GENG4412 Engineering Research Project Part 2

A camera-equipped, ESP32-S3 microcontroller-based mobile robot design capable of autonomous navigation via visual lane detection

Benjamin Wright

22244602

School of Engineering, University of Western Australia

Supervisor: Professor Thomas Bräunl

School of Engineering, University of Western Australia

Co-Supervisor: Kieran Quirke-Brown

School of Engineering, University of Western Australia

Word count: 7,980 words

School of Engineering
University of Western Australia

Submitted: October 14th, 2024

Project Summary

Recent improvements in low-cost microcontrollers have made it viable for camera-equipped autonomous mobile robots to be designed around them as the central brain of the system. This is opposed to using relatively more expensive single-board computers such as the Raspberry Pi. This project set out to redesign a small mobile robot currently based on the Raspberry Pi, known as the EyeBot 8, around the ESP32-S3 based LILYGO® T-Display-S3 microcontroller instead. The current EyeBot 8 is used for educational purposes at the University of Western Australia (UWA), primarily in their Embedded Systems unit (unit code ELEC3020). This new T-Display-S3 EyeBot design supports a core subset of the EyeBot 8's features, such as processing video from an on-board camera, drawing to an integrated touchscreen display, reading button and touch inputs, reading from a distance sensor, and driving itself using a pair of wheeled motors. This subset of the EyeBot 8's features supported in the T-Display-S3 EyeBot is sufficient to meet the demands of the Embedded Systems unit. Writing programs for this T-Display-S3 EyeBot has been made easier via its main software library implementing an API known as RoBIOS-7. This API provides users access to the EyeBot's major hardware and software features using simple function calls in C/C++. This EyeBot redesign would allow UWA to replace their fleet of EyeBot 8's with cheaper and simpler versions that are able to meet the same demands of the engineering course units that utilise them. For this design project, a previously designed chassis equipped with wheeled motors lacking encoders, an analogue short-range distance sensor, and a breadboard mounted on top was used. The remaining choices regarding software and hardware components were made within these hardware constraints. This design project also sought to discover whether autonomous navigation via visual lane detection could be feasibly implemented for the T-Display-S3 EyeBot to discover if the computing power of the T-Display-S3 microcontroller, or the main software library's RoBIOS-7 API implementation, prevented the T-Display-S3 EyeBot from being able to perform complex tasks beyond the demands of the Embedded Systems unit. We discovered however that the T-Display-S3 EyeBot could comfortably perform this task, as an autonomous navigation program was successfully written for it using a computationally cheap form of line segment detection that identifies lane markings within the view of the EyeBot's camera. With this program the T-Display-S3 EyeBot was able to complete full consecutive laps of the UWA Robotics Lab's test circuit. This research and development for computationally cheap lane-based navigation and line segment detector techniques can hopefully contribute to the UWA Robotics Lab's body of knowledge for future software development.

Acknowledgements

I would like to thank both Professor Thomas Bräunl and Kieran Quirke-Brown for assisting me in solving hardware and electrical issues that arose during the T-Display-S3 EyeBot design process.

Table of Contents

List of Figures

List of Tables

1. Context

1.1 Introduction

As semiconductor technology has advanced, it has become increasingly more feasible for microcontrollers to take over the roles in electronics once dominated by more complex and relatively more powerful single board computers (such as the Raspberry Pi). Improvements in the ease in which microcontrollers can be programmed have also been accompanied by significant improvements in computational power and in the hardware features they can boast, such as responsive touch-displays and Wi-Fi connectivity. The ESP32 series of microcontrollers are particularly popular for their extensive and practical feature-set, compact form factor, ease of integration with other electronics, and low price point when compared to single board computers. The Embedded Systems unit (unit code ELEC3020) at the University of Western Australia (UWA) teaches the concepts behind the hardware and software of embedded systems primarily through lab work involving ESP32 microcontrollers. The main ESP32 microcontroller used throughout the unit has been the LILYGO® T-Display, featuring heavily in the practical lab work and assignments, with the skills acquired by the students learning to use it being transferable to many other microcontrollers. The student experience within the Embedded Systems unit however has been dampened by the practical work involving the EyeBot 8.

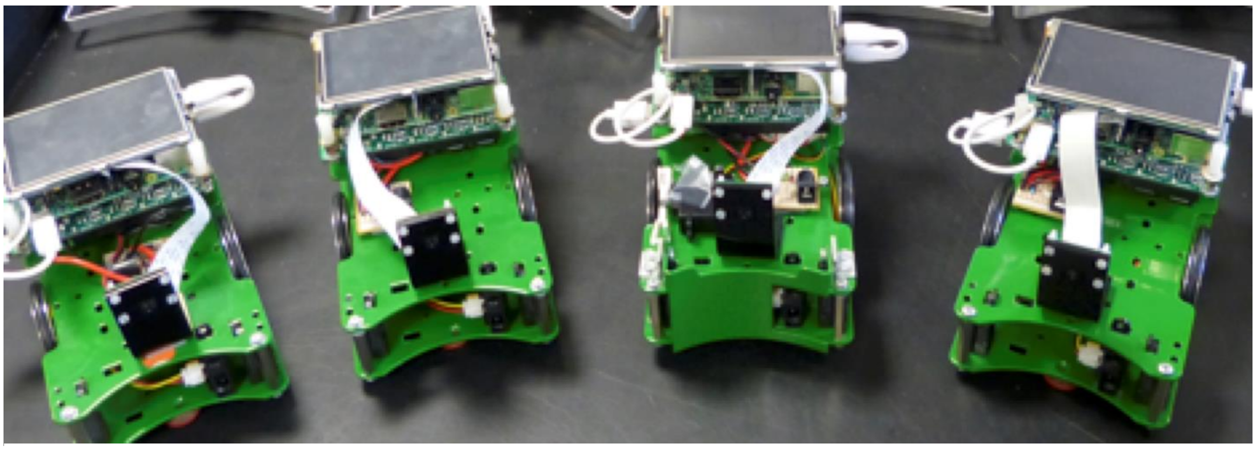
Figure 1: An image of four EyeBot 8's [1].

The EyeBot 8 [1] (depicted in Figure 1) is a programmable mobile robot complete with a pair of motors, a touch-display, a camera, and an array of distance sensors. It has been used in the Embedded Systems unit to provide students the opportunity to test the theory they have been taught regarding software control systems. This usually involves them programming the EyeBot 8 to complete specific challenges, such as writing a program whereby the EyeBot 8 identifies a red object in the view of its camera and proceeds to drive towards it, stopping before colliding with any objects based off its distance sensor readings. The EyeBot 8 implements an Application Programming Interface (API) known as RoBIOS-7 [2] that students can use to control its major hardware and software features via simple C/C++ function calls in their programs. While the features and capabilities of the EyeBot 8 are technically sufficient to meet the demands of the Embedded Systems unit, the hurdles it raises for the experience of students are a result of it being designed around the Raspberry Pi 3 Model B. The most significant of these hurdles raised by the Raspberry Pi is the process of programming the EyeBot 8 itself. Due to the Raspberry Pi requiring a Linux-based operating system, the process involves first powering on the EyeBot 8 and waiting for its operating system and Wi-Fi access point to boot up. The user then must read the Wi-Fi access point's SSID and password from the EyeBot 8's display, and then connect to the EyeBot 8's unique Wi-Fi access point from their own computer. They then must open a terminal on their computer and remotely log into the EyeBot 8 via SSH. Within that terminal session they must then either write or copy over the desired C/C++ program into the Raspberry Pi's filesystem. They then compile their program using a custom compiler command available only on the EyeBot 8, and finally must manually execute their program from the command line. If the EyeBot 8 is powered down and then restarted, this process must then be repeated. For students in the Embedded Systems unit who are inexperienced with Linux operating systems and the command line, this convoluted process is usually daunting and difficult. The EyeBot 8 relies on many custom-made hardware components, and to accommodate the greater power demands of the Raspberry Pi and the rest of the electronic components, the EyeBot 8 also requires a bulky power supply. Hardware factors such as these contribute to making the EyeBot 8 difficult for Embedded System unit facilitators to maintain.

These and other issues with the EyeBot 8's design prompted the UWA Robotics Lab (who supply the EyeBot 8's to the Embedded Systems unit) to investigate whether an EyeBot re-design could solve these usability and maintainability issues, at a price point that could warrant the current fleet of EyeBot 8's being replaced. It was noticed by the Lab that much of the functionality performed by the Raspberry Pi in the EyeBot 8 could theoretically be performed by an ESP32 microcontroller. Due to their prior positive experience with the LILYGO® T-Display series, the latest model, the LILYGO® T-Display-S3, was proposed as the new brain for this ESP32 EyeBot design. The T-Display-S3 is based on an updated ESP32-S3 with a dual-core 240 MHz CPU [3] and 8 MB of RAM [4], alongside boasting an integrated touchscreen display that's relatively large for a microcontroller. The process of programming a T-Display-S3 is also greatly simplified due to its inclusion in the Integrated Development Environment program Arduino IDE. Writing a program for the T-Display-S3 simply involves a student downloading Arduino IDE onto their computer, configuring it once, connecting the T-Display-S3 to the computer via a USB cable, writing their C/C++ program within Arduino IDE, and with one button they can compile and upload their program to the T-Display-S3, which automatically runs when the device is powered on. This operating-system agnostic process is greatly superior to the process of programming the EyeBot 8 and would immediately improve the student experience, conforming well with the rest of the coursework in the Embedded Systems unit that also involves programming ESP32 microcontrollers. There also existed a simple avenue to implement the RoBIOS-7 API for the T-Display-S3 as a library accessible via Arduino IDE. The T-Display-S3 can be powered through its USB-C port by at least a 5V/1A power source, and its footprint allows it to be comfortably mounted on a small breadboard, with its General-Purpose Input and Output (GPIO) pins allowing simple wired and programmable connections with other electronic components. As a result of these factors, a T-Display-S3 EyeBot promised to be a significant upgrade over the EyeBot 8 in terms of cost, usability, and maintainability.

1.2 Project Objectives

This project primarily seeks to benefit the UWA Robotics Lab by offering them the opportunity to upgrade their fleet of EyeBot 8's to cheaper and more maintainable models that retain a feature set sufficient for its educational purpose in the Embedded Systems unit. The new T-Display-S3 EyeBot programmable through Arduino IDE should improve the educational experience of students and others using it. It is also hoped that the T-Display-S3 EyeBot will also be able to support complex applications beyond even the demands of the Embedded Systems unit's content so that it can potentially be utilised in more challenging contexts. The primary test for the extent of this T-Display-S3 EyeBot's capabilities will be an autonomous navigation challenge involving visual lane detection. The UWA Robotics Lab hosts a small-scale circuit marked by white street-lane markings for the purpose of testing the lane detection implementations of various small autonomous robots (as depicted in Figure 2). Successful navigation of this test circuit would signify that the new EyeBot is suitable for applications of potentially greater complexity. In the case of failure, however, the goal would still be to demonstrate that the T-Display-S3 EyeBot and its implementation of RoBIOS-7 API is sufficient to support the lighter demands of the Embedded Systems unit's educational content. This would be achieved by writing a program whereby the user can select a specific colour and threshold for the T-Display-S3 EyeBot so that it can seek and drive towards the largest object in its view whose colour is within the threshold, stopping before colliding with any objects head-on.

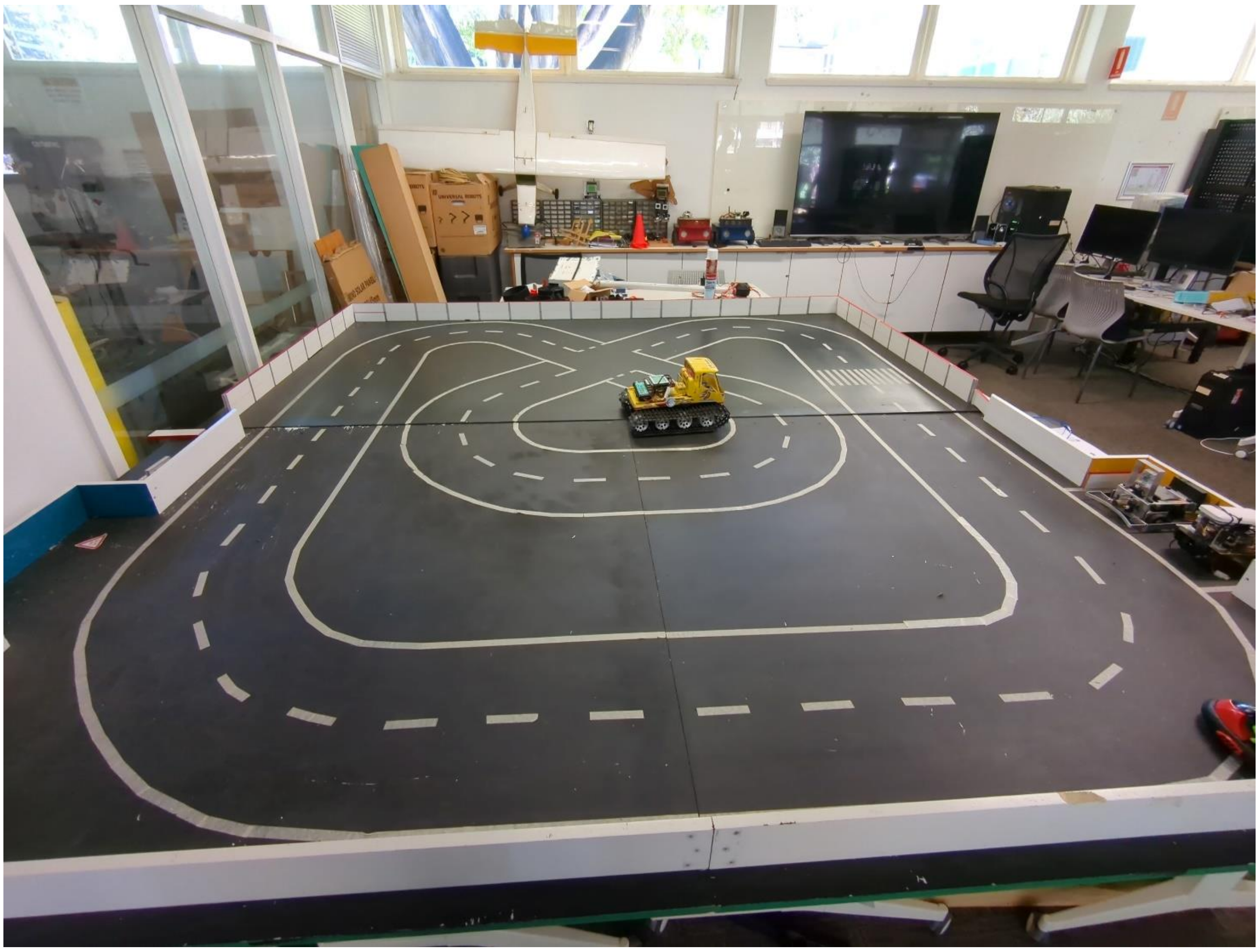
Figure 2: Image of the UWA Robotics Lab's test circuit.

1.3 Preliminary Work

Prior to this specific project commencing, work had already begun within the UWA Robotics Lab to design some of the hardware and software components of the new T-Display-S3 EyeBot. The most significant contribution of this prior work was a custom chassis already integrated with some key electronic components. The chassis consisted of two identical rectangular plates with rounded ends sandwiched together, separated by adjustable spacers. Between the two plates were a pair of DC powered, 3-6 V dual-axis TT gear motors with wheels attached on the EyeBot's sides, a Sharp GP2Y0A41SK0F analogue short-range infrared distance sensor mounted at the front of the EyeBot facing forwards, and space in between the motors and the sensor for a slim USB battery bank. A 5V/1A USB battery bank was initially provided as well. On the underside of the chassis' bottom plate, directly beneath the distance sensor, was a metal ball caster that provided the EyeBot a third point of contact with the ground along the EyeBot's centreline. Fixed on top of the EyeBot chassis' top plate was a breadboard with a LILYGO® T-Display originally mounted on it, as well as a L298N dual-channel motor driver connected to the T-Display sitting loose on top. The original state of the T-Display EyeBot design can be seen in Figure 3.

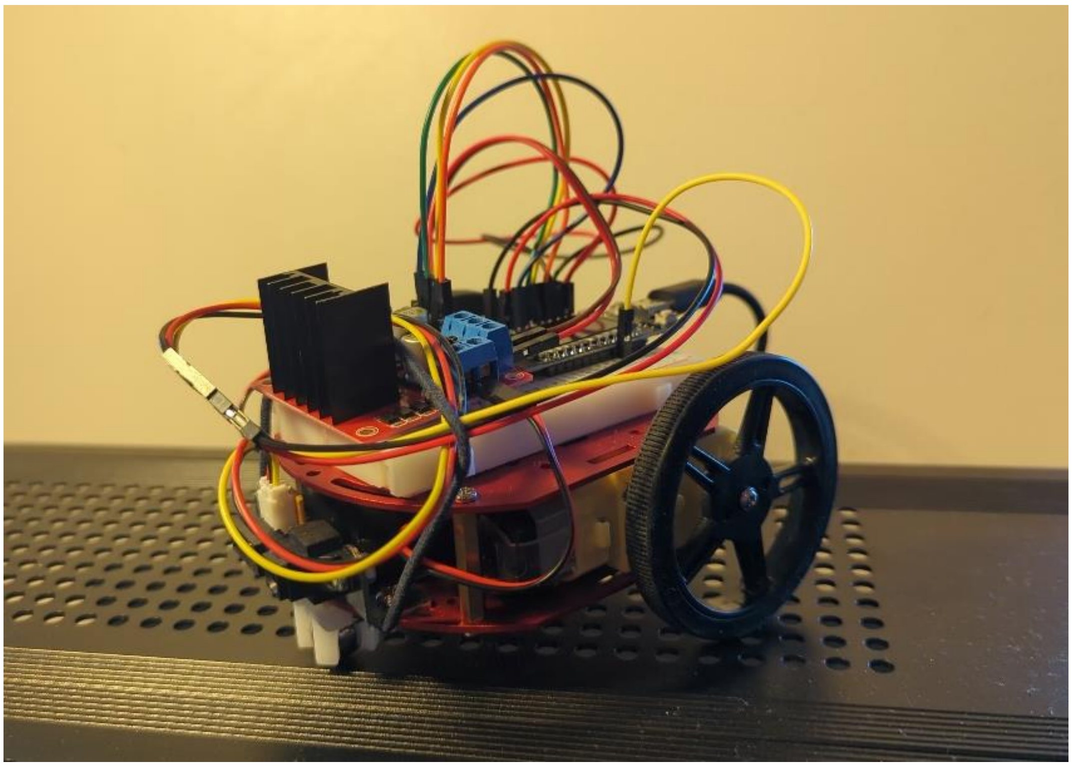

Figure 3: Angled view of the original T-Display EyeBot design.

Since the end goal for the EyeBot was also for it to be camera-equipped, standalone experiments had been performed to discover a method of reliable image transfer between a T-Display and a camera. The camera the UWA Robotics Lab advocated for in the design was the Ai-Thinker ESP32-CAM. The resolution of the images captured by the ESP32-CAM was only required to be QQVGA (or 160 by 120 pixels). However, methods of image transfer devised by that point between the T-Display and the ESP32-CAM, such as asynchronous serial wired communication or the ESP-NOW wireless communication protocol, were deemed insufficient by the supervisors due to their low rate of image transfer (measured in images-per-second). A rate of RGB565 pixel-format QQVGA-resolution image transfer identified by the supervisors as sufficient for responsive vision-based navigation logic was approximately 7 images-per-second, and these prior implementations were unable to attain that.

1.4 Literature Review

Finding existing literature relevant to the unique hardware and software design problems of this project was challenging. Searches across the databases of UWA's OneSearch, ACM Digital Library and IEEE Xplore using various combinations of the search terms "microcontroller", "esp32", "self-driving", "autonomous" and "mobile robot" yielded only a couple of noteworthy design reports from recent years. Shobika *et al.* [5] developed a four-wheeled autonomous robot equipped with a time-of-flight sensor that was centrally controlled with an ESP32, the only ESP32 based project that was found. Since it lacked image processing however there was virtually no information applicable to this project that could be drawn from it. Ueyama *et al.* [6] designed an Arduino microcontroller based autonomous mobile robot equipped with a laser range finder for a similar undergraduate course to the Embedded Systems unit. The most valuable conclusions it drew were that microcontrollers such as Arduino's were preferable over Raspberry Pi's for educational purposes due to their ease of programming through Arduino IDE, according to a student survey they conducted upon the completion of the course [6]. Most of the literature discovered within these search parameters however were concerned with designs where microcontrollers were only responsible for sub-systems such as the motors in mobile robots, as opposed to the central system control required for this project.

Much of the software implementation for the RoBIOS-7 API did not warrant a special literature review due to its simplicity, as well as the fact that some of the more complex logic that is required (e.g. image processing techniques such as Laplace and Sobel edge detection) will be largely ported directly from the EyeBot 8 implementation. The primary research undertaken for the literature review component of this project however was for the extension challenge involving visual-based lane detection and navigation methods, with a focus on identifying techniques that could feasibly be implemented and performed on a T-Display-S3 with the greatest computational efficiency and the smallest memory demands. A survey of the state-of-the-art for automatic lane detection conducted by Zaidi *et al.* in 2023 [7] identified five categories of lane detection techniques: colour thresholding, gradient based, Hough transform, edge detection, and convolutional neural networks (CNN). CNNs, though regarded as the most accurate method [7], are immediately ruled out from consideration due to their dependence on GPU hardware, significant storage and memory demands, and reliance on substantial high quality training data [7] [8] [9], which are not feasible for this project. Techniques depending on the Hough transform for line detection as well, even though it is considered to be general purpose and reliable, will similarly be relegated to a method of last resort due to the apparent consensus regarding its high computational and memory cost [7] [8] [9]. On the other hand, colour thresholding is classed as the least resource intensive of the techniques [7], and the survey puts forward a method devised in 2005 by Chiu and Lin titled "Colour-Based Segmentation" that boasts resilience to varying lighting conditions and does not rely on the Hough transform for line detection [10]. As such, this technique was prioritized for implementation on the T-Display-S3. As for gradient based techniques, there does not appear to be much representation for them in the database searches conducted outside of those explicitly listed in the survey, and the survey seems to prefer other technique categories over it [7]. Edge detection techniques, on the other hand, are heavily represented in the wider literature. The variations of the technique each share a similar processing pipeline involving first image pre-processing to reduce visual noise, identifying pixels that significantly contrast against their neighbours and most likely form an edge, line detection to map start and end points to contiguous edges, and then optional line tracking (hysteresis) for when edges are temporarily occluded or undetectable between image frames [7] [11]. Each of these stages are relatively modular, and thus its possible for a customised pipeline to be developed with stages suitable for the T-Display-S3. The most computationally expensive of these pipeline stages however is the line detection algorithm. A survey of non-CNN line detection algorithms conducted by Shi *et al.* in 2023 [8] concluded in their tests that the most computationally efficient method with a small trade off in accuracy was the Enhanced Line Segment Drawing (ELSED) algorithm. ELSED was developed specifically for "very low-end hardware" [9, p. 1], and thus ELSED was also prioritised behind the colour thresholding technique in the alternate implementations for the lane detection and navigation program. Another line detection algorithm that wasn't identified in the various literature surveys however was developed by Yilmaz and Baykal and is titled the "Ultrafast Line Detector" [12]. Published in 2022, it boasts itself to be 14.88 times faster than EDLines [12] (the second fastest line detector behind ELSED [8]), as opposed to ELSED only claiming to be two times faster than EDLines [9]. This is via an unconventional look-up table that's around 64 KB in size, which is suitable for the T-Display-S3's memory constraints. The "Ultrafast" line detector method was thus prioritised over both the colour thresholding method and ELSED due to a combination of the project's time constraints, the relative simplicity of its algorithm compared to ELSED, and the more general applicability of line detectors than colour thresholding techniques beyond lane navigation problems. Each of these three selected lane detection methods based on their algorithms appear to have a time complexity of approximately $O(N)$, with N being the number of pixels in the image's region of interest. Due to the time limitations of this project, only these three competing methods were considered. Unfortunately, none of these three methods in their source literature shared a common accuracy and speed benchmark which their claims can be compared to each other with.

After potential lanes have been detected, either the EyeBot's relative position in between the lanes or the trajectory of the lanes must be estimated. A simple lane departure warning system that was found and could be adapted as the guidance system for the T-Display-S3 EyeBot was developed by Gamal *et al* [13], which is dependent on strong assumptions being made about the orientation and perspective of the camera. Considering the well-defined and limited environment the EyeBot is expected to operate in for the navigation of the test circuit, as well as the constraints of the T-Display-S3 EyeBot design itself, these can be safely assumed.

2. Design Approach

2.1 Constraints

As already noted in Section 1.3 of this report, preliminary work on the hardware of the new EyeBot had provided a chassis complete with TT gear motors lacking encoders, a distance sensor, and a breadboard. For the sake of saving time, it was decided at the commencement of this project that the addition of all other hardware components required for the full functionality of the T-Display-S3 EyeBot should fit within the provided chassis' form-factor. There was freedom however in the replacement of other provided components such as the L298N motor driver and the 5V/1A USB battery bank. It was also settled at the commencement of the project that the original T-Display provided with the chassis would be replaced with the LILYGO® T-Display-S3 (the touchscreen variant if it could be accommodated), and that the on-board camera should be the Ai-Thinker ESP32-CAM. The guiding criterion for further hardware component choices was the minimisation of costs to keep the total cost of the T-Display-S3 EyeBot design below that of the EyeBot 8, and the simplicity of hardware integration to make the T-Display-S3 EyeBot easier to construct and maintain than the EyeBot 8. All wiring connections between the various electronic components of the T-Display-S3 EyeBot were also to be through the breadboard.

As for the software of the new EyeBot, the primary expectation was for the entire RoBIOS-7 API [2] to be ported to the T-Display-S3 and be accessible as a library through Arduino IDE. Since the number of hardware and software features of the T-Display-S3 EyeBot do not match that of the EyeBot 8 however (such as the T-Display-S3 EyeBot not having a filesystem), not all the exposed API functions that assume these extra features were expected to be implemented. The only API functions expected to be implemented were those relating to drawing and printing to the display, receiving button and touch input, retrieving images from the camera, processing images, reading measurements from the distance sensor, and controlling the motors through the V-Omega driving interface. This subset was chosen due to it being sufficient to support programs that could meet the demands of the Embedded Systems unit. All other remaining API functions were permitted to return error codes when invoked to indicate they were unimplemented. Due to some of the unique characteristics of the T-Display-S3 EyeBot however, permission was also granted to add new functions to the API that followed similar conventions to the other functions.

2.2 Success Criteria

The following success criteria for the project are listed in descending priority order:

1. Port the entire RoBIOS-7 API to the T-Display-S3 as a software library, used by including a single C/C++ header file and linked to through Arduino IDE. Functional implementations should at least be provided for those API functions relating to drawing to the display, receiving button and touch input, retrieving images from the camera, processing images, reading measurements from the distance sensor, and controlling the motors through the V-Omega driving interface. All other remaining functions in the API are expected to compile but return error codes when invoked.
2. Write a program for the T-Display-S3 EyeBot using the RoBIOS-7 API that involves the EyeBot seeking out and driving towards the largest object in the view of its camera whose colour is within a specified threshold, stopping before colliding with any object head-on. The user must be able to select the desired colour and threshold values.
3. Write at least one program for the new EyeBot using the RoBIOS-7 API that involves the EyeBot autonomously navigating at least one full lap of the UWA Robotics Lab's test circuit (as seen in Figure 2) via visual lane detection.

The first two success criteria were necessary for the entire project to be deemed a success. If the first criteria were deemed to be completed but the second was not, then it would indicate that the port of the RoBIOS-7 API would be insufficient for the demands of the T-Display-S3 EyeBot's use in the Embedded Systems unit. The program described in the second criterion is similar in difficulty to the most challenging assignment involving the EyeBot 8 currently presented to students in the Embedded Systems unit. The third criterion on the other hand was more of an extension challenge, since it was about discovering what the T-Display-S3 EyeBot was computationally capable of. Failure to complete this criterion would not have ruled the whole project as a failure, but it would indicate that the computational capability of the T-Display-S3 EyeBot would be limited beyond its use in the Embedded Systems unit.

2.3 Process

2.3.1 RoBIOS-7 API Implementation

Porting the RoBIOS-7 API to the T-Display-S3 was a simple process of working through the list of functions that were required to be implemented. The problem of reliably retrieving images from the ESP32-CAM was allocated the most time to solve, since having access to the camera is the most important feature of the EyeBot. The transfer of a QQVGA resolution image from the ESP32-CAM to the T-Display-S3 had already proven itself to be the most significant performance bottleneck for the EyeBot (based off the preliminary results of separate experiments involving asynchronous serial wired and ESP-NOW wireless communication), so an alternative method of transfer that would be significantly superior to previously tested methods was sought after. This problem also involved writing a specialised image capture and transfer program specifically for the ESP32-CAM. The solution devised for this problem would also dictate the wiring or lack thereof between the T-Display-S3 and the ESP32-CAM. The image transfer system was to be deemed sufficient if it could reliably achieve an image transfer rate from the ESP32-CAM to the T-Display-S3 of at least seven images-per-second and could maintain continuous image transfer even when all the T-Display-S3's other electronic components were in full operation.

The next set of API functions that were prioritised were those that involved drawing and printing to the T-Display-S3's display, with the success criteria simply to have the output to the T-Display-S3's screen conform to the individual function's specifications. The image processing, button and touch input functions were likewise expected to conform to the API's specifications to be deemed a success.

The next most important subset of RoBIOS-7 functions to be implemented were the V-Omega driving interface functions. The V-Omega driving interface is designed to hide the details of the electrical control signals required to be sent from the T-Display-S3 to the motor driver to control the motor pair's individual speeds and directions behind commands where the linear and angular speed of the whole EyeBot can be dictated in mm/s and degrees/s respectively. This mapping from requested linear and angular speeds to output Pulse-Width Modulation (PWM) signals for the motor driver required experimentation to find approximations suitable for the demands of the EyeBot in its educational setting. These experiments simply involved supplying the maximum output PWM signals from the T-Display-S3 to both motors in the same direction to discover the maximum linear speed and supplying the maximum PWM signal to one of the motors with the other stationary to discover the maximum angular speed. The wiring and control signals required between the T-Display-S3 and the motor driver was to be dictated by the final choice of motor driver for the T-Display-S3 EyeBot design. The criterion for successful implementation was if the EyeBot could consistently manoeuvre itself towards the right direction and orientation at approximately the linear and angular speed requested, as well as the distance if requested, in the call to V-Omega driving interface function.

The final subset of API functions to be implemented were related to the distance sensor. For the Sharp GP2Y0A41SK0F, the relationship between the distance detected and the analogue output value was discovered experimentally by placing obstructions in front of the T-Display-S3 EyeBot's distance sensor at varying distances and recording the analogue output values across the range. The wiring between the distance sensor and the T-Display-S3 was also determined at this stage. The criterion for success was deemed to be the distance sensing functions providing approximately accurate measurements relevant to the feasible range that can be detected by the analogue distance sensor.

2.3.2 Colour-Tracking Navigation Program

Once the T-Display-S3 EyeBot library had a reliable implementation for the required subset of the RoBIOS-7 API, the goal was then to develop the colour-tracking navigation program described in project success criterion 2 of Section 2.2 using the API. The project would not be allowed to proceed until this criterion was accomplished, either by improving the program, improving the API implementation, or improving the hardware design of the T-Display-S3 EyeBot.

2.3.3 Visual Lane Detection and Navigation Program

The final stage of the project was to develop an autonomous navigation program via visual lane detection as described in project success criterion 3 of Section 2.2. Between the three methods of lane detection identified as being possibly viable for the T-Display-S3 EyeBot in the literature review of Section 1.4, development was prioritised in descending order like so:

1. Ultrafast Line Detector [12]
2. Colour-Based Segmentation [10]
3. ELSED [9]

The Ultrafast Line Detector method was prioritised first since it boasted to be the fastest and most memory efficient line detector amongst the line detector algorithms that were discovered during the literature review. Parts of its pipeline such as the Gaussian Blur and the Canny Edge Detector are also extractable and possibly useful in other software projects in the UWA Robotics Lab. Upon completion of the Ultrafast Line Detector version of the program and a demonstration that it could or could not be used to navigate a full lap of the test circuit (and if there existed time-remaining within the project), the Colour-Based Segmentation version would be embarked upon. This would be for the purpose of comparing the performance claims of colour-thresholding techniques to line detection techniques. ELSED was relegated to the method of least priority for development, even though it was the fastest peer-reviewed method discovered in the line detector literature surveys, due its greater complexity compared to the Ultrafast Line Detector. It would only be considered if the Ultrafast Line Detector and the Colour-Based Segmentation techniques were determined to be insufficient for lane detection in practice. Each of these programs were also expected to interface with the hardware of the T-Display-S3 EyeBot through the RoBIOS-7 API.

3. Results and Discussion

3.1 Final T-Display-S3 EyeBot Design

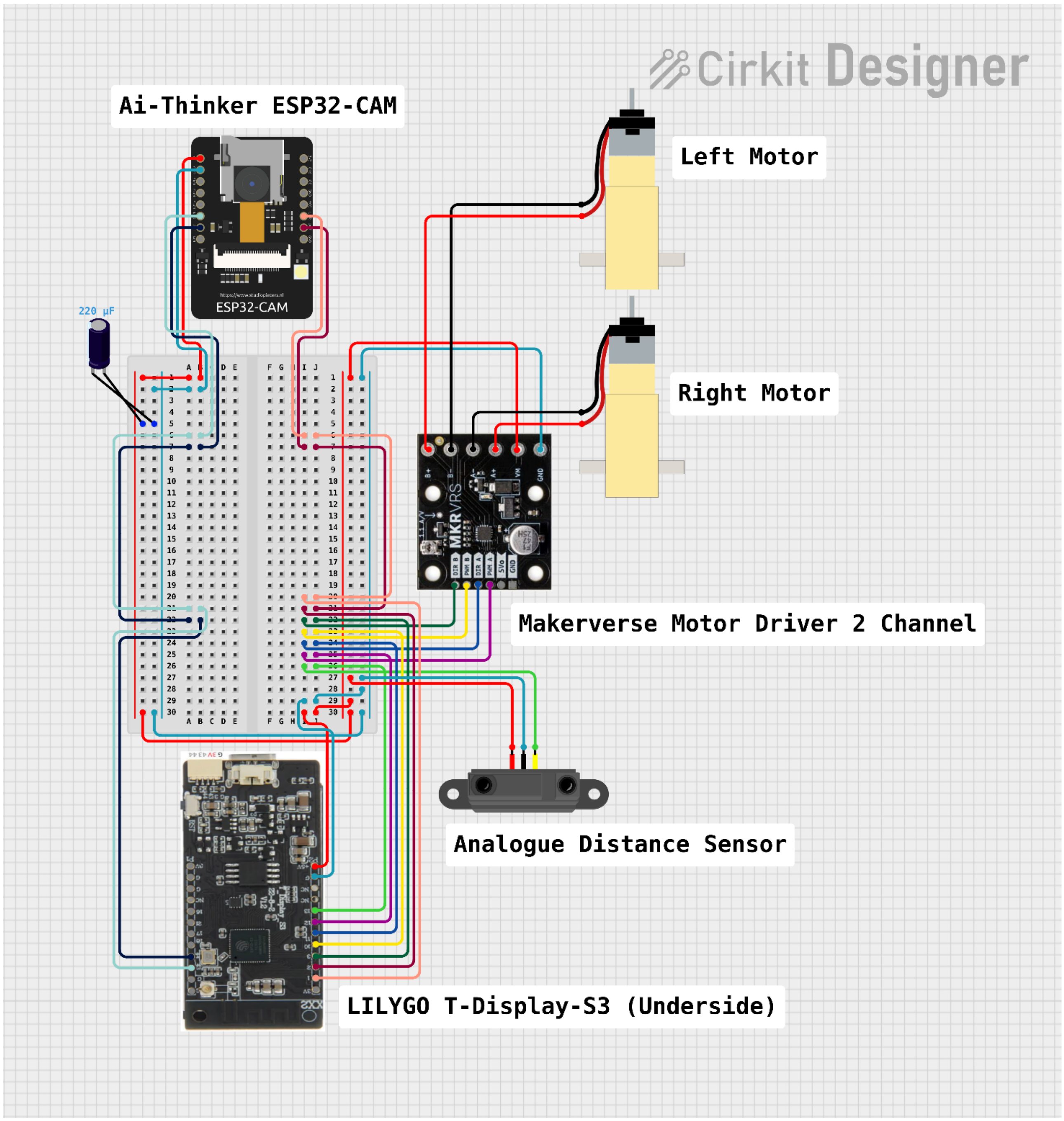

Figure 4: A pinout diagram depicting the wiring required between the electronic components of the final T-Display-S3 EyeBot design.

Table 1: The required GPIO pin connections between the T-Display-S3 and the ESP32-CAM.

T-Display-S3 GPIO Pin	ESP32-CAM GPIO Pin
43	14
44	2
1	3
2	1

Table 2: The required connections between the T-Display-S3's GPIO pins and the Makerverse 2 Channel Motor Driver's pins.

T-Display-S3 GPIO Pin	Makerverse 2-Channel Motor Driver Pin
3	DIR B
10	PWM B
11	DIR A
12	PWM A

Table 3: The required connection from the Sharp GP2Y0A41SK0F Analogue Distance Sensor's data output wire to the T-Display-S3's GPIO pin.

T-Display-S3 GPIO Pin	Analogue Distance Sensor wire
13	Data Output

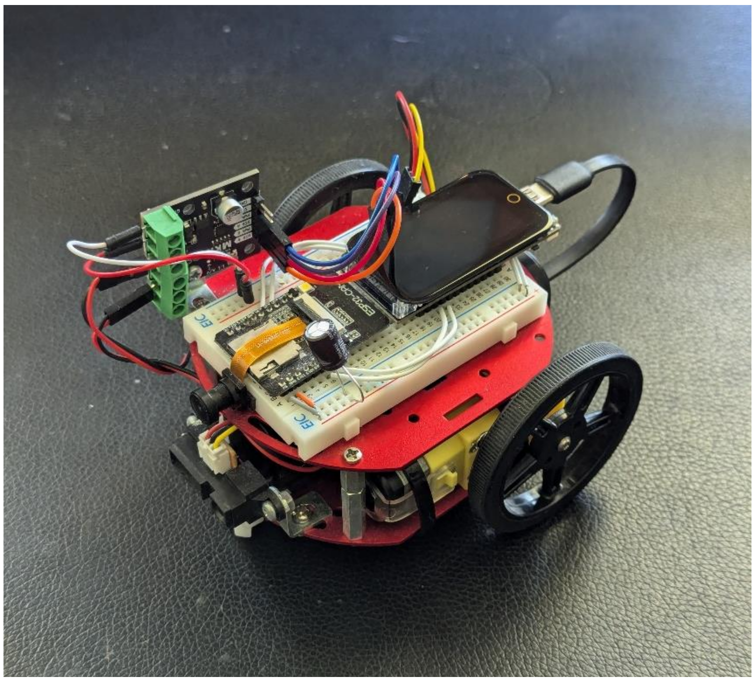

Figure 5: Angled view of the final T-Display-S3 EyeBot design.

3.1.1 Ai-Thinker ESP32-CAM

The solution settled upon for image transfer from the ESP32-CAM to the T-Display-S3 was to utilise the Serial Peripheral Interface (SPI) wired communication protocol. The SPI protocol operates in a master-slave configuration, with the T-Display-S3 configured to be the master device and the ESP32-CAM as the slave device. Data communication is unidirectional from the ESP32-CAM to the T-Display-S3 at a clock rate of 10 MHz (i.e. data rate of 10 Mb/s). At this data transfer rate, the performance bottleneck for the SPI transfer of a RGB565 pixel-format QQVGA resolution image, with a total data size of 38,400 bytes, was not in the speed of transmission but rather the rate at which the ESP32-CAM could supply images to be transferred. The ESP32-CAM was programmed to maintain a triple-buffer that was continually updated in the background with the latest image captured by the attached OV2640 camera module. Whenever at least one of the buffers has an image, the software prepares it for transfer by first swapping the endianness of each of the image's pixels from its default big-endian mode to little-endian mode. After this is done, it signals the T-Display-S3 via a wire connecting the two of them that it has an image to transfer, and then the ESP32-CAM waits for the T-Display-S3 as the master device to initiate the SPI transaction. Since the total data size of an RGB565 pixel-format QQVGA resolution image was 38,400 bytes and the maximum data size of a single SPI transaction (discovered experimentally) was approximately 5000 bytes, eight separate SPI transactions are initiated for a single image, with the next 4800 bytes of the image being transferred sequentially. The T-Display-S3 must be signalled using the wire for each of these image segments, and it is the responsibility of the T-Display-S3 to keep track of whether it is assembling the image in the correct sequence.

The OV2640 camera module was selected because it is the default camera module supported by the ESP32-CAM. The default variant of the OV2640 camera module with a 10 mm ribbon connector and a 60-degree field-of-view lens was replaced by one with a 40 mm ribbon connector and a 120-degree field-of-view lens. The greater field-of-view was selected so that the lanes of the UWA Robotics Lab test circuit in the peripheral vision of the camera could be more easily observed, and the longer ribbon length was for the purpose of mounting the ESP32-CAM in a low-profile orientation on-board the EyeBot, as seen in Figure 6.

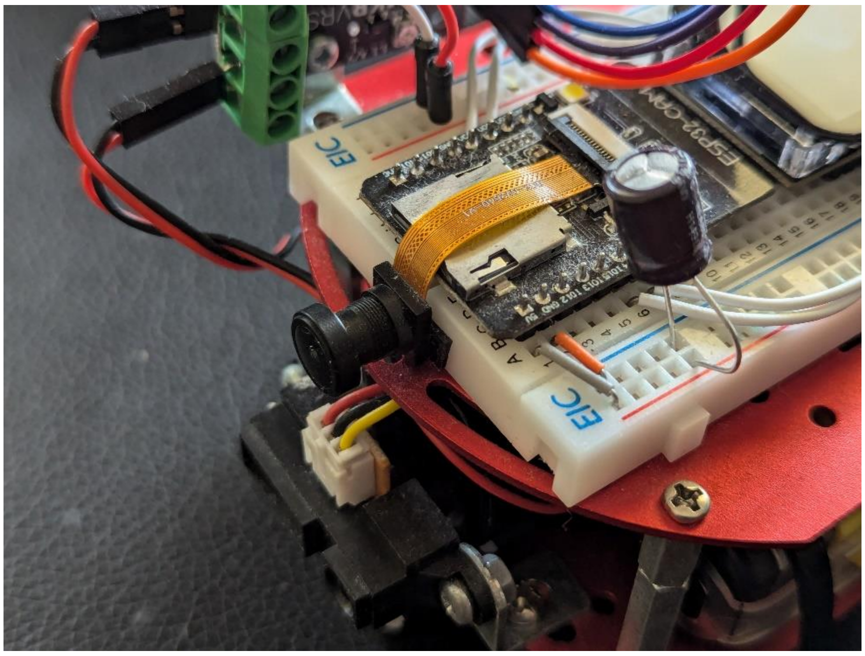

Figure 6: Close-up view of the ESP32-CAM and the attached OV2640 camera module's orientation on-board the T-Display-S3 EyeBot.

3.1.2 LILYGO® T-Display-S3

The T-Display-S3 has 13 free GPIO pins to physically connect and control electronic peripherals via, and all 13 were used for the final EyeBot hardware design: 4 for the ESP32-CAM connections, 4 for the Makerverse 2 Channel Motor Driver connections, 4 for the T-Display-S3's internally integrated touchscreen, and 1 for the analogue distance sensor connection. The specific wiring between the T-Display-S3 and these various components of the EyeBot (aside from the touchscreen, which is configured in software) are listed in Tables 1-3.

Regarding the implementation of the RoBIOS-7 API in the T-Display-S3 EyeBot's main software library, there is not much to note due to the relative simplicity of each of the functions. All the API functions that were implemented in the library are listed in Appendix A. The function for retrieving images from the ESP32-CAM waits for the ESP32-CAM to signal that it has an image segment ready, but the function will timeout after a set period (usually in the scenario of an ESP32-CAM not being properly connected to the T-Display-S3) and will instead return an image filled with a solid colour to the user. The function also assumes that eight consecutive image segments retrieved from the ESP32-

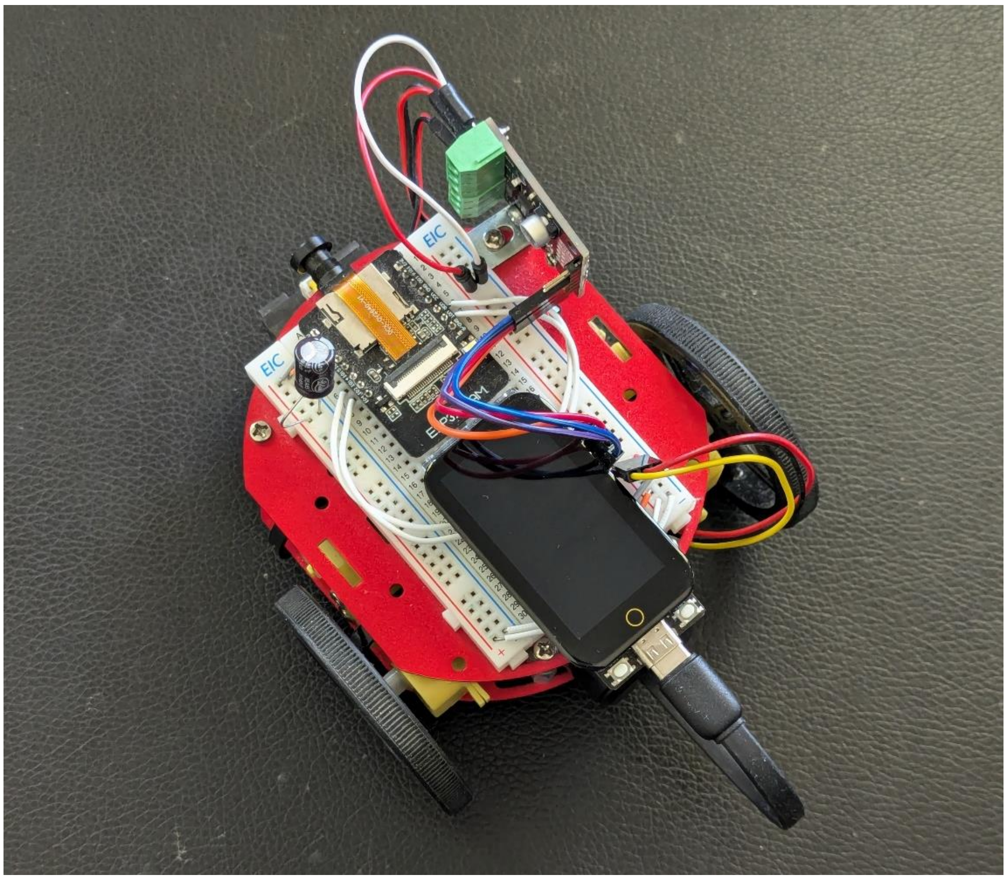
Figure 7: Overhead view of the T-Display-S3 EyeBot.

CAM constitute a complete image in the correct order, which is sometimes not true if the two devices are not reset at the same time. The only software solution to this problem would be to allocate another signal pin from the T-Display-S3 to the ESP32-CAM, which is not possible due to the lack of free GPIO pins on the T-Display-S3. The problem can be quickly resolved however by powering the whole T-Display-S3 EyeBot off and on again. The rate at which a whole image from the ESP32-CAM can be retrieved and displayed it to the T-Display-S3's screen is approximately 13 to 14 images-per-second, nearly double the rate deemed sufficient. This approximated result was gleaned from the instantaneous image transfer rate being calculated and printed to the T-Display-S3's display to be read.

Major assumptions were made in the implementations for the V-Omega driving interface subset of functions within the RoBIOS-7 API. Since there is no encoder feedback available from the motor pair, mappings from the requested linear and angular speeds to the required control signals for the motor driver are purely estimations based on simple heuristics. The calculations for requested linear speed involve a percentage calculation of the requested speed compared to the maximum linear speed of the EyeBot discovered through experimentation, which is approximately 500 mm/s. This percentage is then multiplied with the maximum PWM output value to produce the required PWM output value. The requested angular speed similarly calculates a percentage compared to the maximum angular speed of approximately 225 degrees/s that was also discovered through experimentation. This percentage is then multiplied with the maximum PWM output value, and then this new value is subtracted from the PWM output value of the motor that is desired to be pivoted around (e.g., subtracted from the right motor to turn right), originally calculated from the requested linear speed. These heuristics have proven to be sufficient for the EyeBot's low stakes demands in its driving functions. Speed variations between instances of the TT gear motors also led to a new function being added to the EyeBot library whereby the user can specify PWM offsets for each of the motors to potentially mitigate motor output inequality.

Such heuristics however have impacted the reliability of the position tracking functions of the V-Omega driving interface. These functions have been implemented in a way that involves the trigonometric change in distance and orientation over time being calculated whenever a new linear and angular speed for the EyeBot is set, which is then used to update the EyeBot's estimate for its global position and orientation. The idealistic assumptions made in these calculations combined with the lack of encoder feedback from the motors means these estimates quickly become inaccurate in practice, however.

3.1.3 Makerverse 2 Channel Motor Driver

The motor driver that was selected for the final T-Display-S3 EyeBot hardware design was the Makerverse 2 Channel Motor Driver, as seen in Figure 8.

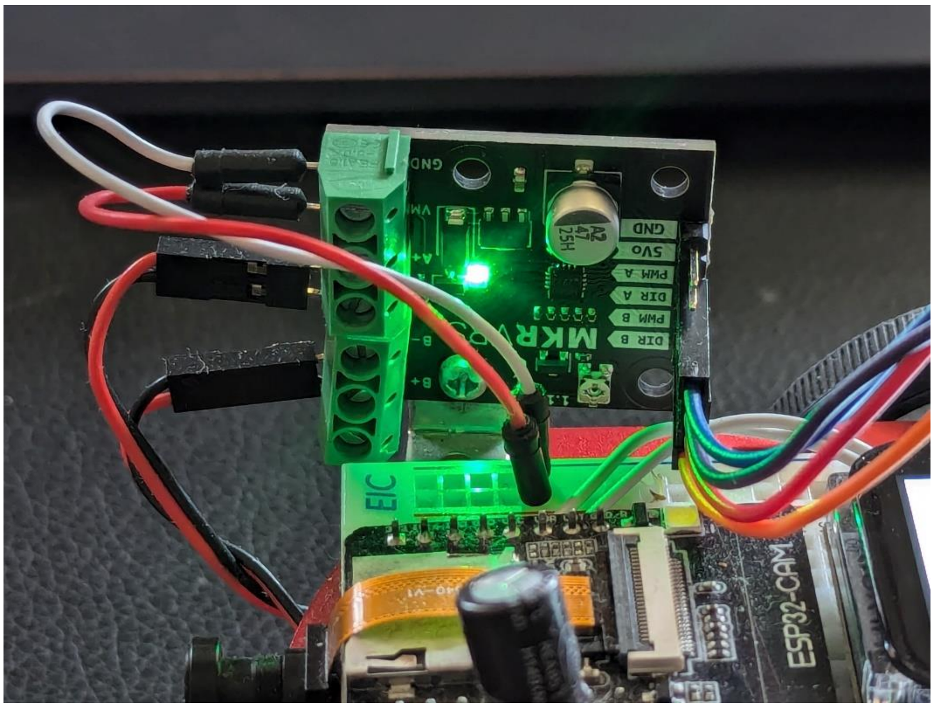

Figure 8: Close-up view of the Makerverse 2 Channel Motor Driver in the powered state on the T-Display-S3 EyeBot.

The previous L298N dual channel motor driver was replaced due to its relatively large size and subsequent difficulties mounting it on the EyeBot chassis. In contrast, the Makerverse 2 Channel Motor Driver is much smaller and can be easily mounted on the EyeBot's chassis. It also only requires four input pins (as opposed to the L298N requiring six) to control the connected motor pair. It can power itself and the motor pair directly from the T-Display-S3's 5V output pin, and very rarely trips out unless the PWM signals it's provided are at their maximum value while the directional control pin signals are being flipped. The simple workaround for this situation has been to set the PWM signal values to zero before changing the directional control pin signals.

3.1.4 Hoco J113 Energy-Bar with USB Type-C Connector

The whole T-Display-S3 EyeBot is powered from a single 5V/2A USB battery bank source, the Hoco J113 Energy-Bar. The J113 can be recharged using likes of a laptop or an AC power adapter through its female USB Type-C port, and the T-Display-S3 is powered in-turn through the J113's male USB

Type-C cable. The T-Display-S3's 5V output pin then administers power to the rest of the EyeBot's electronic components via the breadboard's power rails (as depicted in Figure 4). The initial 5V/1A USB battery bank that was provided with the EyeBot chassis at the start of the project proved to be insufficient to power both the motor pair at full speed and the ESP32-CAM at the same time. The large current draw in the circuit that was incurred when the motors were turning would drop the circuit voltage below the operational voltage of the ESP32-CAM, resulting in images with severe interference being captured and transferred to the T-Display-S3. The 5V/2A output of the J113 alleviated this issue while the motors were turning, but the problem persisted during the initial motor startup from stationary to rotating. The solution to this problem however was to place a 16V, 220 microfarad capacitor along the power rail connecting the T-Display-S3's 5V output pin and the ESP32-CAM 5V input pin, which buffers the brief voltage drop sufficiently for the interference in the captured camera images to be greatly mitigated.

3.1.6 Chassis and Remaining Work

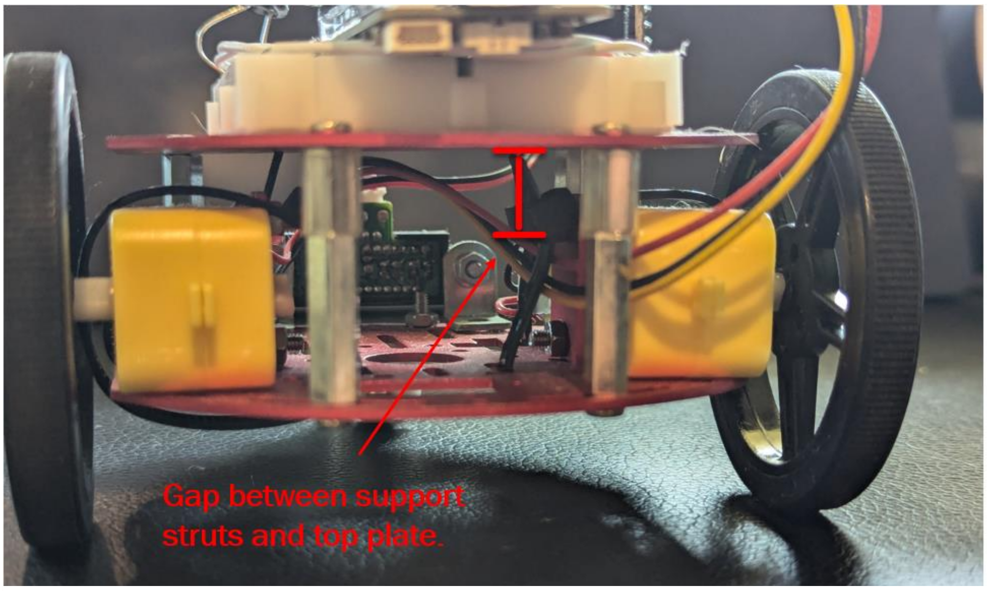

Figure 9: Close-up of the interior of the chassis with the J113 battery bank removed. The image is annotated to show the gap between the motor's support struts and the chassis top plate.

The battery compartment space in-between the top and bottom plate of the chassis was initially too small to accommodate the height of the J113 battery bank. The space between the plates was subsequently increased, but an unintended consequence of this modification though was that the support struts that each of the TT gear motors were separately bolted onto no longer reached across from the bottom to the top plate. The motor pair therefore had no fixed mounting point on the EyeBot's chassis. The temporary remedy for this was to tie down the motors to the base plate using cable ties (as depicted in Figure 9), but ultimately new supports should be created that can bridge the gap between the two plates and remove the need for the cable ties.

3.1.6 Overall Cost

An accurate estimation of the total cost of purchasing all the components necessary to build the T-Display-S3 EyeBot is difficult due to the lack of information regarding all the inputs for this design, such as how much it cost to manufacture the top and bottom plates used in the chassis. For the sake of this report a pessimistic cost estimation was calculated based on the assumption that all the major electronic components would not be bought in bulk. Shipping costs were excluded. Below is a list of the approximate cost (in Australian dollars, rounded up to the nearest dollar) of each major electronic component in the T-Display-S3 EyeBot design:

- LILYGO® T-Display-S3 (touchscreen variant with soldered pins): $35 [4]
- Ai-Thinker ESP32-CAM: $13 [14]
- OV2640 Camera Module with 40 mm ribbon and 120-degree lens: $8 [15]
- Makerverse 2 Channel Motor Driver: $9 [16]
- Hoco J113 Energy-bar (USB Type-C): $25 [17]
- 3-6V TT Gear Motor (x2): $14 [18]
- Sharp GP2Y0A41SK0F Analog Distance Sensor: $23 [19]
- 16V, 220 microfarad capacitor: $1 [20]

This sum of each of these electronic components is approximately $128. While the total cost of an EyeBot 8 is unknown, it is known that the cost of the EyeBot 8's interface board is $80 USD by itself [1]. It can be confidently concluded, therefore, that the T-Display-S3 EyeBot design is significantly cheaper than the EyeBot 8.

<u>3.2 Programs</u>

All instructions to upload the necessary code to the ESP32-CAM, install the T-Display-S3 EyeBot software library within Arduino IDE, and write programs for the EyeBot can be found in the source code repository in the possession of the project supervisors.

3.2.1 Colour-Tracking Navigation

Project success criterion 2 of Section 2.2 was achieved with the successful implementation of this program. In this program the user can select a pixel colour within the view of the EyeBot's camera, alongside setting the hue, saturation, and intensity thresholds above and below this selected colour. The user can then initiate the seeking phase, whereby the EyeBot searches for the highest concentration of pixels in its view whose colour falls within the threshold. To do this, a vertical and horizontal colour histogram of the input image is calculated to track which row and column contain the highest number of pixels that fall within the colour threshold. This identified row and column is then depicted as red crosshairs overlaid on the live camera feed, as depicted in Figure 10. The EyeBot turns on the spot at the set angular speed until the column falls within the central horizontal region of the image. At this point the EyeBot then drives forward at the set linear speed until either the column falls outside of the central horizontal region (causing the EyeBot to turn on the spot again, repeating the process), or until the distance sensor detects it is about to collide with an object, causing the seeking phase to come to an end.

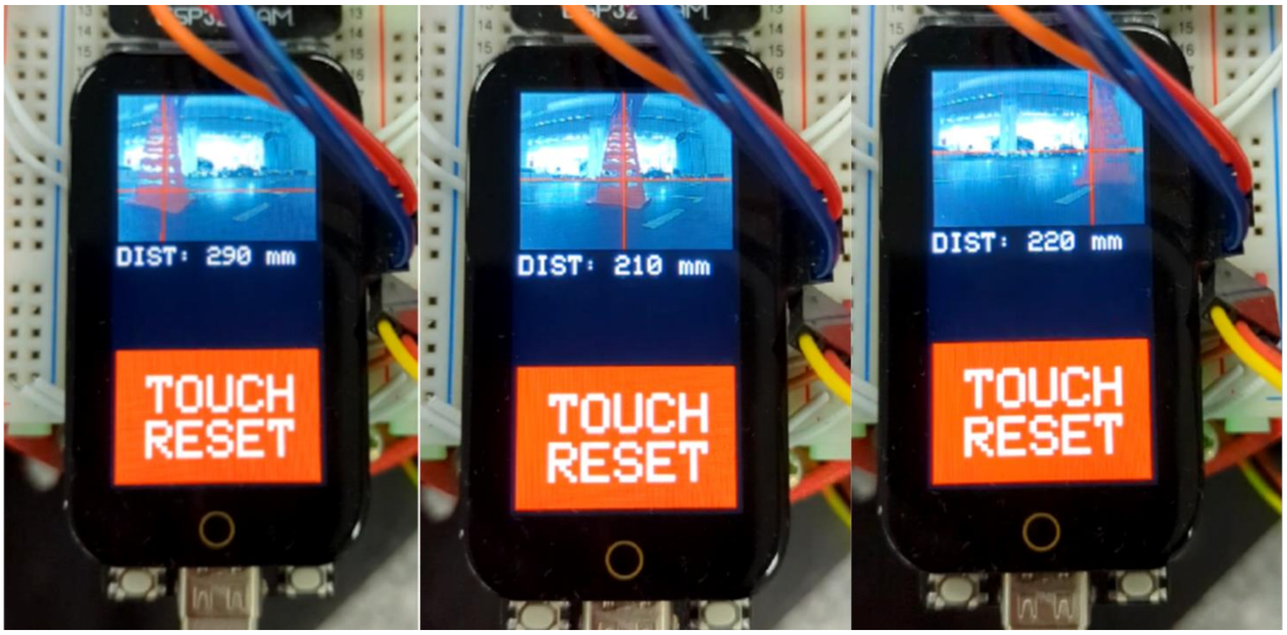

Figure 10: Sequence of images captured from a video of the Colour-Tracking Navigation program's seeking phase. In each image the red crosshairs are positioned over the orange cone, due to the colour threshold being set to detect the cone's colour.

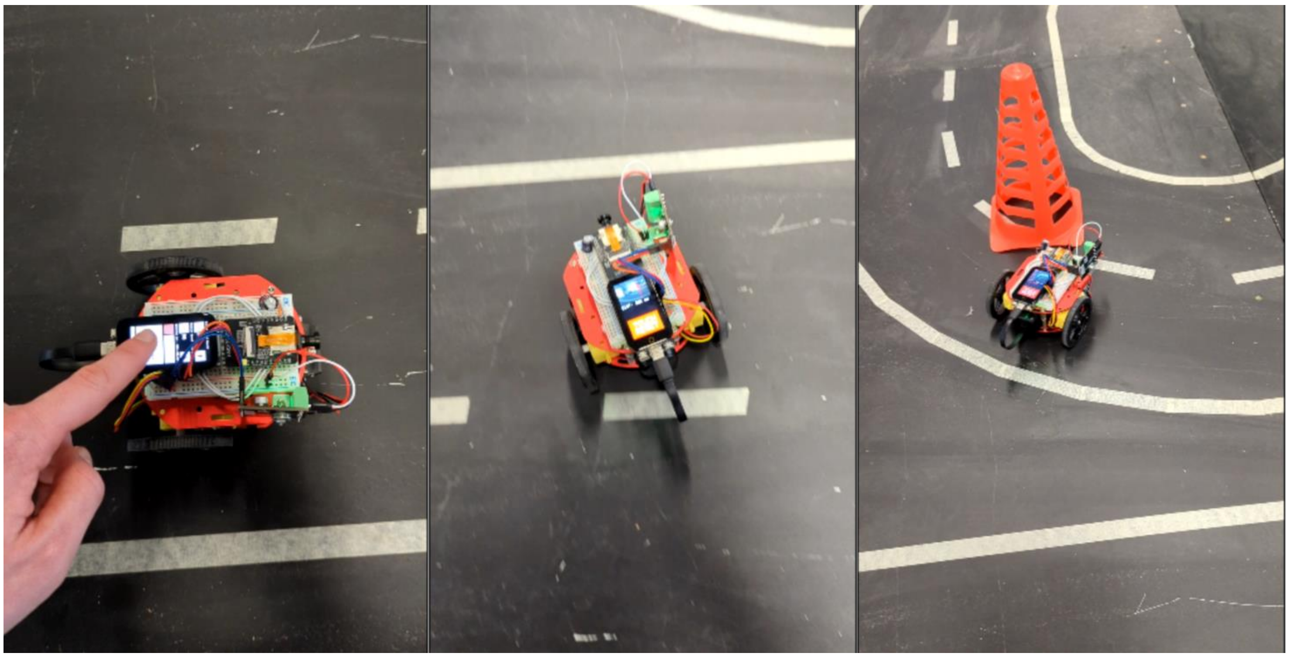
Figure 11: Sequence of images captured from a video, from left to right, showing the successful execution of the Colour-Tracking Navigation program. The third image shows the EyeBot stopping before colliding with the orange cone.

3.2.2 Ultrafast Line Detector Lane Navigation

Project success criteria 3 of Section 2.2 was also achieved with the successful implementation of this program. The algorithm for the Ultrafast line detector was implemented largely as is from the original paper [12], minus the corner joining and precise curve detection logic. For the denoising stage of the algorithm, a modified and stripped-down version of Basile Fraboni's Fast Gaussian Blur [21] was applied to a select region of interest (ROI) within the image. The ROI selected was the bottom quarter of the image, since it contained the EyeBot's entire view of the test circuit's surface and lane-markings. For the edge detection stage, the Canny Edge Detector [22] was implemented from scratch and applied to this same ROI.

The heuristic that was followed for lane detection from the set of lines the Ultrafast line detector produced was inspired by the lane departure warning system highlighted in the literature review [13]. Within the ROI, a line is designated as most likely representing a left or right lane depending on if they were outside of a central horizontal sub-region in the ROI, how close to it they were, if they are angled towards it, and on which side of the sub-region they were on. The sub-region chosen was the middle 40-pixel wide area spanning from the top to the bottom of the ROI, with the ROI itself being 160-pixels wide in total. If multiple lines qualified as possibly being a lane, the line with an end closest to the sub-region's horizontal border was chosen as the lane candidate. If a potential left or right lane was identified, an intersection test between the line and the horizontal border of the central sub-region was performed by calculating the line's gradient and then projecting it from the line's coordinates to the border. If the projected line intersected the border before intersecting the top of the ROI, then that as was regarded as turning inwards, and the EyeBot should turn in response until a line possibly identifying a lane only intersected with the top of the ROI instead. If both lanes are regarded as turning inwards, or no potential lanes are identified, the EyeBot drives continues to drive straight. This process is illustrated in Figure 12.

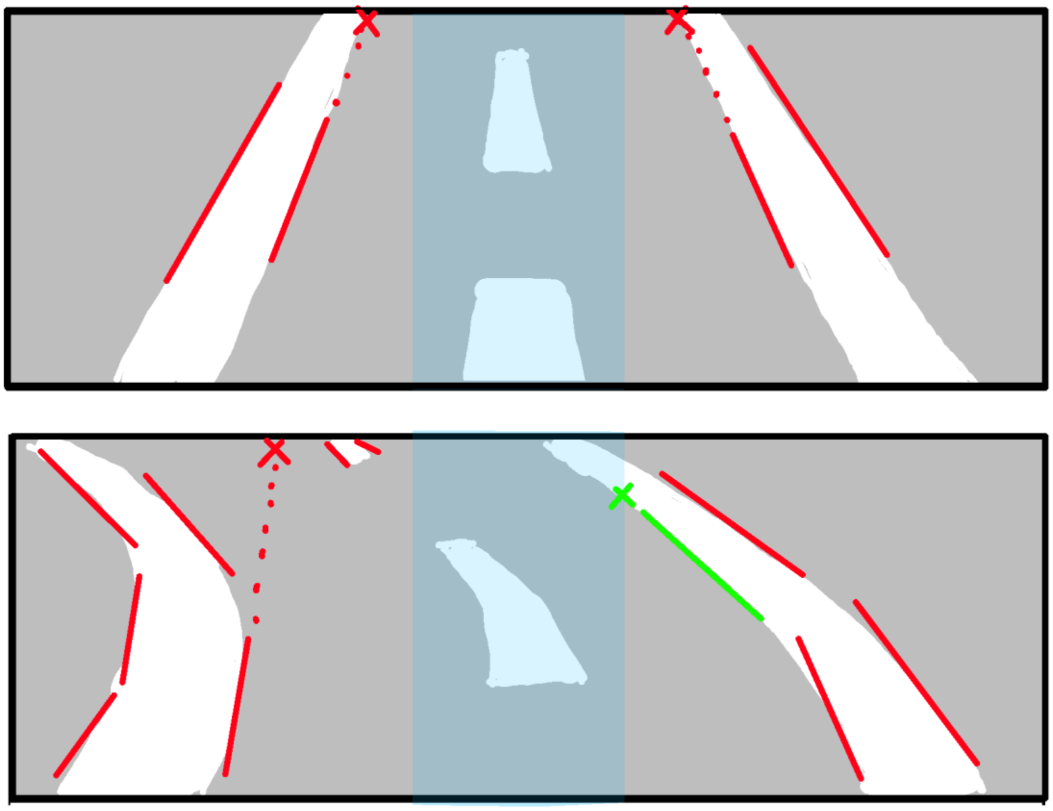

Figure 12: Two diagrams visually depicting the lane navigation algorithm implemented for the Ultrafast Line Detector Lane Navigation program. In both diagrams the transparent blue region represents the central sub-region of the ROI. In the top diagram, the left and right lane candidates are the red lines with dots extending out from them, showing how neither of them intercepts the sub-region. The EyeBot will therefore drive straight ahead. On the other hand, the green line in the bottom diagram, representing the right lane candidate, intercepts the sub-region. The EyeBot will therefore turn left in response.

The example program that has implemented this algorithm has been successful in navigating around the UWA Robotics Lab test circuit for multiple consecutive laps until testing ceased, all the while having the EyeBot's main body staying within the solid lane markings. The project supervisor approved of this satisfying the project success criterion. The program runs on the T-Display-S3 with minimal computational overhead, with the primary navigation logic (i.e., retrieving an image from the camera, detecting lanes within it, adjusting the motors, and finally drawing to the display) being performed at approximately 12 to 13 iterations per second, down only from the ceiling rate of 13 to 14 iterations per second that can be achieved by only retrieving and displaying an image. The approximate iteration rate of this program was gleaned by printing the measured instantaneous rate to the display to be read.

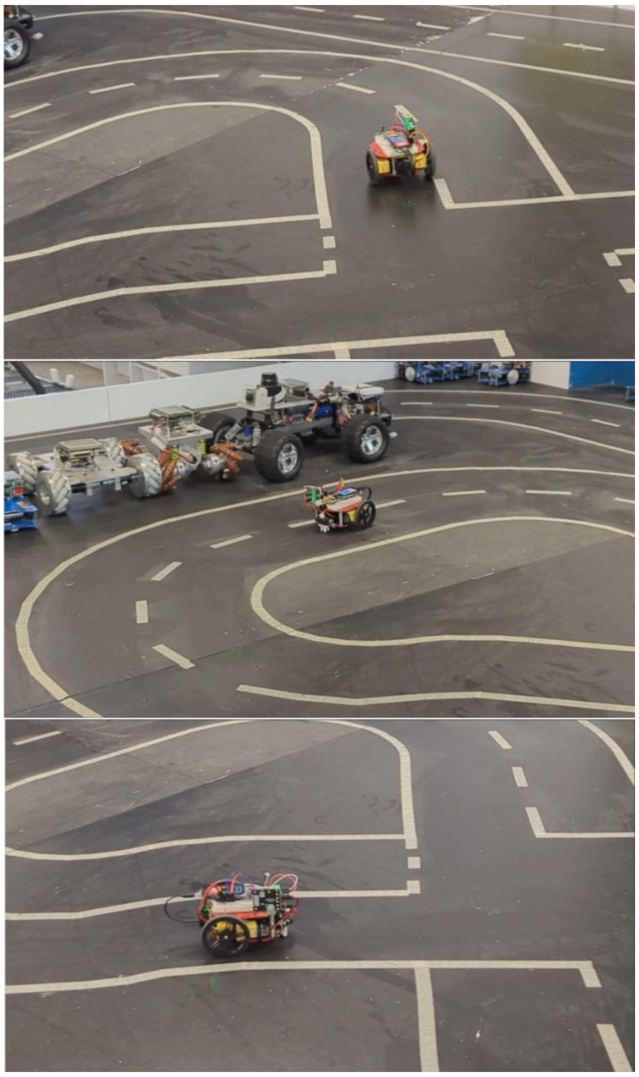

Figure 13: Sequence of images captured from a video, from top to bottom, depicting the successful execution of the Ultrafast Line Detector Lane Navigation program as the EyeBot rounds a bend.

In terms of limitations or flaws, because the Ultrafast line detector works by dividing the ROI into 4 by 4-pixel squares, and because the input image already had a low resolution of QQVGA, the detector often mistakenly joins closely parallel lines. This problem did not have a noticeable impact on lane detection however and would be alleviated if the process was applied to higher resolution images. The Ultrafast Line Detector is also very much dependent on a precise single-pixel width edge detected image being supplied to it. The author's implementation of the Canny Edge Detector in this program is unfortunately flawed however, often producing double-pixel width edges that are incompatible with the Ultrafast line detection method, which in turn causes the program to miss some lines. Considering the performance headroom leftover by this program on the T-Display-S3 however, it is certain that a superior version of this program could be written for it.

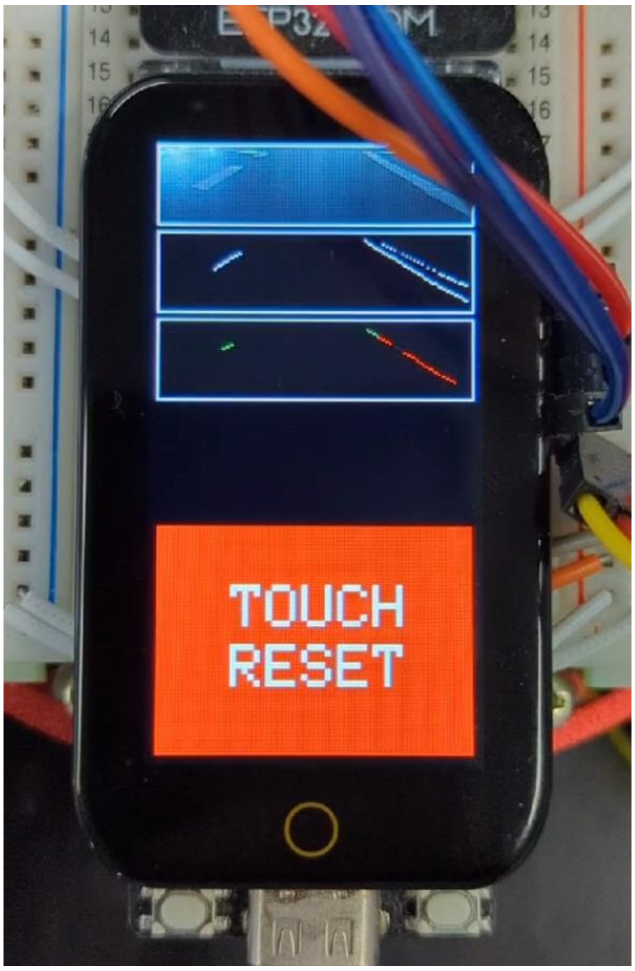

Figure 14: Close-up of the Ultrafast Line Detector Lane Navigation program's navigation screen. The ROI at three stages of the program's pipeline is displayed at the top half of the screen. From top to bottom, the first ROI is the raw colour image, the second is the output of the Canny Edge Detector, and the third is the output of the Ultrafast Line Detector.

3.2.3 Colour-Based Segmentation Lane Navigation

Due to the working implementation of the Ultrafast Line Detector Lane Navigation program being completed, work began on implementing the Colour-Based Segmentation technique for lane detection [10]. Unfortunately, however, there did not remain enough time in this project for substantial work on this program to be completed to a stage that is worthy of discussing, but the work-in-progress program can be found in the examples folder of the project source code in the possession of the supervisors.

4. Conclusions and Future Work

This project has been able to demonstrate that the T-Display-S3 EyeBot design proposed in this report is more than capable of assuming the role the EyeBot 8 currently occupies in the Embedded Systems unit. The T-Display-S3 EyeBot is significantly cheaper than the EyeBot 8, and it can perform the same core functionality the EyeBot 8 is used for in the Embedded Systems unit, as demonstrated by the Colour Tracking Navigation program that was written for it. The T-Display-S3 EyeBot is also capable of supporting programs that exceed the demands of the Embedded Systems unit, as demonstrated with the Ultrafast Line Detector Lane Navigation program. The minimal computational overhead this program incurred also indicates that more complex programs can be supported by the T-Display-S3 EyeBot as well. Both the Colour-Tracking Navigation and Ultrafast Line Detector Lane Navigation programs also demonstrate that the implementation of the RoBIOS-7 API for the T-Display-S3 is sufficiently performant and reliable. Users of the T-Display-S3 EyeBot can write programs for it using this API through Arduino IDE, which also removes one of the most significant usability hurdles that is present in the programming process required for the EyeBot 8. Each of these factors suggest that the T-Display-S3 EyeBot will improve the experience of students in the Embedded Systems unit, and the simpler process of assembling and maintaining the T-Display-S3 EyeBot compared to the EyeBot 8 will also improve the experience of facilitators for the unit. While there does remain work regarding manufacturing better support struts for the motors to replace the interim fix currently present in the design, this does not inhibit the current T-Display-S3 EyeBot design's usability. In conclusion, the UWA Robotics Lab should consider replacing their existing fleet of EyeBot 8's with T-Display-S3 EyeBot's for use in the Embedded Systems unit. The software and hardware designs developed for the T-Display-S3 EyeBot can also hopefully contribute to the UWA Robotics Lab's body of knowledge for future software and hardware development.

References

[1] T. Bräunl, “EyeBot 8,” UWA Robotics Lab, [Online]. Available: https://roblab.org/eyebot/. [Accessed 10 October 2024].

[2] T. Bräunl, “RoBIOS-7 Library Functions,” UWA Robotics Lab, January 2023. [Online]. Available: https://roblab.org/eyebot/robios.html. [Accessed 10 October 2024].

[3] “Xinyuan-LilyGO/T-Display-S3,” GitHub, [Online]. Available: https://github.com/Xinyuan-LilyGO/T-Display-S3. [Accessed 10 October 2024].

[4] “T-Display-S3,” LILYGO®, [Online]. Available: https://www.lilygo.cc/products/t-display-s3?variant=42589373268149. [Accessed 10 October 2024].

[5] R. Shobika, S. Mohanbaabhu, B. Hemalatha and S. Tamilselvan, “IoT-Based Autonomous Vehicle Control,” in *2023 Third International Conference on Smart Technologies, Communication and Robotics (STCR)*, 2023.

[6] Y. Ueyama, T. Sago, T. Kurihara and M. Harada, “An Inexpensive Autonomous Mobile Robot for Undergraduate Education: Integration of Arduino and Hokuyo Laser Range Finders,” *IEEE Access,* vol. 10, pp. 79029-79040, 2022.

[7] M. Zaidi, H. Daud, M. Shafique and H. A. Jamal, “Lane Detection in Autonomous Driving: A Comprehensive Survey of Methods and Performance,” in *2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC)*, 2024.

[8] L. Shi, J. Tan, B. Xu and K. Li, “A Research of Local-based Line Segment Detection Algorithms,” in *Proceedings of the 2023 International Conference on Computer, Vision and Intelligent Technology*, New York, NY, USA, 2023.

[9] I. Suárez, J. M. Buenaposada and L. Baumela, “ELSED: Enhanced line SEgment drawing,” *Pattern Recognition,* vol. 127, p. 108619, 2022.

[10] K.-Y. Chiu and S.-F. Lin, “Lane detection using color-based segmentation,” in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005.

[11] N. J. Zakaria, M. I. Shapiai, R. A. Ghani, M. N. M. Yassin, M. Z. Ibrahim and N. Wahid, “Lane Detection in Autonomous Vehicles: A Systematic Review,” *IEEE Access,* vol. 11, pp. 3729-3765, 2023.

[12] I. C. Yilmaz and I. C. Baykal, “Ultrafast line detector,” *Journal of Electronic Imaging,* vol. 31, p. 043019, 2022.

[13] I. Gamal, A. Badawy, A. M. W. Al-Habal, M. E. K. Adawy, K. K. Khalil, M. A. El-Moursy and A. Khattab, “A Robust, Real-Time and Calibration-Free Lane Departure Warning System,” in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019.

[14] “ESP32-CAM Development Board With OV2640 Camera,” Lonely Binary, [Online]. Available: https://lonelybinary.com/products/board-esp32-cam?variant=43890713034909&currency=AUD&utm_medium=product_sync&utm_source=google&utm_content=sag_organic&utm_campaign=sag_organic&gad_source=1&gcl

id=EAIaIQobChMI-4Hc1OiCiQMVlNkWBR3a_RDcEAQYASABEgK0EPD_BwE. [Accessed 10 October 2024].

[15] “New OV2640 Camera Module for ESP32 CAM 2.4G Wifi Module 200 222 30 45 120 160 Degrees 850nm Night Vision DVP 24PIN Night Vision,” AliExpress, [Online]. Available: https://www.aliexpress.com/item/1005004518669324.html?channel=twinner. [Accessed 10 October 2024].

[16] “Makerverse Motor Driver 2 Channel,” Core Electronics, [Online]. Available: https://core-electronics.com.au/makerverse-motor-driver-2-channel.html?gad_source=1&gclid=EAIaIQobChMI5dKM3-mCiQMVqVoPAh2CtRBuEAQYASABEgJ_bfD_BwE. [Accessed 10 October 2024].

[17] “Hoco J113 Energy-bar | 5000mAh power bank with cable - Type-C - Black,” Hoco Group (Australia), [Online]. Available: https://www.hoco.com.au/shop/j113-c-bk-hoco-j113-energy-bar-5000mah-power-bank-with-cable-type-c-black-6583#attr=. [Accessed 10 October 2024].

[18] “DC Gearbox Motor - TT Motor - 200RPM - 3 to 6VDC,” Core Electronics, [Online]. Available: https://core-electronics.com.au/dc-gearbox-motor-tt-motor-200rpm-3-to-6vdc.html?gad_source=1&gclid=EAIaIQobChMI44S8puqCiQMVRSV7Bx1u_AYsEAQYASABEgLFGfD_BwE. [Accessed 10 October 2024].

[19] “Sharp GP2Y0A41SK0F Analog Distance Sensor 4-30cm,” Core Electronics, [Online]. Available: https://core-electronics.com.au/sharp-gp2y0a41sk0f-analog-distance-sensor-4-30cm.html?gad_source=1&gclid=EAIaIQobChMI5MX_zOqCiQMVpuMWBR06-RwCEAQYASABEgKZrvD_BwE. [Accessed 10 October 2024].

[20] “220uf 16V PCB Electrolytic Capacitor,” Altronics, [Online]. Available: https://www.altronics.com.au/p/r5143-lelon-220uf-16v-pcb-electrolytic-capacitor/. [Accessed 10 October 2024].

[21] B. Fraboni, “Fast Gaussian Blur,” GitHub, [Online]. Available: https://github.com/bfraboni/FastGaussianBlur. [Accessed 19 8 2024].

[22] J. Canny, "A Computational Approach to Edge Detection," *IEEE transactions on pattern analysis and machine intelligence,* Vols. PAMI-8, pp. 679-698, 1986.

Appendices

Appendix A RoBIOS-7 API C++ Functions implemented for the T-Display-S3 EyeBot

Below is a list of signatures of the RoBIOS-7 API [2] functions that have been implemented for the T-Display-S3 EyeBot, with the remaining API functions that have not been listed here returning only error codes if they are invoked:

```
int LCDPrintf(const char* format, ...)
int LCDSetPrintf(int row, int column, const char *format, ...)
int LCDClear()
int LCDSetPos(int row, int column)
int LCDGetPos(int *row, int *column)
int LCDSetColor(COLOR fg, COLOR bg)
int LCDSetFont(int font, int variation)
int LCDSetFontSize(int fontsize)
int LCDGetSize(int *x, int *y)
int LCDPixel(int x, int y, COLOR col)
COLOR LCDGetPixel(int x, int y)
int LCDLine(int x1, int y1, int x2, int y2, COLOR col)
int LCDArea(int x1, int y1, int x2, int y2, COLOR col, int fill)
int LCDCircle(int x1, int y1, int radius, COLOR col, int fill)
int LCDImageStart(int x, int y, int xs, int ys)
int LCDImage(BYTE *img)
int LCDImageGray(BYTE *g)
int LCDImageBinary(BYTE *b)
int LCDRefresh(void)
int KEYGet(void)
int KEYRead(void)
int KEYWait(int key)
int KEYGetXY (int *x, int *y)
int KEYReadXY(int *x, int *y)
int CAMGet(BYTE *buf)
int CAMGetGray(BYTE *buf)
void IPLaplace(BYTE* grayIn, BYTE* grayOut)
void IPSobel(BYTE* grayIn, BYTE* grayOut)
void IPCol2Gray(BYTE* imgIn, BYTE* grayOut)
void IPGray2Col(BYTE* imgIn, BYTE* colOut)
void IPRGB2Col(BYTE* r, BYTE* g, BYTE* b, BYTE* imgOut)
void IPCol2HSI(BYTE* img, BYTE* h, BYTE* s, BYTE* i)
void IPOverlay(BYTE* c1, BYTE* c2, BYTE* cOut)
void IPOverlayGray(BYTE* g1, BYTE* g2, COLOR col, BYTE* cOut)
COLOR IPPRGB2Col(BYTE r, BYTE g, BYTE b)
void IPPCol2RGB(COLOR col, BYTE* r, BYTE* g, BYTE* b)
void IPPCol2HSI(COLOR col, BYTE *h, BYTE *s, BYTE *i)
BYTE IPPRGB2Hue(BYTE r, BYTE g, BYTE b)
void IPPRGB2HSI(BYTE r, BYTE g, BYTE b, BYTE* h, BYTE* s, BYTE* i)
int PSDGet(int psd)
int PSDGetRaw(int psd)
int VWSetSpeed(int lin_speed, int ang_speed)
int VWGetSpeed(int *linSpeed, int *angSpeed)
int VWSetPosition(int x, int y, int phi)
int VWGetPosition(int *x, int *y, int *phi)
int VWStraight(int dist, int lin_speed)
int VWTurn(int angle, int ang_speed)
int VWCurve(int dist, int angle, int lin_speed)
int VWDrive(int dx, int dy, int lin_speed)
int VWRemain(void)
int VWDone(void)
int VWWait(void)
```

Below are two functions that were added and implemented alongside the RoBIOS-7 API in the T-Display-S3 EyeBot's software library:

```
int EYEBOTInit();
int VWSetOffsets(int left_offset, int right_offset);
```