GENG4412/5512 Engineering Research Project

EyeBot-32: An ESP-32 powered robotic platform for educational purposes
Semester 2, 2025

Parham Bahrami

23426998

School of Engineering, University of Western Australia

Supervisor: Professor Thomas Bräunl

School of Engineering, University of Western Australia

Word Count: 6986

School of Engineering
University of Western Australia

Submitted: 13/10/2025

Project Summary

The EyeBot-32 project aimed to develop a next-generation educational robotics platform for use in the University of Western Australia's Embedded Systems (ELEC3020) laboratories. The existing EyeBot-8 platform, based on a Raspberry Pi controller, had become increasingly unreliable, costly, and unsuitable for modern laboratory use due to software difficulties, high power consumption, and maintenance complexity. To address these issues, the EyeBot-32 was designed from the ground up using the ESP32-S3 microcontroller as its core, offering a compact, power-efficient, and affordable alternative capable of supporting real-time control, multitasking, and image processing in a single integrated unit.

The project encompassed a complete hardware and software redesign. A custom printed circuit board (PCB) was developed to integrate motor drivers, encoder feedback channels, time-of-flight distance sensors, and a camera interface, while maintaining ease of assembly and classroom scalability. Pin budgeting and power analysis were carried out to optimise current distribution and ensure safe operation. Two protected 18650 lithium-ion cells in series were selected as the power source, providing 7.4 V for motor drive and 5 V logic regulation through a high-efficiency buck converter, achieving over three hours of continuous operation per charge.

On the software side, the RoBIOS32 library was written to replicate the familiar EyeBot API within the Arduino/PlatformIO environment, enabling seamless student transition from previous systems. A FreeRTOS-based dual-core architecture was implemented to separate user programs from time-critical control loops, ensuring reliable multitasking and deterministic real-time behaviour. A closed-loop PID motor control system with encoder feedback was developed for precise speed and distance control, alongside an updated image processing pipeline supporting both RGB565 and RGB888 formats for vision-based tasks.

Testing demonstrated that the EyeBot-32 successfully performed all ELEC3020 laboratory exercises, including wall following, colour tracking, and can manipulation, with high repeatability and minimal tuning. A facilitator survey confirmed major improvements in usability, reliability, and engagement compared to prior platforms. The system's modular design and documented API now support future expansion into other robotics units and research-based projects.

In summary, the EyeBot-32 project delivered a robust, low-cost, and educationally focused robotics platform that re-establishes the use of physical robots in embedded systems teaching at UWA. It demonstrates the successful integration of embedded hardware design, real-time software engineering, and pedagogical innovation, laying a sustainable foundation for future research, laboratory development, and cross-disciplinary applications.

Acknowledgements

I would like to thank my supervisor, Professor Thomas Bräunl, whose support was paramount to the success of this project and whose advice was invaluable. It was a privilege to complete my final-year project under the guidance of one of the leading robotics professors in the field.

I would like to thank the UWA Automation and Robotics group and the UWA Robotics Club for providing me with a home away from home for the duration of my degree. I will dearly miss the time I spent with all of them in pursuit of this project and the many years leading up to it.

Finally, I would like to thank my family and friends who never stopped supporting me during this time – through the late nights, the weekend lab visits, and the dinner runs, without these connections I would not have been able to produce this piece of work.

Table of Contents

List of Figures

List of Tables

Introduction

The teaching of robotics at the University of Western Australia (UWA) begins with the core Embedded Systems (ELEC3020) unit, taken by all Robotics, Software, Electrical, and Biomedical Engineering students in their second or third year. Robotics students then progress to advanced topics in Mobile Robots, Robot Manipulators and Automation, and complete a capstone design unit in Mechatronics. Other disciplines may pursue further robotics studies, though these are largely limited to the Mobile Robots unit (AUTO4508).

Each year, hundreds of students complete ELEC3020 and AUTO4508. As part of their learning, students work with the EyeBot, UWA's in-house differential drive robot, alongside the EyeSim simulator. While EyeSim is considered useful for conceptual learning [1], the physical EyeBot has several hardware-related limitations. The latest version, the EyeBot 8, includes dual motor drivers with encoder support, servo outputs, an analog input, and four Pololu distance sensor ports. The robot can be seen in Figure 1. It runs on a Raspberry Pi 4 using the Linux-based RoBIOS system, a library designed for easy access to EyeBot peripherals [2], and includes a touchscreen LCD for debugging and feedback [3].

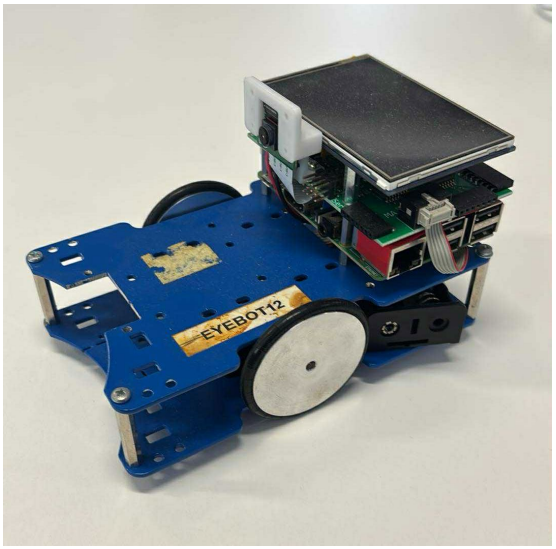


Figure 1: EyeBot-8 Robot

Despite its features, the Raspberry Pi 4 has proven to be overpowered for the EyeBot's purpose. It requires a 3A, 5V supply and often draws more current than all peripherals combined [4]. This compromises its ability to last the required three-hour ELEC3020 lab sessions. The board's complexity also makes troubleshooting difficult in large classes, and many students find the Linux-based environment unintuitive compared to simpler alternatives like the Arduino IDE. On top of that, the high cost per unit and maintenance burden makes large-scale deployment less viable.

Since 2023, UWA's Robotics Lab has increasingly adopted ESP32-based microcontrollers, specifically the TTGO T-Display and T-Display S3, for teaching. These devices offer built-in Wi-Fi, Bluetooth, a colour display, USB-C accessibility, and compatibility with the Arduino framework [5, 6]. With this shift, a clear opportunity has emerged to redesign the EyeBot using the TTGO platform.

This report hence aims to explore the viability of a newly developed EyeBot iteration, the EyeBot-32, and directly compare its viability to the previous generation of EyeBots. This includes reporting on the technical differences between the two iterations as well as an analysis on the effectiveness of

the new generation in a classroom environment, after the production of 20 of these new EyeBot-32 units were rolled out in ELEC3020 classrooms for student's final three lab assignments.

Background

UWA has a tradition of in-house EyeBot development, with the EyeBot 8 being the latest iteration. Alongside the physical robots, the EyeSim simulation system has also been developed, making robotics more accessible for students at home [1]. As stated previously, however, the physical EyeBots were beginning to degrade. The EyeBots are never used in the unit Mobile Robots, deemed too unreliable to carry out the complex tasks and algorithms required to satisfy the courses requirements. In 2024, the decision was made to switch to EyeSim for all of the Embedded Systems EyeBot laboratories. Given that the Embedded Systems unit is seen as the 'gateway' to further study in robotics at UWA, it would be beneficial for students to return to using a physical robot in their labs again.

In selecting a new platform for the EyeBot, a range of commercially available embedded systems and single-board computers were evaluated based on processing capability, cost, power consumption, and ease of integration into an educational environment, as shown in Table 1. High-performance boards such as the NVIDIA Jetson Nano, Khadas VIM3, and ODROID-C4 offer excellent computational capability but at a cost and power draw unsuitable for battery-operated classroom robots. Similarly, while the Raspberry Pi 4 and BeagleBone Black provide full Linux environments and strong community support, they introduce unnecessary software complexity for introductory embedded systems teaching. Simpler microcontroller-based options such as the Arduino Mega, while easy to use, lack the processing power required for more complex robotics tasks.

Table 1: Comparison of Candidate Embedded Platforms

Platform	RAM	Power Draw	Programming Environment	Approx. Cost	Educational and Robotics Suitability
Arduino Mega 2560	8 KB	<100 mA	Arduino	$30	Very beginner-friendly but limited for complex tasks.
Raspberry Pi 4 Model B	2-8 GB	2-3 A @5V	Full Linux (Raspberry Pi OS)	$70	High performance, supports OpenCV and ROS, but overpowered for simple labs.
Beaglebone Black	512 MB	500 mA @5V	Linux / Debian	$80	Good I/O expandability, industrial-grade reliability but more complex environment.
ODRIOID-C4	4 GB	1 A @5V	Linux (Ubuntu)	$85	High performance SBC, more suited for heavy compute tasks.
Banana Pi M2 Zero	512 MB	300 mA @5 V	Linux / Armbian	$50	Low-cost SBC, but cannot drive displays. Poor user reviews.

Khadas VIM3	2-4 GB	1 A @5V	Android / Ubuntu	$150	Powerful AI edge board, but expensive and over-featured.
NVIDIA Jetson Nano	4 GB	2-3 A @5V	Ubuntu / JetPack SDK	$200	Excellent for AI/vision research but too costly and complex for class sets.
ESP32-S3 (TTGO T-Display S3)	512 KB + 6 MB PSRAM	250 mA @5V	Arduino IDE/ PlatformIO / ESP-IDF	$20	Low-cost, dual core, battery friendly. Ideal for embedded robotics education.

The ESP32-S3 platform, integrated in the TTGO T-Display S3 development board, offered an optimal balance of cost, processing power, and feature integration. Its dual-core architecture with hardware floating-point support, built-in Wi-Fi and Bluetooth connectivity, and compatibility with the Arduino and PlatformIO ecosystems made it ideal for both teaching and research. Additionally, its low power consumption (< 250 mA at 5 V) allows extended battery operation without overheating. Based on these comparative advantages, the ESP32-S3 was selected as the core of the new EyeBot-32 design.

Beyond UWA, multiple peer-reviewed studies support the ESP32's suitability for educational robotics. Zhang and Zhang [7] outlined the ESP32's strong peripheral support and real-time performance in an academic control system. Peters et al. [8] introduced the PL-TOON platform, an ESP32-based multi-agent system for teaching decentralised control. It featured onboard PID control, velocity estimation via optical flow, and wireless communication using ESP-NOW and MQTT, all within a $30 unit cost. Benítez et al. [9] developed a remotely controlled ESP32-based robotic arm to teach kinematics and motion control during the pandemic, successfully enabling remote project-based learning. Finally, Goyal [10] compared the ESP32 with other platforms and concluded it is better suited than the Raspberry Pi 4 for embedded robotics. While the Pi excels in computational tasks, it requires a 3A power supply, runs a full Linux OS, and has significant power overhead, making it less suitable for mobile, battery-powered classroom robots. In contrast, the ESP32 supports dual-core processing up to 240 MHz, consumes minimal power, and offers faster boot-up and lower complexity, making it a more efficient and scalable platform for robotics education.

Given the state of the art, a previous attempt was made by Wright [4] to transition the EyeBot to the ESP32 platform. Wright demonstrated the feasibility of using the TTGO T-Display to replicate key functions of the EyeBot 8. However, several limitations remained: the prototype lacked motor encoders, used a breadboard assembly unsuitable for classroom deployment, and relied on a single analogue distance sensor. Testing was limited to a lane-following task, without validating performance across the full suite of ELEC3020 laboratory exercises. While Wright's simplified RoBIOS implementation included image processing and basic drive control, it did not support encoder-based speed or distance feedback or tri-directional distance sensing, features required for motion control in ELEC3020 labs and in future research. Hence, a gap exists for a robust, production ready EyeBot which utilises an ESP32, replacing the current fleet of EyeBot-8 robots for use in ELEC3020 laboratories.

Project Objectives

Based on the reviewed literature, it is expected that a TTGO-based EyeBot will match or outperform the current Raspberry Pi-based EyeBot 8 in educational applications. To validate this hypothesis, several key objectives have been identified.

First, the core physical capabilities of the EyeBot 8 must be retained. The TTGO-based EyeBot must include: a camera, three distance sensors, and two motors with encoder feedback. Additionally, it must be battery drive and have an on/off switch. These components are essential for completing the three ELEC3020 laboratory exercises that rely on the EyeBot. The new design must be simple, robust, and easy to troubleshoot.. Importantly, the EyeBot must also be power-efficient, capable of operating for at least three continuous hours to meet lab session requirements.

The next objective is to ensure full compatibility with all three ELEC3020 labs. These include: (1) a basic driving task using distance sensors to follow predefined patterns [11], (2) image processing to track a coloured object within the camera frame [12], and (3) an object manipulation task where the robot must locate red cans and push them back to a home area [13]. To support these labs, a new version of RoBIOS will be developed to simplify core driving and vision tasks for students.

To evaluate the new EyeBot's effectiveness, a survey will be conducted with the ELEC3020 teaching staff. Facilitators, all of whom have previously completed the unit themselves, will assess ease of use, technical performance, troubleshooting, and interface quality, comparing the new EyeBot-32s performance against the old EyeBot.

Manufacturability is also a key focus. Supporting documentation, including wiring diagrams, I/O tables and a bill of materials, will be developed to enable long-term scalability and reduce dependency on individual maintainers. Finally, the project aims to reduce material costs. The TTGO board is significantly cheaper than the Raspberry Pi 4, retailing around $20 AUD compared to $70 AUD. The Raspberry Pi-compatible EyeBot 8 board alone costs approximately $121 AUD. Excluding peripherals like motors and batteries, the current EyeBot costs over $190 to produce. A TTGO-based board could also be made to be significantly smaller, reducing it's footprint.

Given that the EyeBot is students' first exposure to physical robotics in ELEC3020, a more reliable and accessible platform is also likely to increase student engagement and interest in UWA's robotics stream. The EyeBot-32 is likely to provide groundwork for future research platforms at UWA and potentially for use in other courses if found to be successful.

Design Constraints

The design of the new TTGO-based EyeBot is shaped by several critical constraints, the foremost being that the TTGO T-Display S3 microcontroller forms the core of the platform. This board features a dual-core ESP32-S3 processor, a colour TFT display, and only 13 usable GPIO pins. Many of the processors GPIO pins are already allocated internally to the display, Wi-Fi, and Bluetooth modules [6]. As such, all external hardware must be selected and wired carefully to minimise pin usage and ensure efficient integration. A key advantage of the dual-core processor is its ability to support real-time operating systems (RTOS), enabling concurrent execution of tasks such as encoder reading and sensor polling, an essential feature for responsive robotics.

Given the limited GPIO availability, devices must be prioritised and configured to use shared communication protocols wherever possible. For instance, only one encoder per motor may be connected for feedback if full direction-direction control is not required, since motor polarity will be known in all lab contexts. Additionally, motor drivers should be selected based on minimal control line requirements. Where possible, components that can share communication buses or allow address remapping, such as VL53L0X sensors with dynamic I^2C addressing, are favoured. This careful pin budgeting ensures that critical functionality such as distance sensing, camera data transfer, and basic actuation can all coexist without expanding beyond the TTGO's physical limits.

A custom printed circuit board (PCB) is essential for a robust and maintainable design. The PCB will eliminate fragile jumper wire connections by using fixed female header pins for components such as distance sensors and the camera. This reduces the risk of accidental disconnections and enables quicker diagnosis of issues by demonstrators. Most components, including the TTGO, VL53L0X sensors, and camera, operate at 5V. Motors, however, draw significantly more current and will be powered directly from the system's battery pack at a higher voltage. To protect sensitive components, the PCB will electrically isolate the motor supply from the regulated 5V rail. A 5V regulator with a current rating of at least 2A is required to support the TTGO (up to 500 mA), camera (up to 500 mA), and sensors (up to 100 mA each).

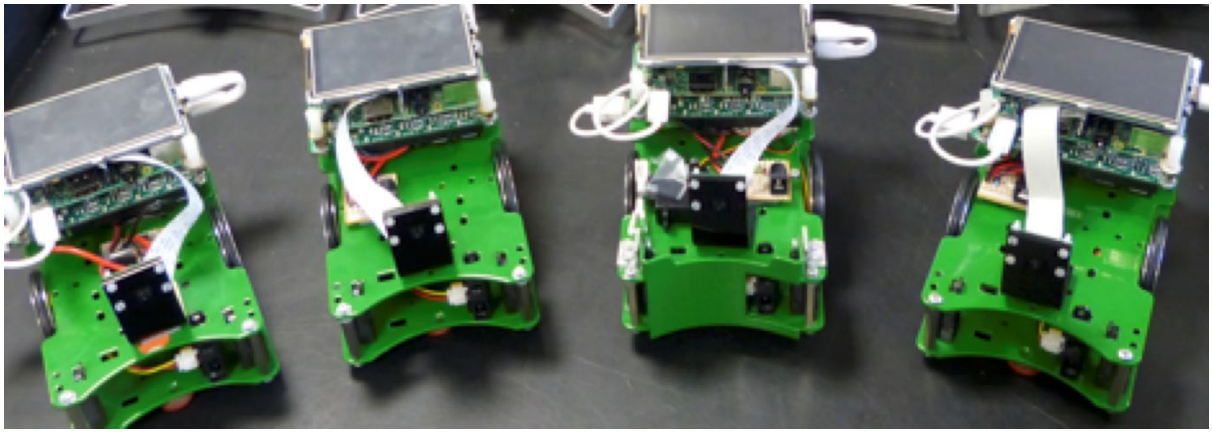


Figure 2: EyeBot Robot Chassis [14]

Mechanically, the existing EyeBot 8 uses a two-layer resin chassis as seen in Figure 2, where the lower layer houses the motors and battery, and the upper layer mounts the PCB and display [14]. This arrangement enables a slim profile, though the overall footprint remains large. This project introduces an opportunity to reduce the EyeBot's physical dimensions. The PCB must sit at a height that allows sensors and the camera to clear the chassis without exceeding the detection height required for ground-level obstacles. A concave front-facing curve will also be added to the chassis to assist with the final lab task involving can manipulation—allowing the robot to "hug" and push objects more effectively.

A lightweight port of the existing RoBIOS API will be ported to the ESP32 platform, offering high-level functions for motion control, vision processing, and sensor reading. Particular attention must be paid to startup sequences for devices using I2C and SPI. These peripherals often require explicit initialisation routines to function reliably across multiple executions. These routines will be included as part of a student-facing template, ensuring ease of integration.

Physical Design Process

Prototyping Stage and Electronics Selection

The EyeBot-32 first began with a prototyping stage where individual components were initially chosen and tested before being integrated to see if they could run together. The main components which needed to be chosen were the motors, distance sensors, camera carrier and camera module.

The motors to be used needed to have encoders within them, as required to successfully carry out any speed or distance control tasks. It was decided early on, to maintain familiarity with the ELEC3020 students and any future researchers, the Sparkfun DG01D-E DC motor was to be used. This motor is used by students in lab assignment 7 of ELEC3020, has a metal gearbox and an encoder built in, as seen in Figure 3. It can operate between a range of 3-9V and is relatively cheap, at $20 AUD per unit [15]. A motor driver board was also needed to regulate the PWM voltage into the motors. Wright opted for a board which was based off of the L9110S H-bridge motor driver. This motor driver board was more complex to program than traditional motor driver boards, as it utilised two input pins to control direction and power, rather than two pins for direction and one for power. However, the trade-off between using less pins but having less intuitive code proved to be valuable, as driving two motors with only four TTGO pins remained a good allocation of pin usage. Hence, a DIP 8 L9110 motor driver was chosen [16], with the intent to integrate this as part of a PCB later, as seen in Figure 4.

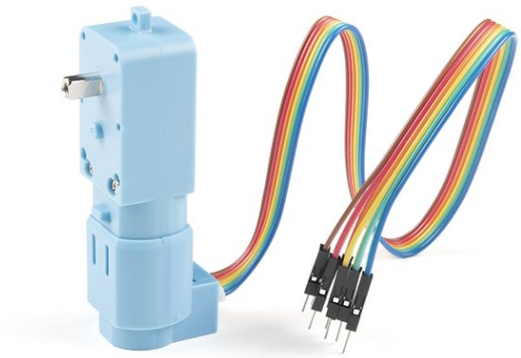


Figure 3: Sparkfun DG01D-E DC Motor with Encoder [15]

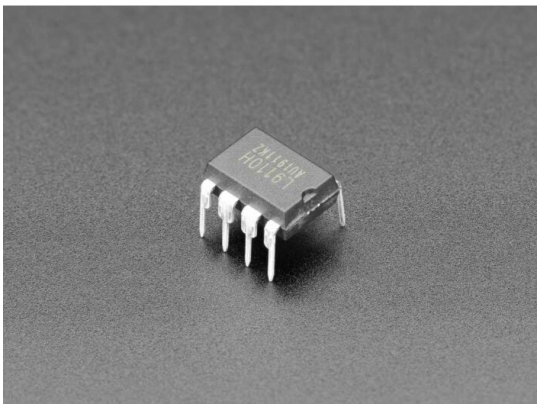


Figure 4: L9110 DIP 8 Motor Driver IC [16]

The next component to be selected were the distance sensors. Before the EyeBot-8, the EyeBot-7 utilised the Sharp GP2Y0A41SK0F analogue PSD sensors, which can be seen in Figure 5 and mounted to the robot in Figure 2. This was also utilised by Wright [4]. These sensors measure distance using a relationship between the detected distance and a voltage output [17]. A hardware description table (HDT) can be used to relate the voltage output to its corresponding distance. While these sensors were tested, it quickly became obvious why these were deprecated. The sensors misread close ranges (less than 80mm) as very high distances, and the voltage differences at distances greater than 400mm remained extremely sensitive, with small changes in voltage corresponding to big changes in distance. As such, it was determined not to utilise these sensors. The other option tested were Polulu's VL53L0X time-of-flight (ToF) distance sensors [18], pictured in Figure 6. These sensors are utilised on the EyeBot-8 (as seen in Figure 1) and are highly accurate distance sensors which output a millimetre distance value to a listed maximum range of 2000mm across an I^2C communication protocol. They also work reliably across an operating voltage of 2.6V to 5.5V, like the TTGO. The Polulu carrier board also came with library functions, an XSHUT pin to control turning the sensor off and on, as well as the ability to dynamically change the I^2C address of each carrier board, which was critical to running multiple distance sensors in parallel. These distance sensors were chosen, however it is important to note that this component is a lot more expensive, at a unit price of $20 USD per board if not bought in bulk.

Figure 5: Sharp GP2Y0A41SK0F [17]

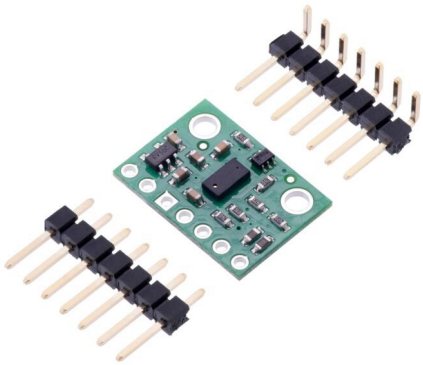


Figure 6: Polulu VL53L0X ToF Distance Sensor [18]

The next component to be chosen was the camera board. Several options emerged, first with the ESP32-AI Thinker Cam. This board was used in previous years of ELEC3020 as part of students semester projects and are extremely well documented online [19]. They are small, cheap and reliable, with Wright opting to use this camera in his own design [4]. However, the camera came with one critical shortcoming; the lack of an onboard programmer. To program the board, one must insert it into a programmer attachment, removing it from its current use [20] as seen in Figure 7. While this would not be a problem for ELEC3020, as the cameras are pre-programmed and not intended to be changed between labs, if the EyeBot-32 is to be used in other applications such as research, it may be tiresome and potentially damaging to the board if it is to be continually swapped in and out just to program it. An attempt was made to integrate a programmer for this module to the PCB, however this did not turn out successful as explained in the PCB Design section.

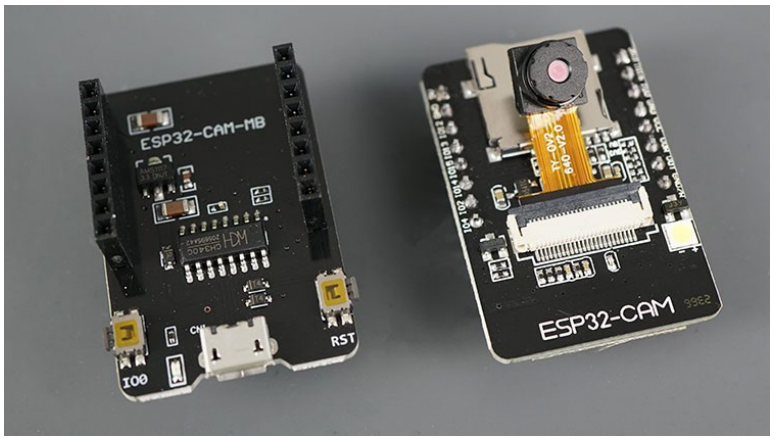


Figure 7:AI Thinker ESP32-CAM with external programmer [19]

Hence, alternatives were considered. The board which was opted for was based around the ESP32-S3 WROOM range of boards, which had a camera carrier as seen in Figure 8. This board remained cheap and had an onboard programmer with a USB-C port, same as the TTGO. The WROOM board clearly outperforms the previous camera board, with a newer chip, the S3, featured on it as well. While some small modifications needed to be made to the code to make it communicate in the same way with the TTGO, it proved to be a highly reliable board. The camera lenses considered were between the OV2640 and OV5640 camera module. After testing, the OV5640 proved unreliable as overheating issues caused it to have a distorted resultant image. This is because the OV5640 is untested with the ESP32 camera carrier boards [21]. Hence, the OV2640, with a wider angle lens was chosen to be used as the camera

Figure 8: ESP32-S3 FREENOVE WROOM Board

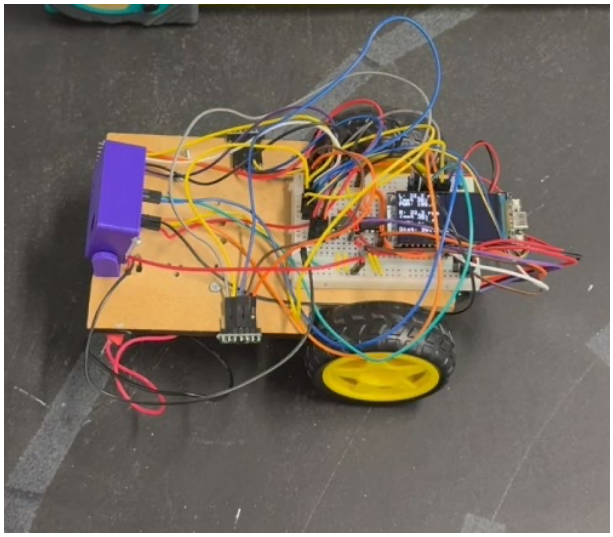


Figure 9: EyeBot-32 Protoype

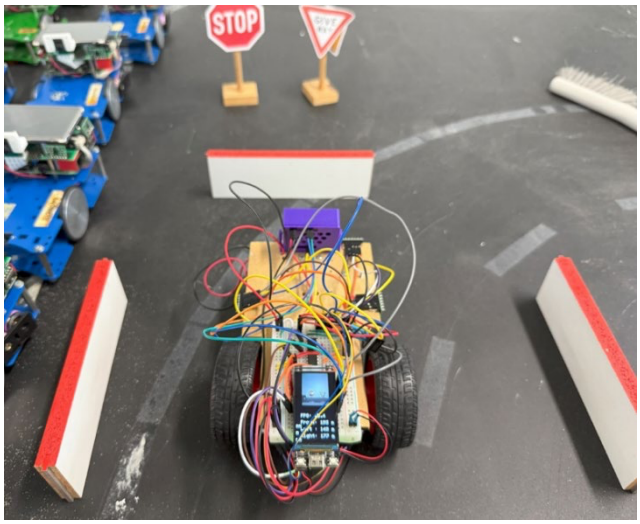


Figure 10: EyeBot-32 Prototype with image and distance sensing

After the selection of the components, a prototype EyeBot was made, by connecting all of the necessary components through jumper wires and a breadboard. It was attached onto a simple wooden

chassis as seen in Figure 9 and 10. This prototype was used to verify that the TTGO could handle simultaneous operation of all the components

Bill of Materials and Cost Analysis

Table 2: EyeBot-32 Components Bill of Materials

Item	Quantity per robot	Price ($ EA)	Price per robot ($)
Battery holders	1	2.145	2.145
Pololu VL53L0X	3	20.63	61.89
Sparkfun DC Motors with Encoders (DG01D-E)	2	20.25	40.5
Wheel for TT Motor	2	1.72	3.44
Polulu Ball Caster with 3/4" Metal Ball	1	7.54	7.54
6 Pin Shrouded Header for Motor Connection	2	1.85	3.70
ESP32-S3 WROOM (excluding camera module)	1	6.13	6.13
OV2640 160DFOV Camera	1	5.73	5.73
LILYGO TTGO T-Display S3	1	20.00	20.00
		TOTAL	**151.075**

Table 2 shows the total price of a single EyeBot unit, excluding the minimal costs of PCB manufacturing and battery costs. This lowered price fulfills the requirements of the new EyeBot-32 costing less than the EyeBot-8. In terms of cost analysis, it is difficult to estimate the precise cost of the EyeBot-8. However, on the EyeBot website [14] it states that one can purchase a PCB board for the EyeBot-8 for around $121 AUD (converted from $80 USD). Purchasing a Raspberry Pi 4 to accompany it retails for $70 AUD, and purchasing three Polulu VL53L0X distance sensors costs at least $60 AUD. Without including other significant cost drivers such as the motors, camera module, touch screen or chassis of the EyeBot-8, these three components combined cost atleast $251. The EyeBot-32, with all of it's PCB components and electronic components costs around $150 for a single robot, as previously referred to in Table 1. This represents a significant cost decrease for each robot and makes maintenance far cheaper and more accessible for the UWA Robotics Lab.

Electrical Wiring

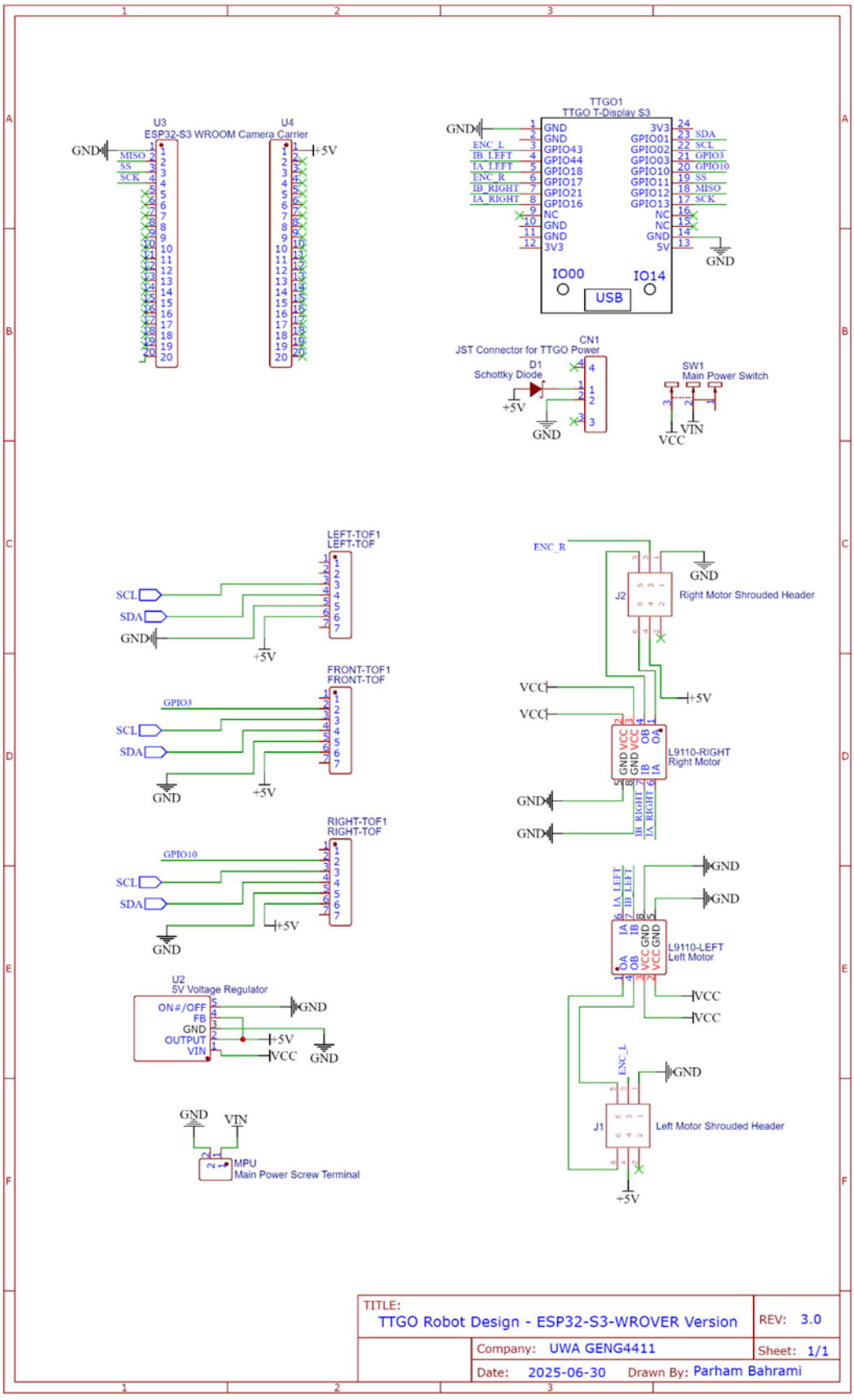


Figure 11: EyeBot-32 Final Electrical Wiring Diagram

Table 3: Required GPIO pin connections between the T-Display-S3 and Peripherals

TTGO T-Display S3 Pin	Peripheral Pin/Description
11	WROOM Camera Pin 20
12	WROOM Camera Pin 21
13	WROOM Camera Pin 19
44	Left Motor Driver IB
18	Left Motor Driver IA
21	Right Motor Driver IB
16	Right Motor Driver IA
43	Left Motor – Encoder Channel A
17	Right Motor – Encoder Channel A
1	ToF Sensors SDA
2	ToF Sensors SCL
3	XSHUT Pin Front ToF Sensor
10	XSHUT Pin Right ToF Sensor

Given the limited number of general-purpose input/output (GPIO) pins available on the TTGO T-Display S3, careful pin budgeting was essential throughout the design process. After iteratively testing configurations of pin assignments, a final wiring diagram and GPIO truth table was made as seen in Figure 11 and Table 3.

Many of the ESP32-S3's pins are internally allocated to the onboard TFT display, USB-C interface, and serial communication functions, leaving only a subset of pins accessible for external peripherals [5]. To maximise functionality, shared communication protocols such as I²C SPI were utilised wherever possible. I²C was particularly valuable as it allowed multiple devices, such as the three VL53L0X distance sensors, to operate on a common two-wire bus by assigning unique dynamic addresses. Similarly, SPI was used to interface high-bandwidth devices like the camera. The SPI connection was kept as a single direction line, with only the camera communicating to the TTGO, to reduce the number of pins required by one. This efficient use of digital buses not only reduced wiring complexity but also simplified PCB routing and improved system reliability.

The EyeBot-32 is powered using two protected 18650 lithium-ion cells in series. This provides 7.4V to the motors and is stepped down to 5V to power all other electrical peripherals. During testing, these batteries could easily last for three hours of continuous use and days worth of intermittent continuous use, clearly completing the requirement of a design that could last for an entire three-hour laboratory.

PCB Design

As outlined in the project requirements, a robust PCB design is needed in order to maintain a reliable electrical connection between all the chosen modules. The PCB design is the main driver of much of the chassis design. This determined where each of the components are, the size of the chassis and ease of use. A goal while creating the chassis was to ensure as small of a footprint as possible. In the first iteration, two PCB designs were fabricated, one based around the ESP-32 AI Thinker Cam and another based around the ESP-32 S3 WROOM cam. An attempt was also made to design a programmer for the ESP-32 AI Thinker cam using a USB to UART programming chip.

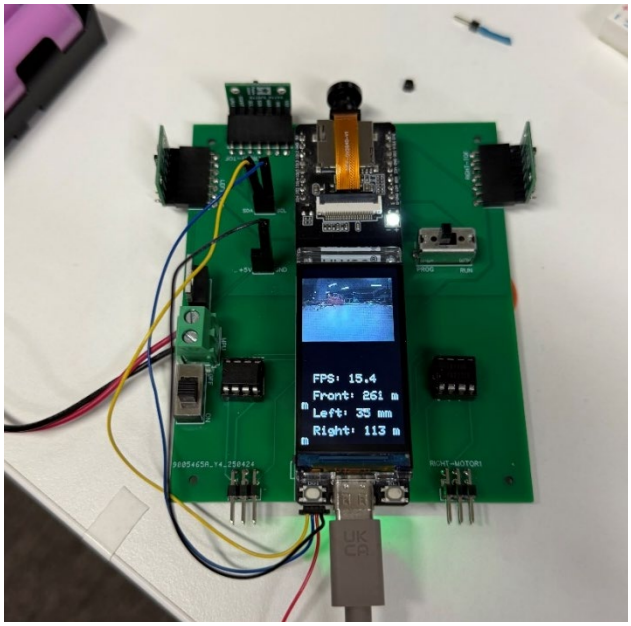


Figure 12: ESP32 AI Thinker Cam PCB Design

As can be seen in Figure 12, the PCB design featured 6 pin headers for the motors, 7 pin 90 degree female headers for the ToF sensors, and two rows of headers for the TTGO and the ESP32-Cam. It also included screw terminals for external power, a power switch, a 5V voltage regulator and two 2-pin female header pins for the TTGO power and I2C breakout pins. It also featured a serial programmer and USB-C port, which were to act as a programmer for the ESP32 AI Thinker Cam. The design of this programmer was based off of multiple open source designs of these programmers [22, 23], and was adapted for our use case. A double pole, double throw switch is used to change the camera from a running mode to a programming mode, as the SPI connection to the TTGO would need to be cut during programming of the camera board. A MOSFET is also used to control the drive of power to pin 0 of the ESP32 CAM by the TTL converter. When this was tested, while the computer would detect the camera board when plugged in, the programmer was unsuccessful in driving the MOSFET in the right sequence that was required to have the camera enter a programming mode.

The WROOM edition of the PCB was the exact same as previously described for the ESP32-CAM, however did not require an external programmer. Due to the WROOM cam being longer than the ESP32-CAM, it caused a considerable increase in the length of this PCB, as seen in Figure 14. Hence, it was determined that for future iterations to decrease the length of the board by placing the TTGO horizontally along its longer side and to rearrange the parts to create a slimmer profile. The final PCB is shown in Figure 13. It also experimented with placing a Schottky diode and a JST connecter to the TTGO power cable. During testing, however, the Schottky diode did not work as intended and the TTGO could still back feed to the system when powered through USB-C. Additionally, the JST

connecter that powers the TTGO is a 2-pin, 1mm SH cable [6], which due to its small size is difficult to crimp into a JST connector. Hence it was determined that all the TTGO power cables would be soldered directly onto the board. An opportunity exists to redesign the board with more robust terminations for the TTGO power cable. This board also made use of Amphenol shrouded headers, which made terminating the motor cables to the board easy and solderless [24, 25].

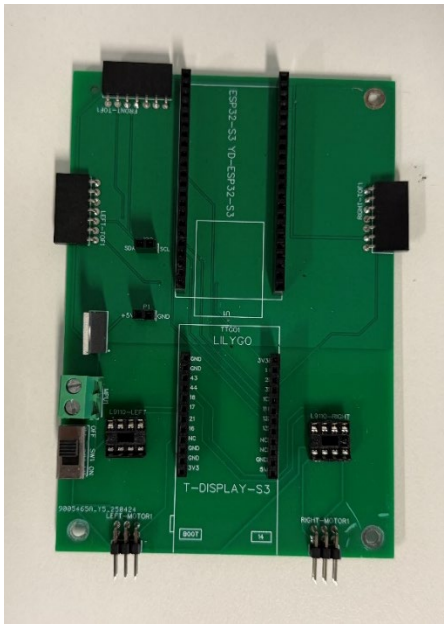


Figure 13: Initial WROOM Cam PCB

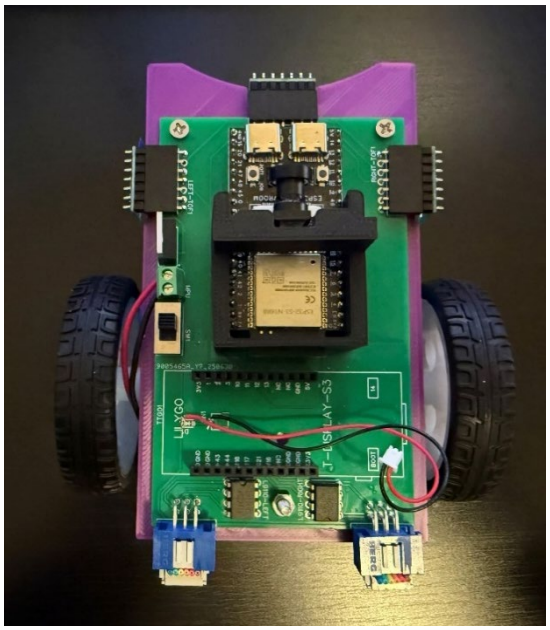


Figure 14: Final WROOM PCB, mounted onto EyeBot-32 Chassis

Chassis Design

The design of the EyeBot-32's chassis evolved with the new iterations of the PCBs. One of the objectives of a chassis design was to be as minimal as possible, smaller and lighter than the EyeBot-8 chassis design which remained largely unchanged for years. Multiple chassis designs were created. Due to the need to rapidly redesign the chassis of the robot inexpensively, it was decided that the frame would be made from PLA filament, and designs fabricated using the UWA Robotics Lab's 3D printers.

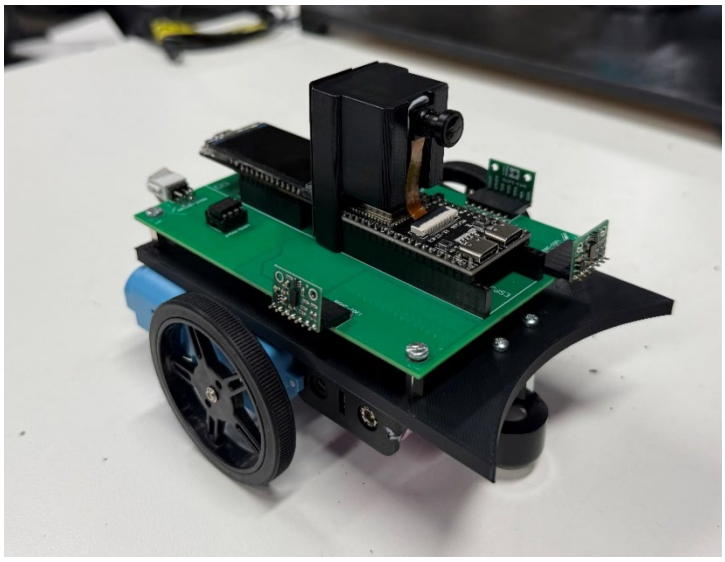


Figure 15: Initial EyeBot Chassis Design

The first design of the chassis as seen in Figure 15, featured the initial WROOM PCB. This design featured the motors being placed in an ‘inverted’ fashion, which pushed the wheels to the centre of the chassis. A single caster wheel was placed in front of the motors and the battery holder, which was held using M3 screws through its mounting holes. This design was far too bulky, as well as having a fundamental flaw of no rear support, meaning that the robot would often tip backwards and scrape its rear along the floor while driving. The camera holder was also far too bulky, and took up too much space. Future designs were to be more minimal.

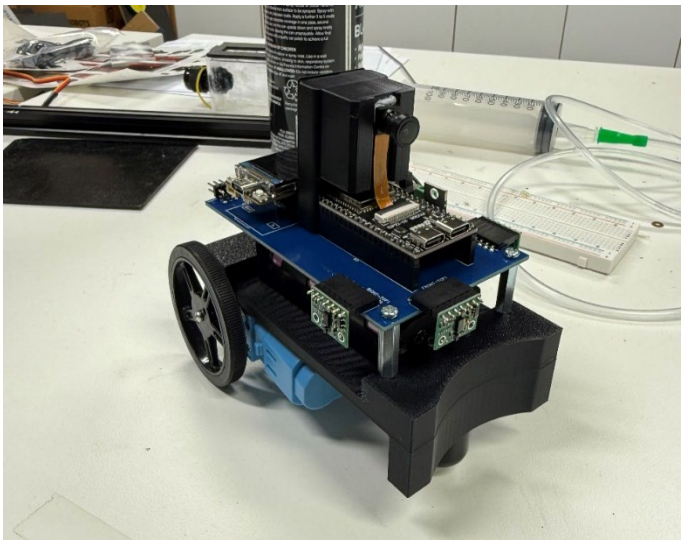


Figure 16: Second Iteration of EyeBot Chassis

The second iteration of the EyeBot-32 chassis as seen in Figure 16 reduced the width and length of the design to match that of the PCB. With this decrease in size, the battery pack also needed to be moved, as it could no longer fit between the motors and the caster wheel. A design was made which elevated the PCB using standoff screws and included a central cutout in the chassis to fit the battery pack between the PCB and the chassis. A ‘pusher’ was also extruded towards the floor from the front of the robot, to support its use in the final lab of ELEC3020. This resulted in an increase in the height of the robot and contributed negatively to the lowering of the robot’s footprint. However, an opportunty was realised in the distance sensors being inverted. In this way the sensors were lower to the floor which was ideal in detecting objects, were better protected as they were more in line with the chassis’ profile. The front sensor could also be centred as the inverted configuration would no longer interfere with USB-C port of the WROOM camera board.

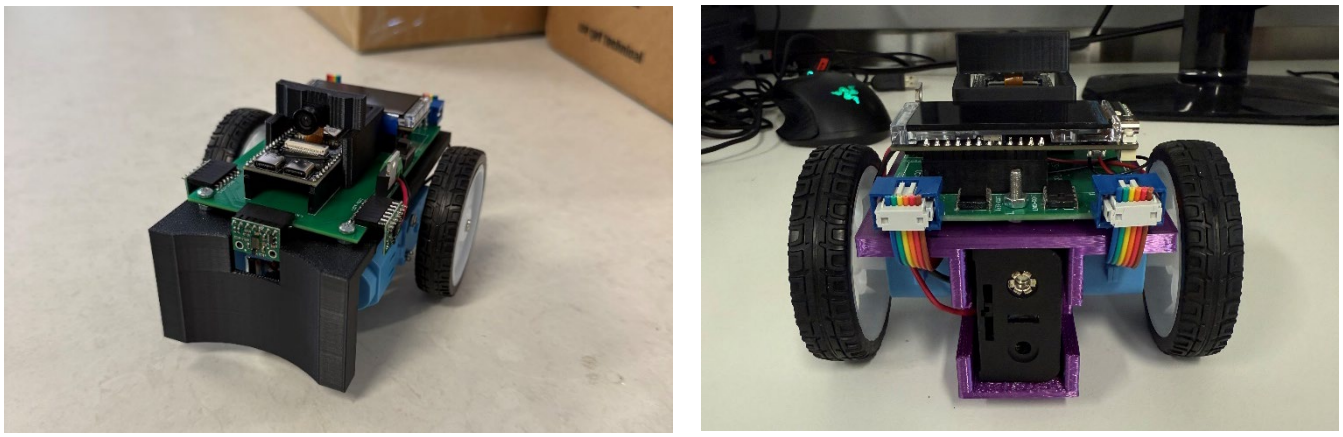


Figure 17: Final EyeBot Chassis Design - Angled Front View and Rear View

The final design of the EyeBot chassis, as seen in Figure 17, features the final minimalistic style. The major difference is the location of the battery pack. After measuring each components dimensions, a design was made which neatly tucks the battery pack between the motors. To support this, the motors were also moved several millimetres lower to maintain a solid ground clearance to the floor. The front pusher was also extended lower to the floor, and cutouts were made where each of the distance sensors were located. A new camera holder was also designed, which significantly reduced the footprint of the robot. This new design resulted in the smallest and most stable design of the EyeBot-32. Twenty of the EyeBot-32s were built during semester two in readiness for deployment in the ELEC3020 laboratories at the end of Semester 2, 2025.

Final Design Comparison

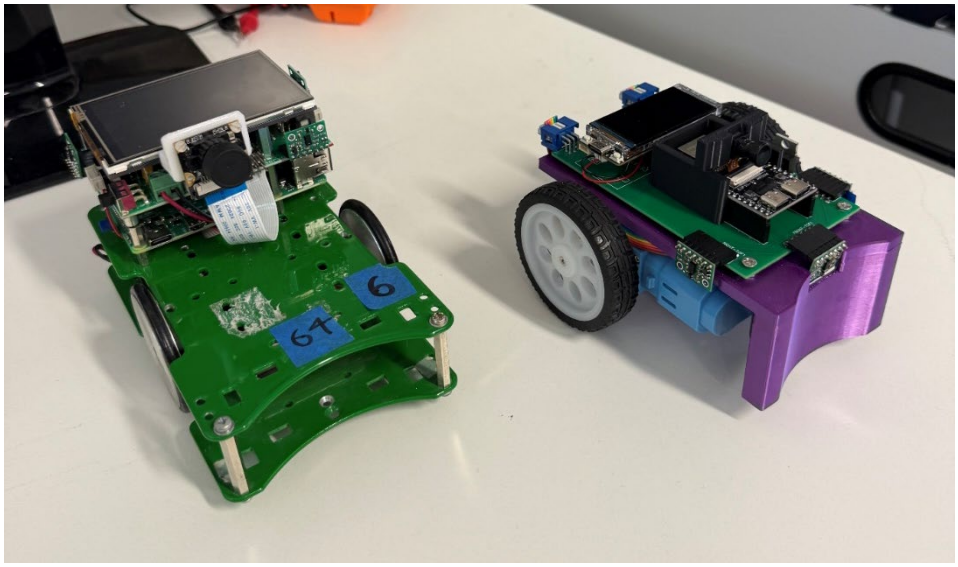


Figure 18: EyeBot 8 and EyeBot-32 Angled View

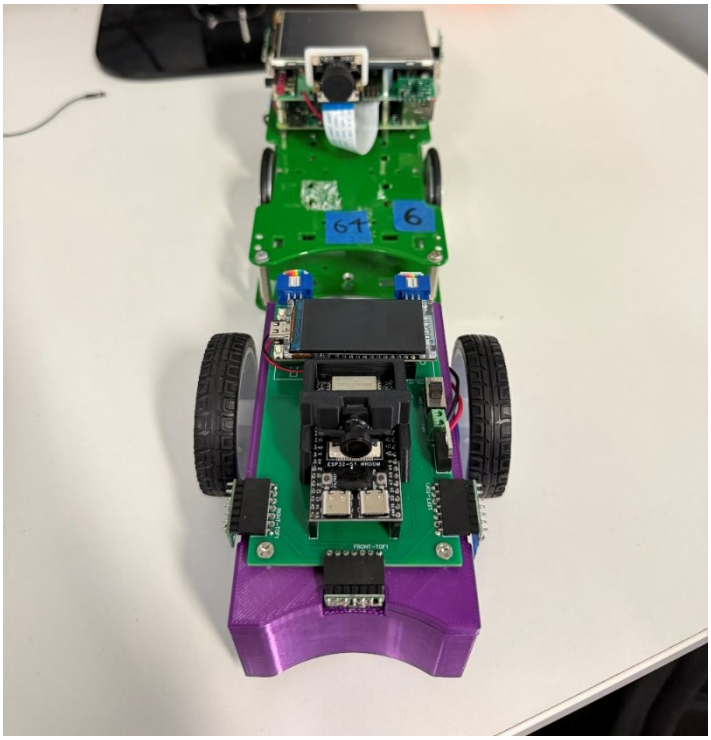


Figure 19: EyeBot-8 and EyeBot-32 Width Comparison

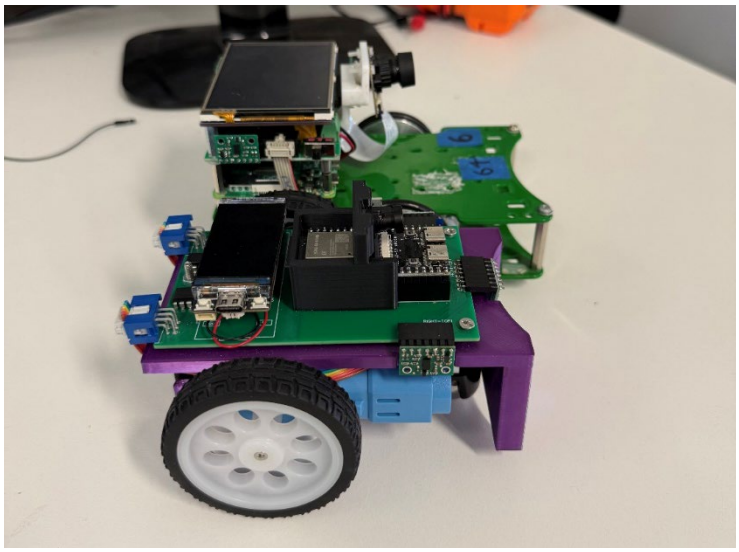


Figure 20: EyeBot-8 and EyeBot-32 Length Comparison

Table 4: Physical Dimensions of EyeBot-8 and EyeBot-32

	EyeBot-8	**EyeBot-32**	**Difference**
Length	155mm	130mm	-25mm
Width	97mm	120mm	+23mm
Height (ToFs)	80mm	60mm	-20mm
Height (Camera)	100mm	90mm	-10mm
Weight	841.3g	438.9g	-400.4g

As can be seen in Table 4, the EyeBot-32 successfully completes our project requirements of developing a more minimalistic EyeBot design. The length, height of the ToFs, and the weight of the robot have decreased as seen in Figure 11 and 13 while the width of the robot from wheel to wheel has increased, as seen in Figure 12. This increase in width can be attributed to the width of the wheels, as the chassis is thinner than the EyeBot-8, and the width can be decreased by purchasing thinner wheels. The only component which saw a drawback was the height of the camera lens, which has a lower height than that of the EyeBot-8. To compensate for this, the camera holder implements a 10 degree incline angle which orients the camera lens to be facing slightly upwards.

Software Design Process

RoBIOS32 Library Overview

As mentioned in the project requirements, a comprehensive port of the original RoBIOS programming interface was required to ensure the EyeBot-32 could directly replace the EyeBot-8 in the ELEC3020 laboratories. This led to the development of a new library named RoBIOS32, implemented using the Arduino framework and compatible with both the PlatformIO and Arduino IDEs.

RoBIOS32 provides a simplified, educationally oriented API that abstracts away lower-level hardware control, allowing students to focus on algorithmic development rather than microcontroller details. A total of 67 accessible functions were implemented, grouped into the following categories:

- LCD Output – image and text rendering functions for the integrated display
- Key Input – button reading and edge detection functions
- Camera – image capture in RGB565 and RGB888 formats
- Image Processing (IP) – filtering, thresholding, and morphological operations
- Distance Sensors (PSD) –distance measurement using VL53L0X ToF sensors
- Motors – direct PWM motor control functions
- V–Ω (VW) Interface – velocity- and position-based motion primitives

While Wright [4] had begun porting a small subset of these functions to the TTGO platform, substantial work was required to complete and stabilise the implementation for full classroom use. This particularly involved the integration of accurate speed and distance control using the motor encoders. A full list of the integrated functions may be found in the appendix under Appendix Entry A.

PlatformIO Project Environment

All software was developed using the PlatformIO IDE, chosen for its robust dependency management and reproducible build environment. Dependencies including VL53L0X, TFT_eSPI, and Wire libraries were declared within the project's *platformio.ini* file to ensure consistent compilation across all development systems. The environment also enabled automated firmware uploads via USB-C and debugging via serial monitoring. This workflow allowed rapid testing and repetability across all 20 EyeBot-32 units.

Multitasking Architecture

A critical component of the software architecture was the implementation of true multitasking using FreeRTOS, the real-time operating system built into the ESP32-S3. FreeRTOS allows concurrent task scheduling across the processor's two cores, enabling time-critical control processes to run independently from user-level code.

- Core 0 executes the motor control task, responsible for encoder feedback, PID regulation, odometry updates, and VW motion execution. This task runs at a fixed period of 50ms for encoder polling and PID response.
- Core 1 handles the main user program, including image capture, processing, and general logic defined by the student's application code.

Shared data such as odometry variables are protected using mutex locks to prevent race conditions between tasks. This structure provides responsive control, efficient CPU utilisation, and consistent real-time performance during lab experiments.

Motor and Motion Control

The integration of motor encoders enabled accurate speed and distance feedback, allowing implementation of a closed-loop PID controller for both motors. Each VW command (e.g. *VWSetSpeed, VWCurve, VWTurn*) uses inverse-kinematic equations to calculate wheel speeds and

durations, which are then passed to a motor control task on Core 0. PID tuning was performed experimentally using response plots to select proportional, integral, and derivative constants that yielded optimal behaviour across all robots. Odometry functions were also implemented, tracking global x, y, ϕ coordinates in real time. This functionality had not been reliably available in earlier EyeBot versions and significantly improved the accuracy and repeatability of drive commands.

Camera and Image Processing

The camera functions were adapted for the ESP32-S3 WROOM module, which captures images in RGB565 format by default. Since the TFT display also uses RGB565 encoding [6], display functions could directly render captured frames. However, as most computer-vision operations require RGB888 data, the library includes automatic conversion functions that unpack each pixel into 8-bit red, green, and blue components. Two corresponding function sets were implemented, standard RGB888 functions (matching the RoBIOS naming scheme) and RGB565-specific variants (suffixed with "565"). Additional features include grayscale conversion, brightness/contrast adjustment, and morphological opening (IPOpen) to remove salt-and-pepper noise. These operations form the foundation for colour tracking and object detection used in ELEC3020 laboratory tasks. The camera can stream reliably at around 13FPS.

Inputs, Sensors, and Display

The key input subsystem manages the two onboard TTGO buttons, providing both blocking and non-blocking reads, and recognising simultaneous dual-button presses as a third state. The distance-sensor module, still referenced as "PSD" for backward compatibility, wraps the VL53L0X ToF library to provide millimetre-accurate readings via the PSDGet() function. These sensors are set to have continuous ranging, polling every 20-30ms, with a non-blocking read allowing access to the current distances when required. Finally, the LCD module provides a lightweight interface to the TFT display using the TFT_eSPI library's backend. Support was added for both RGB888 and RGB565 images, with the former automatically converted before display.

Results and Discussion

Lab Implementations

With the RoBIOS library at its draft stage, the EyeBots were tested by implementing all three of the ELEC3020 EyeBot laboratory assignments, and fixing any bugs or performance issues found in the library functions.

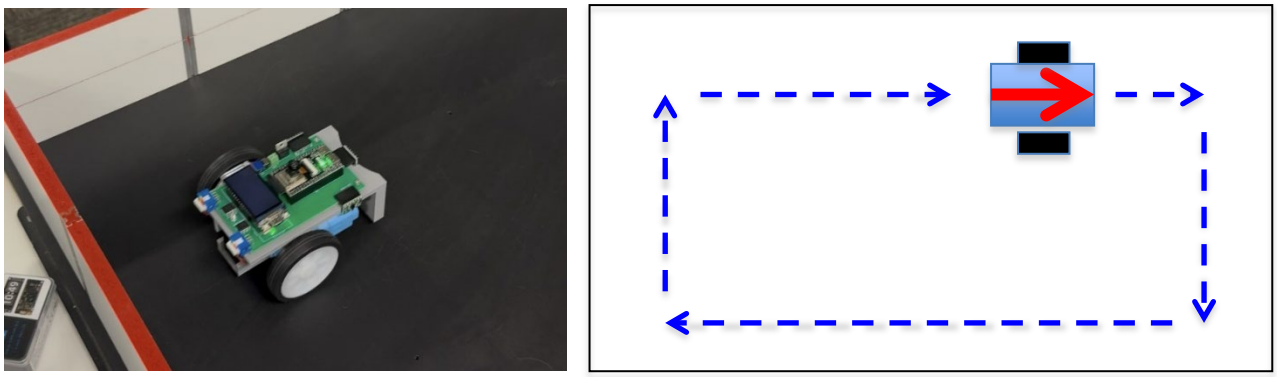


Figure 21: EyeBot-32 Wall Following Task [11]

The first laboratory, lab assignment 8 [11] was simple to implement. Turning parallel to the wall was best achieved by calculating the angle of turn required by using the distance sensors and performing an on the spot turn to become parallel to the wall. Wall following was implemented next, as shown in Figure 21, which was also simple. Using a on-off controller could correct any drift with the robot by either turning away from the wall if the robot got too close and turning into the wall if it got too far.

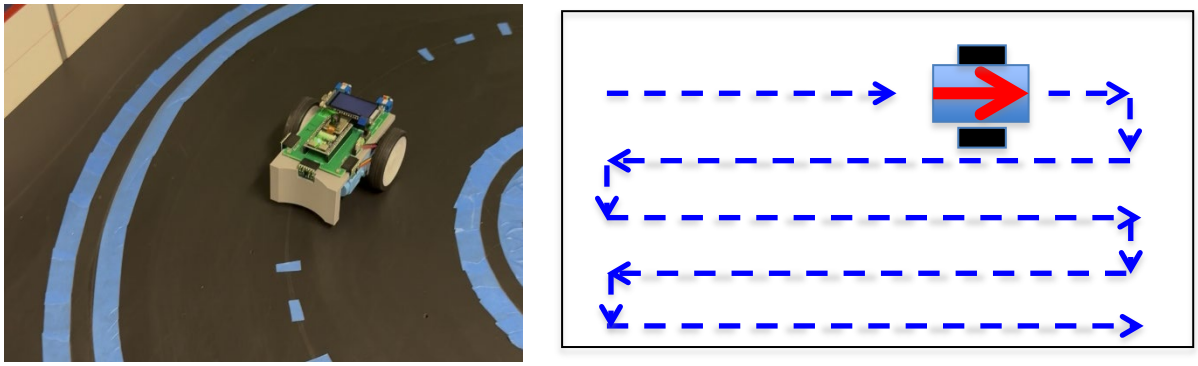


Figure 22: EyeBot-32 Lawnmower Task [11]

The lawnmower task was also very easy to implement due to highly accurate and reliable distance commands such as VWTurn and VWStraight. The robot, when commanded to drive straight, would be reliably able to drive straight and not drift a significant amount, however it is noted that students are expected to correct drift as part of their lab code. Testing indicated that for every 1m travelled, the robot would drift around 5cm laterally, making it more than accurate enough for the tasks in the labs.

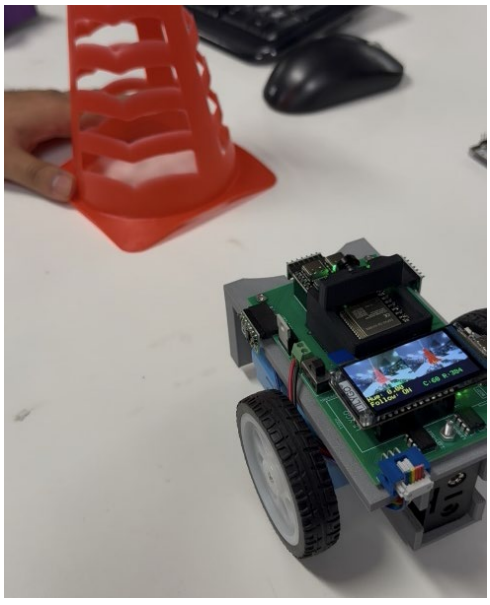


Figure 23: EyeBot-32 Lab 9 Completion [12]

Lab assignment nine [12] was more complex to implement, but far more reliable than the previous driving task due to their being no drift to compensate for. The camera functions and image processing functions worked very reliably, as seen in Figure 23. However the memory restrictions on the TTGO became apparent. Since the usable DRAM (direct RAM) of the ESP32 is 328kB, declaring arrays for the images that are of size 57.6kB quickly consumes the DRAM and could lead to an overflow [5]. The TTGO, however, contains 6mB of external PSRAM (pseudostatic RAM) memory [6]. This external memory is incorporated in the memory map and is usable in the same way as DRAM. This

extra 6mB of memory allows multiple images to be saved without issue, allowing simple completion of Lab 9.

Figure 24: EyeBot-32 Lab 10 Completion [13]

Lab assignment 10 [13], as seen in Figure 24, is a culmination of both lab assignments 8 and 9. Implementing this was relatively simple as long as it was done slowly and steadily, with frequent checks to ensure that the robot is aligned with the red object.

Classroom Rollout and ELEC3020 Facilitator Feedback

To support the rollout of the robots in the ELEC3020 classrooms, a user manual was created. This user manual specified how each of the components work and how to install the library provided to the students. The library which was given included a precompiled source file and a fully documented and Doxygen compliant commenting for each library function, which aided student's in their use of the library. It lists all of the available RoBIOS functions, what parameters they take and how they work. Finally, the user manual specified some troubleshooting steps for the robot as well as a short example. It can be found in the appendix under Appendix Entry B.

To prevent bias in reporting the results, a survey was distributed to all of the ELEC3020 facilitators, which was answered after their Week 11 lab assignment 9 classes. This survey presented a mix of both Likert-scale questions as well as open ended text questions.

Figure 25: Lab 9 Completion in ELEC3020 Laboratory, Semester 2 2025

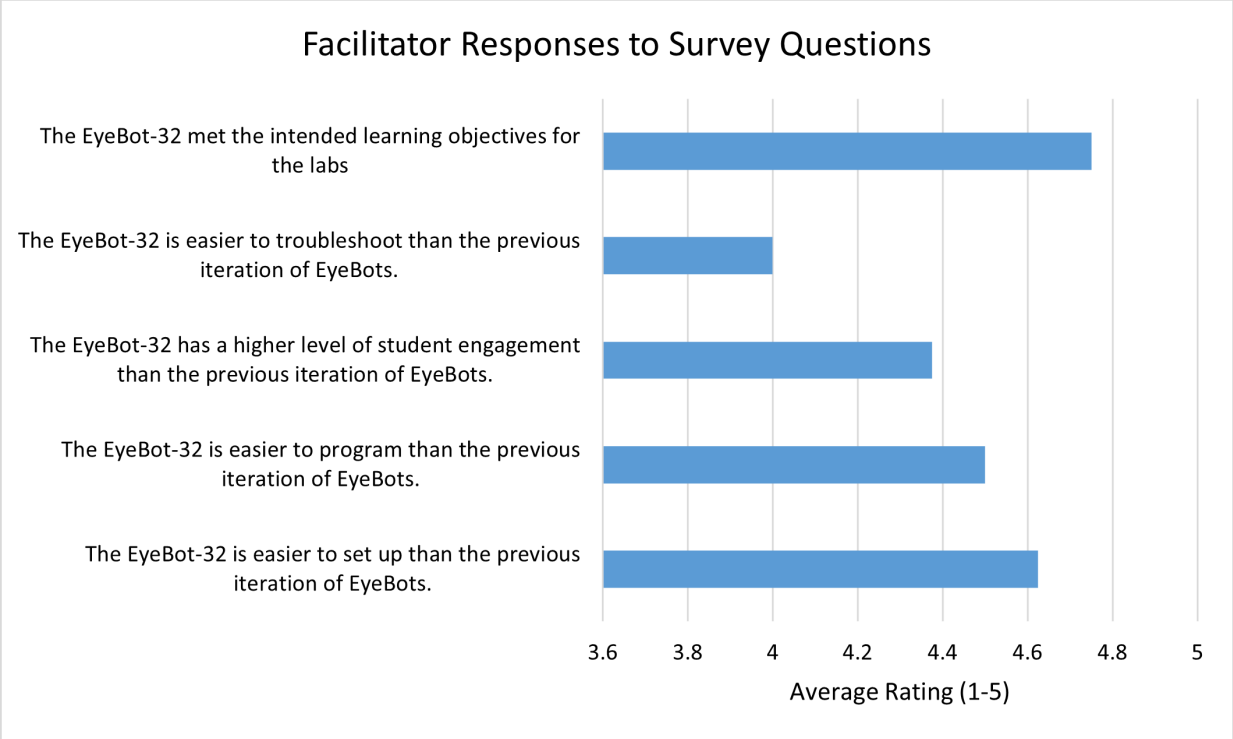


Figure 26: Facilitator Responses to Survey Questions (Likert-scale)

Facilitator responses were overwhelmingly positive across all criteria, with average scores ranging between 4.0 and 4.75 out of 5, with 5 being "Strongly Agree" and 4 being "Agree". The highest-rated aspects were "Ease of setup" and "Achievement of learning objectives," indicating that the new EyeBot-32 platform successfully addressed key limitations of the previous generation.

When asked about what improvements they would like to see, facilitators overwhelmingly highlighted documentation consistency as the primary area for improvement. Several responses mentioned that the setup and library instructions were "inconsistent" or "unclear in some parts." Many recommended producing clearer, consolidated documentation and step-by-step teaching materials for both students and facilitators. Hardware and software suggestions were comparatively minor but included requests for a more powerful microcontroller, increased RAM, and minor refinements to the power and sensor systems to ensure reliability. Overall, the feedback indicated that the platform's core performance was strong, with refinements mainly focused on improving the user-facing resources rather than the design itself.

When asked what single change the facilitators would like to see, most respondents reiterated the need for better documentation and instructional clarity, particularly around threading, battery connection, and memory management. One facilitator suggested revising the battery holder to make incorrect insertion impossible, while another mentioned improving the camera's memory handling.

The majority of facilitators agreed that the EyeBot-32 platform also has strong potential for expansion into other robotics and control units, with specific mentions of AUTO4508 (Mobile Robots). Some noted that the system could support higher-level programming or vision-based exercises, aligning with UWA's robotics curriculum progression. A few respondents were uncertain which units would directly benefit but acknowledged the platform's flexibility and adaptability.

Conclusion and Future Work

In conclusion, the EyeBot-32 successfully met its core objectives as a classroom-ready replacement for the Raspberry Pi-based EyeBot 8. The redesigned platform, centred around the TTGO T-Display S3 microcontroller, demonstrated a significant reduction in size, weight, and cost while maintaining the essential hardware capabilities required for ELEC3020 laboratory exercises. The integration of encoder-based motor control, tri-directional distance sensing, and real-time image processing on an embedded system marked a major improvement in both functionality and reliability. In testing, the EyeBot-32 performed all three laboratory tasks; wall following, colour tracking, and can manipulation, consistently and with minimal drift, validating its suitability for large-scale classroom deployment. Feedback from facilitators confirmed that the new platform was easier to use, quicker to troubleshoot, and more engaging for students compared to the previous generation

The successful porting of the RoBIOS library to the ESP32 environment also represents a key milestone, bridging the familiarity of the previous API with the accessibility of Arduino and PlatformIO frameworks. The adoption of FreeRTOS for multi-core control enabled robust background task management and smooth performance even during simultaneous camera and motor operation. From a manufacturing standpoint, the use of a custom PCB and 3D-printed chassis allowed for streamlined assembly and easier maintenance.

There remains ample opportunity, however, for refinement and future development. On the hardware side, future iterations could explore lower-cost distance sensors to further reduce manufacturing costs. The power system could also be improved to prevent USB back-feeding and simplify cable management. Software-wise, ongoing refinement of the RoBIOS library will be essential to improve efficiency, memory management, and reliability. Additional image-processing algorithms could be implemented directly on the TTGO, such as object segmentation, feature matching, or visual SLAM, to expand the robot's research and educational potential beyond basic colour tracking. On the educational side, more comprehensive and clear documentation needs to be made to ensure that students have more clarity on how the robot works. Finally, implementing wireless communication via Wi-Fi or Bluetooth would enable multi-robot coordination experiments and real-time telemetry streaming to external dashboards, providing a strong foundation for future research at UWA.

Overall, the EyeBot-32 project demonstrates that modern microcontroller-based systems can deliver high educational value with minimal cost and complexity. Continued refinement of both hardware and software will ensure that the EyeBot-32 remains a sustainable, scalable, and pedagogically effective platform for teaching embedded systems and robotics at the University of Western Australia.

References

[1] T. P. Michael Finn, Joel Frewin, Thomas Braunl, "Robot Simulation for Teaching Engineering Concepts," presented at the Research in Engineering Education Symposium & Australasian Association for Engineering Education Conference, Perth, WA, 2021.

[2] T. Bräunl. "RoBIOS-7 Library Functions." https://roblab.org/eyebot/robios.html (accessed 12/10/2025.

[3] M. P. Thomas Bräunl, Remi Keat, Daniel Holding, "EyeBot 8 User Guide," University of Western Australia, 2024.

[4] B. Wright, "A camera-equipped, ESP32-S3 microcontroller-based mobile robot design capable of autonomous navigation via visual lane detection," School of Engineering, University of Western Australia, 2024.

[5] E. Systems, "ESP32-S3 Series Datasheet," ed, 2025.

[6] X. LilyGO. "T-Display-S3." https://github.com/Xinyuan-LilyGO/T-Display-S3 (accessed 12/10/2025.

[7] Y. Zhang and L. Zhang, "Research on Education Robot Control System Based on ESP32," *Journal of Education and Educational Research,* vol. 7, pp. 299-302, 02/29 2024, doi: 10.54097/3x86qp78.

[8] A. A. Peters, F. J. Vargas, C. Garrido, C. Andrade, and F. Villenas, "PL-TOON: A Low-Cost Experimental Platform for Teaching and Research on Decentralized Cooperative Control," *Sensors*, vol. 21, no. 6, doi: 10.3390/s21062072.

[9] V. H. Benitez, R. Symonds, and D. E. Elguezabal, "Design of an affordable IoT open-source robot arm for online teaching of robotics courses during the pandemic contingency," *HardwareX,* vol. 8, p. e00158, 2020/10/01/ 2020, doi: https://doi.org/10.1016/j.ohx.2020.e00158.

[10] Y. Goyal, "Comparative Study of Microcontoller: ARDUINO UNO, RASPBERRY PI 4, ESP 32," *International Journal for Research in Applied Science & Engineering Technology (IJRASET),* vol. 12, no. VII July 2024, 2024.

[11] T. Braunl, "Lab Assignment 8 - Robot Driving. Embedded Systems ELEC3020," 2024. [Online]. Available: https://roblab.org/courses/embedded/labs/lab08-E-drive.pdf

[12] T. Braunl, "Lab Assignment 9 – Image Processing. Embedded Systems ELEC3020," 2024. [Online]. Available: https://roblab.org/courses/embedded/labs/lab09-E-camera.pdf

[13] T. Braunl, "Lab Assignment 10 – Robot Driving and Object Detection," 2024. [Online]. Available: https://roblab.org/courses/embedded/labs/lab10-E-cans.pdf

[14] T. Bräunl. "EyeBot." https://roblab.org/eyebot/ (accessed 12/10/2025.

[15] Sparkfun, "Hobby Motor with Encoder - Metal Gear (DG01D-E)," 2025. [Online]. Available: https://www.sparkfun.com/hobby-motor-with-encoder-metal-gear-dg01d-e.html

[16] Adafruit. "L9110H H-Bridge Motor Driver for DC Motors - 8 DIP - 2.5V-12V 800mAh." https://www.adafruit.com/product/4489 (accessed 12/10/2025.

[17] Polulu. "Sharp/Socle GP2Y0A41SK0F Analog Distance Sensor 4-30cm." https://www.pololu.com/product/2464 (accessed 12/10/2025.

[18] Polulu. "VL53L0X Time-of-Flight Distance Sensor Carrier with Voltage Regulator, 200cm Max." http://pololu.com/product/2490 (accessed 12/10/2025.

[19] ESPBoards. "AI Thinker ESP32-CAM Development Board." https://www.espboards.dev/esp32/esp32cam/ (accessed 12/10/2025.

[20] R. N. Tutorials. "Upload Code to ESP32-CAM AI-Thinker using ESP32-CAM-MB USB Programmer (easiest way)." https://randomnerdtutorials.com/upload-code-esp32-cam-mb-usb/ (accessed 12/10/2025.

[21] M. Pivovarsky. "Freenove ESP32-S3-Wroom." https://github.com/prusa3d/Prusa-Firmware-ESP32-Cam/tree/master/doc/Freenove%20ESP32-S3-Wroom (accessed 12/10/2025.

[22] M. Balwierz. "cam-prog instructions." https://bitluni.net/camprog (accessed 12/10/2025.

[23] G. C. Insan Sains. "ESP32-Cam Programmer." https://github.com/geraicerdas/ESP32-Cam-Programmer (accessed 12/10/2025.

[24] Amphenol, "Quickie® IDC receptacle, Wire to Board, 2.54mm(0.100in), Double Row, 6 positions," 2025. [Online]. Available: https://www.amphenol-cs.com/product/71600006lf.html.
[25] Amphenol. "Quickie®, Wire to Board Connector, Double Row, 6 Positions, 2.54 mm (0.100in), Right Angle, Shrouded Header 3.81 µm (150 µin.) Tin Mating Plating." https://www.amphenol-cs.com/product/75867231lf.html (accessed 12/10/2025.

Appendix

Entry A: List of RoBIOS-32 Functions

Below is a list of all the functions implemented in the new RoBIOS-32 API:

Unless stated otherwise, all non-void functions return 0 on success and -1 on failure.

Camera:

```
int CAMInit(int resolution);                  // Prepare camera; resolution must be QQVGA.
int CAMGet565(uint16_t* gray565);             // Capture one frame into RGB565 buffer (QQVGA).
int CAMGet565Gray();                          // Capture grayscale encoded as RGB565
int CAMRelease();                             // Tear down camera (no-op/safe). 0 on success.
int CAMGet(BYTE* img);                        // Capture RGB888 (QQVGA*3 bytes) into img.
int CAMGetGray(BYTE* gimg);                   // Capture 8-bit grayscale (QQVGA) into gimg.
BYTE* RGB888ToRGB565(const BYTE* rgb888);     // Convert QQVGA RGB888 → RGB565
```

PSD: (Position Sensitive Devices; time-of-flight distance sensors):

```
int PSDGet(int psd);                          // Read VL53L0X distance in mm (1=front,2=left,3=right)
```

V-Omega Driving Interface: (high-level interface for differential driving)

```
int VWSetSpeed(int linSpeed, int angSpeed);          // Start background drive: linear (mm/s) &
                                                     angular (deg/s)
int VWGetSpeed(double* leftOut, double* rightOut);   // Get filtered wheel speeds (mm/s)
int VWSetPosition(int x, int y, int phi);            // Set odometry (mm, mm, deg)
int VWGetPosition(int* x, int* y, int* phi);         // Get odometry (mm, mm, deg)
int VWRemain();                                      // Remaining straight-line distance (mm) or -1
                                                     if not distance-driving.
int VWWait();                                        // Block until current distance/turn motion
                                                     completes
int VWDone();                                        // Driving state flag: 1 while driving, 0 when
                                                     idle/done.
int VWStraight(int dist, int lin_speed);             // Non-blocking straight drive (mm at mm/s)
int VWTurn(int angle, int ang_speed);                // Blocking on-spot turn (deg at deg/s); returns
                                                     when turn finishes.
int VWCurve(int dist, int angle, int lin_speed);     // Non-blocking arc dist mm, angle deg, mm/s).
int VWDrive(int dx, int dy, int lin_speed);          // Non-blocking curve, dx,dy offset; mm/s.
int VWStalled();                                     // 0=none, 1=left, 2=right, 3=both stalled
                                                     (cmd>0 but speed≈0).
int VWGoTo(int dx, int dy, int lin_speed);           // Turn toward (dx,dy) then drive straight
int VWStop();                                        // Stop all motors and kills any VW task.
```

Motor Control: Low-level motor control and encoders

```
int MOTORDrive(int motor, int speed);         // Direct wheel: motor 1=left, 2=right; speed -100..100
int MOTORPID(int motor, int p, int i, int d); // Set per-wheel PID gains for speed control.
int MOTORPIDOff(int motor);                   // Disable/clear PID for wheel (safe no-op)
```

LCD:

```
int   LCDPrintf(const char* format, ...);                 // printf at current cursor.
int   LCDSetPrintf(int row,int column,const char*,...);   // Move cursor (row,col in px) then printf text.
int   LCDClear();                                         // Clear screen to black.
int   LCDSetPos(int row, int column);                     // Set cursor (row=y, column=x) in pixels.
int   LCDGetPos(int *row, int *column);                   // Get cursor (row=y, column=x). 0 on success.
int   LCDSetColor(COLOR fg, COLOR bg);                    // Set text colors (COLOR = 0xRRGGBB).
int   LCDSetFont(int font, int variation);                // Select TFT_eSPI font index.
int   LCDSetFontSize(int fontsize);                       // Set text scale factor.
int   LCDSetMode(int mode);                               // Reserved/unsupported (returns -1).
int   LCDGetSize(int *x, int *y);                         // Get display width & height in pixels. 0 on
                                                             success.
int   LCDPixel(int x, int y, COLOR col);                  // Draw one pixel (0xRRGGBB).
COLOR LCDGetPixel(int x, int y);                          // Read one pixel color (if supported by
                                                             panel/driver).
int   LCDLine(int x1,int y1,int x2,int y2,COLOR col);     // Draw line.
int   LCDArea(int x1,int y1,int x2,int y2,COLOR col,int fill); // Draw Rectangle
int   LCDCircle(int x,int y,int radius,COLOR col,int fill); // Circle (fill=1 solid, 0 outline).
int   LCDImageStart(int x,int y,int xs,int ys);           // Define target area for subsequent image
                                                             draws.
int   LCDImage(BYTE *img);                                // Draw RGB888 image (QQVGA) into area set by
                                                             LCDImageStart().
int   LCDImage565(const uint16_t* buf);                   // Draw 160×120 RGB565 image at (0,0).
int   LCDImageGray(BYTE *g);                              // Draw 8-bit grayscale image into current area.
int   LCDImageBinary(BYTE *b, uint8_t threshold=128);     // Draw binary (>=thr white, else black) from 8-
                                                             bit gray.
int   LCDRefresh(void);                                   // No-op; TFT refreshes as drawn.
```

Buttons:

```
int KEYGet();                                  // Wait for a key press; returns bitmask
                                               (KEY_LEFT|KEY_RIGHT) after debounce.
int KEYRead();                                 // Instantaneous key read
int KEYWait(int wantedMask);                   // Block until any of wanted buttons are pressed
int KEYReadEdge();                             // Rising-edge bitmask since last call.
```

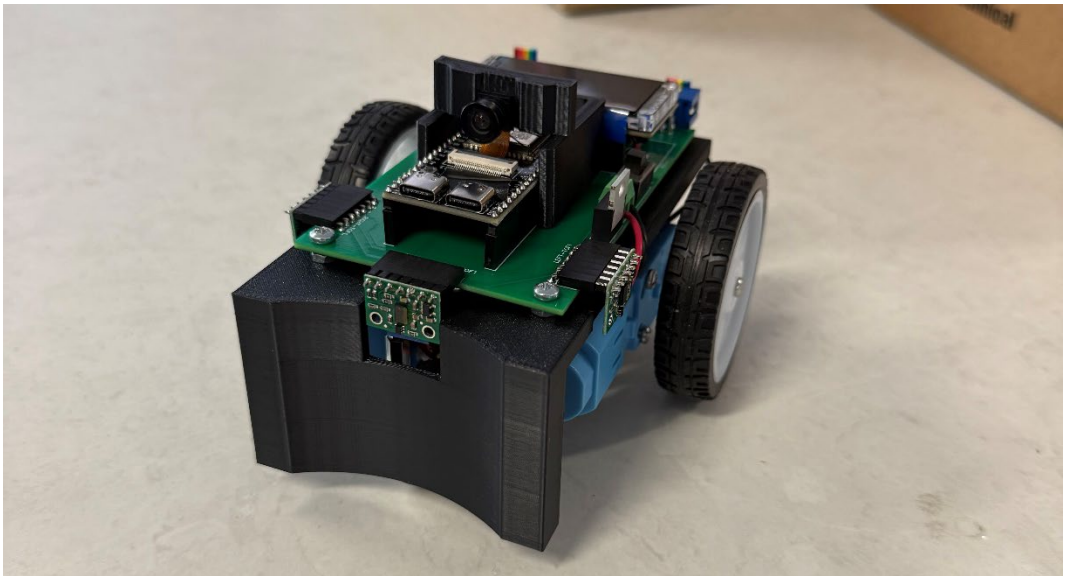
Image Processing:

```
void IPLaplace(BYTE* grayIn, BYTE* grayOut);             // 3×3 Laplacian edge magnitude on 8-bit
                                                            gray (0..255).
void IPSobel(BYTE* grayIn, BYTE* grayOut);               // True Sobel gradient magnitude on 8-bit
                                                            gray (0..255).
void IPCol2Gray(BYTE* imgIn, BYTE* grayOut);             // RGB888 → grayscale (luma) per pixel.
void IPGray2Col(BYTE* imgIn, BYTE* colOut);              // Grayscale → RGB888 (replicate Y into
                                                            R=G=B).
void IPRGB2Col(BYTE* r,BYTE* g,BYTE* b,BYTE* imgOut);    // Pack planar R/G/B channels into
                                                            interleaved RGB888.
void IPCol2HSI(BYTE* img,BYTE* h,BYTE* s,BYTE* i);       // RGB888 → H,S,I (each 0..255) per pixel.
void IPOverlay(BYTE* c1,BYTE* c2,BYTE* cOut);            // Overlay RGB888 c2 on c1 (pure black in
                                                            c2 = transparent).
void IPOverlayGray(BYTE* g1,BYTE* g2,BYTE r,BYTE g,BYTE b,BYTE* cOut); // Colorize mask g2 over gray
                                                            g1 with (r,g,b).
void IPPCol2RGB(COLOR col,BYTE* r,BYTE* g,BYTE* b);      // Split 0xRRGGBB into 8-bit R,G,B.
void IPPCol2HSI(BYTE* imgIn,BYTE* H,BYTE* S,BYTE* I);    // RGB888 → H,S,I (per pixel), pointer alias
                                                            of IPCol2HSI.
void IPPRGB2HSI(BYTE r,BYTE g,BYTE b,BYTE* H,BYTE* S,BYTE* I); // One-pixel RGB → H,S,I (0..255).
void IPRGB565toHSI(uint16_t* src,int w,int h,
                   uint8_t* H,uint8_t* S,uint8_t* I8);       // RGB565 image → H,S,I planes.
void IPRGB565toGray(uint16_t* src,int w,int h,uint8_t* gray); // RGB565 → 8-bit grayscale.
void IP565Adjust(uint16_t* buf,size_t pixels,
                 int bright,float contrast,bool swapBytes); // In-place RGB565 brightness/contrast
void IP565Brightness(uint16_t* buf,size_t pixels,
                     int bright,bool swapBytes);          // RGB565 brightness only.
void IP565Contrast(uint16_t* buf,size_t pixels,
                   float contrast,bool swapBytes);        // RGB565 contrast only.
void IPOpen(uint8_t* binaryImg);                         // 3×3 morphological opening on binary
                                                            (0/1) image, in-place.
void IPAdjust(BYTE* rgb,size_t pixels,
              int bright,float contrast);                // In-place RGB888 brightness/contrast.
void IPBrightness(BYTE* rgb,size_t pixels,int bright);   // RGB888 brightness only.
void IPContrast(BYTE* rgb,size_t pixels,float contrast); // RGB888 contrast only.
```

EyeBot-32 Manual

Designed by Parham Bahrami, supervised by Thomas Bräunl

Contact: 23426998@student.uwa.edu.au
Thomas.Braunl@UWA.edu.au
https://roblab.org/eyebot/



EyeBot-32 is a classroom-ready mobile robot built around the ESP32-S3. It combines a colour camera, three time-of-flight distance sensors, two encoder-driven motors, and a bright TFT so you can *see, think, and move* in real time. The robot is programmed with PlatformIO in C with a tiny API (RoBIOS-style) that hides hardware complexity and lets you focus on autonomy: follow walls, track lines, detect edges, and navigate.

What it is:

- Controller: TTGO T-Display S3
- Camera: OV2640 @ QQVGA for fast vision labs
- Distance sensors: 3× VL53L0X (front/left/right)
- Drive: 2× DC gearmotors + encoders
- UI: 1.9” TFT + two buttons
- Power: dual 18650 protected batteries

These EyeBots have been designed as part of a final year student project – we hope that you find it useful for your learning in ‘Embedded Systems’!

PlatformIO Library Installation Guide:

Steps to install the EyeBot library:

1. Download the zip file of the EyeBot library and the VL53L0X library.
2. Extract both these libraries into your PlatformIO project's library folder.
3. Ensure your .ini file contains the below:

```
[env:lilygo-t-display-s3]
platform = espressif32
board = lilygo-t-display-s3
framework = arduino
monitor_speed = 115200
monitor_filters = esp32_exception_decoder

lib_deps =
   Bodmer/TFT_eSPI
   Wire
   esp32-camera

build_flags =
   -Llib/eyebot/src
   -leyebot
```

Take note of the following:

1. If you use your own TFT library, you can remove the TFT_eSPI entry.
2. You will need to define the T-Display S3 as your board inside of the TFT library, as usual.
3. If you save the library as part of your PlatformIO global libraries instead of a project specific library, you will need to change the first build flag to be the path of your library. For example:

   -L/Users/yourname/.platformio/lib/eyebot/src

Quick Start Guide

EyeBot is equipped with the brand new RoBIOS API, which contains 67 accessible functions which range from driving, button presses, camera usage, image processing and screen writing. These simplify how you program the robot by taking away the brunt work of repetitive tasks such as taking a camera image or driving straight and allow you to build code mostly based on logical building blocks as opposed to hardware integration.

What is RoBIOS?

RoBIOS is a small, stable Application Programming Interface (API) that hides the hardware details so students can write portable robot code fast. On EyeBot-32, we expose the old RoBIOS-style on top of Arduino/FreeRTOS:

- LCD* – text and simple graphics
- CAM* – grab a camera frame (RGB565 colour or grayscale)
- IP* – image processing (Sobel, Laplace, morph ops, grayscale/colour convert)
- VW* – “vehicle” motion primitives (straight/turn/curve, set speed)
- MOTOR* – direct wheel control and PID tuning
- KEY* – read the two onboard buttons
- PSD* – read front/left/right distance sensors in mm

You still write Arduino-style programs with setup()/loop(), but most labs can be solved just calling these functions. You do not need to know the internals of the functions to do well in the labs, only their inputs and outputs and what they do.

The header for the library is called “eyebot.h”. You will need to add this to your library declarations at the top of your main.cpp file.

In order to control the robot, we program the TTGO using the library which sends and receives various signals to control the robot’s actions.

When your sketch boots, as long as you have some RoBIOS function in your code, eyebot.h:

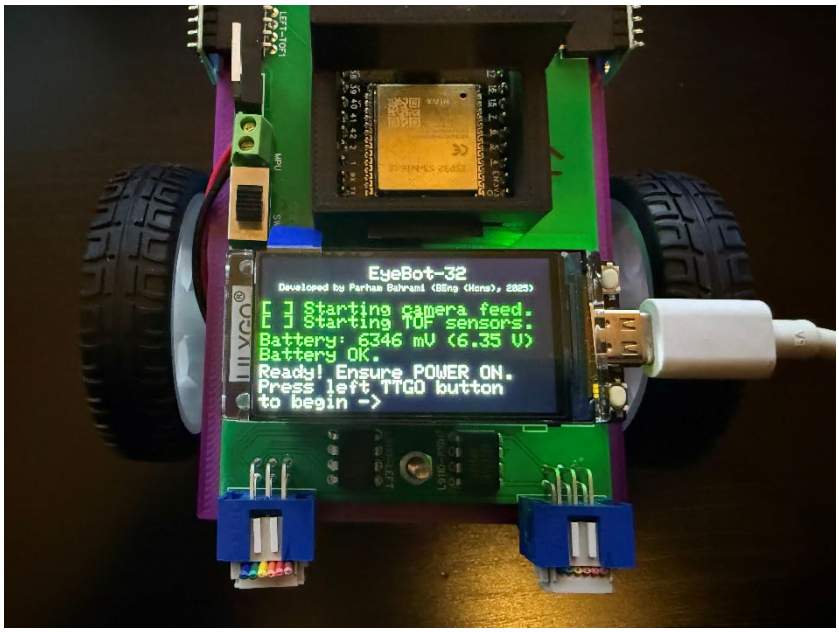
- Powers the board rails and backlight, configures buttons and encoders, and brings the motors to a safe OFF state.
- Initialises SPI (camera), I²C (ToF sensors), and the TFT display; runs a camera sanity check so you see if the feed is live.
- Performs a battery sanity gate with on-screen guidance; you confirm with the left button before anything can move. **If your battery is not on, you cannot run your code.**
- Sets up mutexes/tasks so later drive and sensing functions can run concurrently without you thinking about threads.
- Allows a single button press to begin your code.

This sequence keeps fleets safe in a lab and makes setup deterministic across many units.

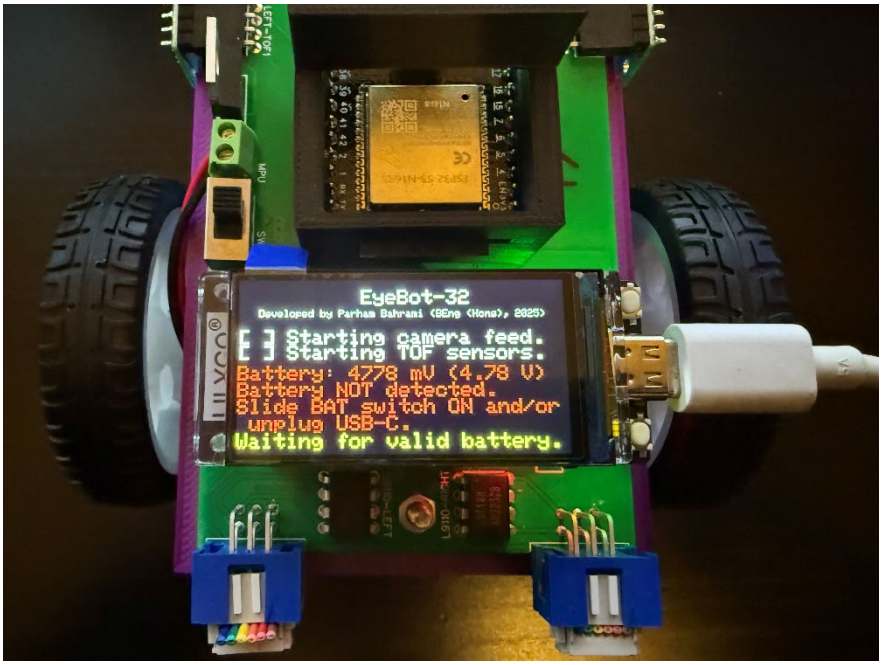
To

```
#include "eyebot.h"
void setup() {
  // your code here
}
void loop() {
  // your code here
}
```

program the robot, simply plug it in using a USB-C cable and upload your code as usual.



Please pay attention to the boot screen which accompanies the initialisation function. On this boot screen, the TTGO will check it has an established connection between the camera module and the distance sensors. If it cannot establish this connection, a warning will come up and the TTGO will enter an infinite loop, requiring you to fix the issue and then restart the TTGO. Please ask a facilitator for help if the issue isn't obvious, but the biggest issues are missing distance sensor or an unplugged camera.



Additionally, the TTGO will be powered on when it is plugged in via USB-C, as well as when it is powered on through the battery switch located above the TTGO, to its left. You may flash code onto the TTGO without powering on the board, however the EyeBot library will prevent you from executing your code if the battery hasn't been turned on. This is to prevent you from power surging your TTGO in the case that your motors are running in your code without the battery. The TTGO on its own is not equipped to power the whole circuit. If you see the red warning saying that the battery

is not detected, simply turn on the TTGO's power and please unplug your USB-C cable. There is no need to reset the TTGO after you turn the battery power on. Another indication that your battery power is not working correctly is the bright red LED underneath the TTGO.

Example Code:

Below is an example of how you would drive towards a wall and stop in time. This is done as a standard C-program.

```
1  #include "eyebot.h"
2
3  int main()
4  { VWSetSpeed(100,0); /* drive */
5    while (PSDGet(PSD_FRONT) > 200) /* wait */ ;
6    VWSetSpeed(0,0); /* stop */
7  }
```

When coding the same in PlatformIO we don't need a 'while'-loop as it is already part of the 'loop' function.

```
1  #include "eyebot.h"
2
3  void setup()
4  { VWSetSpeed(100,0); }  // drive
5
6  void loop(){
7  { if (PSDGet(PSD_FRONT)<200) VWSetSpeed(0,0); }  // stop
```

More example code can be found in the following textbook:

Braunl, T. (2023). *Mobile Robot Programming : Adventures in Python and C* (Second edition). Springer. https://doi.org/10.1007/978-3-031-32797-1

RoBIOS Library:

Unless stated otherwise, all non-void functions return 0 on success and -1 on failure.

Camera:

```
int CAMInit(int resolution);                  // Prepare camera; resolution must be QQVGA.
int CAMGet565(uint16_t* gray565);             // Capture one frame into RGB565 buffer (QQVGA).
int CAMGet565Gray();                          // Capture grayscale encoded as RGB565
int CAMRelease();                             // Tear down camera (no-op/safe). 0 on success.
int CAMGet(BYTE* img);                        // Capture RGB888 (QQVGA*3 bytes) into img.
int CAMGetGray(BYTE* gimg);                   // Capture 8-bit grayscale (QQVGA) into gimg.
BYTE* RGB888ToRGB565(const BYTE* rgb888);     // Convert QQVGA RGB888 → RGB565
```

PSD: (Position Sensitive Devices; time-of-flight distance sensors):

```
int PSDGet(int psd);                          // Read VL53L0X distance in mm (1=front,2=left,3=right)
```

V-Omega Driving Interface: (high-level interface for differential driving)

```
int VWSetSpeed(int linSpeed, int angSpeed);          // Start background drive: linear (mm/s) &
                                                     angular (deg/s)
int VWGetSpeed(double* leftOut, double* rightOut);   // Get filtered wheel speeds (mm/s)
int VWSetPosition(int x, int y, int phi);            // Set odometry (mm, mm, deg)
int VWGetPosition(int* x, int* y, int* phi);         // Get odometry (mm, mm, deg)
int VWRemain();                                      // Remaining straight-line distance (mm) or -1
                                                     if not distance-driving.
int VWWait();                                        // Block until current distance/turn motion
                                                     completes
int VWDone();                                        // Driving state flag: 1 while driving, 0 when
                                                     idle/done.
int VWStraight(int dist, int lin_speed);             // Non-blocking straight drive (mm at mm/s)
int VWTurn(int angle, int ang_speed);                // Blocking on-spot turn (deg at deg/s); returns
                                                     when turn finishes.
int VWCurve(int dist, int angle, int lin_speed);     // Non-blocking arc dist mm, angle deg, mm/s).
int VWDrive(int dx, int dy, int lin_speed);          // Non-blocking curve, dx,dy offset; mm/s.
int VWStalled();                                     // 0=none, 1=left, 2=right, 3=both stalled
                                                     (cmd>0 but speed≈0).
int VWGoTo(int dx, int dy, int lin_speed);           // Turn toward (dx,dy) then drive straight
int VWStop();                                        // Stop all motors and kills any VW task.
```

Motor Control: Low-level motor control and encoders

```
int MOTORDrive(int motor, int speed);         // Direct wheel: motor 1=left, 2=right; speed -100..100
int MOTORPID(int motor, int p, int i, int d); // Set per-wheel PID gains for speed control.
int MOTORPIDOff(int motor);                   // Disable/clear PID for wheel (safe no-op)
```

LCD:

```
int   LCDPrintf(const char* format, ...);                 // printf at current cursor.
int   LCDSetPrintf(int row,int column,const char*,...);   // Move cursor (row,col in px) then printf text.
int   LCDClear();                                         // Clear screen to black.
int   LCDSetPos(int row, int column);                     // Set cursor (row=y, column=x) in pixels.
int   LCDGetPos(int *row, int *column);                   // Get cursor (row=y, column=x). 0 on success.
int   LCDSetColor(COLOR fg, COLOR bg);                    // Set text colors (COLOR = 0xRRGGBB).
int   LCDSetFont(int font, int variation);                // Select TFT_eSPI font index.
int   LCDSetFontSize(int fontsize);                       // Set text scale factor.
int   LCDSetMode(int mode);                               // Reserved/unsupported (returns -1).
int   LCDGetSize(int *x, int *y);                         // Get display width & height in pixels. 0 on
                                                             success.
int   LCDPixel(int x, int y, COLOR col);                  // Draw one pixel (0xRRGGBB).
COLOR LCDGetPixel(int x, int y);                          // Read one pixel color (if supported by
                                                             panel/driver).
int   LCDLine(int x1,int y1,int x2,int y2,COLOR col);     // Draw line.
int   LCDArea(int x1,int y1,int x2,int y2,COLOR col,int fill); // Draw Rectangle
int   LCDCircle(int x,int y,int radius,COLOR col,int fill); // Circle (fill=1 solid, 0 outline).
int   LCDImageStart(int x,int y,int xs,int ys);           // Define target area for subsequent image
                                                             draws.
int   LCDImage(BYTE *img);                                // Draw RGB888 image (QQVGA) into area set by
                                                             LCDImageStart().
int   LCDImage565(const uint16_t* buf);                   // Draw 160×120 RGB565 image at (0,0).
int   LCDImageGray(BYTE *g);                              // Draw 8-bit grayscale image into current area.
int   LCDImageBinary(BYTE *b, uint8_t threshold=128);     // Draw binary (>=thr white, else black) from 8-
                                                             bit gray.
int   LCDRefresh(void);                                   // No-op; TFT refreshes as drawn.
```

Note – you may still use the TFT functions, and an instance of the screen is already initialised as an extern variable in the library, called tft. Note not all of the functions have an operation, they are designed to replicate exactly the old EyeBot. Please note the difference between the different LCDImage functions.

Buttons:

```
int KEYGet();                                  // Wait for a key press; returns bitmask
                                                  (KEY_LEFT|KEY_RIGHT) after debounce.
int KEYRead();                                 // Instantaneous key read
int KEYWait(int wantedMask);                   // Block until any of wanted buttons are pressed
int KEYReadEdge();                             // Rising-edge bitmask since last call.
```

These functions use the two buttons on the TTGO. Note that KEY_LEFT and KEY_RIGHT have been defined in the library and can be used with these functions.

Image Processing:

Basic image processing functions using the previously set camera resolution are included in the RoBIOS library. For more complex functions use the OpenCV library.

```
void IPLaplace(BYTE* grayIn, BYTE* grayOut);            // 3×3 Laplacian edge magnitude on 8-bit
                                                           gray (0..255).
void IPSobel(BYTE* grayIn, BYTE* grayOut);              // True Sobel gradient magnitude on 8-bit
                                                           gray (0..255).
void IPCol2Gray(BYTE* imgIn, BYTE* grayOut);            // RGB888 → grayscale (luma) per pixel.
void IPGray2Col(BYTE* imgIn, BYTE* colOut);             // Grayscale → RGB888 (replicate Y into
                                                           R=G=B).
void IPRGB2Col(BYTE* r,BYTE* g,BYTE* b,BYTE* imgOut);   // Pack planar R/G/B channels into
                                                           interleaved RGB888.
void IPCol2HSI(BYTE* img,BYTE* h,BYTE* s,BYTE* i);      // RGB888 → H,S,I (each 0..255) per pixel.
void IPOverlay(BYTE* c1,BYTE* c2,BYTE* cOut);           // Overlay RGB888 c2 on c1 (pure black in
                                                           c2 = transparent).
void IPOverlayGray(BYTE* g1,BYTE* g2,BYTE r,BYTE g,BYTE b,BYTE* cOut); // Colorize mask g2 over gray
                                                           g1 with (r,g,b).
void IPPCol2RGB(COLOR col,BYTE* r,BYTE* g,BYTE* b);     // Split 0xRRGGBB into 8-bit R,G,B.
void IPPCol2HSI(BYTE* imgIn,BYTE* H,BYTE* S,BYTE* I);   // RGB888 → H,S,I (per pixel), pointer alias
                                                           of IPCol2HSI.
void IPPRGB2HSI(BYTE r,BYTE g,BYTE b,BYTE* H,BYTE* S,BYTE* I); // One-pixel RGB → H,S,I (0..255).
void IPRGB565toHSI(uint16_t* src,int w,int h,
                   uint8_t* H,uint8_t* S,uint8_t* I8);      // RGB565 image → H,S,I planes.
void IPRGB565toGray(uint16_t* src,int w,int h,uint8_t* gray); // RGB565 → 8-bit grayscale.
void IP565Adjust(uint16_t* buf,size_t pixels,
                 int bright,float contrast,bool swapBytes); // In-place RGB565 brightness/contrast
void IP565Brightness(uint16_t* buf,size_t pixels,
                     int bright,bool swapBytes);         // RGB565 brightness only.
void IP565Contrast(uint16_t* buf,size_t pixels,
                   float contrast,bool swapBytes);       // RGB565 contrast only.
void IPOpen(uint8_t* binaryImg);                        // 3×3 morphological opening on binary
                                                           (0/1) image, in-place.
void IPAdjust(BYTE* rgb,size_t pixels,
              int bright,float contrast);               // In-place RGB888 brightness/contrast.
void IPBrightness(BYTE* rgb,size_t pixels,int bright);  // RGB888 brightness only.
void IPContrast(BYTE* rgb,size_t pixels,float contrast); // RGB888 contrast only.
```