint8)]
```

How are Masks Used?

- **Imported** in functions.py and main.py.
- Passed into image processing functions like find_contours() to segment and find contours of the specific color within a region of interest (ROI).
- Used in stripe detection, wall following, and turn logic.

Example Usage

```
from masks import rOrange, rBlack
```

 # In an image processing function:
orange_contours = find_contours(img_lab, rOrange, ROI3)
black_contours = find_contours(img_lab, rBlack, ROI1)

- This finds all orange stripes in the main ROI and all black wall contours in the left ROI.

Tuning and Updating Masks

LAB color masks may need to be adjusted for different lighting conditions, cameras, or colored materials. To update these values:

5. **Use the LAB Mask Tuner Tool (lab_mask_tuner.py in this repo):**
   – This script lets you interactively adjust LAB range sliders while viewing a live mask overlay.
   – Place your sample object in the ROI, tune the sliders, and press q to print the final LAB bounds.
6. **Replace the values in masks.py:**
   – Copy the printed lower and upper bounds into your mask definitions.

This ensures your robot detects colors reliably in your specific environment.

Additional Variable

- **lotType**: Set to "light" (purpose: for lighting condition or mode selection, but not actively used in main code).

Why Is This Important?

Correct color mask definitions are critical for reliable robot vision. - Too narrow: robot may miss stripes/walls under different lighting. - Too broad: robot may detect noise or wrong objects.

Troubleshooting

- If your robot fails to detect stripes or walls, use the LAB Mask Tuner to find better values for your lighting/camera.
- Use OpenCV tools to sample LAB values from your camera images in your actual environment.

References

- LAB Color Space
- NumPy Arrays
- OpenCV InRange Function
- LAB Mask Tuner Tool

Strategic Plans for Completing the Open Challenge Round

WRO Future Engineers – Open Round Participation

1. Introduction

The **World Robot Olympiad (WRO) Future Engineers category** challenges participants to design autonomous robotic vehicles capable of navigating dynamic and unpredictable race tracks. Unlike conventional robotics contests, the **Open Challenge Round** has no static traffic signs or predefined layouts. Instead, the arena configuration is randomized before each round using **coin tosses and dice rolls**, ensuring that participants must design robust, adaptive systems.

This report documents our **team's systematic strategy** for preparing the autonomous bot to successfully complete the Open Challenge Round. The emphasis of our work was on **vision-based navigation**, **dynamic path planning**, **real-time obstacle detection**, and **autonomous lap counting** under varying corridor widths and randomized starting conditions.

2. Understanding Challenge Constraints

According to the **general rules of the WRO Future Engineers Open Round**:

1. **Starting Section Determination**
    - Two coin tosses decide the arena section where the bot will be placed.
    - Example: *Tails & Heads = Section X.*
2. **Corridor Width Assignment**
    - Four sequential coin tosses define corridor widths in each section:
        - **Heads = Wide Corridor (~1000 mm ±100 mm)**
        - **Tails = Narrow Corridor (~600 mm ±100 mm)**
    - These variations simulate **real-world driving conditions**, where lane widths differ.
3. **Starting Zone Selection**
    - A dice roll (1–6) specifies the exact starting grid within the chosen section.
    - If the roll corresponds to a wall-enclosed area, the dice is rolled again.

Thus, every round forces the vehicle to adapt to **randomized starting points, corridor geometries, and navigation pathways**, reinforcing the competition's emphasis on **autonomous adaptability**.

3. Initial Approach and Limitations

We began with a **three-zone vision model**:

- **Zones A & B:** Used for wall detection.
- **Zone C:** Used for detecting arena color lines (orange/blue) to decide driving direction (clockwise/anticlockwise).

While this enabled **basic detection**, the bot struggled with **sharp turns**, failed to re-align in **narrow corridors**, and exhibited frequent **wall collisions** due to insufficient environmental awareness.

This led us to refine the architecture into a **five-zone vision system**, supported by adaptive control algorithms.

4. Refined Strategic Plan

4.1 Camera-Based Zonal Division

The bot employs an **LG 1080p PC Camera** as its primary vision sensor. The captured frame is segmented into **five functional zones** (Figure 1):

- **Zone A (Upper-Left):** Anticlockwise turn prediction.
- **Zone B (Left-Center):** Left corridor detection.
- **Zone C (Center):** Directional color line detection and lap counting.
- **Zone D (Right-Center):** Right corridor detection.
- **Zone E (Upper-Right):** Clockwise turn prediction.

This segmentation provides **redundant yet complementary data** for robust path planning.

4.2 Direction Recognition via Color Detection

The arena floor contains **two contrasting line colors (orange and blue)**. The first detected line determines direction:

- **Orange Line:** Clockwise movement.
- **Blue Line:** Anticlockwise movement.

The bot is programmed with a **single-color tracking algorithm**:

- Once a direction is locked, the bot ignores the alternate color.
- This prevents **false triggers** when crossing overlapping or residual lines.

4.3 Corridor Navigation and Centralization

- **Zones B & D** continuously monitor the **percentage of black pixels** (representing corridor walls).
- If asymmetry is detected (e.g., B > D), the bot corrects by steering proportionally toward the less obstructed side.
- This ensures the vehicle maintains a **central trajectory** regardless of whether the corridor is **wide or narrow**, as dictated by the coin toss sequence.

This strategy effectively simulates **lane-keeping assist systems** in real autonomous vehicles.

4.4 Turn Detection and Execution

- **Zone A (for anticlockwise runs)** and **Zone E (for clockwise runs)** are activated during cornering.
- As the bot approaches a section boundary, these zones detect **white (arena mat)**.
- Upon detection:
    - The bot reduces speed (for stability).
    - Executes a **curved trajectory** turn instead of a sharp pivot.
    - Gradually accelerates after realignment into the new section.

This approach eliminates sudden jerks, ensuring **smooth trajectory transitions** and minimizing collision probability.

4.5 Lap Counting Algorithm

The bot uses **color detection** and a **counter** to track laps:

- The **first detected color** in Zone C (orange or blue) becomes the **reference**.

- Each time the bot crosses the **reference line (orange)**, the counter increases.
- **Lap milestones**:
    - 4 counts → Lap 1 complete
    - 8 counts → Lap 2 complete
    - 12 counts (orange) → Lap 3 nearly complete
    - +1 blue detection → Confirms final section.
- After the **12th orange** and **1 blue** detection, the bot moves forward for **4 seconds** before stopping to ensure it halts in the final zone or section from where the bot started.

This method provides **consistent lap recognition without relying on external markers**.

5. Integrated System Workflow

1. **Initialization** – Bot is placed according to toss/dice results.
2. **Direction Selection** – Zone C detects the first line color → direction locked.
3. **Corridor Alignment** – Zones B & D keep vehicle centralized.
4. **Turning** – Zone A or E activates when white mat detected → smooth cornering.
5. **Lap Monitoring** – Zone C counts color detections.
6. **Termination** – At 12 detections, bot halts after 4 seconds → completion.

This cycle allows the bot to operate **autonomously with adaptive decision-making** in randomized arenas.

6. Results and Observations

Through iterative training and testing:

- **Wall Collisions Eliminated** – Centralization strategy (Zones B & D) maintained stability.
- **Smooth Turns Achieved** – Predictive cornering ensured minimal overshoot.
- **Accurate Lap Completion** – Counter-based detection consistently identified 3 laps.

- **Rule Compliance** – Bot adhered to randomized start/arena conditions as per WRO guidelines.

The bot demonstrated **consistent, reliable navigation** across both narrow (600 mm) and wide (1000 mm) corridors.

Our team successfully implemented a **camera-zonal vision model** integrated with **decision-based algorithms** for direction, centralization, turning, and lap recognition. By combining **color-based navigation, corridor detection, and adaptive control strategies**, the bot achieved **stable autonomous driving** in compliance with WRO Future Engineers Open Round rules.

Final Vehicle Design and Source Code

a) Mobility Management

Mobility management in the vehicle primarily focuses on how the bot's movements are controlled and coordinated based on sensory input and motor actuation. The system architecture is designed to ensure smooth navigation, obstacle avoidance, and accurate response to arena conditions during both the **Open Challenge** and the **Obstacle Challenge**.

Sensors and Control Input

The main sensory unit integrated into the bot is the **LG PC 1080P camera module**, strategically trained according to the gameplay rules of the arena. The vision of this camera is segmented into **five regions of interest (ROI)** as shown in Figure.1:

- **A –** Left Wall/ Corridor Detection Zone
- **B –** Left Zone (White/Black Detection for Turns)
- **C –** Central Zone (Color Line Detection & Lap Counting)
- **D –** Right Zone (White/Black Detection for Turns)
- **E –** Right Wall/ Corridor Detection Zone

1. **Two regions (A & E)** are dedicated to **wall/corridor detection**, enabling the bot to remain center-aligned by maintaining equal spacing from the arena boundaries.
2. **Two regions (B & D)** are used for detecting **black/white zones**, which assist in identifying turning points, thereby enabling smooth cornering and avoiding collisions.
3. The **fifth region ©** is dedicated to **color line detection**. The first detected color is considered as the reference, and the bot counts laps based on it until the completion of three laps in the Open Challenge round. This ensures consistent movement either in a clockwise or anticlockwise direction, while ignoring the alternate color line.

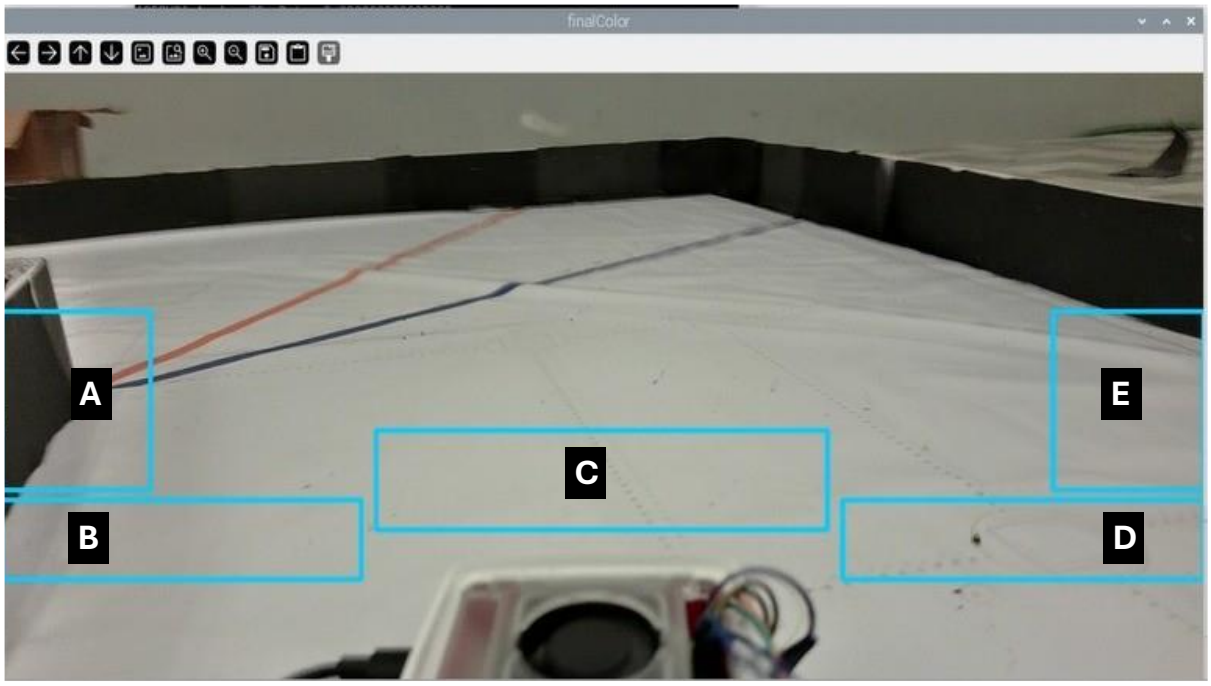


Fig.1: Camera Vision ROI Segmentation for Open Round Challenge

Additionally, a **Raspberry Pi 5 MP camera module** is mounted at the rear side of the chassis, specifically for assisting in **reverse movement and parking operations**.

All sensory inputs from the cameras are processed by the **Raspberry Pi 5 (8 GB)** microcontroller unit. This controller was selected as it provides the necessary computational power, memory, and accuracy to handle real-time image processing and motor control with minimal discrepancies.

Actuation and Motor Selection

The bot is equipped with **two types of motors**, carefully chosen to meet torque, speed, and functionality requirements:

1. **Steering Motor – High Torque Servo Motor (180° rotation):**
    - Used to control the steering mechanism of the bot.
    - Provides precise angular control, ensuring smooth and accurate turning while transitioning between arena sections.
    - High torque capability ensures reliable steering under load and during sharp maneuvers.
2. **Driving Motor – DC Motor:**
    - Mounted to the rear wheels of the vehicle for forward and reverse locomotion.
    - **Anticlockwise rotation** → forward movement.

- **Clockwise rotation** → backward movement (used mainly for parking in designated zones).
- Selected for its balance of speed and torque, suitable for stable linear motion within the arena.

The **integration of motors with the sensory system** ensures that the bot reacts dynamically to the environment, aligning, turning, or reversing as per the detected input.

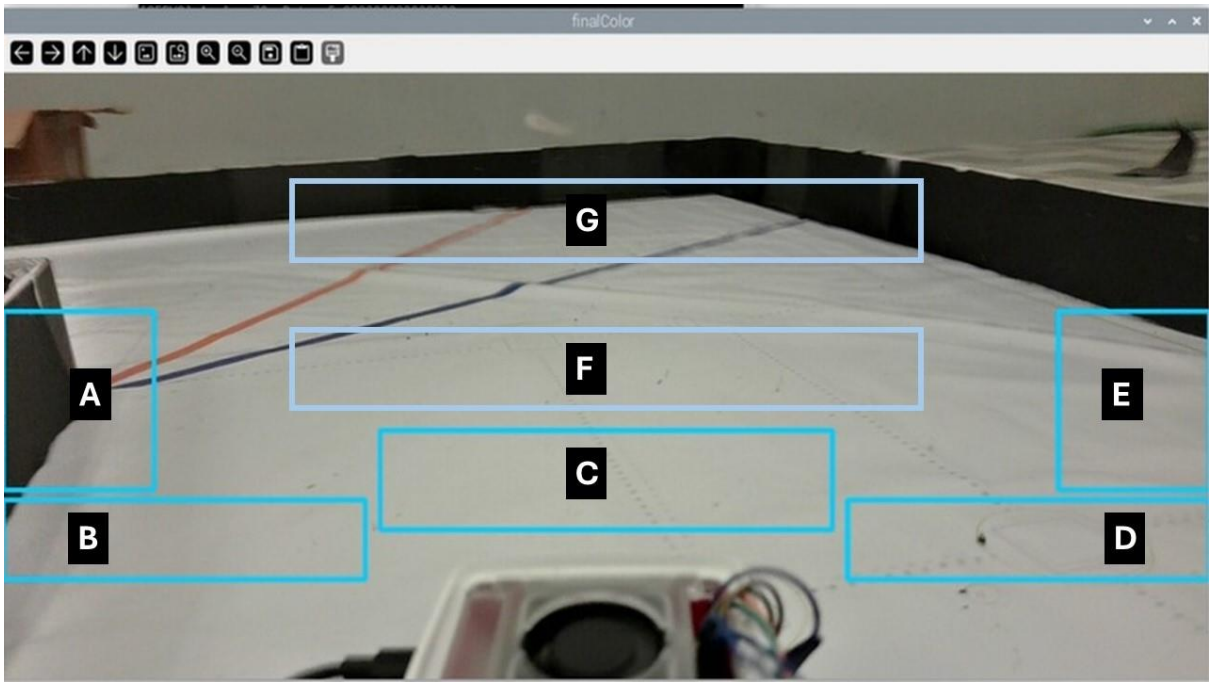


Fig.2: Camera Vision ROI Segmentation for Obstacle Challenge

Chassis Design and Fabrication

The chassis was **custom-developed using 3D CAD software** to precisely meet the dimensional specifications outlined for the competition challenges. The design process emphasized:

- **Compactness and adaptability**, enabling the bot to effectively perform in both Open and Obstacle challenge scenarios.
- **Systematic placement of all components**—including cameras, microcontroller, motors, and wiring—ensuring a clean layout that avoids clutter while maintaining functional interconnections.
- **Iterative refinement through multiple design trials and error corrections**, resulting in a final optimized structure ready for assembly and deployment.

The finalized model, with all major components labelled, is shown in **Fig.3.**

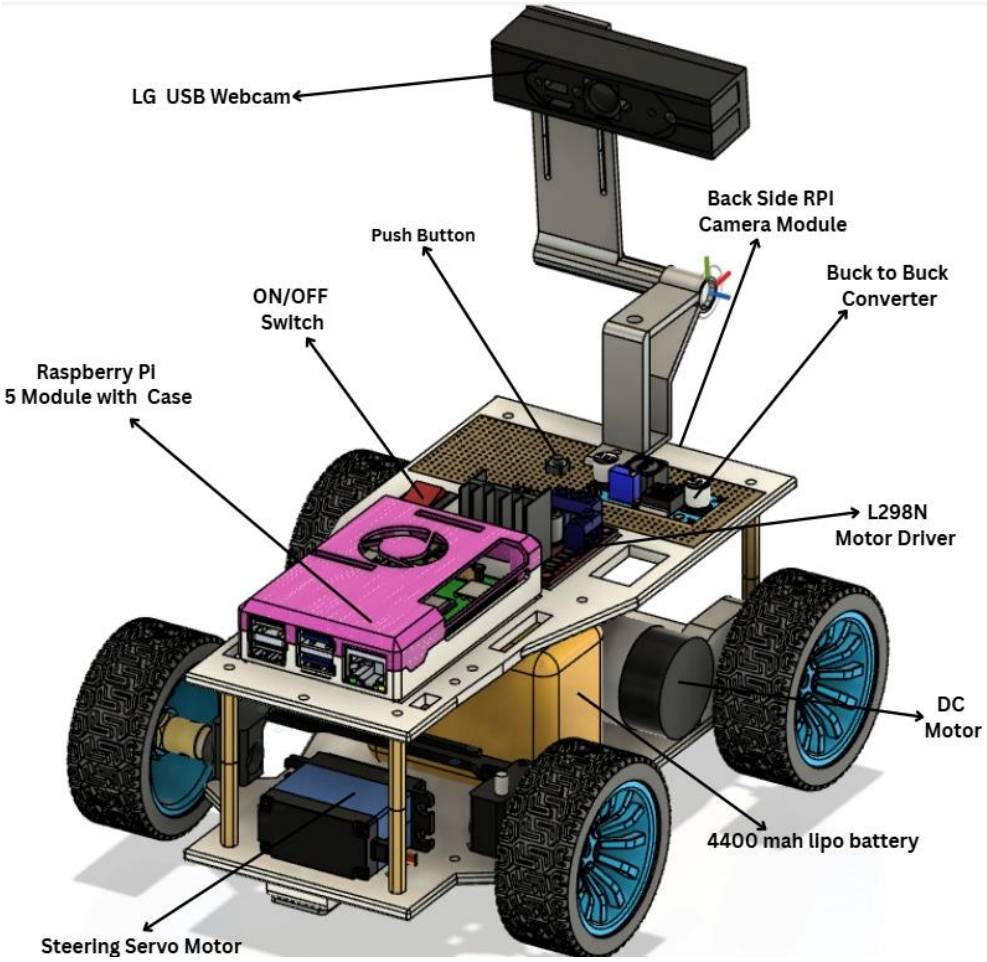


Fig.3: Assembled 3D Design of the Vehicle with Labelled Components

Once finalized, the chassis and additional mounting parts were fabricated using **Fused Deposition Modeling (FDM) 3D printing technology** with **PLA (Polylactic Acid) filament**.

3D Printing Parameters:

- **Scaling:** 100% (exact design measurements used).
- **Infill density:** 70% with **grid pattern**, ensuring strength while minimizing material usage.
- **Support structures:** Enabled where required, especially for the camera holders.
- **Layer thickness:** 0.2 mm.
- **Nozzle temperature:** 215°C.
- **Bed temperature:** 60°C.

This process ensured that the final chassis structure was **rigid, lightweight, and precise**, making it well-suited for mounting and supporting all components effectively.

Following the design phase, the chassis and supporting parts were **3D printed using Fused Deposition Modeling (FDM) technology** with **PLA material**, chosen for its strength, lightweight nature, and ease of fabrication. Once fabricated, the electronic and mechanical components were systematically mounted onto the printed chassis, resulting in the fully functional prototype. The completed assembly is illustrated in **Figure.3.**

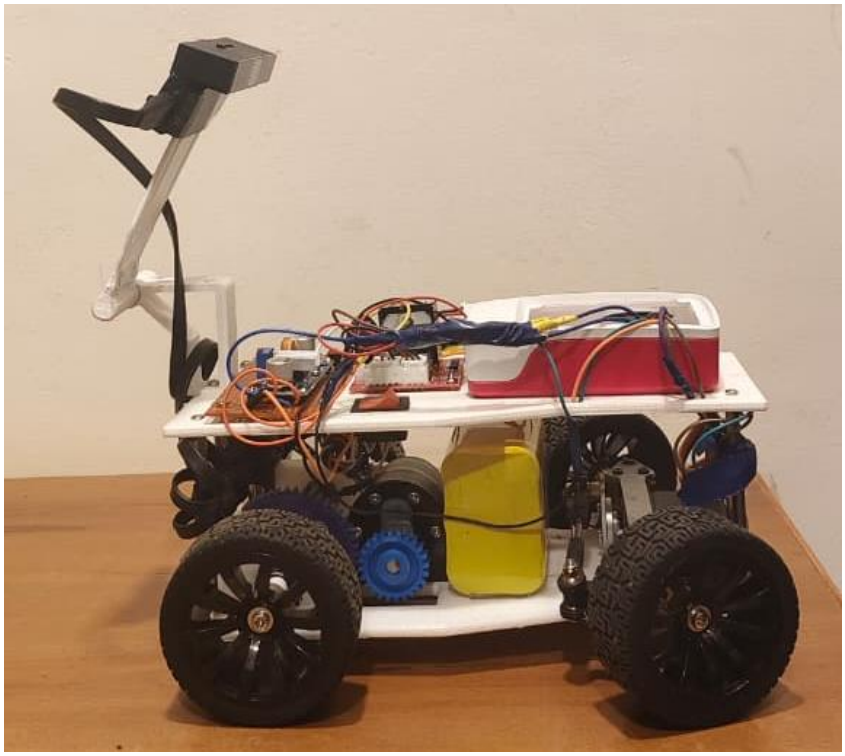


Fig.4: 3D Printed Chassis with Assembled Components

Summary

By combining a well-trained camera vision system, a powerful Raspberry Pi controller, and carefully selected servo and DC motors, the bot achieves efficient mobility management within the arena. The custom 3D-designed chassis, fabricated using optimized FDM printing parameters, further enhances the bot's robustness and functionality, ensuring readiness for the Open Challenge and Obstacle Challenge rounds.

b) Power and Sensing Management

Power Source and Distribution

The robot is powered by a **12V, 4400 mAh Li-ion battery**, chosen for its balance between high energy density and portability. The distribution system is optimized for efficiency and safe operation:

- **12V Line (Direct Supply):**
  - L298N Motor Driver → 12V DC Motor with Encoder.
- **5V Line (Regulated by XL4015 Buck Converter):**
  - Raspberry Pi 5 → Acts as the central controller.
  - Raspberry Pi Camera (5MP Module).
  - LG USB Camera (1080P).
  - Servo Motor.
  - Push Button interface.

The **On/Off Switch** ensures safe startup and shutdown, while the **buck converter** provides stable 5V to sensitive components, preventing overload or voltage dips.

Sensing Strategy and Components

LG PC Camera (1080P, USB Camera)

- **Open Challenge:**
  - Detects blue/orange lines to decide rotation direction.
  - Identifies inner/outer black walls to keep the robot centered.
  - Handles wide/narrow corridor detection through zone-based balancing.
  - Manages turns using forward detection (zones A & E).
  - Counts laps (5, 9, 12 detections) and triggers final stop after 3 laps.

Raspberry Pi Camera (5MP)

- **Obstacle Challenge:**
  - Works like a dash cam during parallel parking.
  - Provides accurate visual feedback for parking alignment.

- Lightweight, directly powered via Pi GPIO.

Novelty and Optimization

- Only **two cameras** handle all challenges, instead of using multiple ultrasonic or IR sensors.
- This reduces **power consumption, wiring complexity, and weight**, allowing faster response and smoother navigation.
- Optimized **camera training and image processing** ensure smooth execution under arena randomness.

Power Consumption Management

- **Motors:** Draw the largest current (peaks at sharp turns).
- **Raspberry Pi + Cameras:** Draw stable 5V regulated load (~1.5–2A).
- **Servo Motor:** Low intermittent load, activated during parking.
- **Total System Runtime:** The 4400 mAh battery ensures multiple full trial runs before recharge.

Professional Documentation (Wiring & BOM)

The wiring follows **professional standards**, with clear separation of 12V and 5V lines:

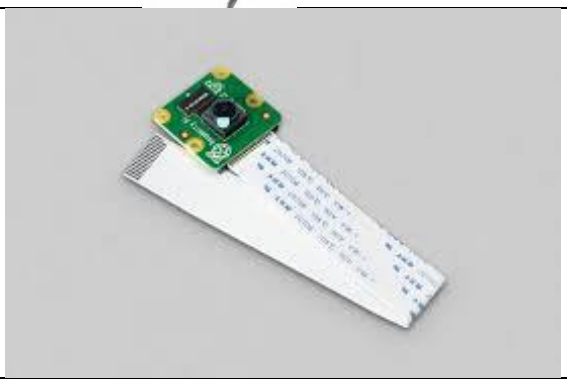
- **12V → L298N → Motors.**
- **12V → Buck Converter → 5V → Raspberry Pi → Cameras, Servo, Button.**
- Switch + PCB for safe operation and connections.

A detailed **Bill of Materials (BOM)** lists each component with ratings, ensuring compliance with WRO evaluation standards.

Bill Of Materials (BOM) — summary table

Item (ref)	Qty	Nominal Voltage	Typical Current (A)	Typical Power (W)	Purpose / Notes
12V 4400 mAh battery	1	12 V	—	52.8 Wh (capacity)	Main energy source (12V, 4.4 Ah → 52.8 Wh)
ON / OFF switch	1	12 V	—	—	Main battery switch
XL4015 step-down (12→5V)	1	12 V in → 5 V out	up to 3 A (set)	up to 15 W out	Supplies Raspberry Pi and 5V peripherals (set current limit safely)
Raspberry Pi 5 (controller)	1	5 V	**1.5** (typical)	**7.5**	Main compute & vision processing — fed from 5V converter
LG USB Camera (1080p)	1	5 V	0.20	1.0	Primary vision for open round and Obstacle challenge
Raspberry Pi Camera (5MP)	1	5 V	0.20	1.0	Parking / close-range vision
Servo motor (steering/parking)	1	5 V	0.50 (avg)	2.5	Actuator for parking/steering adjustments
L298N motor driver	1	12 V (motors), 5 V logic	0.1 (logic) + motor pass-through	1.2 (logic) + motor power	Motor driver — motors draw directly from 12V
12V DC motor with encoder	1	12 V	**2.0** each (avg running)	**24** each → **48** total	Drive motors — **largest power consumers** (stall >> average)
Push button	1	5 V	0.01	0.05	Start/stop or mode switch

Sl. No.	Component Name	Image	Approximate Cost (₹)
1.	Raspberry Pi 5		6,000
2.	L298D Motor Driver		90
3.	DC Motor with Encoder		700
4.	High Torque Servo Motor		350

5.	Lithium-Ion Battery		950
6.	Buck-to-Buck Converter		160
7.	Red On/Off Switch		50 - 100
8.	Push Buttons		10 - 30 per button
9.	Raspberry pi Camera		350

10	LG USB Camera 1080P		1300

Block Diagram:

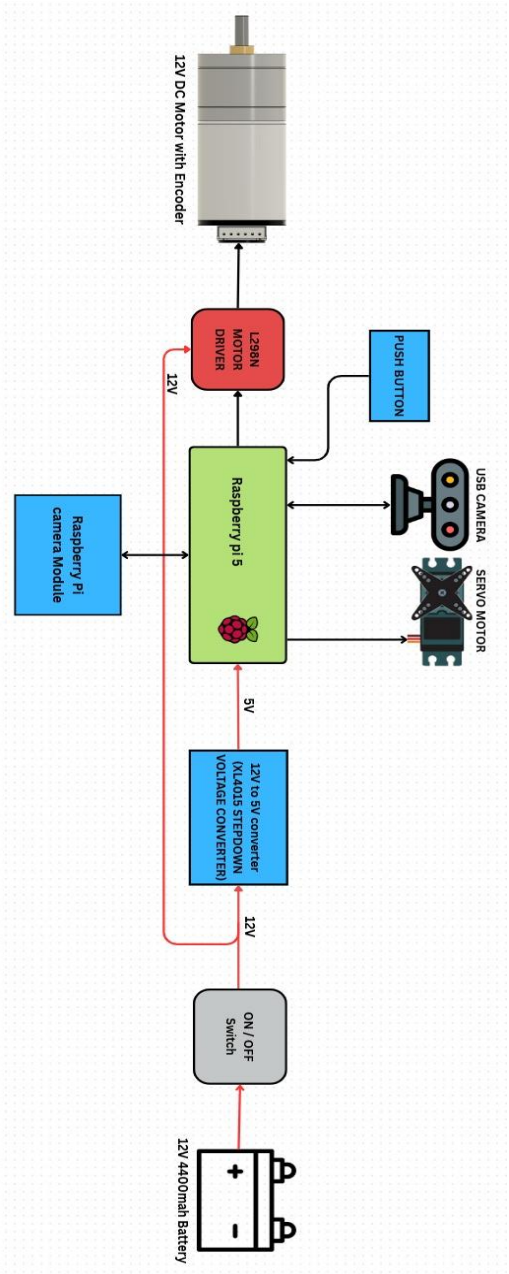


Fig. 5: Block Diagram of Autonomous Vehicle

c) Obstacle Management

As part of our preparation for the **Obstacle Challenge**, we have trained our bot to perform in a similar manner to the Open Challenge round, with the additional requirement of navigating through dynamic obstacles. The primary objective remains the same: the bot must complete **three laps without colliding with the arena walls**.

Initialization and Parking Lot Exit

According to the official rules and guidelines, the **bot initialization section** is determined through a toss. Whichever section is selected, the corresponding **parking lot** will be positioned, and the bot must initialize from there.

Our bot has been specifically trained to **exit the parking lot autonomously**. Considering the bot's length of **260 mm**, the parking lot is defined as 1.5 × bot length = 390 mm. With the bot placed centrally, this leaves **65 mm clearance in the front and rear**. Maneuvering out of this confined space is a key technical challenge.

To achieve this, our bot is equipped with **two cameras**:

- **Top Camera (LG PC 1080p)** – performs multi-task detection, including wall avoidance, line following, and direction control.
- **Rear Camera (Raspberry Pi 5MP module)** – detects the **pink-colored parking lot** as per guidelines.

During initialization, the rear camera confirms the parking zone by detecting pink, while the top camera activates its **five defined vision segments (A–E) as shown in Figure.1:**

- **A & E** – detect white lines for turns.
- **C** – detects arena color lines to determine clockwise or counterclockwise movement and track lap progression.
- **B & D** – detect black (walls) to maintain a safe central alignment and prevent collisions.

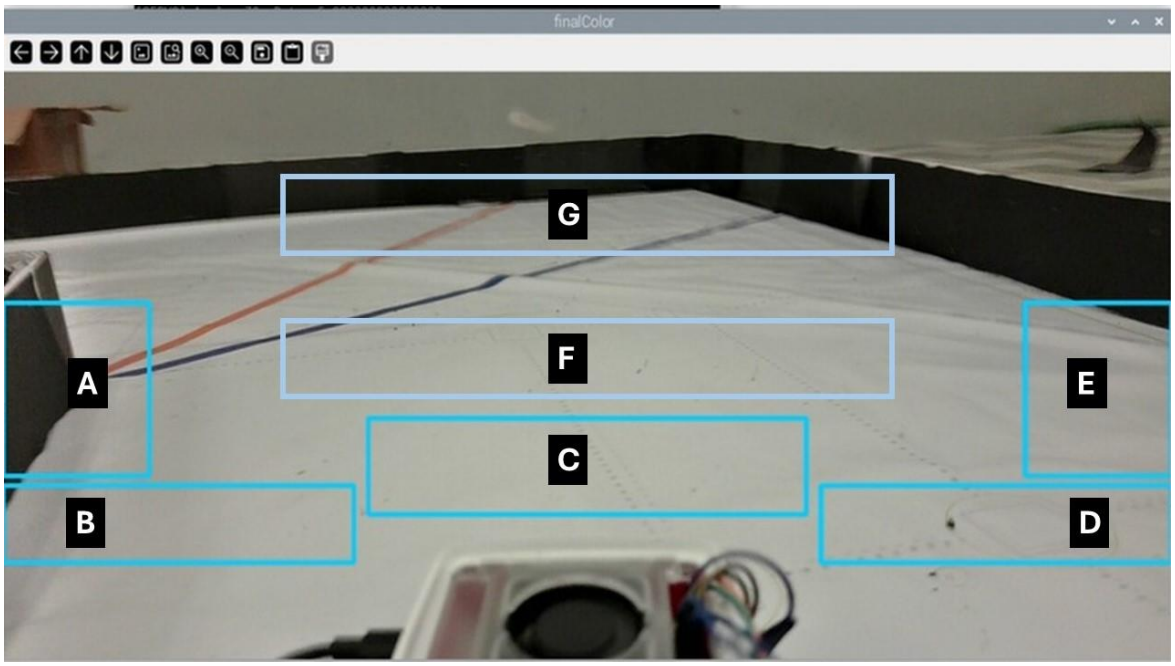


Figure.1: Camera Vision Segmentation Zones (A–G) for Obstacle Challenge

For exiting the parking lot, once **A or E detects white**, the bot first reverses **40 mm** to create clearance. If **E detects white**, the bot executes a **30° controlled turn** toward that direction and exits the parking lot smoothly, minimizing collision risk.

Obstacle Handling and Traffic Signal Detection

After initialization, **obstacles are introduced** based on cards drawn (36 possibilities), which may include **traffic signals (red and green)**. To handle these, our bot's vision system defines **two additional detection zones (F & G)** above the C segment.

- **F Zone**: Primary detection of traffic signals.
- **G Zone**: Secondary detection, used in critical cases (e.g., signals placed immediately after turns).

Traffic Signal Logic:

- **Green Signal (F detected):** Bot makes a **slight left deviation**, then continues forward, aligning with the next available white line in its vision.
- **Red Signal (F detected):** Bot executes a **slight right deviation**, overriding the obstacle while maintaining track orientation.
- **Critical Turns (G detected first):** If a signal is detected immediately after a turn, the **G zone** takes priority. The bot immediately responds (left for green, right for red) before stabilizing into its forward path.

This dual-zone detection system ensures **safe navigation, obstacle avoidance, and compliance with challenge rules**.

Lap Counting and Progression

Lap progression is tracked through **line detection in the C zone**.

- **4 lines = 1 Lap**
- **8 lines = 2 Laps**
- **12 lines = 3 Laps**

This ensures precise monitoring of laps completed.

Final Task – Re-parking the Bot

After completing the **12th line detection (end of Lap 3)**, the bot must **return to its original parking lot**.

- The bot advances for ~**4 seconds**.
- The top camera detects **wall (black) and white alignment markers**.
- On detecting black, the bot makes a **30° corrective turn** toward the parking side.
- Using fine adjustments, the rear camera confirms the **pink-colored parking lot**, and the bot reverses into its **initial position**, completing the challenge.

d) Engineering factor

Front View

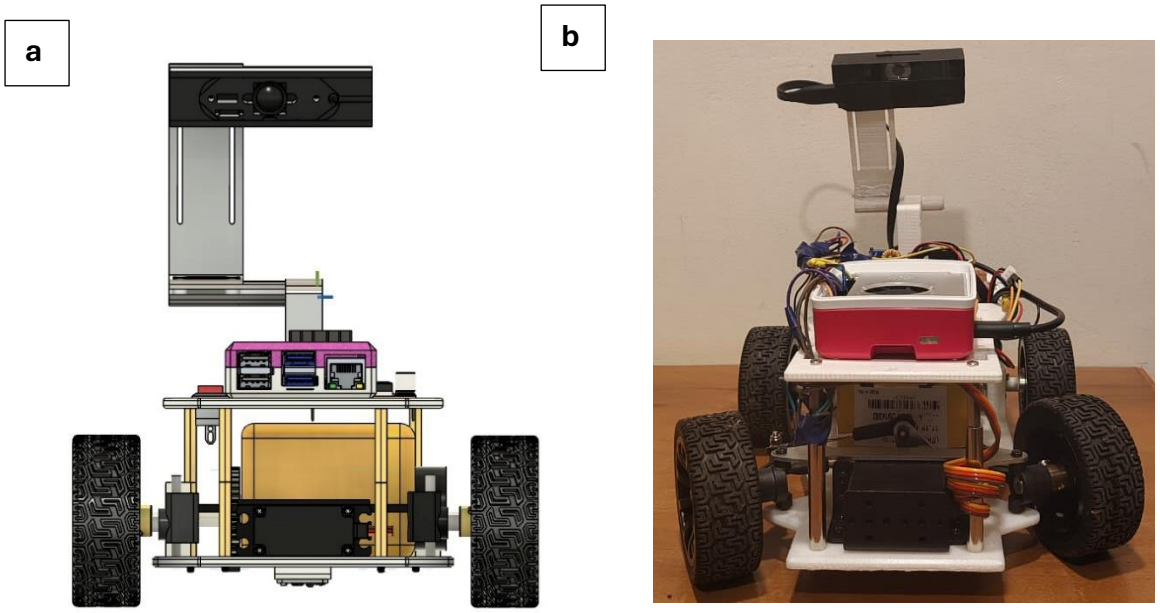


Fig. 6: Front view of the autonomous vehicle – (a) CAD design, (b) completed model.

Back View

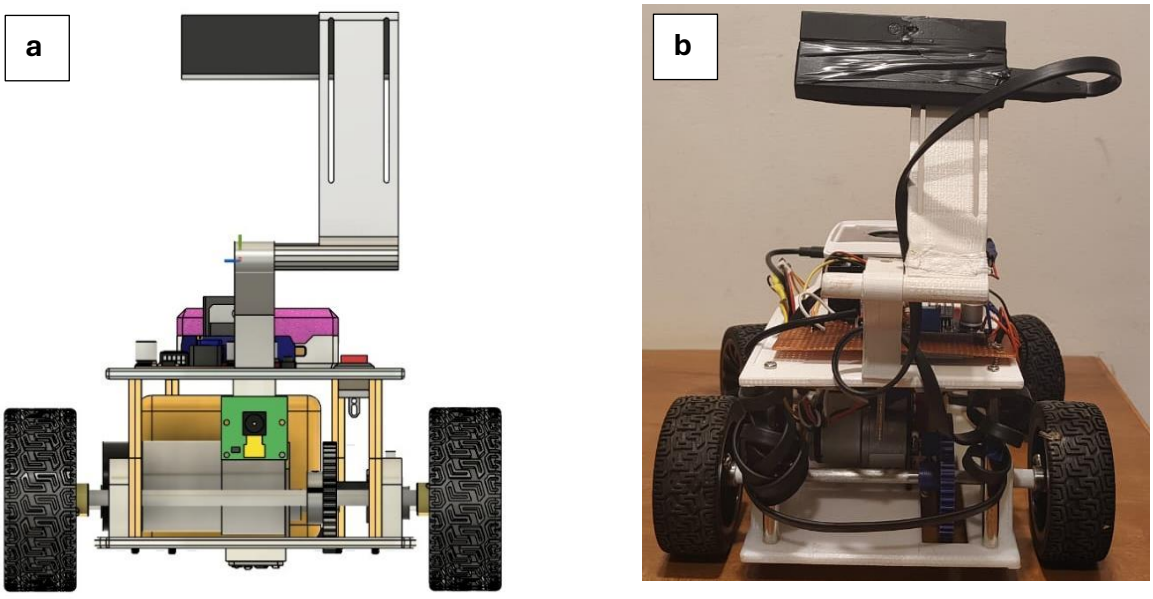


Fig. 7: Back view of the autonomous vehicle – (a) CAD design, (b) completed model.

Isometric view

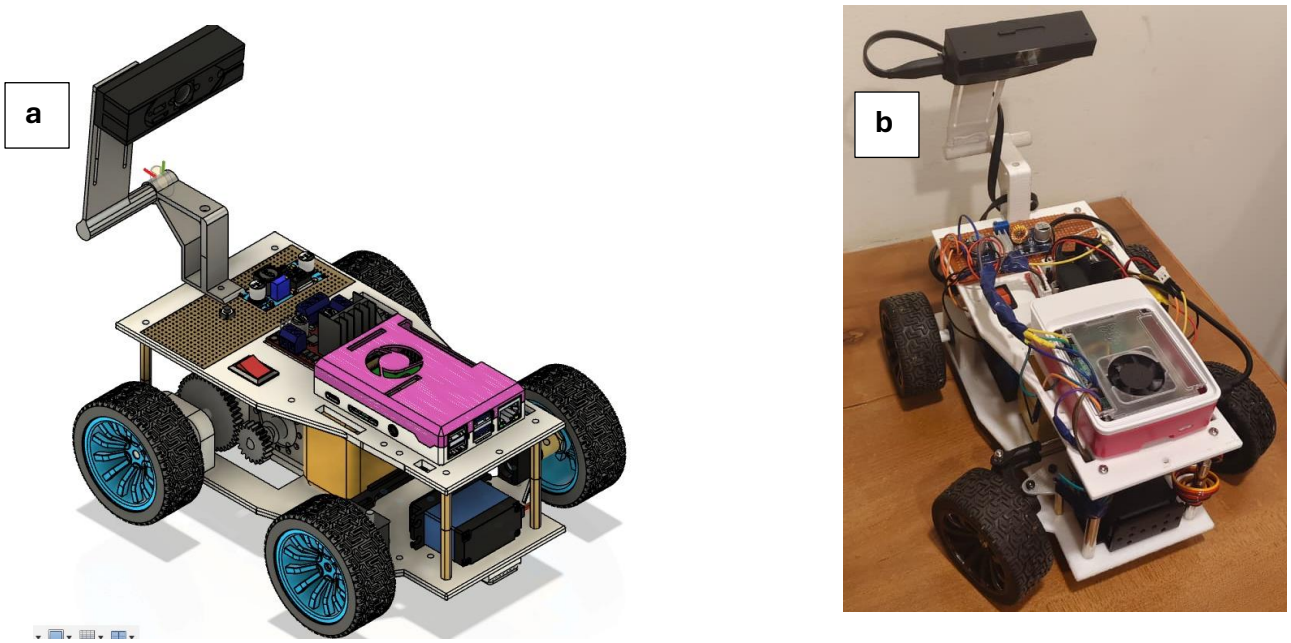


Fig. 8: Isometric view of the autonomous vehicle – (a) CAD design, (b) completed model.

Left Side View

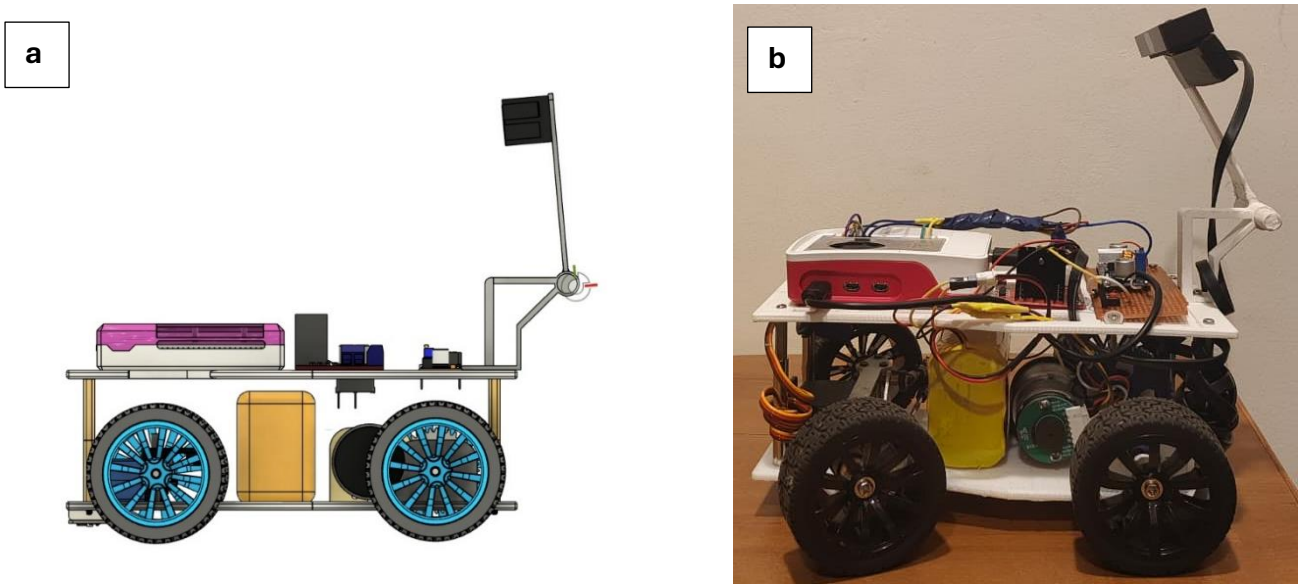


Fig. 9: Left ide view of the autonomous vehicle – (a) CAD design, (b) completed model.

Right Side View

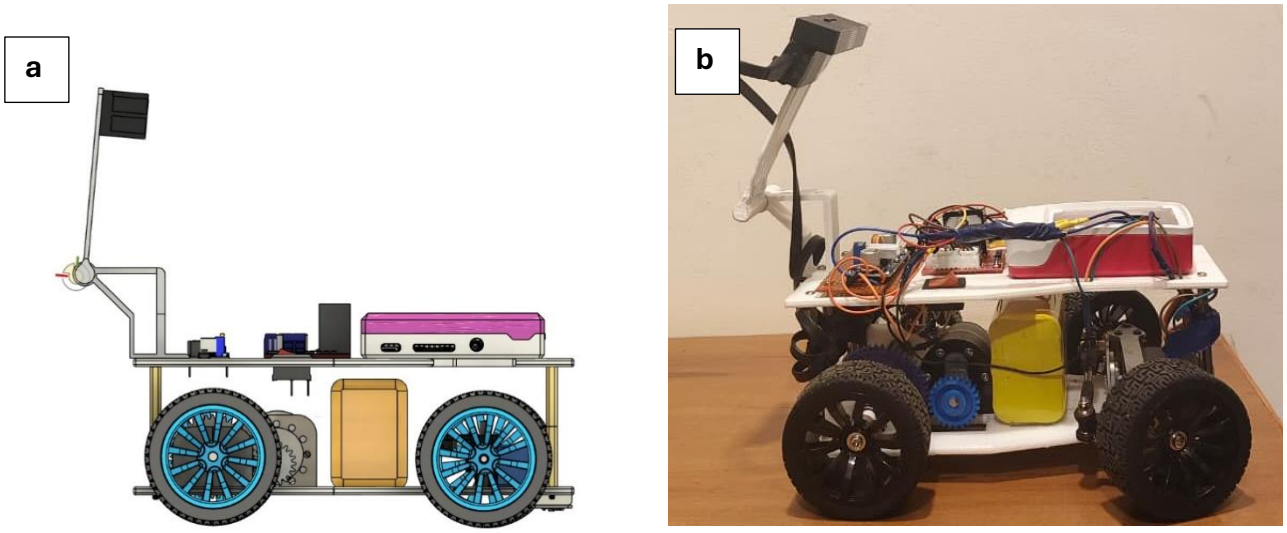


Fig. 9: Right view of the autonomous vehicle – (a) CAD design, (b) completed model.

Top View

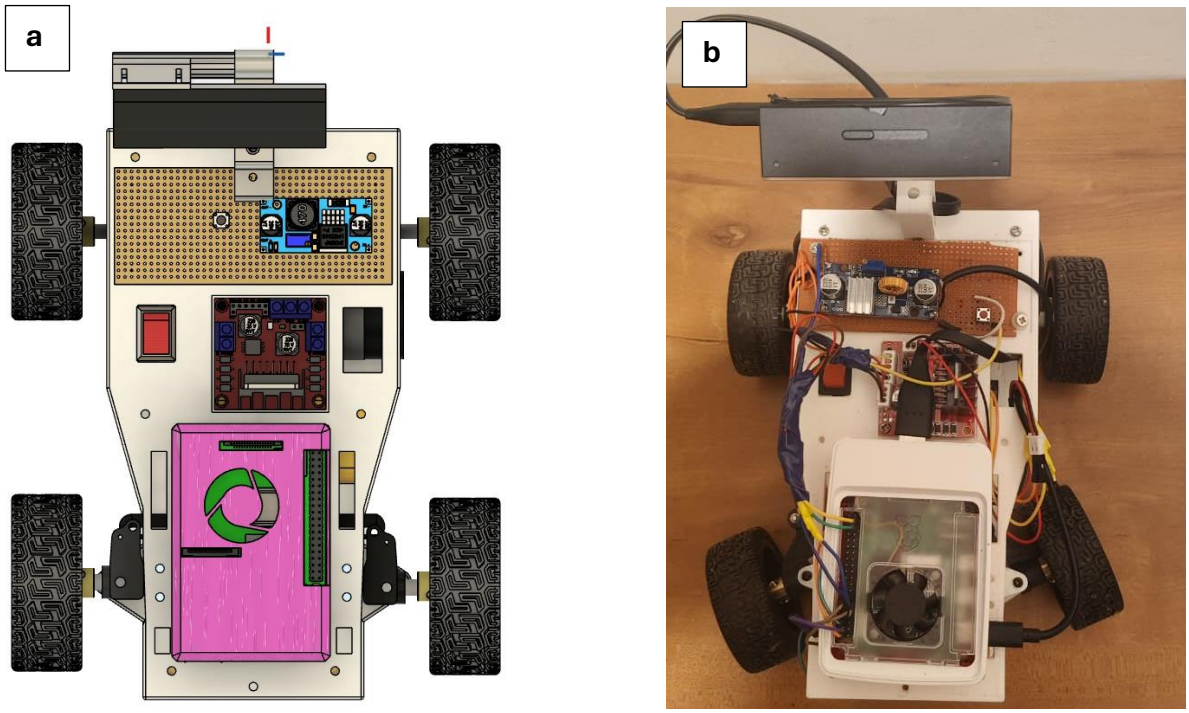


Fig. 10: Top view of the autonomous vehicle – (a) CAD design, (b) completed model.

Bottom View

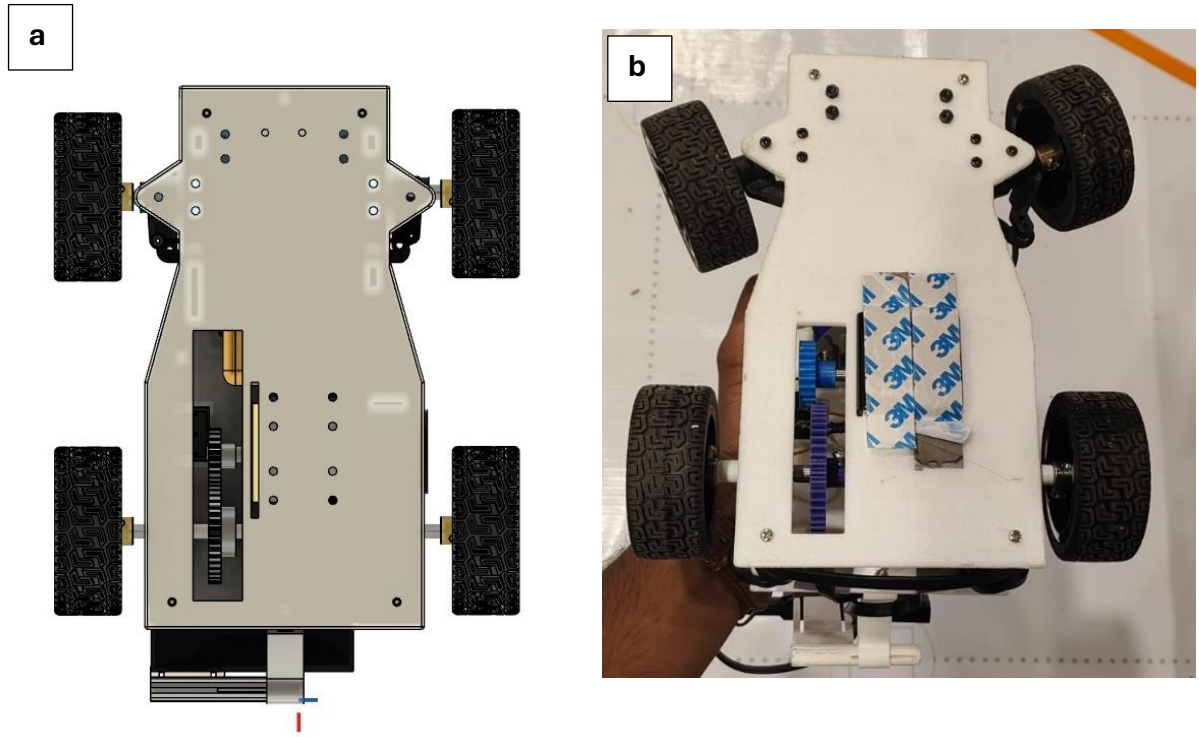


Fig. 11: Bottom view of the autonomous vehicle – (a) CAD design, (b) completed model.

Parts Dimensions:

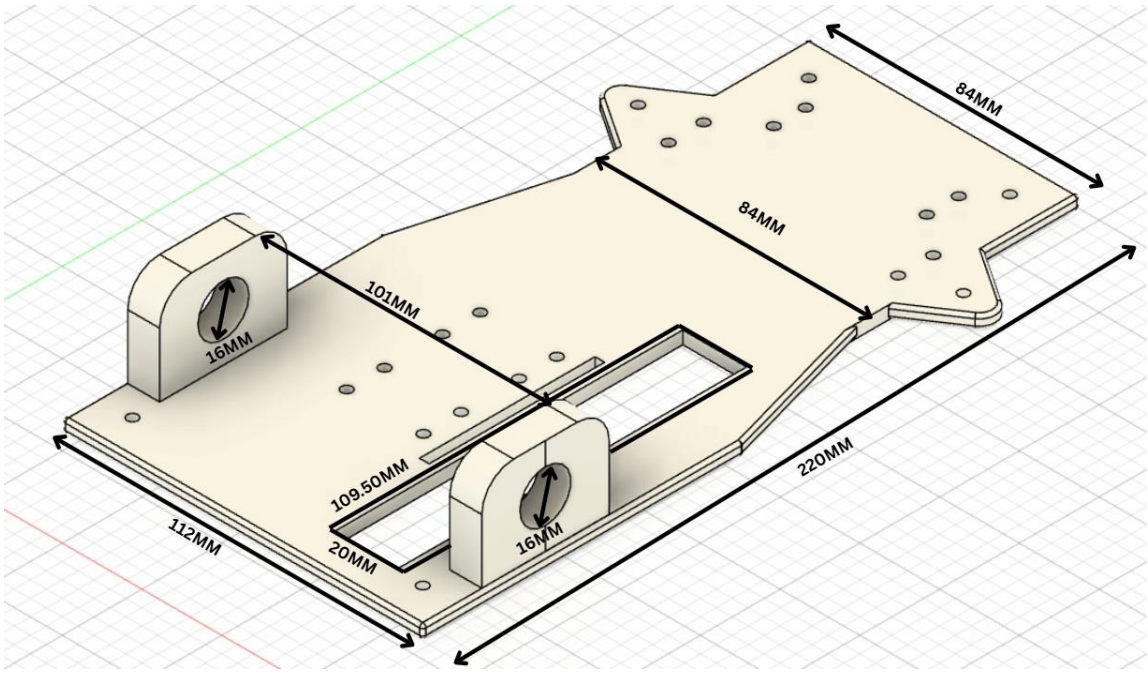


Fig.12: 3D CAD design of the lower chassis with dimensions

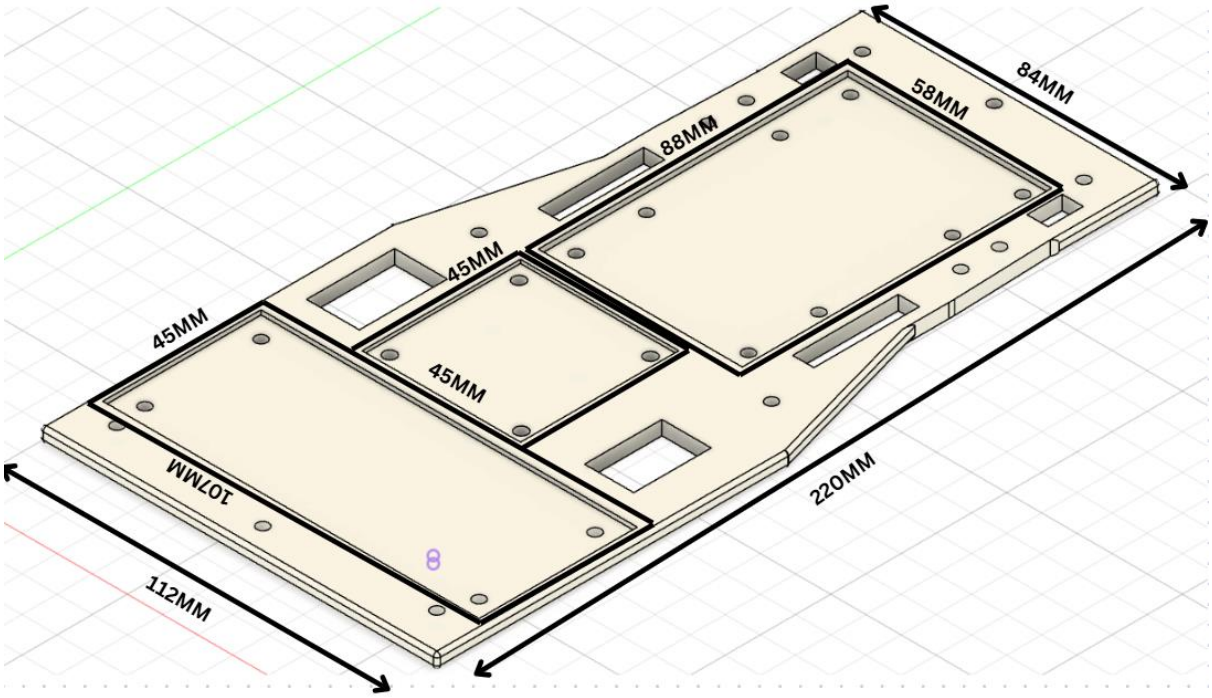


Fig.13: 3D CAD design of the upper chassis with dimensions

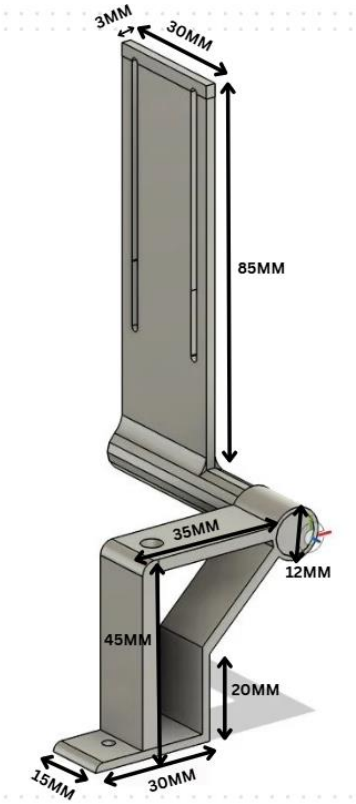


Fig.14: 3D CAD design of the Camera Holder with dimensions

Teamwork and Contributions

Preparing for WRO Future Engineers was one of the busiest yet most rewarding experiences we've had as a team. With classes, exams, and project deadlines running in parallel, we had to manage our time carefully. The last month especially turned into a "day-and-night" sprint, where robotics became a regular part of our routine. Even though it was hectic, it was also one of the best parts of our preparation.

We worked in different ways depending on our schedules — sometimes late at night, sometimes squeezing in sessions between classes, and sometimes setting up in the lab or arena and just staying there until we figured out the next problem.

Each of us had our own main responsibilities, but there was always overlap and teamwork at every step:

- Nanditha supported the team with software programming and helped execute many of the strategic plans. She also took charge of documentation, with technical writing, recording the team's progress, and planning the next steps. This made it easier for all of us to stay on track and keep improving.
- Sudhir took the lead on hardware, wiring, and connections. He was also responsible for choosing and purchasing components, which often meant making trips to buy parts, and for setting up the arena for testing. His logical approach helped in identifying practical solutions when things didn't go as planned.
- Harshavardhan was deeply involved in coding. You could often find him sitting in a corner with his laptop, writing and refining code whenever he had free time. He worked closely with Sudhir to test different strategies and to translate the mentor's ideas and plans into functioning code on the robot.

We often discussed strategies together, got feedback from our mentor, and then divided the work so progress could continue even if we weren't physically in the same place. Communication was constant — whether it was sharing updates, debugging issues, or just brainstorming ideas.

Even though it was tiring at times, the process taught us how to balance academics with team projects, how to divide tasks based on strengths, and how to stay motivated under pressure.

More than anything, it showed us the value of working as a team, where everyone contributes not just in their own area but also supports each other whenever needed.

Looking back, we wouldn’t trade this experience for anything. It was busy, yes, but also full of learning, fun, and teamwork — and that’s what made it special.

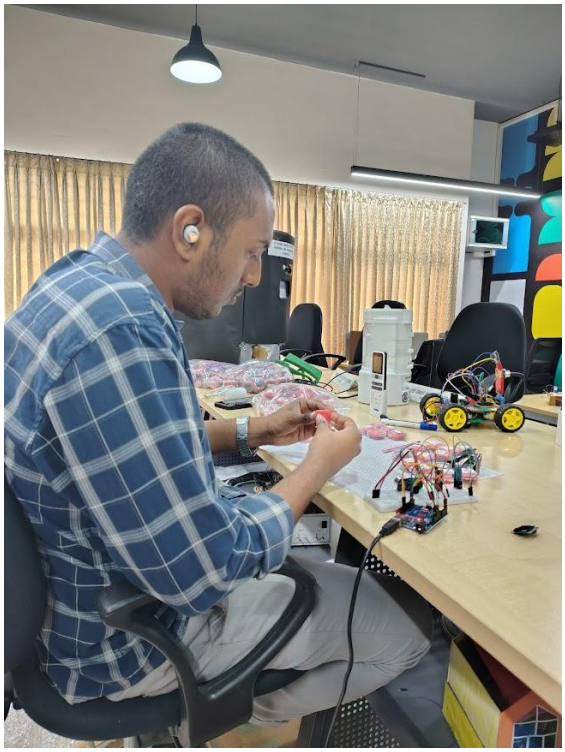


Figure 14: Teamwork During the Robot Development Process

Conclusion

This journey has been one of the most memorable experiences for our team. From long nights of coding and testing, to redesigning parts that didn't work the first time, to finally seeing our robot complete laps on the track — every moment has been both challenging and rewarding. Balancing academics, exams, and project work was never easy, but it made every small success feel even more meaningful.

What excites us the most is not just what we've achieved so far, but what lies ahead. We have developed new strategies, explored novel ideas, and refined our robot step by step. Now, we are ready to bring all of this to the event, execute our plans with confidence, and give our best performance. For us, participating in WRO Future Engineers 2025 is not just about racing, but about learning, innovating, and sharing our passion for robotics on a global stage.

We are proud of how far we've come, and even more excited to take the next step — competing with teams from around the world and experiencing the thrill of the championship.

Acknowledgment

We sincerely thank Alliance University for providing us with space, resources, and access to Makerspace. Our heartfelt thanks to our mentor Dr. Harinath Aireddy for his constant guidance, and to our friends and families for their encouragement throughout this journey.