GENG4412 Engineering Research Project

Final Report
Integrating Autonomous Driving using Microcontrollers for Educational and Research Purposes

Semester 1, 2026

Shaf Kashif
23364696
School of Engineering, University of Western Australia

Supervisor: Thomas Bräunl
School of Engineering, University of Western Australia

Word Count: 7875

School of Engineering
University of Western Australia

Submitted: 20/05/2025

Project Summary

This project investigated the feasibility of implementing autonomous driving functionality using low-cost ESP32 microcontroller platforms for educational and research purposes. Modern autonomous systems typically rely on powerful and expensive embedded computing hardware, limiting accessibility for schools, universities, and small-scale robotics research. The project therefore aimed to explore whether meaningful autonomous vehicle behaviour could be achieved using simplified microcontroller-based architectures suitable for educational environments and beginner robotics users.

The project consisted of two primary areas of work: the development of an autonomous ride-on car platform and the validation of UWA's ESP32-based EyeBots used in the Embedded Systems unit (ELEC3020). The ride-on car component focused on integrating GPS localisation, IMU heading estimation, obstacle detection sensors, wireless telemetry, and actuator control into a single cohesive ESP32-based autonomous driving system. A browser-based webserver hosted directly on the ESP32 was also developed to allow users to remotely monitor the vehicle, visualise live GPS location data, and input autonomous waypoint destinations through an interactive map interface. The EyeBot component focused on validating and redeveloping camera-processing functions within the ROBIOS library to ensure compatibility with the newer ESP32-based EyeBot platform used in teaching laboratories.

The project adopted a hardware-first methodology involving iterative subsystem testing and real-world validation on physical hardware platforms. Testing of the ride-on car was conducted primarily at James Oval, University of Western Australia, where GPS navigation, IMU heading correction, obstacle detection, and wireless telemetry systems were evaluated. The EyeBot camera functions were validated through completion of embedded systems laboratory tasks including colour detection and object tracking.

Results demonstrated that the ESP32 platform was capable of performing meaningful autonomous vehicle functions despite significant hardware and processing limitations. The ride-on car successfully navigated toward user-defined GPS waypoints using continuous GPS and IMU feedback while providing live telemetry through the webserver interface. The modular hardware architecture also proved highly suitable for educational implementation due to its low cost, simple wiring design, and minimal soldering requirements.
Several limitations were identified during testing, particularly relating to the absence of steering angle feedback and the limited obstacle detection range of the Sharp PSD sensors. These limitations resulted in oscillatory steering behaviour and restricted obstacle avoidance capability during autonomous navigation. Future work should therefore focus on integrating more advanced sensing technologies such as LiDAR or depth-based vision systems to support improved obstacle avoidance and more sophisticated autonomous navigation algorithms including DistBug and Pure Pursuit.

Additionally, the updated EyeBot camera-processing functions operated reliably during large-scale classroom deployment within ELEC3020 laboratories

Overall, the project successfully demonstrated that low-cost ESP32 microcontrollers can provide an accessible and educationally valuable platform for autonomous driving and robotics research.

Acknowledgements

I would like to thank my supervisor, Professor Thomas Bräunl, whose support allowed me to contribute to such a beneficial and interesting project, his advice was invaluable. It was a privilege to complete my final year research project under his guidance as one of the leading professors in robotics at The University of Western Australia.

Introduction

Autonomous driving is one of the most transformative developments in modern engineering, driving progress in Industry 3.0 and 4.0 [1]. Its applications span from personal transportation to industrial automation at mine sites. However, most autonomous systems depend on powerful and expensive computing platforms such as industrial supercomputers or advanced embedded processors. This cost barrier limits accessibility for small-scale research, prototyping, and education. Schools and universities often lack the resources or expertise to use such systems effectively, making them impractical for teaching environments.

Since the late 2000s, microcontrollers -compact low-cost computers on a single chip- have become common in university and hobbyist projects. They are widely used for teaching basic automation and solving everyday problems, yet their potential for larger-scale applications, such as vehicle autonomy, remains underexplored.

This project aims to bridge that gap by investigating whether an ESP32-based architecture can perform autonomous vehicle functions traditionally handled by high-performance systems. The research question asks: to what extent can an ESP32 microcontroller deliver GPS-based navigation, sensor integration, and autonomous vehicle control in small-scale vehicles? The objectives are to: (1) design an integrated ESP32 control system for a ride-on car, (2) enable GPS-guided navigation via an ESP32 webserver, and (3) evaluate UWA's ESP32-based EyeBots.

The first two objectives focus on developing a simplified autonomous ride-on-car (seen in Figure 1) platform intended for education and research purposes. A key aim of the project is to keep the hardware architecture as accessible and reproducible as possible so that secondary school students and beginner robotics users can replicate the system [2] with minimal cost and technical experience. This includes the use of modular components, simple wiring methods, and minimal soldering to reduce assembly complexity while still demonstrating core autonomous driving concepts such as GPS navigation, wireless communication, and sensor integration.

Figure 1: Children's Ride-on-Car

The third objective evaluates UWA's ESP32-based EyeBots, which were developed as low-cost educational robots for embedded systems and robotics laboratories. Their inclusion in this project provides further insight into the suitability of ESP32 microcontrollers for robotics education and autonomous applications.

Ultimately, this project seeks to demonstrate how affordable microcontrollers can support autonomous driving research and education while reducing reliance on costly hardware.

Background

Embedded Systems EyeBots

The EyeBots are small autonomous robots used in UWA's Embedded Systems unit (ELEC3020) for the final laboratory tasks. They were originally based on a Raspberry Pi platform – EyeBot 8 - with the ROBIOS library but were recently migrated to a T-Display S3 ESP32 microcontroller-based platform (Eyebot-32) by honours student Parham Bahrami [3]. This upgrade aimed to reduce cost, power use, and complexity while aligning the platform with modern microcontroller-based teaching. Both Eyebot 8 and Eyebot-32 can be seen in Figure 2.

While Bahrami successfully transitioned the hardware and began porting key libraries, full testing and implementation of the labs remained incomplete. In particular, it was unclear whether the ESP32 libraries could fully replicate previous driving and camera functions, or whether new custom functions would be required. Supervisor Bräunl extended this work by requesting further assistance in implementing and validating the labs as part of the current project.

This project builds directly on Bahrami's work by testing the ESP32 EyeBot libraries, addressing compatibility gaps, and enabling the robots to complete lab tasks such as line following and object tracking. In doing so, it supports both the continuity of teaching and the broader investigation into the feasibility of microcontroller-based autonomous driving.

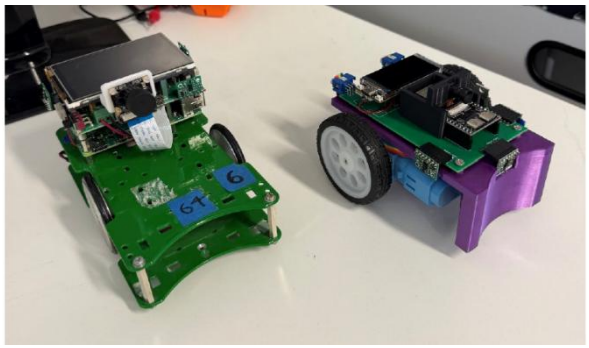


Figure 2: Eyebot 8 (left) & Eyebot-32 (right)

Autonomous Ride on Car

The Ride-on-car was originally modified by the UWA Robotics Lab team to be a Raspberry Pi based control system, the Raberry Pi based control box can be seen in Figure 3. This version was already used as high school STEM project with schools such as Perth Modern School, Scared Heart College, Sorrento, Australind Senior High School, Waroona District High school, Bunbury Senior High School and Sevenoaks Senior College.

Figure 3: The Raspberry Pi Control Box for The Ride-on- Car

However, to improve the research on this topic and to find a more cost-effective solution, it was decided a similar project was to be researched and designed using a more simplistic approach with more cost-effective parts such as using an ESP32 controller instead of a Raspberry Pi controller.

The Esp32 Ride-on car Project was initiated by an honour's student, Jiantong Guo, who concluded his work at the end of Semester 1, 2025. Guo's objective was to autonomise a commercially available children's ride-on car using a T-Display S3 ESP32 microcontroller. His approach involved integrating GPS for waypoint navigation, passive infrared PSD sensors for obstacle detection and safety, and a camera module for basic computer vision tasks. Each of these features was successfully demonstrated in isolation, showing that the ESP32 platform could process inputs from diverse sensors and execute corresponding control outputs [4].

However, the project's main limitation was the absence of a unified control architecture. While GPS-based navigation, PSD-based safety features, and camera-based perception were individually operational, they were not combined into a single cohesive system. As such, the ride-on car was not capable of continuous autonomous driving where localisation, navigation, and collision avoidance functioned simultaneously [4]. This reflects a common challenge in embedded autonomous systems: the integration of multiple sensor modalities into real time decision making on a constrained hardware platform.

The purpose of Guo's project was not only technical but also educational. By designing a low-cost autonomous vehicle framework, the project aimed to create a platform that could be replicated by secondary education students to introduce them to automation and robotics concepts. Compared with industrial autonomous vehicle research, which relies on high-performance processors and sophisticated software frameworks, the ride-on car project sought to demonstrate a simplified, accessible model of autonomy with minimal hardware and coding complexity.

Building on Guo's work, this project directly addresses the integration gap by developing a full autonomous ESP32 based control architecture capable of managing navigation, obstacle avoidance and actuation all under one unified program. By unifying previously independent modules, the system will enable full autonomous GPS navigation with integrated safety features. The aim is to simplify the hardware as much as possible to make the project an educational tool that is able to be utilised by secondary school students with their current knowledge and resources.

Project Objectives

Based on the reviewed literature and prior student projects, it is expected that an ESP32-based control architecture can deliver meaningful autonomous driving functions on low-cost vehicles. To validate this hypothesis, two key strands of work have been identified: the ride-on car and the EyeBots.

The first main objective is to autonomise a commercially available ride-on car using a TTGO T-Display S3 ESP32 microcontroller. To achieve this, all subsystems, including the GPS for localisation, PSD sensors for obstacle detection and safety, and actuator control systems, must be integrated into a single cohesive program. In addition, an ESP32-hosted webserver must be developed to allow users to input GPS destinations and receive real-time feedback from the vehicle, including current location, next destination, and PSD sensor readings. This webserver will enable remote monitoring and control of the autonomised ride-on car through a standard mobile phone or laptop interface.

The project therefore aims to develop a low-cost autonomous platform capable of navigating to user-defined GPS coordinates while actively detecting and avoiding obstacles. This must be achieved while working within several practical constraints, including limited processing power, low-cost sensors, restricted hardware interfaces, and the absence of true steering feedback from the vehicle. Furthermore, to make the platform educationally viable, the overall design must remain simple, reliable, and affordable, allowing secondary school students and beginner robotics users to replicate and adapt the system using basic tools, modular components, and minimal soldering.

The second main objective focuses on UWA's EyeBots, which were recently migrated from a Raspberry Pi to a T-Display S3 ESP32 platform by Bahrami. While the hardware transition was successful, validation of the EyeBot's libraries and their compatibility with ELEC3020 laboratory tasks remains incomplete. The objective is to test the ROBIOS library functions created for the new EyeBots to see if they perform the functionality that is required, if the functions do not perform as intended then create new functions or redefine the created function so they are able to achieve their specific functionality. Additionally, after all the functions have been created or fixed, the objective is to complete the last three of the embedded systems unit's lab - driving with distance sensors, camera-based object tracking, and task-based navigation- using functions from the ROBIOS library. Lastly another objective is to provide feedback to Bahrami on how the T-display S3 perform for autonomous driving. This work directly supports the continuity of teaching in ELEC3020 while contributing to the broader understanding of microcontroller-based autonomy.

Project Design Approach and Process

Embedded Systems EyeBots

The methodology adopted for the Embedded Systems EyeBots project focused primarily on validating and redeveloping the camera and image-processing functions within the ROBIOS library for operation on the ESP32-based EyeBot platform. The objective was to ensure reliable hardware performance while maintaining compatibility with the existing ELEC3020 Embedded Systems laboratory exercises. Unlike earlier implementations that were developed largely without access to the physical hardware, this methodology prioritised practical testing directly on the EyeBot platform.

Initially, each camera-related ROBIOS function was tested on the TTGO T-Display S3 ESP32 [5] (shown in Figure 4) hardware to determine whether it operated correctly under real-world conditions. Testing involved validating image capture, image formatting, colour extraction, and real-time image-processing behaviour. Although the ESP32 camera hardware itself successfully produced image output, significant issues were identified in the previous image-processing implementation. Earlier OpenCV-style processing functions expected image data in a different format from that natively produced by the ESP32 camera module, resulting in image interpretation and processing failures during testing.

Figure 4 shows the T-Display S3 ESP32

Rather than attempting to patch the unreliable implementations, the camera-processing functions were redeveloped from first principles. This involved redesigning the image-processing pipeline to correctly interpret the native camera output format while ensuring compatibility with the memory and processing limitations of the ESP32 platform. Functions were rebuilt incrementally and continuously validated on physical hardware to ensure reliable operation before integration into the wider ROBIOS framework.

An important aspect of the methodology was validating the updated functions against actual ELEC3020 laboratory tasks. Since the EyeBots are used for educational purposes, functional correctness alone was insufficient unless the system could reliably perform the intended student exercises. Testing included colour detection, colour-based object following, and real-time object tracking using hue-based image processing [3]. The updated ROBIOS functions were ultimately deployed during the previous semester's ELEC3020 Embedded Systems laboratories, where they were tested extensively across multiple student groups in a large-scale educational environment.

Overall, the methodology emphasised hardware-first validation, iterative redevelopment, and educational reliability to ensure the ESP32-based EyeBot platform could operate consistently within teaching.

Autonomous Ride on Car

Design Requirements and Constraints

Design requirements set out by Professor Thomas Bräunl specified that the project was to be developed using the UWA REV Team's existing Ride-on Car platform, shown in Figure 1, as the foundation for the autonomous vehicle system. Prior to the commencement of this project, the vehicle had already undergone several hardware modifications to improve safety and simplify integration with autonomous control systems [6].

One of the key pre-existing modifications was the integration of a wireless emergency stop (E-Stop) system using a FOB remote (shown in Figure 5), allowing the vehicle to be immediately disabled during emergency situations. Additionally, an external 24 V power access point was added through a 2.1 mm barrel connector mounted on the vehicle chassis. This modification routed power directly from the ride-on car's internal 24 V battery system to an externally accessible socket, allowing external electronics and control systems to be powered without altering the original vehicle wiring. In this project, this power source was used to supply the custom ESP32-based control box through a DC-DC converter [6].

Figure 5 shows the FOB remote for E-Stop

The ride-on car also already utilised a drive-by-wire remote-control architecture, where steering and vehicle motion were controlled through a wireless remote controller. Previous UWA Robotics Club work demonstrated that the remote controller PCB could be electronically interfaced by replacing the push buttons with transistor-based switching circuits, enabling external microcontrollers to simulate button presses programmatically, the board is shown in Figure 6 [6].

Figure 6 shows the modified remote controller PCB that was used in the project

While this approach provided a simple and practical method for achieving digital vehicle control, it also introduced several limitations. As the car was originally designed only for manual remote operation, there was no direct electronic interface available for autonomous control. Consequently, interfacing with the existing remote controller became the easiest way to control the vehicle digitally without redesigning the internal steering and motor electronics.

However, the autonomous system inherited the behavioural limitations of the original remote-control design. In particular, the steering system did not automatically return to a neutral position after a steering command was released. Instead, the front wheels remained fixed at their steering angle until an opposite steering command was applied. Furthermore, the vehicle provided no steering feedback to the controller, as there was no steering encoder, potentiometer, or other sensor capable of measuring wheel orientation. This meant the autonomous system could not directly determine the steering angle of the vehicle at any given moment.

Another major design requirement specified by Professor Bräunl was the use of the ESP32 TTGO T-Display S3 as the primary embedded controller. The module offered several advantages, including integrated WiFi capability, compact size, onboard display functionality, and sufficient processing capability for embedded autonomous control tasks. However, the ESP32 also introduced hardware constraints, particularly due to its limited number of accessible GPIO and communication pins. These limitations became important when integrating multiple peripherals including the GPS module, IMU, PSD safety sensors, and modified remote-control circuitry. Careful allocation of GPIO pins and communication buses was therefore required to ensure reliable system integration [5].

Finally, one of the most important objectives of the project was for the autonomous ride-on car to function as an educational STEM platform for high school students. As a result, the system needed to remain low-cost, simple, and reproducible. The hardware modifications had to be achievable using basic tools, simple soldering techniques, and commonly available components, while the software architecture also needed to remain understandable for students with limited prior experience in electronics or robotics.

Hardware Component Selection

Controller Used

The TTGO T-Display S3 ESP32 -shown in Figure 4- serves as the central processing unit of the autonomous ride-on car, integrating all sensing, control, and communication functions into a single low-cost embedded platform [5]. The microcontroller processes localisation data from the GPS module, heading information from the IMU, and obstacle detection data from the PSD sensors to make real-time driving decisions. Additionally, the ESP32 hosts the onboard webserver, enabling wireless communication between the ride-on car and the user interface. The board was selected due to its low cost, integrated WiFi capability, compact size, and suitability for educational robotics applications.

Power System Components

The LM2596 buck convertor (shown in Figure 7) is used to safely step down the ride on car's 24V battery supply to a stable 5V power supply [7] for the ESP32 module, which in turn powers all the sensors and other electronic components used. It is sufficient to maintain a simple and low-cost electrical architecture.

Figure 7 LM2956 DC-DC voltage convertor

Ride-on Car Remote Control Interface

The ride-on car remote control (shown in Figure 8) acts as the interface between the ESP32 and the vehicle's existing drive system. Rather than directly modifying the vehicle's internal motor controllers, the ESP32 simulates button presses on the remote-control circuitry to command forward, reverse, left, and right movement. This approach significantly simplifies integration with the commercially available ride-on car while preserving the original safety and drive electronics. It also reduces hardware complexity and makes the system easier for students to replicate using basic electronics knowledge [6].

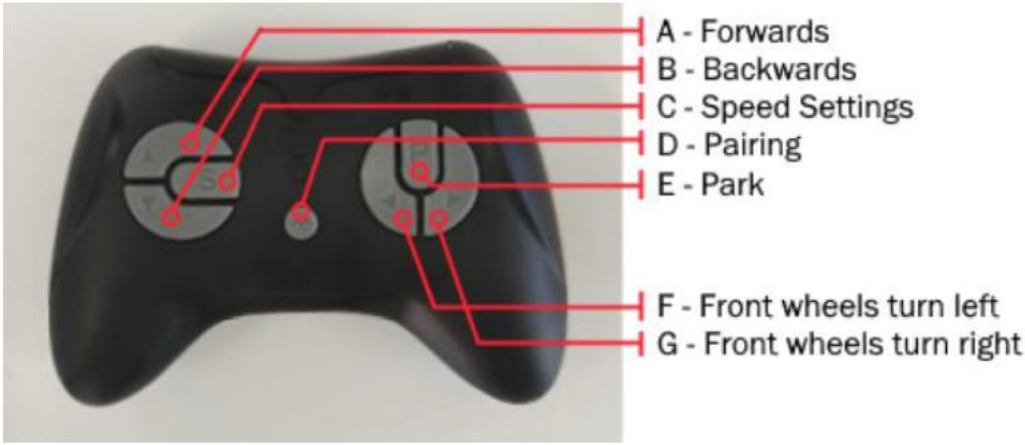


Figure 8: The original remote controller for the ride-on-car and its labelled button commands

GPS selection & final choice

The original ride-on car platform inherited from Guo utilised a Columbus V-800 USB GPS connected to the ESP32 using an Arduino USB Host Shield [4]. While this configuration successfully enabled GPS communication with the microcontroller, the USB Host Shield consumed approximately four GPIO pins and introduced significant additional hardware size and complexity within the already space-constrained control box. Since one of the major project objectives was to minimise hardware complexity and preserve GPIO resources on the ESP32, the USB-based GPS architecture was considered unsuitable for the final design.

To reduce GPIO usage and simplify communication, a DB9 serial GPS module was then investigated as an alternative solution. Unlike the USB-based GPS, the DB9 module only required two communication pins (RX and TX) through the ESP32 UART interface. However, significant implementation issues were encountered during testing. Although serial data was being received, the output could not initially be interpreted correctly by the ESP32. Further research revealed that the GPS operated using RS232 communication levels rather than the TTL logic levels expected by the ESP32[8]. To address this issue, a MAX3232 RS232-to-TTL conversion module was introduced to translate the signal levels between the GPS and the microcontroller [8].

Despite resolving the communication-level incompatibility, further complications arose relating to power delivery. The DB9 GPS module relied on a non-standard power configuration where operating voltage was supplied through a custom pin within the DB9 connector itself. As a result, the MAX3232 module could not directly provide power to the GPS without additional custom circuitry and wiring modifications. Continuing with this implementation path would have significantly increased hardware complexity while reducing the educational simplicity and reproducibility of the overall system.

As a result, the NEO-6M GPS module (shown in Figure 9) was selected for the final design. Connected to the ESP32 through a standard UART serial interface, the GPS continuously transmits latitude and longitude coordinates [9] used to determine the vehicle's current location relative to the target destination. Unlike the previous GPS solutions, the NEO-6M was specifically designed for Arduino and microcontroller-based systems, operating directly at compatible voltage levels without requiring additional interface hardware. This significantly simplified system integration, reduced GPIO usage, minimised control box complexity, and aligned more effectively with the project's low-cost and educational design objectives.

Figure 9: Arduino compatible u-blox NEO-6M GPS

IMU/ Heading feedback Components

Due to the absence of steering angle feedback within the ride-on car platform, an additional sensing system was required to estimate the vehicle's direction of travel during autonomous navigation. Without this feedback, the ESP32 would be unable to determine whether steering corrections were successfully changing the vehicle's heading toward the target destination. To address this limitation, the Gravity 10 DOF IMU AHRS BNO055 (shown in Figure 10) was integrated to provide real-time orientation and heading information. Connected via an I2C interface, the BNO055 contains integrated sensor fusion capabilities which combine accelerometer, gyroscope, and magnetometer data internally to provide stable heading estimation. This heading data allows the ESP32 to compare the vehicle's current orientation against the required GPS bearing toward the destination [10]. The sensor was selected into the final design because it simplifies orientation estimation while reducing computational requirements on the ESP32.

Figure 10: Gravity 10 DOF IMU used for the control box

Safety and Obstacle Avoidance Sensors

The Sharp GP2Y0A21 PSD sensors (shown in Figure 11) provide short-range obstacle detection capabilities for collision avoidance and operational safety. These sensors were chosen by Guo during his iteration of this project, and a decision was made to continue using the same sensors as there was hardware such as the 7 pin socket and 9-core shield data cable was already present on the modified car [4]. The sensors also worked well with the new hardware architecture due to their low cost, simple analogue interface, and minimal processing requirements [11] compared to more advanced sensing technologies such as LiDAR or computer vision systems.

Figure 11: The SHARP PSD sensors used for the project

Final Design

System Architecture

Given the limited number of GPIO pins available on the TTGO T-Display S3 [5], careful pin budgeting was essential throughout the process, the allocated PIN allocation can be seen in Table 1 and the associated full wiring diagram of the control box of the final design can be seen in Figure 12.

Table 1: GPIO pin connections between hardware components and T-Display S3 ESP32 Microcontroller

TTGO T-Display S3 Pin	**Peripheral Pin/Description**
43	*IMU Sensor SDA*
44	*IMU Sensor SCL*
18	*GPS UART RX*
17	*GPS UART TX*
16	*Left PSD*
10	*Right PSD*
11	*Front PSD*
12	*Remote Down PIN (Backwards)*
13	*Remote Up PIN (Forward)*
1	*Remote Right PIN*
2	*Remote Park PIN*
3	*Remote Left PIN*
21	*Remote Connection PIN*

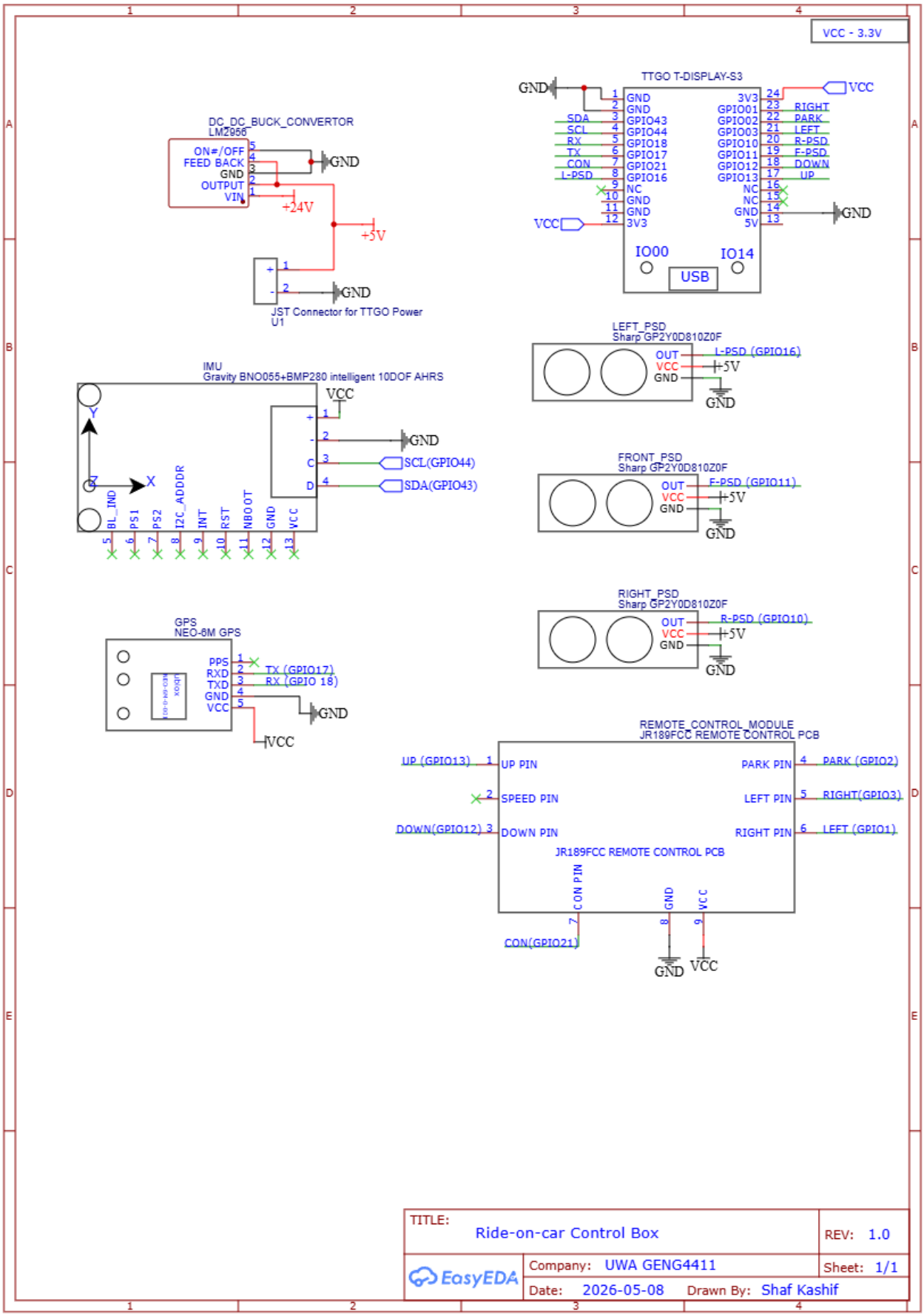


Figure 12: Ride-on-car's control box's final wiring diagram

Hardware Integration

Power System

As the supplied power coming from the Ride-on-car's battery is 24V and as the ESP32 and all other components operating voltage is between 5V – 3.3V, the battery's voltage must be stepped down using the LM2596 DC-DC convertor. The decision was made to step down the voltage directly to 5V and supply it to the ESP32 through its JST power which in turn will power the other components such as PSDs through the 5V pin and the IMU, GPS and remote module through the various 3.3V pins. All ground are common; all components ground are attached to the ESP and the ESP32's JST power's connector is connected to the ground through the buck convertor's terminal blocks to the ground of the battery.

Ride-On-Car Modifications

The ride-on car platform utilised in this project had previously undergone several modifications as part of earlier autonomous vehicle investigations conducted within the UWA REV Team. Existing modifications included the integration of a wireless emergency stop (E-Stop) system for operational safety, a 24 V external barrel connector for simplified power access, and a remote-control-based drive-by-wire system which enabled electronic control of the vehicle without directly modifying the original motor controllers [6]. Building on this foundation, additional hardware including the ESP32 control box, GPS module, IMU, PSD sensors, and associated wiring were integrated into the vehicle to support autonomous navigation and obstacle detection. As the project was a continuation of Guo's project and work on the ride-on-car some of the hardware built by Guo was used which included the modified remote-control module that allowed us to control the car through a controller and the work done on PSD sensors for mounting and connecting it to the controller [4].

A key design objective throughout the integration process was to minimise permanent modifications to the commercially available ride-on car. This approach preserved the original vehicle electronics and safety systems while avoiding invasive rewiring or custom hardware fabrication. Maintaining a modular and low-complexity architecture also improved maintainability, troubleshooting, and educational accessibility. Components were mounted in accessible locations using simple fastening methods and modular wiring connections, allowing the system to be more easily replicated by secondary school students and beginner robotics users. To further support educational adoption, a detailed user manual outlining the full hardware replication and setup process has been included in the appendix of this report.

Sensor Mounting

The three PSD sensors were mounted at the front of the ride-on car, with one sensor positioned facing directly forward, one angled slightly to the left, and the third angled slightly to the right, as shown in Figure 13. Since the vehicle primarily operates using forward motion, the front-mounted configuration provides the most effective coverage for obstacle detection, collision avoidance, and operational safety. This arrangement enables the system to detect obstacles directly ahead of the vehicle while also monitoring nearby objects approaching from either side of the driving path.

Figure 13: The 3 PSD sensors positioned in front of the car at different angles

The IMU was mounted inside the control box, as shown in Figure 14, to simplify system integration and protect the sensor from external damage and environmental exposure. Since the IMU contains an onboard magnetometer used to determine the vehicle's heading relative to the Earth's magnetic field (North, East, South, and West), careful attention was given to its orientation during installation. The IMU's forward axis was aligned with the forward direction of the ride-on car to ensure that the heading and orientation data accurately corresponded to the vehicle's actual movement direction as shown in Figure 15 [12].

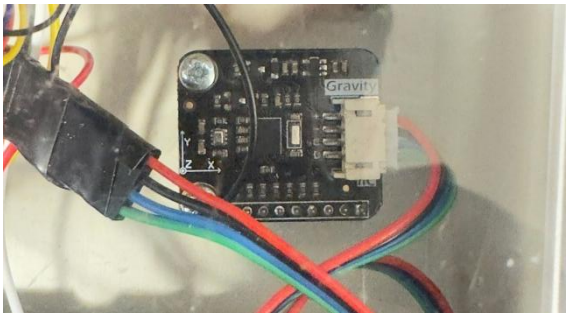


Figure 14: The mounted IMU inside the control box

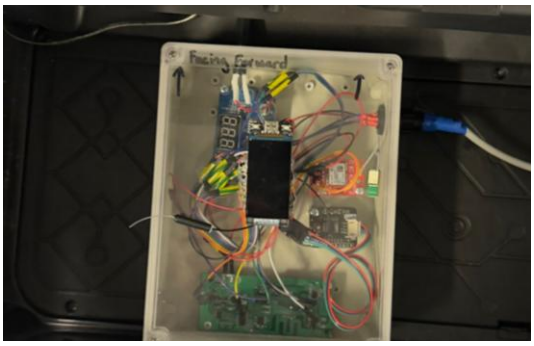


Figure 15: The Control box needs to have a forward-facing side due to IMU needing to know the corresponding direction of the car

Similarly, the GPS module was mounted inside the control box to maintain a compact and organised system layout. However, as the GPS antenna requires a clear line of sight to satellite signals and can experience reduced performance when obstructed by solid materials, the antenna was routed externally through the lid of the enclosure, as shown in Figure 16. The antenna was positioned to face upward and outward from the control box to maximise signal reception and improve positional accuracy.

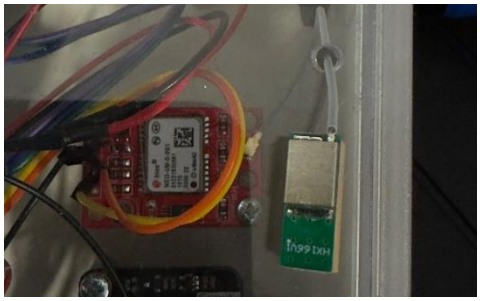


Figure 16: The GPS and then antenna mounted on the control box

Additionally, the control box itself was mounted behind the vehicle seats in an open, uncovered area of the ride-on car, as shown in Figure 17. This placement further reduced potential signal obstruction from the vehicle body and provided the GPS antenna with a clearer view of the sky, improving the reliability and consistency of GPS communication during operation.

Figure 17: Shows the control box mounted behind the seats of the open top car to provide clear signal for the GPS antenna

Wiring and Cable Management

All wiring connected to the control box was designed to require minimal soldering. Instead, the majority of sensors and modules were connected directly to the ESP32 controller using Dupont connector cables, simplifying assembly, maintenance, and replacement for students and future users. This approach also improved the accessibility and reproducibility of the system, making it easier for educational environments to replicate the design without requiring advanced soldering skills.

To improve reliability, any connected Dupont cable pairs were secured together using electrical tape to prevent accidental disconnection caused by vehicle movement or vibrations during operation. Furthermore, cable management was carefully considered throughout the system design. In particular, the long wiring routed between the front-mounted sensors and the rear control box was secured along the underside and side sections of the ride-on car using cable management clips. This prevented loose or dangling cables from being visible, interfering with vehicle operation, or creating potential safety hazards during driving.

Bill of Materials & Cost Analysis

Table 2: Shows the bill of Materials for the Ride-On-Car

Item	Quantity	Unit Cost (AUD)	Total Cost (AUD)
Ride-on Car	1	$699.00	$699.00
Additional Remote Transmitter	1	$0.00	$0.00
ABS Control Box Enclosure	1	$33.25	$33.25
Wireless E-Stop FOB Remote	1	$45.00	$45.00
Wireless E-Stop Receiver	1	$69.95	$69.95
TTGO T-Display S3 ESP32	1	$23.00	$23.00
NEO-6M GPS Module	1	$17.95	$17.95
BNO055 IMU	1	$21.50	$21.50
LM2596 DC-DC Buck Converter	1	$8.55	$8.55
Sharp PSD Sensors	3	$14.41	$43.23
Waterproof Socket & Plug Connectors	2	$22.50	$45.00
JST SH 4-pin Cable	1	$0.95	$0.95
Transistors	7	$0.35	$2.45
Resistors	1	$0.60	$0.60
Prototype Ribbon Cables	2	$5.20	$10.40
DC Power Sockets & Cables	3	$5.60	$17.00
Shielded Data Cable	3	$2.25	$6.75
Total			**$1,039.23**

The total hardware cost of the autonomous ride-on car platform was approximately AUD $1,039 as seen in Table 2, including the commercially available ride-on car itself. Compared to previous Raspberry Pi–based autonomous vehicle platforms used within UWA, which cost approximately AUD $1,200 [6] while relying on older hardware architectures, the ESP32-based system achieved a lower overall cost while utilising newer and more compact embedded hardware. The use of the TTGO T-Display S3 ESP32 significantly reduced complexity by integrating WiFi communication, processing, and display functionality into a single low-cost module, removing the need for additional computing hardware or supporting peripherals commonly required in Raspberry Pi systems.

Furthermore, compared to many modern autonomous vehicle research platforms which rely on laptop-class processors, industrial PCs, or high-performance embedded computing systems, the proposed architecture remained substantially more affordable and accessible. The simplified hardware design, modular wiring approach, and minimal soldering requirements also made the platform significantly easier for secondary school students and beginner robotics users to understand, assemble, and maintain. This supports the project's primary objective of developing an educational autonomous driving platform that balances functionality, simplicity, and affordability.

Software Architecture

The software architecture of the autonomous ride-on car was designed around a modular embedded control system running entirely on the TTGO T-Display S3 ESP32 microcontroller. The ESP32 acts as the central processing unit responsible for sensor acquisition, navigation calculations, webserver communication, safety monitoring, and actuator control. A simplified software architecture flowchart is shown in Figure 18. The overall software structure was designed to remain lightweight and efficient due to the limited processing and memory resources available on the microcontroller [5] while still allowing multiple subsystems to operate simultaneously in real time.

During system startup, the ESP32 initialises all required hardware peripherals and communication interfaces. This includes configuring GPIO outputs for the ride-on car remote-control interface, establishing UART communication with the NEO-6M GPS module, initialising the BNO055 IMU through the I2C bus, configuring analogue inputs for the PSD sensors, and connecting to the local WiFi network. Once connected to WiFi, the ESP32 hosts a local webserver which allows the user to remotely interact with the autonomous vehicle through a mobile phone or laptop browser. The TFT display on the TTGO T-Display S3 is also initialised to provide local system feedback including IP address information, GPS data, heading information, and autonomous navigation status.

After initialisation, the software operates using a continuous non-blocking main loop structure. Within this loop, the ESP32 repeatedly polls all sensor inputs, updates navigation calculations, handles webserver requests, processes manual driving commands, and executes autonomous driving logic. GPS data is continuously decoded through the UART interface to determine the vehicle's current latitude and longitude position, while the IMU provides updated heading and orientation information. Simultaneously, the PSD sensors are sampled at regular intervals to monitor nearby obstacles and ensure operational safety during movement.

The webserver subsystem operates concurrently with the navigation system and enables real-time communication between the user and the vehicle. Through the browser interface, users can manually control the ride-on car, input target GPS coordinates, start or stop autonomous mode, and monitor live telemetry including GPS position, heading angle, target distance, and PSD sensor readings. Safety-related functions are prioritised within the software architecture, meaning obstacle detection routines can override navigation commands when required. If an obstacle is detected within the predefined threshold distance, the ESP32 immediately stops vehicle movement regardless of the current navigation state.

The modular structure of the software architecture also simplifies debugging, future expansion, and educational use. Individual subsystems such as GPS processing, webserver communication, obstacle detection, and actuator control can be independently modified or extended without requiring significant restructuring of the entire program. This supports the project's overall objective of developing an accessible and educationally reproducible autonomous driving platform using low-cost embedded hardware.

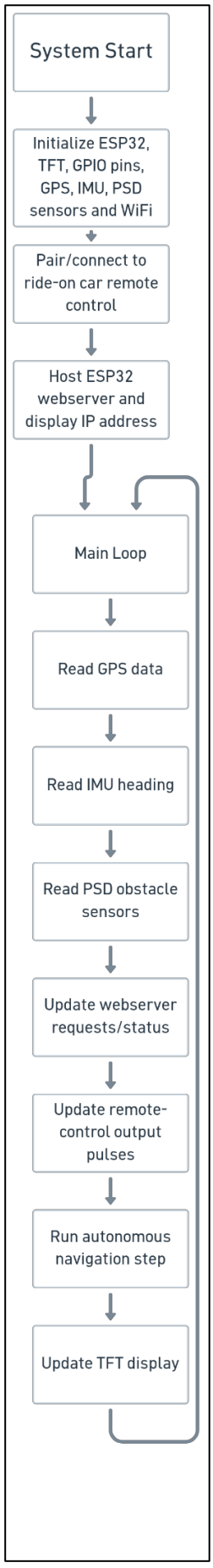


Figure 18: The software architecture flow chart

Vehicle Navigation and Driving Logic

The vehicle navigation and driving logic is based on a continuous closed-loop navigation approach which repeatedly updates the vehicle's heading and position while travelling toward a user-defined GPS coordinate. Rather than calculating a direction once and driving continuously, the ESP32 continuously polls sensor data throughout operation to dynamically adjust the vehicle's steering and movement. This allows the ride-on car to continuously correct its trajectory during navigation while also responding to nearby obstacles detected by the PSD sensors. Figure 19 presents a simplified flowchart of the navigation logic implemented within the system.

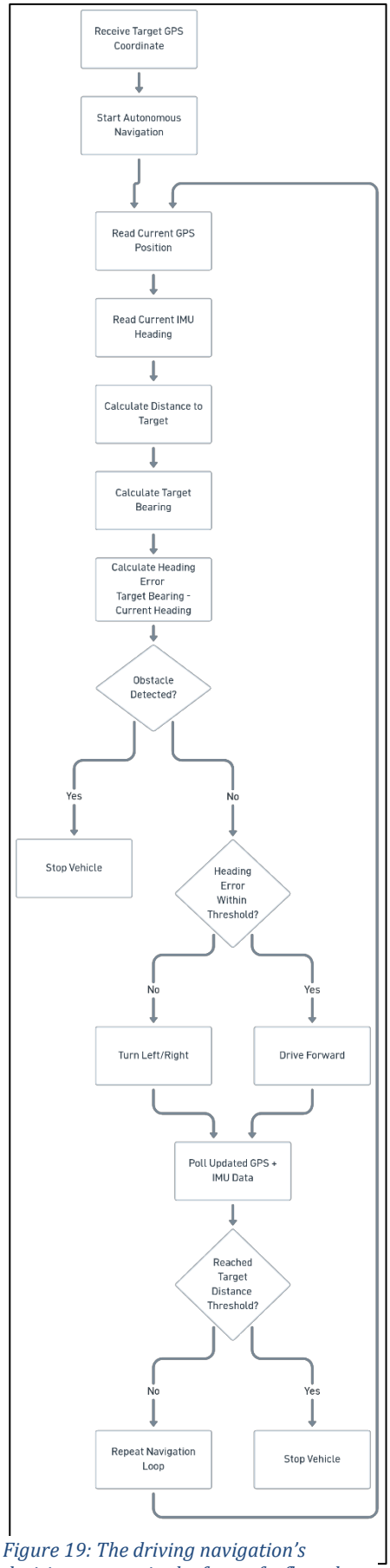


Figure 19: The driving navigation's decision process in the form of a flow chart

The navigation process begins when the user enters a target GPS coordinate through the ESP32-hosted webserver. Once autonomous mode is enabled, the ESP32 first checks that a valid GPS signal and IMU heading are available. The current GPS position is then compared with the target coordinate to calculate both the distance to the destination and the required bearing angle. Distance between the current location and target waypoint is calculated using the haversine formula, which estimates the great-circle distance between two latitude and longitude coordinates on the Earth's surface [13][14]:

$$a = sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2tan^{-1}\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right)$$

$$d = Rc$$

where a is an intermediate haversine calculation representing the angular separation between the two GPS coordinates, c is the angular distance in radians, $\phi_1$and $\phi_2$represent the current and target latitudes, $\lambda_1$and $\lambda_2$represent the current and target longitudes, $\Delta\phi$ and $\Delta\lambda$ are the differences in latitude and longitude between the two coordinates, R is the Earth's radius, and d is the calculated distance to the waypoint.

The target bearing toward the destination is then calculated using:

$$\theta = tan^{-1}\left(\frac{sin(\Delta\lambda)cos(\phi_2)}{\cos(\phi_1)\sin(\phi_2) - \sin(\phi_1)\cos(\phi_2)\cos(\Delta\lambda)}\right)$$

where θ represents the target bearing angle toward the waypoint. Simultaneously, the IMU provides the vehicle's current heading direction. The software then determines the heading error using:

$$e_\theta = \theta_{target} - \theta_{current}$$

where $e_\theta$represents the heading error between the required target bearing and the current IMU heading. Based on this error, the ESP32 decides whether the ride-on car should steer left, steer right, or continue driving forward. Instead of continuously controlling steering angles, the system uses short timed steering and throttle pulses through the modified remote-control interface. After each movement

step, the GPS and IMU data are polled again, allowing the ESP32 to recalculate the required bearing and heading correction. This repeated update cycle creates a simplified feedback-control system where the vehicle continuously adjusts its direction while moving toward the target coordinate.

Obstacle detection is prioritised throughout navigation using the front-mounted PSD sensors. Before each autonomous movement step, the ESP32 checks whether any obstacle is detected within a predefined safety threshold distance. If an obstacle is identified, the navigation logic overrides all movement commands and immediately stops the vehicle to prevent collision. The system then enters an obstacle-detection state until the path becomes clear or autonomous mode is disabled by the user.

Due to the absence of true steering angle feedback, the vehicle cannot precisely measure its wheel orientation. As a result, steering corrections are estimated using timed pulses rather than exact angular positioning. Combined with low GPS update rates and sensor noise, this can result in oscillatory or "stuttering" movement behaviour where the vehicle repeatedly corrects its heading while approaching the destination. Despite these limitations, the navigation logic demonstrates that meaningful autonomous GPS navigation can still be achieved using a low-cost ESP32-based architecture.

ESP32 Webserver Design

The ESP32 webserver was designed to provide a lightweight and accessible interface for remotely monitoring and controlling the autonomous ride-on car. One of the primary objectives of the webserver was to allow users to interact with the vehicle without requiring specialised software or additional hardware beyond a standard mobile phone or laptop. By hosting the webserver directly on the TTGO T-Display S3 ESP32 microcontroller, the system remained self-contained, low-cost, and suitable for educational environments where simplicity and ease of deployment were important design considerations.

The webserver operates through a shared WiFi network connection. Upon startup, the ESP32 connects to the selected wireless network and displays its assigned IP address on the integrated TFT display as seen in Figure 20. A user connected to the same WiFi network can then enter this IP address into a web browser to access the control interface. This approach eliminates the need for dedicated mobile applications or external servers while allowing real-time wireless communication between the user and the autonomous vehicle.

Figure 20: Shows the TFT Display on the ESP32 and the webserver IP address

The webserver interface was divided into two primary functional areas. The first and most significant component was the integrated live map display implemented using the Leaflet JavaScript mapping library as seen in Figure 21. The map primarily focused on James Oval at the University of Western Australia, which served as the main testing environment for autonomous navigation experiments. However, because the map dynamically updates using live GPS coordinates received from the ESP32, the system is capable of displaying vehicle location information anywhere in the world. A location marker is continuously updated on the map using GPS latitude and longitude data transmitted from the ride-on car, allowing users to remotely monitor the vehicle's position during operation. The map interface also enables users to select and send GPS target destinations directly to the vehicle for autonomous navigation.

The second component of the webserver displays live telemetry and system information generated by the ESP32, as seen in Figure 22. This includes current GPS coordinates, IMU heading information, target destination coordinates, distance to target, heading error, autonomous driving status, and PSD obstacle sensor readings. By presenting both navigation data and vehicle status information in real time, the webserver acts as both a remote-control interface and a live diagnostic tool during autonomous testing. The integration of mapping, telemetry, and wireless control into a single browser-based interface demonstrates how low-cost embedded hardware can support meaningful autonomous vehicle monitoring and interaction without relying on high-performance computing systems.

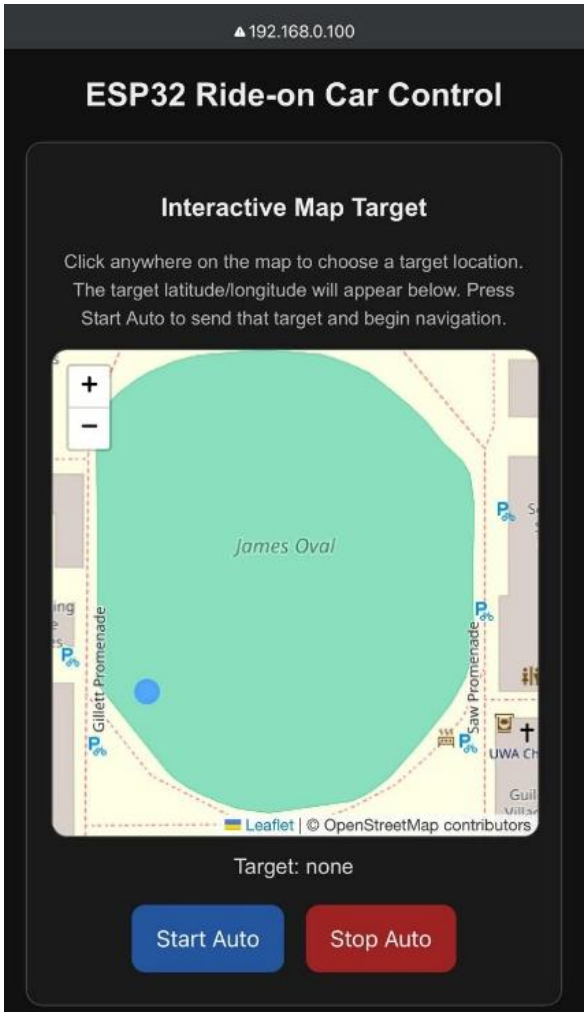


Figure 22: Shows the interactive leaflet-sourced Map and the current location of the car (blue marker)

Figure 21: Shows the Webserver's Live Telemetry & system information

Testing and Validation Methodology

Testing and validation of the autonomous ride-on car were primarily conducted outdoors at James Oval, as the NEO-6M GPS module was unable to reliably receive satellite signals indoors. Initial testing focused on validating GPS connectivity and localisation performance. Once outdoors, the ESP32 was successfully able to receive live latitude and longitude coordinates; however, the GPS accuracy was typically limited to approximately 4 m [8] from the vehicle's actual location. Due to this limitation, a target distance threshold was introduced into the navigation logic. Without this threshold, the vehicle would theoretically continue attempting to reach an exact coordinate that could not be reliably achieved using low-cost GPS hardware.

Following GPS validation, testing moved to the IMU heading system. During early testing it was observed that, immediately after startup, the IMU would frequently assume that the vehicle's current heading was 0° regardless of its actual orientation. This issue was traced to incomplete magnetometer calibration during initialisation. As the navigation logic relied heavily on accurate heading information, this caused incorrect bearing calculations and steering behaviour proportional to the heading offset. To address this issue, the IMU was manually calibrated during startup by physically moving the control box in a figure-eight motion, allowing the magnetometer to interact with the Earth's magnetic field and establish an accurate heading reference. Additional calibration could also be achieved by manually driving the ride-on car in circular motions prior to autonomous operation.

Results & Discussion

Autonomous Ride on Car

The autonomous ride-on car platform was successfully integrated using the TTGO T-Display S3 ESP32 microcontroller as the primary control system. The overall hardware architecture proved effective for educational and low-cost autonomous vehicle implementation, particularly due to the simplicity of integrating sensors through standard communication protocols such as UART for the NEO-6M GPS module and I2C for the BNO055 IMU. The modular wiring approach and minimal soldering requirements allowed the control box and sensor system to be assembled without specialised hardware fabrication or advanced electronics knowledge, supporting the project's objective of creating a platform suitable for secondary school robotics education. The simplified hardware design also improved maintainability and ease of troubleshooting while enabling the entire system to remain low cost and reproducible.

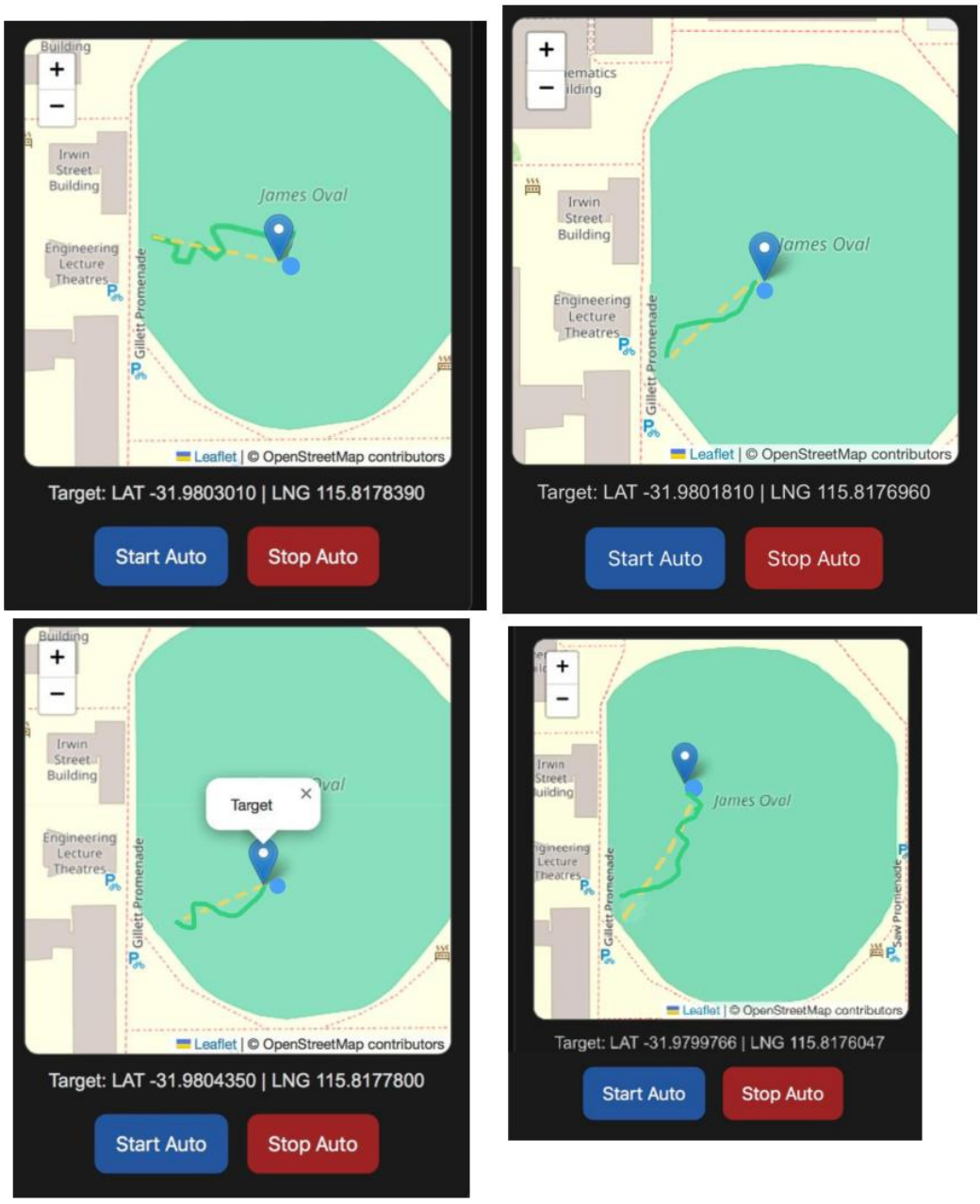


Figure 23: Webserver interface during autonomous waypoint navigation trials at James Oval, showing live GPS position tracking, user-defined target waypoints, and recorded vehicle navigation paths during autonomous operation.

The autonomous navigation system successfully demonstrated GPS waypoint-based driving using only the ESP32 microcontroller and low-cost sensors. Testing conducted at James Oval (as seen in Figure 23) showed that the ride-on car was capable of autonomously navigating from one location to another without major operational issues, provided the environment remained relatively open and free from dense obstacles. The software architecture performed reliably despite the limited processing power and GPIO availability of the ESP32 platform. The continuous navigation loop, which repeatedly recalculated GPS bearing and IMU heading during movement, enabled the vehicle to dynamically correct its trajectory throughout operation.

The ESP32-hosted webserver also performed successfully and became one of the most effective components of the system. The browser-based interface provided an intuitive and responsive method of controlling the vehicle while displaying real-time telemetry and diagnostic information. The Leaflet-based map interface allowed users to visualise both the target destination and current vehicle location directly on a live map using incoming GPS coordinates. This can also be seen in Figure 23 of trial photos. This significantly improved usability during testing as users could easily compare the virtual vehicle position displayed on the webserver with the physical location of the ride-on car within the testing environment. Additionally, live telemetry including heading information, target distance, heading error, autonomous driving state, and PSD sensor values greatly simplified debugging and system validation by allowing faults and unexpected behaviours to be identified in real time.

Despite the successful implementation of autonomous waypoint navigation, several engineering limitations affected overall driving performance. The most significant limitation was the absence of steering angle feedback. Because the steering system had no encoder or positional feedback mechanism, steering corrections relied entirely on timed control pulses and IMU-estimated heading changes. As a result, the vehicle frequently exhibited oscillatory or “snake-like” movement (as seen in Figure 23 & Figure 24) behaviour where repeated heading corrections caused curved driving paths rather than perfectly straight trajectories. While this behaviour could not be fully eliminated due to the hardware limitations of the platform, tuning of steering pulse durations, heading thresholds, and navigation update rates significantly reduced excessive overcorrection and improved overall driving stability.

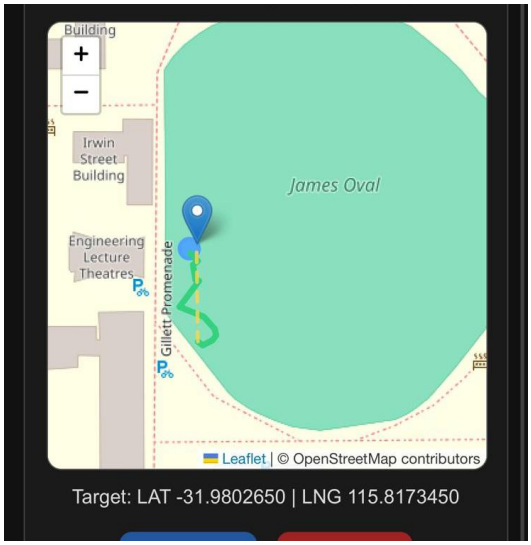


Figure 24 Shows the path taken of an autonomously navigated route, the green path shows the path taken, whilst the yellow dotted path shows the optimal path

Another major limitation involved the obstacle detection system. The Sharp GP2Y0A21 PSD sensors were intentionally selected to maintain low cost, simplicity, and educational accessibility; however, their operating range of approximately 10 cm to 80 cm [11] significantly limited their

effectiveness for outdoor autonomous vehicle navigation. In open-space environments such as James Oval, realistic stopping distances are typically measured in metres rather than centimetres. Consequently, the sensors functioned more effectively as short-range emergency stop devices rather than true obstacle avoidance sensors. The PSD sensors also occasionally produced false-positive detections which unnecessarily stopped the vehicle during operation. Although more advanced sensing technologies such as LiDAR or computer vision systems could have improved obstacle detection performance, these approaches would have significantly increased hardware complexity, computational requirements, and overall project cost, conflicting with the project's educational objectives.

Several improvements were made over the previous ride-on car implementation developed by Guo. One of the most significant improvements was the removal of the USB host shield architecture and replacement with a GPS module directly compatible with microcontroller UART communication. This reduced hardware complexity and freed valuable GPIO resources on the ESP32, enabling integration of the BNO055 IMU which substantially improved heading estimation and navigation performance. The physical implementation of the control system was also improved through better cable management, sensor mounting, and enclosure organisation, resulting in a cleaner and more stable control box architecture. Mounting the control box and WiFi hotspot securely behind the vehicle seats prevented hardware movement during driving and improved the overall reliability and professionalism of the system.

Overall, the project successfully demonstrated that a low-cost ESP32 microcontroller can perform meaningful autonomous vehicle functions including GPS waypoint navigation, wireless telemetry, remote monitoring, and basic obstacle detection using simplified hardware suitable for educational implementation. Although limitations existed due to the low-cost architecture and simplified sensing systems, the project achieved its primary objective of developing an accessible autonomous driving platform capable of introducing secondary school students to real-world robotics and automation concepts.

Embedded Systems EyeBots

The updated camera and image-processing functions were successfully validated through the implementation of the ELEC3020 laboratory tasks involving colour detection and object tracking. Testing demonstrated that the ESP32-S3 WROOM camera module reliably captured and processed image data in real time, with the revised image-processing pipeline correctly handling the native RGB565 image format produced by the camera hardware. This resolved the compatibility issues present in earlier implementations, where OpenCV-style processing functions expected a different image structure and therefore failed during hardware testing.

The rebuilt camera functions successfully supported core image-processing tasks including grayscale conversion, RGB extraction, HSI colour-space processing, and hue-based object tracking. During testing, the EyeBots were able to consistently detect and follow coloured objects with reliable performance and minimal instability.

Importantly, the updated functions were not only validated in isolated testing but were also deployed within the Semester 2 ELEC3020 Embedded Systems laboratories, where they were used by many students in that unit [3]. The camera functions operated reliably throughout the laboratory rollout, allowing students to complete vision-based tasks such as colour tracking and object interaction successfully. This large-scale classroom testing demonstrated the educational reliability and robustness of the updated camera-processing implementation within a real teaching environment [3].

Conclusion & Future Work

This project investigated the feasibility of implementing autonomous driving functionality using low-cost ESP32 microcontroller hardware for educational and research purposes. Through the integration of GPS localisation, IMU heading estimation, obstacle detection sensors, wireless telemetry, and a browser-based control interface, the project successfully demonstrated that meaningful autonomous vehicle behaviour can be achieved without relying on high-performance embedded computing platforms. The ride-on car was capable of autonomously navigating toward user-defined GPS waypoints while continuously recalculating heading and trajectory using real-time sensor feedback.

A major contribution of the project was the development of a simplified and reproducible hardware architecture suitable for educational environments. By prioritising modular wiring, minimal soldering, and low-cost commercially available components, the system was designed to be accessible to secondary school students and beginner robotics users. The ESP32-hosted webserver also successfully provided live telemetry, remote vehicle control, and interactive GPS waypoint selection through a standard browser interface, further supporting the project's educational objectives.

Despite the successful implementation of autonomous GPS navigation, several limitations remained. The absence of steering angle feedback significantly affected driving smoothness and resulted in oscillatory steering behaviour during navigation. Additionally, the Sharp PSD sensors provided only limited short-range obstacle detection capability and were insufficient for advanced obstacle avoidance in large open environments. These limitations highlight the challenges associated with implementing autonomous driving systems on simplified low-cost hardware architectures.

Future work should focus on improving obstacle detection and autonomous driving capability through the integration of more advanced sensing technologies such as LiDAR or depth-based vision systems. This would allow implementation of more sophisticated autonomous navigation algorithms including DistBug [15], Pure Pursuit [16], or other path-planning and obstacle-avoidance methods that require accurate environmental perception beyond short-range proximity sensing. Additional future improvements could include the addition of steering angle feedback through encoders or potentiometers to enable true closed-loop steering control and improve navigation smoothness.

The project additionally contributed to the continued development of UWA's ESP32-based EyeBot platform through the validation and redevelopment of camera-processing functions used in ELEC3020 Embedded Systems laboratories. The updated ROBIOS functions successfully enabled reliable colour tracking and object-following tasks during large-scale classroom deployment, demonstrating the suitability of low-cost ESP32 platforms for robotics education.

Overall, the project demonstrated that ESP32-based autonomous vehicle systems can provide a meaningful, low-cost, and educationally accessible platform for introducing students to real-world robotics, embedded systems, and autonomous driving concepts.

References

1. Lee, J., Bagheri, B., & Kao, H. A. (2015). A cyber-physical systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters, 3*, 18–23. https://doi.org/10.1016/j.mfglet.2014.12.001

2. Benitti, F. B. V. (2012). Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education, 58*(3), 978–988. https://doi.org/10.1016/j.compedu.2011.10.006

3. Bahrami, P. (2025). *EyeBot-32: An ESP-32 powered robotic platform for educational purposes.* Perth : University of Western Australia.

4. Guo, J. (2025). *Exploring the Potential of the ESP-32 Microcontroller for.* Perth: University of Western Australia.

5. Espressif Systems. (2024). *ESP32-S3 series datasheet*. Espressif Systems. https://www.espressif.com/
6. UWA Robotics Club. (2022). *Autonomous Ride-on Car Ver. 1.0: A STEM project for high schools*. UWA Robotics & Automation Lab. Retrieved September 13, 2024, from https://roblab.org/rideon
7. Texas Instruments. (2016). *LM2596 SIMPLE SWITCHER® power converter datasheet*. Texas Instruments. https://www.ti.com/
8. Texas Instruments. (2015). *MAX3232 3-V to 5.5-V multichannel RS-232 line driver and receiver datasheet*. Texas Instruments. https://www.ti.com/
9. u-blox. (2023). *NEO-6 GPS modules data sheet*. u-blox AG. https://www.u-blox.com/
10. Bosch Sensortec. (2021). *BNO055 intelligent 9-axis absolute orientation sensor datasheet*. Bosch Sensortec GmbH. https://www.bosch-sensortec.com/
11. Sharp Corporation. (2019). *GP2Y0A21YK0F distance measuring sensor unit datasheet*. Sharp Corporation.
12. Madgwick, S. O. H. (2010). *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. University of Bristol.
13. Movable Type Scripts. (n.d.). *Calculate distance, bearing and more between latitude/longitude points*. https://www.movable-type.co.uk/scripts/latlong.html
14. Esri. (2017, October 5). *Distance on a sphere: The haversine formula*. Esri Community. https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128
15. Kamon, I., Rivlin, E., & Rimon, E. (1998). TangentBug: A range-sensor-based navigation algorithm. *The International Journal of Robotics Research, 17*(9), 934–953. https://doi.org/10.1177/027836499801700903
16. Coulter, R. C. (1992). *Implementation of the pure pursuit path tracking algorithm*. Carnegie Mellon University.

Appendices

Appendix A: Ride-on-Car Code

Platform IO File:

```
[env:lilygo-t-display-s3]
platform = espressif32
board = lilygo-t-display-s3
framework = arduino
monitor_speed = 115200
board_build.filesystem = littlefs

lib_deps =
  Bodmer/TFT_eSPI@^2.5.35
  adafruit/Adafruit BNO055 @ ^1.6.4
  adafruit/Adafruit Unified Sensor @ ^1.1.9
  mikalhart/TinyGPSPlus@^1.0.3
  https://github.com/me-no-dev/ESPAsyncWebServer.git
  https://github.com/me-no-dev/AsyncTCP.git


build_flags =
  -DUSER_SETUP_LOADED=1
  -DUSER_SETUP_FILE="TTGO_T_Display_S3.h"
  -include $PROJECT_LIBDEPS_DIR/$PIOENV/TFT_eSPI/User_Setups/Setup206_LilyGo_T_Display_S3.h


; (Optional but handy)
monitor_filters = esp32_exception_decoder, time
```

Important Pin Allocation

```
include <ESPAsyncWebServer.h>

#include <Arduino.h>

#include <Wire.h>

#include <TFT_eSPI.h>

#include <SPI.h>

#include <TinyGPS++.h>

#include <Adafruit_Sensor.h>

#include <Adafruit_BNO055.h>

#include <utility/imumaths.h>

#include <math.h>


#define TFT_BL 38
```

```
TFT_eSPI tft = TFT_eSPI();
const int pinFrontPSD = 11;
const int pinLeftPSD  = 16;
const int pinRightPSD = 10;

// I2C pins (shared by BNO055)
const int I2C_SDA = 43;
const int I2C_SCL = 44;

// ---------------- Serial Pins for GPS ----------------
const int pinRX = 18; // RX of GPS
const int pinTX = 17; // TX of GPS

// ---------------- Controller Pins ----------------
const int pinLeft  = 3;   // Green wire
const int pinRight = 1;   // Orange wire
const int pinUp    = 13;  // White wire
const int pinDown  = 12;  // Grey wire
const int CON      = 21;  // Blue wire
const int parkPin  = 2;   // Yellow wire
//const int speedPin = 11;  // Purple wire (disabled for now)

// ---------------- GPS ----------------
#define GPS_BAUD 9600
TinyGPSPlus gps;
HardwareSerial gpsSerial(2);
```

Appendix B: Bill of Materials for Ride-on-Car

PART CODE	DESCRIPTION	QUANTITY	UNIT COST	TOTAL	SUPPLIER
RIDE ON CAR	Beach Buggy Dune, 400W	1	$699.00	$699.00	ArtInToys
aditional remote transmitter	additional transmitter for integrstion with embedded controller	1		$0.00	ArtInToys
CONTROL BOX COMPONENTS					
H0330	186Lx146Wx75Hmm IP65 Sealed ABS Enclosure	1	$33.25	$33.25	Altronics
AA1017A	FOB43301L PentaFOB Single Large Button 433Mhz Remote	1	$45.00	$45.00	Altronics
A1025A	PCR43301RE Penta Series 433Mhz 1 Channel Receiver	1	$69.95	$69.95	Altronics
T-Display-S3	LILYGO T-Display-S3 ESP32-S3 Development Board	1	$23.00	$23.00	Amazon
NEO-6M	NEO-6M GPS Breakout Module with Antenna	1	$17.95	$17.95	Altronics
DF-SEN0253	Gravity BNO055+BMP280 intelligent 10DOF AHRS IMU	1	$21.50	$21.50	Core Electronics
DFR0379	20W Adjustable DC-DC Buck Converter with Digital Display	1	$8.55	$8.55	Core Electronics
GP2Y0A21YK0F	GP2Y0A21YK0F Analog Distance Sensor 10-80cm	3	$14.41	$43.23	DigiKey
P9407	7 Pin 5A Screw-On Female Line IP66 Waterproof Socket	1	$30.50	$30.50	Altronics
P9467A	7 Pin 5A Locking Male Chassis IP67 Waterproof Plug	1	$18.50	$18.50	Altronics
CONTROLLER					
Z1042	NPN BC548B T092h General Purpose Transistor	7	$0.35	$2.45	Altronics
R7046	1k 0.25W 5% Carbon Film Resistor PK 10	1	$0.60	$0.60	Altronics
P1021	Pin To Socket 30 Way Prototyping Ribbon Strips 175mm	1	$5.20	$5.20	Altronics
P1023	Socket To Socket 30 Way Prototyping Ribbon Strips 150mm	1	$5.20	$5.20	Altronics
ADDITIONAL WIRING					
P0622	2.1mm Metal Chassis Mount DC Power Socket	2	$4.50	$9.00	Altronics
W2176	18AWG Black Double Insulated Speaker Cable	1	$1.40	$1.40	Altronics
P6717	0.5m 2.1mm DC Plug to 2.1mm DC Plug Cable	1	$8.00	$8.00	Altronics
W2712	9 Core Shielded Data Cable	3	$2.25	$6.75	Altronics
			TOTAL	**$1,042.28**	

Autonomous Ride-on Car

Ver. 1.3, May 2026

A STEM Project for High Schools

roblab.org/rideon

UWA Robotics Club
UWA Robotics & Automation Lab

Robotics is a fantastic research area and an ideal STEM project for high schools. As robot platforms are very expensive and model cars are rather limited, we decided to develop a robotics platform based on a kids' ride-on car. These cars already have drive-by-wire built-in and are fully remote controlled, so the only hardware interfacing required is to modify the remote transmitter and to add a remote emergency stop for safety reasons. The result is an open-ended autonomous vehicle project for high schools. The information given in this document can be freely distributed and we hope we will get a number of high schools to participate and build their own autonomous ride-on car (ARC), maybe even set up a competition.

The goal for each team is to let the car drive along a sequence of given GPS way points.
We decided to not include too much detail on application software, because we want high school teams to develop their own programs rather than copy our solution.

Please feel free to contact us for any questions (see addresses at the end of this document) and, of course, send us some videos of your builds. We hope this project will create an interest to study Automation & Robotics Engineering at UWA:
https://www.uwa.edu.au/study/courses/automation-and-robotics-engineering

Bill of Material – *Your Shopping List*

(see separate file for easier access to web links)

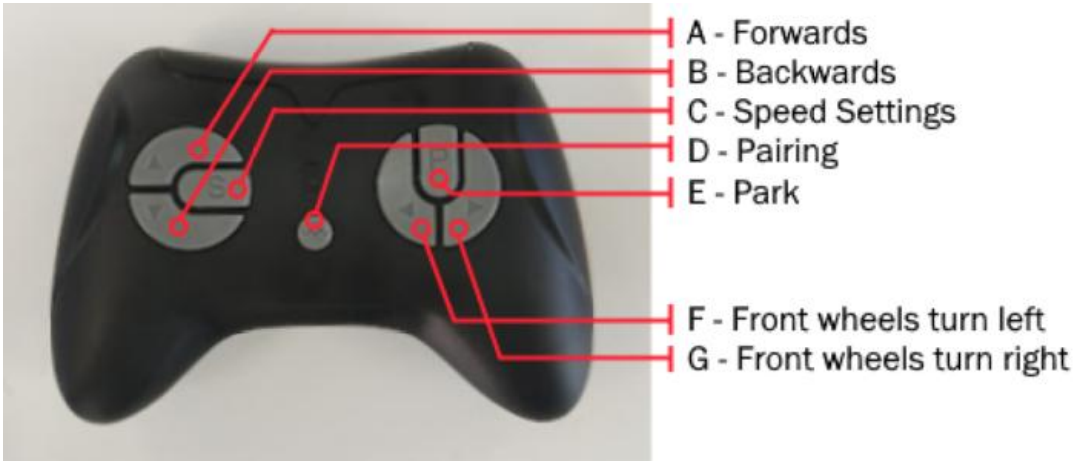
PART CODE	DESCRIPTION	QUANTITY	UNIT COST	TOTAL	SUPPLIER	USED FOR?
RIDE ON CAR	Beach Buggy Dune, 400W	1	$699.00	$699.00	ArtInToys	Car
aditional remote transmitter	additional transmitter for integrstion with embedded controller	1		$0.00	ArtInToys	Car
CONTROL BOX COMPONENTS						
H0330	186Lx146Wx75Hmm IP65 Sealed ABS Enclosure	1	$33.25	$33.25	Altronics	Enclosure for electronics
AA1017A	FOB43301L PentaFOB Single Large Button 433Mhz Remote	1	$45.00	$45.00	Altronics	Emergency stop switch
A1025A	PCR43301RE Penta Series 433Mhz 1 Channel Receiver	1	$69.95	$69.95	Altronics	Emergency stop switch
T-Display-S3	LILYGO T-Display-S3 ESP32-S3 Development Board	1	$23.00	$23.00	Amazon	Controller
NEO-6M	NEO-6M GPS Breakout Module with Antenna	1	$17.95	$17.95	Altronics	GPS
DF-SEN0253	Gravity BNO055+BMP280 intelligent 10DOF AHRS IMU	1	$21.50	$21.50	Core Electronics	IMU
DFR0379	20W Adjustable DC-DC Buck Converter with Digital Display	1	$8.55	$8.55	Core Electronics	Step down voltage
GP2Y0A21YK0F	GP2Y0A21YK0F Analog Distance Sensor 10-80cm	3	$14.41	$43.23	DigiKey	Distance Sensor
P9407	7 Pin 5A Locking Female Line IP67 Waterproof Socket	1	$26.50	$26.50	Altronics	Mounting Electronics
P9467A	7 Pin 5A Locking Male Chassis IP67 Waterproof Plug	1	$18.50	$18.50	Altronics	Mounting Electronics
JST SH 4-pin	STEMMA QT / Qwiic JST SH 4-pin to Premium Male	1	$0.95	$0.95		Wiring
CONTROLLER						
Z1042	NPN BC548B T092h General Purpose Transistor	7	$0.35	$2.45	Altronics	Controller Modification
R7046	1k 0.25W 5% Carbon Film Resistor PK 10	1	$0.60	$0.60	Altronics	Controller Modification
P1021	Pin To Socket 30 Way Prototyping Ribbon Strips 175mm	1	$5.20	$5.20	Altronics	Controller Modification
P1023	Socket To Socket 30 Way Prototyping Ribbon Strips 150mm	1	$5.20	$5.20	Altronics	Controller Modification
ADDITIONAL WIRING						
P0622	2.1mm Metal Chassis Mount DC Power Socket	2	$4.50	$9.00	Altronics	Car Wiring
W2176	18AWG Black Double Insulated Speaker Cable	1	$1.40	$1.40	Altronics	Car Wiring
P6717	0.5m 2.1mm DC Plug to 2.1mm DC Plug Cable	1	$8.00	$8.00	Altronics	Car Wiring
W2712	9 Core Shielded Data Cable	3	$2.25	$6.75	Altronics	Electronics Cable
		TOTAL		**$1,039.23**		

Link
https://www.artintoys.com.au/product/400-w-24-v-beach-buggy-dune-kids-ride-on-car-white/
Given to us free of charge from ArtInToys
https://www.altronics.com.au/p/h0330-ritec-186lx146wx75hmm-ip65-sealed-abs-enclosure/
https://www.altronics.com.au/p/aa1017a-elsema-pentafob-fob43301l-433mhz-single-large-button-remote-control/
https://www.altronics.com.au/p/a1025a-elsema-penta-pcr43301re-433mhz-1-channel-receiver/
https://www.amazon.com.au/LILYGO-T-Display-S3-ESP32-S3-Development-Soldering/dp/B09J112YR7?th=1
https://www.altronics.com.au/p/z6318-neo-6m-gps-breakout-module-with-antenna/
https://core-electronics.com.au/10-dof-mems-imu-sensor.html?
https://core-electronics.com.au/20w-adjustable-dc-dc-buck-converter-with-digital-display.html?
https://www.digikey.com.au/en/products/detail/sharp-socle-technology/GP2Y0A21YK0F/720159?
https://www.altronics.com.au/p/p9487-amphenol-ltw-7-pin-5a-locking-female-line-ip67-waterproof-socket/
https://www.altronics.com.au/p/p9467a-amphenol-ltw-7-pin-5a-locking-male-chassis-ip67-waterproof-plug/
https://www.adafruit.com/product/4209
https://www.altronics.com.au/p/z1042-npn-bc548-t092h-general-purpose-transistor/
https://www.altronics.com.au/p/r7048-1k5-0.25w-carbon-film-resistor-pk-10/
https://www.altronics.com.au/p/p1021-pin-to-socket-30-way-prototyping-ribbon-strips/
https://www.altronics.com.au/p/p1023-socket-to-socket-30-way-prototyping-ribbon-strips/
https://www.altronics.com.au/p/p0622-2.1mm-female-metal-chassis-mount-dc-power-socket/
https://www.altronics.com.au/p/w2176-24-0.2-black-double-insulated-figure-8-cable/
https://www.altronics.com.au/p/p6717-0.5m-2.1mm-dc-plug-to-2.1mm-dc-plug-cable/
www.altronics.com.au/p/w2712-9-core-shielded-data-cable/

TOOLS + CONSUMABLES NEEDED	WHERE TO GET	COST
Soldering Iron	Altronics	$20.00
Leaded Solder	Altronics	$2.70
Hot Glue Gun	Altronics	$14.50
Hot Glue Sticks	Altronics	$5.65
Drill	Bunnings	$50.00
Screw Drivers	Altronics/Bunnings	$15.00
M3 Screws	Bunnings	$3.00
M3 Metal Spacers	JayCar/Bunnings	$14.00
Heatshrink	Altronics	$10.00
	TOTAL:	**$134.85**

Remote Controller

- The controller has 7 buttons to correspond to different functions of the ride-on car, as shown below.

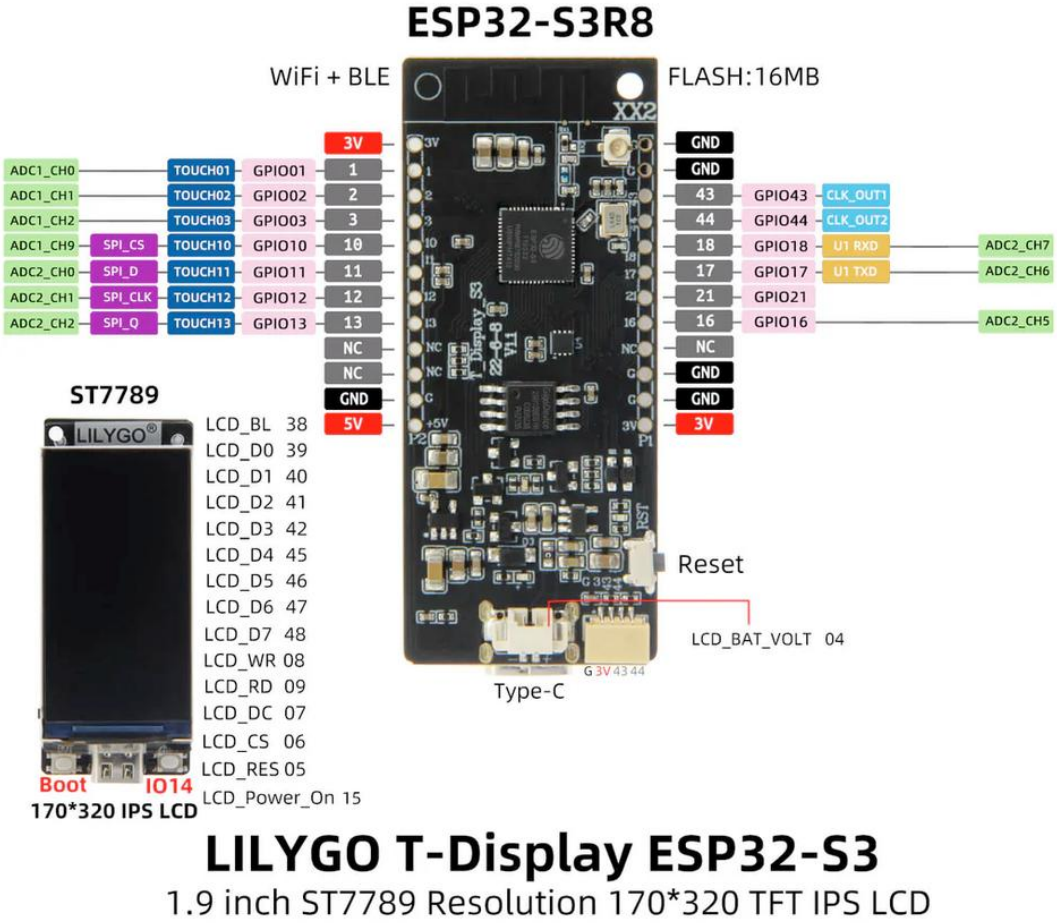


- NOTE: There are three LEDs on the controller which indicate information about the car.
- When changing the speed setting, 1 lit LED will indicate low speed, up to a maximum of 3 lit LEDs to indicate high speed.
- NOTE: To pair the remote with the car, the pairing button must be held down for 3 seconds, after which the controller LED begins to flash on and off. You must then power on the car to complete pairing. Once pairing completes, the LED will stop flashing.
- NOTE: If all three LEDs are flashing, the car is in parking mode.
- NOTE: The LEDs will “sleep” if no buttons are pressed in the last 5 or so seconds. This does not mean the pairing connection has been broken.
- NOTE: If attempting to hold down two buttons that act in opposite directions (such as pressing forwards and backwards), the remote will stop working for 3 seconds.

- The controller’s circuit board is rated for ~3V. This should be supplied by the ESP32’s 3v3 pin, which supplies 3.3V.

ESP32 LILYGO T-Displasy-S3

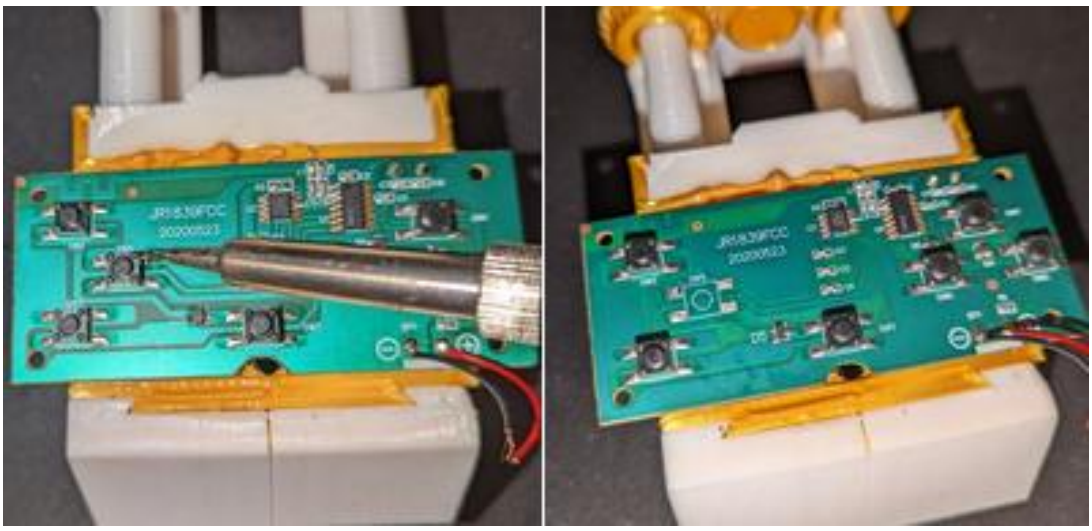
- This is a useful website to determine which output pins are suitable on the ESP32: https://github.com/Xinyuan-LilyGO/T-Display-S3
- Using GPIO pins to control the remote is recommended.
- NOTE: Each GPIO Pin may correspond to the same Wiring Pin number. The Wiring Pin number is what is used in the code, when any digitalWrite calls are made.
- NOTE: GPIO 1 – 13 are the only ADC meaning any analogue sensors must be connected to those.



Implementation Steps

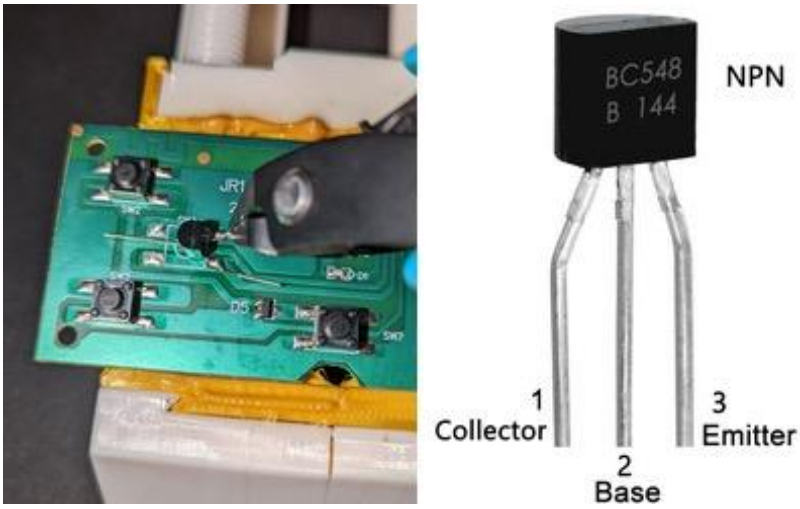
Remote Control Mod

- Take out the circuit board from the remote controller. De-solder each of the 7 push buttons from the circuit board.

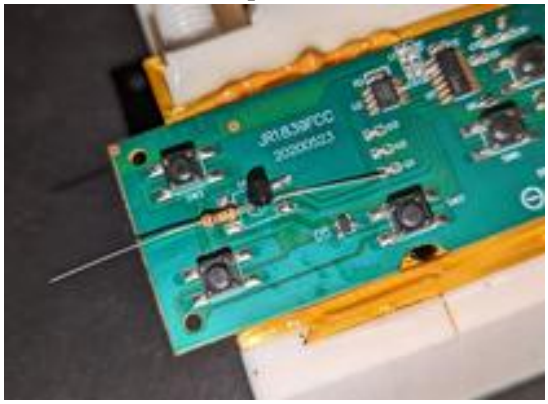


- Solder BC548 transistors in place of the buttons. Connect the emitter pin to a ground plate, and the collector pin to live.

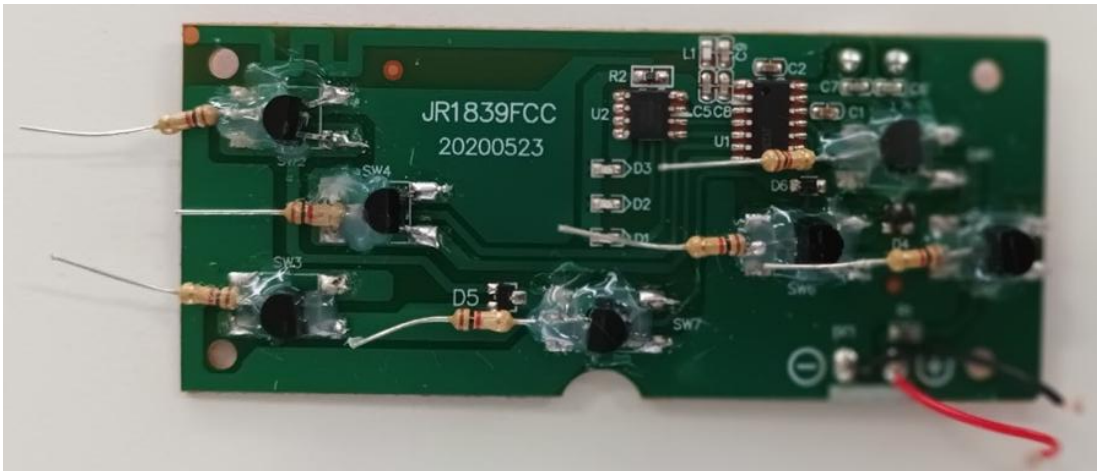
  NOTE: The light green sections of the circuit board are electrically connected. This can be used to find the ground side of the push buttons. The ground plates will be electrically connected to the section with a negative ground symbol. Similarly, the live plates are electrically connected to where the live symbol is.



- On the base pin of the transistors, solder a 1KΩ resistor, making sure that the base pin does not touch the circuit board. Clip off the excess wire on the resistor if it is too long.



- Use hot glue to stabilize the transistors and prevent the resistor pins from touching the circuit board.

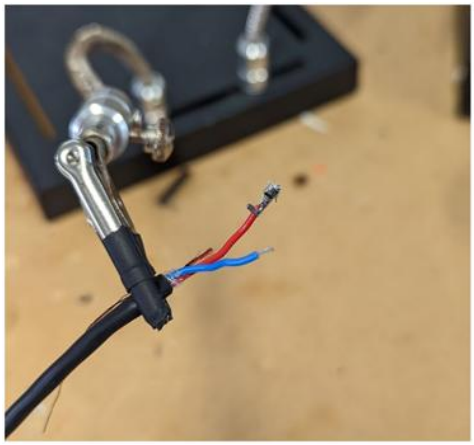
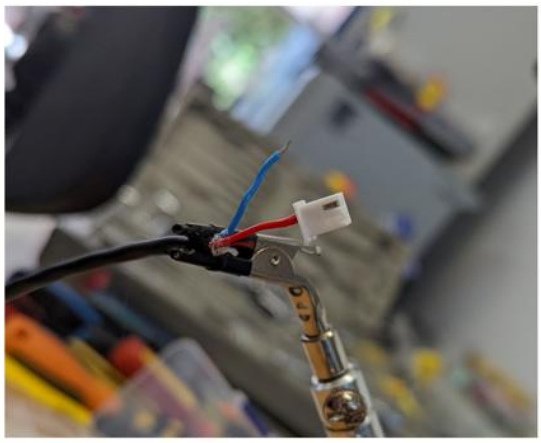


- On the circuit board, solder male/male jumper cables to the ground and positive terminals. Power can be supplied to the board using female/female jumper cables connected to the ESP32.
- Use female/female jumper cables to connect the resistors to different output pins on the ESP32. See the provided Table to see where each pin should be connected. Each pin can be used to simulate a button press on the remote controller. Now the car can be controlled remotely by uploading code to the ESP32.
  IMPORTANT Completing this step allows early testing of programs and making sure that the wiring is correct. Later you will want to solder male/female jumper cables to the resistors instead of using the temporary female/female cables.

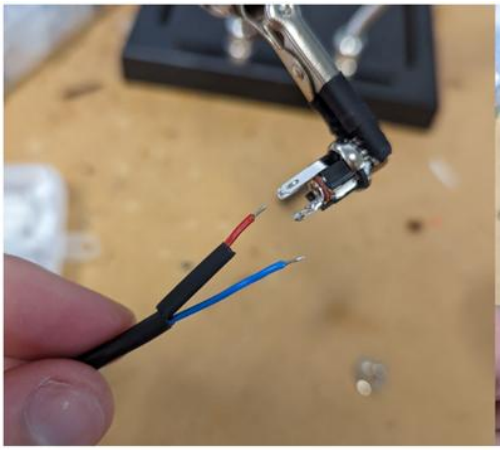
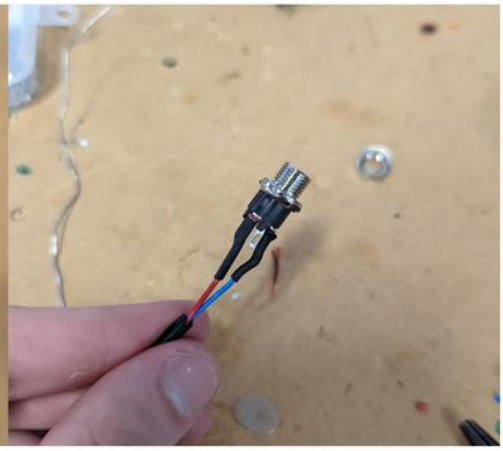
Additional Soldering/Electronics

- To connect the car power to the control box we will make a couple of cables.
- Firstly, take dual core wire and strip a couple centimeters off each side like shown below. Then crimp a JST Crimp Pin to each wire. Insert the pins into the JST housing, make sure the positive side

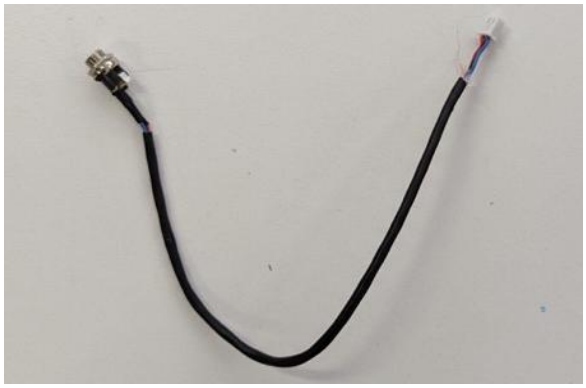
is connected as shown below.

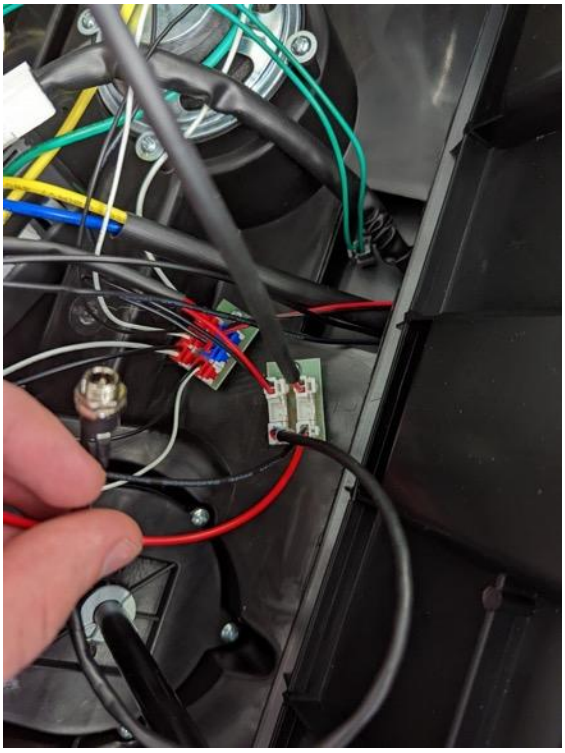
- On the other side of the cable, we are going to solder a 2.1mm chassis mount socket that will provide power to the outside of the car. Solder the positive wire to the centre pin of the connector and the negative wire to the outer pin. Use some heat shrink to cover the connections.

- The final cable is shown below.



- Find the bus bar with JST sockets inside the hood of the car shown below, this is connected to the cars 24V battery. Plug in the JST connect of the cable you made and drill a hole in the foot well of the car for screwing in the 2.1mm socket. This will allow the 24V power to be accessed outside the hood of the car.



Setting up the E-Stop Receiver

- Unscrew the top off the PentaFob Receiver

- Follow the instructions inside to set the FOB into LATCHING mode.

- When powered with 24V the FOB key will allow the car to be stopped wirelessly.

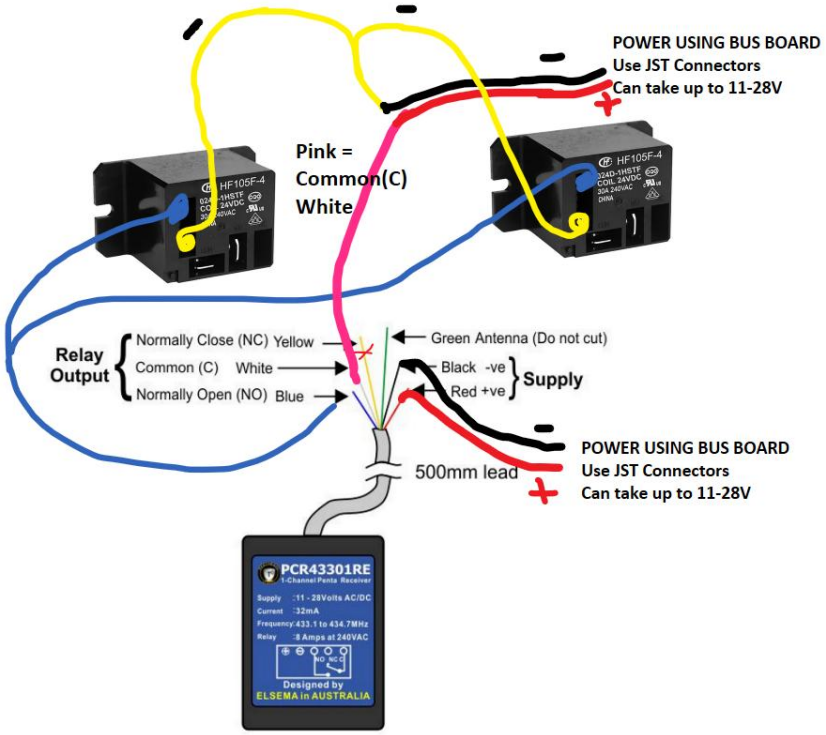
Wiring up the Emergency Stop:

Parts you need:

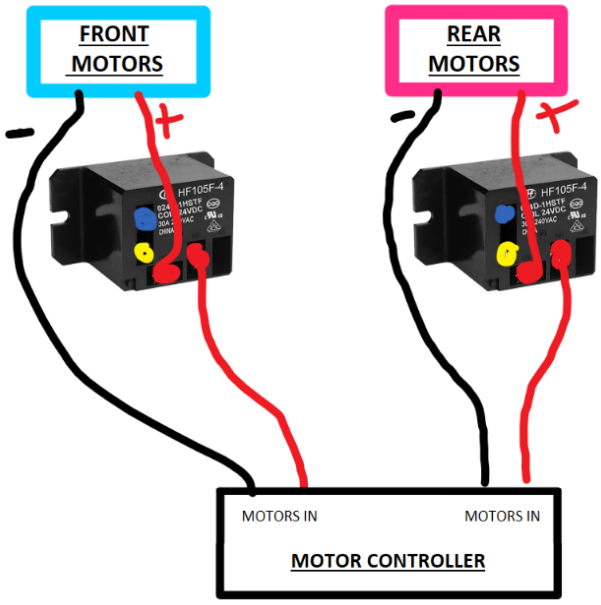
- E-Stop and Key fob (look at the BOM)
- Two Relays (look at the BOM)
- Car's Internal Motor Controller
- Cables and suitable crimps/connectors.

Step 1: Open and set the E-Stop to the correct mode, flick the switches according to the diagram to **LATCHING** mode. Find the Car's motor cables and motor controller and disconnect the motors from the controller.

Step 3: Wiring up the E-Stop to the power and Relays. (One relay for front two motors, and another for rear motors). Power for the E-Stop is from the car's bus board. Use crimp connectors and JST connectors to make these connections. [Check diagrams below and the pictures provided]



Step 4: Wiring up the Relays to the motor controller and motors. When wiring up the motors to the relay, please ensure that you use the cables that are suitable for the current rating of the motors. The relays act as a switch triggered by the E-stop, and they help stop the motors in case of an emergency. [Check diagrams below and the pictures provided]



Step 5: Testing and Usage.

You will need the key fob to activate the E-Stop.

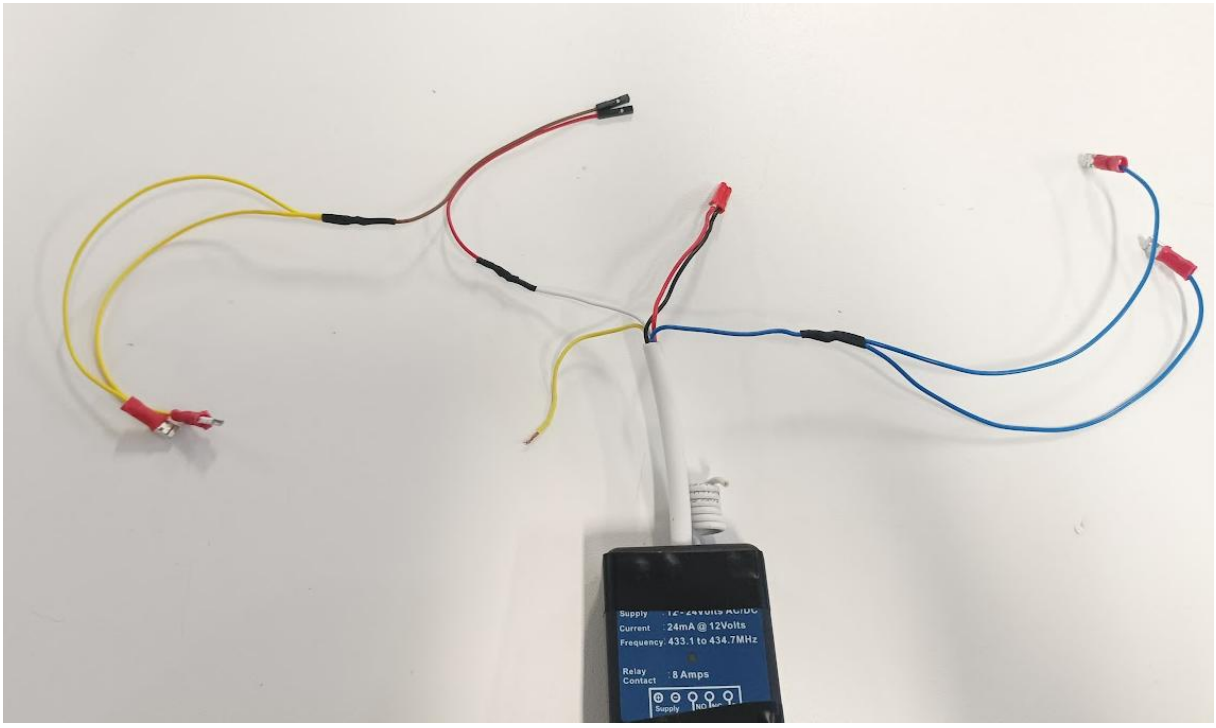
When the Car starts, you will have to press the fob to let the relays allow current flow to the motors. You should be able to hear an audible click which indicate that the relays are switching and the red light on the E-Stop. After the key press, the car should be able to move.

If an emergency arises while in operation, please press the fob key to activate the E-stop and trigger the relays which cut the power to the motors.

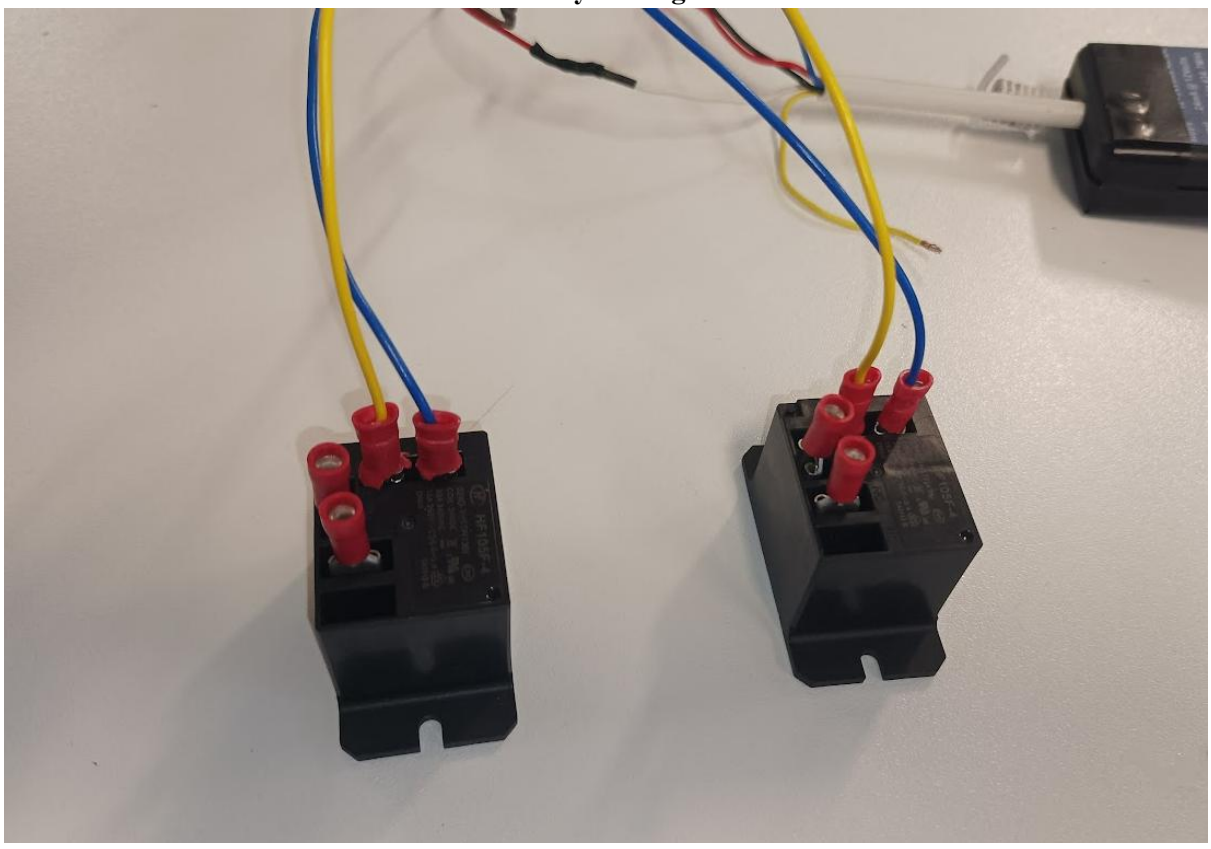
PLEASE NOTE: With this configuration, the E-Stop only cuts power to the motors. The remaining electronic hardware on the car is still active and powered.

Photos:

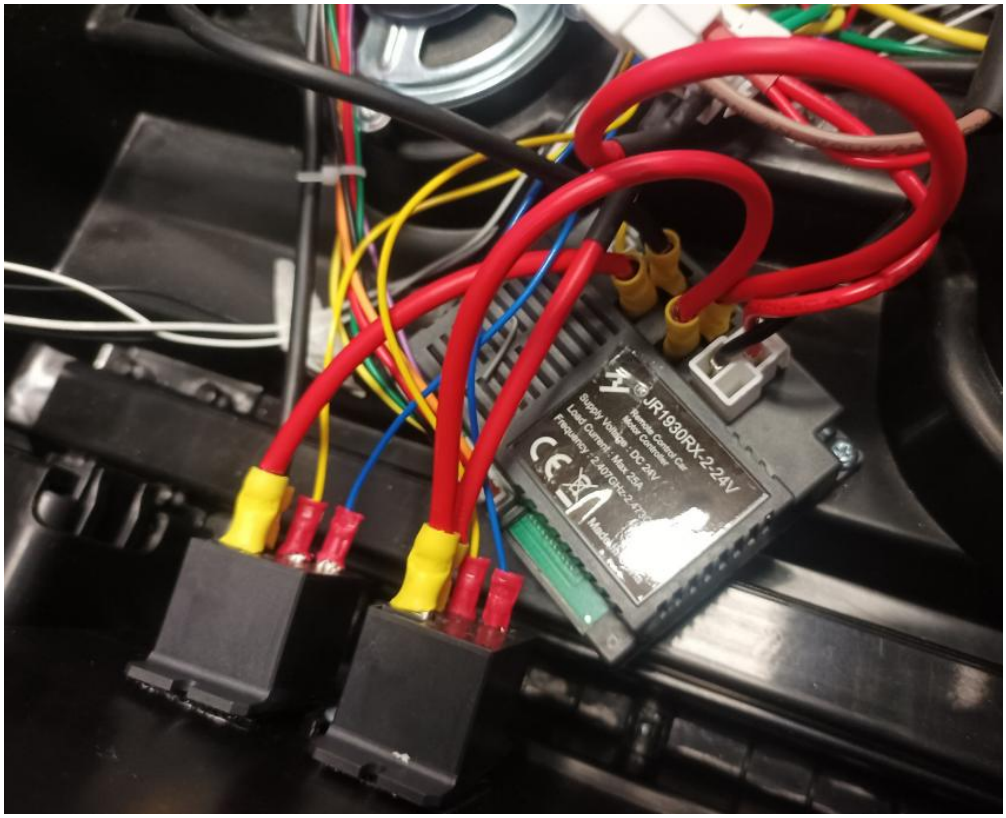
E-Stop Wiring:



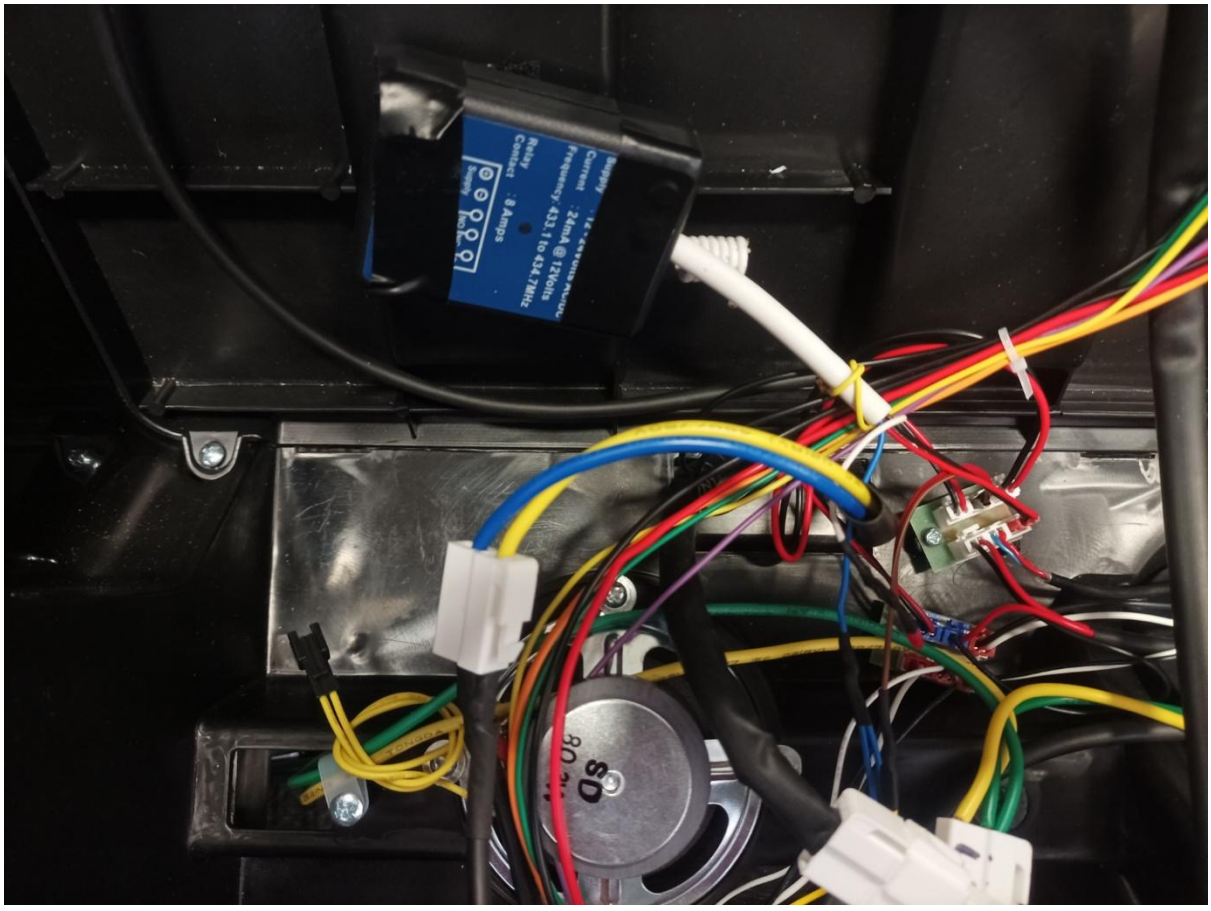
Relay Wiring:



Relay Wiring to Motor Controller and Mounting of Relays:



E-Stop Mounting:



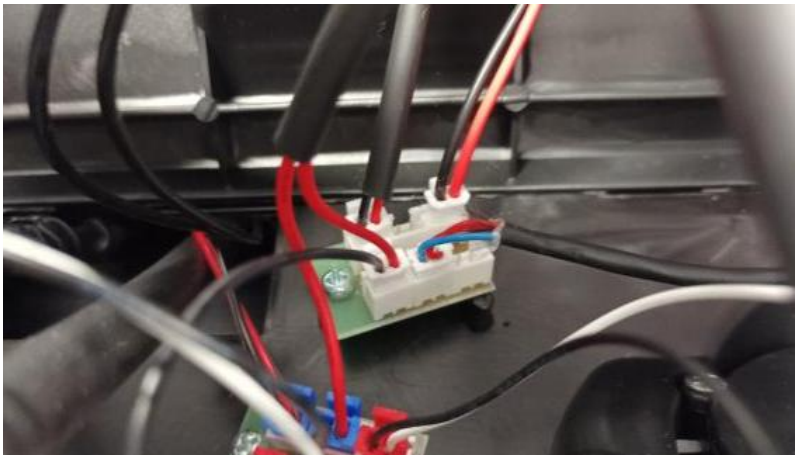
Final Modifications to the Car

- Unscrew the car's dashboard and pry out the clips on the hood to expose the internal wiring. To unclip the plastic hood, it may be easier to rotate the car onto its side.

- Disconnect the cable leading to the steering wheel horn. Disconnect the speaker cables to prevent loud audio playing from the speakers.

- You can now reach the interior wall of the car footrest. Drill a hole suitable for the 5.5mm socket such that it is accessible from the exterior side of the footrest.

- Locate the 24V JST connector from inside the car, which looks like this:

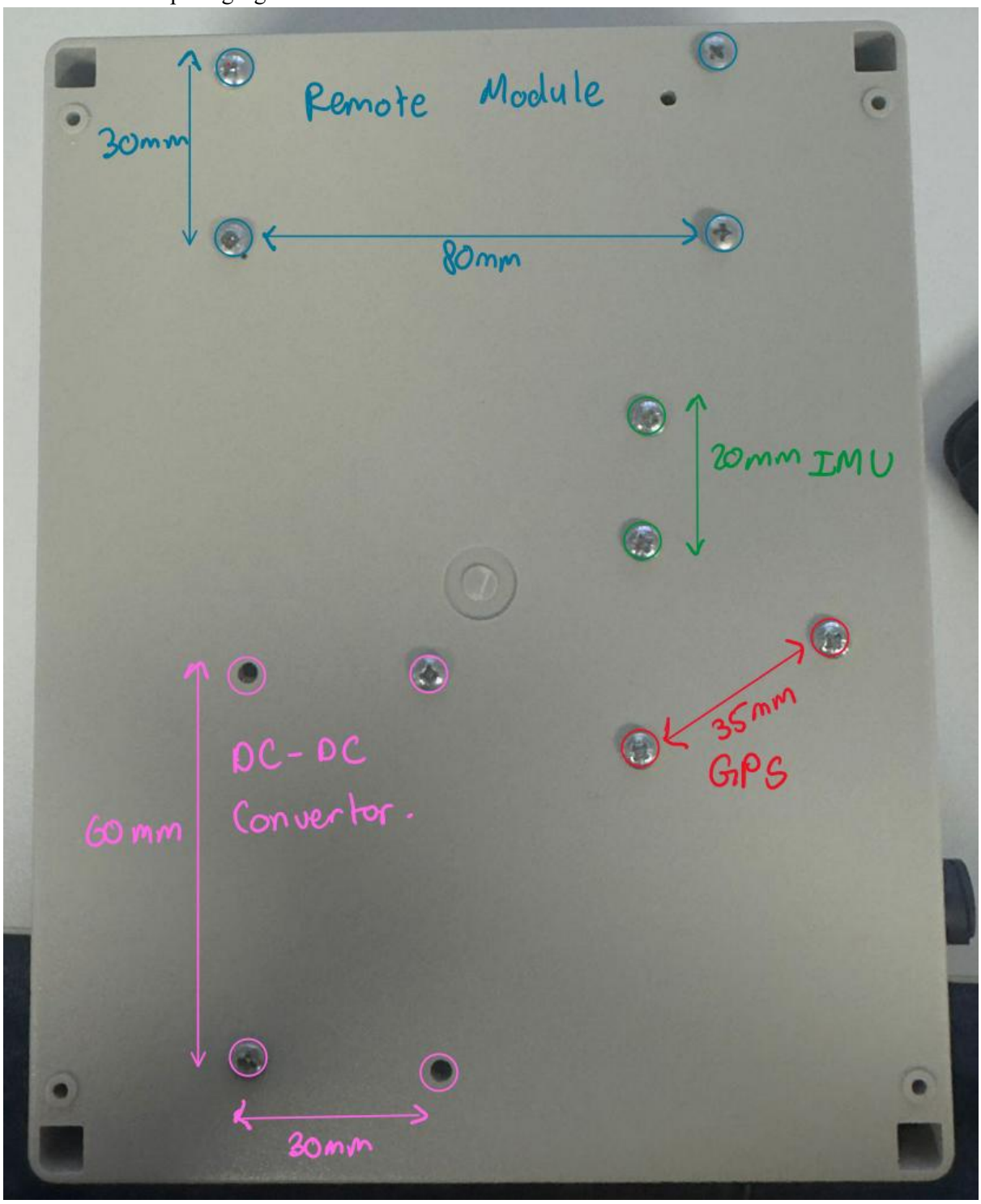


- Attach the modified JST cable and fix the soldered 2.1mm power socket through a hole in the footrest.

- From the inside of the car, you can now connect the electronics box and the power socket using the 2.1mm barrel plug cable.
- The dashboard and hood can now be screwed back in.
- To explain this procedure: we are routing the 24V supply from the car battery to the power socket fixed into the footrest. This is connected to the 24-5V converter which provides the 5V required for the ESP32. The electronics box can now be left behind the seat with the power cable routed on the side next to the seats.

Housing Electronics

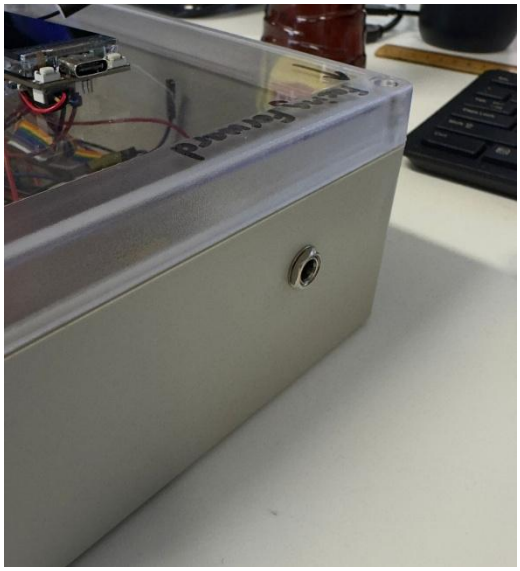
Screw Hole and spacing figure

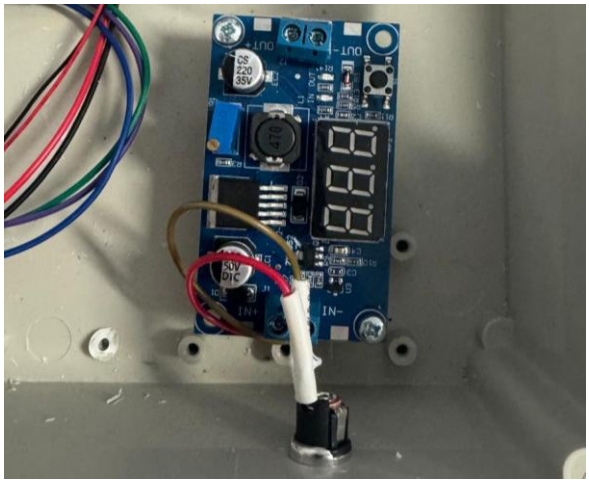


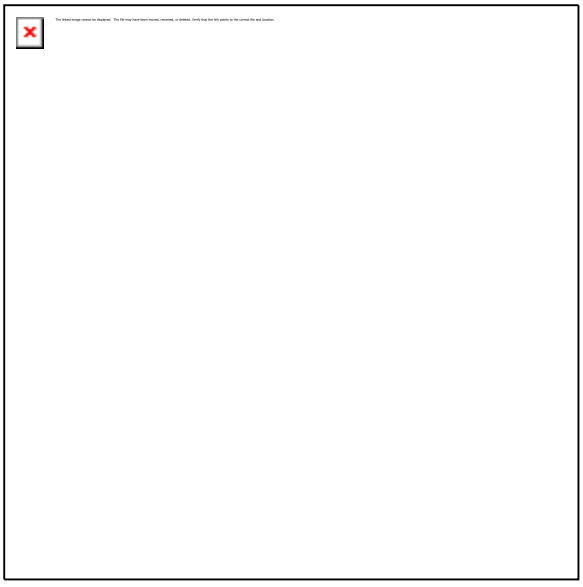
- Drill the appropriate holes using a 3.0mm drill bit into the control box base as per the figure shown above. This is to secure the electrical components such as the GPS, IMU, remote module and the DC- DC convertor to the box. Once drilled place a M3 screw from the bottom of the box and screw a M3 spacer from the inside, mounting the spacer to the floor of the control box, do that for the rest of the holes.

- Once the spacers are mounted, place the components as mentioned in the Figure above on top of the spacers aligning the mounting holes of the component with the spacer hole and screw them in using M3 screws

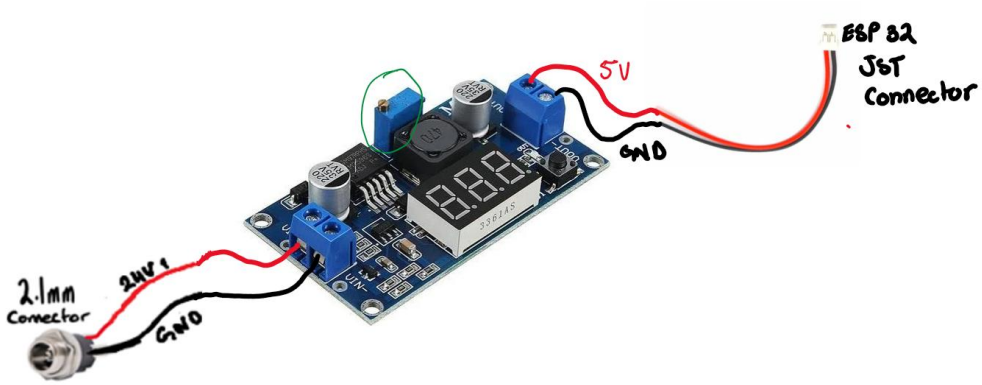
- Drill a hole into the wall suitable for the 2.1mm power socket and screw the socket as shown in the Figure below.



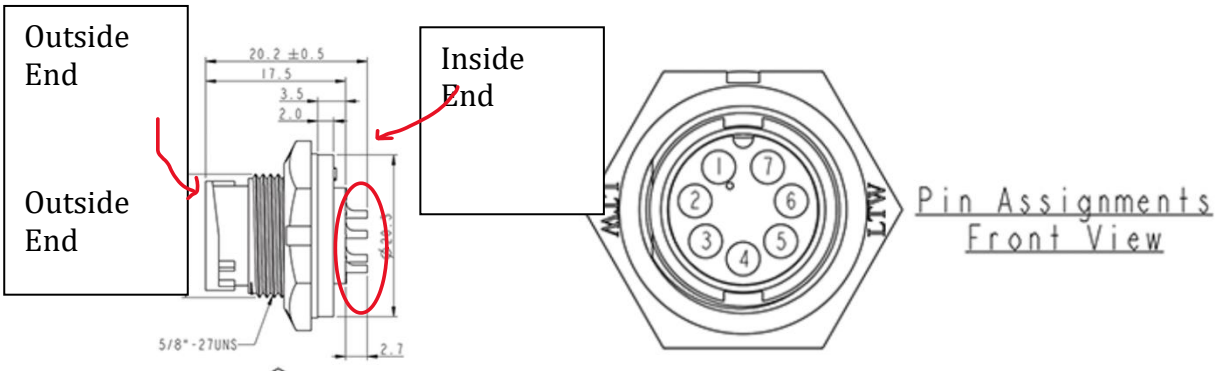




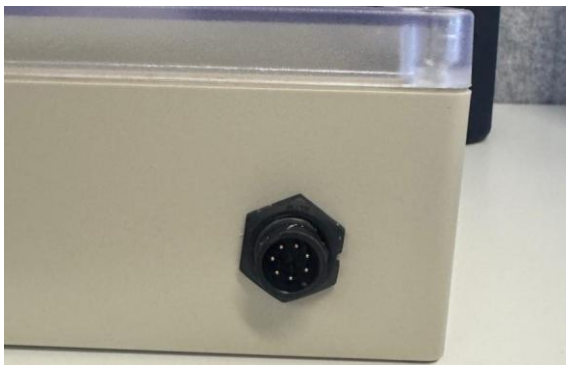
- The wiring diagram for the power is shown below. The 24V and GND wire coming from the 2.1 barrel connector is to be placed inside the DC-DC convertor's VIN+ and VIN- respectively, once placed power the module and adjust the voltage using the potentiometer (circled green in the Figure) until the output voltage reaches 5V as seen on the LCD screen. Then connect the ESP32's JST power connector to the convertor's VOUT terminal block as seen in the Figure.

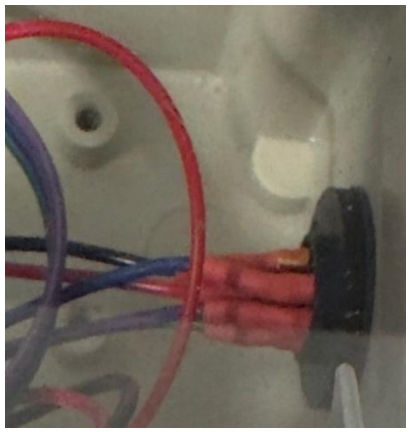


- Using female Dupont cables of different colours (red, black, green, purple, blue), cut and splice the end of one side of the cables.
-We then will be working on the 7 Pin 5A Locking Male Chassis IP67 Waterproof Plug, see images below of what they look like.
-The soldering end of the pins are the one circles in red in the below picture, we can classify that as the inside end of the connector
-Solder the splice end of the red cable to pin 1, black cable to pin 7, and the various colors green, purple, blue to pins 3,4,5 respectively. After soldering heat shrink each cable individually to prevent any short circuiting

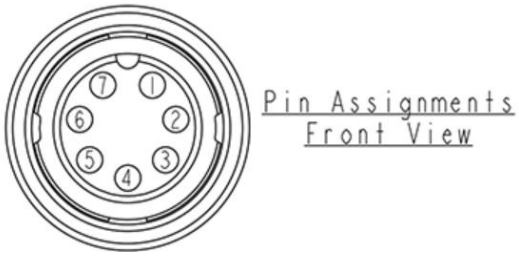


-Drill a large hole of size approximately 21mm on the side of the control box for the 7 Pin 5A Locking Male Chassis IP67 Waterproof Plug and mount it with the inside end facing inside the control box and outside end of the connector facing outside the control box as seen in figure below





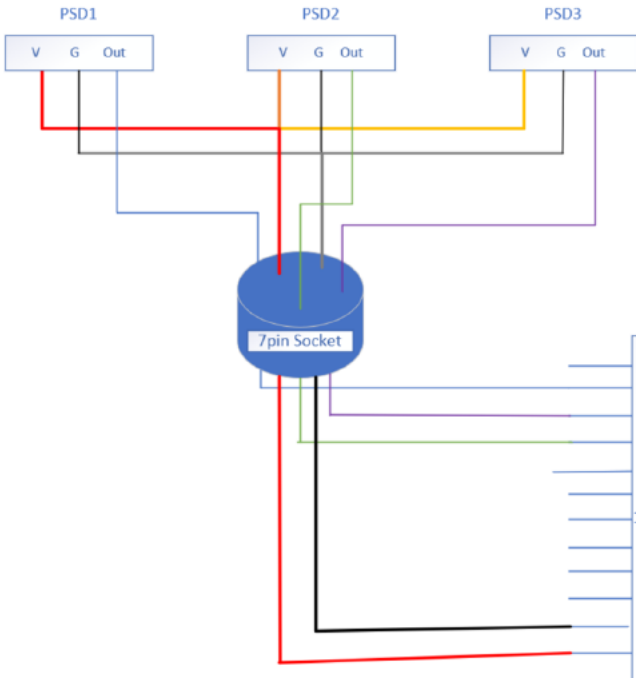
- Now we will be working on the 7 Pin 5A Screw-On Female Line IP67 Waterproof Socket & the 9 Core Shielded Data Cable.
- Using the 9 core shield data cable, identify the colours (red, black, green, purple and blue inside the cable), the rest of the 4 cables will not be used and can be cut off or taped off.
- Using the Pin assignment listed below, solder one end of the 9-core shield data cables each useful wire accordingly to the 7 Pin 5A Screw-On Female Line IP67 Waterproof Socket, make sure to



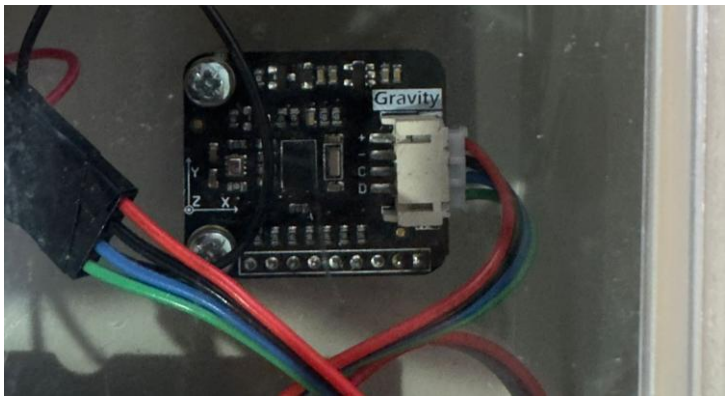
solder each colour to each pin as it was done to the male socket,

- Note that the male pin assignment and female pin assignment are not the same and are a mirror of each other, critically thinking why this is important, visualise how each pin will align with the other end, hence it is important to keep the red cable to pin 1, black cable to pin 7, and the various colours green, purple, blue to pins 3,4,5 respectively. After soldering heat shrink each cable individually to prevent any short circuiting
- Now we will be working on the other end of the 9-core shield cable and attach it to the 3 PSD sensors.
- Take a PSD sensor and cut and splice the end of the cable so there is exposed wire for power, ground and signal.
- Solder the signal wire of the PSD to one of the various 3 colours, then repeat this for the other 2 PSD for the other 2 colours left
- Then take a spare red wire (for exentsion) and splice both ends for exposed wire, perform a 4 way solder between each red wire of the PSD and one end of the extension wire, then solder the other end of the extension wire to the red wire coming from the 9-core shield cable
- Perform the same step as above for the black (ground wire)

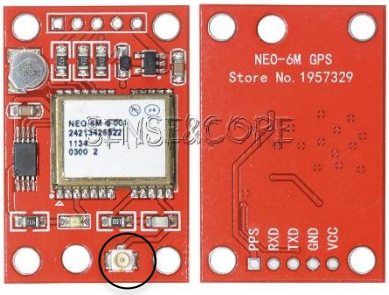
-

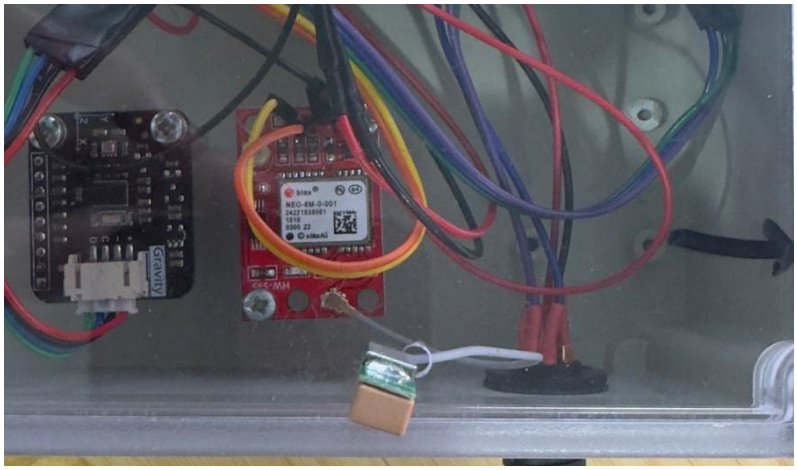

PSD1
V G Out
PSD2
V G Out
PSD3
V G Out
7pin Socket

-Attach the provided IMU cable (Gravity: 4Pin I2C/UART Sensor Cable for Arduino) to the Gravity BBNO IMU as shown below

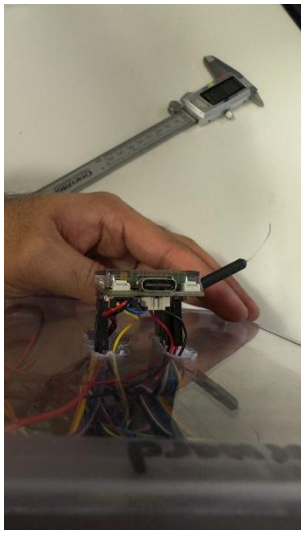


-Now take 4 Male to Female Dupont Cables in the colours of red, black, yellow and orange. Using the image below plug the male end of the red cable into the GPS's VCC solder port, the black cable into GND solder port, orange cable into the TXD solder port and the yellow cable into the RXD solder port.
-Additionally drill a small hole of size 3mm into the lid of the control box exactly above the antenna port circles in the picture and route the antenna connector through the hole and plug it into the antenna port
The resulting image should look like this:

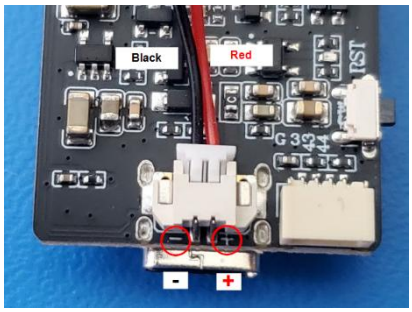




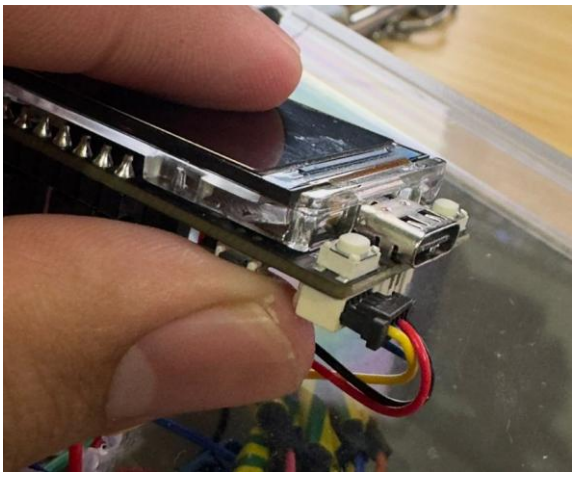
- On the lid either using a drill or a Dremel create 2 pockets for wiring to be routed and the ESP32's pins to be routed towards the inside of the box. This will allow the ESP32 to sit flush on the lid whilst making its pins accessible towards the inside of the box for connectivity to components and power.



-Connect the Esp32 power JST connector coming from the DC-DC convertor like shown below



-Use the 4pin i2C JST connector and connect it to the bottom of the ESP32 as shown below

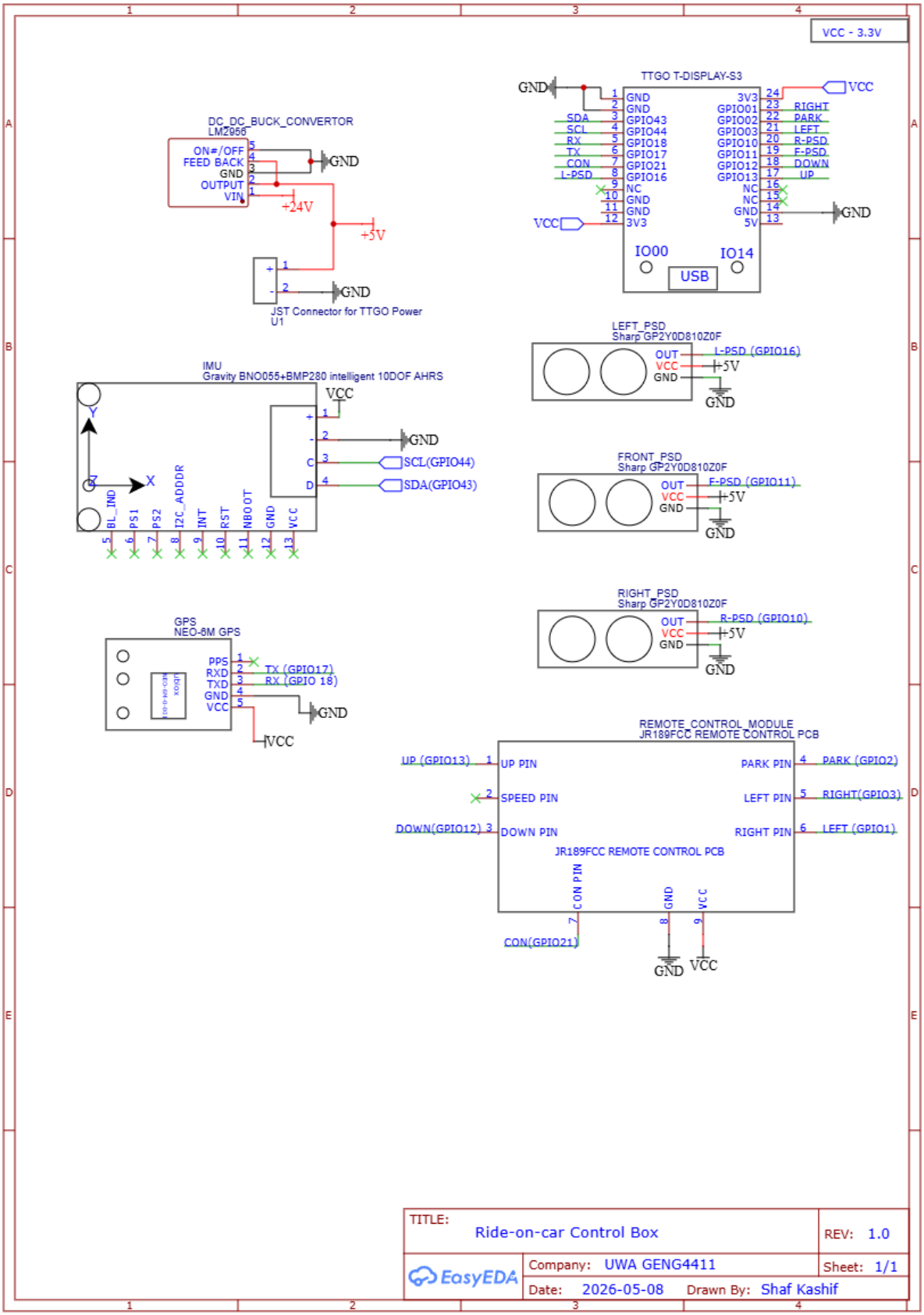


Now its time to connect all the components to the ESP32. Using the Table below connect each component to the right pin.

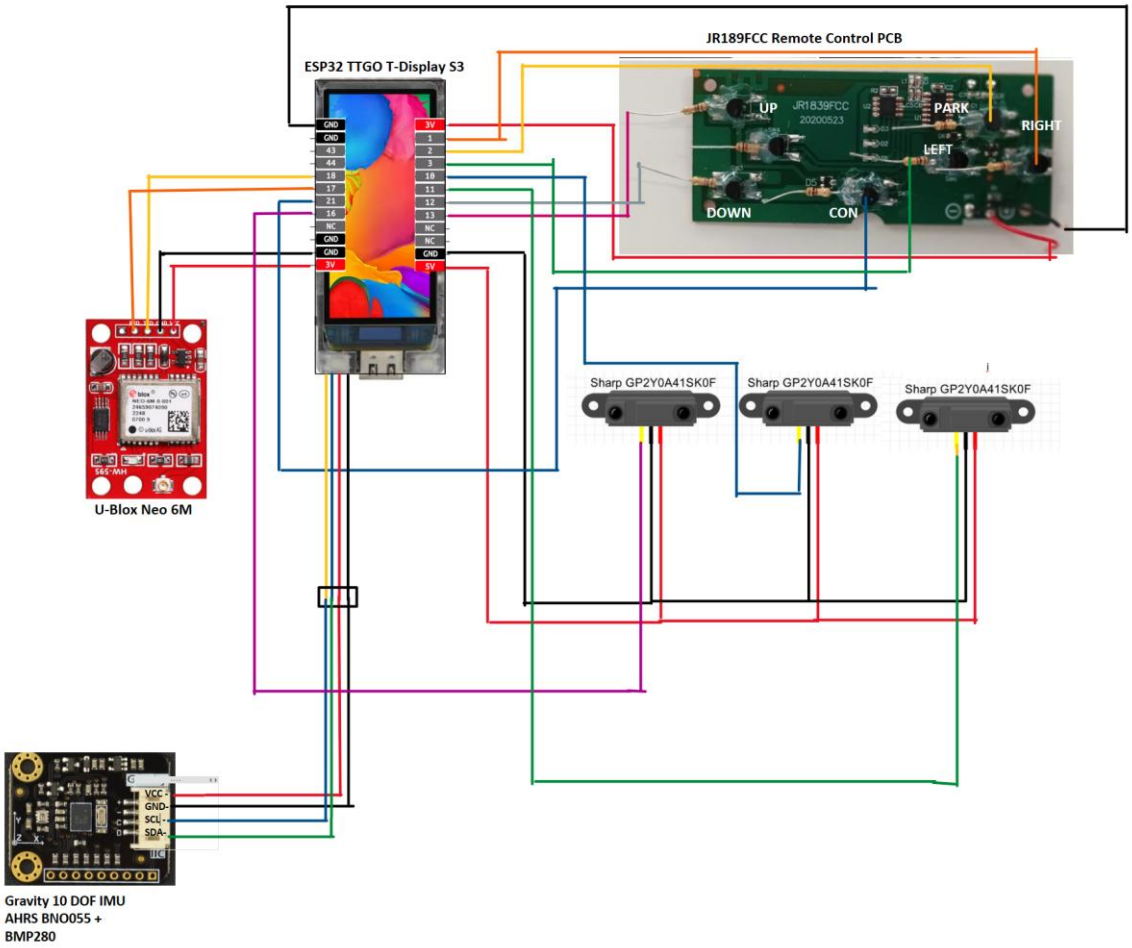
-Note for the IMU, the i2c cable that comes with IMU uses red for VCC, black for GND, blue for SCL and green for SDA, additonallu the i2c JST cable for the ESP32 uses red for VCC and black for ground, however uses blue for SDA and yellow for SCL.
-Note for the UART GPS connection, the RXD on GPS is to be connected to the TX on the ESP32 (orange wire) and the TXD on the GPS is supposed to be connected to the RX on the ESP32 (Yellow wire)

TTGO T-Display S3 Pin	**Colour / where the wire is coming from**	**Peripheral Pin/Description**
43	*Coming from the i2c blue cable*	*IMU Sensor SDA (Green cable from i2c)*
44	*Coming from the i2c yellow cable*	*IMU Sensor SCL (Blue cable from i2c)*
18	*Yellow cable coming from GPS*	*GPS UART RX*
17	*Orange cable coming from GPS*	*GPS UART TX*
16	*Green cable coming from 9-core shield cable*	*Left PSD*
10	*Blue cable coming from 9-core shield cable*	*Right PSD*
11	*Purple cable coming from 9-core shield cable*	*Front PSD*
12	*Grey wire coming from remote*	*Remote Down PIN (Backwards)*
13	*Purple wire coming from remote*	*Remote Up PIN (Forward)*
1	*Orange wire coming from remote*	*Remote Right PIN*
2	*Yellow wire coming from remote*	*Remote Park PIN*
3	*Green wire coming from remote*	*Remote Left PIN*
21	*Blue wire coming from remote*	*Remote Connection PIN*

- Connect the GPS and Remote control red wire (VCC) to the 3.3V on the ESP32 using the female dupont cables attached to the components
- The IMU VCC cable is already supplied with 3.3V through the 2 i2c cables and the grounds are attached.
- The PSD sensors and thus the 7 pin connector that has the female dupont cables attached, the red wire should be connected to 5V on the esp32
- All grounds of the components can be connected to any grounds on the esp32
- The wiring diagram can be seen below, additionally a simplified colour version of the diagrams can be seen below

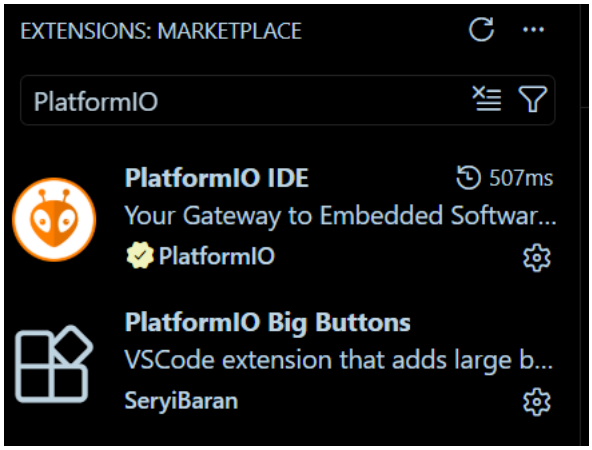

VCC - 3.3V
DC_DC_BUCK_CONVERTOR
LM2956
ON#/OFF
FEED BACK
GND
OUTPUT
VIN
+24V
+5V
GND
JST Connector for TTGO Power
U1
TTGO T-DISPLAY-S3
GND
VCC
SDA
SCL
RX
TX
CON
L-PSD
GPIO43
GPIO44
GPIO18
GPIO17
GPIO21
GPIO16
3V3
GPIO01
GPIO02
GPIO03
GPIO10
GPIO11
GPIO12
GPIO13
RIGHT
PARK
LEFT
R-PSD
F-PSD
DOWN
UP
IO00
IO14
USB
IMU
Gravity BNO055+BMP280 intelligent 10DOF AHRS
SCL(GPIO44)
SDA(GPIO43)
BL_IND
PS1
PS2
I2C_ADDDR
INT
RST
NBOOT
LEFT_PSD
Sharp GP2Y0D810Z0F
OUT
L-PSD (GPIO16)
FRONT_PSD
F-PSD (GPIO11)
RIGHT_PSD
R-PSD (GPIO10)
GPS
NEO-6M GPS
PPS
RXD
TXD
TX (GPIO17)
RX (GPIO 18)
REMOTE_CONTROL_MODULE
JR189FCC REMOTE CONTROL PCB
UP (GPIO13)
UP PIN
SPEED PIN
DOWN(GPIO12)
DOWN PIN
PARK PIN
PARK (GPIO2)
LEFT PIN
RIGHT(GPIO3)
RIGHT PIN
LEFT (GPIO1)
CON PIN
CON(GPIO21)
TITLE:
Ride-on-car Control Box
REV: 1.0
EasyEDA
Company: UWA GENG4411
Sheet: 1/1
Date: 2026-05-08
Drawn By: Shaf Kashif

-

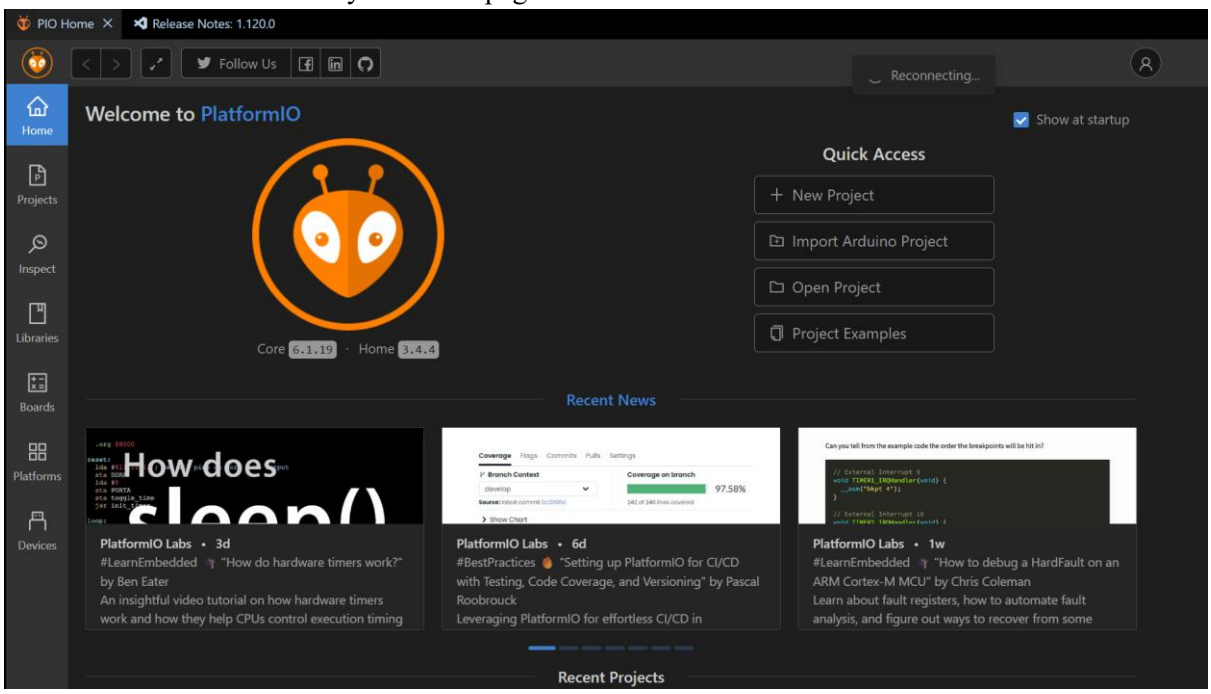

JR189FCC Remote Control PCB
ESP32 TTGO T-Display S3
UP
JR1839FCC
20200523
PARK
RIGHT
LEFT
DOWN
CON
Sharp GP2Y0A41SK0F
Sharp GP2Y0A41SK0F
Sharp GP2Y0A41SK0F
U-Blox Neo 6M
VCC
GND
SCL
SDA
Gravity 10 DOF IMU
AHRS BNO055 +
BMP280

Software Side

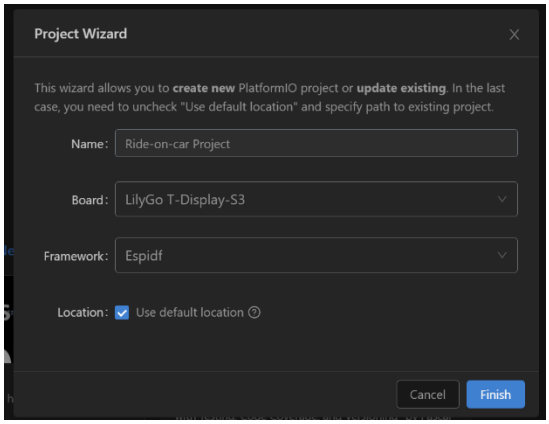
- Using a laptop or computer desktop, install VS Code.
- Once VS Code is installed, go to extensions and install PlatformIO IDE & PlatformIO big buttons



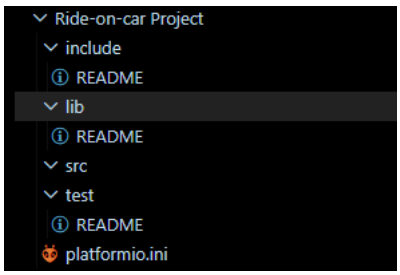
- Once installed it should land you on this page



- Click on the New Project Button and Enter the following details



- Once completed it should open up the workspace as follow:



- Click on the platformio.ini file and copy across the following

```
- [env:lilygo-t-display-s3]
- platform = espressif32
- board = lilygo-t-display-s3
- framework = arduino
- monitor_speed = 115200
- board_build.filesystem = littlefs
-
- lib_deps =
-   Bodmer/TFT_eSPI@^2.5.35
-   adafruit/Adafruit BNO055 @ ^1.6.4
-   adafruit/Adafruit Unified Sensor @ ^1.1.9
-   mikalhart/TinyGPSPlus@^1.0.3
-   https://github.com/me-no-dev/ESPAsyncWebServer.git
-   https://github.com/me-no-dev/AsyncTCP.git
-
-
- build_flags =
-   -DUSER_SETUP_LOADED=1
-   -DUSER_SETUP_FILE="TTGO_T_Display_S3.h"
-   -include
  $PROJECT_LIBDEPS_DIR/$PIOENV/TFT_eSPI/User_Setups/Setup206_LilyGo_T_Display_S3.h
-
-
- ; (Optional but handy)
- monitor_filters = esp32_exception_decoder, time
-
```

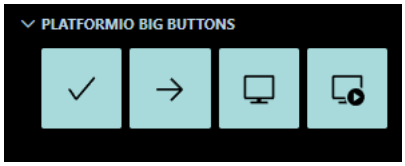
-Then click on the src folder and create a new file and save it as main.cpp
-This is the main file where you would do the primary coding
-Add the following libraries and pin allocation

```
include <ESPAsyncWebServer.h>
#include <Arduino.h>
#include <Wire.h>
#include <TFT_eSPI.h>
#include <SPI.h>
#include <TinyGPS++.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <math.h>

#define TFT_BL 38
TFT_eSPI tft = TFT_eSPI();
```

```
const int pinFrontPSD = 11;
const int pinLeftPSD  = 16;
const int pinRightPSD = 10;

// I2C pins (shared by BNO055)
const int I2C_SDA = 43;
const int I2C_SCL = 44;

// ---------------- Serial Pins for GPS ----------------
const int pinRX = 18; // RX of GPS
const int pinTX = 17; // TX of GPS

// ---------------- Controller Pins ----------------
const int pinLeft  = 3;   // Green wire
const int pinRight = 1;   // Orange wire
const int pinUp    = 13;  // White wire
const int pinDown  = 12;  // Grey wire
const int CON      = 21;  // Blue wire
const int parkPin  = 2;   // Yellow wire
//const int speedPin = 11;  // Purple wire (disabled for now)


// ---------------- GPS ----------------
#define GPS_BAUD 9600
TinyGPSPlus gps;
HardwareSerial gpsSerial(2);
```

-You are now ready to start coding the ride-on-car

-Once you have some code you want to upload click on the tick button to compile and
-> button to upload the code on to the ESP32, make sure you have a USB cable plugged from your computer on the usb c port on the ESP32



Contact

UWA Robotics Club

https://uwarobotics.com.au
club@uwarobotics.com.au
Jake Lorkin
Tiziano Wehrli
Won Chen Qin
Agnibho Gangopadhyay
Shaf Kashif

UWA Robotics & Automation Research Lab

https://roblab.org *or* https://robotics.ee.uwa.edu.au
thomas.braunl@ uwa.edu.au
Professor Thomas Bräunl