**ENGINEERS**

WRO
Future
Engineer2025

NOVA Engineers

Documentation

Abstract:

Abstract Objective: This document outlines our strategy for the WRO Future Engineer 2025 competition, focusing on the design and program development of a vehicle capable of navigating obstacles and handling diverse challenges. Key improvements include advanced motion control, energy management, and adaptable programming, enabling responsive and efficient performance.

Methods:

To help the robot move accurately, we used advanced motion control and custom programming. We also added energy-saving methods to use power more efficiently. Regular checks are done to make sure the vehicle is ready to work. For better decision-making, the robot uses data from sensors and live camera analysis to quickly adapt to different situations.

Key Engineering Factors:

This project shows important ways to help the robot move around obstacles. It uses smart design and flexible coding so the robot can quickly react to challenges during the competition. The code and algorithms were made to help the robot work well and stay reliable in different situations.

Conclusion:

This document reflects our comprehensive engineering and programming approaches, demonstrating a commitment to excellence in the WRO Future Engineer competition through innovative design and real-time adaptability.

Content:

CONTENT	Number
About us	(1
Work Schedule	(2
Project Estimated Cost	(3
Components of Our Robot	(4
The Diagrams	(5
Robot Photos	(6
Explanation of the rounds	(7
Flowcharts	(8
Open Challenge Code	(9
Obstacle Challenge Code	(10
Applications Used	(11

Documentation Video (YouTube Link)

This video is made with great care and dedication. It compiles all the information we included in the document and presents it in words, follow this link to view the video we created:

https://youtu.be/C_BoXmBWQ4o?si=mJF7Yn2e1AimlPT-

1) About Us:

The name "NOVA" represents a new star that suddenly shines with renewed energy — symbolizing innovation, brilliance, and growth. It reflects our team's spirit of creativity and ambition, as we strive to illuminate the field of engineering and technology with fresh ideas that drive progress and inspire excellence.

Introduction to our team:

#Raghad Wissam Al-Jaabari

- From Hebron

- 16 years old

- My hobbies are horse riding and badminton

- Responsible for assembling the robot and executing programming tasks, integrating mechanical components with software to achieve the required performance and ensure the system operates efficiently

- ragadaljabari1@gmail.com

#Rasha Islam Al-Fakhouri

- From Hebron
- 15 years old
- I love reading literature and philosophy, and I enjoy playing badminton Responsible for technical and organizational aspects, participated in presenting changes in project structures, in addition to continuous follow-up between stops to clarify the overall vision
- rashfakh9@gmail.com

#Layan Yousri Amro

- From Hebron
- 14 years old
- my hobbies are sports and badminton
- She is responsible for documenting and writing reports on the project, and works on organizing the content and accurately documenting the work stages and results to present them in a professional and clear manner
- amrolayan95@gmail.com

Our Vision and Goal

Our vision is to develop an intelligent robot capable of performing smart commands independently. We aim to build a system that can analyze its surroundings, make decisions, and complete tasks efficiently without direct human control. This project represents our first step toward creating innovative solutions that support our community and inspire future technological advancements Through this competition, we have gained valuable experience in robotics, programming, and teamwork. We also developed essential skills such as critical thinking, creativity, and planning in the future, we plan to use what we've learned to improve our projects, support younger students, and continue developing technologies that make a positive impact.

2) Work Schedules:

we created a clear and organized plan to show our commitment and teamwork. The plan includes goals and deadlines to help us manage time well, track progress, and handle any unexpected problems. This method helps us stay focused and work efficiently. Our plan acts as a guide that helps us move faster and stay accurate as we work toward success

Nova Team Action Plan	Week 1 (20-26 Sep)	Week 2 (28 Sep-4 Oct)	Week 3 (6-12 Oct)	Week 4 (14-20 Oct)	Week 5 (22-28 oct)	Week 6 (30 Oct-5 Nov)	Week 7 (7-13 Nov)
Saturday	• Discuss objectives and assign roles. • Brainstorm ideas • Creat project timeline.	• Start 3D printing robot body. • Print robot structure and assemble prototype. • Test printed components.	• Writing code for 3 spins and 90-degree rotation. • Continuously testing this code.	• Change the processor. • Change the structure of the robot.	• Refined the obstacle course program and adjusted robot timing.	• Review and fix code errors identified during testing.	• Optimize robot speed and obstacle response. • Run multiple trail rounds to verify improvements
Monday	• Select and test robotic components. • Install motors and wiring.	• Making a robot installation video.	• Making a wire diagram and circuit diagram.	• Take team and robot photos.	• Commit diagrams, flowcharts, team and robot photos, and others on GitHub.	• Record an open challenge, and the obstacle challenge videos	• Simulate final competition rounds. • Conduct stress tests to ensure robot durability.
Tuesday	• Creat robot design sketches. • Begin robot chassis design.	• Test partial robot functions.	• Draw flowchart	• Record final demonstration video.	• Finish and add the report explaining the robot's electronic parts, and its mechanism of operation.	• Edit final demonstration video.	• Perform full system testing for sensors, motors, and controls. • Calibrate sensors for accurate data collection.
Thursday	• Research control systems. • Start writing a report to explain the robot parts.	• Program movement algorithm.	• Review team progress.	• Start writing obstacle avoidance code.	• Design 3D models and commit them on GitHub.	• Finalize report layout and ensure all sections are complete in GitHub.	• Conduct final review meeting. • Finalize the codes.

Figure 1: Work Schedule type1

| Work	Start day	End day	Duration	Assignee
Discuss objectives and assign roles.	20/9/25	21/9/25	2	Layan Rasha Raghad
Select and test robotic components.	22/9/25	24/9/25	3	Layan Raghad
Creat robot design sketches. Begin	23/9/25	24/9/25	2	Rasha
Research control system. start	24/9/25	25/9/25	2	Layan
Start 3D printing robot body	25/9/25	27/9/25	3	Rasha Raghad
Making a robot installation video	28/9/25	29/9/25	2	Layan Rasha Raghad
Test partial robot functions	30/9/25	30/9/25	1	Layan Rasha Raghad
Program movement algorithm.	1/10/25	3/10/25	3	Layan Raghad
Writing code for 3 spins and 90-	3/10/25	13/10/25	11	Raghad
Making a wire diagram and circuit	1/10/25	2/10/25	2	Rasha
Draw flowchart	3/10/25	4/10/25	2	Rasha
Review team progress	5/10/25	5/10/25	1	Layan Rasha Raghad
Change the processor. Change the	6/10/25	8/10/25	3	Layan Raghad
Take team and robot photos	9/10/25	10/10/25	2	Rasha Raghad
Record final demonstration video	10/10/25	12/10/25	3	Layan Rasha Raghad
Start writing obstacle avoidance	13/10/25	15/10/25	3	Layan Rasha Raghad
Refined the obstacle course	16/10/25	25/10/25	10	Raghad
Commit diagrams, flowchart, team	13/10/25	13/10/25	1	Layan Rasha
Finish and add the report explaining	14/10/25	15/10/25	2	Layan
Design 3D models. Commit them on	16/10/25	17/10/25	2	Layan Rasha
Review and fix code errors identified	18/10/25	30/10/25	13	Raghad
Record an open challenge, and the	31/10/25	31/10/25	1	Layan Rasha Raghad
Edit final demonstration video.	1/11/25	4/11/25	4	Rasha Raghad
Finalize report layout and ensure all	5/11/25	6/11/25	2	Layan Rasha
Optimize robot speed and obstacle	7/11/25	8/11/25	2	Raghad
Simulate final competition rounds.	9/11/25	11/11/25	3	Rasha Raghad
Perform full system testing for	12/11/25	12/11/25	1	Layan Rasha Raghad
Conduct final review meeting.	13/11/25	13/11/25	1	Layan Rasha Raghad

Figure 2: Work Schedule type2

"We're a team; we work as a whole, think as though we have a hundred heads, and know each other as if we've been together for years. We've come together, gathered the hands of each member, and moved forward with one, united mind. "

3) Project Estimated Cost:

Hardware Components Estimated cost:

	component	Estimated cost
1	**ASRC-CM-DIY Version**	$142
2	**Raspberry Pi 5 Model B**	$35
3	**Motor Driver L298N**	5$
4	**Motor DC**	20$
5	**Servo Motor**	6$
6	**voltage regulator XL4015**	$7
7	**gyroscope MPU6050**	$15
8	**Four Ultrasonics**	$40
9	**PC I7**	$1000
10	**smart cars motors GA25 370**	$100
	Title	$1370

4) Components of Our Robot:

Our robot is built using several key components that help it perform tasks efficiently during the WRO competition. It includes a **Raspberry Pi** as the main controller, which runs the Python code and connects to various sensors. We use a **distance sensor** to detect obstacles, a **gyroscope** to measure rotation and direction, and a **USB camera** for visual input and image processing. The robot also has an **L298N motor driver** to control the movement of the wheels. All these parts work together to help the robot navigate, make decisions, and respond to challenges in real time.

4) 1) Software:

4) 1.1) Python Language:

We chose Python as the main programming language for our WRO robot because it is simple, powerful, and widely used in robotics and computer vision. Python allows us to write clear and organized code, making it easier to test, debug, and improve our program One of the main reasons we selected Python is its strong support for the OpenCV library, which we used to process images and recognize objects or patterns. OpenCV gave us the ability to detect lines, colors, and QR codes — features that were essential for our robot's mission. Python also works perfectly with Raspberry Pi, which we used as the main controller. Its flexibility allowed us to easily connect sensors and motors, and integrate the results from OpenCV to make real-time decisions.

Using Python helped us focus on problem-solving and teamwork, since the language is easy to understand and modify. Overall, Python and OpenCV together provided a powerful combination that made our robot more intelligent and efficient during the WRO challenge.

Python is used in our project because it is easy to learn and helps us control the robot accurately. With Python, we can program the robot's movements, read sensor data, and analyze images from the camera. It also supports powerful libraries like OpenCV, NumPy, and more, which help us complete tasks quickly and efficiently during the competition challenges.

4.1.2) The Libraries:

RPi.GPIO Library:

The RPi.GPIO library is specifically designed to interact directly with the physical world through the general-purpose input/output pins on the Raspberry Pi. Imported here as GPIO, its primary function is to manage the flow of digital signals. You use it to set a pin as an output to send a High (1) or Low (0) signal, which controls devices like LEDs, motors, and actuators (turning them ON or OFF). Conversely, you can set a pin as an input to read the state of physical components, such as checking whether a button has been pressed or if a sensor has detected an object. This library is the foundational layer for low-level hardware control.

OpenCV Library:

OpenCV is a powerful and widely used library for computer vision tasks. In our WRO project, we use OpenCV to help the robot understand its surroundings through camera input. It allows the robot to detect colors, follow lines, and recognize objects or obstacles. OpenCV works well with Python and provides many ready-to-use functions that make image processing faster and easier. This helps improve the robot's accuracy and responsiveness during competition challenges.

time Library:

The time library is a standard module built into Python that provides essential functions for managing time within a program. It is crucial for hardware projects, as physical devices often require specific timing to operate correctly. Its most common function is time.sleep(X), which pauses the execution of the program for a specified number of seconds (X). This delay is vital for tasks like creating rhythmic flashing patterns, allowing mechanical components time to move into position, or stabilizing sensor readings before processing them. It also offers functions for measuring time intervals and retrieving the current time.

Smbus Library:

The smbus library serves as a software interface for communicating with external devices using the I²C (Inter-Integrated Circuit) protocol, which is a very common serial communication method in embedded systems. I²C allows multiple devices, such as sophisticated sensors, memory chips, and display drivers, to communicate with the Raspberry Pi using only two wires (SDA and SCL). By importing smbus, Python code gains the ability to address specific external chips and send commands (write operations) or receive complex data (read operations) from them. This is essential for integrating advanced components that require a dedicated communication bus rather than simple ON/OFF signals.

5) The challenges we faced:

Problem:
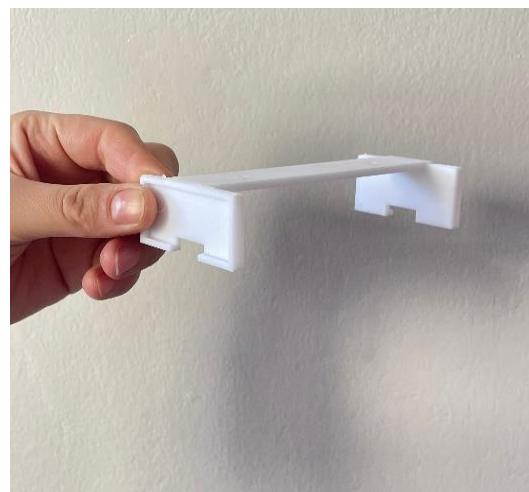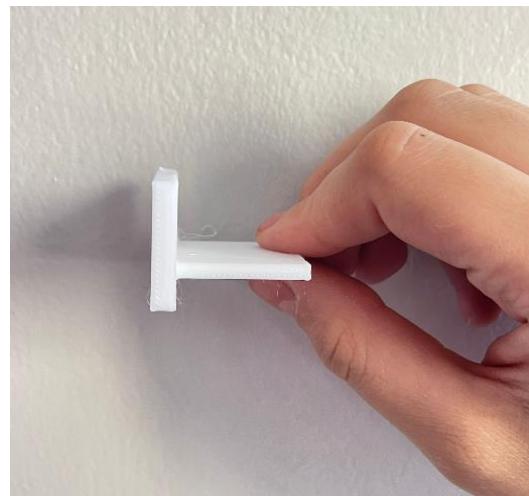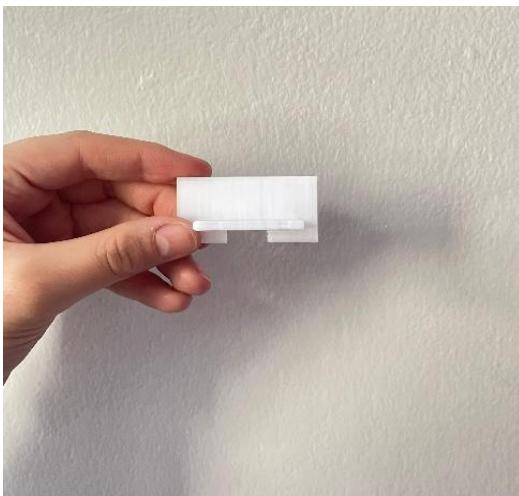
During robot operation, we noticed that the sensors were not stable enough, which led to inaccurate readings and negatively affected task performance.

Solution:

We designed and installed a custom sensor mount using 3D printing, ensuring precise and secure placement of the sensors.

Benefits:

- Improved sensor accuracy.

- Reduced vibrations during robot movement.

- Enhanced structural and professional appearance of the robot

Problem:

During robot testing, we noticed that the Control Device (SD0

was slow, causing delays in command execution and poor sensor responsiveness.

Solution:

We replaced the unit with a new one featuring higher performance and faster processing speed, which significantly improved the robot's responsiveness and task accuracy

Benefits:

- Faster execution of programmed commands.

- Better interaction with sensors and motors.

- Reduced errors caused by time delays.

Old SD   **New SD**

Category	SD1 Old	SD 2 New
Model	SanDisk Ultra microSDXC	SanDiskUltra microSDXC A1
Capacity	64GB	64GB
Speed Class	Class 10	Class 10
UHS Standard	UHS-1	UHS-1
Application Performance Rating	None	A1 (Optimized for running applications
UHS Speed Rating	Not specified	U1(Minimum write speed 10MB/s
Performance with OS (e.g., Raspberry Pi)	Average	High performance optimized for system booting
Boot & App Loading Speed	Slower	Faster and more stable
Recommended Use	Basic file storage	Robotics OS running, Raspberry Pi, real-time tasks
Reason Not Selected	Can cause lag and slow loading	Chosen for stability, speed, and smooth robot performance

Problem:
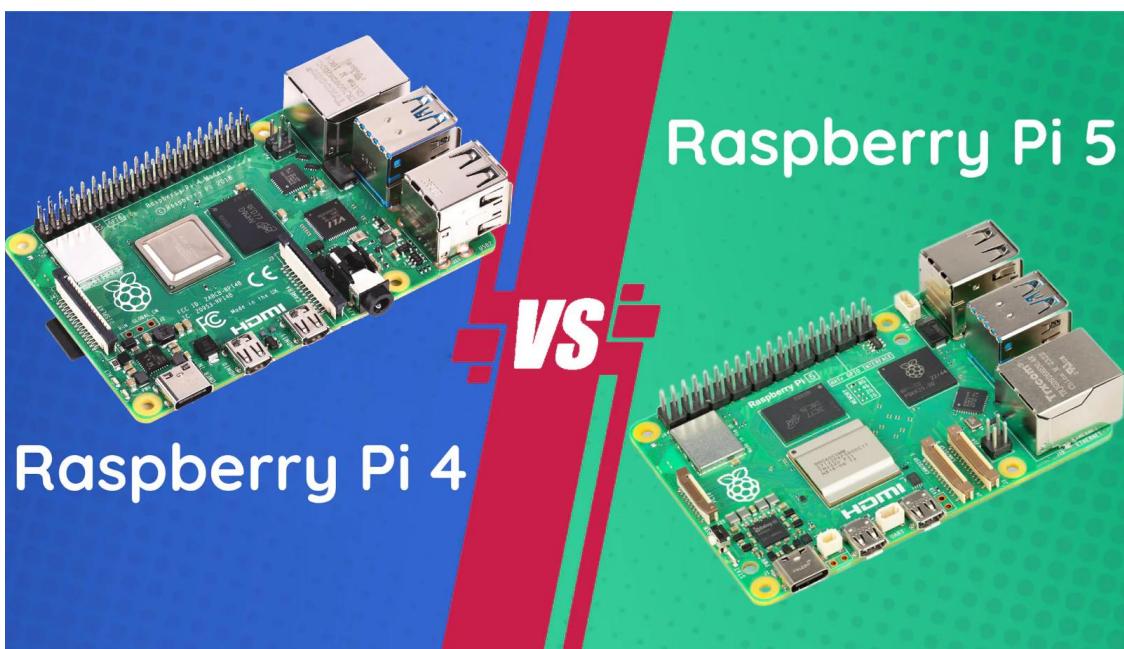
While using the Raspberry Pi 4, we encountered several issues, including slow performance and unexpected system freezes. These problems affected the overall efficiency and reliability of our setup.

Solution:

To overcome these challenges, we upgraded our system to the Raspberry Pi 5, which offers improved processing power, better thermal management, and enhanced stability.

Benefits:

- Faster and more responsive performance

- Reduced system crashes and downtime

- Improved reliability and efficiency for continuous operation

Problem:

At the beginning, we used a camera because it provided good performance and accurate detection. However, after modifying the robot's engineering design and adding the 3D funnel, the camera no longer fit properly. Its position affected the field of view, which reduced its efficiency and response accuracy during movement.

Solution:

We decided to replace the camera with New Camera that better match the updated design and work effectively with the new 3D funnel structure. This allowed us to maintain reliable environmental sensing without compromising the robot's mechanical design or functionality.

Benefits:

With this change, the robot achieved better integration between the mechanical structure and the electronic components. The new Camera setup worked efficiently and provided a clear detection path, improving overall performance, stability, and reducing technical issues caused by unsuitable camera placement.

Old Camera   **New Camera**

Problem:

At the beginning of the project, we planned to use a color sensor to determine directions and guide the robot's movement. However, we faced difficulties with color detection due to lighting changes and surface reflections in the competition environment. This affected the sensor's accuracy and could cause the robot to make incorrect decisions.

Solution:

Instead of using the color sensor, we adopted a more reliable method by using the distance sensor. If the right sensor detects more than 160 cm, this means the robot should move counterclockwise. On the other hand, if the left sensor detects more than 160 cm, this means the robot should move clockwise. This approach allowed the robot to determine its direction accurately without relying on color detection.

Benefits:

This solution provided faster and more accurate direction control, making the robot less affected by lighting conditions or surface changes. It improved movement stability, enhanced performance, and increased reliability, helping the robot complete its path smoothly and efficiently.

Problem:

In the beginning, our robot relied only on the Ultrasonic sensor to determine its direction. However, we noticed that the robot sometimes drifted and did not stay perfectly on the intended path, especially when turning or moving for a long distance. This caused slight deviations that could affect its accuracy during the mission.

Solution:

To solve this issue, we added a gyroscope sensor alongside the distance sensor. The gyroscope helps the robot maintain a stable direction and corrects any drift by constantly monitoring the robot's rotation angle. By combining both sensors, the robot can detect direction accurately and stay aligned with the path.

Benefits:

With the gyroscope added, the robot became more stable and precise in its movements. It no longer drifts easily, and its turns became smoother and more controlled. This upgrade improved the overall performance, reliability, and accuracy of the robot during the competition missions.

Problem:

After installing the camera on the robot, we noticed it wasn't securely fixed, which caused vibrations during movement and reduced image accuracy.

Solution:

We designed a custom camera mount using 3D printing to ensure stable positioning and optimal viewing angle.

Benefits:

- Secure and stable camera installation.

- Improved image quality and visual processing.

- Easy adjustment of the camera angle when needed.

4) 2) Hardware:

4) 2.1) The robot chassis:

Firstly, the structure of the robot

This car is a self-assembled (DIY) model that is built on a metallic chassis and uses an Ackermann Steering system to simulate the motion of real cars.

The components of the car were connected and operated using a Raspberry Pi, programmed in Python to control movement and steering.

This is the chassis:

This car is not a remote-controlled vehicle; rather, it is an experimental platform developed to apply the principles of mechanics, control, and programming in the field of intelligent vehicles.

Thanks to the use of the Ackermann Steering system, the car replicates the actual steering mechanism of real vehicles, unlike the differential steering systems used in many traditional robots.

The car was entirely assembled and programmed manually as part of participation in the

The wheels of this smart robot car have specific material and performance characteristics:

Wheel Rims (Hubs): The rim material is ABS (Acrylonitrile Butadiene Styrene). They are purely aesthetic.

Tires (Rubber): The tire material is Rubber.

Performance: The rubber material provides a large coefficient of friction and strong grip (traction) force. This design is crucial for stable and controlled movement across different surfaces.

Internal Structure: All tires are internally fitted with a foam lining (insert). This foam insert provides necessary support and structure to the soft rubber tire, which is essential for consistent performance and shock absorption in RC and robot cars.

4) 2.2) Ackermann Steering:

This is the servo that controls the wheels using the Ackermann Steering system:

Ackermann Steering Geometry: Precision Path Management

The chassis design of our autonomous vehicle is fundamentally based on the Ackermann Steering Geometry. We implemented a custom-modified version of this linkage system to ensure optimal cornering performance.

1. Principle of Operation:

Unlike simpler steering methods (like skid steering), the Ackermann principle ensures that during a turn, the steering axis of all four wheels intersects at a single, momentary center point. This is achieved by ensuring the inner wheel (the wheel closer to the turn's center) rotates at a sharper angle than the outer wheel.

This difference in rotation angle is critical because it forces the wheels to follow four distinct radii, allowing the inner wheel to travel a shorter path and the outer wheel a longer path, preventing any lateral slippage.

In our RWD system, the front wheels execute the steering motion around their respective pivots, while the single DC motor drives the non-steering rear wheels.
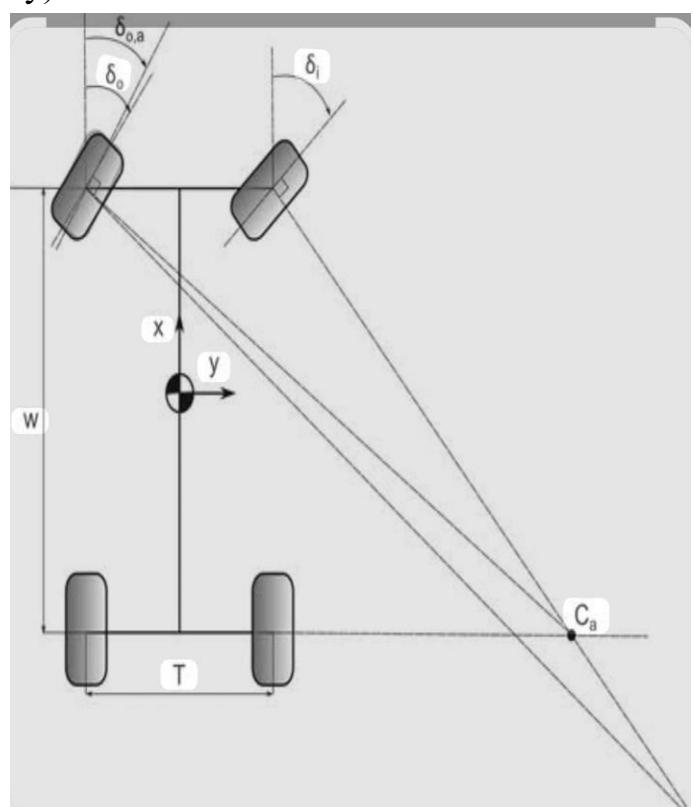
2. Mathematical Description (The Ideal Geometry):

This diagram illustrates the Ideal Ackerman

Steering Geometry. This geometry ensures that all four wheels, when steered, trace concentric circles around a single, common point, the Ackerman Center (C_a).

This fundamental relationship is governed by the vehicle's fixed dimensions—the Track Width (T) and the Wheelbase (W)—and the required steering angles:

$$\cot(a\_o) - \cot(a\_i) = T/W$$

Where a_i is the inner wheel steering angle and a_o is the outer wheel steering angle.

In this ideal setup:

The Inner Wheel Angle (a_i) is always greater than the Outer Wheel Angle (a_o) (a_i > a_o).

The lines drawn perpendicular to the plane of each wheel intersect precisely at C_a, ensuring all wheels roll without excessive side-slip.

Note: Real-world steering systems use a Modified Ackerman principle to optimize performance across the entire steering range, deviating slightly from this ideal formula.

3 .Advantages for Our Robotic System:

Implementing the Ackermann geometry was a strategic choice that provided several specific benefits essential for the competition requirements:

Elimination of Tire Slippage: The main advantage is the prevention of tire scrubbing and slippage, which is crucial for accurate path following and maintaining maximum traction on the carpet surface.

Precise Path Management: The system offers high control over the front wheel's angle, allowing for precise steering control necessary to navigate between colored obstacles and adhere to strict course boundaries.

Single-Motor Compliance: This configuration naturally supports the requirement of using only a single DC motor for propulsion, simplifying the mechanical drivetrain while maintaining high maneuverability.

Part Name	Quantity	Description
Front metal chassis plate	1 pcs	Front base for Ackermann steering system
Rear metal chassis plate	1 pcs	Rear base for motor mounting
Electronics mounting plate	1 pcs	Plate for Arduino / controller installation
Rubber wheels	4 pcs	High-traction rubber wheels
Suspension spring	1 pcs	Simple shock absorption system
Motor/servo mounting bracket	1 pcs	Holder for servo or steering mechanism
Pulley / gear wheels	2 pcs	Used in steering / drive linkage
Metal rods	Several	Used for steering linkage and frame connection
Bearings	Several	To reduce friction and improve smooth steering
Rod ends / ball joints	4 pcs	Steering linkage endpoints
Servo mount brackets	2 pcs	For attaching the steering servo
Metal steering arm	1 pcs	Transfers servo motion to the wheels
Servo motor	1 pcs	Controls front wheel steering
Spacer washers	4 pcs	To maintain spacing between plates
Hex standoffs	Multiple	Structural support and spacing
L-shaped brackets	2 pcs	Support brackets for structure
Screws (M2 / M3 assorted)	Set	Used for assembly
Nuts (M2 / M3 assorted)	Set	Used with screws
Metal shafts / pins	Multiple	For wheel/steering mechanism
Bearings housings	2 pcs	Mounting for wheel axles
Plastic components	Several	Mechanical support parts

4) 2.3) Robot mind: The Raspberry Pi 5 Single-Board Computer (SBC):

The Raspberry Pi 5 serves as the core computational brain of the robot, tasked with executing the Python-based control program, managing all peripherals, and performing computationally intensive tasks like Computer Vision and Sensor Fusion in real-time. The decision to utilize the Pi 5— a significant upgrade from its predecessors—is driven by its unparalleled performance uplift crucial for autonomous mobility.

- Performance and Processing Core:

  The Pi 5 is architecturally superior, featuring a 2.4GHz quad-core Arm Cortex-A76 processor with 512KB per-core L2 caches and a 2MB shared L3 cache. This configuration delivers a 2 to 3 times increase in CPU performance relative to the Pi 4, which is paramount for ensuring low-latency decision-making and the fast execution of complex path-planning algorithms.

  Graphics processing is managed by the 800MHz Video Core VII GPU, offering a substantial uplift in graphics performance and supporting modern APIs like Vulkan 1.2. This hardware acceleration is vital for speeding up OpenCV operations and overall image processing, which directly translates to faster detection and response times for our Computer Vision system (e.g., traffic light recognition).

  Furthermore, the availability of high-speed LPDDR4X-4267 SDRAM (with options up to 16GB) provides the necessary memory bandwidth and capacity to manage large data streams from the USB camera and support memory-intensive Sensor Fusion algorithms effectively.

- Enhanced I/O and Peripheral Connectivity:
  The Pi 5's revolutionary RP1 "southbridge" chip provides a major step change in peripheral performance, directly addressing critical bottlenecks for robotics applications:
- USB Bandwidth:
  The two USB 3.0 ports now support simultaneous 5Gbps operation. This is critical for connecting our high-speed USB camera, guaranteeing a high, consistent frame rate vital for real-time traffic light detection and line following without data latency issues.
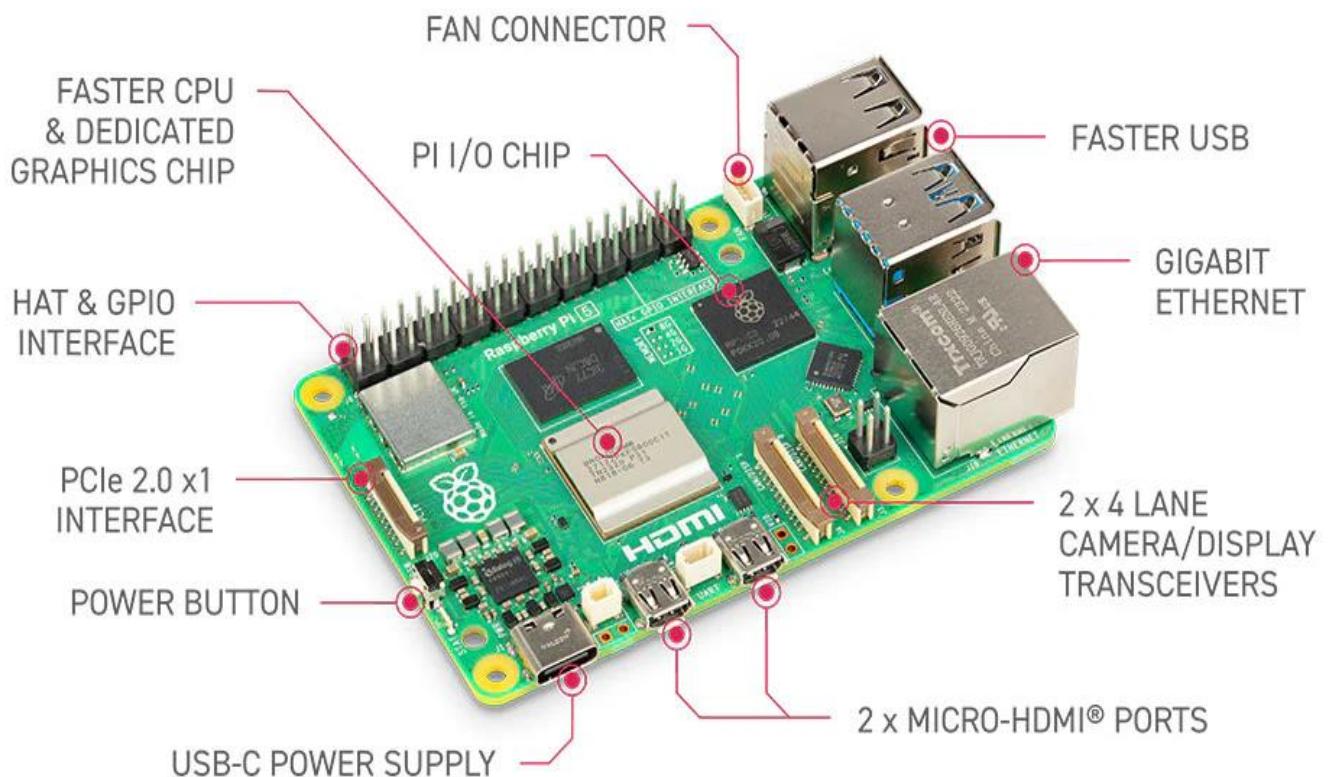
- Camera Interfaces:
  The platform replaces the single CSI interface with a pair of 4-lane 1.5Gbps MIPI transceivers, effectively tripling the total camera bandwidth. This significantly improves the processing capability for the vision system and provides a robust future pathway for integrating dedicated CSI-2 modules or stereoscopic vision systems.

- Low-Level Control:

  The standard 40-pin GPIO header remains the primary interface for all low-level communication. This includes PWM signals to the motor driver and servo, and reliable serial buses (I2C and SPI) for data exchange with the MPU-6050 IMU and the Encoder.

- Power Management and Reliability:

  The Raspberry Pi 5 requires a dedicated 5V/5A DC power input. To meet this demand from the robot's 9V battery, the LM2596 Voltage Regulator is essential for stepping down the input voltage to the stable 5V rail. This ensures the Pi 5 operates reliably, utilizing its Real-Time Clock (RTC) and on-chip oscillator for precise timing across all autonomous operations, which activates the main Python script.

4) 2.4.1) Motor Driver L298N:

L298N Dual H-Bridge Motor Driver: The Power Bridge The L298N module serves as an essential protective and intermediary power stage between the robot's battery and its drive motor. The Necessity of a Driver: The Raspberry Pi 5 (our main controller) operates using low-power signals and cannot safely source or sink the high current required by the DC propulsion motor. Attempting to connect the high-power battery circuit directly to the Pi's low-voltage GPIO pins would result in immediate and irreversible damage to the controller. Therefore, the primary role of the L298N is to act as a current amplifier and insulator, safely handling the large electrical load demanded by the motor. How the System Works (The Dam Analogy): The L298N can be visualized as a large, reinforced concrete dam built to manage the powerful current flowing from the 9V battery (the reservoir) to the motor (the turbine). The Raspberry Pi acts as the control mechanism that precisely regulates the dam's sluice gates (via PWM signals). The Pi merely sends low-voltage commands to the L298N, instructing when and how much of the high-power battery current should be directed to the motor, without ever having to handle the power itself. Component Description: This module is based on the widely used L298 Dual H-Bridge Integrated Circuit. While the board is capable of controlling two DC motors independently (up to 2A peak current each), our application utilizes only one motor port to drive our single-motor RWD system. The unit is optimized for microcontroller interfacing, requiring only a few digital lines for full control. The board also features essential integrated components, including LED power indicators, internal protection diodes, and an onboard +5V voltage regulator which can supply power to the low-voltage controller (like the Raspberry Pi) for convenience. Key Technical Data:

4) 2.4.2) Driver: L298N Dual H-Bridge DC Motor Driver

Operating Voltage Range: DC 5 V - 35 V (Motor Power)

Peak Current: 2 Amp per motor

Input Logic Voltage (Control Signal):
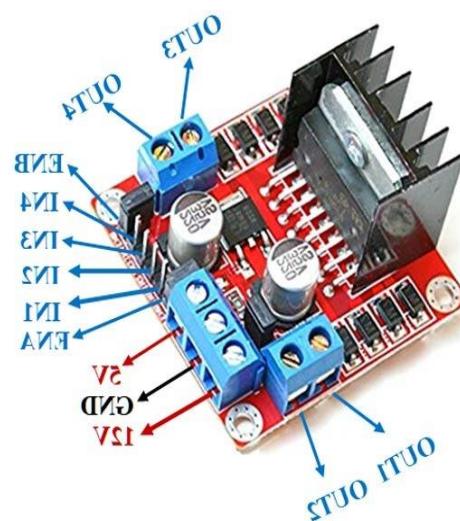
Low: $-0.3 \leq Vin \leq 1.5V$ (control signal is invalid).

High: $2.3V \leq Vin \leq Vss$ (control signal active).

Maximum Power Dissipation: \text{20W} (at $T = 75^\circ C$)

On-board Feature: Integrated +5V regulated output supply.

Approximate Dimensions: 4.3cm × 4.3cm × 2.7cm

4) 2.5) Motor DC:

This motor belongs to the class of DC Geared Motors, specifically the JGA25-370 series, known for providing a necessary trade-off between speed and torque for robotics and automation applications. It is designed to be highly compatible with popular microcontroller platforms such as Arduino, Raspberry Pi, and STM32, utilizing common DC motor driver modules.

- Model Name and Type: JGA25-370 DC Gear Motor. The name implies a 25mm gearbox diameter attached to a 370 series motor.
- Operating Voltage: The nominal operating voltage is 12V DC.
- Output Speed (No-load): It provides an output speed of 210RPM (Revolutions Per Minute) under no-load conditions.
- Torque Capability: The motor delivers up to 12Kg • cm of torque. This makes it suitable for medium-load mechanical motion control projects.
- Gearbox Material: It features a durable all-metal gearbox for enhanced strength and long service life, ensuring stable and reliable continuous operation.
- Output Shaft: The typical output shaft diameter is 6mm.
- Typical Applications: This motor is ideal for building robots, conveyor systems, smart vehicles, and other DIY projects requiring controlled motion.
- Stall Current: The internal motor (before the gearbox) can draw a Stall Current (maximum current draw when the shaft is blocked) of approximately 2.2 Amps at 12V.
- Motor Driver Requirement: This means the chosen motor driver (the module linking the motor to the Raspberry Pi) must have a continuous current rating exceeding 2.2 Amps per channel to safely operate the motor, especially when the car is starting or pushing against an obstacle.

4) 2.6) Servo Motor:

Precise Steering Actuator the Servo Motor is the critical actuator responsible for translating the path-planning commands from the Raspberry Pi into physical steering motion Integration with Ackermann Geometry Function: The servo is connected via a custom linkage to the front wheels, enabling the precise, differential steering angles required by the Ackermann Steering Geometry. Model: Based on its characteristics (and likely MG996R designation), the model used is a Digital, High-Torque Servo with Metal Gears. The high torque rating10 ~) kg・ cm (ensures sufficient force to overcome the mechanical friction and maintain the steering angle under load.

Control and Power Control Protocol: The servo angle is precisely controlled by sending Pulse Width Modulation (PWM) signals from the Raspberry Pi's GPIO header. The duration of the pulse dictates the angular position of the angular position of the wheel. Power: The servo requires a stable operating voltage typically in the 5V - 6V range. This power is reliably supplied by the Step-down Module (Voltage Regulator), ensuring steady operation independent of the main 9V battery fluctuations.
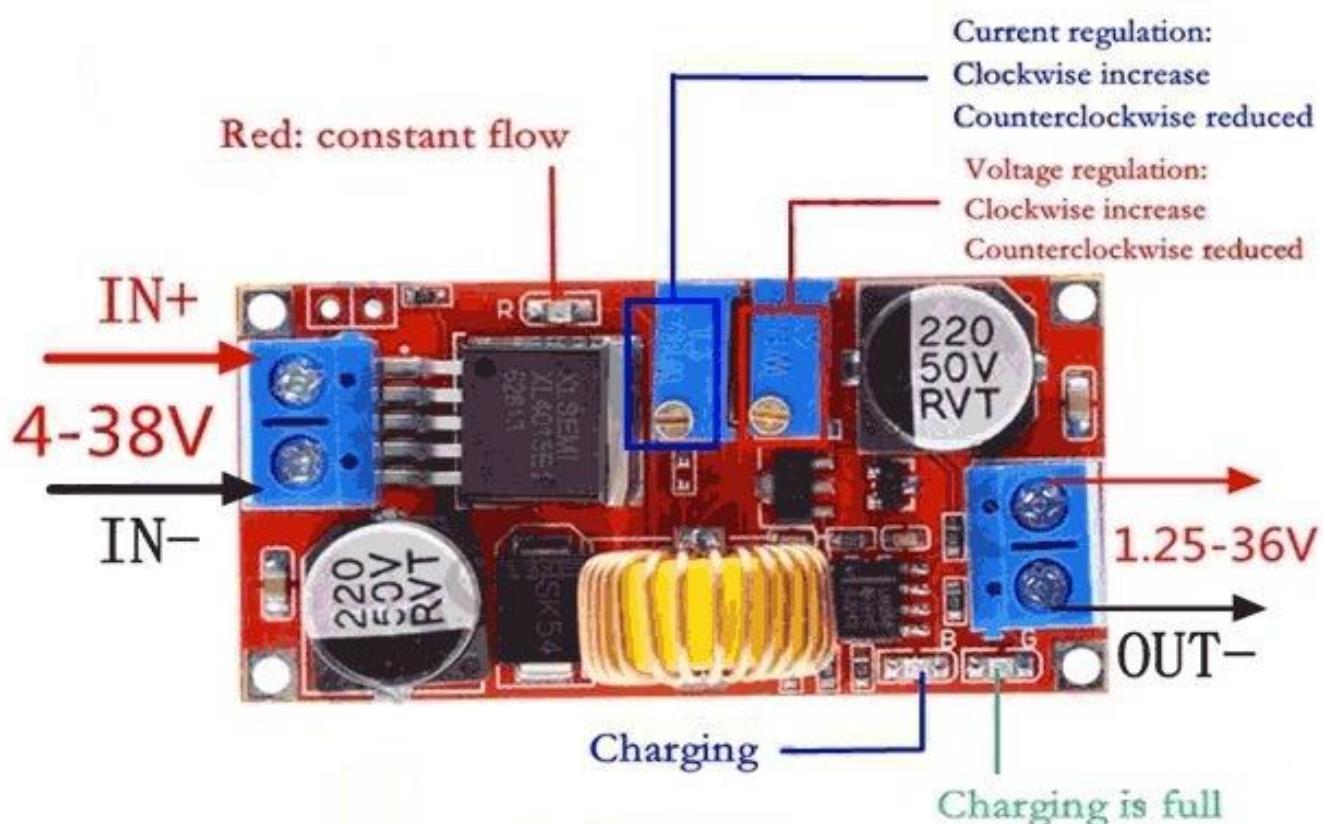
4) 2.7) Voltage regulator XL4015:

The DC-DC Step-down Module is a non-isolated buck converter based on the widely used LM2596 integrated circuit. It serves as a crucial power conditioning component, managing the transition from the high-power drive system to the sensitive control electronics.

Power Regulation Necessity The robot's propulsion system operates on a 9V battery supply, necessary for the DC drive motor. However, the sensitive low-voltage components— namely the Raspberry Pi 4 5V/3A minimum), the Servo Motor, and all digital Sensors— require a stable and lower operating voltage. The primary function of this module is to safely and efficiently step-down the 9V input voltage to a regulated 5V output. This prevents potential damage to the control board and ensures power stability, which is vital for the reliable operation of the Raspberry Pi's CPU and GPIO communications.

Key Operational Data The module accepts a wide input range 4.5V to 40V), easily accommodating the 12V battery output. It is manually tuned to provide a precise 5V output voltage and is rated for a continuous output current of approximately 3A, providing a sufficient power budget for all low-power electronics in the system. The high conversion efficiency (> 80%) minimizes energy loss as heat, preserving battery life.

4) 2.8.1) Gyroscope MPU6050:

Inertial Measurement Unit (IMU): MPU-6050 The MPU-6050 Inertial Measurement Unit (IMU) serves as the primary sensor for motion and orientation tracking, correcting for mechanical and environmental inaccuracies that cannot be managed by vision or distance sensors.

Function and Integrated Components The MPU-6050 is a sophisticated MotionTracking device that integrates a 3-axis gyroscope and a 3-axis accelerometer. It utilizes 16-bit Analog-to-Digital Converters (ADCs) for high-precision data capture.
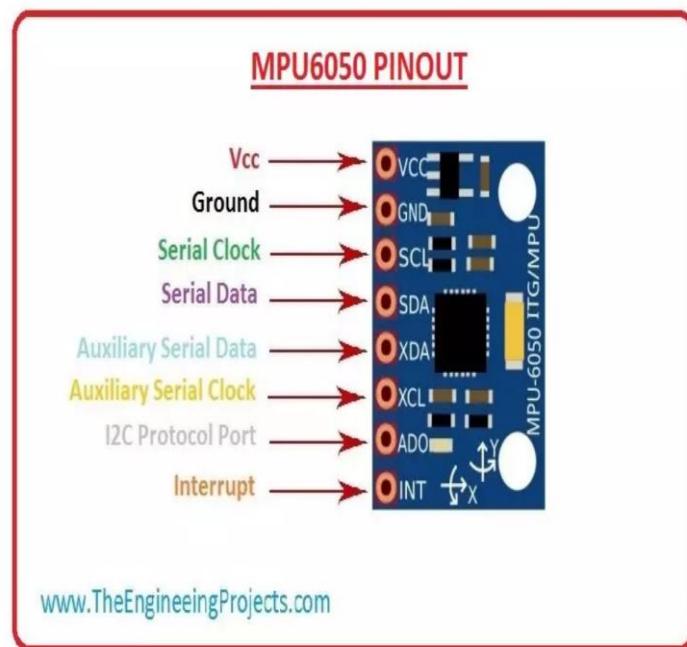
Gyroscope (Angular Rate): Measures the rate of rotation around the X, Y, and Z axes. This is essential for detecting unwanted yaw (heading deviation) and precisely correcting the Ackermann steering angle. The range is user-programmable up to +- 2000º/sec.

Accelerometer (Linear Acceleration): Measures linear acceleration and gravity, used to determine the robot's tilt angle (roll and pitch) and estimate linear distance traveled. The range is programmable up to +-16g.

Digital Motion Processor (DMP): This integrated hardware unit processes complex 6-axis Motionfusion algorithms. Crucially, the DMP offloads computation from the Raspberry Pi by processing raw sensor data into clean, ready-to-use orientation information directly on the chip.

System Integration and Power The MPU-6050 communicates with the Raspberry Pi 4 using the efficient I2C serial communication protocol. This protocol is highly efficient, requiring a minimum number of dedicated GPIO pins.

Power: The module operates between 3V to 5V and is safely powered by the 5V} regulated output from the LM2596 module.   Role in Navigation the IMU provides the core data for internal navigation (dead reckoning). Regardless of whether the system implements explicit sensor fusion with the ultrasonic sensors, the highly accurate angular rate and acceleration data are crucial for stabilizing the robot's movement and ensuring that the programmed path is maintained against physical disturbances.

MPU6050 PINOUT

Vcc → VCC
Ground → GND
Serial Clock → SCL
Serial Data → SDA
Auxiliary Serial Data → XDA
Auxiliary Serial Clock → XCL
I2C Protocol Port → ADO
Interrupt → INT

MPU-6050 ITG/MPU

www.TheEngineeingProjects.com

4) 2.8.2) Gyroscope Working Principle:

A gyroscope is a sensor used to measure the rate of rotation around the different axes of an object, allowing it to detect how the body rotates in space. Gyroscopes usually work by measuring the Coriolis force acting on a moving element inside the sensor and converting it into an electrical signal. (Note: Coriolis force is an apparent force that arises when an object moves in a rotating frame of reference.) This signal enables the calculation of the change in angle over time, allowing robots to know their orientation accurately, maintain balance, or move in a straight line. Gyroscopes are widely used in robotics for navigation and motion control

Equation

$$\theta = \int \omega \, dt$$

Symbols Explanation:
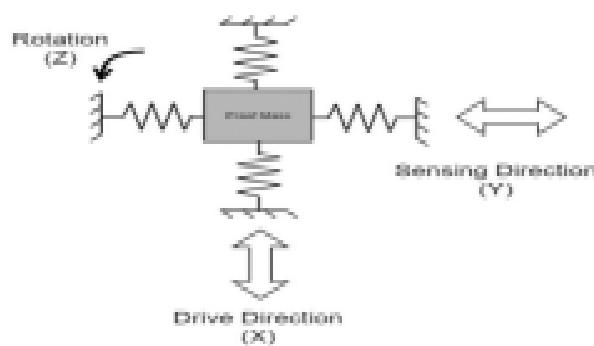
Angle rotated (degrees or radians). : θ

Rotation rate (degrees/s or rad/s). : ω

Time interval between gyroscope readings (seconds): dt

• Role of the Gyroscope:
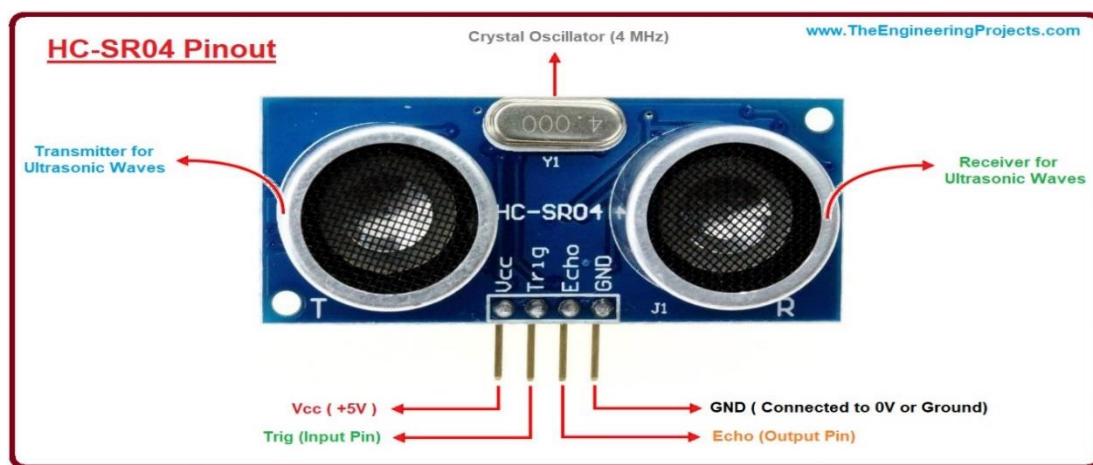
In our robot, the gyroscope helps determine the robot's orientation precisely when facing turns and obstacles. In the Obstacle Challenge, it measures the robot's rotation rate and calculates the change in angle to decide whether to turn left or right, keeping it aligned with the path. In the Open Challenge, the gyroscope works alongside distance sensors to avoid collisions, maintain a fixed distance (~20 cm) from walls or obstacles, and allow the robot to make smart turning decisions without relying solely on the path lines

4) 2.9.1) Ultrasonic In our robot:

Ultrasonic Sensors (HC-SR04): Comprehensive ToF Ranging System The robot is equipped with a total of four HC-SR04 Ultrasonic Sensors that form the primary short-range ranging system. These sensors provide the vehicle with echolocation capabilities, on par with how bats and dolphins locate objects in complete darkness and beneath the water surface. This array is fundamental for robust, real-time obstacle avoidance and precise navigation along the course boundaries. Strategic Implementation and System Integration The four HC-SR04 sensors are strategically positioned to maximize situational awareness: one mounted in the front, one in the rear, and one on each of the right and left sides of the chassis. This arrangement minimizes blind spots and provides the angular data necessary for advanced Sensor Fusion. Power and Interface: The sensors require a stable +5V regulated supply, drawing less than 15mA each. Each sensor utilizes two dedicated GPIO pin (Trigger and Echo) on the Raspberry Pi for control and data acquisition. Sensor Fusion Role: The simultaneous input from four points allows the control system to accurately determine both the distance and the angular position of an obstacle. This redundancy is vital for high-reliability autonomy, allowing the algorithm to differentiate between a critical frontal obstacle and a simple boundary wall alongside the vehicle.
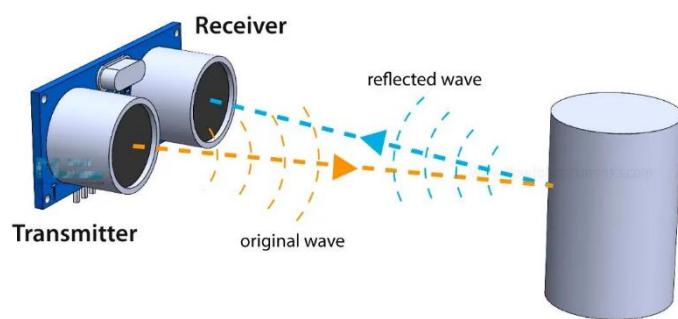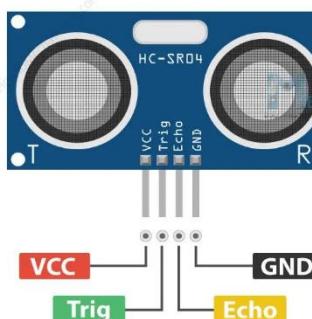
Principle of Operation and Technical Specification The HC-SR04 module operates on the principle of measuring the Time-of-Flight (ToF) of an ultrasonic sound wave. 1–Triggering: Measurement begins when the Raspberry Pi sends a minimum 10uS high pulse to the Trigger pin, prompting the transmitter to emit an 8-cycle burst of ultrasonic sound at 40kHz. 2– Echo Reception: The sound wave reflects off an object, and the duration it takes to return is measured. The sensor outputs a high pulse on the Echo pin, the width of which is directly proportional to the distance. 3– Calculation: The Raspberry Pi measures this pulse width (time), and the distance is calculated using the formula Distance = Speed × Time, divided by two since the time measured is for the signal's round trip. The sensor boasts a practical measuring range of 2cm to 400cm and an accuracy that can reach 3mm. This combination of range and precision makes it perfectly suited for the short-range, dynamic obstacle avoidance challenges presented in the competition course.

# 4) 2.9.2) Ultrasonic Sensor Working Principle:

The ultrasonic sensor measures distance by emitting high-frequency sound waves that are beyond human hearing, typically above 20 kHz, and detecting the echo that bounces back from an object. The sensor consists of a transmitter that sends ultrasonic waves and a receiver that listens for the reflected echo. When the transmitter emits a short burst of sound waves, they travel through the air, hit an object, and return to the sensor as an echo
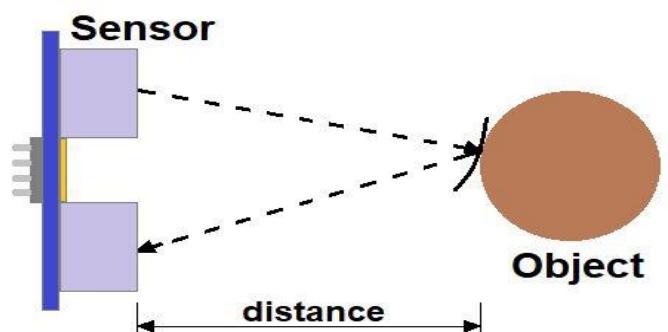
When the transmitter emits a short burst of sound waves, they travel through the air, hit an object, and return to the sensor as an echo. The sensor calculates the distance by measuring the time interval between sending the wave and receiving the echo, using the formula:

Distance = (Speed × Time) / 2

Dividing by two accounts for the round trip of the waves. The speed of sound is approximately 343 meters per second at room temperature, and it can vary with environmental conditions such as temperature and humidity, Ultrasonic sensors are typically accurate within ±3 millimeters and can measure distances ranging from 2 centimeters to 400 centimeters. They usually have a detection angle of 15 to 30 degrees, so objects outside this range may not be detected. These sensors are widely used in robotics for obstacle detection, in cars as parking sensors, in measuring liquid levels, and in various industrial automation applications

$$Distance = \frac{Time \; x \; Speed \; of \; Sound \; in \; Air \; (343m/s)}{2}$$

- ## **Role of the Distance Sensor:**

To navigate the initial stage intelligently, we developed a dedicated algorithm that empowered the robot to make autonomous directional decisions, Prior to encountering the first turn, the robot was in a state of uncertainty, unsure whether to proceed clockwise or counterclockwise.

We incorporated ultrasonic sensors into the robot's design to provide it with continuous spatial awareness. These sensors measured the distance to nearby obstacles on both the left and right sides.

Right Sensor: Conversely, if the right sensor measured a distance beyond 160 cm, the robot opted to move counterclockwise

Figure 3: the robot moved on the game field counterclockwise

Left Sensor: If the left ultrasonic sensor detected a distance exceeding 160 cm (indicating ample space on that side), the robot interpreted this as a signal to move clockwise

Figure 4: the robot moved on the game field with clockwise

Figure 5: Start counterclockwise
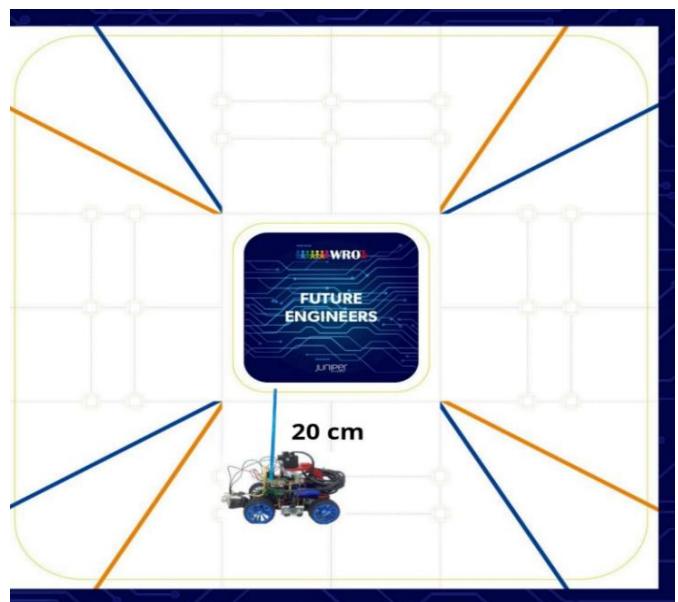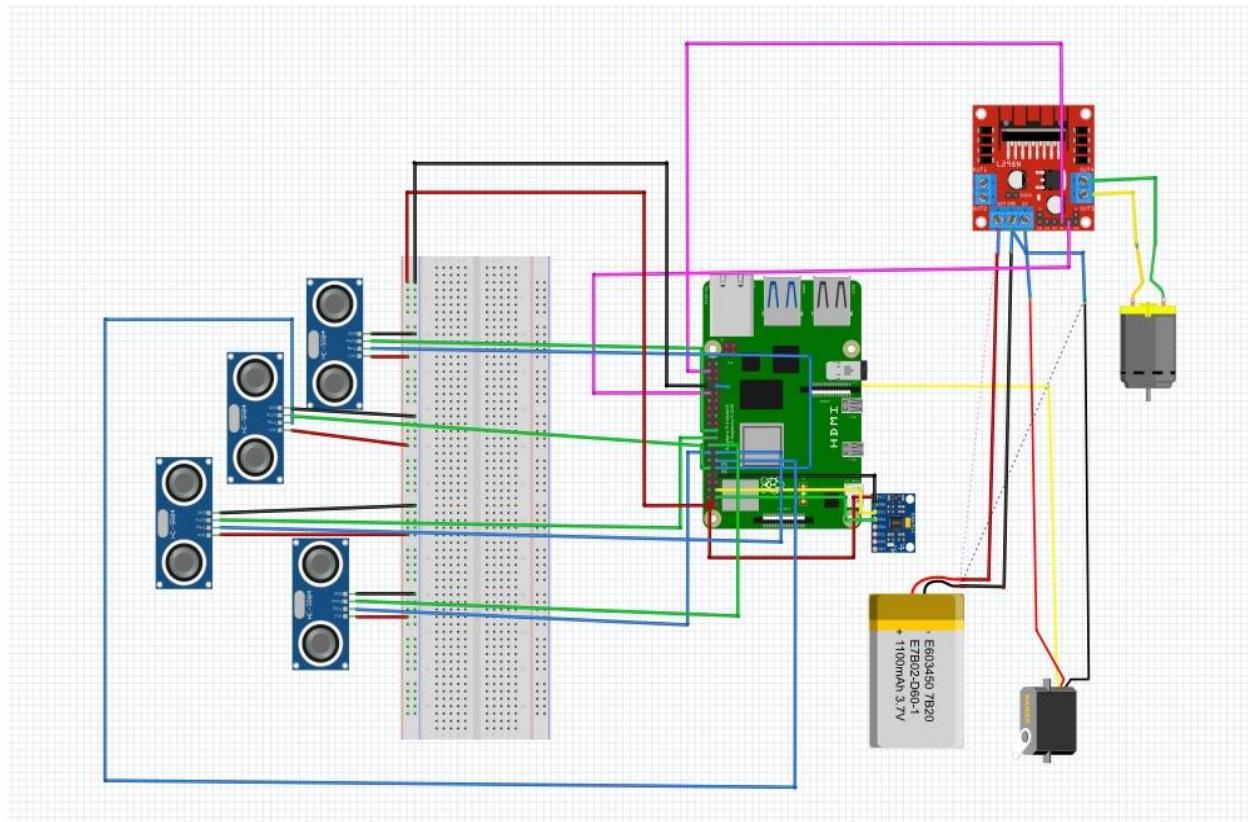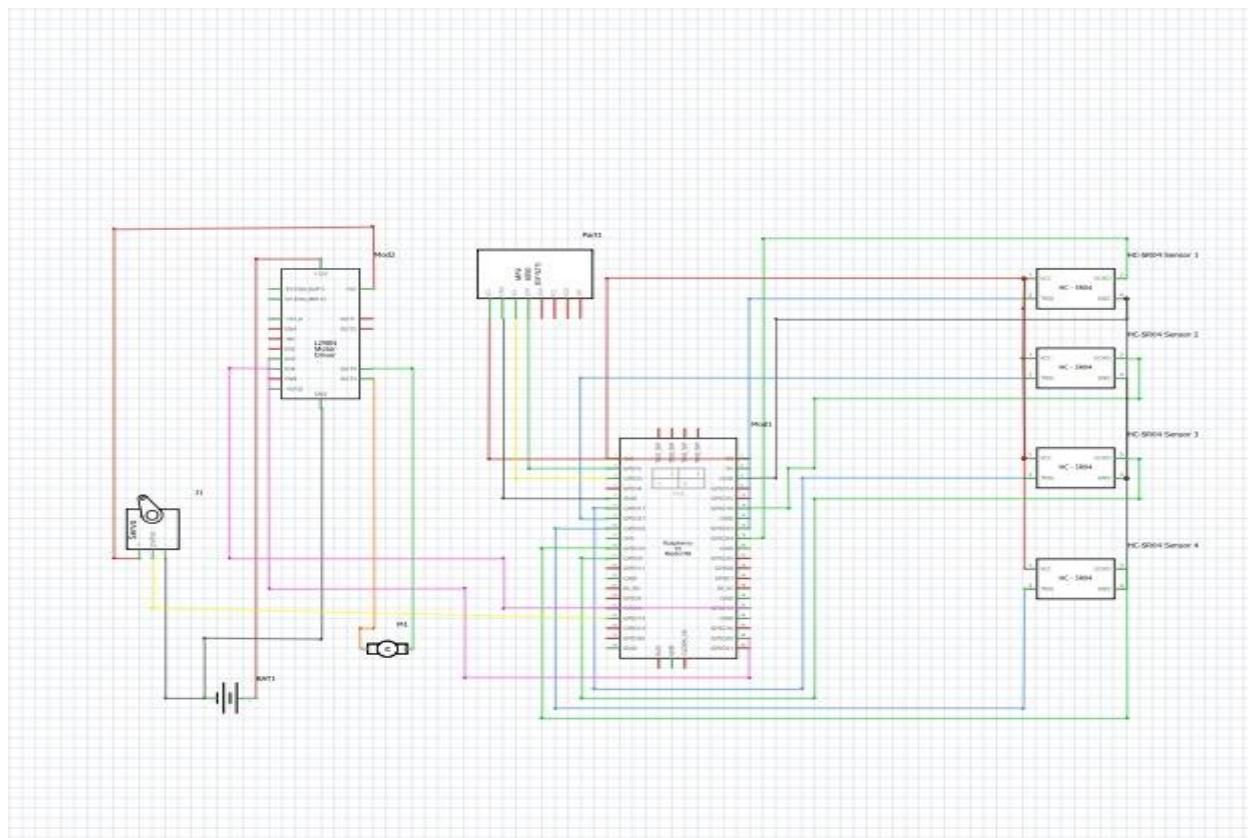
Figure 6: Start with clockwise
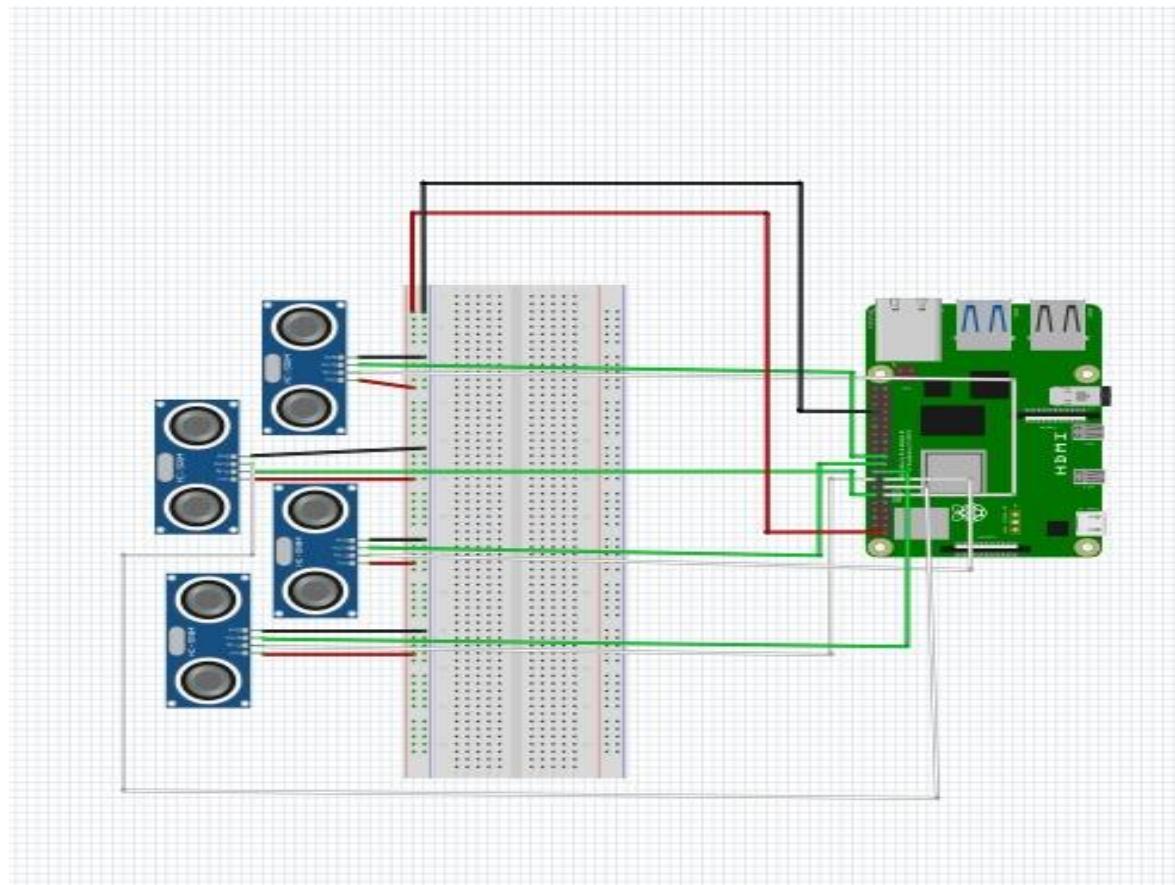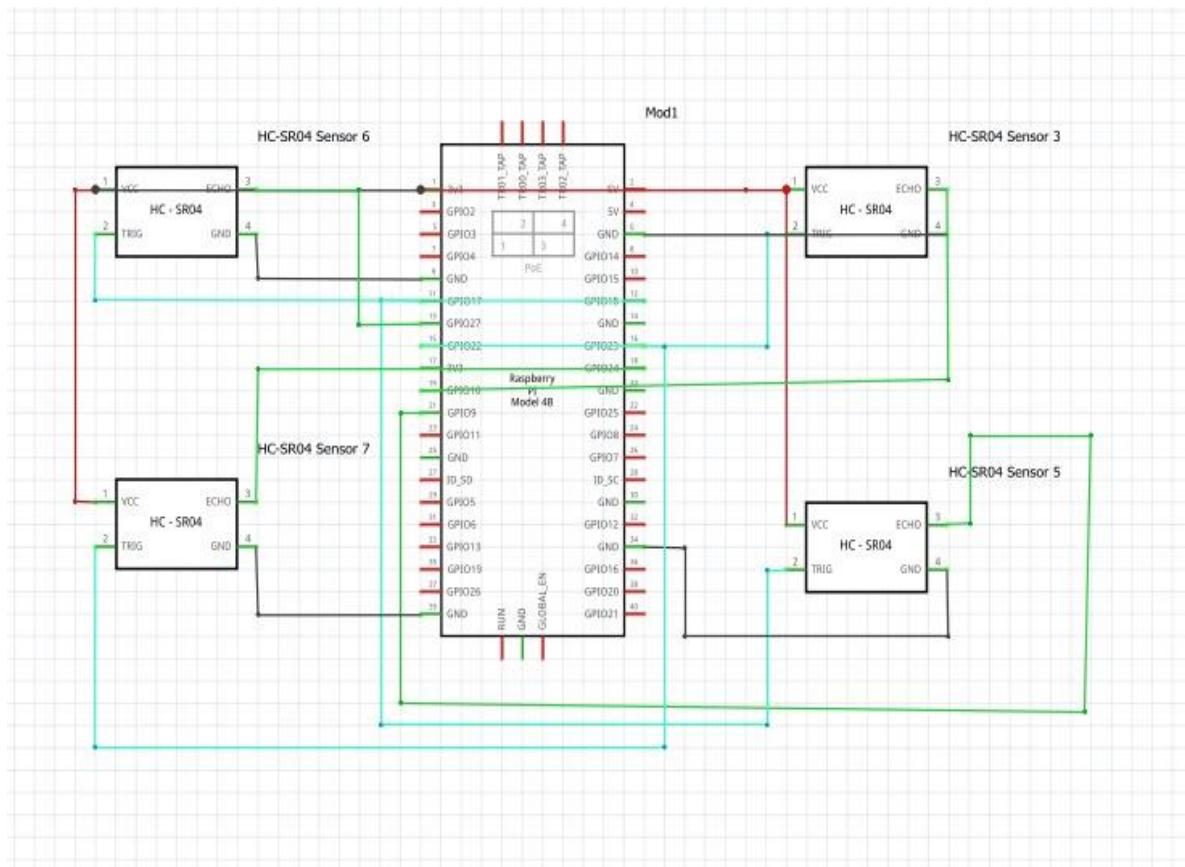
Moreover, while traveling in a straight path the robot maintained a safety buffer of 20cm from the inner wall. If this distance was breached, the system automatically adjusted the trajectory to keep the robot stable and balanced This algorithm allowed the robot to dynamically adapt to its surroundings, making intelligent path choices and real-time corrections, embodying a sense of spatial awareness similar to advanced autonomous systems

5) The Diagrams:
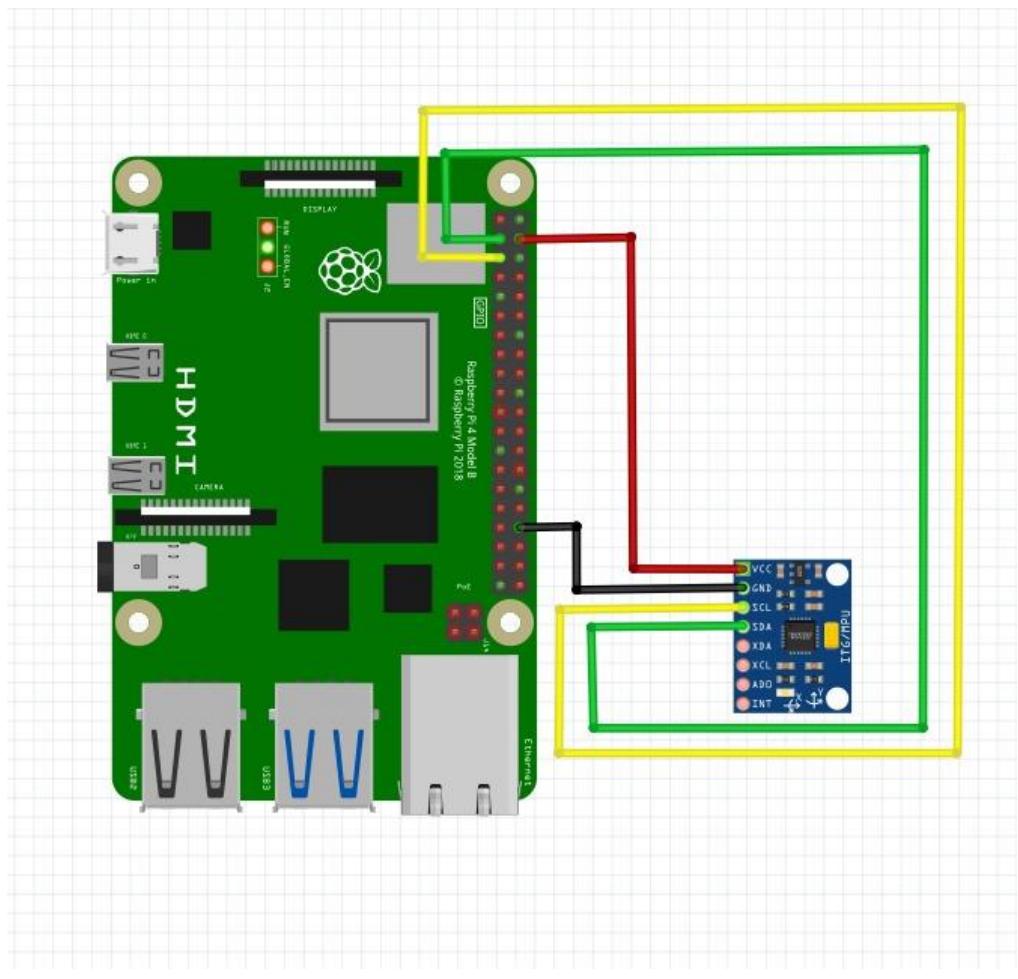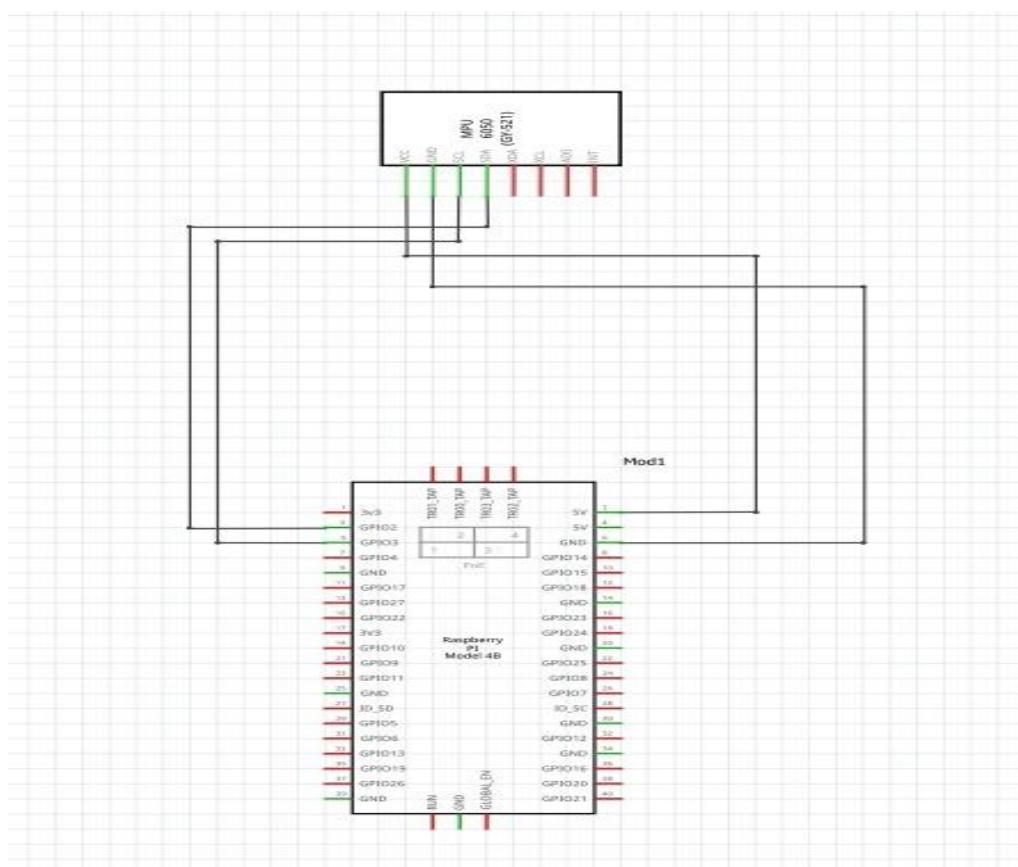
5.1) Integrated Wiring Diagram for Smart Autonomous Vehicle System:

5.2) Detailed Wiring Diagram for Range and Direction Sensors:

5.3) Attitude and Angle Sensing Wiring:

6) Robot Photos:

This is our robot photos, show it with six di

Right view
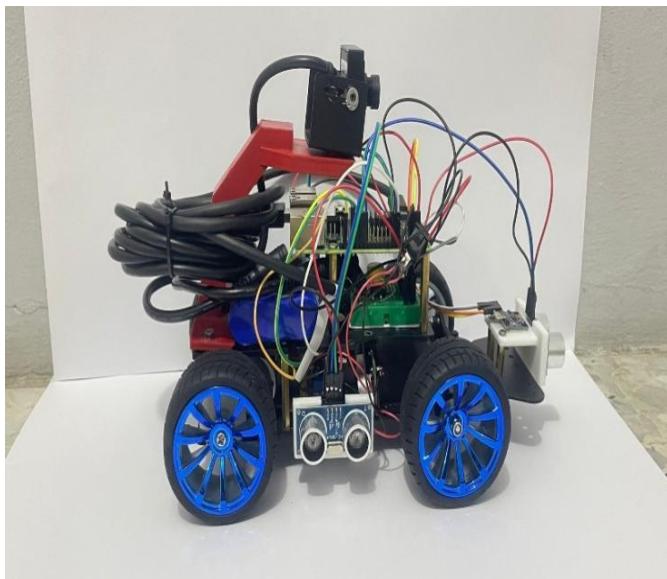
Figure 8: Right-side of the robot

Left view
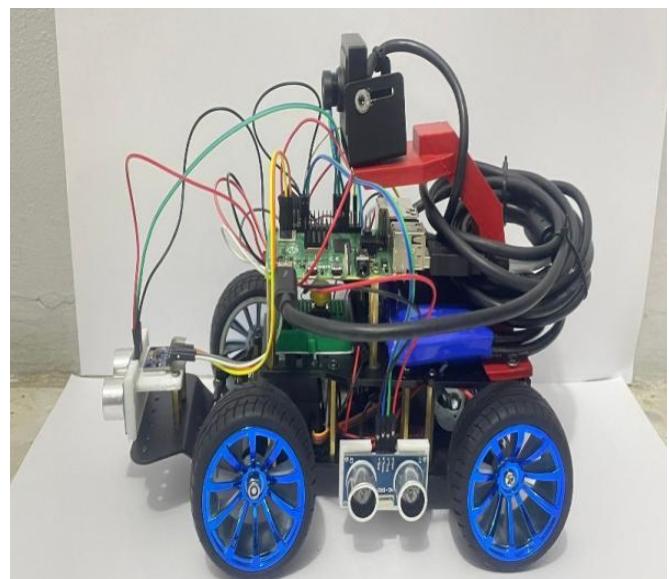
Figure 9: Left-side of the robot

Front view
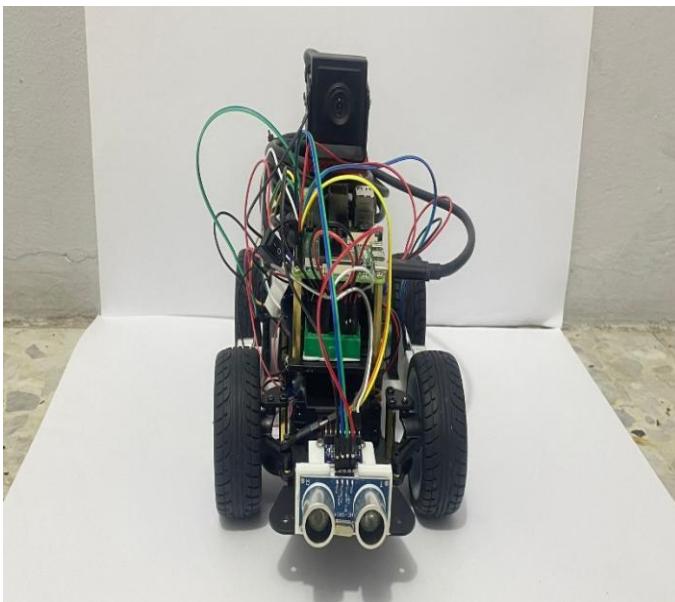
Figure 10: Front-side of the robot

Back view
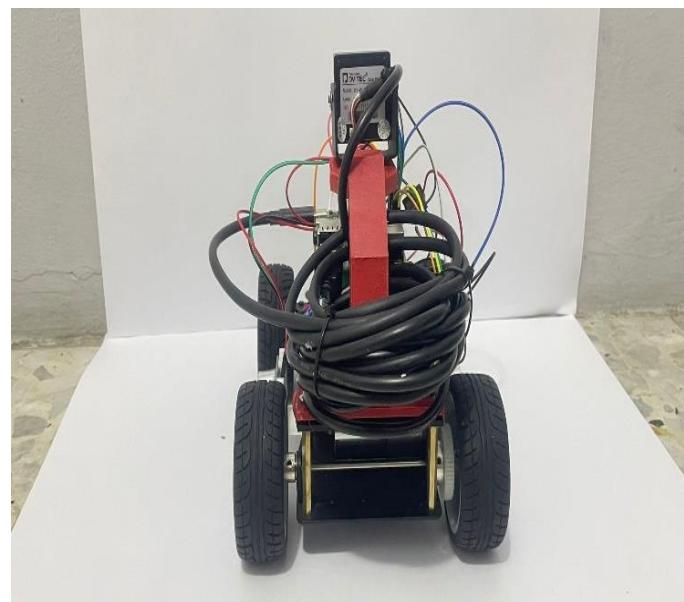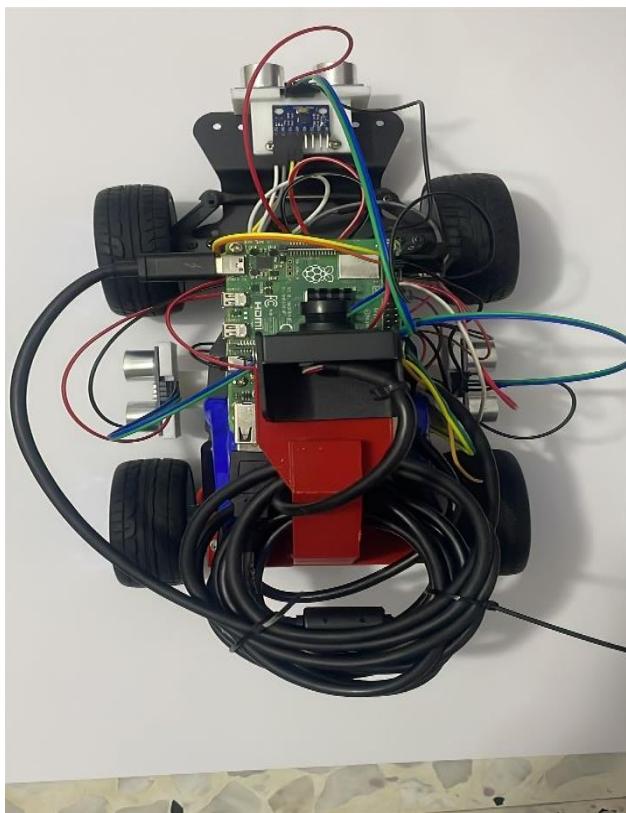
Figure 11: Back-side of the robot
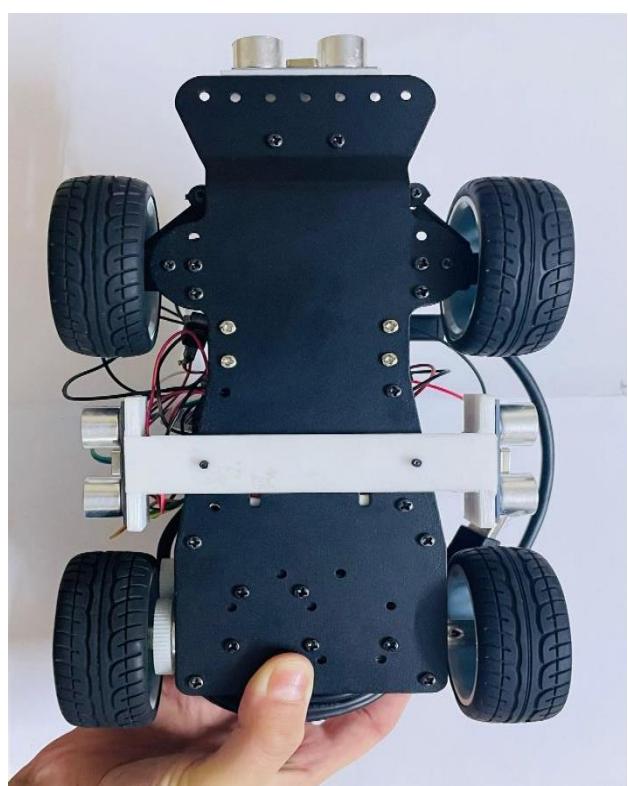
Figure 12: Top-side of the robot



Figure 13: Bottom-side of the robot
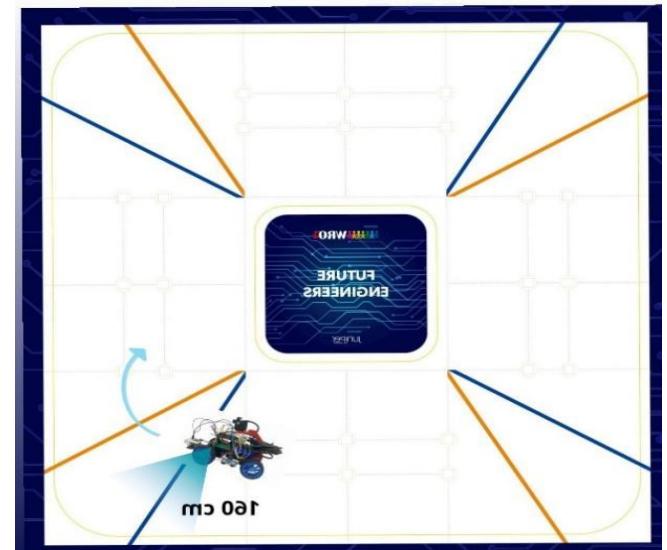
7) Explanation of the rounds:

7.1) Open Challenge round:

In this round, our robot must complete three (3) laps on the track with random

placements of the inside track walls within 3 minutes.

This video shows our robot completing the first round (Open Challenge), you

can view the video we created from this link
https://youtu.be/wE9Jfp3p2G0?si=66htXvw86zWMX0ru

The strategy

7.2) Obstacle Challenge Round:

In this round, the robot must complete three laps on a track with randomly placed green and red traffic signs.

- o **Red Obstacle:** The robot must keep to the right side of the line.

- o **Green Obstacle:** The robot must keep to the left side of the line
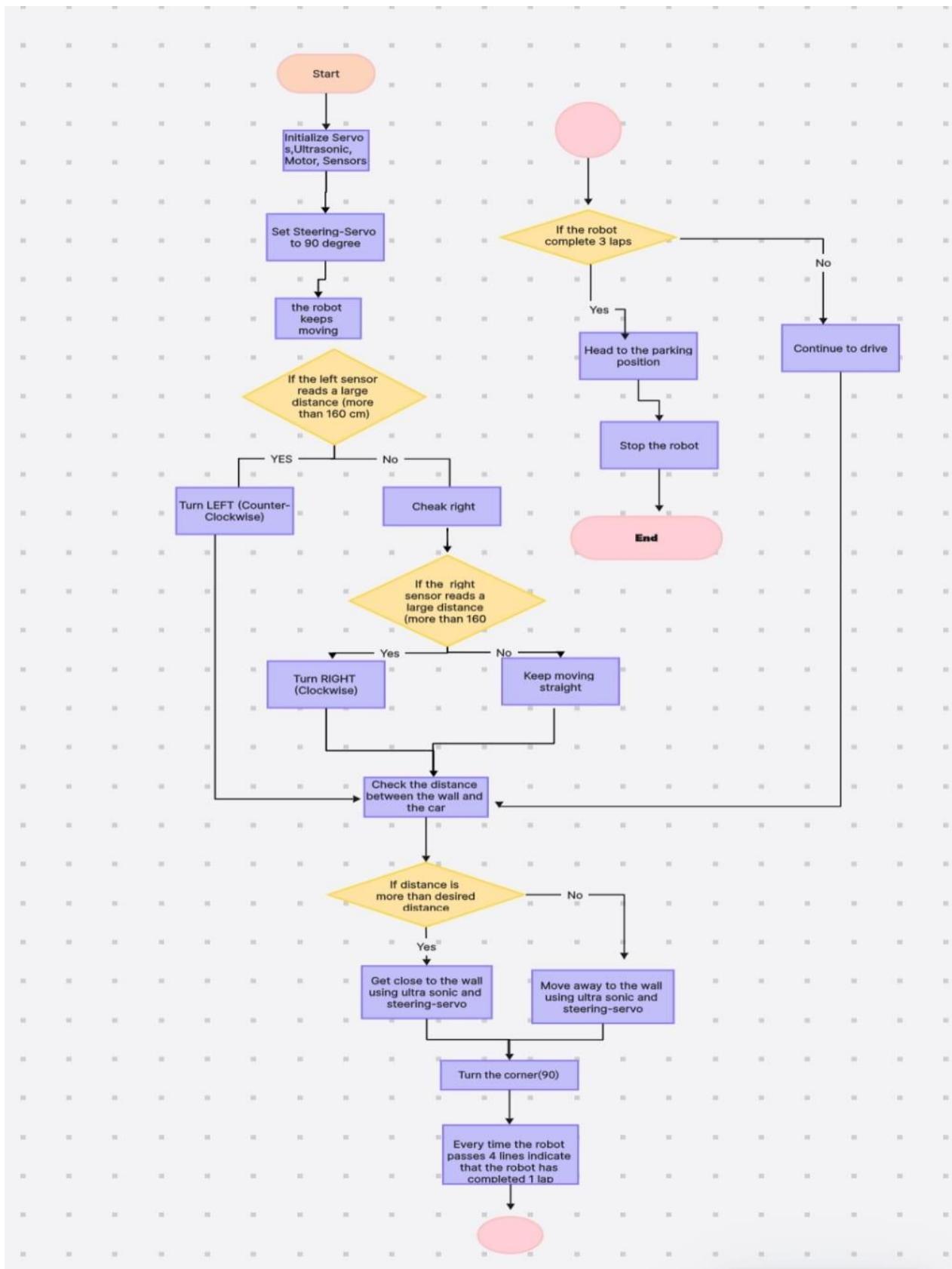
o After finishing the three laps, the robot must find a parking lot and perform parallel parking
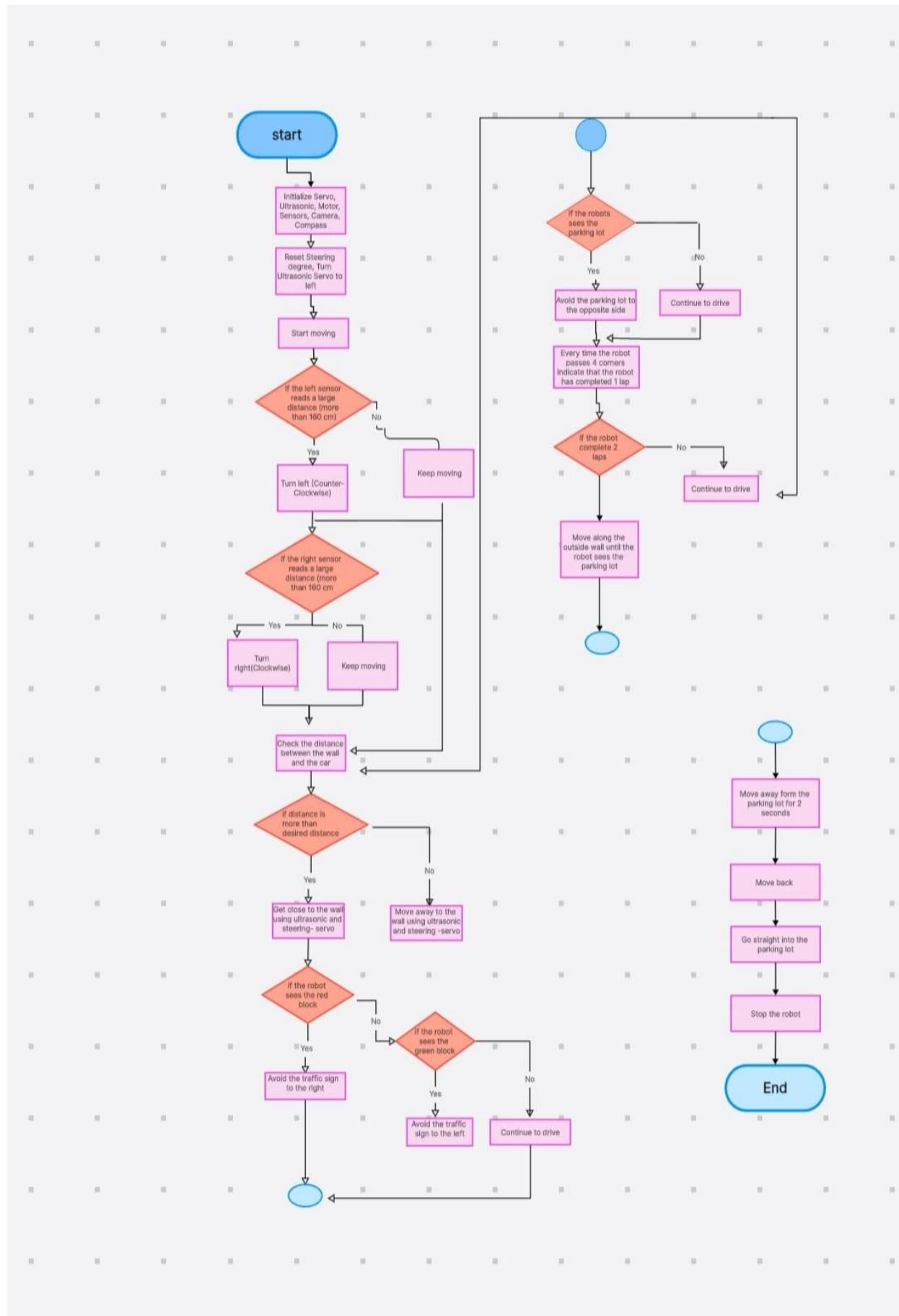
8) Flowcharts:

8.1) Flowchart [Open Challenge round]:

8.2) Flowchart [Obstacles Challenge round]:

9) The Codes explanation:

9.1) Open Challenge Code:

Basic Import and Initialization

```python
import RPi.GPIO as GPIO

import time

import smbus


MOTOR_IN3 = 38

MOTOR_IN4 = 32

SERVO_PIN = 37


FRONT_TRIG = 13

FRONT_ECHO = 11

RIGHT_TRIG = 31

RIGHT_ECHO = 29

LEFT_TRIG = 10

LEFT_ECHO = 8


SERVO_STRAIGHT = 60

SERVO_RIGHT = 95

SERVO_LEFT = 10


MPU6050_ADDR = 0x68

bus = smbus.SMBus(1)


DETECTION_DISTANCE = 80

DIRECTION_CHECK_DISTANCE = 360

TURN_DURATION = 1.1

STRAIGHT_AFTER_TURN = 1.5
```

Importing the basic libraries to control the robot, defining the ports connected to the Raspberry Pi for each component, setting the three servo angles, and defining the distances that constitute a close obstacle requiring the robot to stop and detect the path direction (clockwise or counter-clockwise).

Initialization GPIO Initialization mpu6050 Initialization ultrasonic

```python
def setup_gpio():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)

    GPIO.setup(MOTOR_IN3, GPIO.OUT)
    GPIO.setup(MOTOR_IN4, GPIO.OUT)
    GPIO.setup(SERVO_PIN, GPIO.OUT)

    for trig in [FRONT_TRIG, RIGHT_TRIG, LEFT_TRIG]:
        GPIO.setup(trig, GPIO.OUT)
        GPIO.output(trig, False)

    for echo in [FRONT_ECHO, RIGHT_ECHO, LEFT_ECHO]:
        GPIO.setup(echo, GPIO.IN)


def mpu6050_init():
    try:
        bus.write_byte_data(MPU6050_ADDR, 0x6B, 0)
        time.sleep(0.1)
        return True
    except:
        return False
```

Motor Control [The motor runs, stops]

```python
def motor_forward():

    GPIO.output(MOTOR_IN3, GPIO.LOW)

    GPIO.output(MOTOR_IN4, GPIO.HIGH)


def motor_stop():

    GPIO.output(MOTOR_IN3, GPIO.LOW)

    GPIO.output(MOTOR_IN4, GPIO.LOW)
```

Controlling the servo, preparing it, and setting it to a specific angle.

```python
def setup_servo:()
    global servo_pwm
    servo_pwm = GPIO.PWM(SERVO_PIN, 50)
    servo_pwm.start(0)
    time.sleep(0.1)


def set_servo_angle(angle):
    duty = 2.5 + (angle / 180.0) * 10
    servo_pwm.ChangeDutyCycle(duty)
    time.sleep(0.3)
    servo_pwm.ChangeDutyCycle(0)


def steer_straight:()
    set_servo_angle(SERVO_STRAIGHT)


def steer_left:()
    set_servo_angle(SERVO_LEFT)

def steer_right:()
    set_servo_angle(SERVO_RIGHT)

Reading the distance using the ultrasonic sensor

def get_distance(trig_pin, echo_pin):
    GPIO.output(trig_pin, True)
    time.sleep(0.00001)
    GPIO.output(trig_pin, False)


while GPIO.input(echo_pin) == 0:
        pulse_start = time.time()
        if pulse_start - timeout > 0.1:
            return -1
```

```python
while GPIO.input(echo_pin) == 1:
    pulse_end = time.time()
    if pulse_end - pulse_start > 0.1:
        return -1


pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    distance = round(distance, 1)


    return distance if distance < 400 else -1.
```

```python
def get_front_distance():
    return get_distance(FRONT_TRIG, FRONT_ECHO)


def get_right_distance():
    return get_distance(RIGHT_TRIG, RIGHT_ECHO)

def get_left_distance():
    return get_distance(LEFT_TRIG, LEFT_ECHO)

def detect_direction():
    left_readings[] =
    right_readings[] =

    for i in range:(10)
        left = get_left_distance()
        right = get_right_distance()

        if left > 0:
            left_readings.append(left)
        if right > 0:
            right_readings.append(right)
        time.sleep(0.1)


avg_left = sum(left_readings) / len(left_readings) if left_readings else 0
    avg_right = sum(right_readings) / len(right_readings) if right_readings else 0


if avg_left > DIRECTION_CHECK_DISTANCE:
        return 'CCW'
    elif avg_right > DIRECTION_CHECK_DISTANCE:
        return 'CW'
```

```
    else:

        if avg_left > avg_right:

            return 'CCW'

        else:

            return 'CW'
```

If Left > 360 cm → CCW (Counter-Clockwise)

If Right > 360 cm → CW (Clockwise)

If both are < 360 → Choose the wider path

```
        if avg_left > avg_right:

            return 'CCW'
```

```python
def turn_90(turn_right):
    if turn_right:
        steer_right()
    else:
        steer_left()


    motor_forward()
    time.sleep(TURN_DURATION)
    steer_straight()
```

90-degree rotation.

```python
def navigate():
    global sections_completed, direction, running


    motor_forward()
    steer_straight()



    while running and sections_completed < TOTAL_SECTIONS:
        front_dist = get_front_distance()


        if front_dist <= DETECTION_DISTANCE and front_dist > 0:
            motor_stop()



if direction is None:
        time.sleep(2)
        direction = detect_direction()
    else:
        time.sleep(0.1)
```

If the separation distance ≥ 80 cm → Stop.

First time: Wait for 2 seconds and detect the error.

Subsequent times (Rest of the times): Wait for only 0.1 seconds.

```python
def main():
    global running


    try:
        setup_gpio()
        setup_servo()
        mpu6050_init()
        time.sleep(0.5)


        steer_straight()
        motor_stop()


        running = True
        navigate()
```

The code begins to execute.

10) Obstacle Challenge Code:

This script uses the OpenCV (cv2) library to perform real-time color detection (specifically Red and Green) from a camera feed. It employs Gray World Color Normalization and the LAB color space to achieve robust detection regardless of varying lighting conditions.

```python
import cv2

import numpy as np


def normalize_color(frame):
"""   Gray world color normalization to reduce lighting effect"""
    result = frame.astype(np.float32)

    avg_b = np.mean(result[:, :, 0])

    avg_g = np.mean(result[:, :, 1])

    avg_r = np.mean(result[:, :, 2])

    avg_gray = (avg_b + avg_g + avg_r) / 3


    result[:, :, 0] = np.minimum(result[:, :, 0] * (avg_gray / avg_b), 255)

    result[:, :, 1] = np.minimum(result[:, :, 1] * (avg_gray / avg_g), 255)

    result[:, :, 2] = np.minimum(result[:, :, 2] * (avg_gray / avg_r), 255)


    return result.astype(np.uint8)
```

1 .normalize_color(frame) Function

1 .normalize_color(frame) Function

This function implements the Gray World color normalization algorithm to correct color casts caused by uneven lighting. It assumes that the average color of a scene should be gray.

- **Conversion and Averaging: The frame is converted to float32. It calculates the mean intensity (avg_b, avg_g, avg_r) for each BGR channel and the overall average intensity (avg_gray).**
- **Correction: Each channel's pixels are scaled by a factor of avg_gray/avg_channel. This adjusts channels that are too bright or too dim to match the expected gray mean.**
- **Clamping: np.minimum(..., 255) ensures the corrected values stay within the valid 0-255 range before converting back to uint8.**

```
#Initialize camera (USB or Pi camera)
camera = cv2.VideoCapture(0)
camera.set(640 ,3)
camera.set(480 ,4)


while True:
    ret, frame = camera.read()
    if not ret:
        break


#   Optional: normalize to reduce color cast from lighting
    frame = normalize_color(frame)
```

Camera Initialization and Frame Capture

Initialization: Opens the default camera (0) and sets the resolution to 640 $\times$ 480.

Loop: The main loop continuously reads frames. ret is a boolean indicating successful read.

Normalization: The optional normalize_color() function is called to process the frame before color detection begins.

```
#   Convert BGR to LAB
    lab = cv2.cvtColor(frame, cv2.COLOR_BGR2LAB)
```

```
L, A, B = cv2.split(lab)
```

--- #   Detect RED (A-channel is high for red, low for green)---

```
red_mask = cv2.inRange(A, 150, 255)   # high A → red

green_mask = cv2.inRange(A, 0, 110)   # low A → green
```

LAB Conversion and Color Masking

- **LAB Conversion: The BGR frame is converted to the LAB color space. LAB is preferred as the Luminance channel is separate from the A (Green to Red) chromaticity channels, making color detection less sensitive to brightness variations. The A-channel is then used exclusively to detect Red and Green.**
- **Masking (A-Channel):**
  - **Red Mask: High values in the A-channel (150 to 255) are isolated to create the red_mask.**
  - **Green Mask: Low values in the A-channel (0 to 110) are isolated to create the green_mask.**

Clean up noise

```
kernel = np.ones((5, 5), np.uint8)

red_mask = cv2.morphologyEx(red_mask, cv2.MORPH_OPEN, kernel)

green_mask = cv2.morphologyEx(green_mask, cv2.MORPH_OPEN, kernel)
```

Find contours

```
contours_red, _ = cv2.findContours(red_mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

contours_green, _ = cv2.findContours(green_mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

Morphological Operations and Contour Finding

- **Noise Cleanup (Opening): The morphological opening operation is applied to both masks. This operation (erosion followed by dilation) effectively removes small specks of noise and smooths the boundaries of the detected color regions.**

- **Contour Detection: cv2.findContours() identifies the boundaries of the white (detected color) regions in the cleaned-up binary masks.**

```python
#   Draw bounding boxes for red
  for c in contours_red:
    area = cv2.contourArea(c)
    if area > 400:
      x, y, w, h = cv2.boundingRect(c)
      cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
      cv2.putText(frame, "RED", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)


#   Draw bounding boxes for green
  for c in contours_green:
    area = cv2.contourArea(c)
    if area > 400:
      x, y, w, h = cv2.boundingRect(c)
      cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
      cv2.putText(frame, "GREEN", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
```

Contour Filtering and Annotation
- Filtering: Contours with an area less than 400 pixels are ignored to prevent drawing boxes around minor noise.
- Bounding Box: For valid contours, cv2.boundingRect() calculates the smallest upright rectangle surrounding the contour (x, y, w, h).
- Drawing: A rectangle and a text label are drawn on the original frame in the appropriate color (BGR format: Red = (0, 0, 255), Green = (0, 255, 0)).

```python
#   Display the output
  cv2.imshow("Color Detection (LAB-based)", frame)
```

```python
    if cv2.waitKey(1) & 0xFF == 27:  # ESC key to exit
        break


camera.release()

cv2.destroyAllWindows()
```

Display and Cleanup

- **Display: The processed frame with annotations is shown in a window.**
- **Exit Condition: The application terminates if the ESC key (ASCII 27) is pressed.**
- **Resource Release: The camera resource is released, and all display windows are closed.**

11) References and Applications Used:

During the development of our WRO robot project, we used several applications to design, plan, and present our work efficiently:

- Flowchart Design – BoardMix

We used BoardMix to create clear and organized flowcharts for our robot's logic and program structure. This helped us plan the sequence of operations before writing the actual code.

- Circuit Design – Fritzing

Fritzing was used to design and visualize the electronic circuits of the robot. It allowed us to connect sensors, motors, and other components on a virtual breadboard before physical assembly.

- Presentation and Background – Canva

Canva was used to create professional backgrounds, diagrams, and visual materials for presentations and documentation, making our project visually appealing and easy to understand.

Here we document our work and projects step by step, as part of our participation in Future Engineers. We are a team of female students who work passionately to design and program innovative solutions that combine engineering, programming, and creativity Follow our repositories to learn about our progress and achievements.