Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

UltraNav

Project Engineering

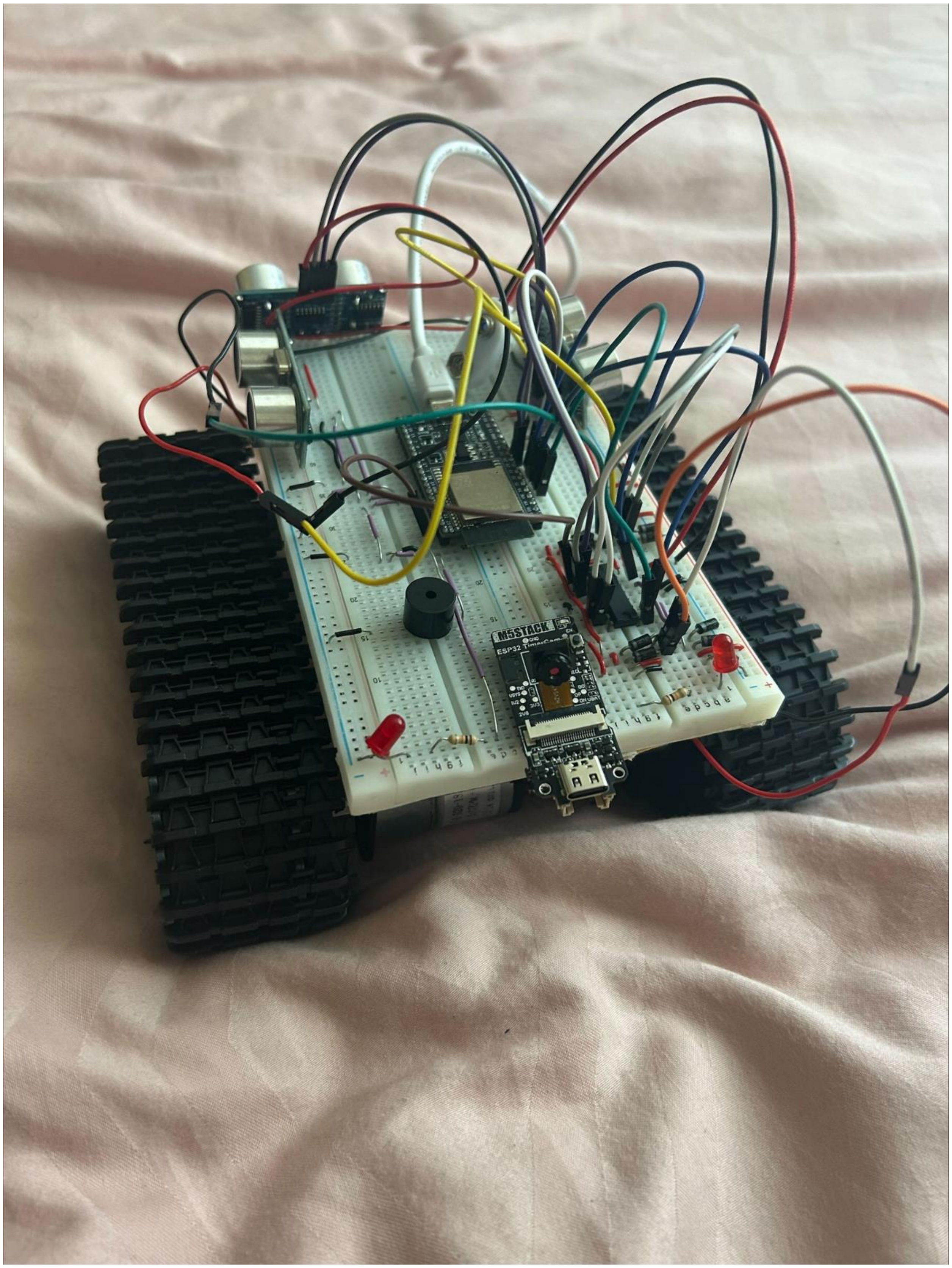
Year 4

Denis Johnson G00403126

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Atlantic Technological University

2024/2025


M5STACK
ESP32 TimerCam

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

____

Table of Contents

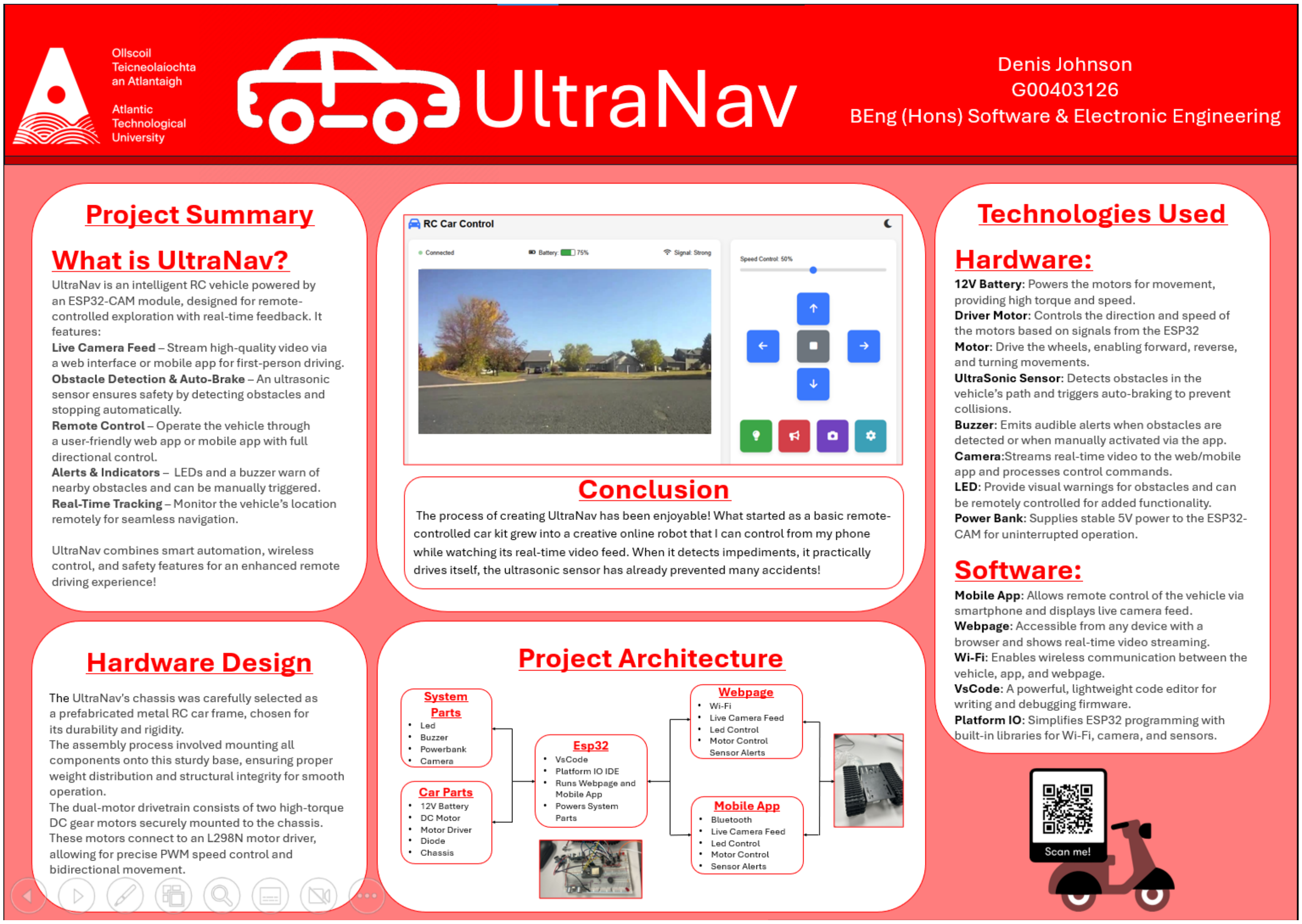
1 Summary

UltraNav is an intelligent RC vehicle powered by an ESP32, designed for remote controlled exploration with real-time feedback. It features a Live Camera Feed from the M5Stack ESP32 Timer Camera to display Stream high-quality video via a webpage and mobile app for first-person driving [17]. Obstacle Detection & Autobrake using an ultrasonic sensor. This ensures safety by detecting obstacles and stopping automatically. Parking sensors where the user will be alerted while reversing if an object is near and automatically slow the car down. Remote Control to operate the vehicle through a user-friendly web app or mobile app with full directional control. Alerts & Indicators using LEDs and a buzzer warning of nearby obstacles can be manually triggered.

The UltraNav is made this by connecting 3 ultrasonic sensors, 2 LEDs, a buzzer, 2 dc motors and a camera. I used an ESP32 to program and power the parts. It uses a motor driver and a 9V battery to power the motors and a 5V power bank to power the ESP32. The UltraNav is programmed using C++ through Visual Studio Code using Platform IO to debug and upload the ESP32. [15]

I decided to make this project as I have a big interest in building projects instead of just coding it. I decided to pick a car because I've always been interested in them. My goal is to learn more about hardware and how parts work together. On the software side I was interested in finding out more about how much you could do on a webpage and how to make a mobile app and how to connect it to the project. The approach I had to the project was start out with what I knew how to do then do heavy research on the things I was unfamiliar with. In this project I got mostly everything I wanted to work that I planned from the start. I also learned tons about hardware and software while also doing lots of research about the parts. I was also impressed with how much I was interested in the project the whole time, always looking on how to improve it.

2 Poster



Ollscoil Teicneolaíochta an Atlantaigh
Atlantic Technological University

UltraNav

Denis Johnson
G00403126
BEng (Hons) Software & Electronic Engineering

Project Summary

What is UltraNav?

UltraNav is an intelligent RC vehicle powered by an ESP32-CAM module, designed for remote-controlled exploration with real-time feedback. It features:

Live Camera Feed – Stream high-quality video via a web interface or mobile app for first-person driving.
Obstacle Detection & Auto-Brake – An ultrasonic sensor ensures safety by detecting obstacles and stopping automatically.
Remote Control – Operate the vehicle through a user-friendly web app or mobile app with full directional control.
Alerts & Indicators – LEDs and a buzzer warn of nearby obstacles and can be manually triggered.
Real-Time Tracking – Monitor the vehicle's location remotely for seamless navigation.

UltraNav combines smart automation, wireless control, and safety features for an enhanced remote driving experience!

Conclusion

The process of creating UltraNav has been enjoyable! What started as a basic remote-controlled car kit grew into a creative online robot that I can control from my phone while watching its real-time video feed. When it detects impediments, it practically drives itself, the ultrasonic sensor has already prevented many accidents!

Technologies Used

Hardware:

12V Battery: Powers the motors for movement, providing high torque and speed.
Driver Motor: Controls the direction and speed of the motors based on signals from the ESP32
Motor: Drive the wheels, enabling forward, reverse, and turning movements.
UltraSonic Sensor: Detects obstacles in the vehicle's path and triggers auto-braking to prevent collisions.
Buzzer: Emits audible alerts when obstacles are detected or when manually activated via the app.
Camera:Streams real-time video to the web/mobile app and processes control commands.
LED: Provide visual warnings for obstacles and can be remotely controlled for added functionality.
Power Bank: Supplies stable 5V power to the ESP32-CAM for uninterrupted operation.

Software:

Mobile App: Allows remote control of the vehicle via smartphone and displays live camera feed.
Webpage: Accessible from any device with a browser and shows real-time video streaming.
Wi-Fi: Enables wireless communication between the vehicle, app, and webpage.
VsCode: A powerful, lightweight code editor for writing and debugging firmware.
Platform IO: Simplifies ESP32 programming with built-in libraries for Wi-Fi, camera, and sensors.

Hardware Design

The UltraNav's chassis was carefully selected as a prefabricated metal RC car frame, chosen for its durability and rigidity.
The assembly process involved mounting all components onto this sturdy base, ensuring proper weight distribution and structural integrity for smooth operation.
The dual-motor drivetrain consists of two high-torque DC gear motors securely mounted to the chassis. These motors connect to an L298N motor driver, allowing for precise PWM speed control and bidirectional movement.

Project Architecture

System Parts
- Led
- Buzzer
- Powerbank
- Camera

Car Parts
- 12V Battery
- DC Motor
- Motor Driver
- Diode
- Chassis

Esp32
- VsCode
- Platform IO IDE
- Runs Webpage and Mobile App
- Powers System Parts

Webpage
- Wi-Fi
- Live Camera Feed
- Led Control
- Motor Control
- Sensor Alerts

Mobile App
- Bluetooth
- Live Camera Feed
- Led Control
- Motor Control
- Sensor Alerts

Scan me!

3 Introduction

The UltraNav project showcases a unique combination of robotics and IoT technology to provide an intelligent remote-controlled vehicle that can be used for exploring and stream real-time data. This system uses an ESP32-CAM microcontroller to provide live video streaming with a safety system that includes obstacle detection and autonomous braking. Unlike typical RC cars that are controlled only by the user, UltraNav allows the vehicle to recognize its environment and engage with it in a useful manner, therefore treating a simple car as a smart car, rather than just a car.

The motivation for the project came from wanting to see the connection between the theoretical programming knowledge learned in class and applying that knowledge to actual hardware. Software only projects certainly present valuable learning opportunities, and hardware only perhaps even greater learning opportunities, but when the two are integrated, learning opportunities and challenges abound. The design of UltraNav brought together the processing of sensor data, motor control, wireless communication, and all under one umbrella. UltraNav is intended not only be an educational opportunity for developing embedded systems, but it also demonstrates how inexpensive, off-the-shelf components can be integrated into complex IoT applications.

UltraNav's primary objectives are to offer real-time operational feedback and user safety. The system offers high-definition video streaming into a proprietary web interface, allowing first-person-like navigation. Simultaneously, ultrasonic sensors continuously scan the vehicle's surroundings, triggering automatic braking upon detecting obstacles and providing audible/visual alerts during parking. These functions are supported by a robust hardware platform, including DC motors for movement, LEDs and buzzers for notifications, and a dual power system for reliable performance.

Technically, the project shows how the ESP32 platform is suited for multitasking—Wi-Fi communication, video processing, and sensor management—at the same moment. The design combines efficient C++ programming of device control with web technology to enable remote control, showing an end-of-end IoT workflow. The subsequent report will describe the design process, implementation problems, and performance results, with the added discussion on potential future applications such as autonomous navigation and incorporation with machine learning. UltraNav thus stands as a testament to the exciting potential at the confluence of computer vision, robotics, and networked devices.

4 Background

My passion for building and studying cars developed at a young age, when I would spend hours reading about and experimenting with model cars. This early passion for automotive systems branched out into a greater interest in robotics and embedded electronics, particularly in how modern cars incorporate smart technologies. My professional experience with Valeo, the world leader in automotive technology, further broadened this interest by exposing me to practical experience with advanced camera and sensor systems utilized in real-world automotive applications.

During my time at Valeo, I spent most of my time working with advanced driver-assistance systems (ADAS), wherein I set up vision systems, tested ultrasonic sensors, and debugged sensor fusion algorithms. Working on it taught me a lot about how automotive-grade systems process real-time data to facilitate features like parking assistance and collision avoidance. Working with engineers on these cutting-edge technologies taught me the value of solid hardware-software integration and good system design - lessons that directly affected my management of the UltraNav project.

UltraNav is the perfect blend of

my professional expertise and childhood curiosity. Although Valeo's products are designed for full-size vehicles, I had been thinking in terms of a platform of reduced size that would be able to replicate similar intelligent behavior using cost-effective hardware. The ESP32-CAM module was the perfect choice, where camera capability was combined with processing capability in a small package - a situation like the embedded systems that I have worked with in professional automotive applications.
My exposure to ultrasonic sensors at Valeo also informed UltraNav's obstacle detection system, namely sensor placement and signal processing.

This project helped me to transfer professional-level ideas to house construction, bridging the world between industrial car technology and hobby robotics. The issues I faced - from reducing latency during video streaming to offering reliable motor control - were similar to those in real-world engineering that I had seen at Valeo, but just on a smaller magnitude.

UltraNav thus represents both an extension of my life-long interest in cars and a worthwhile exercise of the technical skills that I acquired while working in the automobile sector.

5 Project Architecture

UltraNav is an architecture project that seamlessly integrates hardware and software components. The ESP32 microcontroller is at the heart of it, acting as the central processing unit which is coded with VS Code with Platform IO IDE to drive the system
and handle communication between parts [15]. Hardware is divided into two major divisions: the System Parts, consisting of an LED, buzzer, power bank, and a live video stream camera, and the Car Parts, including a 12V battery, DC motors, motor driver, diodes, and a mobility-providing chassis. ESP32 supports wireless access through Wi-Fi and
Bluetooth for both webpage and mobile application-based interfaces. Such interfaces provide the users with a real-
time video display, LED control, and motor control, along with real-time sensor alerts to detect collisions. Such a system ensures an integrated and Response-
oriented system with the proper provision of computation capability, sensor integration, and user access for a smart remote vehicle.

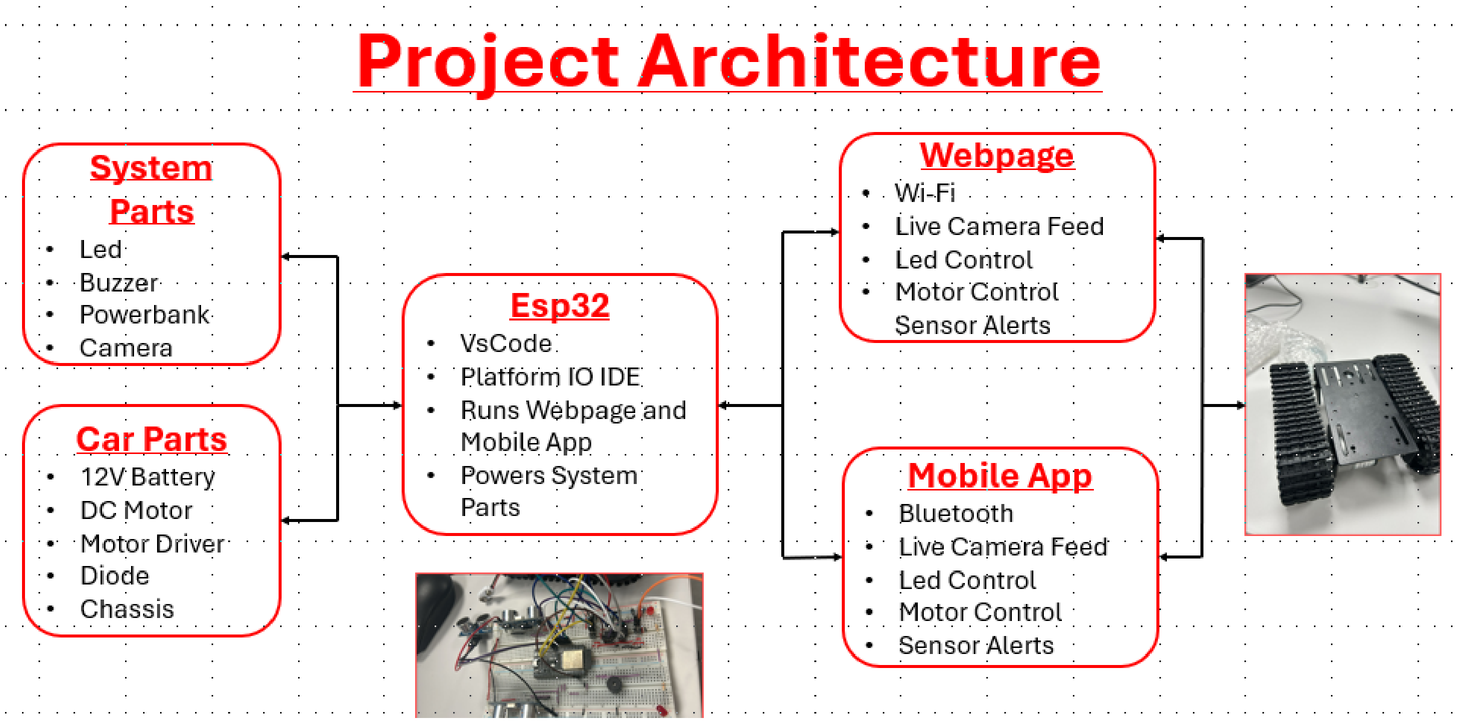


Figure 5-1 Architecture Diagram

6 Project Plan

The UltraNav project follows a formal development schedule to ensure systematic progress. The project initiation phase began with the project proposal (KAM-2), wherein we set up the basic objectives: creating an intelligent RC vehicle with live video streaming, real-time obstacle sensing, and two control interfaces. This was followed by component acquisition (KAM-3), in which we sourced all the needed components like the ESP32-CAM module, ultrasonic sensors, motor drivers, and power systems with 20% additional key components ordered as spares to balance potential hardware failure.

The physical structure (KAM-5) included crafting the motorized chassis with care, ensuring consistent weight distribution and stability. We placed three ultrasonic sensors in the perfect positions - front, rear, and side - to provide full perception of the environment. The power system was crafted with extreme caution employing individual voltage regulators for the motors and control electronics in order to prevent interference.
Electrical safeguards like diodes and capacitors were employed for safeguarding sensitive components from voltage spikes and noise. [1]

Software integration began with the implementation of the web control interface (KAM-6), which was built with a combination of HTML, CSS, and JavaScript to achieve a responsive dashboard. This interface provides real-time motor control, sensor feedback, and system status monitoring. Camera integration (KAM-7) involved extensive optimization to provide seamless video streaming wherein we compromised resolution and frame rate to weigh acceptable latency against bandwidth conservation. For obstacle detection (KAM-8), we employed robust filtering algorithms in managing sensor readings as well as adjustable thresholds for automatic braking and warning systems. [17]

Mobile application development (KAM-11) expanded our control options by the addition of Bluetooth connectivity, with particular emphasis on creating an intuitive user interface that mimics the functionality of the web dashboard with the addition of mobile-specific options like haptic feedback. Along the process, we made frequent checks (KAM-9, KAM-12) to review progress, test for system stability across various scenarios, and gather end-user feedback to refine in stages. These steps of quality assurance revealed and worked out various issues of technology like reducing video delay from original 2 seconds below 400 milliseconds by optimization using firmware and electromagnetic interference among the motors and the sensors through proper circuit designing and shielding. [14]

The staged rollout of the project allowed for orderly hardware

assembly integration and staged software feature rollout, scheduling major functions like real-time control and safety features first before expanding later on interface refinement and optimization. The structured development approach ensured that each module was thoroughly tested
and verified before being incorporated into the integrated system, thus yielding a stable and reliable intelligent vehicle platform.

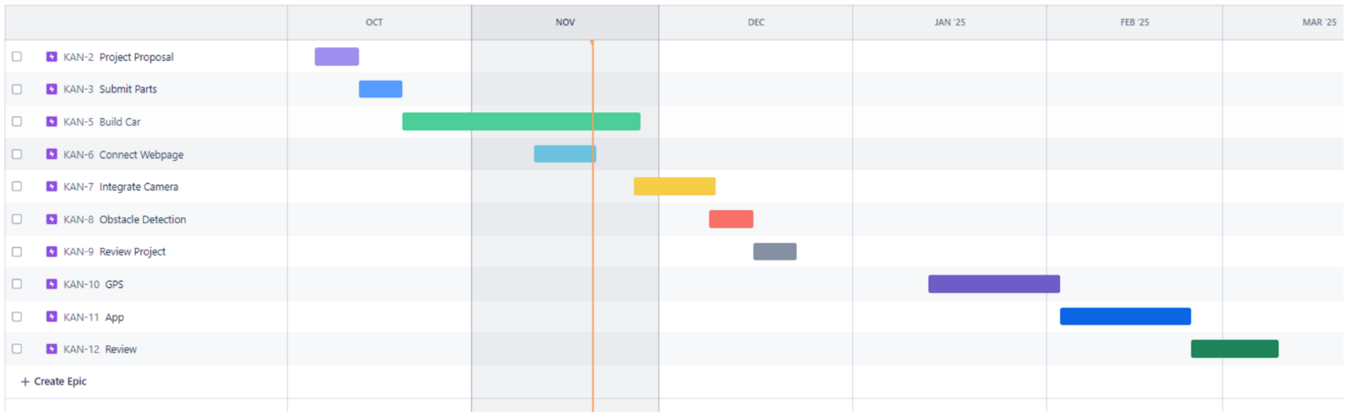
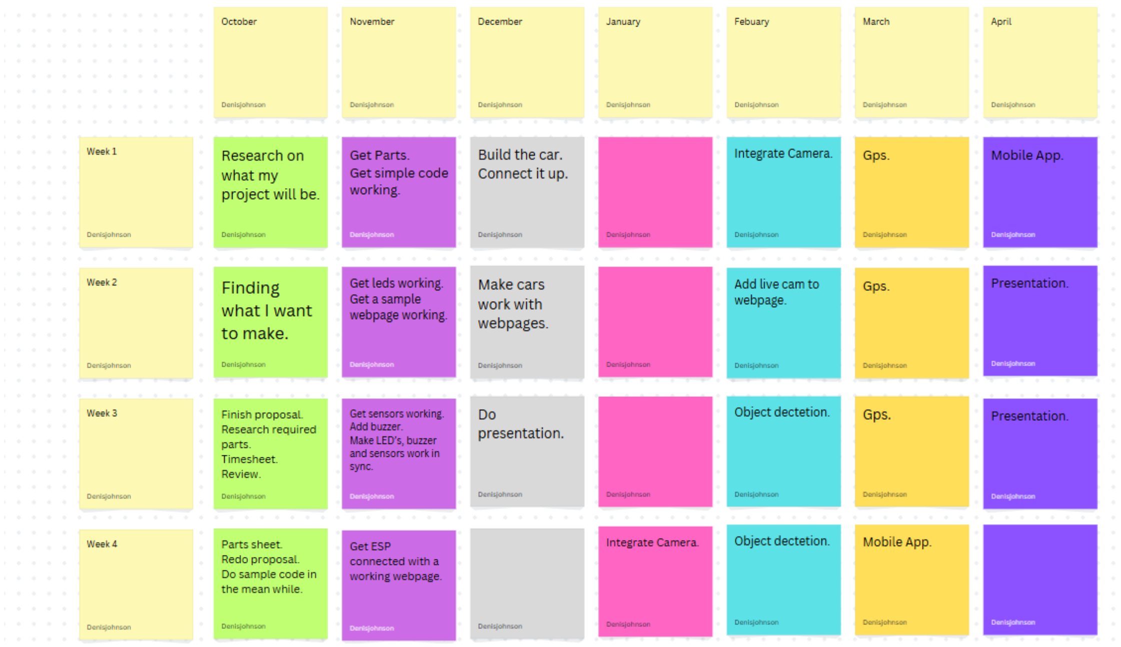


Figure 6-1 Project Plan



	October	November	December	January	Febuary	March	April
Week 1	Research on what my project will be.	Get Parts. Get simple code working.	Build the car. Connect it up.		Integrate Camera.	Gps.	Mobile App.
Week 2	Finding what I want to make.	Get leds working. Get a sample webpage working.	Make cars work with webpages.		Add live cam to webpage.	Gps.	Presentation.
Week 3	Finish proposal. Research required parts. Timesheet. Review.	Get sensors working. Add buzzer. Make LED's, buzzer and sensors work in sync.	Do presentation.		Object dectetion.	Gps.	Presentation.
Week 4	Parts sheet. Redo proposal. Do sample code in the mean while.	Get ESP connected with a working webpage.		Integrate Camera.	Object dectetion.	Mobile App.	

Figure 6-2 Project Plan

7 LED

Purpose and Function within the System:

The LED subsystem of this project is intended to offer immediate, intuitive visual user and observer feedback regarding the operation state of the robot and that of its surroundings. This type of feedback is essential both to usability and to safety since, through it, users are immediately able to evaluate whether the robot is on or off, in sensing mode or in a secure/idle condition. The LED is used as a "status light," like the ones on most commercial electronic products, but with sensor-based intelligence. [2]

Modes of Operation:

1. OFF Mode:

In this mode, the LED is off. This is typically a sign that the robot is turned off, in standby, or no immediate feedback is necessary. It conserves power and avoids unnecessary distraction when the robot is not actively used.

2. ON Mode:

In permanently ON LED mode, when the LED is lit permanently, it shows that the robot is powered on and ready to use. This mode is useful in making sure at a glance if the system is operating and ready to take commands or perform jobs. It may also be used to indicate successful startup or completion of an automated test routine.

3. ULTRASONIC (Dynamic) Mode:

This is the most sophisticated mode. In this mode, the LED's action is dynamically controlled in response to real-time signals from the robot's ultrasonic proximity sensors. The LED will flash at different rates based on the object's proximity:

•Fast Blinking: Indicates a critical range—something is close to the vehicle, and immediate action may be required to avoid a collision.

• Slow Blinking: Indicates a warning state—an object is at a cautionary distance, but not yet at a critical level.

• No Blinking (Off): Indicates that the path is clear and no obstacles are in the warning or critical range.

This dynamic feedback allows users to understand the robot's surroundings and potential hazards without needing to look at a screen or interface, increasing the system's accessibility and usability.

Integration with Other Subsystems:

The LED subsystem is closely coupled with the ultrasonic sensor subsystem. The sensors are continuously sensing the surroundings, and the measurements are processed by the main control logic. Based on measurements, the LED subsystem is instructed to change its mode or blink pattern. This coupling is such that the LED always shows the most recent and suitable information about the robot's surroundings.

In addition, the LED subsystem is designed to be oblivious to the global flow of control. That is, irrespective of whether the robot is engaged in other activities (e.g., movement or buzzing), the LED is able to provide real-time feedback without causing delays or blocking other processes.

Design Considerations:

• Visibility: The chosen LED is bright enough to be noticed under various lighting conditions, and feedback will always be apparent to the user.

• Responsiveness: The system is designed to switch the state of the LED as quickly as possible based on changes in sensor data, providing timely warnings.

• Non-blocking Operation: The logic controlling the LED does not contain long delays or blocking code, thus it does not interfere with the ability of the robot to process other inputs or outputs.

• Modularity: The LED subsystem is made modular as a standalone module and is easy to replace or expand in the future (e.g., to accommodate RGB LEDs or more complex signaling sequences).

Challenges and Solutions:

Challenge 1: Synchronizing LED Feedback with Sensor Data

One of the challenges was to ensure that the blinking of the LED accurately reflected the most current sensor readings, Especially if the robot is moving at high speeds or crashing into obstacles rapidly. This was addressed by refreshing the LED state within the primary control loop to ensure that feedback is always calculated from the most current information.

Challenge 2: Avoiding User Confusion

The second challenge was designing the blink patterns in a manner that users are able to easily distinguish between warning and critical states. Through user testing, the blink rates were calibrated to be clearly distinct and distinguishable even upon rapid glance.

Challenge 3: Power Efficiency

Since the robot might be battery powered, power draw for the LED had to be kept minimal. OFF mode and smart blink logic reduce power usage when brightness is not needed.

Summary

Generally, the LED subsystem is a critical part of the robot's user interface, with efficient, real-time visual feedback regarding the status of the robot and its environment. Its look is a balance between visibility, responsiveness, and power, and it is also robust enough to be used as a complement to the robot's other.

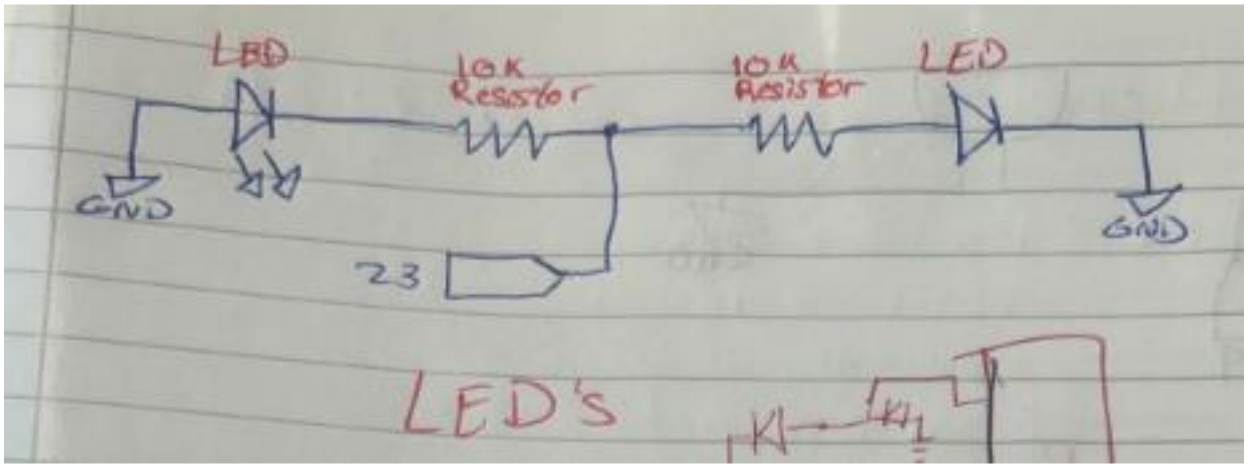
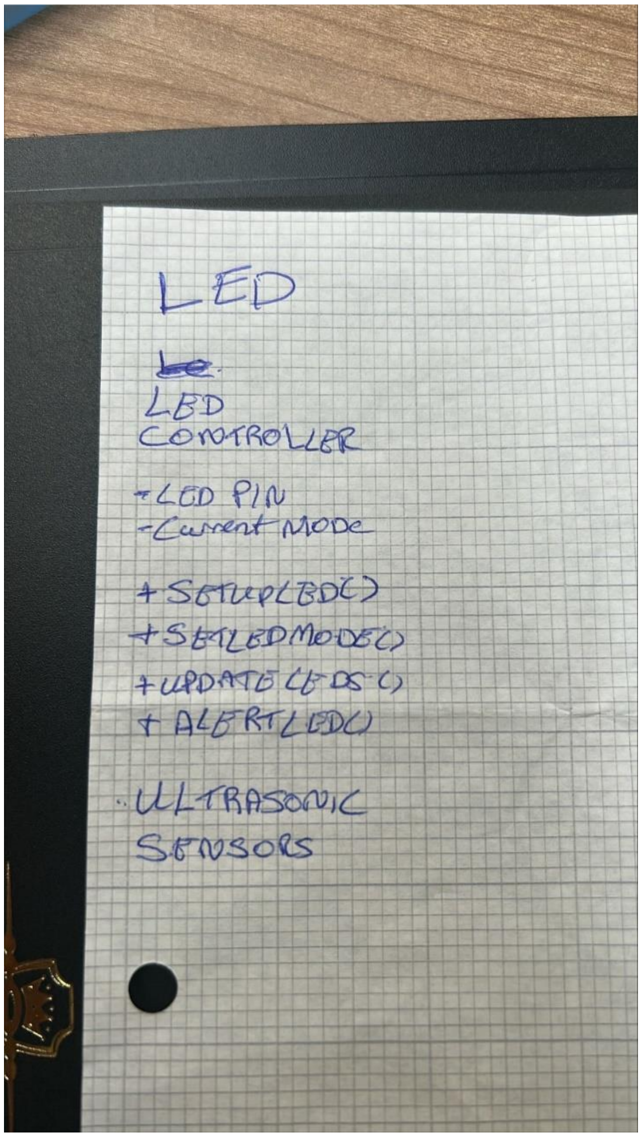


Figure 7-1 LED Diagram



LED

~~Le~~

LED
CONTROLLER

-LED PIN
-Current MODE

+SETUPLED()
+SETLEDMODE()
+UPDATELEDS()
+ALERTLED()

ULTRASONIC
SENSORS

Figure 7-2 LED Design Diagram

8 ESP32

Purpose and Function in the System:

The ESP32 microcontroller is the "brain" of the entire robotic system. Its purpose is to manage all of the hardware modules, process sensor data, execute control algorithms, and regulate communication with the user. The ESP32 dual-core processor, on-board Wi-Fi, and the high GPIO function make it a good choice for a project requiring real-time control, wireless networking, and combining multiple subsystems. [13]

Core Responsibilities:

1. Coordination of the Subsystems:

The ESP32 is accountable for the startup and management of all peripheral subsystems like motors, ultrasonic sensors, LED, and buzzer. It initializes each of the subsystems on startup such that they can be utilized whenever they are needed and coordinates their flawless interaction when it functions.

2. Sensor Data Acquisition and Processing:

The ESP32 continually captures data from the ultrasonic sensors, processes the data, and makes its decision about movement, collision, and user feedback depending on the acquired data.

3. Actuator Control:

Based on sensor feedback and user commands, the ESP32 controls the motor (for motion), the LED (for visual display), and the buzzer (for audio display). It ensures that such actuators respond timely and properly to changing conditions.

4. Wireless Communication and User Interface:

ESP32 executes a Wi-Fi access point and web server, which allows users to remotely observe the robot's status and send control commands via a web interface. This offers wireless, real-time interaction with the robot using any web browser-enabled device. [14]

5. Main Control Loop:

The ESP32 runs a continuous main loop that updates sensor values, runs logic, and services user input. The loop is optimized to be non-blocking and efficient to enable the system to always react to events.

System Integration and Modularity

The ESP32 software is modularly structured, with each of the hardware subsystems (motors, sensors, LED, buzzer) coded as separate modules. ESP32 has all the modules included in the main program where it requests their setup and update methods based on need. This keeps the system traceable, manageable, and open to further growth in the future.

ESP32 is also the communication hub, passing data between the sensors, actuators, and the user interface. For example, when a user presses a button on the web interface, the ESP32 receives the command through Wi-Fi, decodes it, and sends the respective signals to the motors or other actuators.

Design Considerations:

- Real-Time Responsiveness: The main loop of ESP32 is specially designed to avoid long delays or blocking operations so that sensor data are processed, and actuators are updated as soon as possible.

- Wireless Connectivity: ESP32's built-in Wi-Fi capabilities are utilized to provide a robust and friendly web interface for remote control and monitoring without the need for external hardware.
- Power Management: The ESP32 can operate on battery power, and hence attention was paid to the fact that the main loop and peripheral management are optimized so that there is no unnecessary power consumption.
- Scalability and Flexibility: The software is designed in a modular fashion, making it simple to add new features or hardware components in the future, e.g., more sensors or more sophisticated control algorithms.

Challenges and Solutions:

Challenge 1: Ensuring Reliable Wi-Fi Communication

The need for a stable Wi-Fi connection and a fast-responding web server was critical for user interactions. This was addressed via the usage of ESP32's powerful Wi-Fi libraries and also providing simple-to-understand feedback within the web interface as well as the serial monitor for debugging.

Challenge 2: Handling Multiple Real-Time Tasks

Timing of sensor readings, actuator control, and web server feedback with no delay or missed events required careful planning. The solution was to use non-blocking code and modular updates, thus allowing the ESP32 to multitask comfortably.

Challenge 3: Pin Assignment and Hardware Integration

Since multiple peripherals were being utilized, proper planning was required so that there were no pin conflicts, and all the devices could be controlled in a reliable manner. This issue was resolved by pre-planning all the pin assignments and using only output-capable pins for actuators.

Summary

ESP32 microcontroller makes up the heart of the robotic system, providing computation, connectivity, and flexibility adequate to incorporate all the hardware as well as the software elements. Its effective modular structure ensures trouble-free functioning, real-time response, and user-friendly interface and therefore is the ideal platform for this and subsequent robotics projects.

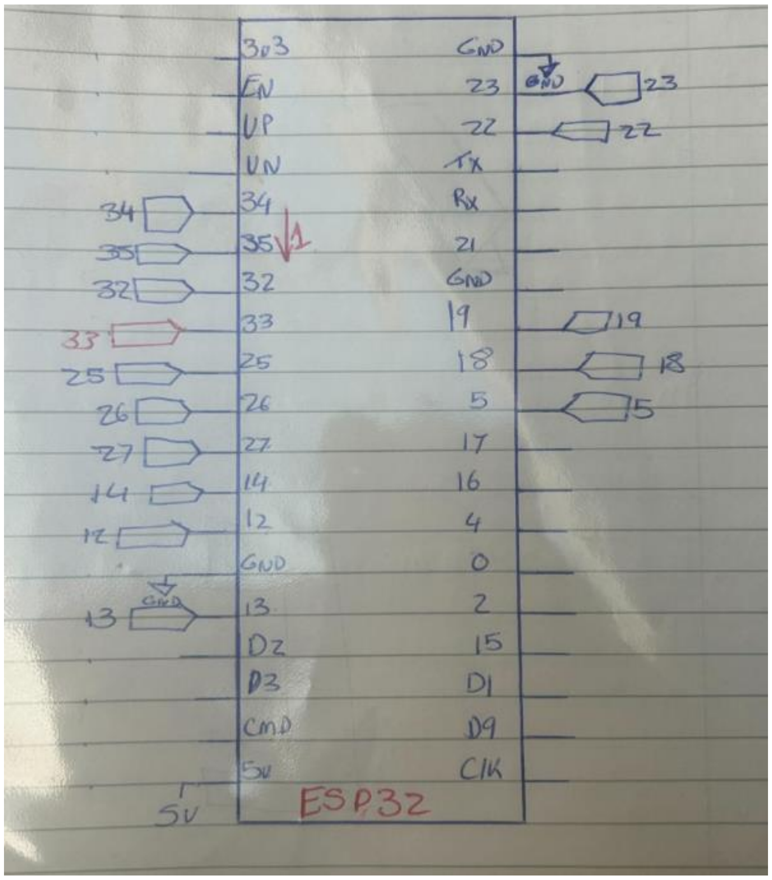


Figure 8-1 ESP32 Diagram

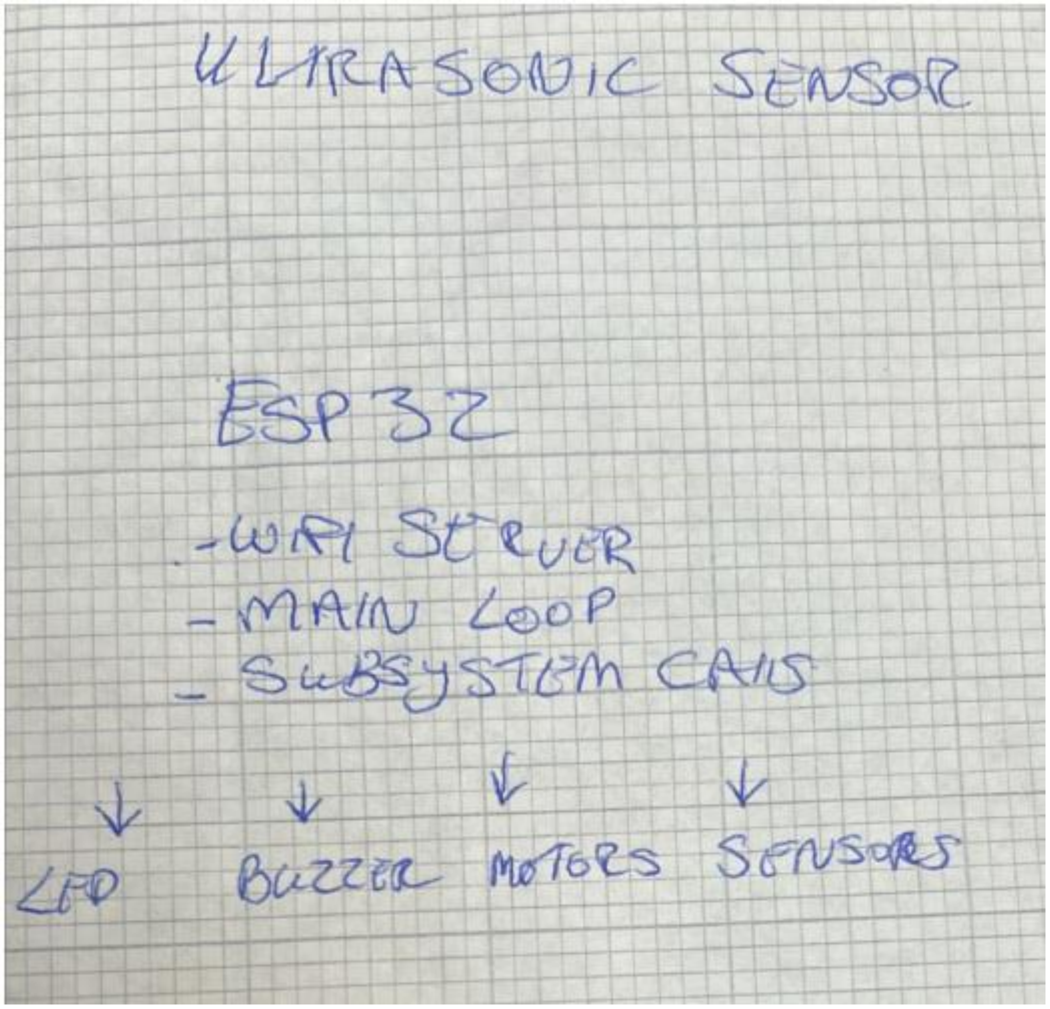


Figure 8-2 ESP32 Design Diagram

9 Buzzer

Purpose and Role within the System:

The buzzer system is designed to provide users and observers with instant, audible feedback regarding the status of the robot and its environment. Such feedback is especially important for informing users of possible hazards, like obstacles in the vicinity, or for confirmation that a command has been received and processed. The buzzer is an "audio signal," complementing visual feedback from the LED and contributing to the overall usability and safety of the system. [13]

Modes of Operation:

1. OFF Mode:

The buzzer is quiet in this mode. This is the default mode where no alarm or feedback is needed, avoiding unnecessary sound generation and conserving power.

2. ON (One-Shot) Mode:

The buzzer emits a single, short beep. This mode is typically used to acknowledge a user command or to signal a specific event, such as successful initialization or task completion. The one-shot beep provides clear, unambiguous feedback without intrusiveness.

3. ULTRASONIC (Dynamic) Mode:

In this operating mode, the buzzer is dynamically controlled based on real-time feedback from ultrasonic sensors. The buzzer beeps at different rates as a function of how close an object is read:

• Rapid Beeping: Indicates a critical proximity—an object is very close to the robot, and immediate action may be required to avoid a collision.

• Slow Beeping: Indicates a warning state—an object is at a caution distance, but not at a critical one.

• Silence: Indicates that the path is clear and there are no obstacles in the warning or critical range.

This dynamic feedback allows individuals to understand the robot's environment and potential hazards without having to look at a screen, making the system more convenient and efficient in noisy or visually dense environments.

Integration with Other Subsystems:

The buzzer subsystem is tightly integrated with the ultrasonic sensor subsystem. Sensors are constantly sensing the environment, and their input is processed by the main control logic. Based on these inputs, the buzzer subsystem is instructed to change its mode or beeping sequence. This integration ensures that the buzzer will always display the current and relevant information about the robot's surroundings.

Furthermore, the buzzer subsystem is also coded to function independently from the principal control flow. In other words, irrespective of whatever activity the robot is engaged in (e.g., moving or modifying the LED), the buzzer can send feedback in real time without creating delays or impeding other functions.

Design Considerations:

• Audibility: The buzzer was selected and placed so that it is loud enough so that alarms would always be audible.

• Responsiveness: The system applies the state of the buzzer to be updated as quickly as possible once the sensor data is changed, providing timely alerts.

• Non-blocking Operation: The control logic of the buzzer does not contain inordinate delays or blocking code and therefore does not interfere with the robot's ability to process other outputs or inputs.

• Modularity: The buzzer subsystem is implemented as an independent module, so it is easy to replace or expand in the future (e.g., to support different alert patterns or multiple buzzers).

Challenges and Solutions:

Challenge 1: Avoiding Annoying or Confusing Alerts

One was making the beeps clearly enough so that users could easily recognize warning and critical states without the buzzer being obnoxious or overwhelming. Through means of user testing, beep rates and durations were changed to be clearly different and easily recognizable, even in noisy environments.

Challenge 2: Synchronizing Buzzer Feedback with Sensor Data

Preventing the buzzer from beeping inaccurately from the latest sensor data, especially when the robot was moving at a high speed or encountering numerous obstacles consecutively, was another problem. This was addressed by updating the buzzer state within the control loop, ensuring any feedback is derived from the newest data.

Challenge 3: Power Efficiency

Since the robot can be battery-powered, there was a need to prevent the buzzer from consuming extra power. The energy-saving beeping logic and the OFF-mode assist in minimizing the use of power when warnings are not required.

Summary

In short, the buzzer subsystem is a critical part of the robot's user interface, providing unmistakable audible, real-time feedback regarding the robot's status and environment. Its responsiveness, efficiency, and audibility are well balanced, and it is reliable enough to function properly in conjunction with the other robot subsystems. The modularity also makes the buzzer function easily adaptable or expandable in future project versions.

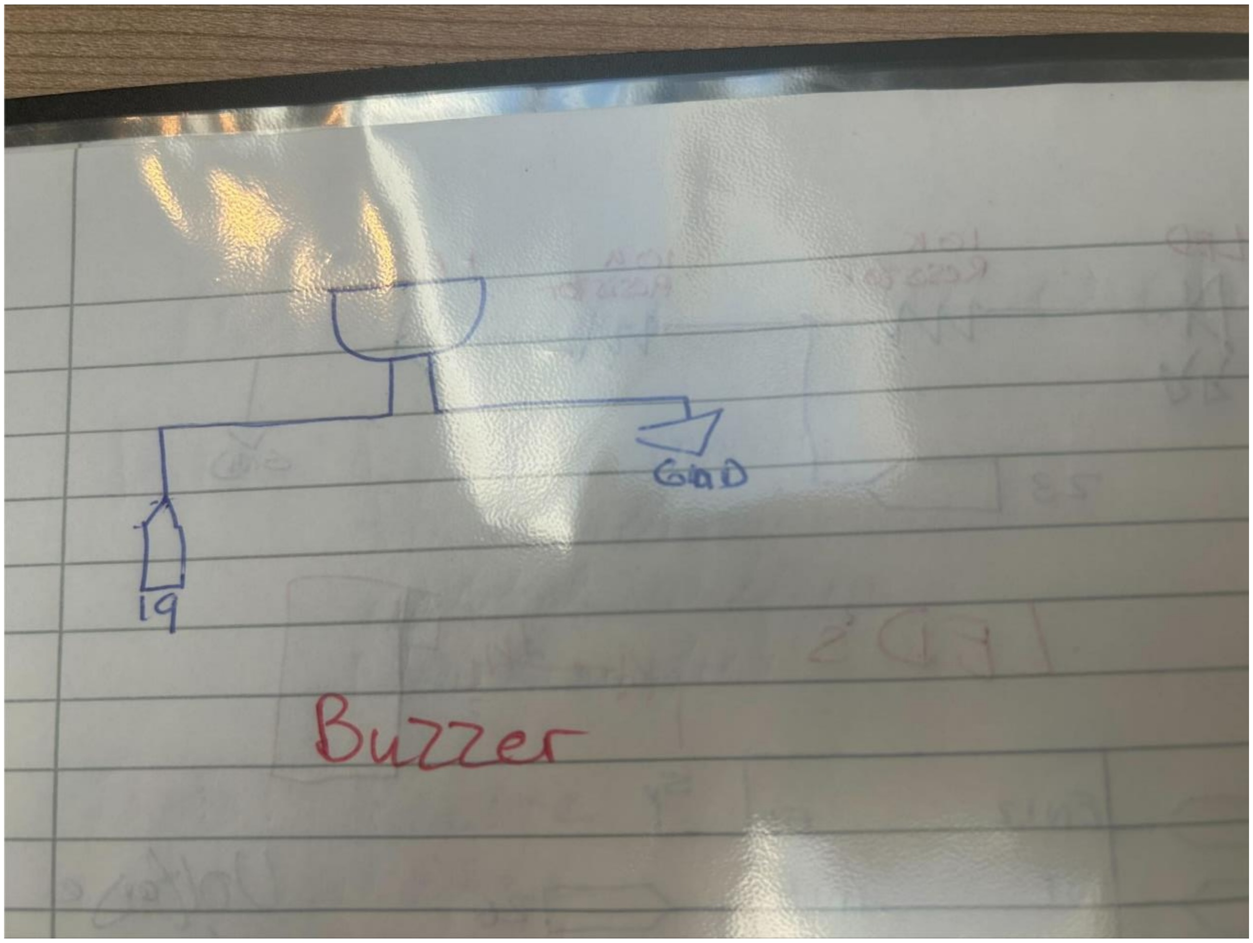


Figure 9-1 Buzzer Diagram

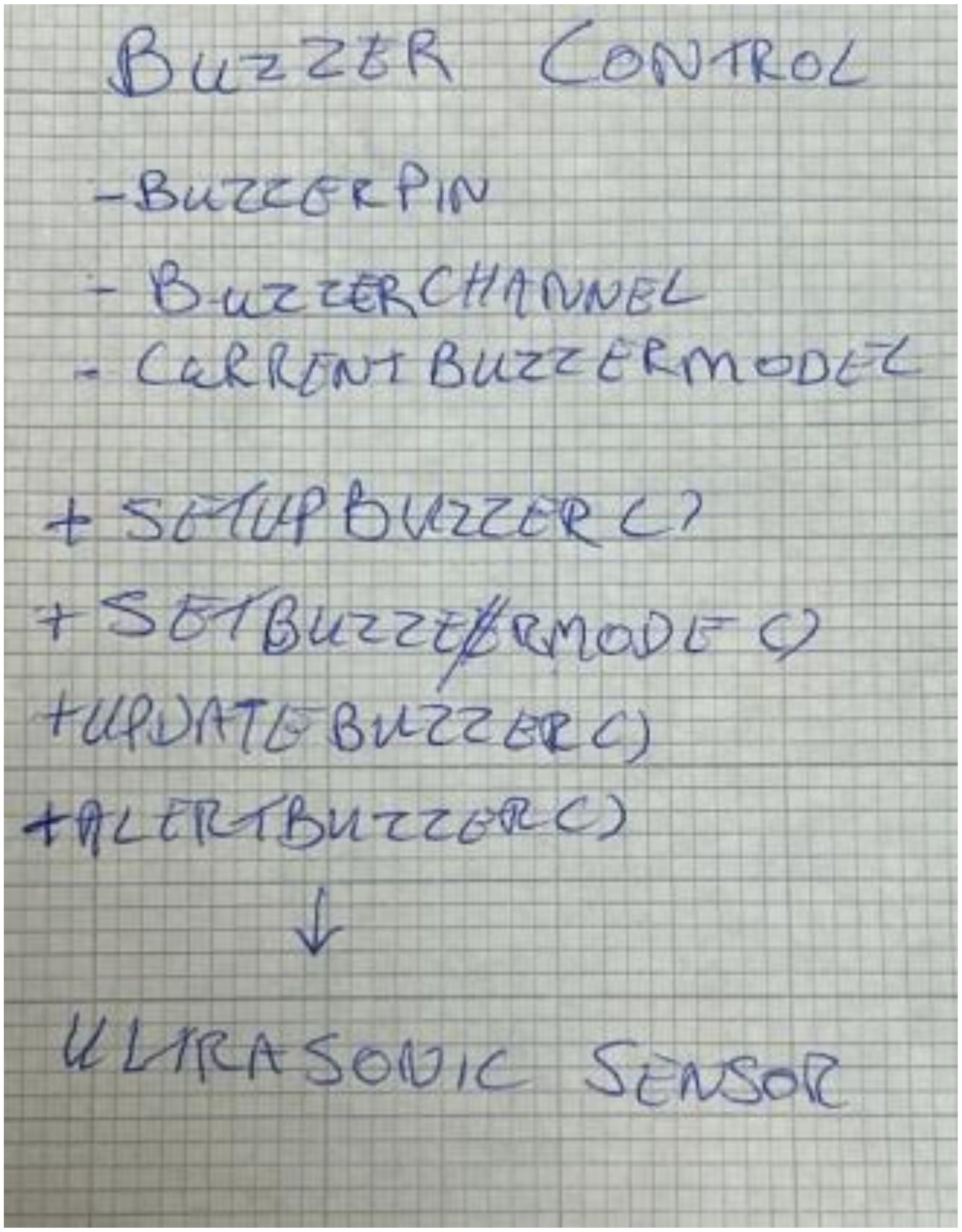


Figure 9-2 Buzzer Design Diagram

10 Ultrasonic Sensor

Purpose and Role in the System:

The ultrasonic sensor system is a vital component of the robot and enables it to perceive its environment and detect obstructions in real time. Sensing distances to objects in various directions (commonly right, front, and back), sensors provide the input necessary for secure navigation, collision detection, and smart feedback to the user. This feature is essential to autonomous or semi-autonomous navigation, as well as to improving user awareness and control. [13]

Principle of Operation:

Ultrasonic sensors work by emitting a high-frequency ultrasonic pulse and measuring the time taken by the echo to return after bouncing off an object. The time-of-flight of the ultrasonic sound is directly proportional to the sensor-object distance. This method is precise, cost-effective, and appropriate for sensing a wide range of obstacles regardless of color or lighting.

Modes of Operation

Three ultrasonic sensors are deployed on the vehicle, each of which is located in one direction (rear, left, and right). This multi-directional setup allows the robot to perceive its surroundings completely and respond to obstacles from any direction.

The sensors take continuous distance measurements, make regular readings, and provide up-to-date data to the central controller. Data from the sensors are used for:

- Generate LED and buzzer warnings visually naturally when detected obstacles reach certain levels.
- Influence the motion of automobiles, for example, braking or changing course to avert collisions.
- Provide real-time feedback to the user via the web interface.

Incorporation with Other Subsystems:

- LED Subsystem: The blink rate of the LEDs is controlled by the proximity of the objects being detected by the sensors.
- Buzzer Subsystem: The buzzer gives warning beeps or critical beeps based on the range of an object from the robot.
- Motor Control: The robot can be automatically guided or stopped in case an object is detected within a critical range.

Design Considerations:

- Accuracy and Reliability: The sensors are positioned and tuned to prevent blind spots and offer maximum detection capability. Proper calibrations are performed to avoid any variation in distance measurements being made.
- Responsiveness: The system has the capability of taking frequent readings and adjusting the robot's action in real-time so that barriers can be discovered and addressed as quickly as possible.
- Non-blocking Operation: Sensor reading logic is implemented such that it won't block or delay other processes, so the robot remains sensitive to user requests and other sensors.
- Environmental Robustness: Ultrasonic sensors are chosen as they can operate under different lighting conditions and can detect objects irrespective of transparency or color.

Challenges and Solutions:

Challenge 1: Dealing with Noisy or Unstable Readings

Ultrasonic sensors occasionally offer unstable readings due to environmental noise, soft or sloping surfaces, or crosstalk from other sensors. This was addressed by:

- Sending short, well-timed pulses and waiting for clean echoes.
- Filtering out false readings from the main control logic, e.g., by removing sudden, unreal distance changes.

Challenge 2: Coordinate Multiple Sensors

With the implementation of multiple ultrasonic sensors, crosstalk (one sensor hearing the other sensor's echo) is a potential hazard. This was solved by:

- Ambiguating the sensors sequentially, with minimal time lag in between measurements, so that there is no coupling between measurements.

Challenge 3: Integration in Real Time with Other Subsystems

Ensuring sensor data was processed in a timely manner to influence the movement and feedback systems of the robot in real time require careful planning. The solution was to update sensor data and dependent subsystems (LED, buzzer, motors) within the main control loop, allowing timely and coordinated responses.

Summary

The ultrasonic sensor subsystem forms the core of the robot's safety and environmental perception capabilities. Through the provision of accurate, real-time distance measurements in multiple directions, it enables the robot to avoid obstacles in safety, alert users, and operate autonomously with security. Its rugged, modular design renders it fail-safe in application and simple to integrate with the rest of the system, rendering it an essential element of the success of the project. [15] [16]

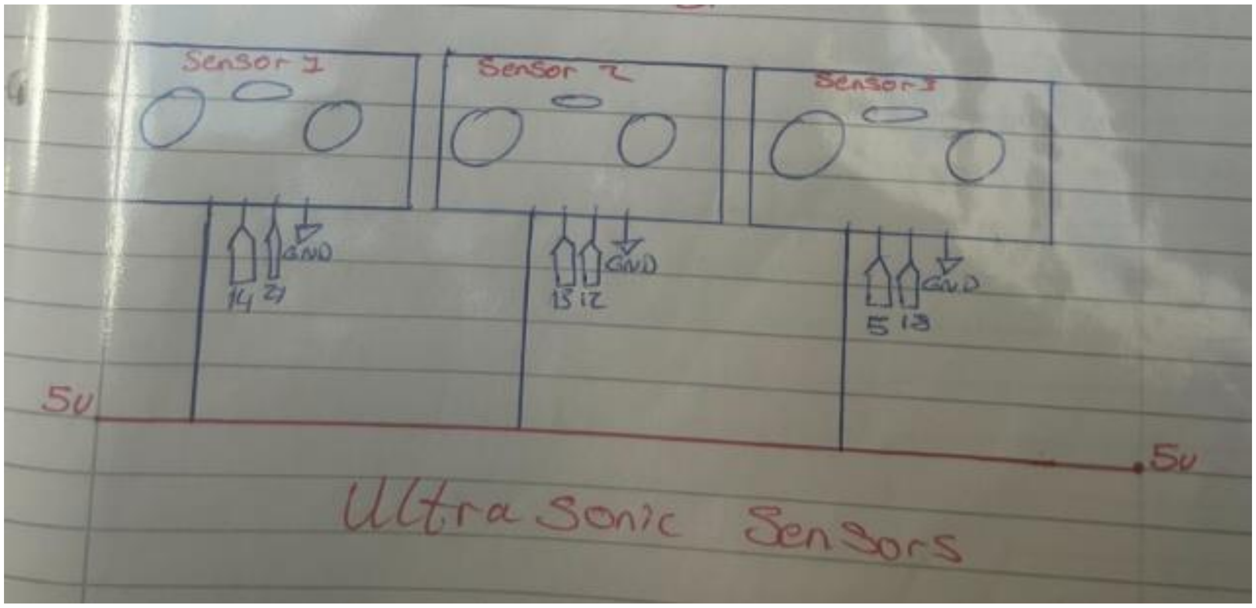


Figure 10-1 Ultrasonic Diagram

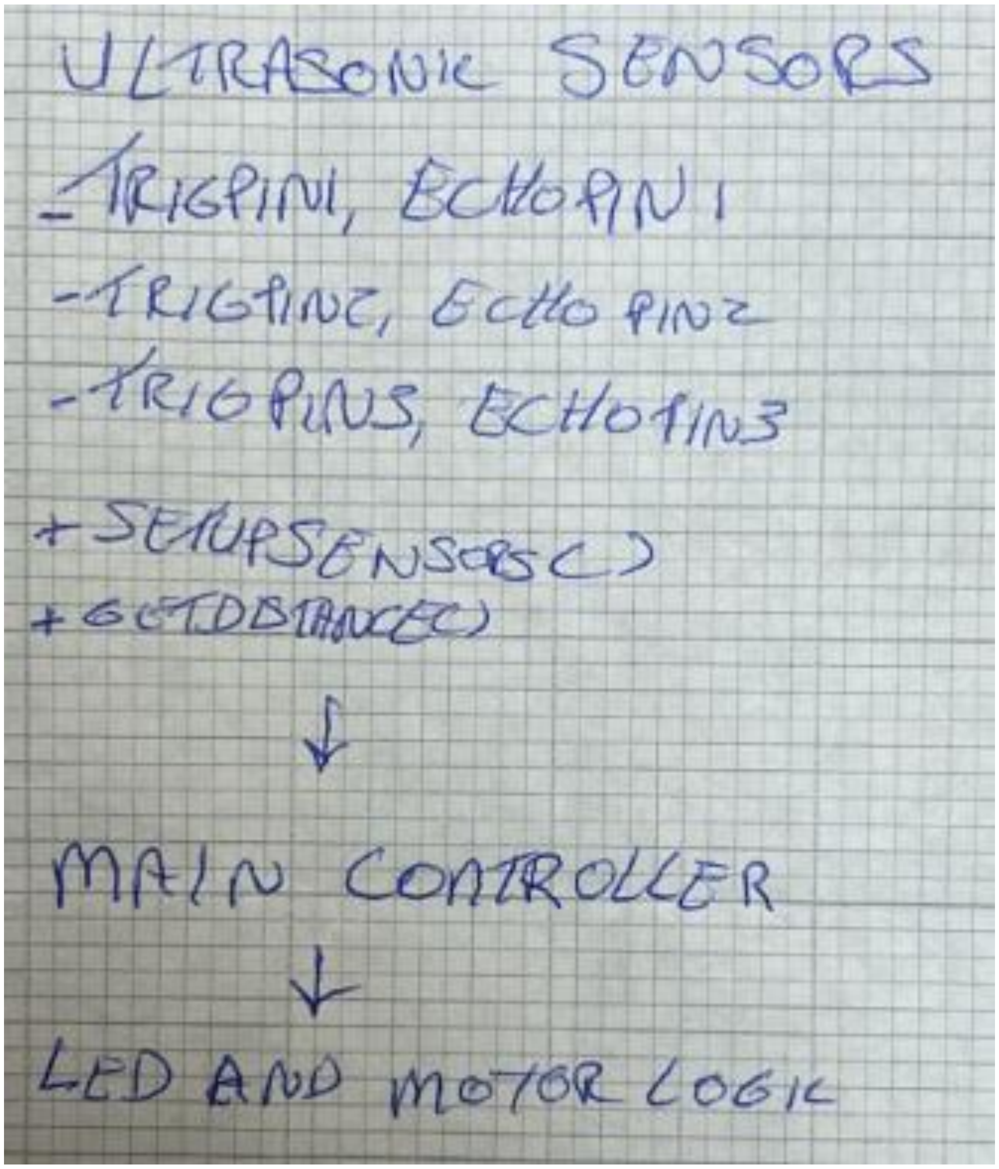


Figure 10-2 Ultrasonic Design Diagram

11 Motor

Purpose and Function in the System:

The motor subsystem enables the robot's physical mobility. It facilitates the movement of the robot within its environment in terms of going forward, moving backward, and turning left and right. The subsystem is the heart of the robot's ability to perform tasks, avoid collisions, and respond to user commands, and as such is an essential component of the system. [12]

Principle of Operation:

The car utilizes two DC motors, one attached to each motor driver. The motor driver acts as an interface between the low-power control signals from the ESP32 microcontroller and the more powerful requirements of the motors. With independent control over the direction and speed of each motor, the robot can achieve a range of motions:

- Forward: Both motors rotate in the same direction.
- Backward: Both motors rotate in the opposite direction.
- Left Turn: A single motor operates while the other motor remains still or operates in the opposite direction.
- Right Turn: The opposite of the left turn.
- Stop: Both motors remain still.

Integration with Other Subsystems:

- Obstacle Avoidance: After the ultrasonic sensors have detected an obstacle at a critical distance, the control logic can automatically reverse direction or stop the motors to prevent a collision.

- User Commands: The user can send movement commands to the robot in real time using the web interface. The ESP32 receives these commands and translates them into motor actions.
- Feedback Systems: LED and buzzer subsystems can also be triggered according to motor actions (e.g., warning the user when the robot is about to move or has stopped due to an obstacle).

Design Considerations:

- Pin Selection and Wiring: The control pins of the motor driver were chosen carefully to be compatible with the output capabilities of the ESP32. Motor control was implemented using only those pins capable of providing digital output.
- Power Management: Motors require far more current than can be supplied by the ESP32 alone. The motor driver is powered separately, and a common ground between the ESP32 and the motor driver is maintained for its proper operation.
- Responsiveness: Motor control code is designed to be responsive to both user commands and sensor input so that the robot can stop or change direction in a timely fashion to avoid collisions.
- Safety: The system is designed such that the motors are disabled by default upon startup, to avoid accidental movement on power-up of the robot.

Challenges and Solutions

Challenge 1: Pin Compatibility and Hardware Integration

Some of the ESP32 pins are input-only and cannot be used to drive the motor driver. This was resolved by carefully selecting only output-capable pins to drive the motor and intensively testing the hardware connections.

Challenge 2: Power Supply and Noise

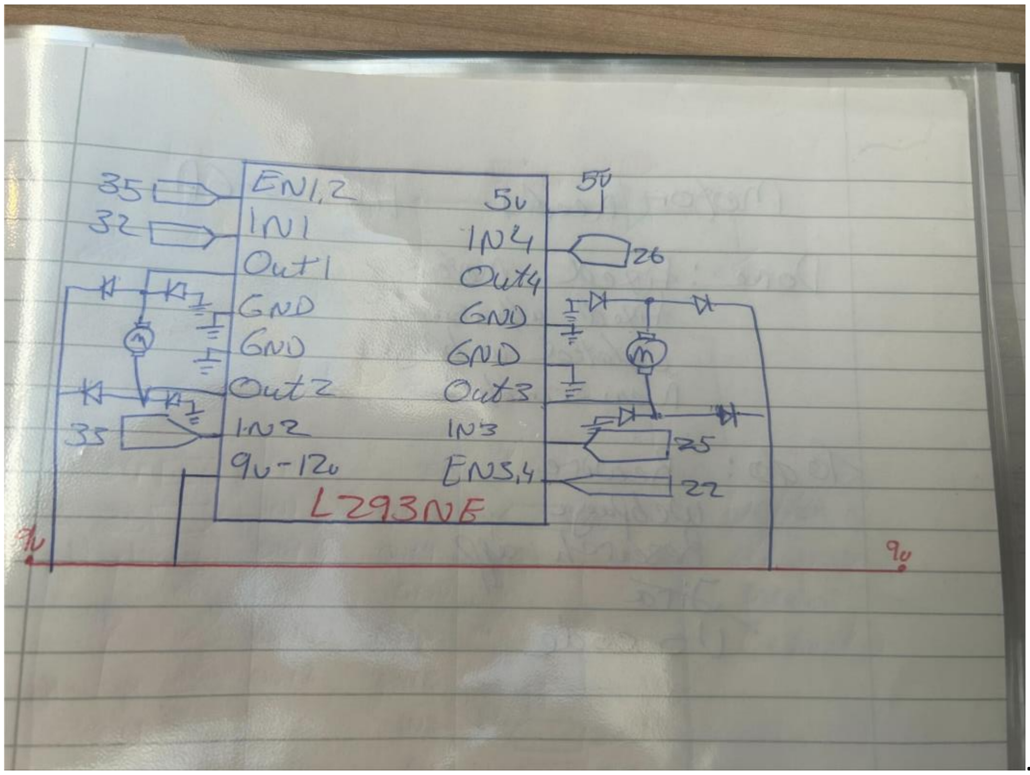
Motors can generate electrical noise and require more current than the microcontroller can provide. As a remedy, the motor driver and motors are supplied from a separate supply, and decoupling capacitors are used to reduce noise.

Challenge 3: Real-Time Control and Safety

Having the robot able to stop immediately when it ran into an obstacle or when it was sent a stop command was critical for safety. The control logic was programmed to prioritize stop commands and sensor-initiated stops for rapid response.

Summary

The motor subsystem delivers the propulsion for the mobility and independence of the robot. Its design allows for reliable, responsive, and safe movement, whether it is under user control or under the control of the robot's own logic. By operating in close coordination with the sensor and feedback subsystems, the motor subsystem enables the robot to navigate through complex spaces, avoid obstacles, and perform tasks effectively. Its rugged and modular design also facilitates easy maintenance and future upgrading, making it a key component of the project’s success. [15]



.

Figure 11-1 Motor Driver Diagram

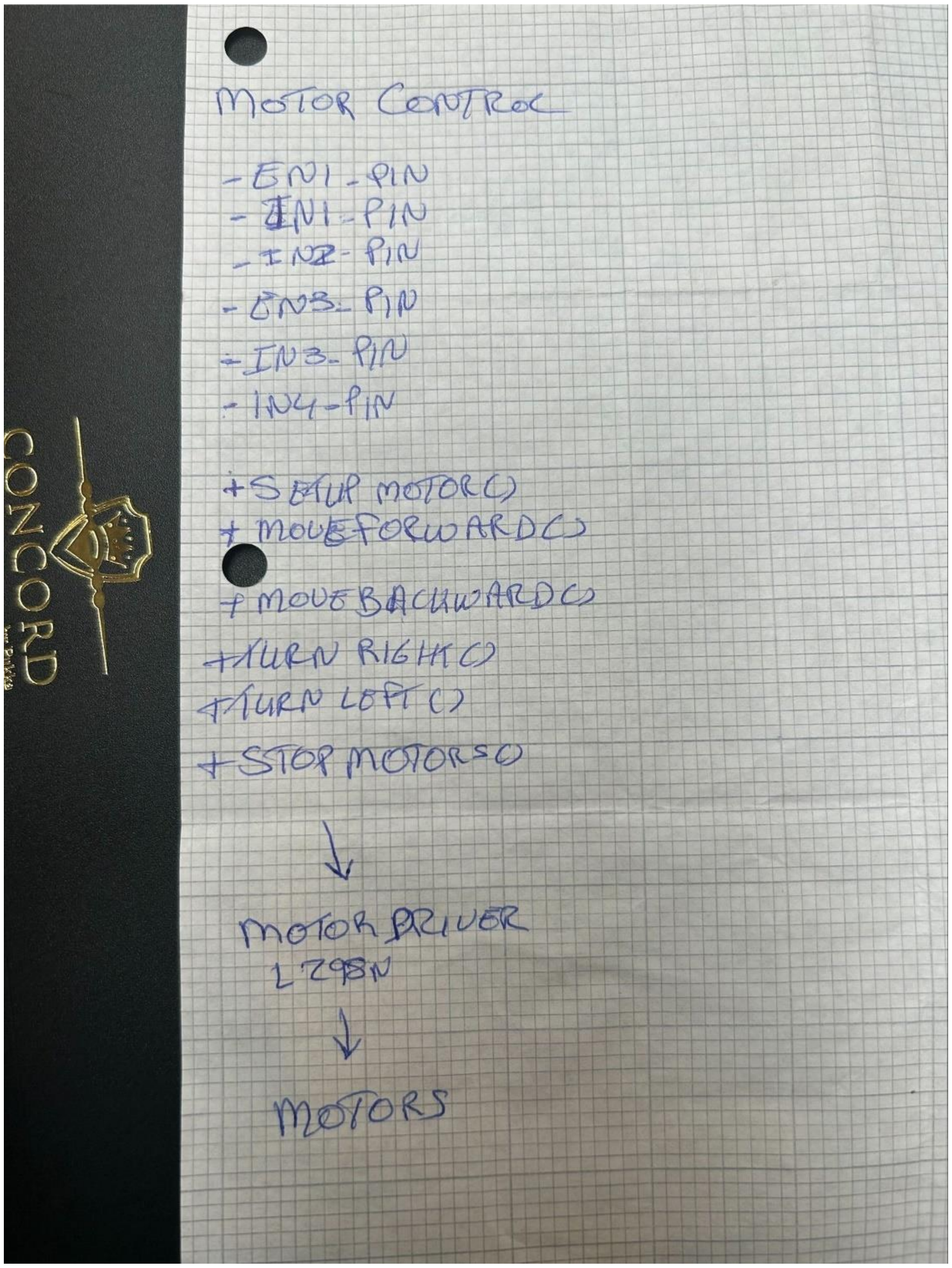


Figure 11-1 Motor Design Diagram

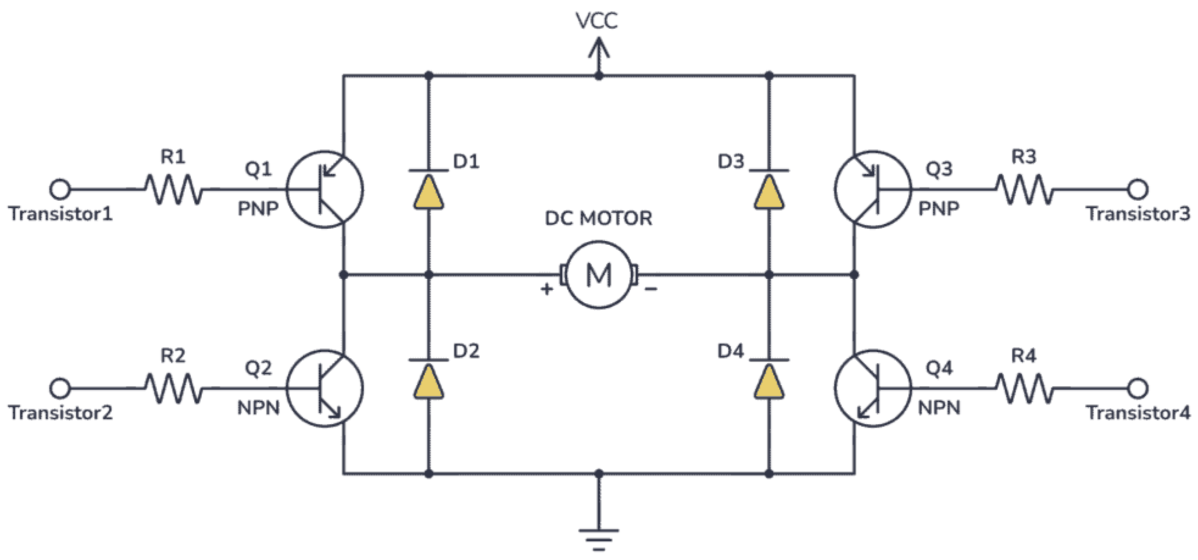


Figure 11-3 H-Gate Diagram

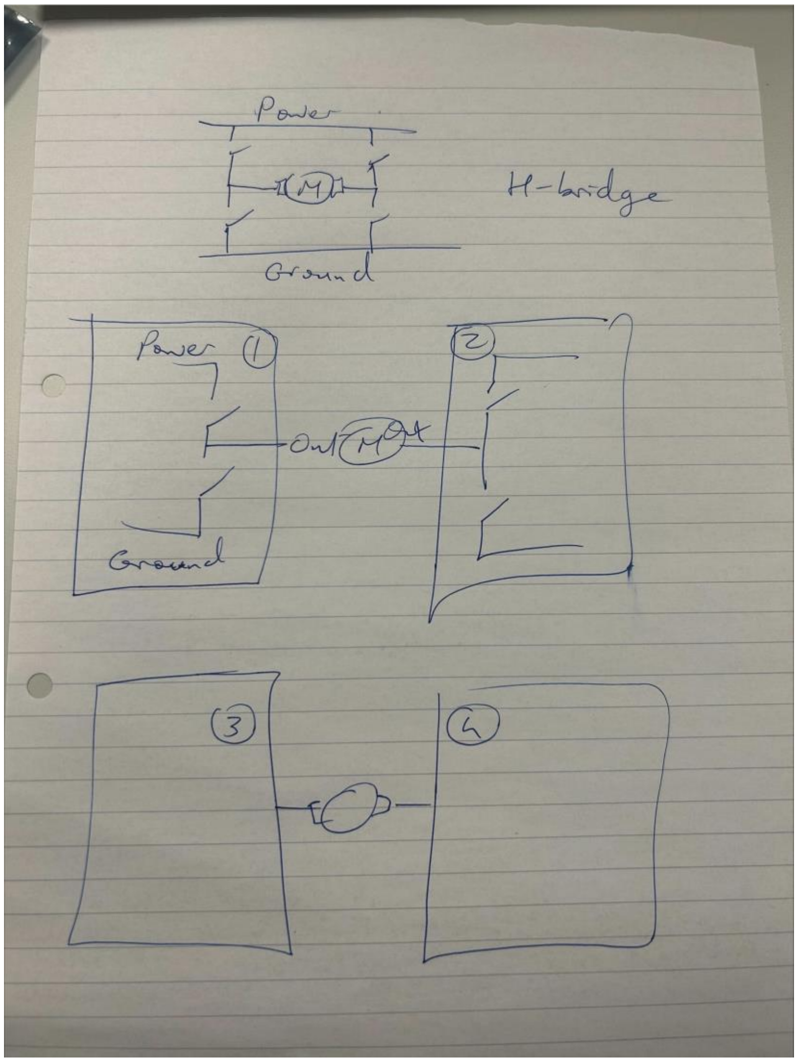


Figure 11-4 H-Gate Diagram

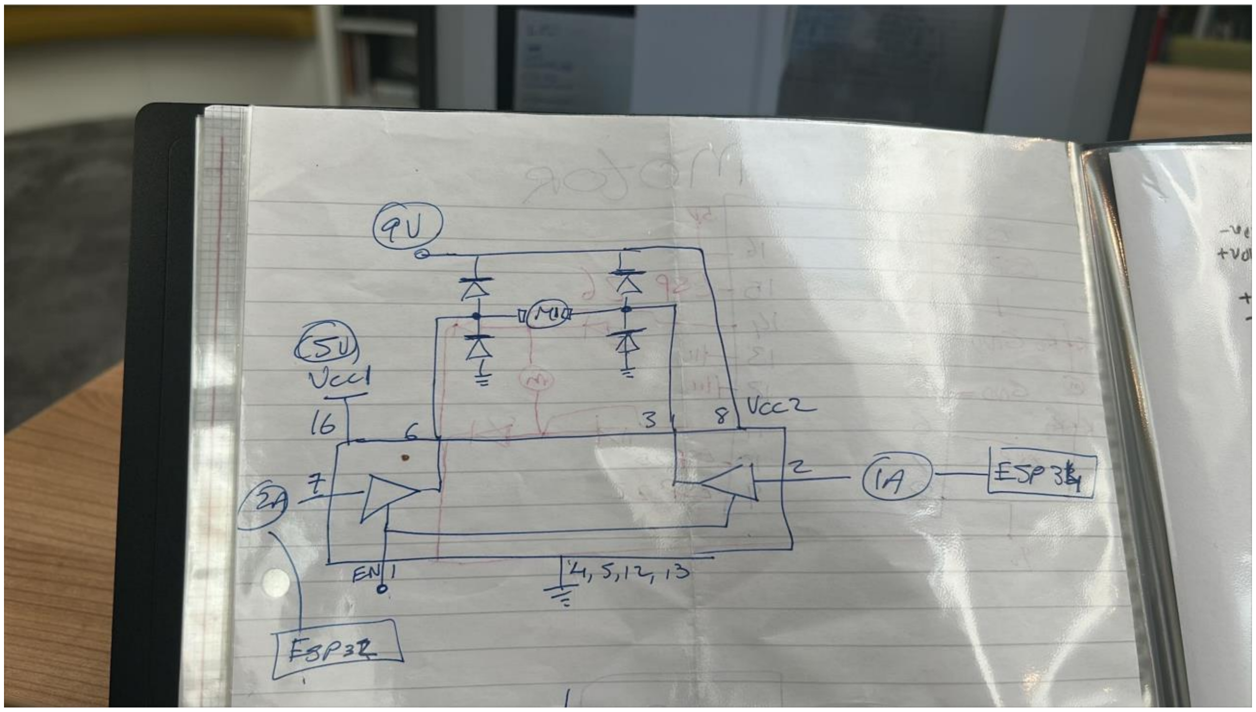


Figure 11-5 H-Gate Diagram

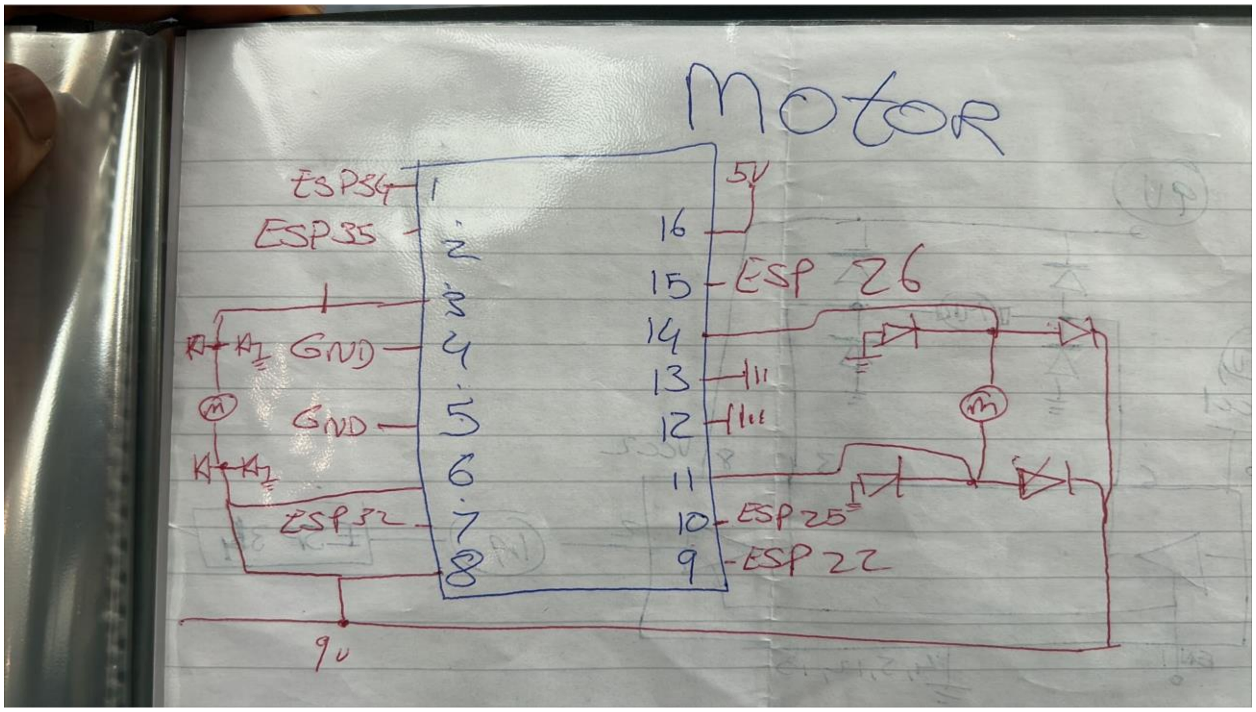


Figure 11-6 Motor Diagram

12 Webpage

Early Stage:

The earliest version of the interface was very basic, as shown above in the first picture. The ESP32 would attempt to connect to Wi-Fi and the serial monitor would show a basic status message. This was critical for confirming internet connectivity before additional features could be added. [14]

Basic Control and Feedback:

Then, I had the ability to craft a basic web page, where I just had one page, which allowed the user to turn on or off the on-board LED using clickable links. Also, the web page was able to display the status of the undercar sensors whether any sensors were triggered at that moment. This is where I got to create remote control and feedback as a basic user experience.

Minimal set of Controls and Sensor Data:

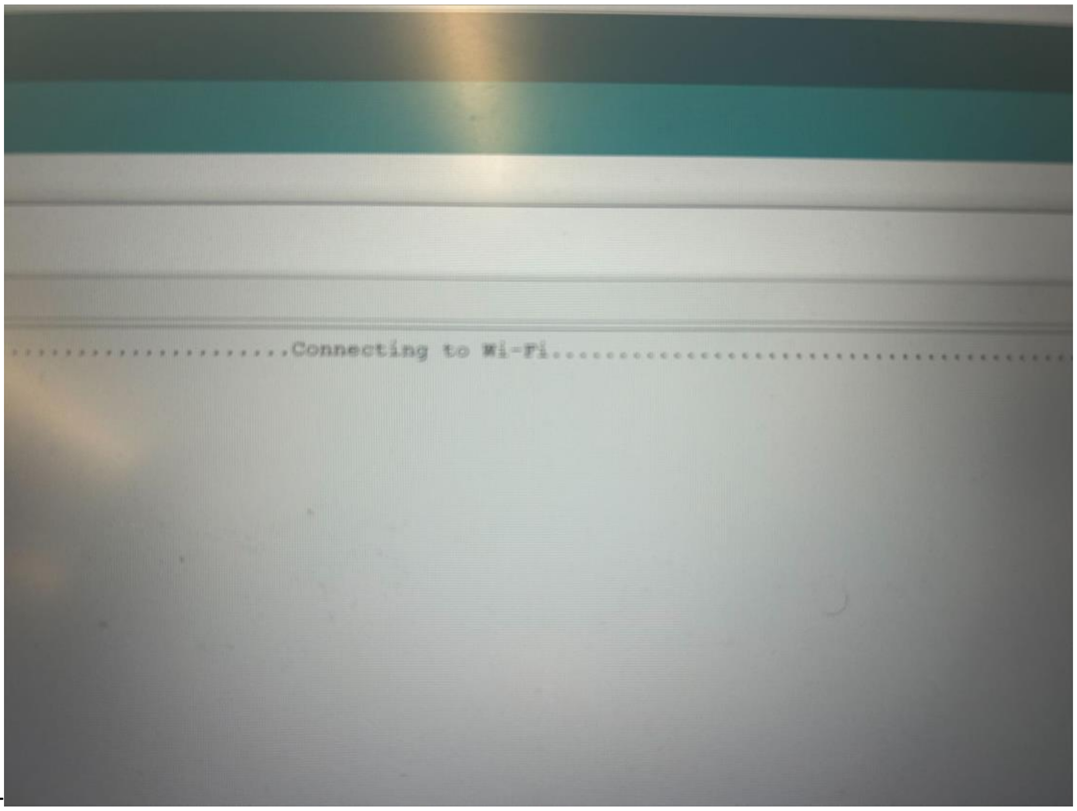
As I progressed, I was able to provide directional controls that moved the motors of the car (Go, Left, Right, Back, Stop) along with distance from the ultrasonic sensors that refreshed in real time to the page. This added up to more interactive remote interaction, as the user could remotely control the movement and use the page to collect and discover environmental data.

Final / more polished interface:

Finally, I was able to create a neat, organized, and user-friendly layout and design for the web interface. The final interface had organized sections for motor control, LED mode, and buzzer mode. Each mode has coloured buttons that are easy to understand and intuitively organized, and the sensors refresh in real-time. Overall, this final design makes the interface more visually appealing, clearly organized or labelled, and provides a level of professionalism that would be suitable for a demo and future development.

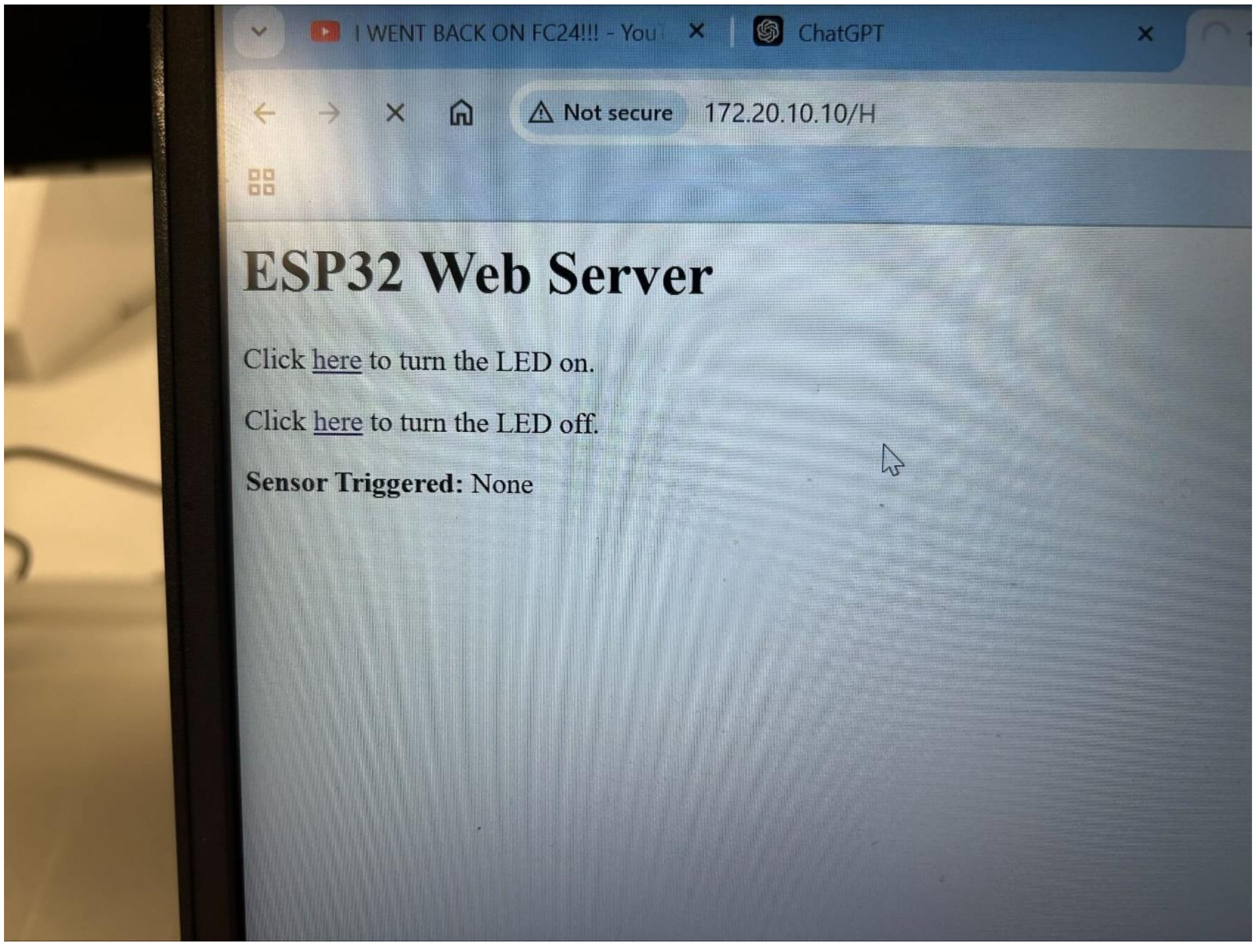
Summary:

Overall, the web interface evolved from a simple connectivity check to a comprehensive control and monitoring dashboard. Each stage of development added new features and improved the user experience, culminating in a robust and visually appealing interface that enables full remote operation of the car. [15] [2]

---

```
....................Connecting to Wi-Fi.....................................
```

Figure 12-1 Connecting to Wi-Fi



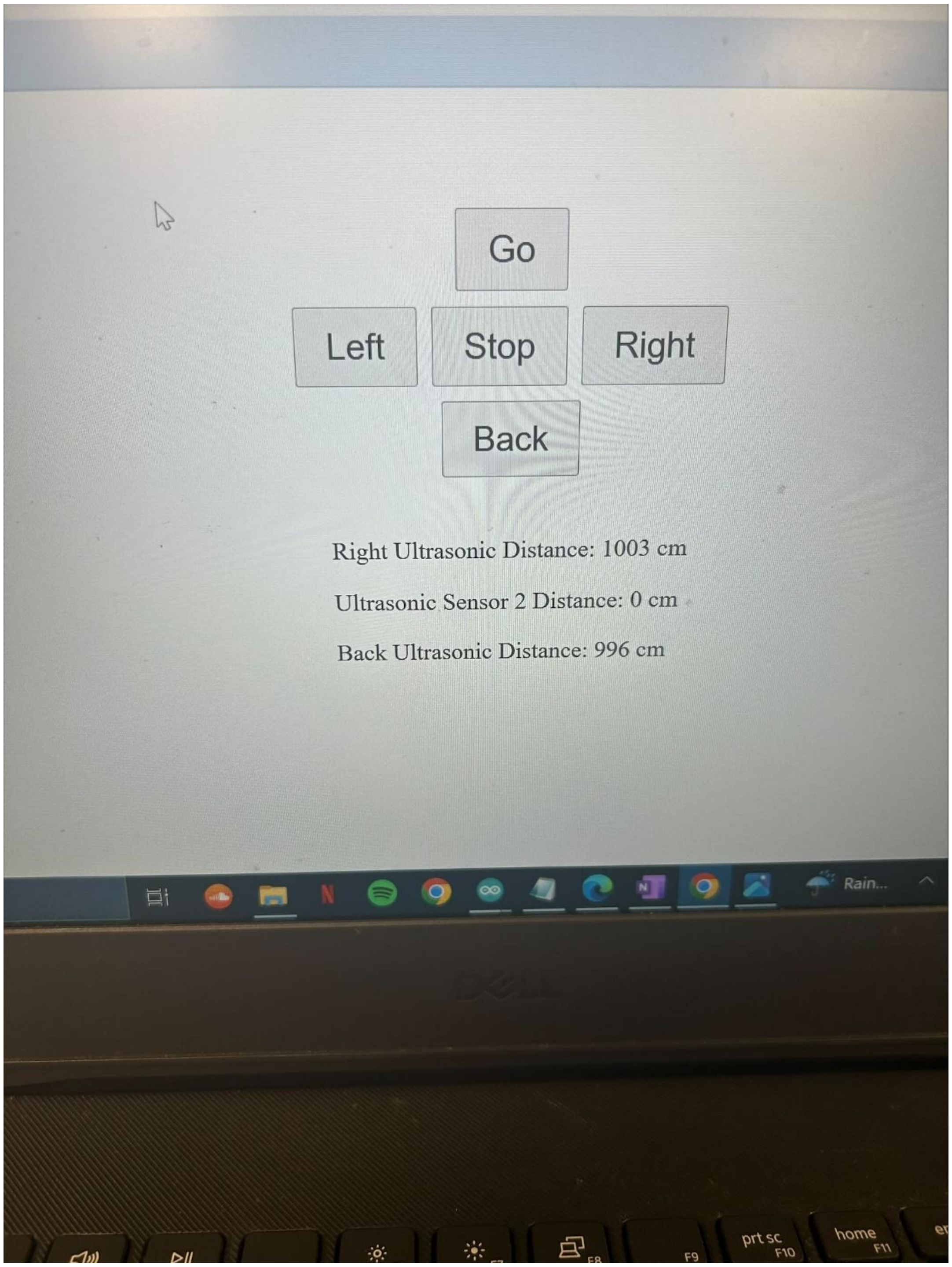
ESP32 Web Server

Click here to turn the LED on.

Click here to turn the LED off.

Sensor Triggered: None

Figure 12-2 LED Control



Right Ultrasonic Distance: 1003 cm

Ultrasonic Sensor 2 Distance: 0 cm

Back Ultrasonic Distance: 996 cm

Figure 12-3 Webpage Motor Control

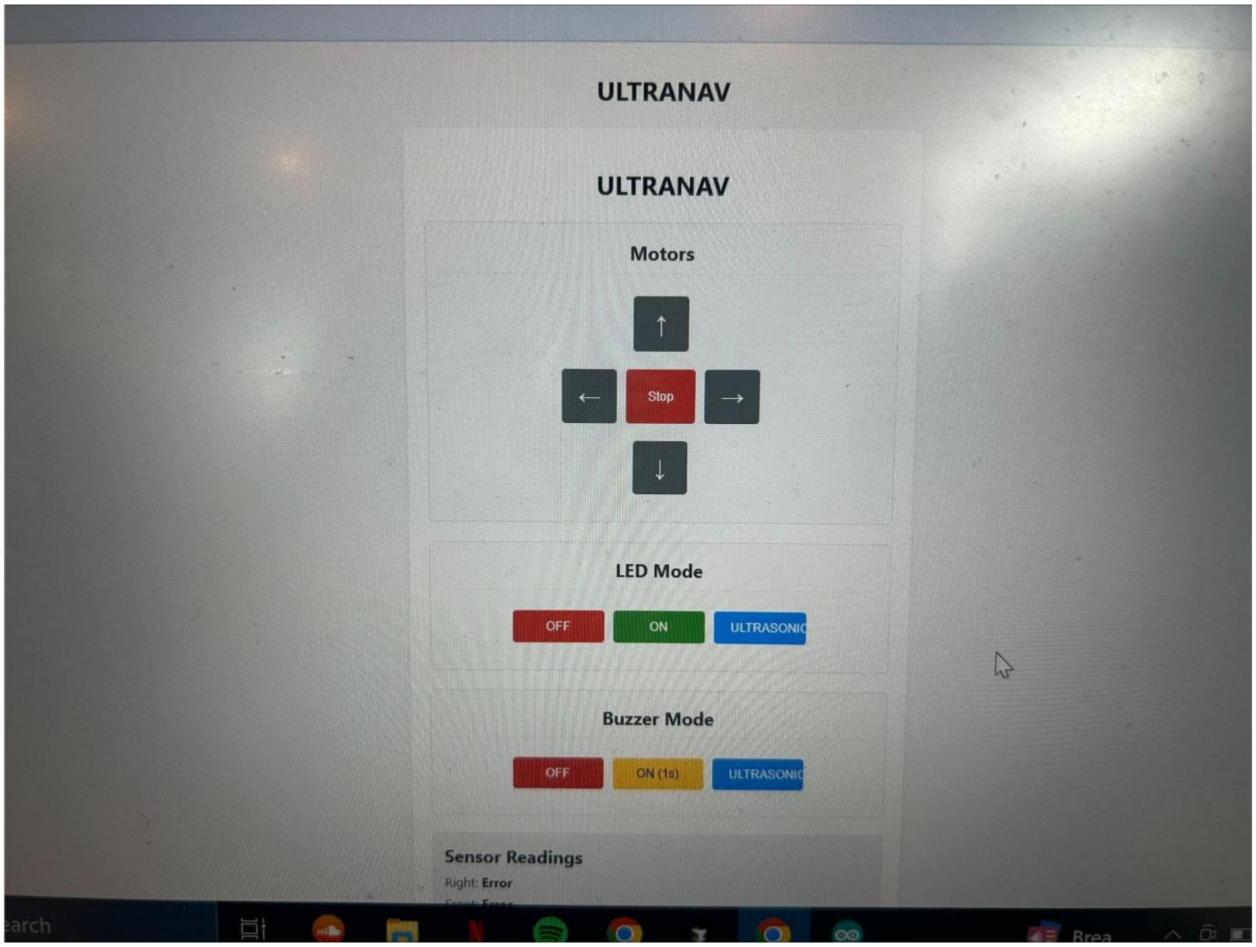


Figure 12-4 Completed Webpage

13 Mobile App

I created a mobile app interface for my robot project to allow users to control and monitor the robot directly from their mobile phones. The design and functionality of the mobile app evolved in order to best support enhanced usability, accessibility and ultimately to accommodate real time feedback.

Initial Design:

The initial design of the mobile app narrowed its focus to critical functionality and clear status. The interface displayed the robot's connect status, allowed users to control the robot LED mode, and provided a straightforward gamepad-like layout for movement commands (forward, backward, left, right and stop). The user's distance sensor measurements to the right, front and back were displayed in real time, providing immediate feedback about the robot's visually perceptive environment. The use of colour coding (e.g. red for "off" or "disconnected") created a visual cue that allowed users to quickly interpret the interface.

Feature Expansion:

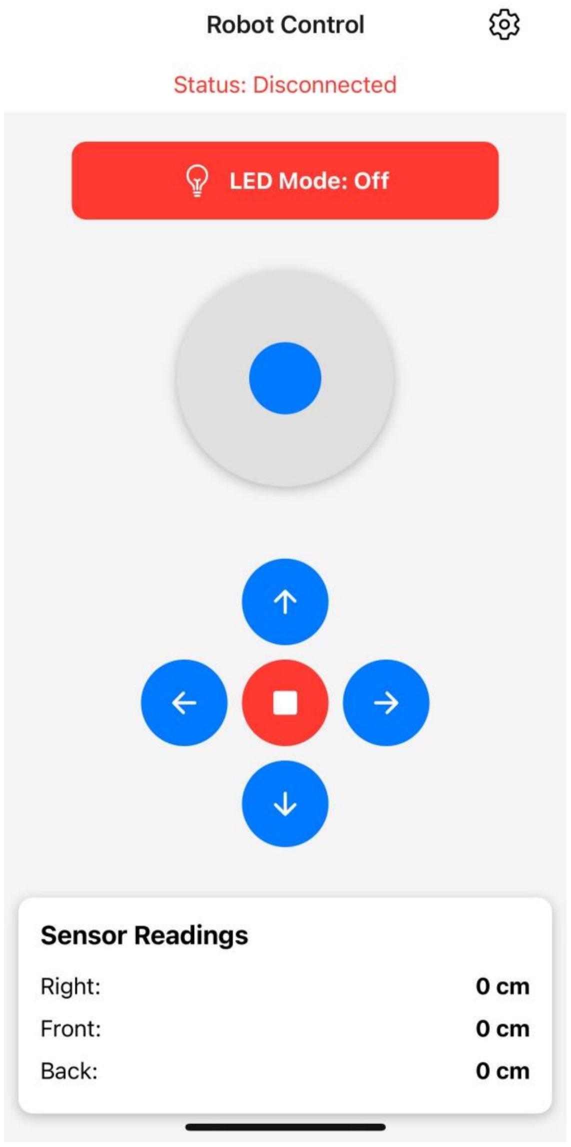
Over time and during the continued development of the mobile app additional features were added to allow users more control over the robot's subsystem. The interface introduced dedicated sections to specifically address LED and buzzer modes. The buttons were large and colour-coded to denote different settings on the robot's LED (OFF, ON, ULTRASONIC) and buzzer (OFF, ON (1s), ULTRASONIC). Enabling easy and quick changes of the robot's feedback mode from a solid distance through the UI of the interface from a mobile phone permitted an additional layer of safety for the user and an interactive engagement with the technology.

User Experience and Design:

The app's design went through several versions during development, improving in usability and clarity for mobile-phone interaction. Controls were made sufficiently large and easy to touch, along with formatting sections clearly for the movement controls, LED controls, the buzzer, and the sensor feedback. The app was designed for responsiveness and to look visually attractive, following a modern design pattern that integrates nicely with the web interface to give a consistent user experience regardless of platform.

Summary:

The mobile app interface changed from a simple control panel to a fully integrated platform that supports an engaging, seamless experience as a user interface. It now allows complete remote operation and monitoring of the robot with real-time feedback and intuitive controls, all operated and monitored from a mobile phone. This increases the viability and attractiveness of the solution to the average person interested in making or just learning.



Sensor Readings

Right:	0 cm
Front:	0 cm
Back:	0 cm

Figure 13-1 Mobile App Expo-Go

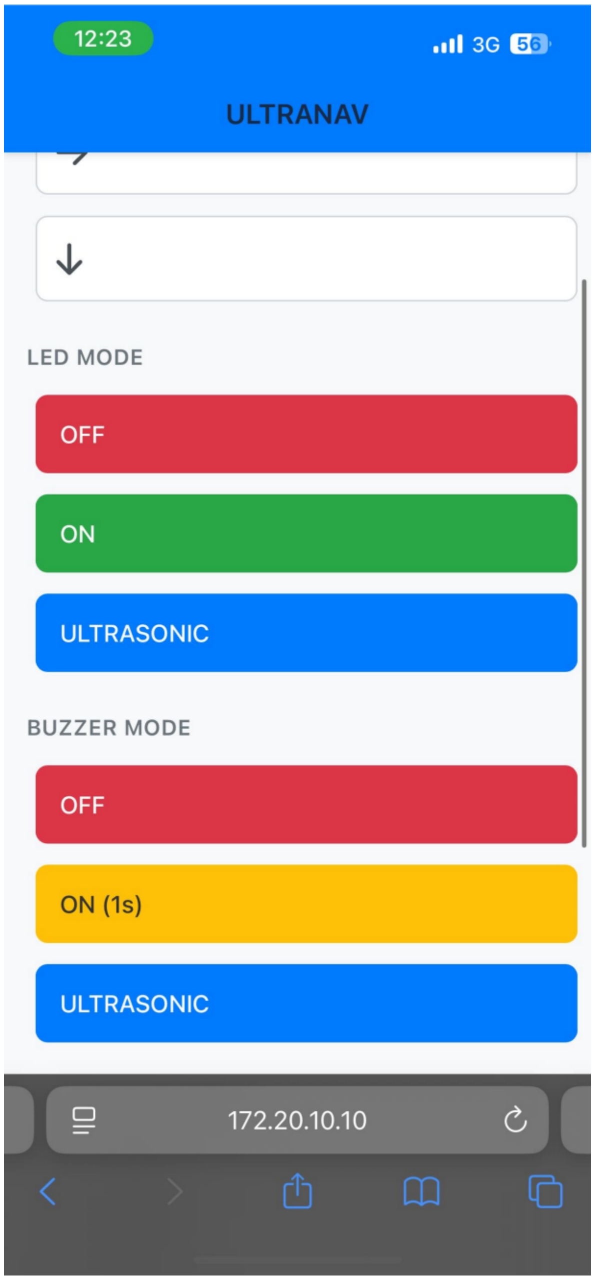


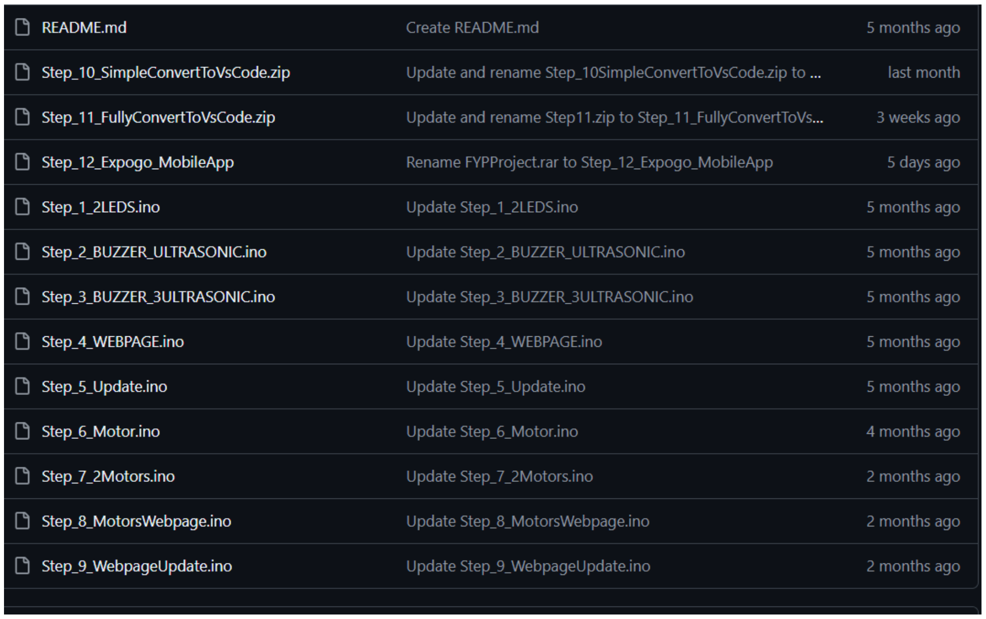
Figure 13-2 Completed Mobile App Local Host

14 GitHub

Throughout the course of my project, I continued to use GitHub as a version control system. I pushed and committed my code into a GitHub repository at each significant step of the project. This allowed me to keep a clean record of my work, divide my work into sequential steps, and revert back to an earlier version if needed.

Each major achievement—such as adding LED control, buzzer support, ultrasonic sensor support, web interface implementation, and mobile application implementation—was committed individually or saved as a whole file. This produced a backup of my work but also made tracing changes easier, debugging simpler, and demonstrating the project progress over time possible.

By using GitHub, I ensured that my project was well documented, collaboration-friendly, and protected against data loss, and also showcased my development process in a professional manner.



README.md	Create README.md	5 months ago
Step_10_SimpleConvertToVsCode.zip	Update and rename Step_10SimpleConvertToVsCode.zip to ...	last month
Step_11_FullyConvertToVsCode.zip	Update and rename Step11.zip to Step_11_FullyConvertToVs...	3 weeks ago
Step_12_Expogo_MobileApp	Rename FYPProject.rar to Step_12_Expogo_MobileApp	5 days ago
Step_1_2LEDS.ino	Update Step_1_2LEDS.ino	5 months ago
Step_2_BUZZER_ULTRASONIC.ino	Update Step_2_BUZZER_ULTRASONIC.ino	5 months ago
Step_3_BUZZER_3ULTRASONIC.ino	Update Step_3_BUZZER_3ULTRASONIC.ino	5 months ago
Step_4_WEBPAGE.ino	Update Step_4_WEBPAGE.ino	5 months ago
Step_5_Update.ino	Update Step_5_Update.ino	5 months ago
Step_6_Motor.ino	Update Step_6_Motor.ino	4 months ago
Step_7_2Motors.ino	Update Step_7_2Motors.ino	2 months ago
Step_8_MotorsWebpage.ino	Update Step_8_MotorsWebpage.ino	2 months ago
Step_9_WebpageUpdate.ino	Update Step_9_WebpageUpdate.ino	2 months ago

Figure 14-1 GitHub Workspace

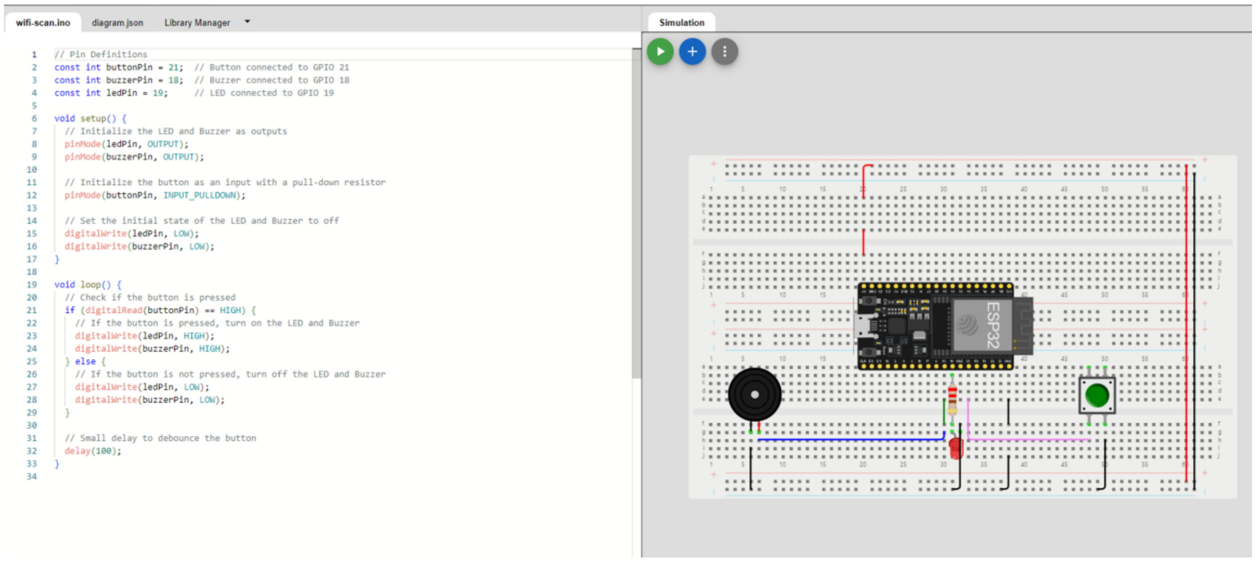
15 Wokwi

For the design and testing of my project, I used Wokwi, a web-based electronics simulator, to model and simulate my ESP32 circuits. Wokwi enabled me to design, visualize, and test my hardware configurations in a virtual environment before I made them physically real.

I was able to:

- Simulate the ESP32 and connect it to virtual components like LEDs, buzzers, and push buttons.
- Write and run Arduino code in the browser while visually observing how the ESP32 would interact with the connected components in real time.
- Efficiently debug and optimize my code and circuit design because Wokwi provided me with immediate feedback and ease of identifying and fixing my mistakes without damaging any hardware.
- Explore other pin mappings and component arrangements for the sake of the design itself before physically wiring it all on a breadboard.

Using Wokwi greatly accelerated my development cycle, reduced repetitive hardware assembly and gave me greater confidence that the final physical build would work as intended. I found it Especially useful for debugging input/output logic, i.e. making an LED and buzzer illuminate on button press, as seen in my simulation.



```cpp
// Pin Definitions
const int buttonPin = 21;  // Button connected to GPIO 21
const int buzzerPin = 18;  // Buzzer connected to GPIO 18
const int ledPin = 19;     // LED connected to GPIO 19

void setup() {
  // Initialize the LED and Buzzer as outputs
  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  // Initialize the button as an input with a pull-down resistor
  pinMode(buttonPin, INPUT_PULLDOWN);

  // Set the initial state of the LED and Buzzer to off
  digitalWrite(ledPin, LOW);
  digitalWrite(buzzerPin, LOW);
}

void loop() {
  // Check if the button is pressed
  if (digitalRead(buttonPin) == HIGH) {
    // If the button is pressed, turn on the LED and Buzzer
    digitalWrite(ledPin, HIGH);
    digitalWrite(buzzerPin, HIGH);
  } else {
    // If the button is not pressed, turn off the LED and Buzzer
    digitalWrite(ledPin, LOW);
    digitalWrite(buzzerPin, LOW);
  }

  // Small delay to debounce the button
  delay(100);
}
```

Figure 15-1 Wokwi Simulation

16 Tinkercad

To enable me to develop and test my ESP32-project, I also utilized Tinkercad, an online circuit simulation. Tinkercad enabled me to design and test my electronic circuits virtually before actual hardware component assembly.

With Tinkercad, I was able to:

- Test the ESP32 and other electronic devices such as LEDs, buzzers, buttons, and sensors in a simulated environment.
- Write and upload Arduino code to the simulated microcontroller, allowing me to observe how the code would control the hardware in real time.
- Debug and optimize my circuit design efficiently, as Tinkercad provided immediate feedback and made it easy to identify wiring or logic errors without risking damage to actual hardware.
- Experiment with different circuit configurations and pin layouts to identify the most effective and long-lasting configuration prior to developing the actual prototype.

Working with Tinkercad simplified my process, minimized the risk of hardware errors, and made me sure that my circuit and code would work correctly when implemented in real life. It proved particularly useful for trying out interactive functionality, like utilizing a button to drive an LED and buzzer, and for seeing how the entire design would look.

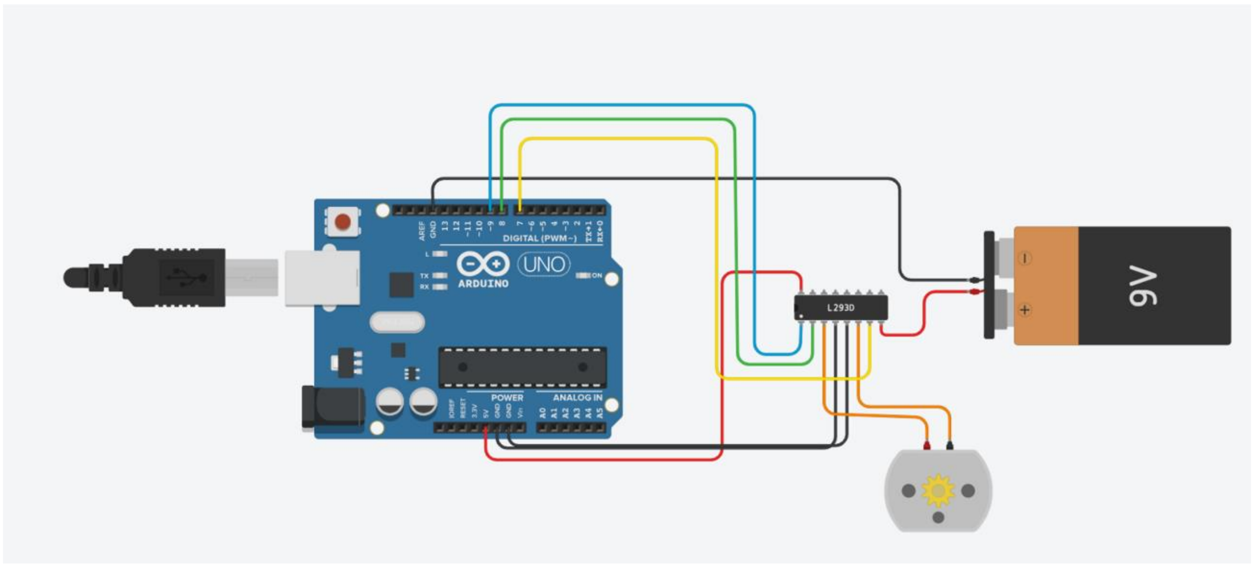


Figure 16-1 Tinkercad Simulation

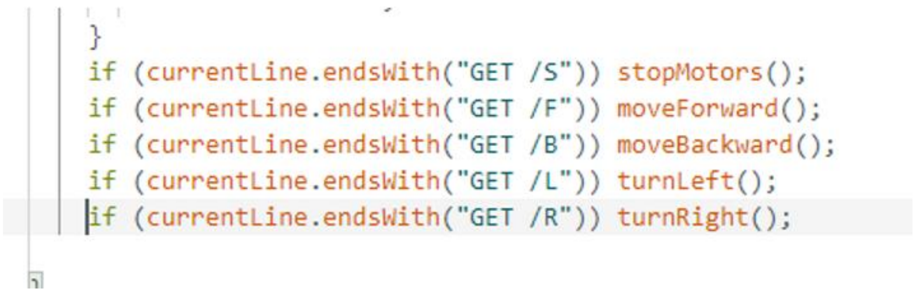
17 Arduino

Before integrating the mobile app and switching to more advanced development environments, I used the Arduino IDE as my primary platform for coding and testing the ESP32. The Arduino IDE provided a simple and intuitive interface for coding, compiling, and uploading code to the microcontroller.

Using the Arduino IDE, I was able to:

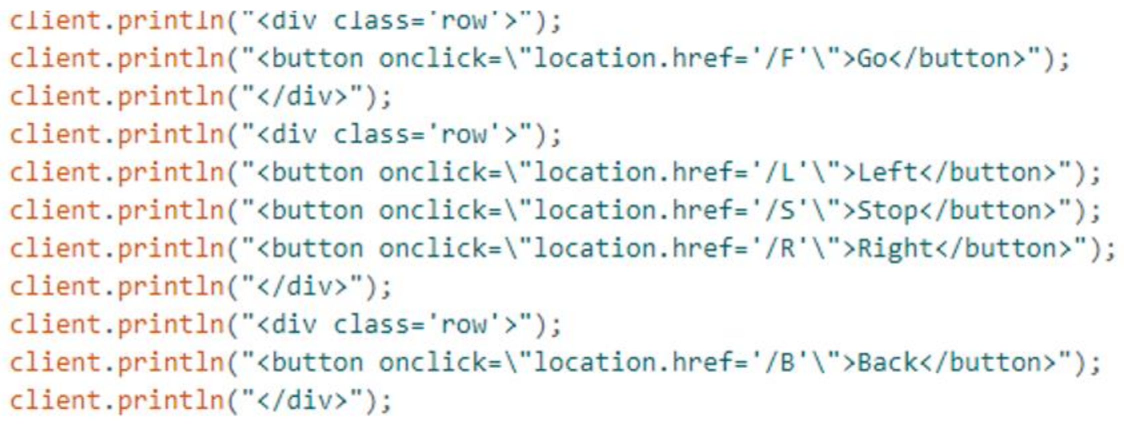
- Quickly prototype and test code for standalone devices such as LEDs, buzzers, ultrasonic sensors, and motors.
- Have access to a huge library of libraries and examples specific to Arduino-compatible boards, which made development and debugging easier.
- Employ serial output monitoring through the integrated Serial Monitor, which made it easy to debug sensor outputs, control logic, and communication with the ESP32.
- Just work with sketches and maintain project files organized as I added to the system's functionality incrementally.

The Arduino IDE proved to be especially helpful in the beginning of the project, allowing me to focus on making the hardware and basic software run properly. Its simplicity and wide community support made it the ideal starting point before transitioning to more advanced tools and facilities, such as the mobile application and web-based interfaces.



```
}
if (currentLine.endsWith("GET /S")) stopMotors();
if (currentLine.endsWith("GET /F")) moveForward();
if (currentLine.endsWith("GET /B")) moveBackward();
if (currentLine.endsWith("GET /L")) turnLeft();
if (currentLine.endsWith("GET /R")) turnRight();
```

Figure 17-1 Motor Code Using Arduino IDE



```
client.println("<div class='row'>");
client.println("<button onclick=\"location.href='/F'\">Go</button>");
client.println("</div>");
client.println("<div class='row'>");
client.println("<button onclick=\"location.href='/L'\">Left</button>");
client.println("<button onclick=\"location.href='/S'\">Stop</button>");
client.println("<button onclick=\"location.href='/R'\">Right</button>");
client.println("</div>");
client.println("<div class='row'>");
client.println("<button onclick=\"location.href='/B'\">Back</button>");
client.println("</div>");
```

Figure 17-2 Wi-Fi Motor Code Using Arduino IDE

18 Visual Studio Code

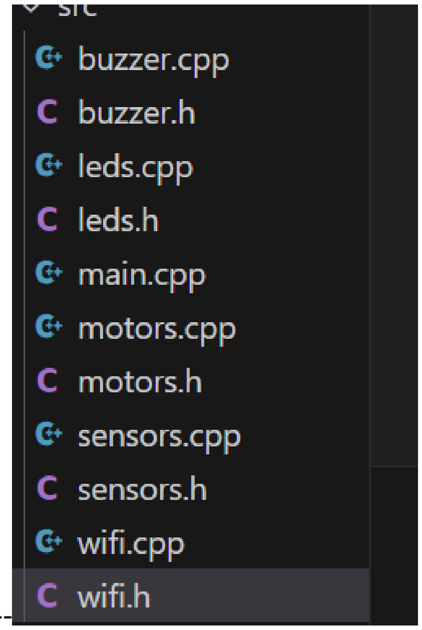
As my project grew in complexity, I transitioned from the Arduino IDE to Visual Studio Code (VS Code) combined with the Platform IO extension. This setup provided a professional, scalable, and highly efficient environment for managing and developing my ESP32-based project. [13]

With VS Code and Platform IO, I was able to:

- Develop, test, and debug each subsystem independently.
- Easily integrate new features such as the web server, mobile app interface, and advanced sensor logic.
- Maintain a clean and organized codebase that is easy to understand and modify.
- Leverage GitHub integration for seamless version control and collaboration.

Summary:

Switching to VS Code and Platform IO transformed my workflow, enabling me to manage a complex, multi-file project with ease and efficiency. The advanced tools, modular structure, and professional features provided by this environment made it far superior to the Arduino IDE for large-scale or long-term projects. [13] [14] [15]



src
buzzer.cpp
buzzer.h
leds.cpp
leds.h
main.cpp
motors.cpp
motors.h
sensors.cpp
sensors.h
wifi.cpp
wifi.h

Figure 18-1 Visual Studio Code Classes

Figure 18-2 Platform IO

19 Conclusion

This project involved creating, developing, and deploying a Wi-Fi based robot system utilizing the ESP32 microcontroller. The robot has various subsystems like motor control, ultrasonic range finding, LED and buzzer feedback, and web and mobile user interfaces to ensure safe, interactive, and remote-controlled operation.

Development Process:

I began the project with prototyping and debugging individual units with the Arduino IDE, providing me with a minimalist environment for initial experimentation. With the rise of system complexity, I began to use Visual Studio Code with Platform IO, through which I gained advanced project management, modular coding, and production-level development environments. Along the way, I used simulation platforms such as Wokwi and Tinkercad to try out circuits and code without damaging hardware and at a faster pace.

Subsystems:

- Motors: Delivered precise motion and navigation, powered by a motor driver and augmented with sensor feedback for obstacle evasion.
- Ultrasonic Sensors: Delivered real-time range data in multiple directions, which was the basis for collision avoidance and feedback systems.
- LED and Buzzer: Delivered immediate visual and auditory feedback, with dynamic response based on sensor data and user commands.
- Wi-Fi/Web Interface: Enabled users to control and monitor the robot remotely via a browser with real-time feedback and easy controls.
- Mobile App: Provided additional convenience and accessibility, with full control and monitoring capabilities from a smartphone with a clean, touch-based interface.

Software Design:

The codebase was organized into module files by subsystem to improve scalability and maintainability. Platform IO's build and dependency management ensured safe compilation and deployment. GitHub acted as a version control mechanism, with the developer committing at regular intervals at each stage of development to keep a record of activity and allow team collaboration.

Challenges and Solutions:

Significant challenges were to achieve real-time responsiveness, handle power and pin constraints, coordinate sensor data with feedback systems, and create an intuitive user interface. These were addressed by proper selection of hardware, modular design of software, use of simulation tools, and iterative testing.

Results:

The resulting system is robust, responsive, and convenient to use. It is observable and controllable remotely, provides good feedback, and is expandable for convenience of use. Professional tools and best practices have been employed throughout the project to produce a quality, maintainable solution for demonstration, further development, or teaching purposes.

20 References

[1] **DBR**. “ESP32 Bluetooth Controlled Care,” [Online]. Available: https://www.instructables.com/ESP32-Bluetooth-Controlled-Car/ .Accessed: 02 10 2024

[2] CUBIT. " DIY Bluetooth-Controlled Car with ESP32: A Step-by-Step Guide || CUBIT," [Online] Available: https://www.youtube.com/watch?v=pT1PbWQu80U. Accessed: 02 10 2024

[3] **hash include electronics**. " PS4 Controller Mecanum wheels car | ESP32 | DIY," [Online] Available: https://www.youtube.com/watch?v=HCLo7tr6X9I. Accessed: 02 10 2024

[4] **hash include electronics**. " Surveillance Car using ESP32 Cam module | ESP32 Camera wi-fi car," [Online] Available: https://www.youtube.com/watch?v=HfQ7lhhgDOk. Accessed: 09 10 2024

[5] **NEON AIRSHIP**. " M5stack Timer Camera X ESP32 PSRAM OV3660 Review," [Online] Available: https://www.youtube.com/watch?v=qT9to3wNn3g&t=151s. Accessed: 04 12 2024

[6] **Arnov Sharma**. " How to use L293D motor driver with ESP32," [Online] Available: https://www.youtube.com/watch?v=aDBF9Yj04MU&t=239s. Accessed: 04 12 2024

[7[**"Datasheet 1.2 V NiMH AA Rechargeable Battery**," [Online] Available: https://docs.rs-online.com/a8af/0900766b8164f56d.pdf. Accessed: 04 12 2024

[8] **"ESP32-DevKitC V4,"** [Online] Available: https://docs.ESPressif.com/projects/ESP-dev-kits/en/latest/ESP32/ESP32-devkitc/user_guide.html. Accessed: 04 12 2024

[9] **"L293NE Datasheet,"** [Online] Available: https://www.alldatasheet.com/datasheet-pdf/view/26888/TI/L293NE.html. Accessed: 04 12 2024

[10] **Debashis Das**. " Control DC Motor with Arduino and L293D Motor Driver IC," [Online] https://circuitdigest.com/microcontroller-projects/interface-l293d-motor-driver-with-arduino. Accessed: 04 12 2024

[11] **winson-DIY**. " DC motors with L298N module - ESP32 Bluetooth SPP control," [Online] Available: https://www.youtube.com/watch?v=F3gByuOwLIU&t=3s. Accessed: 04 12 2024

[12] **"SN6507**," [Online] Available: https://www.ti.com/product/SN6507. Accessed: 04 12 2024

[13] **Tomasz Tarnowski**. " Getting Started with ESP32 - Step-By-Step Tutorial," [Online] Available: https://www.youtube.com/watch?v=tc3Qnf79Ny8&t=589s. Accessed: 13 03 2025

[14] **Tomasz Tarnowski**. " Connect ESP32 to WiFi - Step-By-Step Tutorial," [Online] Available: https://www.youtube.com/watch?v=aAG0bp0Q-y4&t=680s. Accessed: 13 03 2025

[15] **Tomasz Tarnowski."** Getting Started with ESP32-S2, PlatformIO, and Arduino Framework," [Online] Available: https://www.youtube.com/watch?v=fVcru1pq9TY. Accessed: 13 03 2025

[16] **VolosProjects."** M5Timer Camera X ESP32 Camera (Best ESP32 Camera)," [Online] Available: https://www.youtube.com/watch?v=d79t8UJH2gk. Accessed: 13 03 2025

[17] **electronic GURU."** one more ESP32 CAMERA by m5stack | Best video streaming ESP 32 so far | ESP32 CAM | m5cam module," [Online] Available: https://www.youtube.com/watch?v=Ruys-rNLTqw. Accessed: 13 03 2025