West Bengal State Council of Technical & Vocational Education and Skill Development

DASNAGAR GOVERNMENT POLYTECHNIC

(P.O & P.S Dasnagar, Howrah, West Bengal - 711105)

Department of Information Technology & Engineering

MAJOR PROJECT REPORT

on

Object Detection Car Controlled by Android Application

A Report of Project Submitted for partial fulfilment of the requirement for the degree of Diploma in Information Technology & Engineering

Submitted By -

A. **Soumen Mishra**
[D222309148]

B. **Triyasa Dey**
[D232410789]

C. **Nabanita Maity**
[D222309149]

D. **Anirudha Roy**
[D232410788]

E. **Pousali Pal**
[D222309110]

Under Guidance of -

Shri Partha Sarathi Banerjee
[Lecturer in Computer Science and Technology]

Shri Achintya Gopal Mondal
[Workshop Instructor / Instructor (Computer Application)]

Title Page

Project Title :

Object Detection Car Controlled by Android Application

Information Technology Engineering

[Session **: 2022 - 2025]**

Developed by the following Students :

A. Soumen Mishra

B. Triyasa Dey

C. Nabanita Maity

D. Anirudha Roy

E. Pousali Pal

Institution Name :

DASNAGAR GOVERNMENT POLYTECHNIC

(P.O & P.S Dasnagar, Howrah, West Bengal - 711105)

Affiliated to :

West Bengal State Council of Technical & Vocational Education and Skill Development

Certificate of Guidance

This is to certify that this proposal, entitled **Object Detection Car Controlled by Android Application** is a record of bona-fide work, carried out by -

1. Soumen Mishra
2. Nabanita Maity
3. Pousali Pal
4. Triyasa Dey
5. Anirudha Roy

(Student's of Our Institute) under the supervision and guidance through the Dasnagar Government Polytechnic, Howrah, West Bengal - 711105.

Date : 08/05/2025

Signature of the Guide

Shri Partha Sarathi Banerjee

[Lecturer in Computer Science and Technology]

Department of Information Technology & Engineering

DASNAGAR GOVT. POLYTECHNIC

(P.O & P.S Dasnagar, Howrah, West Bengal - 711105)

Date : 08/05/25

Signature of the Guide

Shri Achintya Gopal Mondal

[Workshop Instructor / Instructor (Computer Application)]

Department of Information Technology & Engineering

DASNAGAR GOVT. POLYTECHNIC

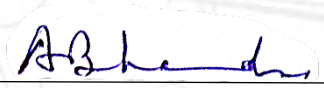
(P.O & P.S Dasnagar, Howrah, West Bengal - 711105)

Certificate of Approval

In my opinion, the report in its present form is in partial fulfilment of all the requirements. It has attained the standard, necessary for submission.

To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Date :



Signature of Head of the Department

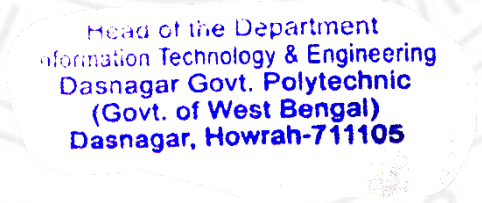
Shri Aniruddha Bhaduri

[Lecturer in Computer Application]

Department of Information Technology & Engineering

DASNAGAR GOVT. POLYTECHNIC

(P.O & P.S Dasnagar, Howrah, West Bengal - 711105)



Certificate by the Institute

Date :

Signature of Principal in Charge

Dr. Manas Kumar Saha

[Lecturer in Mechanical Engineering]

DASNAGAR GOVT. POLYTECHNIC

(P.O & P.S Dasnagar, Howrah, West Bengal - 711105)

Principal-in-charge
Dasnagar Govt. Polytechnic
Govt. of West Bengal
Dasnagar, Howrah- 711105

Acknowledgment

We would like to express our sincere gratitude to Mr. Partha Sarathi Banerjee and Mr. Achintya Gopal Mondal, esteemed faculty members of the Department of Information Technology Engineering at Dasnagar Government Polytechnic, for generously sharing their valuable time and expertise to guide us in undertaking this project.

We are also deeply indebted to all the respected faculty members of the Information Technology Engineering Department for their unwavering support, guidance, and inspiration.

Soumen Mishra

Soumen Mishra Date : 08/05/2025

(D222309148)

Nabanita Maity

Nabanita Maity Date : 08/05/2025

(D222309149)

Anirudha Roy

Anirudha Roy Date : 08/05/2025

(D232410788)

Triyasa Dey

Triyasa Dey Date : 08/05/2025

(D232410789)

Pousali Pal.

Pousali Pal Date : 08/05/2025

(D222309110)

Abstract

This project features a ESP32-based smart car that is wirelessly controlled via an Android application using Wi-Fi and the HTTP protocol and can detect objects in front of it using HC-SR04 Ultrasonic Sensor.

The ESP32 operates as an Access point, receiving commands from the Android application when buttons are pressed. The application sends HTTP GET requests to control the car's movements via Wi-Fi, allowing it to move forward, backward, left, and right. An L293D motor driver IC is used to regulate the BO DC Motors, which are powered by 7.4V batteries, ensuring smooth operation. Additionally, an HC-SR04 ultrasonic sensor is integrated into the system to detect obstacles in the car's path. When an obstacle is detected, the car can automatically stop to avoid collisions, making navigation safer and more efficient. The system operates independently of the internet, as the ESP32 creates its own Wi-Fi Access point, allowing direct communication between the Android application and the car. This project effectively demonstrates *IoT-based wireless control*, real-time responses, and obstacle avoidance. It is a practical implementation of embedded systems and wireless communication, making it ideal for robotics, automation, and smart vehicle applications.

Index

Project Details -

Subject	Major Project
Department	Information Technology Engineering
Project Name	Object Detection Car Controlled by Android Application

Participant Details -

Serial No.	Name of the Participant's	Registration No.	Academic Year
01	Soumen Mishra	D222309148	2022-2025
02	Nabanita Maity	D222309149	
03	Triyasa Dey	D232410789	
04	Pousali Pal	D222309110	
05	Anirudha Roy	D232410788	

Serial No.	Content	Page No.
01	Introduction	08 - 10
02	System Design	11 - 36
03	Working Methodology	37 - 56
04	Images of the Project	57
05	Challenges	58
06	Conclusion	59
07	Future Scope	60 - 65
08	References	66

Introduction

A. The concept of a Object Detection car and its Applications :

The Object Detection Car Controlled by Android Application is a smart, Wi-Fi enabled vehicle designed for collision prevention and reliable navigation. Can be assume as a type of remote-controlled car that uses Wi-Fi technology, Designed for collision prevention and reliable navigation. It allows users to control the car using a smart phone application. The car is equipped with a ESP32 MCU, that connects to a nearby network, measures distance data using an HC-SR04 Ultrasonic Sensor from obstacle's in its path. The car takes immediate corrective actions, to prevent accidents, ensuring, a safer driving experience.

Users can control the car's movements and direction remotely. Wi-Fi controlled cars are also equipped with sensors for real-time feedback. They are popular among hobbyists, researchers, and educators. Wi-Fi controller cars can be used for gaming, education, research and various purposes. They offer a unique combination of technology, innovation, and fun. Wi-Fi controlled cars are an exciting development in the field of Internet of things, Networking, robotics and automation. They have the potential to revolutionize the way we interact with and control machines.

B. Objective :

The primary objective of this project is to design and develop a Wi-Fi controlled smart car using the ESP32 and an Android application. The system enables real-time wireless control of the car's movement through an intuitive mobile interface using HTTP GET requests. Additionally, an HC-SR04 ultrasonic sensor is integrated for obstacle detection, ensuring safe navigation. The project demonstrates IoT-based remote vehicle control, real-time response mechanisms, and embedded system applications.

C. Motivation :

The motivation behind this project is to design and develop a remotely controlled car using Wi-Fi technology. The goal is to create a system that allows users to control the car's movements and direction using a smart phone app. This project combines robotics, automation, and Internet of Things (IOT) technologies to create an innovative and interactive system.

1. **Hands-on IoT and Embedded Systems :** This project provides practical experience in integrating hardware and software for IoT applications.
2. **Wireless Automation :** Eliminating the need for physical controllers, it explores Wi-Fi based remote control solutions.
3. **Obstacle Avoidance :** Implementing an ultrasonic sensor enhances safety and autonomous navigation capabilities.
4. **Scalability :** The project can be expanded for advanced applications like autonomous vehicles, or surveillance robots.
5. **Cost-Effective Learning :** Using affordable components like ESP32 MCU and BO motors makes it accessible for students and hobbyists interested in robotics and IoT.
6. **Skill Development :** Enhances knowledge in Android app development, networking, microcontrollers, and real-time control systems.

D. Problem Statement :

The problem this project solves is the need for a remotely controlled car that can be operated using a smart phone application. In which we can notice the uses of various types of sensors, e.g. - Ultrasonic Sensor. By using we will be aware of the outside environment from inside of the vehicle, and can be safe by automating algorithms. Traditional remote-controlled cars use radio frequency (RF) or infrared (IR) signals, which have limited range and can be interfered with by other devices.

E. Features of the Project :

1. **Longer Range :** Wi-Fi technology allows for a longer range of control, making it possible to operate the car from a distance.
2. **Single User Operation :** Application Delivers a interface having connection to the firebase Realtime Database, to give permission to control the application to a single user at a time.
3. **Interference :** Wi-Fi signals are less prone to interference, providing a more reliable connection between the car and the smart phone application.
4. **Automation :** The project enables automation capabilities, such as programmed routes and obstacle avoidance.
5. **Surveillance :** The car can be equipped with cameras and sensors, enabling real-time surveillance and monitoring.

F. Fields of Applications :

This project has various applications, including **:**

1. **Education :** The project can be used in educational settings to teach robotics, automation, and IOT concepts.
2. **Research :** The car can be used in research applications, such as autonomous vehicle development and robotics research.
3. **Surveillance :** The car can be used for surveillance and monitoring purposes, such as security patrols or environmental monitoring.

G. Project Scope :

The scope of this project includes **:**

1. **Hardware Design :** Designing and building the car's chassis, installing the Microcontroller module, motor driver IC's, Ultrasonic sensor and other necessary components.
2. **Software Development :** Developing a smart phone application that can connect to the car's Wi-Fi module and *send control commands*.
3. **Wi-Fi Connectivity :** Establishing a *reliable Wi-Fi connection* between the car and the smart phone application.
4. **Control System :** Implementing a control system that allows the car to move in different directions based on user input.
5. **Testing and Debugging :** Testing the car's movement and control system, and debugging any issues that arise.
6. **User Interface :** Designing a user-friendly interface for the smart phone application that allows users to easily control the car.
7. **Safety Features :** Implementing safety features such as *emergency stop and obstacle detection.*

H. Project Objective :

This project focuses on designing and developing a remotely controlled car operated via a smartphone app using Wi-Fi technology. The car will be capable of moving in all four directions : forward, backward, left, and right.

It will provide seamless remote control, allowing users to operate it from a distance with ease. The system also integrates an ultrasonic sensor with a microcontroller and motor driver IC to ensure smooth movement and efficient communication between the smartphone and the car. This project aims to enhance wireless control applications, offering a practical and interactive solution for remote vehicle navigation, making it suitable for automation, robotics, and learning purposes.

I. Deliverables :

The project deliverables include :

1. A fully functional Wi-Fi controlled car.
2. A smart phone app that can control the car's movement.
3. Single User at a time Controlling Interface.
4. A detailed report on the project's design, development, and testing.
5. A presentation on the project's objectives, scope, and outcomes.
6. The smartphone app will provide a user-friendly interface for real-time control of the car's movement.
7. The project will implement a reliable Wi-Fi-based communication system between the smartphone and the car for seamless operation.
8. The car will be equipped with an ESP32 microcontroller to process commands and control motor functions efficiently.
9. A stable power supply using rechargeable batteries will be integrated to ensure uninterrupted performance.
10. An optional ultrasonic sensor has been added to detect obstacles and prevent collisions.
11. A well-documented codebase and circuit diagram will be provided for easy replication and future improvements.
12. The system's response time, range, and power consumption will be analyzed for optimization.
13. Recommendations for additional features, such as camera integration, voice control, or GPS tracking, will be discussed.

J. Timeline :

1. Hardware design and building : *4 weeks*
2. Software development : *4 weeks*
3. Wi-Fi connectivity and control system implementation : *2 weeks*
4. Testing and debugging : *1 weeks*
5. User interface design and implementation : *2 weeks*
6. Safety feature implementation : *1 weeks*

7. Project report and presentation : *3 weeks*

System Design

A. Detailed overview of the system architecture :

- **System Architecture -**

Our project consists of an Android application that communicates with a ESP32 over Wi-Fi using the HTTP protocol. Below is the architecture breakdown :

1. System Components :

A) Android Application

- Developed using Android Studio to provide a user-friendly interface.
- Utilizes the OkHttp library to send HTTP requests for seamless communication.
- Features a graphical user interface (GUI) with intuitive buttons for directional control.
- Connects to the ESP32's Wi-Fi access point for direct communication.
- Sends real-time commands to control the car's movement and other functions.

B) ESP32

- Functions as a Wi-Fi Access Point (AP) and an HTTP server, enabling wireless control.
- Listens for incoming HTTP requests from the Android application.
- Processes received commands from application and ultrasonic sensor and executes corresponding actions.
- Sends real-time response to the application.
- Controls various connected components, including LEDs and motors, ensuring smooth operation.

C) Connected Devices

- Includes LEDs, motors, and an ultrasonic sensor (HC-SR04) for obstacle detection.
- Executes actions based on commands received from the Android application.
- Enhances remote control capabilities by providing real-time responses and feedback.

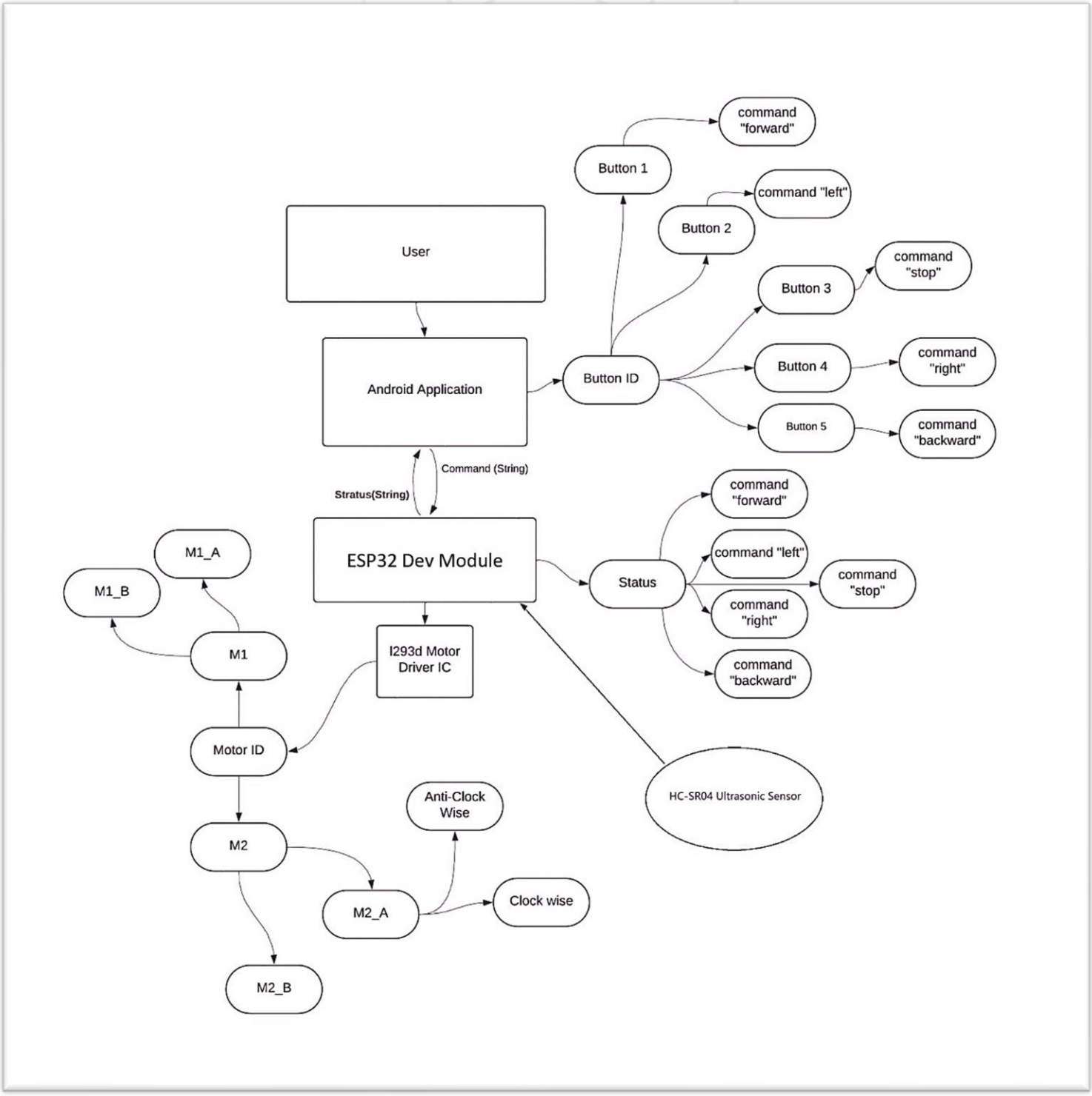
D) L293d Motor Driver IC

- Collects High and low signal from GPIO pins of ESP32.
- Powers the motors to rotate Clock Wise and Counter Clock Wise accordingly.

2. Block Diagram :

A schematic representation illustrating system components as blocks and their interconnections, showing the overall functionality and data flow without detailing internal processes.

Block Diagram of The System is -



3. Workflow :

a) ESP32 Creates Access Point

- The ESP32 is configured to operate as a Wi-Fi Access Point (AP), allowing direct communication without requiring an external router.
- It initializes with a specific IP address (e.g., 192.168.4.1) for the Android device to connect.
- The Android device scans for available Wi-Fi networks and establishes a connection with the ESP32's access point.
- Once connected, the Android device can send control commands to the ESP32.

b) Android App Checks the Available Connections

- The User Checks, is any user is already connected to the car, by pressing the check button.
- If no user is connected to the car, the user clicks the continue button and moves to the next layout.
- User enters their Email Address and Name in the input section, and moves to the layout having control buttons, by pressing continue button.

c) Android App Sends Commands

- The user interacts with the Android application by tapping a button to initiate an action.
- The app, developed in Android Studio, uses the OkHttp library to send an HTTP GET request to the ESP32.
- The request is structured to specify a particular function, such as turning on an LED or controlling a motor.
- Example request: http://192.168.4.1/LED_ON sends an instruction to activate an LED.
- This mechanism allows the app to remotely control the car's movement and other connected devices.

d) ESP32 Receives & Processes Requests

- The ESP32 listens for incoming HTTP requests on its web server.
- Once a request is received, it parses the command from the URL to determine the required action.
- Based on the command, it triggers the corresponding function, such as turning an LED on or making the car move forward.
- It then sends an HTTP response back to the app, confirming execution (e.g., "LED Turned On").

e) Device Executes Action

- The connected hardware components, including LEDs, motors, and sensors, perform the desired task.
- The system ensures smooth real-time execution, enhancing wireless control efficiency.
- The user receives feedback through the app, confirming successful command execution.

4. Data Flow Diagram :

A graphical representation of data movement within a system, showing input, processing, storage, and output, typically using processes, data stores, and data flows.

Data Flow Diagram of The System is –

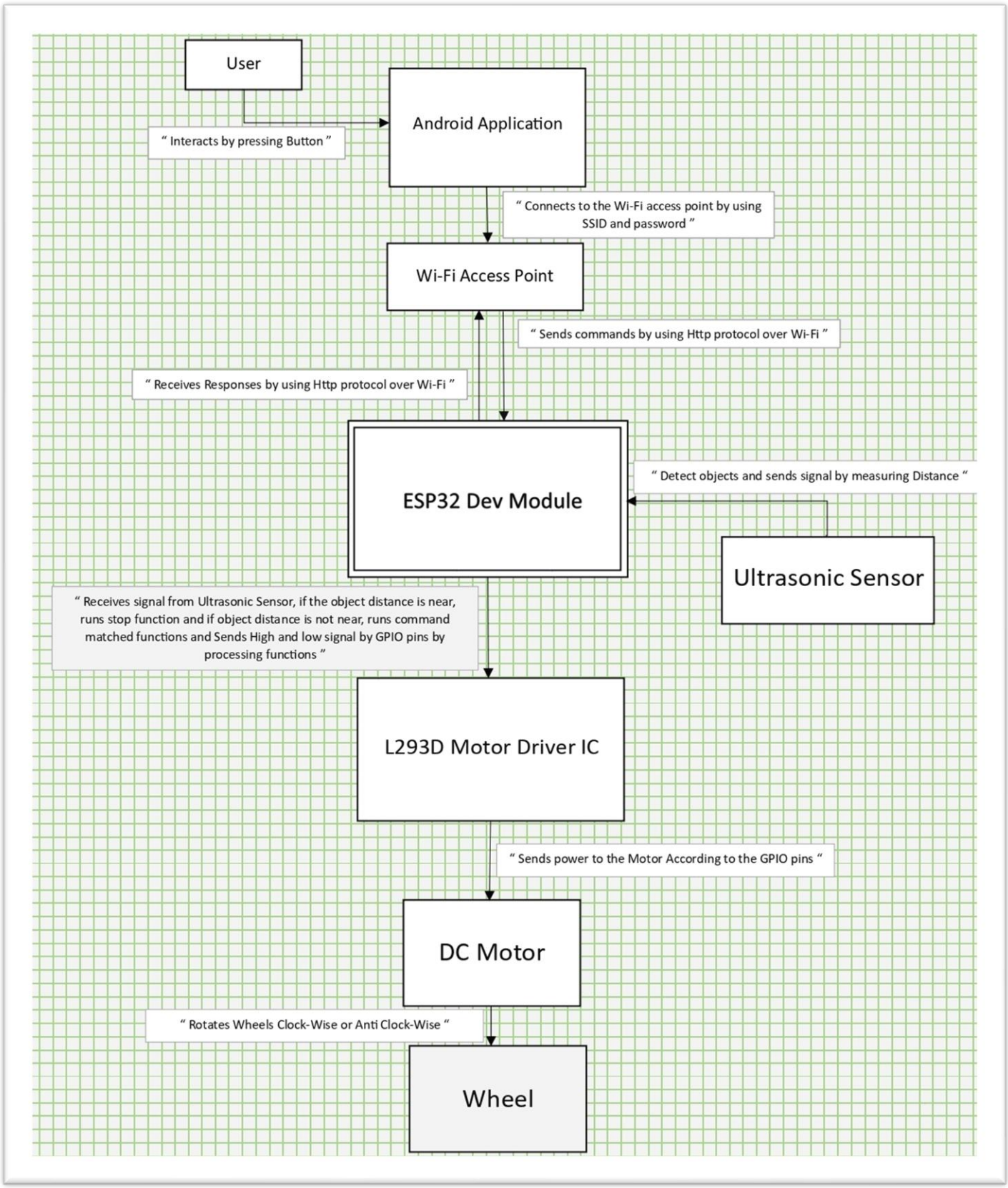


*Figure 2 - **Data Flow Diagram***

5. System Structure Diagram :

A diagram representing database structure, showing entities, attributes, and relationships, useful for designing relational databases with primary and foreign key constraints.

System Structure Diagram is –

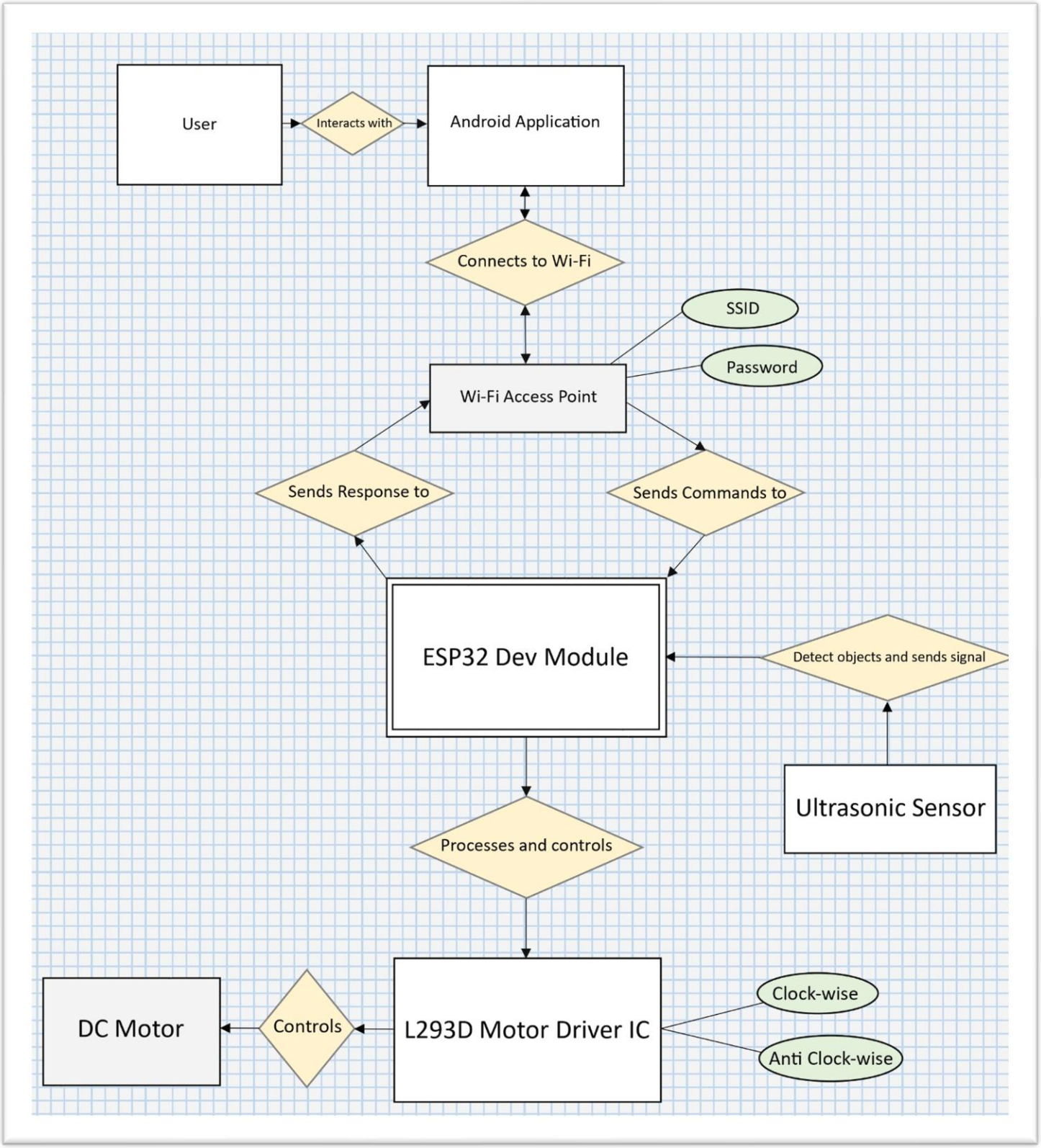


Figure 3 ***– System Structure Diagram***

6. Circuit Diagram :

A visual representation of an electrical circuit using symbols to show components like resistors, wire, IC's and connections between them.

Circuit Diagram of The System is -

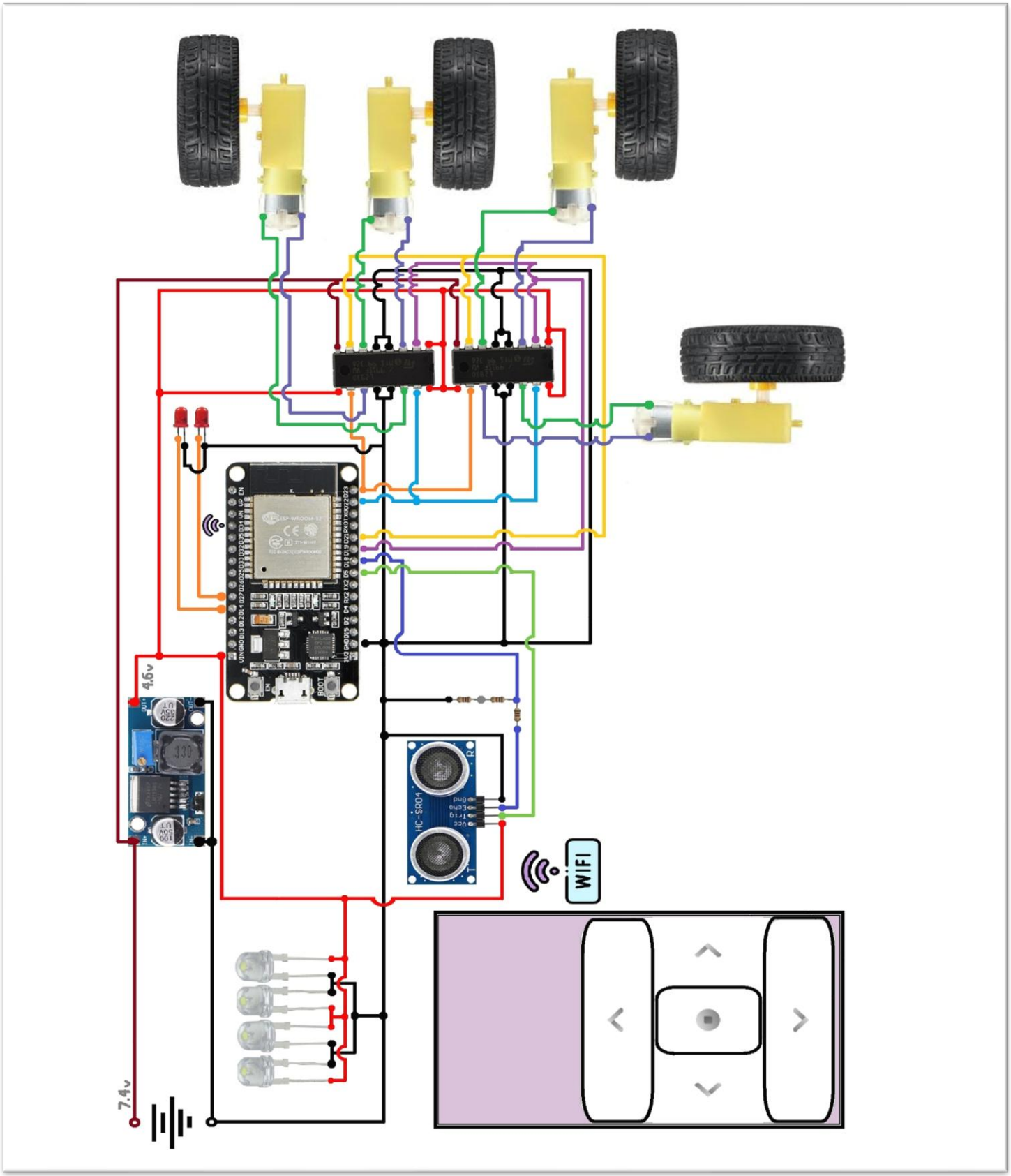


*Figure 4 - **Circuit Diagram***

7. Flowchart :

A diagram representing a process or algorithm using symbols like rectangles (processes), diamonds (decisions), and arrows (flow direction), aiding logical problem-solving.

Flowchart of The System is -

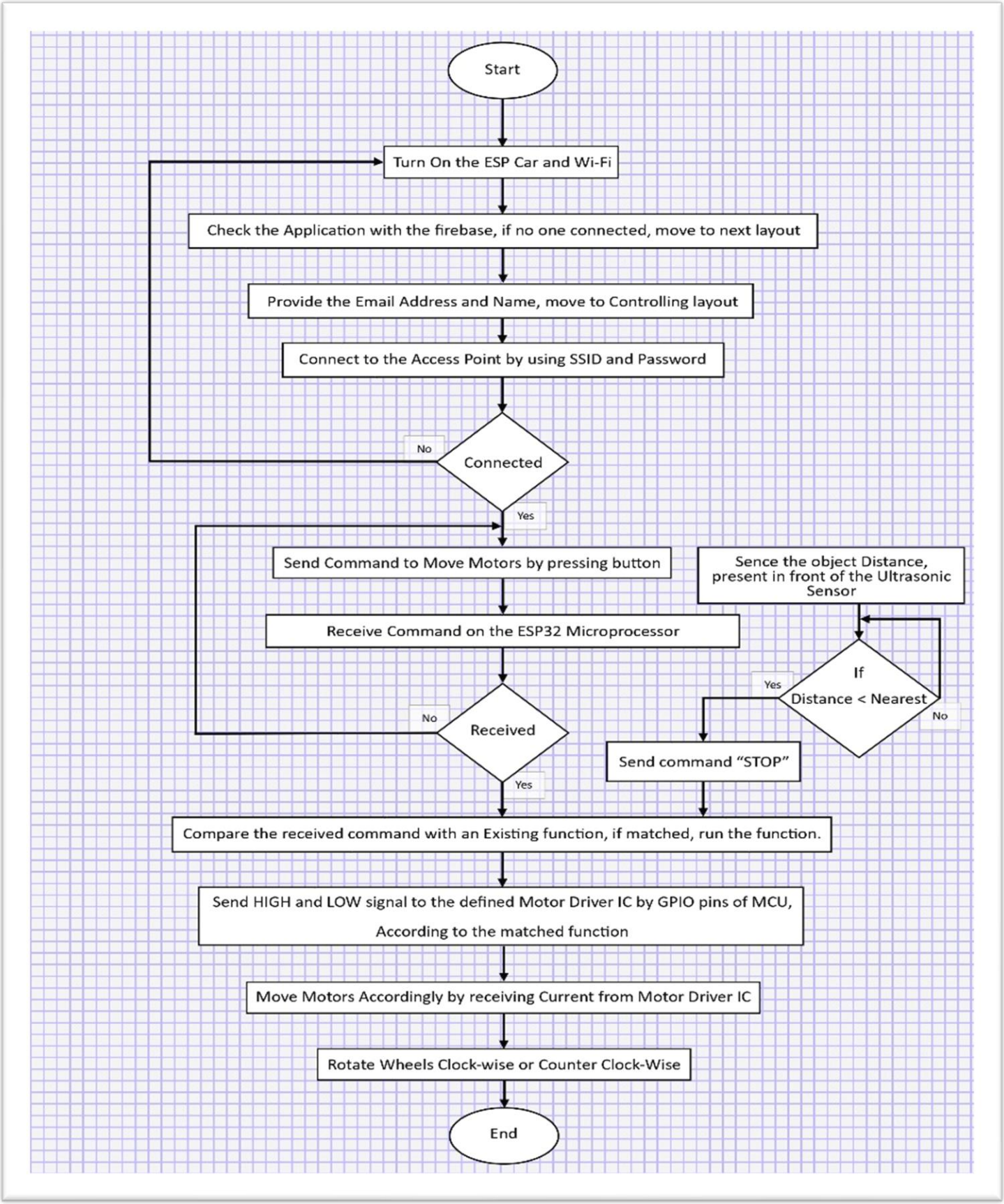


*Figure 5 - **Flowchart***

8. Mechanical Diagram :

Mechanical Diagram of The System is -

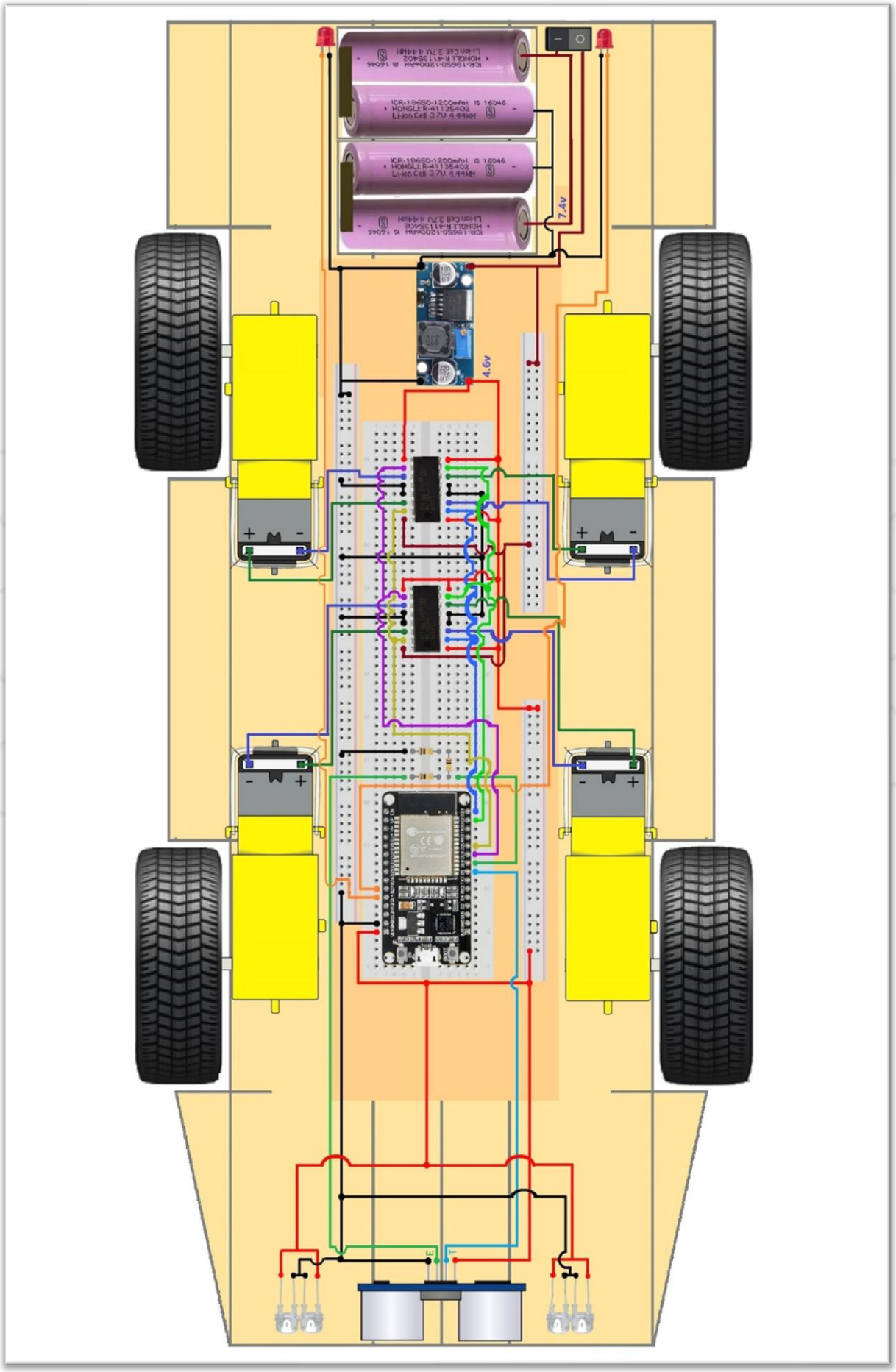


*Figure 6 - **Mechanical Diagram***

9. State Machine Diagram

A graphical representation of a system's states and transitions, used in software and hardware design to model behaviour based on events and conditions.

State Machine Diagram of The System is -

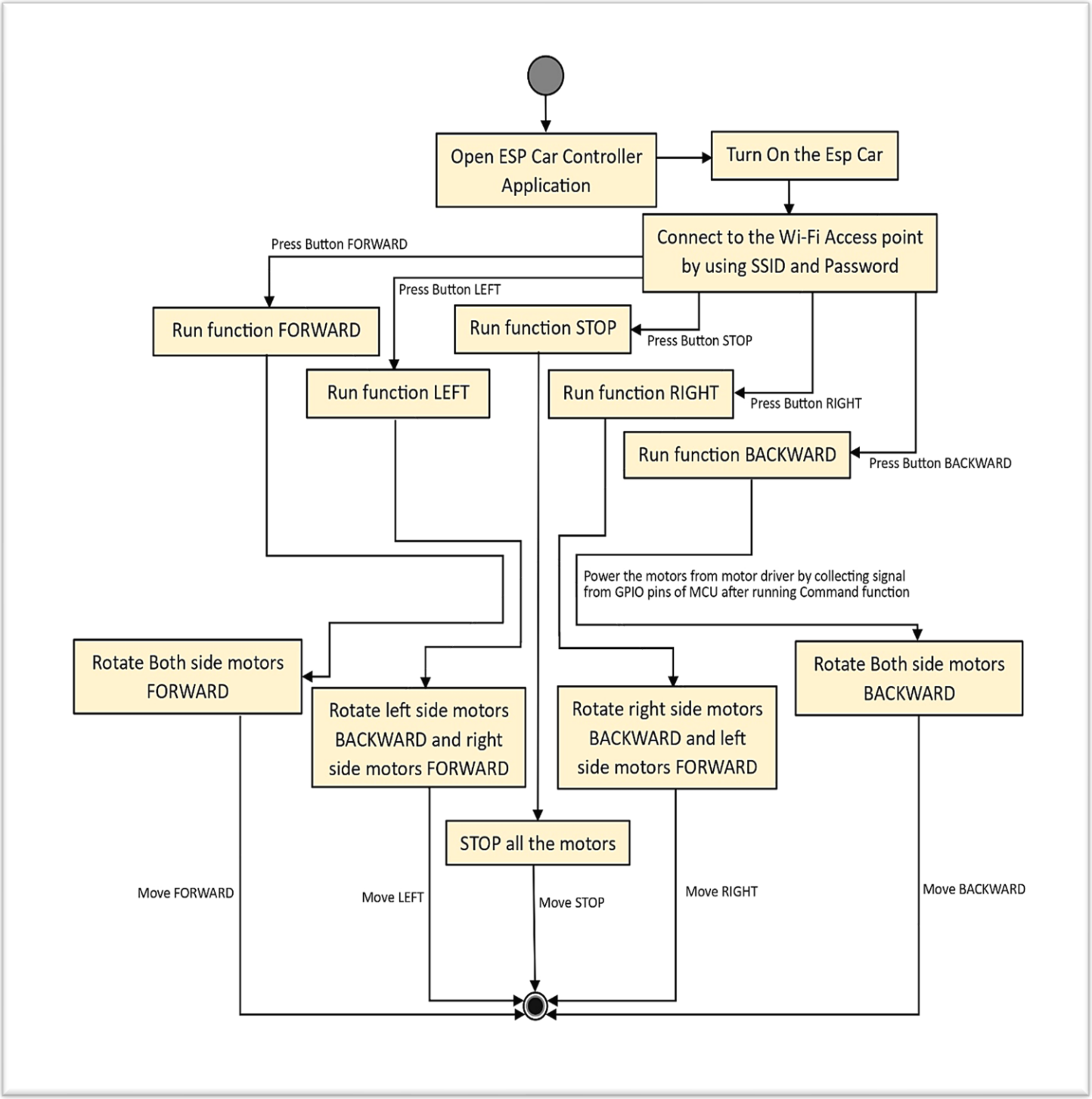


*Figure 7 - **State Machine Diagram***

10. Control System Model :

A mathematical or graphical representation of a control system, showing inputs, outputs, feedback loops, and controllers for analyzing system stability and response.

Control System Model of The System is -

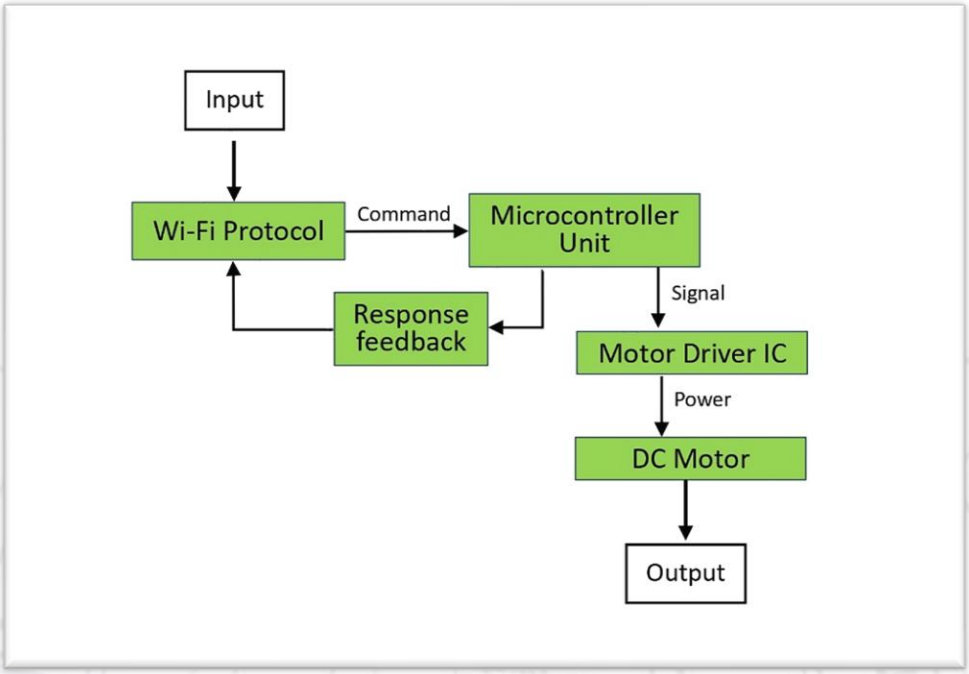


*Figure 8 - **Control System Model***

11. Obstacle Avoidance Algorithm :

A computational method enabling a system (robot/car) to detect and navigate around obstacles using sensors and decision-making logic, ensuring autonomous movement.

Obstacle Avoidance Algorithm of The System is -

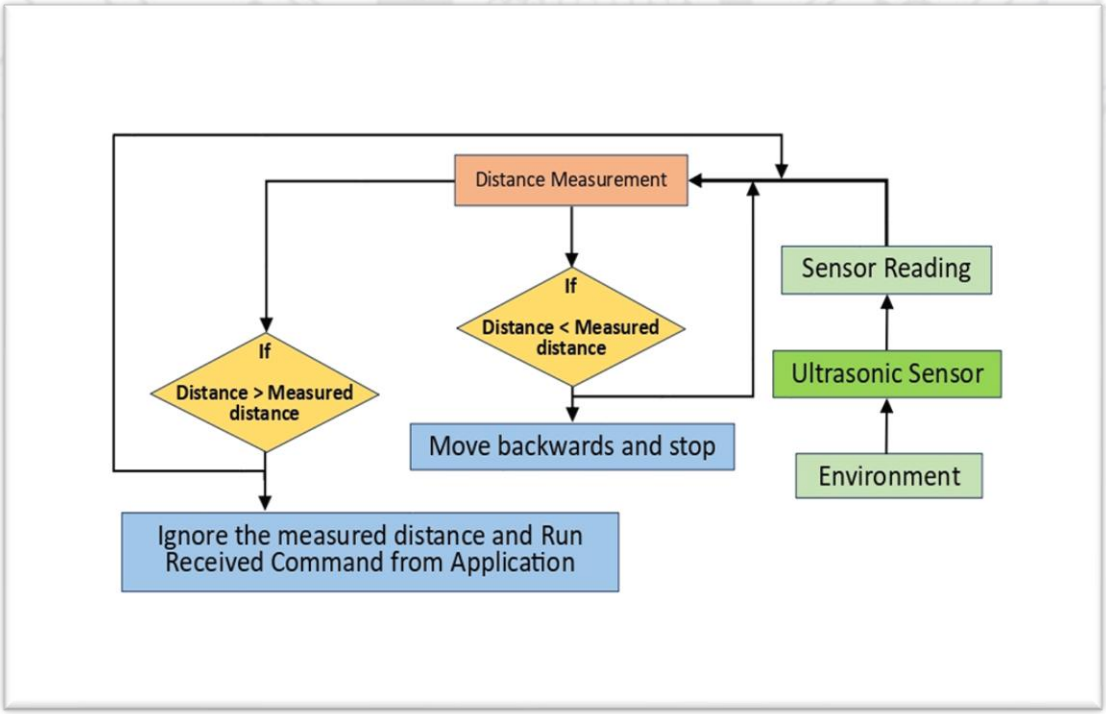


*Figure 9 - **Obstacle Avoidance Algorithm***

- **Communication Protocols -**

1. Wi-Fi Protocol :

The ESP32 microcontroller utilizes the Wi-Fi protocol to establish a wireless connection, enabling communication with a smartphone application by using HTTP protocol.

The ESP32 can operate in three primary Wi-Fi modes :

- **Station Mode (STA)** – Connects to an existing Wi-Fi network for internet access and remote control.
- **Soft Access Point Mode (AP)** – Creates a local Wi-Fi network, allowing direct communication between the smartphone and ESP32.
- **Dual Mode (AP+STA)** – Simultaneously connects to an existing network while hosting its own access point.

This protocol facilitates the transmission of control commands from the smartphone app to the ESP32, enabling real-time operation of the car.

2. System Requirements :

- **Wi-Fi Network**

  A. A stable Wi-Fi network is necessary for seamless communication between the ESP32 and the smartphone application.

  B. If an external router is unavailable, the ESP32 can act as an access point (AP) to establish a direct connection.

- **Smartphone App**

  A. A dedicated smartphone application (E.g. Android) is required to send commands to the ESP32.

  B. The app communicates using HTTP requests over Wi-Fi.

  Features include :

  a. UI buttons for directional control (Forward, Reverse, Left, Right, Stop).

- **Power Supply**

  A stable power supply is essential to ensure uninterrupted operation of the car.

  Possible power sources :

  a. Lithium-ion batteries (18650 cells) – Provide lightweight, rechargeable power.

  b. LiPo battery packs – Deliver high current for motor operation.

  c. DC to DC Step-down voltage regulators (E.g. LM2596) – Convert battery voltage to required ESP32 operating voltage (3.3V) and motor driver voltage (6V–12V).

B. Description's of the Hardware components :

1. ESP32 Dev Module V1

The **ESP32 Dev Module V1** is a powerful and versatile Wi-Fi and Bluetooth-enabled microcontroller module, designed for IoT and embedded applications. It features a dual-core processor, more GPIO pins, and enhanced connectivity compared to the ESP8266. With built-in Wi-Fi and Bluetooth capabilities, it is ideal for smart home systems, industrial automation, and wireless sensor networks. It supports multiple programming environments, including the *Arduino IDE, Micro-Python, and ESP-IDF*.

*Figure 10 - **ESP32 Derv Module V1***

- **Wi-Fi Support :** Supports 802.11 b/g/n Wi-Fi, operating in Station (STA), Access Point (AP), and dual-mode (STA+AP) for flexible connectivity.
- **Operating Voltage :** Operates at 3.3V, with a 5V USB input for power and onboard voltage regulation.
- **Bluetooth Support :** Features Bluetooth 4.2 and BLE (Bluetooth Low Energy) for wireless communication with smartphones and other BLE-enabled devices.
- **Programming :** Can be programmed using Arduino IDE, Micro-Python, ESP-IDF, and Lua scripting for versatile development.
- **Flash Memory :** Typically comes with 4MB to 16MB of flash memory, allowing for larger program storage.
- **GPIO Pins :** Offers 36 GPIO pins, supporting PWM, ADC (analog input), DAC (analog output), touch sensing, I^2C, SPI, UART, and more.
- **Communication Protocols :** Supports UART, SPI, I2C, I2S, CAN, and Ethernet, making it highly compatible with various peripherals and sensors.
- **Processing Power :** Features a dual-core 32-bit Xtensa LX6 processor, operating at 160MHz to 240MHz, providing high-speed performance for IoT applications.

Additional Features :

- Multiple ADC Channels (12-bit resolution) for accurate analog readings.
- Two 8-bit DACs for generating analog signals.
- Capacitive Touch Inputs for touch-sensitive applications.
- Deep Sleep and Low Power Modes for energy-efficient IoT projects.
- Used in IOT devices, home automation, wireless sensors, and smart systems due to its low cost and ease of use.

A. Internal Architecture (ESP32 SoC)

The core of the ESP32 Dev Module V1 is the ESP32 System-on-Chip (SoC) from Espressif, which offers higher processing power, more memory, dual-core capability, and advanced connectivity options.

1. CPU Core

- Features a dual-core Xtensa LX6 processor, capable of running tasks in parallel.
- Clock frequency: Default 160 MHz, can be boosted up to 240 MHz.
- Supports multi-threading and real-time processing with FreeRTOS.
- On-board antenna (PCB or ceramic) and external antenna support for extended range.

2. **Memory**

- **RAM :**

  520 KB SRAM for program execution and data storage.

  8 KB RTC (Real-Time Clock) SRAM for low-power applications.

- **ROM :**

  Stores bootloader and built-in library functions for startup and essential operations.

- **Flash Storage :**

  Externally connected SPI flash, commonly 4MB to 16MB, for program and data storage.

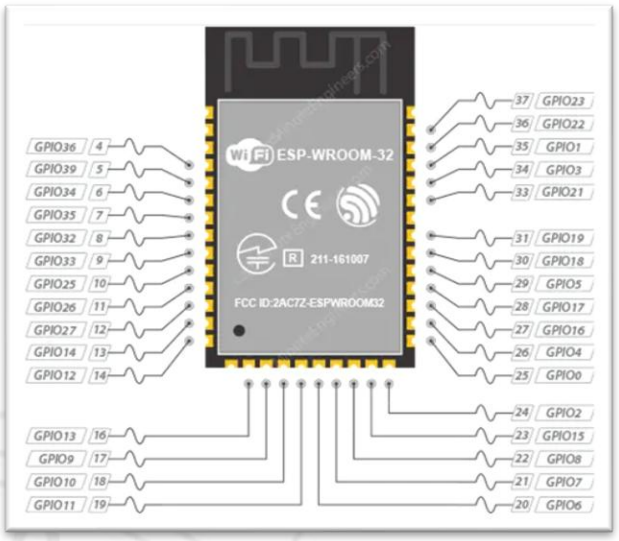


*Figure 11 – **Esp32 Dev Module V1 Pinout reference***

3. **Wi-Fi & Bluetooth**

- Integrated 802.11 b/g/n Wi-Fi radio, supporting Station (STA), Access Point (AP), and STA+AP modes.
- Bluetooth: Supports Bluetooth v4.2 (Classic + BLE) for wireless communication with peripherals.

4. **GPIO and Peripherals**

- Up Requires 3.3 V power supply.
- PWM support on all digital GPIOs.
- Analog-to-Digital Conversion (ADC):
- 18-channel 12-bit ADC (much better than ESP8266's single 10-bit ADC).
- Digital-to-Analog Conversion (DAC):
- 2-channel 8-bit DAC for generating analog signals.
- Touch Capacitive Inputs on specific GPIOs for touch-based applications.

5. **Power Management**

- Operates at 3.3V, with onboard voltage regulation from 5V USB input.
- Multiple power-saving modes, including Deep Sleep and ULP (Ultra Low Power) mode for battery-powered applications.

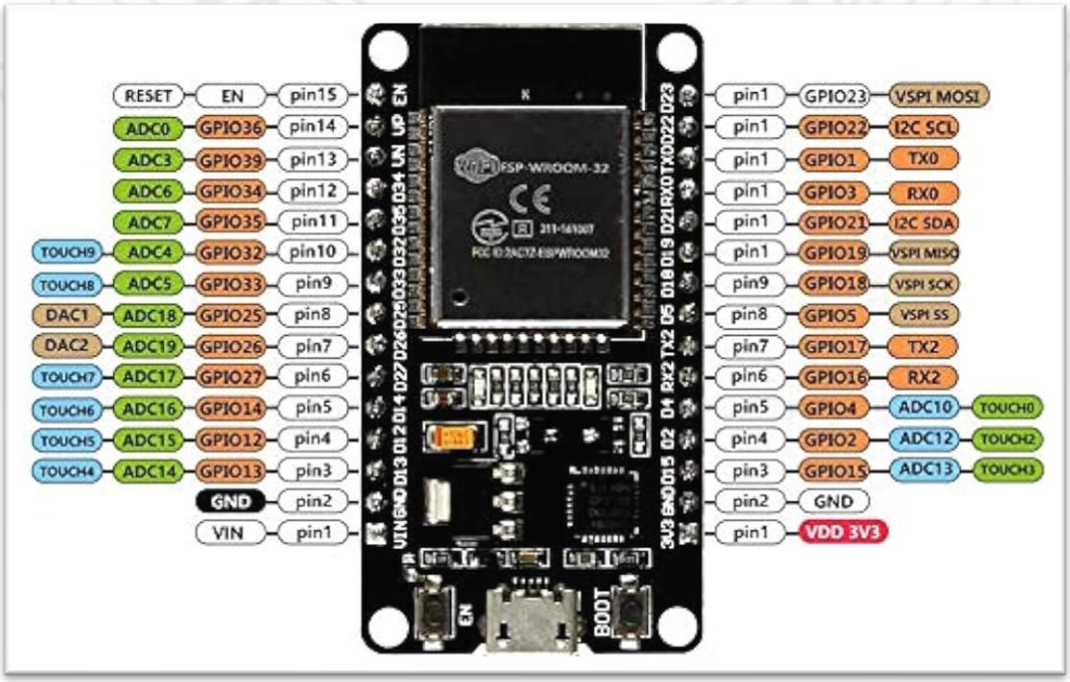


*Figure 12 – **ESP32 Dev Module V1 Pin Diagram***

B. ESP32 Dev Module V1 Size and Layout

While exact dimensions can vary slightly by manufacturer, the common ESP32 Dev Kit v1 form factor is approximately:

- Length: ~48–50 mm
- Width: ~25–26 mm
- Height: ~13 mm (including the height of pin headers and USB connector)

Pin headers are typically spaced at 2.54 mm (0.1 inch) pitch. There are usually 30 pins in total (15 on each side), though some boards may have slightly different pin counts.

C. ESP32 Dev Module V1 Circuit/Board Design

The *ESP32 Dev Module V1* integrates several key components to support development and programming around the ESP32 microcontroller, making it more powerful and versatile :

1. ESP32-WROOM-32 Module

The core of the board is the ESP32-WROOM-32 module, which includes:

- ESP32 SoC (dual-core Xtensa LX6 processor).
- Onboard SPI flash chip (commonly 4MB to 16MB) for program and data storage.
- Integrated PCB or ceramic antenna for Wi-Fi and Bluetooth communication.
- Metal shielding to reduce electromagnetic interference (EMI).

2. USB-to-Serial Converter

- CP2102, CH340G, or FTDI FT232RL acts as the USB-to-UART bridge for programming and serial communication.
- Facilitates easy flashing of firmware using USB without an external adapter.
- Auto-reset circuit allows seamless programming by toggling EN (Enable) and IO0 (Boot Mode) using DTR/RTS signals.

3. Voltage Regulator

- Converts 5V from the USB port down to the required 3.3V for the ESP32.
- Provides stable power output to support high current draw when using Wi-Fi and Bluetooth.
- Common low-dropout (LDO) regulators used :

    a. AMS1117-3.3V (basic).

    b. ME6211 or AP2112 (low-power and efficient).

4. Reset and Flash Circuitry

Includes RESET (EN) and BOOT (IO0) buttons for manual flashing if required.

- Pull-up and pull-down resistors on important pins:

    a. GPIO0 (Boot Mode Selection) – Used to enter flash mode.

    b. EN (Enable) – Resets the ESP32 when toggled.

    c. GPIO2, GPIO15, GPIO5 – Configured to ensure proper boot-up.

2. L293D Motor Driver IC

The *L293D* Motor Driver IC is a popular H-bridge motor driver IC designed to control the speed and direction of two DC motors simultaneously. It enables bidirectional control by allowing current to flow in either direction through the motors, which makes it suitable for robotics and motor-based projects.

- **H-Bridge Configuration :** The L293D uses an H-Bridge circuit configuration that allows for bidirectional motor control. This means it can control the rotation direction of the motors by toggling the input signals.
- **Dual Motor Control :** The IC has two H-Bridge circuits, which enable control of two motors independently. Each motor can be rotated in either forward or reverse directions.
- **Operating Voltage :** The IC operates between 4.5V and 36V, making it suitable for low and medium-power motors.
- **Current Capacity :** It can supply a maximum output current of *600mA per channel* (with peak currents up to 1.2A), which is sufficient for small DC motors.
- **Enable Pins :** The L293D has enable pins for each H-Bridge, which allows the user to enable or disable the motor drivers easily. By controlling the enable pins, the motors can be turned ON or OFF.

*Figure 13 - **L293D Motor Driver IC***

- **Diodes for Protection :** Built-in diodes protect the IC from back electromotive force (EMF), a common issue when dealing with motors.
- **Thermal Shutdown :** The IC is equipped with thermal shutdown to prevent damage due to overheating.
- **Control Inputs :** It requires 4 input pins to control the two motors, with logic high and low signals determining the motor's direction.

A. Internal Architecture

The L293D is a quadruple half-H driver, meaning it can drive up to four half-bridges or, more commonly, two full H-bridges. In simpler terms, it can control two DC motors (one motor per H-bridge pair) or one stepper motor. Key points:

1. H-Bridge Structure

- Each pair of outputs forms an H-bridge: a configuration of transistors that can drive current in either direction through a motor winding.
- By controlling the input signals, you can make the outputs go HIGH or LOW, thus driving a motor forward or reverse.

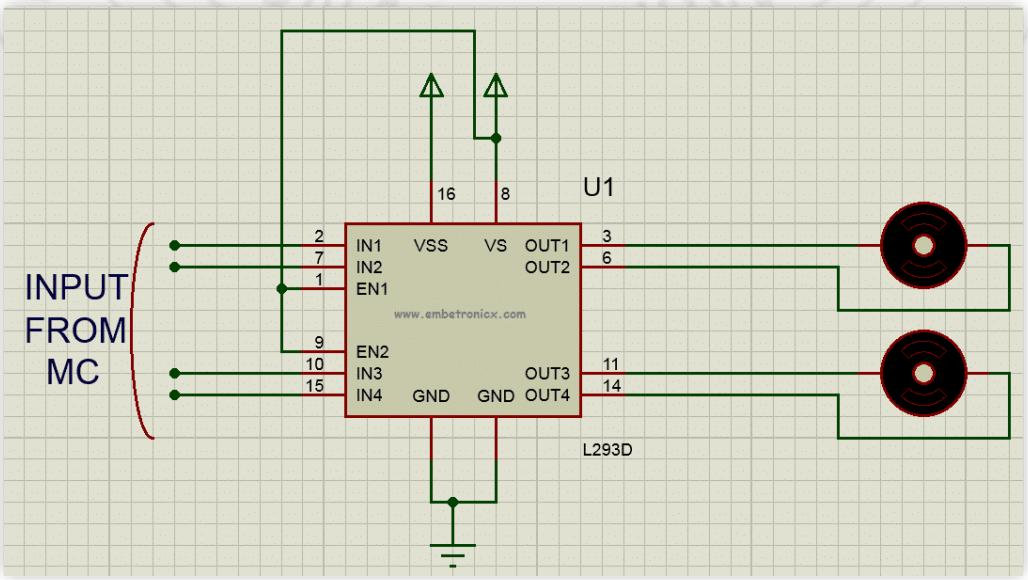


*Figure 14 - **L293D Motor Driver IC Working Operation***

2. **Integrated Clamp Diodes**
    - The "D" suffix (L293D) indicates built-in flyback diodes to protect the IC from voltage spikes generated by inductive loads (motors).
    - This eliminates the need for many external diodes required in older variants (like the original L293).
3. **Dual Supply Pins**
    - **VCC1 (Pin 16) :** Logic supply (typically +5 V). Powers the internal logic and input circuitry.
    - **VCC2 (Pin 8) :** Motor supply (can be from +4.5 V up to +36 V). Powers the H-bridge output stage.
4. **Enable and Input Control**
    - Two enable pins (one for each motor pair): Enable 1,2 (Pin 1) and Enable 3,4 (Pin 9).
    - Four input pins (one for each half-bridge), which typically connect to your microcontroller (e.g., Arduino, etc.).
5. **Current Capability**
    - Up to about 600 mA of continuous output current per channel (with proper heat dissipation).
    - Peak output current of up to ~1.2 A (short bursts).

Internally, each channel has:

- A logic input stage that interprets signals from the microcontroller.
- A power output stage (the transistor H-bridge) that drives the motor with the higher voltage from VCC2.
- Protection diodes and some internal thermal shutdown logic.

B. Physical Size (DIP-16 Package)

The **L293D** in a 16-pin DIP (Dual In-line Package) has a standard form factor with pins spaced at 2.54 mm (0.1 in) pitch. Typical dimensions (approximate, may vary by manufacturer) are:

- **Body Length :** ~19.3 mm to 20.5 mm
- **Body Width :** ~6.0 mm to 7.1 mm
- **Body Thickness :** ~3.0 mm to 4.0 mm
- **Pin Pitch :** 2.54 mm (0.1 in)

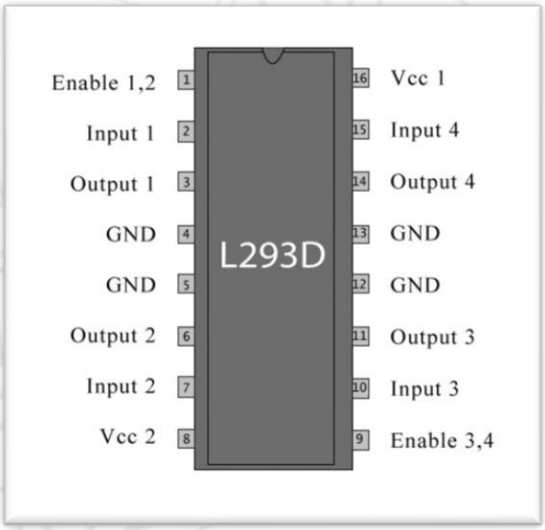


*Figure 15 - **L293D Motor Driver IC Pin Diagram***

C. Typical Application Circuit

A simple example for driving two DC motors:

1. VCC 1 (Pin 16) = +5 V (from your microcontroller's supply).
2. VCC 2 (Pin 8) = +6 to +12 V (for typical DC motors, can be higher if needed).
3. Enable 1, 2 (Pin 1) and Enable 3, 4 (Pin 9) connected to +5 V (or controlled by PWM pins if you want speed control).
4. Inputs (Pins 2, 7, 10, 15) connected to microcontroller outputs.
5. Outputs (Pins 3, 6, 11, 14) connected to the two DC motors.
6. Ground (pins 4, 5, 12, 13) tied to system ground.

Example signals:

- To spin motor A forward, set Input 1 = HIGH, Input 2 = LOW.
- To spin motor A reverse, set Input 1 = LOW, Input 2 = HIGH.
- To stop motor A, either disable Enable 1, 2 or set both inputs to LOW.

D. Key Electrical Characteristics

1. Supply Voltage

- **VCC1 (logic)**: Typically +5 V (can go from ~4.5 V to 7 V).
- **VCC2 (motor)**: 4.5 V to 36 V (check datasheet for max rating).

2. Output Current

- Up to 600 mA continuous per channel.
- 1.2 A peak (non-repetitive, short duration).

3. Thermal Protection

- The L293D includes thermal shutdown to protect against overheating.

4. Operating Temperature

- Typically 0°C to +70°C for commercial versions (check datasheet for exact range).

3. BO DC Motor

The BO DC motor is a *lightweight, geared motor operating on 3V-12V*. It's ideal for robotics, offering sufficient torque and direct shaft attachment compatibility.

- **Type**: BO DC Motor (Battery Operated DC Motor).
- **Functionality**: A small geared motor ideal for driving wheels in robotics applications. Operates on low voltage and provides sufficient torque for lightweight projects.
- **Voltage Range**: Typically operates between 3V to 12V.
- **Output Shaft**: Extended to attach wheels directly.
- **Material**:
- **Motor Housing**: High-quality plastic for lightweight yet durable construction.
- **Gears**: Usually nylon or similar durable material for smooth operation.
- **Dimensions**: Approx. 7cm x 3cm x 2.5cm.
- **Applications**: Suitable for small robot cars, automated toys, and similar projects.

*Figure 16 - **BO DC Motor***

A. Working Process

1. Brushed DC Motor

- Inside the yellow plastic gearbox is a small brushed DC motor. When DC voltage is applied to the motor terminals, current flows through the brushes and commutator, causing the motor shaft to spin.

2. Gear Reduction

- The motor shaft is connected to a series of plastic gears. These gears reduce the motor's high speed down to a lower, more useful output speed while increasing torque.
- Common gear ratios range around *1:48 or 1:120*.

3. **Driving a Wheel**

- The output shaft of the gearbox extends through the plastic housing, where a wheel (usually press-fit or secured via a small screw) is attached.
- By controlling the polarity (direction) and magnitude (voltage) of the DC supply, you can make the wheel spin forward, backward, or at various speeds.

4. **Voltage Range**

- Typically rated for about *3 V to 6 V* (some can handle up to 9 V, but 3 - 6 V is most common).
- Typical speed examples: ~100 - 120 RPM at 3 V, and ~150 - 200 RPM at 6 V (exact values vary).

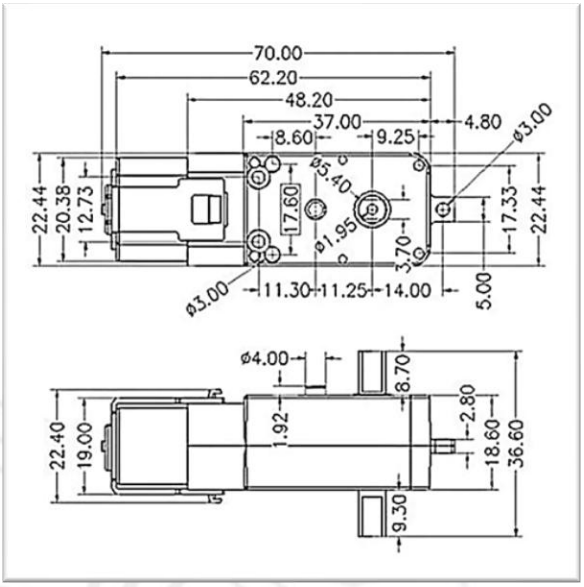


*Figure 17 - **BO DC Motor Dimensions***

B. Size and Dimensions

Though dimensions can differ slightly by brand, here are approximate measurements for the yellow gearbox motor plus wheel

1. **Motor + Gearbox Assembly**

- **Length :** (end of motor can to tip of gearbox shaft) ~65–70 mm
- **Gearbox Section :** ~22–25 mm thick × ~28–30 mm wide × ~30–35 mm long
- **Motor Can :** ~25 mm diameter × ~30–35 mm length (inside the plastic casing)
- **Shaft Diameter :** Often ~5–6 mm for the output shaft (where the wheel attaches).

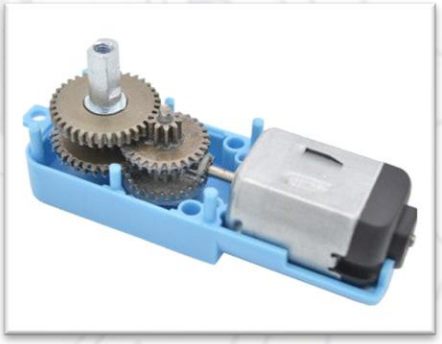


*Figure 18 - **BO DC Motor gear system***

C. Materials

1. **Motor Housing :**
   - The internal motor is typically a metal can with copper windings and carbon/metal brushes.
2. **Gearbox :**
   - Usually made from *ABS* or *nylon* plastic, which is lightweight, durable enough for small loads, and inexpensive.
   - Gears inside are typically plastic (nylon or POM) to keep weight and cost down.

4. BO Wheel

The rubberized wheel, with a plastic hub and treaded surface, ensures strong traction and stability. It measures 65mm in diameter and is perfect for robotic applications.

- **Design :** Rubberized wheel with a tread pattern for improved grip and stability.
- **Connection :** Attached to the motor shaft via a coupling mechanism.
- **Hub :** Plastic center for lightweight construction.
- **Outer Layer :** Rubber coating for better traction and durability.
- **Diameter :** Around 65mm (varies slightly depending on the model).
- **Thickness :** Typically 25mm-30mm.

A. Working Process

*Figure 19 - **BO Wheels***

1. **Basic Role**
   - The wheel is a passive mechanical component attached to the motor shaft.
   - When the DC gear motor turns, it rotates the wheel. This rotational motion is converted to linear motion on the ground via friction between the wheel's tire and the surface.
2. **Coupling to the Motor**
   - Usually, the wheel press-fits or snaps onto the gearbox's output shaft.
   - Some designs use a D-shaped or slot-based mounting hole so the wheel does not slip on the shaft.
3. **Friction and Traction**
   - The rubber (or rubber-like) outer tire provides grip on smooth floors, enabling the wheel to push or pull a lightweight robot chassis without slipping excessively.
4. **Directional Control**
   - By reversing the polarity or applying PWM signals to the motor (via a motor driver), the wheel can be driven forward, backward, or at variable speeds.
   - Steering in a 2WD or 4WD robot is typically done by controlling each side's wheels independently (differential drive).

B. Typical Size and Dimensions

Exact measurements vary slightly, but common "BO wheels" have:

- **Wheel Diameter:** ~60–65 mm (some variants up to ~70 mm).
- **Tire Width (Tread):** ~25–28 mm.
- **Hub Diameter:** The plastic hub (inner section) is usually around 35–40 mm in diameter, but often it's not fully visible because the rubber tire extends beyond it.
- **Mounting Hole:** Shaped or slotted to fit the small yellow gearbox shaft (often around 5–6 mm in diameter).

*Figure 20 - **BO Wheels Dimension***

C. Materials

- **Plastic Hub**
  1. Typically ABS or a similar sturdy plastic.
  2. The hub is molded to fit the motor's output shaft securely.
  3. Spokes or a solid disc shape help reduce weight while maintaining rigidity.
- **Tire/Tread**
  1. Commonly rubber or a thermoplastic rubber (TPR).
  2. Provides friction for traction on smooth surfaces.
  3. Some cheaper variants may use harder plastic tires, but rubbery tires are more common for robotics kits.

5. Lithium-Ion Battery

Lithium-ion battery has a nominal voltage of *3.7V* and a capacity of *2000mAh*, making it ideal for powering small electronics and robotics projects. With an energy rating of *7.4Wh*, it delivers reliable performance over extended durations. Its compact size and lightweight design ensure portability, while the high energy density makes it suitable for devices requiring efficient power storage. This battery is rechargeable, supports multiple charge cycles, and can be integrated with protection circuits for safe operation. It's an excellent choice for applications like DIY robots, IoT devices, and portable gadgets.

*Figure 21 - **Li-ion Battery***

Battery Components

- **Type :** Rechargeable battery with high energy density.
- **Chemistry :** Lithium-ion (Li+) moves between electrodes during charge and discharge.
- **Shape :** Available in cylindrical, prismatic, or pouch form factors.
- **Voltage :** Nominal voltage is usually 3.6V to 3.7V per cell. Battery Operation
- **Range :** Typically varies between 500mAh to 5000mAh or higher for single cells.
- **Energy Density :** Approximately 150-250 W/kg, depending on the specific model and chemistry.
- **Long Lifespan :** Typical lifespan of 300-500 cycles, depending on usage.
- **Fast Charging:** Supports efficient recharging, often in under 2-3 hours.
- **Consistent Voltage :** Delivers steady power output throughout usage.

A. Working Process.

1. Lithium-Ion Chemistry

- A Li-ion cell stores and releases energy through the *intercalation* (insertion) and *de-intercalation* (extraction) of lithium ions between the *cathode* and *anode*.
- The *cathode* (positive electrode) typically contains a lithium metal oxide (e.g., $LiCoO_2$, $Li(NiMnCo)O_2$, etc.), while the *anode* (negative electrode) is usually made of *graphite*.

2. Charge and Discharge

- **Charging :** When the cell is charged, an external power source forces lithium ions to move from the cathode to the anode, where they are stored in the layered graphite structure. Voltage rises from about 3.0–3.2 V (near empty) up to about *4.2 V* (fully charged).
- **Discharging :** When connected to a load, lithium ions flow back from the anode to the cathode, creating an electric current through the external circuit. Voltage decreases from 4.2 V back down to around 3.0 V.

3. Nominal Voltage and Capacity

- **Nominal Voltage :** 3.7 V is the typical rating for a single Li-ion cell.
- **Capacity :** 1650–2000 mAh (milliamp-hours) indicates how much current the cell can supply over one hour (e.g., 1650 mA for 1 hour or 825 mA for 2 hours, etc.).
- **Energy (Wh) :** ~7.4 Wh (Watt-hours) is the product of nominal voltage (3.7 V) and capacity (2.0 Ah, or 2000 mAh).

4. **Protection Circuit**

- Many Li-ion cells (especially standalone "bare" cells) rely on an external protection circuit or battery management system (BMS) to prevent overcharge, over-discharge, and short-circuit.

B. Size and Shape

Although 3.7 V, 1650–2000 mAh Li-ion cells come in various formats, two common types are :

1. **Cylindrical**

- **Name :** 18650 refers to ~18 mm diameter × 65 mm length. Actual measurements can be ~18.2 - 18.6 mm in diameter and ~65 - 66 mm in length.
- **Capacity Range :** Typically 1500–3600 mAh, so a 1650–2000 mAh cell could be an older or mid-range 18650.
- **Weight :** ~35–45 g, depending on capacity and internal construction.

2. **Prismatic or Pouch Cell**

- **Shape :** Flat, rectangular or "pouch" style.

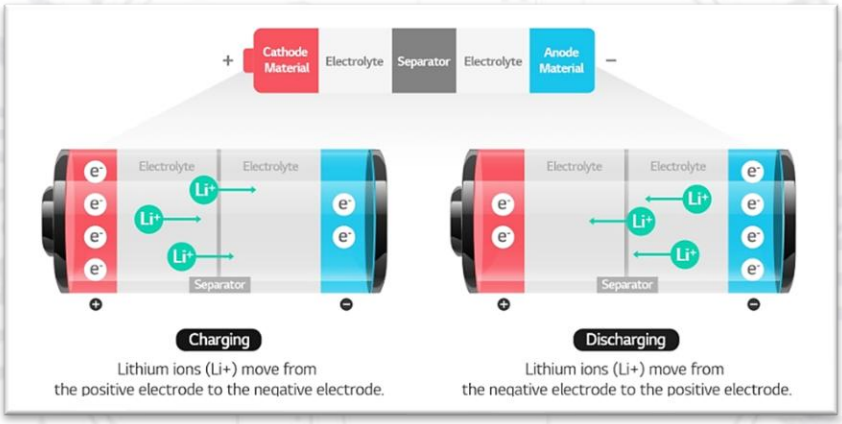


Figure 22 ***Li-ion Battery Structure***

- **Capacity :** Also in the 1000–3000 mAh range for small to mid-sized cells.
- **Weight :** Similar to cylindrical cells for the same capacity.

C. Materials

1. **Cathode (Positive Electrode)**

- Commonly *Lithium Cobalt Oxide ($LiCoO_2$),* or a variant such as *Lithium Nickel Manganese Cobalt Oxide (NMC),* or *Lithium Nickel Cobalt Aluminium Oxide (NCA).*
- Coated onto an aluminium foil current collector.

2. **Anode (Negative Electrode)**

- Typically graphite (carbon-based).
- Coated onto a copper foil current collector.

3. **Electrolyte**

- A lithium salt (e.g., Li-PF_6) dissolved in an organic solvent blend (e.g., ethylene carbonate, dimethyl carbonate).
- Facilitates the movement of lithium ions between anode and cathode..

6. HC-SR04 Ultrasonic sensor

Ultrasonic sensors work by emitting high-frequency sound waves, typically beyond the range of human hearing (above 20 kHz), and measuring the time it takes for these waves to bounce back from objects in their environment.

*Figure 23 - **HC-SR04 Ultrasonic Sensor***

Components :

- **Transducer :** This is the heart of the ultrasonic sensor, responsible for emitting and receiving sound waves. It converts electrical energy into sound waves and vice versa.
- **Microcontroller :** This is the brain of the sensor, which processes the signals received from the transducer and calculates the distance.
- **Power Supply :** This provides the necessary power to the sensor.
- **Emission of Sound Waves :** The transducer emits high-frequency sound waves (typically in the range of 20 kHz to 40 kHz) into the environment.
- **Reflection :** When these sound waves encounter an object, they bounce back (reflect) and return to the transducer.
- **Detection :** The transducer detects the reflected sound waves and converts them back into electrical signals.
- **Signal Processing :** The microcontroller processes the electrical signals and calculates the time difference between the emitted and received sound waves.
- **Level Measurement :** Ultrasonic sensors are used to measure liquid levels in tanks, reservoirs, and other container.

A. Architecture

- **Ultrasonic Transducers :**
  a. **Transmitter:** A piezoelectric ceramic element (often made from PZT – lead zirconate titanate) that converts electrical energy into 40 kHz ultrasonic sound waves.
  b. **Receiver:** Another piezoelectric element tuned to 40 kHz that detects the echo of the transmitted sound.
- **Electronic Circuitry :**
  a. **Trigger & Echo Logic:** The sensor has dedicated pins for receiving a trigger pulse and outputting an echo pulse. The on-board circuitry handles generating a burst of ultrasonic waves and measuring the time until the echo returns.
  b. **Oscillator & Timing Components:** Internal oscillator circuitry generates the precise 40 kHz pulses, and timing circuits (often implemented with comparators and microcontrollers in more integrated designs) calculate the delay between sending and receiving.
- **Pin Configuration :**
  a. **VCC:** Typically +5 V supply.
  b. **Trig:** Input pin that initiates the measurement.
  c. **Echo:** Output pin that produces a pulse whose width is proportional to the distance.
  d. **GND:** Ground reference.

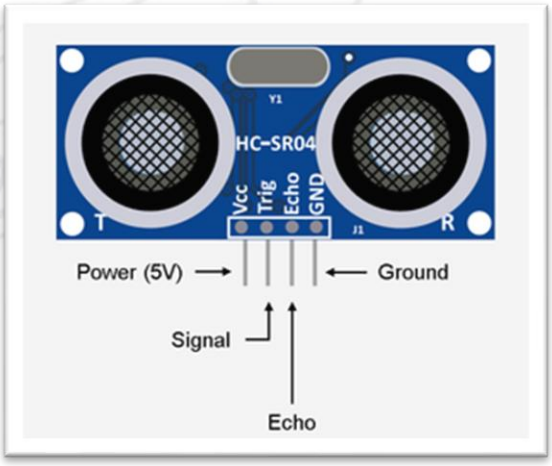


*Figure 24 - **HC-SR04 Ultrasonic Sensor Pin Diagram***

B. Working Process

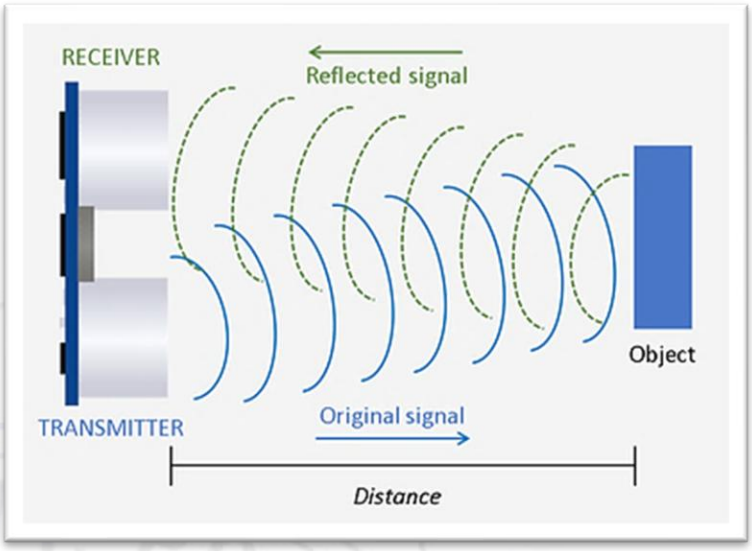


*Figure 25 - **HC-SR04 Ultrasonic Sensor Working Process***

1. Initiation :

- The microcontroller sends a *10 µs pulse* to the *Trig pin* to start the measurement.

2. Emission :

- Upon receiving the trigger, the sensor's internal oscillator sends out an *8-cycle burst* of 40 kHz ultrasonic pulses through the transmitter.

3. Echo Reception :

- The ultrasonic waves travel through the air until they hit an object and are reflected back.
- The receiver picks up the returning echo and, through internal circuitry, converts it into an electrical signal.

4. Distance Calculation :

- The sensor outputs a pulse on the *Echo pin* whose duration is proportional to the time taken for the sound to travel to the object and back.
- Using the speed of sound (approximately 343 m/s), the microcontroller can calculate the distance.

$$\text{Distance} = \frac{\text{Time Delay} \times 343\ \text{m/s}}{2}$$

(Division by 2 accounts for the round-trip distance)

C. Size

- **Overall Dimensions :**
  - **a.** The sensor module is typically around *45 mm (length) × 20 mm (width) × 15 mm (height),* though these dimensions may vary slightly between manufacturers.
- **Design Layout :**
  - **a.** It is built on a small rectangular PCB with the ultrasonic transducers mounted on one side (usually the front).
  - **b.** The board includes labeled pins (VCC, Trig, Echo, GND) for easy interfacing with microcontrollers.

D. Materials

- **PCB :**
  - **a.** Constructed from standard FR4 material.
- **Transducer Elements :**
  - **a.** Made of *piezoelectric ceramics* (typically PZT) that generate and detect the ultrasonic waves.
- **Housing :**
  - **a.** The sensor is usually enclosed in a protective casing made from *ABS plastic* or a similar durable, lightweight polymer.
  - **b.** The plastic cover is designed to be acoustically transparent to allow the ultrasonic waves to pass through with minimal attenuation.
- **Electronic Components :**
  - **a.** Includes resistors, capacitors, and possibly integrated circuits on the PCB for signal conditioning and timing.

7. LM2596 Buck Converter

The LM2596 buck converter module is a step-down voltage regulator that efficiently converts higher DC voltages down to a lower, stable voltage. It is widely used in power supply circuits for microcontrollers, sensors, and embedded systems.

A. LM2596 Regulator IC

The core of the module is the LM2596S-ADJ (Adjustable) switching regulator.

- **Switching Frequency :** 150 kHz, enabling high efficiency and minimal heat dissipation.
- **Output Voltage Range :** 1.23V to 37V (via onboard potentiometer).
- **Maximum Output Current :** 3A (with proper heat dissipation).

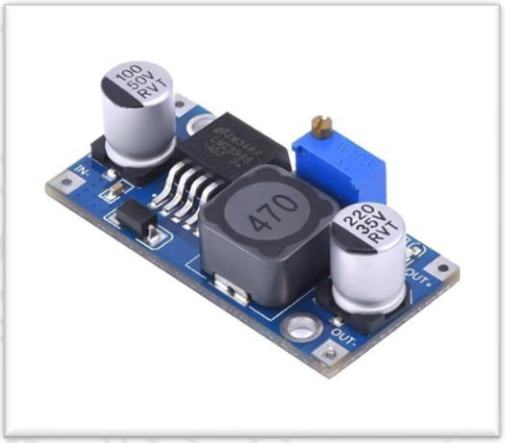


*Figure 26 - **LM2596 Buck Converter***

B. Input and Output Terminals

- **VIN (Input Terminal) :**

a. Accepts 4V to 40V DC as input.

b. Typically connected to a battery, power adapter, or solar panel.

- **VOUT (Output Terminal) :**

a. Provides a regulated output voltage (adjustable or fixed, depending on the module).

b. Used to power microcontrollers, sensors, and other circuits.

- **GND (Ground Terminal) :** Common ground connection for both input and output sides.

C. PCB Layout and Size

- **Dimensions :** 43mm × 21mm × 14mm (standard module size).
- **Pin Headers :** Typically screw terminals for secure connections.
- **Mounting Holes :** Available for easy installation in enclosures.

D. Rectifier Diode (Schottky Diode – 1N5822)

- Protects the circuit by preventing reverse current flow when the power is switched off.
- Reduces power losses and improves efficiency.

E. Inductor (47µH or 68µH)

- Works with the LM2596S IC to store and release energy efficiently.
- Helps maintain stable voltage output and reduces voltage ripples.

C. Descriptions of the Software components :

1. Android Studio IDE

Android Studio is the official Integrated Development Environment (IDE) for Android application development, based on JetBrains IntelliJ IDEA. It provides powerful tools for building, testing, and debugging Android apps efficiently.

A. Core Components of Android Studio

- **Android Gradle Plugin :** Automates the build process, dependency management, and APK generation.
- **Code Editor :** Advanced editor with syntax highlighting, auto-completion, and refactoring tools.
- **Layout Editor :** Drag-and-drop UI designer for creating responsive app interfaces.
- **Emulator :** Built-in Android Virtual Device (AVD) for testing apps on different Android versions and devices.
- **Logcat & Debugger :** Provides real-time logs and debugging tools for troubleshooting.

*Figure 27 - **Android Studio***

B. System Requirements

- **Operating System :** Windows (8/10/11), macOS, Linux (Ubuntu-based).
- **Processor :** Intel/AMD processor with x86_64 architecture.
- **RAM :** Minimum 8GB (16GB recommended for smooth performance).
- **Storage :** At least 8GB free space (SSD recommended).
- **Graphics :** Supports OpenGL 3.2+ for emulator acceleration.

C. Key Features and Tools

- **Android Emulator :** Test apps on virtual devices with different screen sizes and API levels.
- **Profiler :** Monitors CPU, memory, and network usage for performance optimization.
- **Layout Inspector :** Debug and analyze UI elements in real-time.
- **Database Inspector :** View and modify SQLite databases while the app is running.
- **Jetpack Libraries :** Pre-built components like LiveData, ViewModel, Room, and Navigation for modern app architecture.

D. Debugging and Testing

- **Unit Testing :** JUnit and Espresso frameworks for automated testing.
- **Instrumented Testing :** Runs tests on real devices or emulators.
- **Firebase Test Lab :** Cloud-based testing on multiple devices.
- **APK Analyzer :** Inspects APK files for size optimization.

2. Arduino IDE

Arduino IDE (Integrated Development Environment) is an open-source software used for writing, compiling, and uploading code to Arduino boards and other microcontrollers. It provides a simple interface for programming Arduino-based projects in C/C++.

A. Core Components of Arduino IDE

- **Code Editor :** Text editor with syntax highlighting and auto-indentation.
- **Compiler & Build System :** Converts C/C++ code into machine code for microcontrollers.
- **Board Manager :** Allows installation of additional board definitions (ESP32, ESP8266, etc.).
- **Library Manager :** Provides access to thousands of pre-built libraries for sensors, displays, and communication modules.
- **Serial Monitor & Plotter :** Tools for debugging and real-time data visualization via UART.

*Figure 28 – **Arduino IDE***

B. System Requirements

- **Operating System :** Windows (7/8/10/11), macOS, Linux.
- **Processor :** Intel/AMD (x86_64 or ARM architecture).
- **RAM :** Minimum 2GB (4GB+ recommended for large projects).
- **Storage :** At least 500MB free space.
- **USB Ports :** Required for connecting Arduino boards.

C. Debugging and Testing

- **Serial Monitor :** Displays real-time debug messages via USB/UART.
- **Serial Plotter :** Visualizes sensor data in real-time.
- **External Debugging Tools :** Supports JTAG, SWD, and third-party debuggers for advanced debugging.

D. Key Features and Tools

- **Sketch-Based Development :** Each project is called a "sketch" (.ino file).
- **Built-in Examples :** Pre-loaded code snippets for common applications (LED blinking, serial communication, etc.).
- **Board & Port Auto-Detection :** Automatically detects connected Arduino boards.

E. Supported Microcontrollers & Boards

- **Official Arduino Boards :**
  a. Arduino Uno, Mega, Nano, Leonardo, Due, Micro, Pro Mini, etc.
- **Third-Party Boards (via Board Manager) :**
  a. ESP32, ESP8266, STM32, ATtiny, Raspberry Pi Pico (RP2040), etc.

Working Methodology

A. Single User Connection to the Controller

To ensure secure and reliable control of the ESP32-controlled RC car, a single-user connection mechanism has been implemented using Firebase Realtime Database. This system prevents multiple users from simultaneously accessing and controlling the car, thereby reducing the risk of conflicting commands.

a. Purpose and Implementation

The primary purpose of this single-user connection feature is to restrict access to one user at a time. Firebase Realtime Database serves as a centralized system to monitor and manage the active connection. When a user attempts to connect, the application verifies whether any other active connection exists. If no connection is detected, the user is authenticated and granted control.

b. Working Principle

The connection management involves the following key steps :

- **Connection Monitoring :** A dedicated database node named activeConnection tracks the current connection status.
- **Authentication :** Users are required to enter their email and name. Upon submission, a timestamp is generated and stored in the database.
- **Stale Connection Detection :** To prevent lingering inactive sessions, the system checks the timestamp. If the connection is older than 5 minutes, it is considered stale and automatically removed.
- **User Validation :** If no connection exists or the previous connection is deemed stale, the user is granted access. If a connection is already active, a message indicating the device is occupied is displayed.

c. Firebase Database Structure

The Firebase Realtime Database follows a simple structure for tracking active connections :

```
1.
2. {
3.   "activeConnection": {
4.     "email": "user@example.com",
5.     "name": "User Name",
6.     "timestamp": 1711555600000
7.   }
8. }
9.
```

d. Database Rules

Custom rules ensure secure access to the database. Only one user can write to the activeConnection node if it is empty or if the last connection is older than 5 minutes. Validation rules confirm the presence of necessary fields (email, name, timestamp) and ensure the timestamp is a valid number.

```
{
  "rules": {
    "activeConnection": {
      ".read": true,
      ".write": "(!data.exists() || !newData.exists() || newData.child('timestamp').val() > (now - 300000))",
      ".validate": "newData.hasChildren(['email', 'name', 'timestamp']) && newData.child('timestamp').isNumber()"
    }
  }
}
```

e. User Experience

- When a user attempts to connect, the app checks the database for any existing connections.
- If no active connection is detected, the user is authenticated and directed to the car control interface.
- If a connection is active, the user is notified and prevented from proceeding.
- If a stale connection is found, it is automatically removed, allowing the new user to connect.

This single-user connection system ensures the safety and reliability of the RC car, preventing conflicts in command transmission and enhancing the overall user experience.

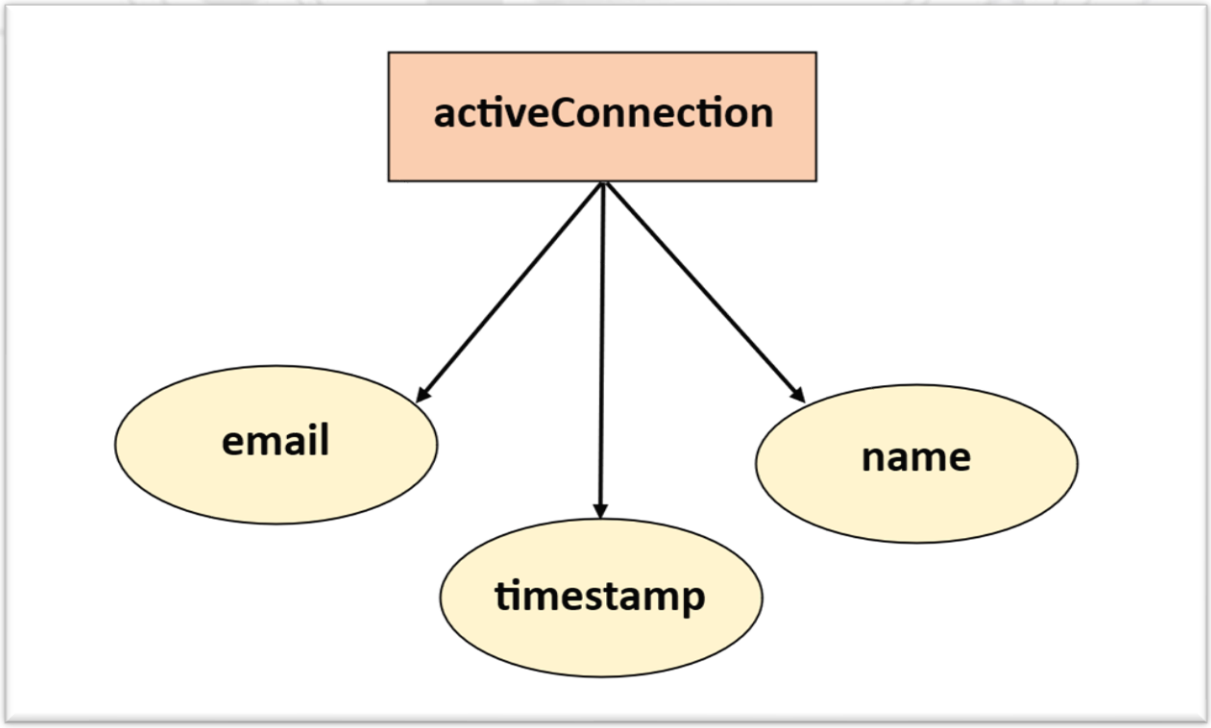


*Figure 28 - **Entity Relationship Diagram of the Firebase Realtime Database***

B. Android app connection to the ESP32 via Wi-Fi

ESP32 as a Wi-Fi Access Point (AP) for RC Car Control Using an Android App

The ESP32 is configured as a Wi-Fi Access Point (AP), enabling direct communication between the ESP32 and an Android application without requiring an external router or internet connection. This setup is useful for remote control applications, such as an RC car, where the ESP32 serves as the central controller for receiving commands and managing motor operations.

a. How the ESP32 Access Point Works

Unlike the usual station mode (STA), where an ESP32 connects to an existing Wi-Fi network, Access Point (AP), mode allows it to create its own Wi-Fi network. In this mode :

- The ESP32 creates a Wi-Fi hotspot with a predefined SSID and password.
- Other devices, like a smartphone running an Android app, can connect directly to this network.
- The ESP32 assigns IP addresses to connected devices using DHCP (Dynamic Host Configuration Protocol).
- The default gateway (IP address of ESP32) is - 192.168.4.1

b. Connecting an Android App to the ESP32

Once the ESP32 is in AP mode, the Android smartphone can connect by :

1. Opening Wi-Fi settings on the smartphone.
2. Selecting the ESP32's SSID (hotspot name) from available networks.
3. Entering the predefined password.
4. Obtaining an IP address (assigned automatically by the ESP32).

After the smartphone connects, the Android app can start communicating with the ESP32 by sending HTTP requests.

c. Advantages of ESP32 as an Access Point

1. No Need for External Wi-Fi : The car can be controlled anywhere without relying on an internet connection.
2. Low Latency : Direct communication ensures fast response times.
3. Flexible and Portable : The Android device can connect and control the car anywhere.
4. Simple Setup : Just power on the ESP32, connect from the app, and control the car.

d. Processing HTTP Requests on the ESP32

- The ESP32 runs a web server on port 80.
- When the Android app sends an HTTP GET request, the ESP32 reads the request.
- It extracts the command from the URL.

- Based on the command, the ESP32:
  - **a.** Controls the DC motors using the L293D motor driver.
  - **b.** Turns LEDs on or off.
  - **c.** Checks for obstacles using an HC-SR04 ultrasonic sensor.
- If an obstacle is detected within 60 cm, the car automatically moves backward for 0.5 second before stopping.

e. Example ESP32 Code Execution Flow

- The ESP32 initializes as an Access Point.
- The Android app connects to the ESP32's Wi-Fi.
- The app sends an HTTP GET request (e.g., /forward).
- The ESP32 parses the request to extract the command.
- The ESP32 checks the ultrasonic sensor :
  - **a.** If an obstacle is closer than 60 cm, the car moves backward and stops.
  - **b.** Otherwise, the command executes normally (moving forward, turning, etc.).
- The ESP32 sends an HTTP response back to the Android app.
- The Android app displays feedback (e.g., "Car moving forward").

f. Example Scenario : Controlling the RC Car

Scenario 1 : Moving the Car Forward

- The Android app sends http://192.168.4.1/forward.
- ESP32 receives the request and extracts the command forward.
- The HC-SR04 ultrasonic sensor checks for obstacles.
  - **a.** If distance > 60 cm, the car moves forward.
  - **b.** If distance < 60 cm, the car moves backward for 0.5 second and stops.

Scenario 2 : Avoiding an Obstacle

- The car is moving forward.
- The HC-SR04 sensor detects an obstacle at 60 cm.
- The ESP32 ignores user commands and moves the car backward for 0.5 second.
- The car stops and waits for the next command.

g. How the Android App Sends Commands

The Android app sends HTTP GET requests to the ESP32 using its IP address (192.168.4.1). These requests contain control commands that tell the ESP32 how to move the car.

Example HTTP Requests from Android App.

The ESP32 receives the request, extracts the command, and controls the motors and LEDs accordingly.

Command	URL Sent to ESP32
Move Forward	http://192.168.4.1/forward
Move Backward	http://192.168.4.1/backward
Turn Left	http://192.168.4.1/left
Turn Right	http://192.168.4.1/right
Stop	http://192.168.4.1/stop

C. Sending HTTP requests for movement control

ESP32 and Android App Communication Using OKHttp

The Android app communicates with the ESP32 using the OKHttp library, which sends HTTP GET requests over the local network. Each request contains a specific command that the ESP32 processes in real-time, allowing it to control the RC car's movements.

1. How OKHttp Works in the Android App

The Android app :

- Connects to the ESP32's Wi-Fi hotspot.
- Sends HTTP GET requests to 192.168.4.1, which is the ESP32's IP address.
- Receives a response from the ESP32 confirming the command.
- Displays feedback to the user (e.g., "Car moving forward").

2. Example HTTP GET Requests

The Android app sends requests based on the user's input. Below are some example URLs :

Command	HTTP Request
Move Forward	http://192.168.4.1/forward
Move Backward	http://192.168.4.1/backward
Turn Left	http://192.168.4.1/left
Turn Right	http://192.168.4.1/right
Stop	http://192.168.4.1/stop

Each request is a GET request, meaning no additional data needs to be sent in the body. The ESP32 extracts the command from the URL, processes it, and controls the motors accordingly.

3. How the ESP32 Handles Requests

The ESP32 runs a web server that :

- Listens for incoming HTTP requests on port 80.
- Extracts the command from the request URL.
- Controls the motors based on the received command.
- Checks for obstacles using the HC-SR04 ultrasonic sensor.
- Sends an HTTP response back to the Android app.

4. Breakdown of ESP32 Request Handling

Step 1 : ESP32 Receives a Request

When the Android app sends a request like :

`http://192.168.4.1/forward`

The ESP32 receives and processes it.

Step 2 : Extracting the Command

The ESP32 extracts "forward" from the request.

Step 3 : Obstacle Detection

Before executing the command, the ESP32 checks for obstacles using the HC-SR04 ultrasonic sensor:

- If an object is detected within 60 cm, the car moves backward for 0.5 second and stops.
- If no obstacle is detected, the car executes the command (e.g., moving forward).

Step 4 : Controlling the Motors

The ESP32 controls the L293D motor driver, which in turn controls the DC motors to move the car.

Step 5 : Sending a Response

After executing the command, the ESP32 sends a response back to the Android app, confirming the action :

```
12.
13. HTTP/1.1 200 OK
14. Content-Type: text/html
15.
16. <html>
17. <body>
18. Car is moving forward
19. </body>
20. </html>
21.
```

The Android app displays this message to the user.

5. Example Android Code Using OKHttp

Below is an example of how the Android app sends HTTP GET requests using the OKHttp library.

A. Import Dependencies

In `build.gradle` , add :

```
1.
2. dependencies {
3.     implementation 'com.squareup.okhttp3:okhttp:4.9.3'
4. }
5.
```

B. Send an HTTP GET Request

The following Java method sends a GET request to the ESP32 :

```
1.
2.  import okhttp3.OkHttpClient;
3.  import okhttp3.Request;
4.  import okhttp3.Response;
5.  import java.io.IOException;
6.
7.  public class ESP32Controller {
8.
9.      private static final String ESP_IP = "http://192.168.4.1/";
10.
11.     public static String sendCommand(String command) {
12.         OkHttpClient client = new OkHttpClient();
13.         String url = ESP_IP + command;
14.
15.         Request request = new Request.Builder().url(url).build();
16.         try (Response response = client.newCall(request).execute()) {
17.             return response.body().string();
18.         } catch (IOException e) {
19.             return "Error: " + e.getMessage();
20.         }
21.     }
22. }
23.
```

C. Trigger the Command from Android UI

```
1.
2.  Button forwardButton = findViewById(R.id.forwardButton);
3.
4.  forwardButton.setOnClickListener(v -> {
5.      new Thread(() -> {
6.          String response = ESP32Controller.sendCommand("forward");
7.      }).start();
8.  });
9.
```

6. ESP32 Code for Request Handling

The ESP32 listens for requests and executes the appropriate motor control functions.

A. Extracting Commands from Requests

```
1.
2.  WiFiClient client = server.available();
3.
4.  if (client) {
5.      String request = "";
6.      while (client.connected()) {
7.          if (client.available()) {
8.              char c = client.read();
9.              request += c;
10.             if (c == '\r') {
11.                 int start = request.indexOf("GET /") + 5;
12.                 int end = request.indexOf("HTTP/");
13.                 String command = request.substring(start, end);
14.
15.                 command.replace("\n", "");
16.                 command.replace("\r", "");
17.                 command.replace(" ", "");
18.                 command.replace("\t", "");
19.
20.                 processCommand(command);
21.                 break;
22.             }
23.         }
24.     }
25. }
26.
```

B. Processing Commands

```
1.
2.  void processCommand(String command) {
3.      if (command.equals("forward")) {
4.          moveForward();
5.      } else if (command.equals("backward")) {
6.          moveBackward();
7.      } else if (command.equals("left")) {
8.          moveLeft();
9.      } else if (command.equals("right")) {
10.         moveRight();
11.     } else if (command.equals("stop")) {
12.         moveStop();
13.     }
14. }
15.
```

C. Obstacle Avoidance Logic

```
1.
2.  void checkObstacle() {
3.      digitalWrite(TRIG_PIN, LOW);
4.      delayMicroseconds(2);
5.      digitalWrite(TRIG_PIN, HIGH);
6.      delayMicroseconds(10);
7.      digitalWrite(TRIG_PIN, LOW);
8.
9.      long duration = pulseIn(ECHO_PIN, HIGH);
10.     float distance = (duration * 0.0343) / 2;
11.
12.     if (distance < 60 && distance > 0) {
13.         moveBackward();
14.         delay(500);
15.         moveStop();
16.     }
17. }
18.
```

D. Processing commands on ESP32

Comprehensive Overview of ESP32 as a Web Server for RC Car Control

The ESP32 is programmed to function as a web server, allowing it to receive, process, and respond to HTTP requests from an Android application. This setup enables remote control of an RC car, utilizing Wi-Fi communication between the ESP32 and a smartphone. The ESP32 processes commands in real-time, allowing seamless control over the car's movements.

1. ESP32 Web Server : Role and Functionality

The ESP32 acts as an HTTP web server, which means :

- It broadcasts a Wi-Fi hotspot (Access Point mode).
- It assigns an IP address (192.168.4.1) to itself.
- It listens on port 80 for HTTP GET requests.
- It extracts movement commands from the requests.
- It controls the car's motors accordingly.

This means that an Android application (or any device connected to the ESP32's network) can send HTTP requests to trigger actions like moving forward, backward, left, right, or stopping.

2. How Commands Are Processed

When an HTTP GET request is received from the Android app, the ESP32 extracts the command from the request URL. The following steps occur :

Step 1 : Receiving an HTTP Request

The ESP32 continuously listens for incoming HTTP requests from the Android application or any web browser.

For example, when the Android app sends :

```
1.
2.  http://192.168.4.1/forward
3.
```

The ESP32 receives this request and processes it.

Step 2 : Extracting the Command

The ESP32 reads the request and extracts the command from the URL.
For instance :

- If the request is http://192.168.4.1/forward the extracted command is "forward".
- If the request is http://192.168.4.1/left the extracted command is "left".

Step 3 : Mapping the Command to a Function

Once the command is extracted, the ESP32 maps it to a predefined function :

Command	Function
"forward"	moveForward()
"backward"	moveBackward()
"left"	moveLeft()
"right"	moveRight()
"stop"	moveStop()

Step 4 : Executing the Function

Each function controls the motor driver pins to move the car in the desired direction.

Example :

If the command is `"forward"`, the function `moveForward()` is called, which activates the motor driver pins to move the car forward.

3. Example HTTP Requests and Responses

Request : Moving Forward

Android app sends :

```
http://192.168.4.1/forward
```

ESP32 processes it and calls `moveForward()`.

ESP32 sends back a response :

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<body>
Car is moving forward
</body>
</html>
```

Request : Stopping the Car

Android app sends :

```
http://192.168.4.1/stop
```

ESP32 processes it and calls `moveStop()`.

ESP32 sends back a response :

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<body>
Car has stopped
</body>
</html>
```

4. How the ESP32 Controls the Motors

The ESP32 controls the car's movement using L293D motor driver, which operates two BO motors (one for each wheel).

Each motor has two control pins :

- Setting one pin HIGH and the other LOW makes the motor rotate in one direction.
- Reversing the signals makes it rotate in the opposite direction.
- Setting both pins LOW stops the motor.

Motor Pin Assignments

```
#define M1_A 21  // Motor 1 Forward
#define M1_B 19  // Motor 1 Backward
#define M2_A 23  // Motor 2 Forward
#define M2_B 22  // Motor 2 Backward
```

5. ESP32 Code : Handling HTTP Requests

The following ESP32 code extracts commands from HTTP GET requests and maps them to movement functions.

Extract and Process HTTP Requests

```
WiFiClient client = server.available();

if (client) {
    String request = "";
    while (client.connected()) {
        if (client.available()) {
            char c = client.read();
            request += c;
            if (c == '\r') {
                int start = request.indexOf("GET /") + 5;
                int end = request.indexOf("HTTP/");
                String command = request.substring(start, end);

                command.replace("\n", "");
                command.replace("\r", "");
                command.replace(" ", "");
                command.replace("\t", "");

                processCommand(command);
                break;
            }
        }
    }
}
```

6. Processing the Command

```
1.
2.  void processCommand(String command) {
3.      if (command.equals("forward")) {
4.          moveForward();
5.      } else if (command.equals("backward")) {
6.          moveBackward();
7.      } else if (command.equals("left")) {
8.          moveLeft();
9.      } else if (command.equals("right")) {
10.         moveRight();
11.     } else if (command.equals("stop")) {
12.         moveStop();
13.     }
14. }
15.
```

7. Movement Functions

Move Forward

```
1.
2.  void moveForward() {
3.      digitalWrite(M1_A, HIGH);
4.      digitalWrite(M1_B, LOW);
5.      digitalWrite(M2_A, HIGH);
6.      digitalWrite(M2_B, LOW);
7.  }
8.
```

Move Backward

```
1.
2.  void moveBackward() {
3.      digitalWrite(M1_A, LOW);
4.      digitalWrite(M1_B, HIGH);
5.      digitalWrite(M2_A, LOW);
6.      digitalWrite(M2_B, HIGH);
7.  }
8.
```

Turn Left

```
1.
2.  void moveLeft() {
3.      digitalWrite(M1_A, HIGH);
4.      digitalWrite(M1_B, LOW);
5.      digitalWrite(M2_A, LOW);
6.      digitalWrite(M2_B, HIGH);
7.  }
8.
```

Turn Right

```
1.
2.  void moveRight() {
3.      digitalWrite(M1_A, LOW);
4.      digitalWrite(M1_B, HIGH);
5.      digitalWrite(M2_A, HIGH);
6.      digitalWrite(M2_B, LOW);
7.  }
8.
```

Stop the Car

```
1.
2.  void moveStop() {
3.      digitalWrite(M1_A, LOW);
4.      digitalWrite(M1_B, LOW);
5.      digitalWrite(M2_A, LOW);
6.      digitalWrite(M2_B, LOW);
7.  }
```

8. Ultrasonic Sensor : Obstacle Avoidance

Before executing any movement, the ESP32 checks for obstacles using an HC-SR04 ultrasonic sensor.

```
1.
2.  void checkObstacle() {
3.      digitalWrite(TRIG_PIN, LOW);
4.      delayMicroseconds(2);
5.      digitalWrite(TRIG_PIN, HIGH);
6.      delayMicroseconds(10);
7.      digitalWrite(TRIG_PIN, LOW);
8.
9.      long duration = pulseIn(ECHO_PIN, HIGH);
10.     float distance = (duration * 0.0343) / 2;
11.
12.     if (distance < 60 && distance > 0) {
13.         moveBackward();
14.         delay(500);
15.         moveStop();
16.     }
17. }
18.
```

E. System Operation

Wi-Fi Controlled RC Car with Obstacle Avoidance

The ESP32-controlled RC car is designed to receive movement commands from a smartphone application via Wi-Fi. These commands are processed by the ESP32, which then controls the L293D motor driver to move the car in the desired direction. Additionally, an HC-SR04 ultrasonic sensor is integrated into the system to ensure obstacle detection and avoidance, preventing collisions by making the car automatically stop and reverse when an obstacle is detected.

1. Communication and Command Processing

- **Smartphone Application as a Controller**

The Android application serves as a wireless remote control for the RC car. The app connects to the ESP32's Wi-Fi Access Point and sends movement commands using HTTP requests. Each request contains a command parameter that tells the ESP32 which action to perform.

Commands Sent from Smartphone App

Command	URL Request Sent to ESP32	Action Performed
Move Forward	http://192.168.4.1/forward	Car moves forward
Move Backward	http://192.168.4.1/backward	Car moves backward
Turn Left	http://192.168.4.1/left	Car turns left
Turn Right	http://192.168.4.1/right	Car turns right
Stop	http://192.168.4.1/stop	Car stops

- **ESP32 as the Central Controller**

  a. The ESP32 listens for incoming HTTP requests on port 80.

  b. Upon receiving a request, it extracts the command from the URL.

  c. The command is then mapped to a corresponding motor control function (e.g., `moveForward()`, `moveBackward()`, etc.).

  d. The ESP32 sends signals to the L293D motor driver to control the DC motors, making the car move in the desired direction.

2. Motor Control System Using L293D Driver

- **Role of L293D Motor Driver**

The L293D motor driver acts as a dual H-bridge circuit, allowing the ESP32 to control two DC motors independently. The ESP32 sets the input pins HIGH or LOW to determine the motor's rotation direction.

Motor Control Logic

Command	Left Wheel's	Right Wheel's
Forward	Rotates Forward	Rotates Forward
Backward	Rotates Backward	Rotates Backward
Left Turn	Rotates Forward	Rotates Backward
Right Turn	Rotates Backward	Rotates Forward
Stop	Motor OFF	Motor OFF

- **Wiring of L293D to ESP32**

L293D Pin	ESP32 Pin	Function
IN1	GPIO 19	Motor 1 Forward
IN2	GPIO 21	Motor 1 Backward
IN3	GPIO 23	Motor 2 Forward
IN4	GPIO 22	Motor 2 Backward

- **Code for Motor Control Functions**

Each function activates the appropriate pins to control motor movement.

Move Forward

```
1.
2.  void moveForward() {
3.      digitalWrite(M1_A, HIGH);
4.      digitalWrite(M1_B, LOW);
5.      digitalWrite(M2_A, HIGH);
6.      digitalWrite(M2_B, LOW);
7.  }
8.
```

Move Backward

```
1.
2.  void moveBackward() {
3.      digitalWrite(M1_A, LOW);
4.      digitalWrite(M1_B, HIGH);
5.      digitalWrite(M2_A, LOW);
6.      digitalWrite(M2_B, HIGH);
7.  }
8.
```

Turn Left

```
1.
2.  void moveLeft() {
3.      digitalWrite(M1_A, HIGH);
4.      digitalWrite(M1_B, LOW);
5.      digitalWrite(M2_A, LOW);
6.      digitalWrite(M2_B, HIGH);
7.  }
8.
```

Turn Right

```
1.
2.  void moveRight() {
3.      digitalWrite(M1_A, LOW);
4.      digitalWrite(M1_B, HIGH);
5.      digitalWrite(M2_A, HIGH);
6.      digitalWrite(M2_B, LOW);
7.  }
8.
```

Stop the Car

```
1.
2.  void moveStop() {
3.      digitalWrite(M1_A, LOW);
4.      digitalWrite(M1_B, LOW);
5.      digitalWrite(M2_A, LOW);
6.      digitalWrite(M2_B, LOW);
7.  }
8.
```

3. Ultrasonic Sensor for Obstacle Detection and Automatic Avoidance

- **Role of HC-SR04 Sensor**

The HC-SR04 ultrasonic sensor continuously monitors the distance between the car and obstacles. If the distance is less than 60 cm, the ESP32 will automatically moves backward for a short duration and stops it.

- **Code for Obstacle Detection**

```
1.
2.  void checkObstacle() {
3.      digitalWrite(TRIG_PIN, LOW);
4.      delayMicroseconds(2);
5.      digitalWrite(TRIG_PIN, HIGH);
6.      delayMicroseconds(10);
7.      digitalWrite(TRIG_PIN, LOW);
8.
9.      long duration = pulseIn(ECHO_PIN, HIGH);
10.     float distance = (duration * 0.0343) / 2;
11.
12.     if (distance < 60 && distance > 0) {
13.         moveBackward();
14.         delay(500);
15.         moveStop();
16.     }
17. }
18.
```

- **How the Obstacle Avoidance Works**

  a. The HC-SR04 sensor sends out an ultrasonic pulse and listens for an echo.

  b. The ESP32 calculates the distance using the time it takes for the echo to return.

  c. If the distance is less than 60 cm, the ESP32 moves the car backwards for 0.5 second and stops the car.

  d. Once the car stops, it waits for a new command from the smartphone.

4. Complete Workflow of the System

- **Summary of Operations**

  a. Smartphone app connects to ESP32's Wi-Fi hotspot.

  b. App sends movement commands via HTTP GET requests.

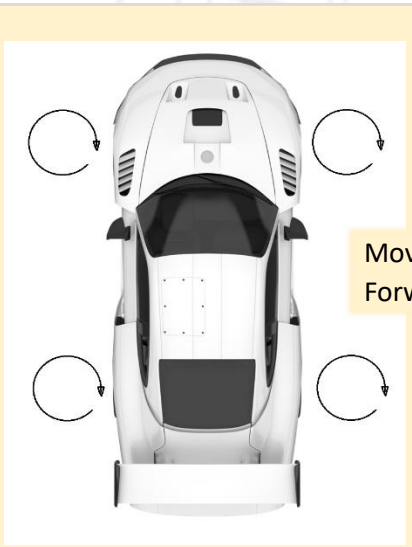
  c. ESP32 processes the request and controls the L293D motor driver.

  d. Car moves forward, backward, left, right, or stops based on command.

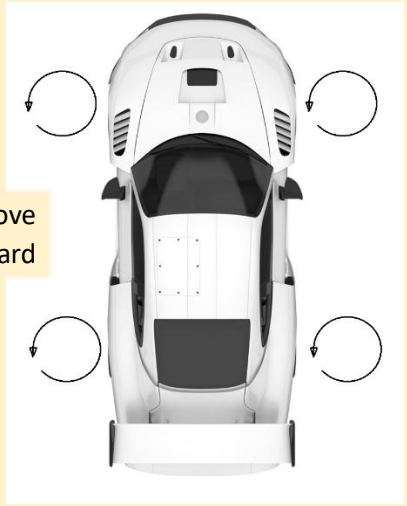
  e. HC-SR04 ultrasonic sensor continuously monitors obstacles.

  f. If an obstacle is detected within 60 cm, the car moves backwards for 0.5 second and stops automatically.

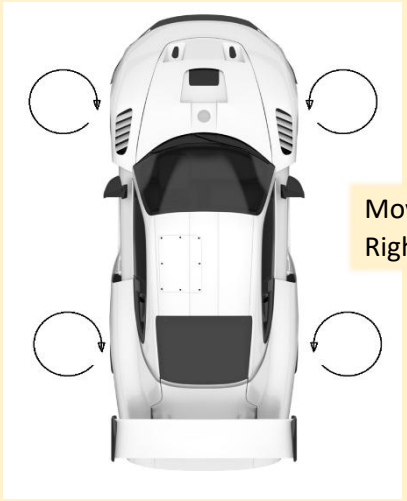
  g. ESP32 waits for the next command from the smartphone.



Move Forward



Move Backward



Move Right

Move Left

F. Obstacle Detection Using HC-SR04

HC-SR04 Ultrasonic Sensor-Based Obstacle Detection and Avoidance

The HC-SR04 ultrasonic sensor plays a crucial role in the autonomous obstacle avoidance system of the ESP32-controlled RC car. It continuously measures the distance between the car and obstacles in its path. If an obstacle is detected within 60 cm, the ESP32 makes it move backward, and then stops to prevent collisions. This ensures safe navigation in dynamic environments.

1. Working Principle of the HC-SR04 Sensor

The HC-SR04 ultrasonic sensor measures distance using sound waves. It consists of two key components :

a. **Trigger Pin (ESP32 → HC-SR04)** : Used to send an ultrasonic pulse.

b. **Echo Pin (HC-SR04 → ESP32)** : Used to receive the reflected pulse after bouncing off an obstacle.

[Steps in Distance Measurement]

- ESP32 sends a 10-microsecond HIGH pulse to the Trigger Pin of the HC-SR04 sensor.
- The sensor emits an ultrasonic wave (40 kHz) into the surrounding environment.
- If the wave hits an obstacle, it bounces back toward the Echo Pin of the HC-SR04.
- The ESP32 measures the time duration (in microseconds) that the pulse takes to return.
- Using the speed of sound (343 m/s or 0.0343 cm/µs), the ESP32 calculates the distance of the obstacle.

2. Communication Between ESP32 and HC-SR04

The ESP32 and HC-SR04 sensor communicate using digital signals.

Component	ESP32 Pin	Function
Trigger Pin (HC-SR04)	GPIO 5	Sends ultrasonic pulses
Echo Pin (HC-SR04)	GPIO 18	Receives reflected signal

3. Distance Calculation Formula

The ESP32 calculates distance using the formula :

$$\text{Distance} = \left(\frac{\text{Pulse Duration} \times 0.03432}{2} \right) \text{cm}$$

Explanation of the Formula :

- **Pulse Duration :** Time taken for the ultrasonic wave to hit an obstacle and return.
- **Speed of Sound :** 343 m/s or 0.0343 cm/µs.

- **Division by 2 :** The wave travels to the obstacle and back, so we divide by 2 to get the actual distance.

Example Calculation :

If the pulse duration is 1764 microseconds, the distance is :

$$\text{Distance} = \left(\frac{1764 \times 0.03432}{2}\right) = \left(\frac{60.5252}{2}\right) = 30.26\text{cm}$$

4. Obstacle Avoidance Logic in ESP32

The ESP32 continuously checks the distance to obstacles and takes appropriate actions.

- **Steps for Obstacle Avoidance**

a. The ESP32 reads the distance from the HC-SR04 sensor.

b. If the distance is greater than 60 cm, the car continues moving according to user commands.

c. If an obstacle is detected within 60 cm, the car stops immediately.

d. The ESP32 reverses the car for 0.5 second to move away from the obstacle.

e. The car stops again and waits for the next command.

5. Code Implementation for HC-SR04 Sensor in ESP32

- **Pin Configuration and Setup**

```
1.
2.  #define TRIG_PIN 5   // Trigger pin connected to GPIO 5
3.  #define ECHO_PIN 18  // Echo pin connected to GPIO 18
4.
5.  void setup() {
6.      pinMode(TRIG_PIN, OUTPUT);
7.      pinMode(ECHO_PIN, INPUT);
8.      Serial.begin(115200);
9.  }
10.
```

- **Function to Measure Distance**

```
1.
2.  float measureDistance() {
3.      digitalWrite(TRIG_PIN, LOW);
4.      delayMicroseconds(2);
5.      digitalWrite(TRIG_PIN, HIGH);
6.      delayMicroseconds(10);
7.      digitalWrite(TRIG_PIN, LOW);
8.
9.      long duration = pulseIn(ECHO_PIN, HIGH);
10.     float distance = (duration * 0.0343) / 2; // Convert time to distance
11.
12.     Serial.print("Distance: ");
13.     Serial.print(distance);
14.     Serial.println(" cm");
15.
16.     return distance;
17. }
18.
```

- **Obstacle Detection and Avoidance Function**

```
1.
2.  void checkObstacle() {
3.      float distance = measureDistance();
4.
5.      if (distance < 60 && distance > 0) {  // If an obstacle is detected
6.          Serial.println("Obstacle detected! Moving backward...");
7.          moveBackward();
8.          delay(500); // Move backward for 0.5 second
9.          moveStop();
10.     }
11. }
12.
```

- **Integrating Obstacle Detection into Main Loop**

```
1.
2.  void loop() {
3.      checkObstacle(); // Continuously check for obstacles
4.      delay(500);  // Wait before next reading
5.  }
6.
```

G. Communication Protocols

1. Wi-Fi Protocol

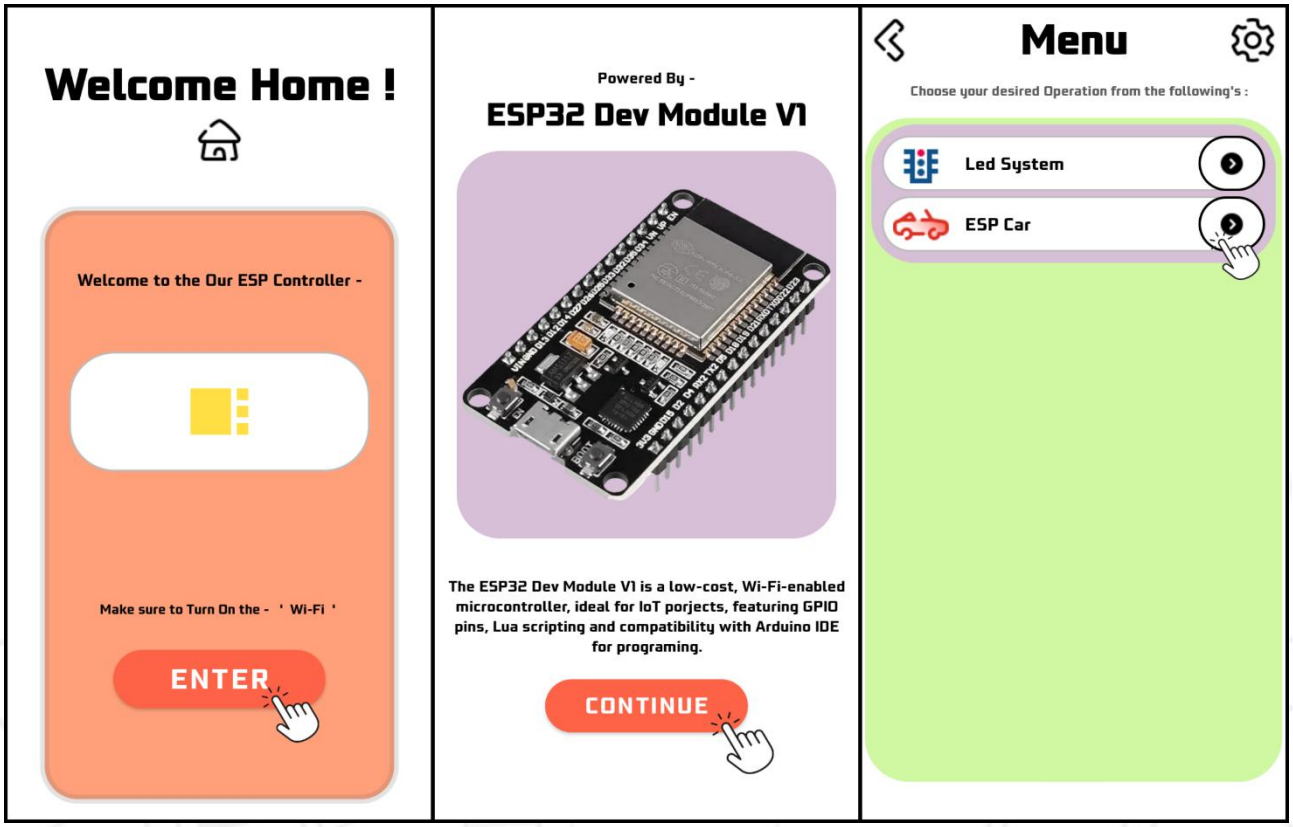
Specification	Details
Protocol	IEEE 802.11 (Wi-Fi)
Frequency	2.4 GHz
Data Rate	Up to 150 Mbps (802.11n)
Range	25m (indoor) / 40m (outdoor)

2. Communication Flow

- **Smartphone App** → Sends command (e.g., /forward).
- **Wi-Fi Module (ESP32 AP Mode)** → Receives command via HTTP GET request.
- **ESP32 Microcontroller** → Processes command, controls motors, sends response.
- **Motor Driver (L293D)** → Receives signal from ESP32 and moves the car.

Images of the Project

A. Application screenshots



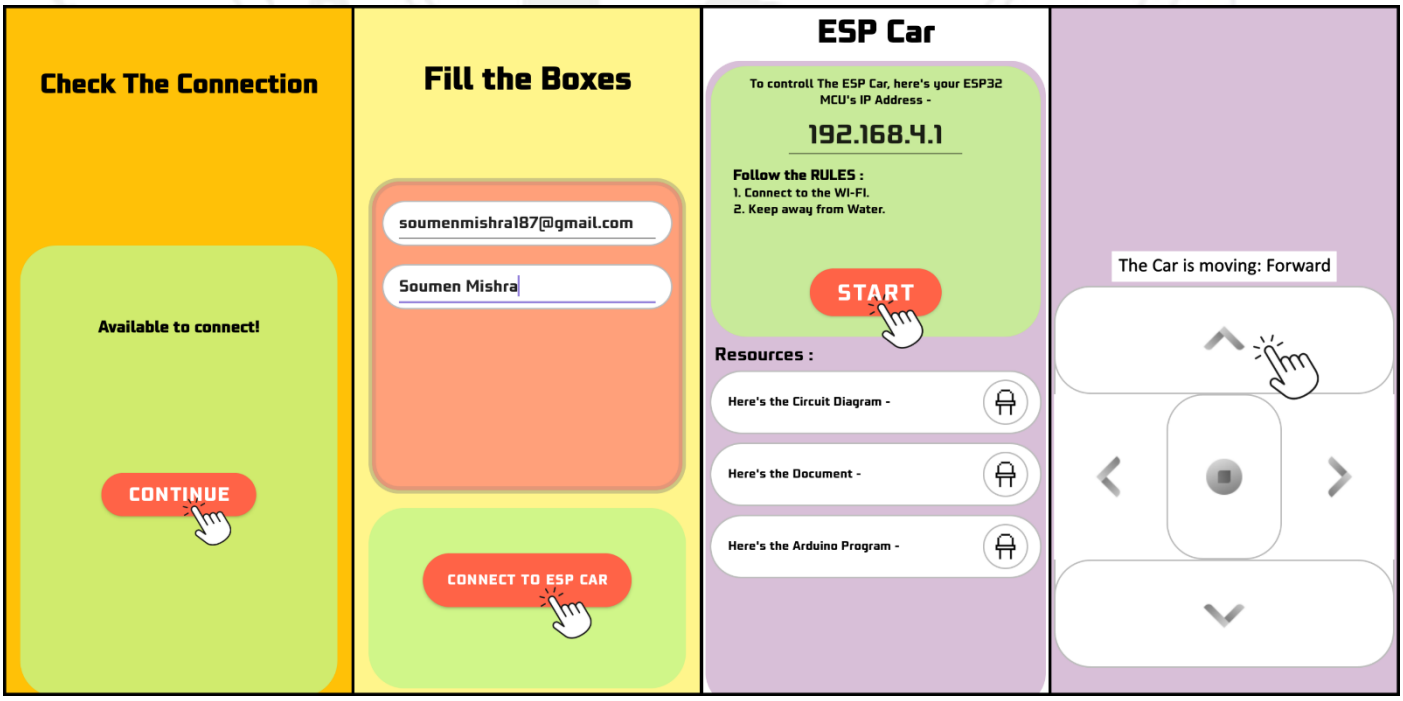


*Figure 29 - **Layout's of the Application***

Challenges

Challenges Faced

During the development of the Wi-Fi Controlled RC Car using ESP32 and Android Application, several challenges were encountered across both the hardware and software domains.

The key challenges faced are outlined below :

B. ESP32 Board Installation and Configuration

- The initial setup of the ESP32 board in the Arduino IDE posed difficulties due to missing board definitions.
- Errors occurred while installing the ESP32 board manager using the Board Manager URL.
- Additionally, selecting the correct COM Port and ensuring driver compatibility for the ESP32 module was a time-consuming process.
- Resolving issues related to the "Failed to connect" error required pressing the BOOT button during the uploading process.

C. Wi-Fi Configuration and Access Point Creation

- Establishing a stable Wi-Fi Access Point (AP) using the ESP32 was challenging due to inconsistent connectivity.
- Implementing a single-client control system to restrict access to only one user required additional logic and debugging.

D. Android Application Development

- While building the Android application using Android Studio, integrating the OkHttp library for sending HTTP GET requests caused compatibility issues.
- Errors related to Gradle dependencies and internet permissions occurred. Configuring the AndroidManifest.xml properly was necessary.

E. Sensor and Motor Control

- Synchronizing the car's movement and implementing the obstacle avoidance algorithm required careful calibration.
- Ensuring proper control signals from the ESP32 to the L293D motor driver for accurate movement was challenging.

F. Power Management

- Efficiently regulating power using the 7.4V battery and a 5V buck converter was necessary to avoid voltage fluctuations.

Conclusion

The development of the Obstacle Detection Car Controlled by Android Application using ESP32 and Android Studio successfully achieved its primary objective of enabling remote car control using a smartphone app over a Wi-Fi network. By establishing a stable access point using the ESP32, the application effectively sent HTTP GET requests to control the car's movements. The integration of the HC-SR04 ultrasonic sensor allowed for real-time obstacle detection and automatic movement adjustments, enhancing the car's functionality and safety.

Throughout the project, several challenges were encountered and overcome. Initial difficulties in installing and configuring the ESP32 board within the Arduino IDE were resolved by troubleshooting driver issues and using correct board settings. Similarly, problems related to Android Studio, such as dependency conflicts with the OkHttp library, were addressed through systematic debugging and proper Gradle management. Implementing single-client control to ensure that only one user could operate the car at a time was a notable enhancement, preventing simultaneous command conflicts.

The project demonstrated significant learning outcomes, including hands-on experience in embedded systems, Wi-Fi networking, Android application development, and sensor integration. Additionally, the implementation of a simple obstacle avoidance mechanism highlighted the practical application of IoT in automated systems.

Overall, the project successfully addressed the initial problem of remote car control using Wi-Fi, providing a responsive and reliable user experience. Future improvements, such as integrating computer vision for advanced obstacle detection and adding GPS-based navigation, can further enhance the car's autonomous capabilities. The experience gained during this project will be valuable in tackling more complex IoT-based applications in the future.

Future Scope

A. Future Work : Obstacle Avoidance Using Computer Vision

In the current system, obstacle detection is performed using an HC-SR04 ultrasonic sensor, which measures distance and stops the car when an obstacle is detected within 60 cm. However, this method has some limitations:

- **Limited Field of View (FOV) :** The ultrasonic sensor detects obstacles only in a narrow beam in front of the car.
- **Lack of Object Recognition :** It cannot distinguish between different objects or classify them.
- **Fixed Distance Threshold :** It stops the car at a fixed distance without considering the obstacle's size or movement.

1. Proposed Future Enhancement : Computer Vision for Obstacle Avoidance

To improve obstacle detection and enhance autonomous navigation, computer vision-based object detection can be integrated into the system using :

- **ESP32-CAM Module** (for real-time video streaming)
- **OpenCV and Machine Learning Algorithms** (for object detection and classification)

2. Implementation Approach

- **Integrating an ESP32-CAM for Real-Time Vision**
  - **a.** The ESP32-CAM module will be mounted on the RC car.
  - **b.** It will stream live video to the smartphone or process images locally.
  - **c.** The camera will act as an eye for the vehicle, helping it detect obstacles dynamically.

- **Object Detection Using OpenCV and Machine Learning**
  - **a.** A machine learning model (e.g., YOLO, MobileNet, or Haar Cascades) can be trained to identify obstacles such as walls, humans, or other cars.
  - **b.** OpenCV will process frames from the ESP32-CAM to detect obstacles and classify objects.

- **Decision Making for Obstacle Avoidance**

  If an obstacle is detected :
  - **a.** **Determine object type** (e.g., moving person vs. static wall).
  - **b.** **Decide the best movement** (stop, turn left/right, slow down).
  - **c.** **Use AI models** to predict the safest path for movement.

3. Comparison : Ultrasonic vs. Computer Vision

Feature	Ultrasonic Sensor	Computer Vision
Detection Range	Limited to ~4m	Can detect objects at long distances
Obstacle Type	Detects any object	Recognizes and classifies objects
Field of View	Narrow (30°)	Wide (up to 180° with a camera)
Precision	Less precise	Highly accurate with AI models
Adaptability	Fixed distance threshold	Can adjust based on object type

4. Benefits of Computer Vision Integration

- **Enhanced Accuracy :** Recognizes and classifies obstacles instead of just measuring distance.
- **Smarter Navigation :** AI can decide when to stop, slow down, or reroute.
- **Better Field of View :** A camera can scan a wider area compared to an ultrasonic sensor.
- **Dynamic Obstacle Handling :** The system can distinguish moving objects from static obstacles.

B. Integration with Other Sensors (GPS, Accelerometer, Camera)

To enhance the functionality and intelligence of the RC car, additional sensors such as GPS, accelerometer, and camera can be integrated. These sensors will provide more control, navigation, and environmental awareness, making the system smarter and more autonomous.

1. GPS (Global Positioning System) Integration

- **Purpose :** Enables outdoor navigation and tracking of the RC car's location.
- **How It Works :**
  - **a.** A GPS module (e.g., NEO-6M or Ublox GPS) can be connected to the ESP32 via UART (TX/RX) communication.
  - **b.** The car can receive real-time latitude and longitude coordinates for location tracking.
  - **c.** The GPS data can be sent to the Android app, allowing users to monitor the car’s position on a map.
  - **d.** Future enhancement: Implement GPS waypoints for autonomous navigation to a specific destination.
- **Connections :**
  - **a.** GPS VCC → 3.3V (ESP32)
  - **b.** GPS GND → GND (ESP32)
  - **c.** GPS TX → RX (ESP32)
  - **d.** GPS RX → TX (ESP32)

- **Example Application :**
  - **a.** Tracking the RC car's position in an open area.
  - **b.** Sending GPS coordinates to a cloud server for remote monitoring.
  - **c.** Programming waypoints for autonomous path-following.

2. Accelerometer & Gyroscope (IMU) Integration

- **Purpose :** Helps in detecting tilt, speed, and orientation of the RC car.
- **How It Works :**
  - **a.** A MPU6050 (Accelerometer + Gyroscope) can be used to measure acceleration and angular velocity.
  - **b.** Helps in collision detection or identifying if the car flipped over.
  - **c.** Can improve navigation by stabilizing movement when driving on uneven terrain.
- **Connections :**
  - **a.** MPU6050 VCC → 3.3V (ESP32)
  - **b.** MPU6050 GND → GND (ESP32)
  - **c.** MPU6050 SDA → GPIO 21 (ESP32) (I2C Communication)
  - **d.** MPU6050 SCL → GPIO 22 (ESP32)
- **Example Application :**
  - **a.** Detecting sudden acceleration or deceleration.
  - **b.** Detecting tilt angle to prevent flipping.
  - **c.** Smoother movement by adjusting motor speed dynamically.

3. Camera (ESP32-CAM) Integration

- **Purpose :** Enables live video streaming and computer vision-based obstacle detection.
- **How It Works :**
  - **a.** The ESP32-CAM module captures real-time video and streams it via Wi-Fi.
  - **b.** The video feed can be viewed in the Android app.
  - **c.** Can be used for AI-based obstacle detection with OpenCV.
- **Connections :**
  - **a.** ESP32-CAM 3.3V → 3.3V (ESP32)
  - **b.** ESP32-CAM GND → GND (ESP32)
  - **c.** ESP32-CAM U0T (TX) → RX (ESP32)
  - **d.** ESP32-CAM U0R (RX) → TX (ESP32)

- **Example Application :**
  - **a.** FPV (First-Person View) mode in the mobile app.
  - **b.** Object recognition using AI-based models (YOLO, TensorFlow).
  - **c.** Automatic parking or lane detection for advanced automation.

- **Comparison of Additional Sensor Integrations**

Sensor	Function	Use Case
GPS	Provides location data	Navigation, waypoint following, tracking
IMU (MPU6050)	Detects movement & tilt	Collision detection, balance control
ESP32-CAM	Streams real-time video	AI-based obstacle detection, FPV mode

C. Enhanced Application Features (e.g., Speed Control, Navigation)

To improve the functionality of the RC car, several enhanced features can be added, such as speed control, navigation, and smart automation. These features will provide more precise control, better user experience, and advanced capabilities.

1. Speed Control

- **Purpose :** Allows the user to control the speed of the RC car dynamically from the Android application.
- **Implementation :**
  - **a.** Instead of just setting the motors to ON/OFF, we can use PWM (Pulse Width Modulation) to adjust the motor speed.
  - **b.** The ESP32 generates PWM signals for the L293D motor driver's enable pins.
  - **c.** The user can send speed commands (0-100%) from the Android app.
- **Connections :**
  - **a.** L293D Enable1 (Pin 1) → GPIO 32 (ESP32 PWM Output for Motor 1)
  - **b.** L293D Enable2 (Pin 9) → GPIO 33 (ESP32 PWM Output for Motor 2)
- **Arduino Code Implementation :**

```
1.
2.  #define ENA 32  // PWM for Motor 1
3.  #define ENB 33  // PWM for Motor 2
4.
5.  void setup() {
6.    pinMode(ENA, OUTPUT);
7.    pinMode(ENB, OUTPUT);
8.  }
9.
```

```
10.
11. // Function to set speed (0 to 255)
12. void setSpeed(int speed) {
13.   analogWrite(ENA, speed);
14.   analogWrite(ENB, speed);
15. }
16.
```

- **Example Application :**
  - **a.** The app slider can send commands like speed/50 to adjust the car speed dynamically.
  - **b.** Allows gradual acceleration and deceleration.

2. Navigation with GPS

- **Purpose :** Enables the RC car to navigate autonomously based on GPS waypoints.
- **Implementation :**
  - **a.** The GPS module (e.g., NEO-6M) sends location data to the ESP32.
  - **b.** The car can be programmed to follow waypoints (specific latitude/longitude coordinates).
  - **c.** The Android app can show the current car location on Google Maps.
  - **d.** This feature can be used for autonomous driving or path following.
- **Arduino Code Implementation :**

```
1.
2.  #include <TinyGPS++.h>
3.  #include <SoftwareSerial.h>
4.
5.  TinyGPSPlus gps;
6.  SoftwareSerial gpsSerial(16, 17);  // RX, TX for GPS module
7.
8.  void setup() {
9.    Serial.begin(115200);
10.   gpsSerial.begin(9600);
11. }
12.
13. void loop() {
14.   while (gpsSerial.available()) {
15.     gps.encode(gpsSerial.read());
16.     if (gps.location.isUpdated()) {
17.       Serial.print("Latitude: "); Serial.println(gps.location.lat(), 6);
18.       Serial.print("Longitude: "); Serial.println(gps.location.lng(), 6);
19.     }
20.   }
21. }
22.
```

- **Example Application :**
  - **a.** Car can drive to a set destination using GPS waypoints.
  - **b.** User can track the car in real-time on a map.

3. Automatic Obstacle Avoidance with AI (Future Enhancement)

- **Purpose :** Instead of basic ultrasonic obstacle avoidance, AI-based computer vision can be used for object detection.

- **Implementation :**

  a. Use an ESP32-CAM module to capture video.

  b. Apply OpenCV-based object detection (e.g., YOLO) to recognize obstacles.

  c. Instead of stopping, the car can detect an open path and navigate around obstacles.

- **Example Application :**

a. Car detects and avoids moving objects (e.g., people, pets).

b. More intelligent decision-making compared to basic ultrasonic sensing.

4. Mobile App Enhancements

To support these advanced features, the Android application can be improved with :

- Speed Control Slider (Adjusts PWM speed).
- GPS Navigation Map (Shows live car location).
- Camera Feed (For FPV driving or AI detection).
- Voice Control (Commands like *"Move forward"*, *"Turn left"*).

References

A. ESP32 wroom-32

ESP32 wroom-32 datasheet PDF : [www.espressif.com]

(https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf)

B. L293D

L293D Push-Pull four channel drivers with diodes : [www.st.com]

(https://www.st.com/en/motor-drivers/l293d.html)

C. HC-SR04 Ultrasonic Sensor

HC - SR04 Ultrasonic Ranging Module PDF : [http://www.sparkfun.com]

(https://cdn.sparkfun.com/datasheets/sensors/proximity/hcsr04.pdf)

D. LM2596 DC to DC Buck Converter

LM2596 data sheet : [www.ti.com]

(https://www.ti.com/product/LM2596)