**UNIVERSITY OF NEWCASTLE**

Faculty of Engineering and the Built Environment

ELEC3850 – Electrical Engineering Design

**Final Report**

Autonomous City Vehicle 2

|  |  |  |
| --- | --- | --- |
| Tom Bailey (C3254803) | - | Computer Vision & Automation |
| Joshua Beverley (C3195884) | - | Motor Control & Power supply |
| Radhika Feron (C3256870) | - | User interface & Remote control |



2nd November 2018

Contents

[2 Introduction 2](#_Toc524100063)

[2.1 Problem Statement 2](#_Toc524100064)

[2.2 Project Objectives 2](#_Toc524100065)

[2.3 Scope 2](#_Toc524100066)

[2.4 Constraints 2](#_Toc524100067)

[3 Proposed Solution 3](#_Toc524100068)

[3.1 Solution Outline 3](#_Toc524100069)

[Computer Vision 3](#_Toc524100070)

[3.2 Interface Specifications 5](#_Toc524100071)

[4 Detailed Design 7](#_Toc524100072)

[4.1 Hardware Configuration 7](#_Toc524100073)

[4.2 Software Modules 9](#_Toc524100074)

[Vision 9](#_Toc524100075)

[Motor Control 12](#_Toc524100076)

[Joystick 12](#_Toc524100077)

[User Interface 12](#_Toc524100078)

[5 Planning Schedule 15](#_Toc524100079)

[5.1 Gantt Chart 15](#_Toc524100080)

[5.2 Milestones 15](#_Toc524100081)

[6 Conclusion 15](#_Toc524100082)

[7 Bibliography 16](#_Toc524100083)

[8 Appendices 17](#_Toc524100084)

[8.1 Appendix A: Gantt Chart 17](#_Toc524100085)

# Introduction

## Problem Statement

Driving a vehicle is one of the riskiest activities that people undertake on a daily basis. In New South Wales, there have already been 376 lives lost on the road during the 12-month period up to August 2018 and countless more people were injured [1]. Factors such as fatigue level and blood alcohol content of the drivers can drastically increase the chance of a crash occurring. The majority of the crashes that occur are due to human factors and thus, this project aims to reduce the human factor and increase the overall safety of driving. An autonomous system would not be susceptible to drunken stupors, risk taking behaviours such as speeding or reckless driving. It cannot experience fatigue and maintains constant surveillance of its surroundings. Autonomous vehicle technologies are designed to minimize accidents by addressing the main cause of collisions such as fatigue, distraction and human error.

The goal of this project is to wirelessly and autonomously control a model vehicle within a small city environment. A wireless connection between the vehicle and a nearby device will allow commands to remotely be sent to the vehicle and executed via on-board hardware. Autonomous city navigation will also be achieved at a basic level utilizing an on-board camera and computer vision techniques.

## Project Objectives

This project aims to design a prototype vehicle with the capabilities to navigate around a predefined track simulating a cityscape. The main objectives are:

* To remotely control the vehicle using a wireless device.
* Create a HTML webpage user interface that has both autonomous and manual control modes.
* Create software that will control the pulse width modulated input to the motors.
* Utilise computer vision techniques to analyse the images obtained from the camera.
* Basic autonomous mode functionality of following the track without assistance.

## Scope

To meet the requirements of the project objectives, included in the scope is the design and implementation of a working prototype which can be remotely controlled. It should also be capable of navigating a track with only the aid of a camera and computer vision facilities. The user interface should be capable of wirelessly communicating with the vehicle to provide commands and gain information on the vehicle status. A HTML webpage will be created to allow the user to choose between manual or autonomous control modes, with a live view from the camera displayed. There may be opportunities to broaden the scope to include more advanced autonomous functionalities once the basic prototype successfully completes the main project objectives.

## Constraints

The design and implementation of the autonomous city vehicle is required to satisfy the following constraints:

* **Time** – The research into potential solutions to the design brief and the creation of the chosen solution needs to be completed by the end of the semester. There is also limited time available to build the autonomous vehicle with each component interfaced correctly. It requires the careful balancing between successfully achieving the design brief and extending it according to the time constraints. The project is also required to achieve three major design milestones by week 6, 9, and 12 which are outlined in the Planning Schedule section.
* **Budget** – The budget of $200 limits the quality and quantity of components that can be bought. Thus, to minimise any exceedance of the budget, the components to be bought must be carefully planned and optimised beforehand.
* **Expertise** – A complete engineering project often requires knowledge from a range of engineering areas, ranging from website development to programming the motor controller. As students of Engineering, there is a limit to the range of engineering activities that we have been exposed to. Website development is often more geared towards IT students, and thus it will require extensive research to become familiar with the mechanisms of creating one to control the vehicle remotely.
* **Design brief** – The project design brief constrains the design project to fulfil a specific goal, however there is opportunity to extend it further if time permits.
* **Safety standards** – It is of the utmost importance that the completed project complies with Australian safety standards to ensure the user, as well as any spectators, will remain safe while testing the prototype vehicle.

# Proposed Solution

## Solution Outline

### Computer Vision

The main input for the guidance and autonomous control of the vehicle will be the Pixy CMUcam5 (Pixycam) attached to the vehicle chassis. A c++ program will be designed to run continuously on the Raspberry Pi, with each frame being processed in real time using OpenCV and other image processing techniques. The number of frames that can be processed per second (fps) will depend on the amount of processing that is being done. A minimum of 10fps is estimated to be required for an adequate amount of data to be collected for real time control.

The control output from the computer vision script is a 1x2 vector of floating point numbers (n), where -1 < n < 1. These numbers will be used by the motor control script to determine the velocity of each motor (right and left). For example, if the control algorithm determined a slow left turn was an appropriate response to the current state, output = [0.2,0.3] may be sent to the motor control script. As both numbers are low, with the right wheel slightly faster than the left, the car will begin a gradual left turn. If the computational overhead for processing each image is too high, it may be necessary to output an mx2 matrix, which defines a set of instructions for the vehicle to follow sequentially until the next frame is processed.

There are two proposed solutions using computer vision to generate real time control outputs for the motor control script.

1. **Algorithms purposely designed to handle different scenarios.**

OpenCV will be used to identify features in the captured image such as lines on each side of the road, intersections and traffic light states. These features will then be used in a control hierarchy to execute various actions. These actions may be a pre-determined series of steps, for example, the method used for turning at an intersection; Or the actions may be instantaneous adjustments to the current control parameters, depending on the frame rate achieved by the image processing algorithm.

This approach is likely to rapidly deliver a working prototype, though it is unlikely that the hierarchical control methodology will be able to encompass all possible environmental changes.

1. **Supervised Artificial Neural Network**

Artificial neural networks can adapt continuously to new data and learn features that are useful for system modelling from arbitrary, noisy data [2]. Like solution 1, features may be manually extracted from the image and passed into the network, or the raw image passed into the network directly. This solution will likely provide a more robust control model, though is limited by the quality and quantity of training data. Training data can be collected by one of the group members driving the car around the track, performing the correct actions at each step. Due to the large number of samples required to train a neural network, it may be difficult to gather an adequate quantity of data in the allocated timeframe.

One issue that arises when using neural networks is the lack of control over the final model. The network consists of several layers of nodes and weights, which have no real translation back to reality. This makes small changes to the final model difficult to make, as the model cannot be modified directly with predictable outcomes.

**Motors**

The DC motors to be used in the vehicle are the supplied 200RPM Hobby Gearmotors, fitted to a Shadow Chassis. The motors will be controlled with a Dual H Bridge DC motor controller, and 3 AA batteries. The motor controller takes 4 inputs (2 for each wheel) from a Raspberry Pi, also on the chassis. One of these inputs is for direction control, and the other input is an enable/pulse width modulation (PWM) input [3]. Initially the aim will be getting the motor to start, stop and reverse.

Once basic movement has been achieved, more advanced movement such as turning corners at different angles will be tested. This will be controlled by changing the PWM input. Changing the PWM input voltage changes the duty cycle over a constant period, changing the ratio of on-time to off-time [4]. The diagram below shows different PWM values (red) and duty cycles (black).

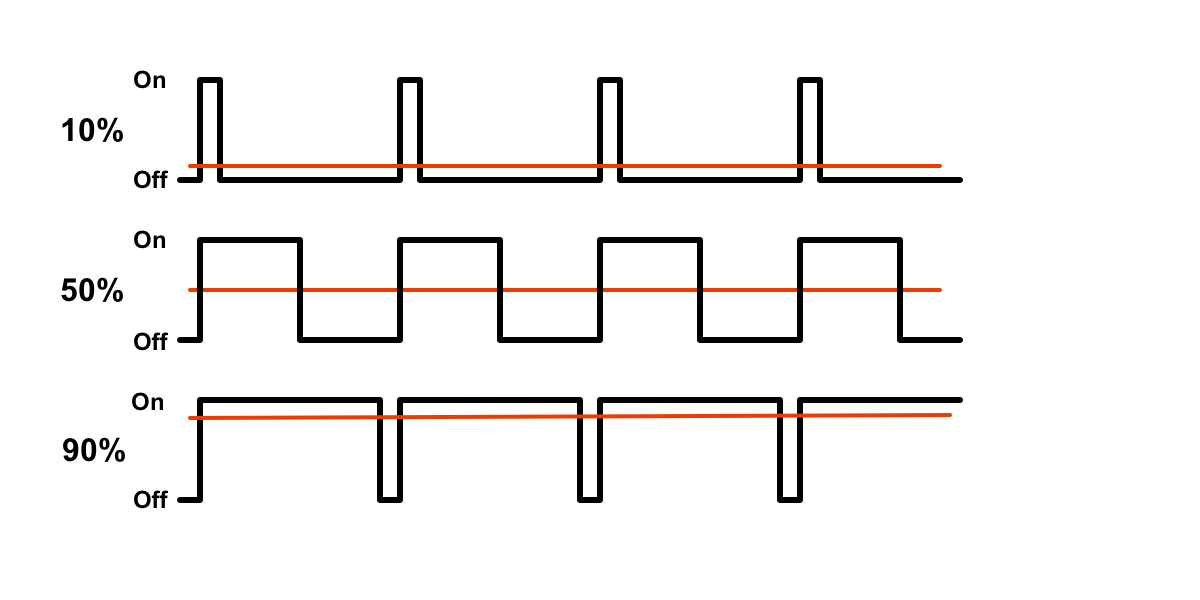


Figure 1 - Different PWM voltages changes the duty cycle over a constant period.

As stated in the computer vision section, the movement speed of each wheel will be altered depending on the expected input (2 floating point numbers between -1 and 1). The vehicle will be controlled autonomously by the computer vision inputs, as well as remotely from a website and a remote controller.

Consideration will also be given to the rate of acceleration; gradual acceleration will need to occur to ensure the vehicle doesn’t lose traction on the surface. Although the tyres are rubber, the surface of the road may be slippery enough that losing traction becomes an issue. Getting a consistent 4.5V DC to the motors when required as well as constant power to the Pi is important, and will be achieved using a 3S 5000mAh Lipo battery (9.9-12.6V DC) with a buck-boost converter to get the required voltages.

**Remote control**

The remote-control module will involve the creation of a HTML webpage as the user interface. It will have the features to choose ‘manual’ or ‘autonomous’ modes, with an arrow pad to control the PWM signal sent to the motor and thus the vehicle motion. The concept of using a webpage as the user interface enhances the portability of the controls since it can be accessed from most wirelessly connected devices such as a phone or a laptop with access to the Raspberry Pi’s Wi-Fi network. The Pi will be connected to either the University’s wi-fi network, or a mobile hotspot, depending on the restrictions imposed by the University network protocols. It will then be setup as a web server for it to host its own website. It will be accessible through the IP address of the Raspberry Pi, or a variation of it.

Web sockets will be used to connect the client and the server for bidirectional communications. A regular HTTP request needs to establish a connection before any transactions can occur, which requires time. A web socket is always ready to send or receive data in both directions, meaning the server can also trigger events in the client with little time delay.

Using this setup, the webpage will retrieve the 1x2 vector of floating point numbers (n) from the current state of the vehicle. The data will be manipulated according to the command provided by the user and sent back to the Raspberry Pi to be translated to a PWM input for the motors. The command to switch from ‘manual’ to ‘autonomous’ mode is required to start the new task of the vehicle autonomously navigating the track, with no interference from the ‘manual’ mode.

In the later stages of the project, the feed from the camera will be transmitted through the web server and onto the webpage to allow the user to see what the vehicle sees.

## Interface Specifications

The interface specifications of the individual software and hardware modules are illustrated in Figure 2. The Raspberry Pi 3 B+ will handle the computer vision processing and motor control through a c++ program, sharing a 2-element vector of floating numbers with a range of -1 to 1. USB interface, the computer vision module will receive the individual frames captured by the Pixycam. The motor control script will interface with the motor controller through the 4 control pins, generating 2 PWM signals for the left and right motors. The signal will range from 0V to 5 volts DC. The main power source will be the 3S 5000mAh Lipo battery which provides 9.9-12.6 Volts DC. The Raspberry Pi requires only 5V DC, thus a Buck converter will be designed as a DC-DC converter to mediate the connection between the battery and the Raspberry PI. This is a similar case for the motor controller.

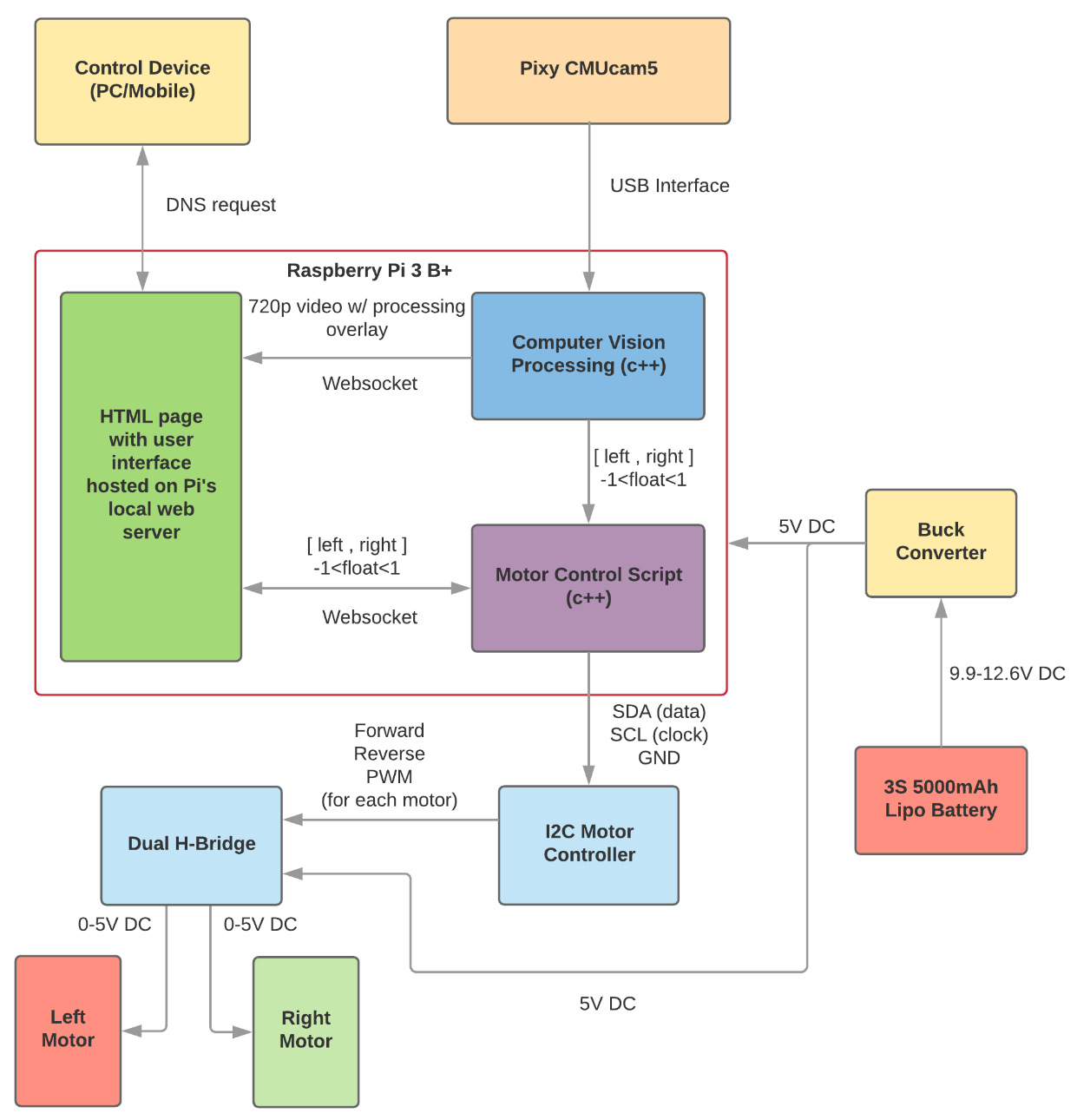


Figure 2 - Block diagram of the hardware and software modules, with their interactions

The Raspberry Pi will also act as a local web server with a HTML webpage accessible through the Pi. This webpage will receive a 720p video with processing overlay from the computer vision software, as well as the vector of floating numbers from the motor control script. The bidirectional interface with the motor control allows the webpage to receive the current status of the vehicle and update the values according to the user input. These values will then need to be sent back to be manipulated by the motor controller. Both connections to the web page will be facilitated by WebSockets. The control device will access the webpage through a DNS request to the Raspberry Pi’s local web server.

# Detailed Design

## Hardware Configuration

A diagram of the hardware configuration is shown below. The Raspberry Pi is connected to a DC motor controller using an I2C bus. This controller provides outputs to a dual h-bridge motor controller, which will control the DC motors depending on the input from the controller. The Pixy CMUcam5 is connected to the Raspberry Pi through USB interface. Both the motors and the Raspberry Pi are powered by the 3S 5000mAh LiPo battery.

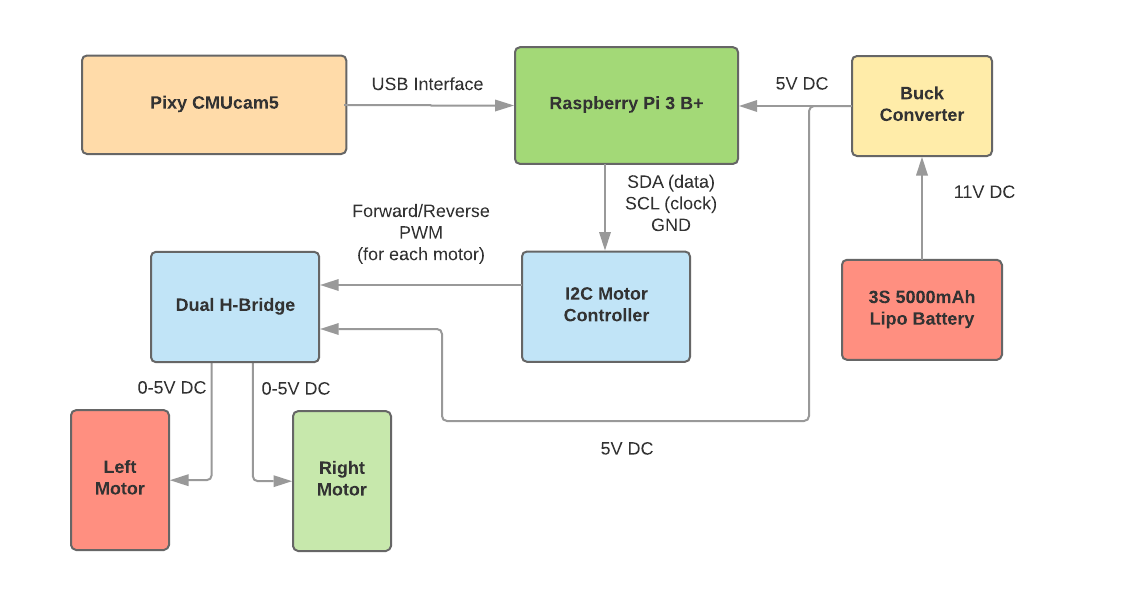


Figure 3 - Block diagram of the hardware components of the system

**DC motors**

A range of options were considered for controlling the DC motors. Initially, software PWM was used. This involved controlling the motors by toggling the motor enable pins directly from the Raspberry Pi. There were many issues using software PWM. When measured on an oscilloscope (shown below in Figure 4), the duty cycle and frequency visibly changed during the measurement. This issue will be compounded when there is more demand on the Raspberry Pi processors; when the computer vision and user interface sections are added to the Pi.

Other options such as Raspberry Pi hardware PWM and using a microcontroller such as the STM32 development board were considered, however each had their drawbacks. There are not enough hardware PWM outputs on the Pi to run two DC motors. Considering the need for a Raspberry Pi for the computer vision and user interface, and lack of space on the vehicle chassis, adding a microcontroller such as the STM32 just for DC motor control would be a poor design choice.

The solution to this is to use an I2C motor controller (such as a 12-bit PWM I2C interface). I2C only requires 3 outputs from the Raspberry Pi (SCL clock line, SDA data line and GND) and with a standard clock speed up to 100kHz, the frequency and duty cycle of the DC motors will be much more accurate compared to using software PWM [8]. This motor controller will be connected to a dual h-bridge motor controller, such as the L298D [10].

|  |  |
| --- | --- |
|  |  |

Figure 4 - Oscilloscope measurements of software PWM (50% duty cycle) of both DC motors. The changes in frequency and duty cycle can be seen from the measurements shown on the right.

**Remote Control/Joystick**

To make testing easier, a wired joystick has been added to the design. This has and will continue to be useful for testing basic and advanced movement for the DC motors. The joystick is connected to an analogue to digital converter (ADC), which is connected to 4 Raspberry Pi GPIO pins. The joystick is shown in Figure 5.

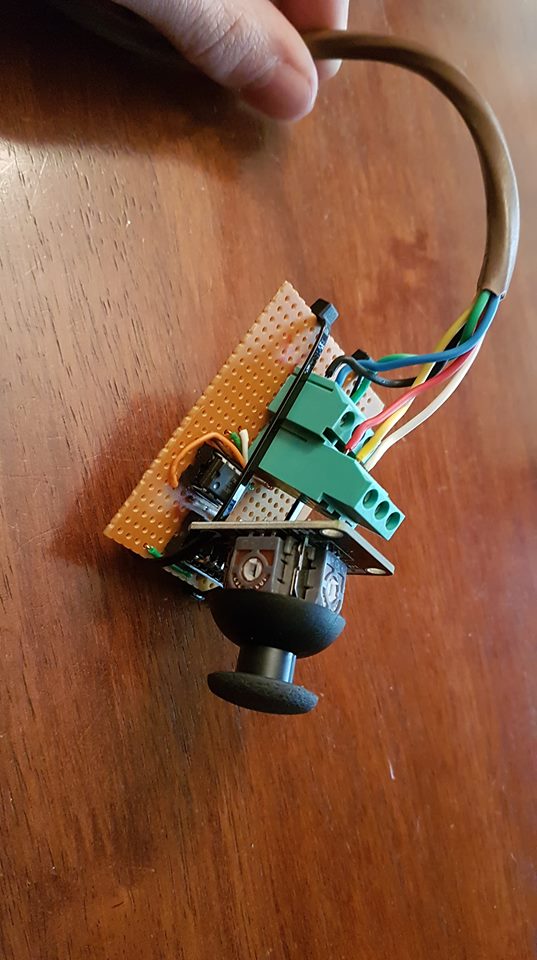


Figure 5 - The joystick used for initial testing.

While the joystick is a good short-term solution, a wireless controller will be used to remotely control the final vehicle. The controller used will be an XBOX 360 controller, connected to the Raspberry Pi using Bluetooth. Use of a wireless controller frees up space on the chassis as well as Raspberry Pi GPIO ports. Additionally, a controller such as the XBOX 360 controller has many additional buttons and joysticks that can be used to control the movements of the vehicle more precisely (for example, adjusting speed drop-off, or controlling each wheel with separate joysticks).

**Battery**

Both the Raspberry Pi and the DC motors require power. This will be provided by a 3S 5000mAh LiPo battery. The battery is rated at 11.1V, so this will be regulated to 5V using a buck-boost converter. A buck converter rated at least 3A is required; the Raspberry Pi draws maximum 2A, while the DC motors draw a maximum of 250mA each. The battery will be connected to the Raspberry Pi using the Pi’s Micro USB port. Powering the Raspberry Pi through the GPIO ports was considered, however this is riskier given there is no regulation or fuse protection on the GPIO ports (leaving it much more vulnerable to current/voltage spikes) [9].

## Software Modules

The main c++ program will be comprised of three primary modules, each running on separate POSIX threads. The vision module will receive image data from the camera, process the image using various techniques and generate a 2-element control vector in real time. Motor Control will use the 2-element control vector to manipulate two bi-directional DC motors. Lastly, the user interface will facilitate the communication between the main c++ program and the HTML web interface, also running on the Raspberry Pi.

### Vision

The vision module is the primary method of autonomous control of the vehicle. This will be achieved using two states and several functions designed for this purpose, supported by the OpenCV c++ library.

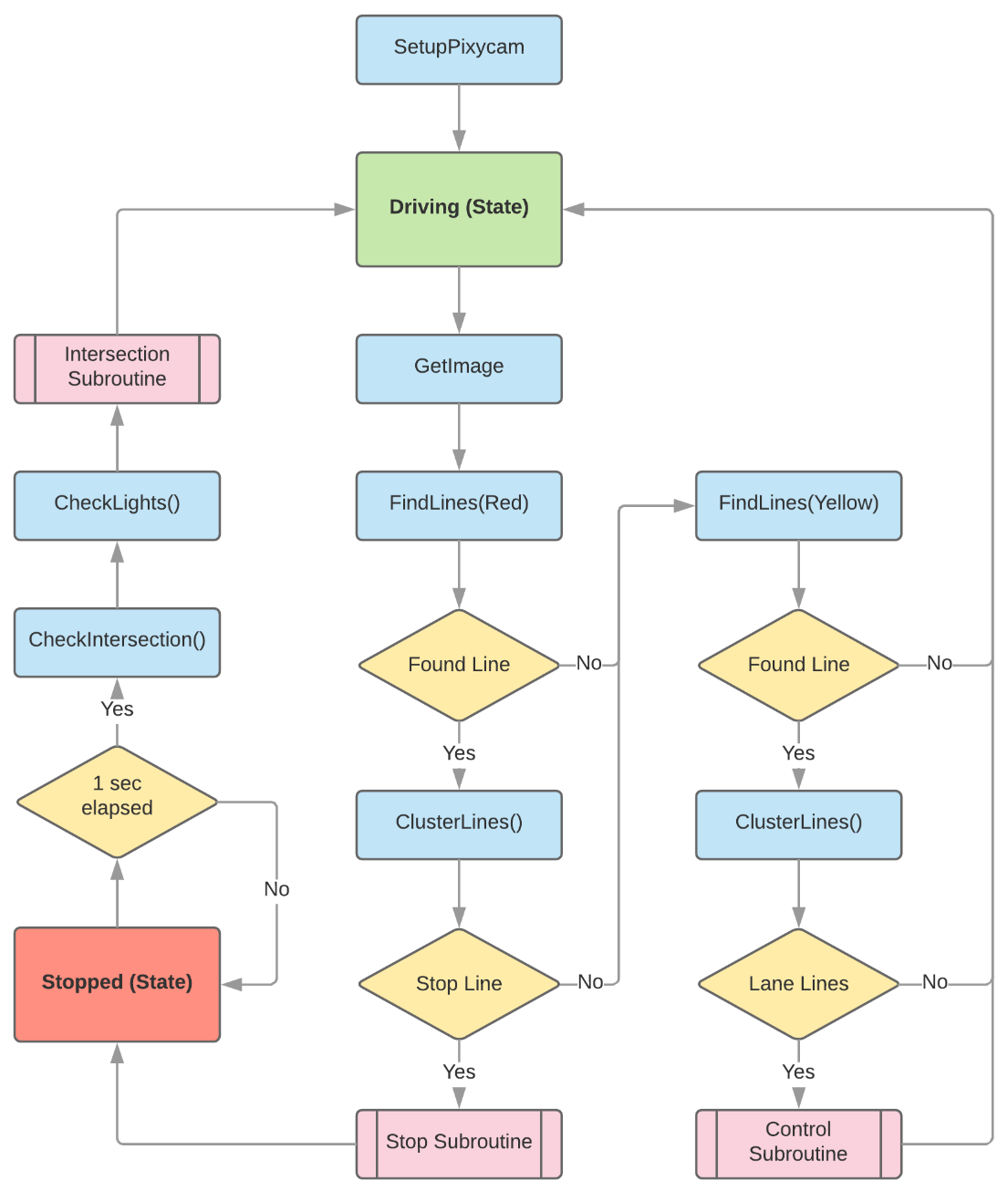


Figure 6 - Vision Module

**Driving State** - As seen in Figure 6, the vision module first initialises the Pixycam using the libpixyusb library. After which, the program is placed in the ‘Driving’ state. From here, a 320x200 image in 8-bit Bayer format is transferred from the 70KB Pixycam on board memory to the Raspberry Pi memory and converted to RGB format using OpenCV. The image is stored in an 8-bit, 3 deep OpenCV matrix (Mat).

Red stop lines are then searched for within the image using the FindLines function. A 3 element 8-bit point vector is used as an input to the function, where the vector represents a predefined HSV (Hue, Saturation, Value). The function returns an n length vector of 4 integer vectors. The four integers represent the start and end coordinates of the lines x0,y0 & x1,y1 respectively.

If lines are found, the ClusterLines function then groups all the similar lines together based on position and gradient. If a near horizontal, sufficiently dense red line is found, it is deemed to be a stop line and the stop subroutine is called. If not, the process then looks for yellow lane lines in the image.

Yellow lane lines are found in a similar way to the red stop lines, using the FindLines function with a different input HSV vector. ClusterLines is then used to cluster the lines within the image and return two distinct lines most likely to represent the lanes. If only one distinct line is found, one line is returned. If at least one distinct lane line is found after clustering, the control subroutine is called.

**Control Subroutine** – The control subroutine receives the line coordinates from the previous FindLines function, as well as how many lines are found, 1 or 2. These coordinates are stored in an n length vector of 4 integer vectors, where n is the number of distinct lines found. The control subroutine then extracts information about these lines such as the relative gradient, position and thickness. This information is then used to decide on the correct motor control vector to achieve the desired control outcome.

There are 4 simple cases, which will be expanded to include more niche circumstances during prototype testing. These cases are symmetrical between sides, so the opposite control parameters would achieve the opposite result.

Case 1: Only the left line is present in the image.

The vehicle is likely over the left line, needs to shift right to get back in the lane. Motor control vector (r) is decreased, causing a slow merge right.

Case 2: |Left line gradient| > |right line gradient|.

The vehicle is too close to the left line, needs to slightly shift right in the lane. Motor control vector (r) is decreased, causing a slow merge right.

Case3: Left line gradient > 0 & right line gradient > 0.

There is a right turn. Motor control vector (l) is increased and motor control vector (r) is decreased, causing the vehicle to turn right.

Case4: |Left line gradient| ≃ |right line gradient|.

The lane is straight and the vehicle is near centre. Motor control vector (l) increase and motor control vector (r) increase, causing the vehicle to accelerate.

After the control subroutine has changed the value of the motor control vector, it returns the program to the ‘Driving’ state.

**Stop Subroutine** – After a stop line has been detected, the stop subroutine is called. Due to the orientation of the camera on top of the vehicle, the stop line is detected when it is approximately 10cm in front of the vehicle. The stop subroutine aims to decrease the vehicle velocity gradually, eventually bringing it to a complete stop before the line. The procedure is as follows:

1. Estimate distance to line
2. Estimate vehicle velocity based on motor control vector
3. Determine vehicle stopping distance (D)
4. When vehicle is D from the stop line, motor control parameters are set to zero.

An experimentally derived function or lookup table will be used to estimate the stopping distance of the vehicle at various velocities. It should also be noted that motor control smoothing is done in the motor control module, so does not need to be done prior to setting the motor control vectors to zero.

After the stop subroutine, the program is placed into the ‘Stopped’ state.

**Stopped State** – Firstly, a 1 second timer will be started to ensure that the vehicle stops for a minimum of 1 second before continuing. During this time, the CheckIntersection function is used to determine the possible directions available at the intersection. There must be a minimum of 1 direction available, with a maximum of 3; Straight, left and right.

The check intersection function obtains this information by masking the image into 3 segments, top left and right. These sections are scanned individually for yellow lane lines and correct orientation using FindLines and ClusterLines functions respectively. After intersection options have been determined, a direction is chosen randomly for the vehicle to travel. This is done to keep the vehicle testing dynamic but could also be passed as a user-controlled parameter via the user interface defining which direction to travel.

The CheckLights function is then used to determine whether there are traffic lights operating at the intersection. Again, this function uses a masked portion of the original image, looking for characteristic colours associated with traffic lights being present on the opposite side of the intersection.

After 1 second has elapsed on the timer, information about the determined intersection travel direction and traffic light presence is passed into the intersection subroutine.

**Intersection Subroutine** – If traffic lights are present, the intersection subroutine will first wait until the light is green to proceed. Light colour detection will be achieved by masking a portion of the image and finding the most dominant traffic light control colour.

Due to the complex nature of lines on the intersections, pre-programmed motor control functions are used to navigate each direction; left, right or straight. These motor control functions will contain an ordered set of motor control vectors, with timing constants controlling the duration of each movement phase. These vectors and timing constants will be determined experimentally during the testing phase of the prototype. After the motor control function has completed the movement, control is passed back to the ‘Driving’ state.

### Motor Control

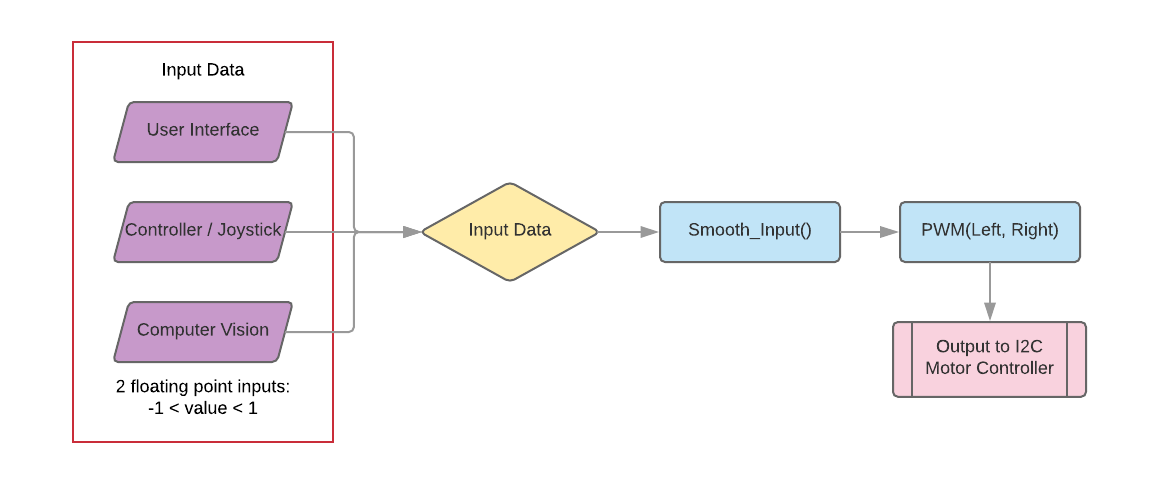


Figure 7 - Block diagram of the software modules related to the DC motor control.

Inputs from either the user interface, computer vision or the controller/joystick will be provided. The computer vision will be the lowest priority, as it can be controlled by the user interface or controller.

The input from any of these will be a vector containing 2 floating point numbers between -1 and 1. These represent the direction and speed of both the left and right motors. That is, the sign of the number dictates if the wheel moves forward (positive) or backwards (negative); while the speed is dictated by the value of the number (100% speed represented by 1.0, values less than this will decrease the speed).

The input will move to the smooth input function. Data coming from the computer vision and controller will not always be very accurate. If not controlled, inaccurate data can lead to jittering or unintended abrupt changes in movement of the motors. To control this, an exponential rolling average will be applied to the inputs. This places emphasis on the most recent data, while still accounting for older data points (weighted less from exponential decay) [11].

Once calculated and averaged, these values are sent to a PWM function, where a for loop is used to calculate the on/off time required for the DC motor. This is then sent to the I2C motor controller, which will alter the direction and speed of the wheels as required.

### Joystick

The joystick sent 3 values back to the Raspberry Pi: two numbers between 0 and 255 representing horizontal and vertical movement, and a binary input controlled by the button (pressing the joystick in). The following calculations were used to find the two inputs to the PWM function (one for each wheel, floating point numbers between -1 and 1):

* Wheel opposite to direction being turned: .
* Wheel adjacent to direction being turned:
* If the joystick is pushed down (intending backwards motion for the vehicle), the subtraction of 127 from the vertical offset ensures the result is negative.

### User Interface

The user interface is built upon a Node.js webserver hosted on the Raspberry Pi and developed using Express and Socket.io. Node.js allows JavaScript code to be run server-side as well as in the web browser. Express is a web application framework for Node and provides the feature to serve static files. Socket.io is a JavaScript library that uses WebSockets to support real-time, bidirectional communication between web servers and clients. In this project, it has three parts:

* HTML & JavaScript client-side library that runs in the browser
* Server-side library for Node.js on the Pi
* C++ client-side library that runs on the Pi

The code is separated into these three sections, each performing specific tasks and interlinked through WebSockets, as shown in Figure 8. Any arrow that crosses from one software module to another is facilitated by the WebSocket. Socket.io sends messages between server-client(s) using the .emit() function and handles incoming messages through the .on() function. Essentially this software module involves the bidirectional communication of the control vector between the webpage client and the C++ client, with the webserver acting at the intermediary. The webpage client will manipulate the vector according to the buttons pressed. The C++ client will act as the connection between the server and the computer vision and motor controller C++ codes. It will pass on the new vector data and will also be capable of interrupting the motor controller to switch between autonomous and manual modes.

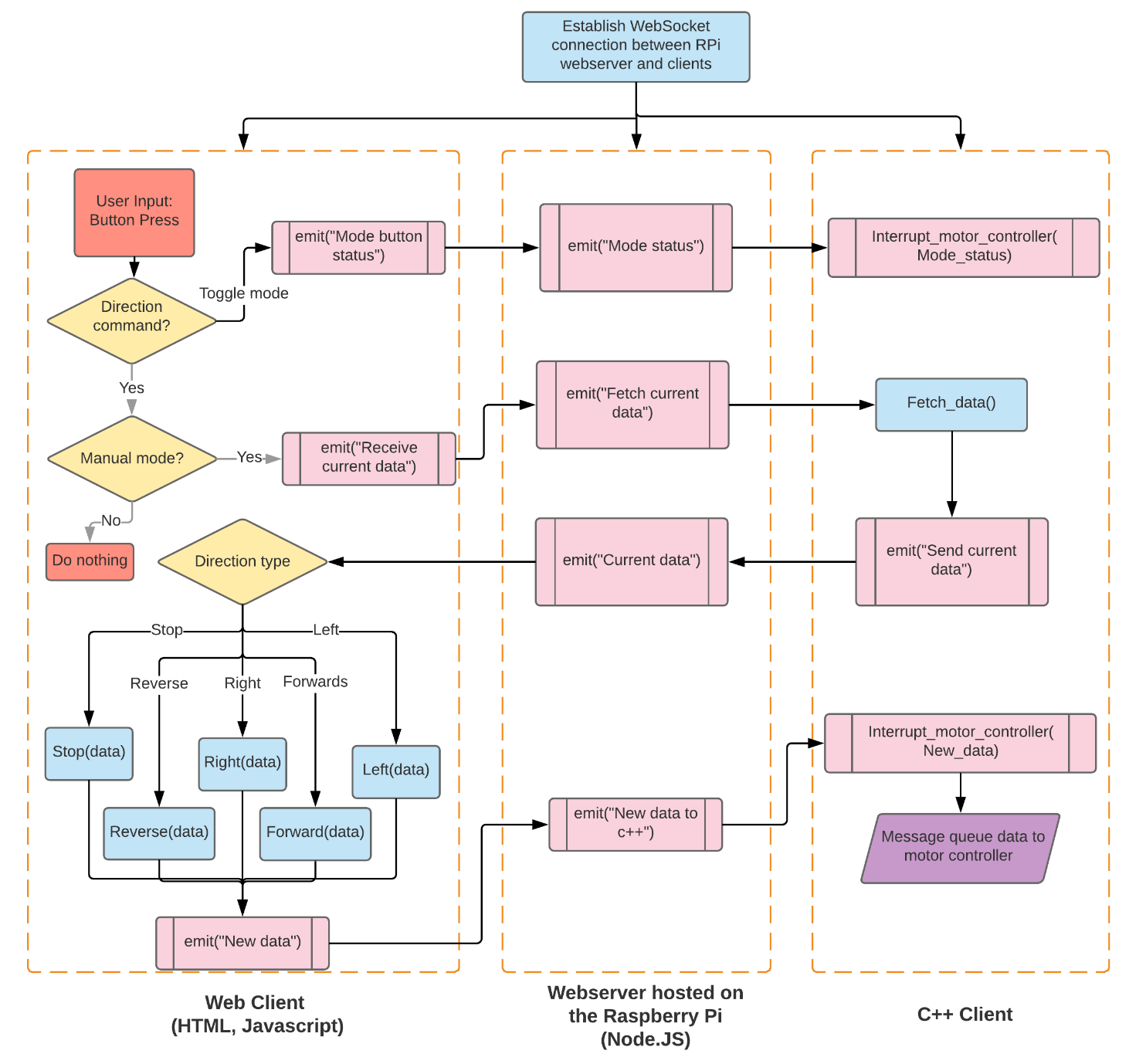


Figure 8 - Block diagram of the user interface

The webserver is hosted on the Raspberry Pi and can serve static files located on the Pi to allow direct access to files such as images and JavaScript files. The Node.js webserver is configured to listen on port 8000, and so any WebSockets created will communicate via that port as well. The webpage client is setup when the user accesses the webserver through the IP address of the Pi. This sets up the WebSockets between the server and the webpage client at the same time. The HTML webpage will provide the user interface through a display of the remote controls. The JavaScript WebSockets allows for changes to the webpage to be monitored and for data to be broadcasted to the server. The WebSocket connection to the C++ client is setup when the entire C++ program with all software modules is run.

When the user presses a button, it can either be one of two button types; the button to toggle the autonomous and manual modes, or the button to control the vehicle motion. The individual software paths that can be followed for each are outlined below.

**Toggling autonomous and manual modes**

The JavaScript client will send the status of the “mode” button to the webserver, which will then send it to the C++ client. A variable will keep track of the current mode the system is in. If the button status is the same as the current status of the system, then it will not be sent to the server. The C++ client will also check if the new state received is different from the current state and will interrupt the motor controller if it needs to switch modes. This will change the source of the control vector to be either the web server or the C++ computer vision code, depending if it is in the autonomous or manual mode. If the current system is in manual mode, the computer vision thread will be in “sleep” mode, meaning it does not control the vehicle. It will wait for a signal from the message queue connected to the C++ client indicating that the mode has changed to autonomous.

**Controlling vehicle motion**

If one of the “Forward”, “Reverse”, “Left”, “Right”, and “Stop” buttons were clicked, a request will be emitted to the webserver to obtain the 2-element control vector currently controlling the bi-directional DC motors. However, if the system is currently in autonomous mode, no request will be sent. If it is in the manual mode, the server will handle the request by emitting a “Fetch current data” message to the C++ client. The C++ client receives this and calls Fetch\_data() to access the current values of the control vector through a message queue. It will then emit a “current data” message containing the two vector values to the server, which will then be emitted to the webpage client. The client-side function will then determine which direction button was pressed and call the corresponding function. Each function will manipulate the vector data accordingly, and then emit the new data back to the server, which is then sent to the C++ client. There, the C++ client will initiate an interrupt to the motor controller by sending the new vector elements. The motor controller script will then adjust the output to the two motors and change the vehicle motion.

This entire process is restarted each time the user clicks a button on the webpage.

# Planning Schedule

## Gantt Chart

See Appendix A for the Gantt Chart which contains the project schedule and the individual tasks each project member is undertaking.

## Milestones

**Week 6 Milestone**

Have a functional remote-control car with advanced movement (turning corners, forward and reverse driving), controlled through either physical buttons/joystick or by PC (connection through SSH PuTTY). This milestone has been successfully achieved through the joystick control of the car. Extra progress has been made on the Computer Vision and User Interface modules as well.

**Week 9 Milestone**

Have the computer vision software able to recognise the track, including intersections.

**Week 12 Milestone**

Have a fully autonomous car capable of following the track independent of the remote control. The car will interact with the basics of the track including intersections.

# Results

## Motor Controller

## Computer Vision

## User Interface

# Contribution of each group member

# Brief user’s guide

# Bill of materials (appendix????)

# Conclusion

The HTML webpage has been created with basic direction controls and a mode button. A successful connection between the webpage client and the Node.js webserver has been established, with further work needing to be done to establish a similar connection between the C++ client and the server. Ultimately, the aim is the facilitate the sharing of data between the two clients, and integrate the C++ client into the main C++ program.

A remote-control vehicle has been achieved using a joystick, with basic movements such as forward driving and turning corners achieved. The current system uses software PWM, which due to issues maintaining a consistent frequency and duty cycle, will be replaced with an I2C motor controller connected to a dual h-bridge motor controller. Additionally, the joystick will be replaced with a wireless controller later in the project.

The Raspberry Pi and the two DC motors will be powered with a 3S 5000mAh LiPo battery which, with a rating of 11.1V, will be regulated down to the 5V required using a buck converter.

The computer vision module is capable of colour recognition and can detect the presence of lines on the track. A development tool has also been developed to easily determine the HSV values of colours in real time, to enable quick and easy colour identification during development. Future work to be done includes developing robust methods to cluster lines based on various features; Including gradient, thickness and location. Basic logic to use these features to manipulate the motor control vector also needs to be developed.

# Bibliography

[1] NSW Government. (2018, August 12). *Statistics*. Retrieved from Transport for NSW: Centre for Road Safety: http://roadsafety.transport.nsw.gov.au/statistics/index.html

[2] A. Abraham, "Artificial Neural Networks", *Handbook of Measuring System Design*, 2005.

[3] Barnett, "Controlling DC Motors Using Python With a Raspberry Pi", Business Envato Tuts+, 2014. [Online]. Available: https://business.tutsplus.com/tutorials/controlling-dc-motors-using-python-with-a-raspberry-pi--cms-20051. [Accessed: 16- Aug- 2018].

[4] "Sirius microSystems - PWM programming activity", Siriusmicro.com, 2018. [Online]. Available: https://www.siriusmicro.com/chrp3/pwm-c.html. [Accessed: 16- Aug- 2018].

[5] Llamerada, “Using Socket.IO C++ Client”, Hatena Blog, 2015. Available: http://llamerad-jp.hatenablog.com/entry/2015/05/22/151140. [Accessed: 22 August 2018].

[6] A. Errity, “Getting started with Node, Express and Socket.io”, Github, 2016. Available: https://github.com/aerrity/socket-click-example. [Accessed: 18 August 2018].

[7] “Introduction”, Socket.io, 2018. Available: https://socket.io/get-started/chat/. [Accessed: 15 August 2018].

[8] "I2C tutorial", Robot-electronics.co.uk, 2018. [Online]. Available: http://www.robot-electronics.co.uk/i2c-tutorial. [Accessed: 06- Sep- 2018].

[9] "How do I power my Raspberry Pi", MODMYPI LTD, 2015. [Online]. Available: https://www.modmypi.com/blog/how-do-i-power-my-raspberry-pi. [Accessed: 06- Sep- 2018].

[10] S. Kovacik and D. 4 Wheel Drive, "4 Wheel Drive, DC Motor Control for Raspberry Pi", Hackster.io, 2018. [Online]. Available: https://www.hackster.io/RedMnMGuy/4-wheel-drive-dc-motor-control-for-raspberry-pi-1300c4. [Accessed: 06- Sep- 2018].

[11] Staff, "Exponential Moving Average (EMA)", Investopedia, 2018. [Online]. Available: https://www.investopedia.com/terms/e/ema.asp. [Accessed: 07- Sep- 2018].

# Appendices

## Appendix A: Gantt Chart

