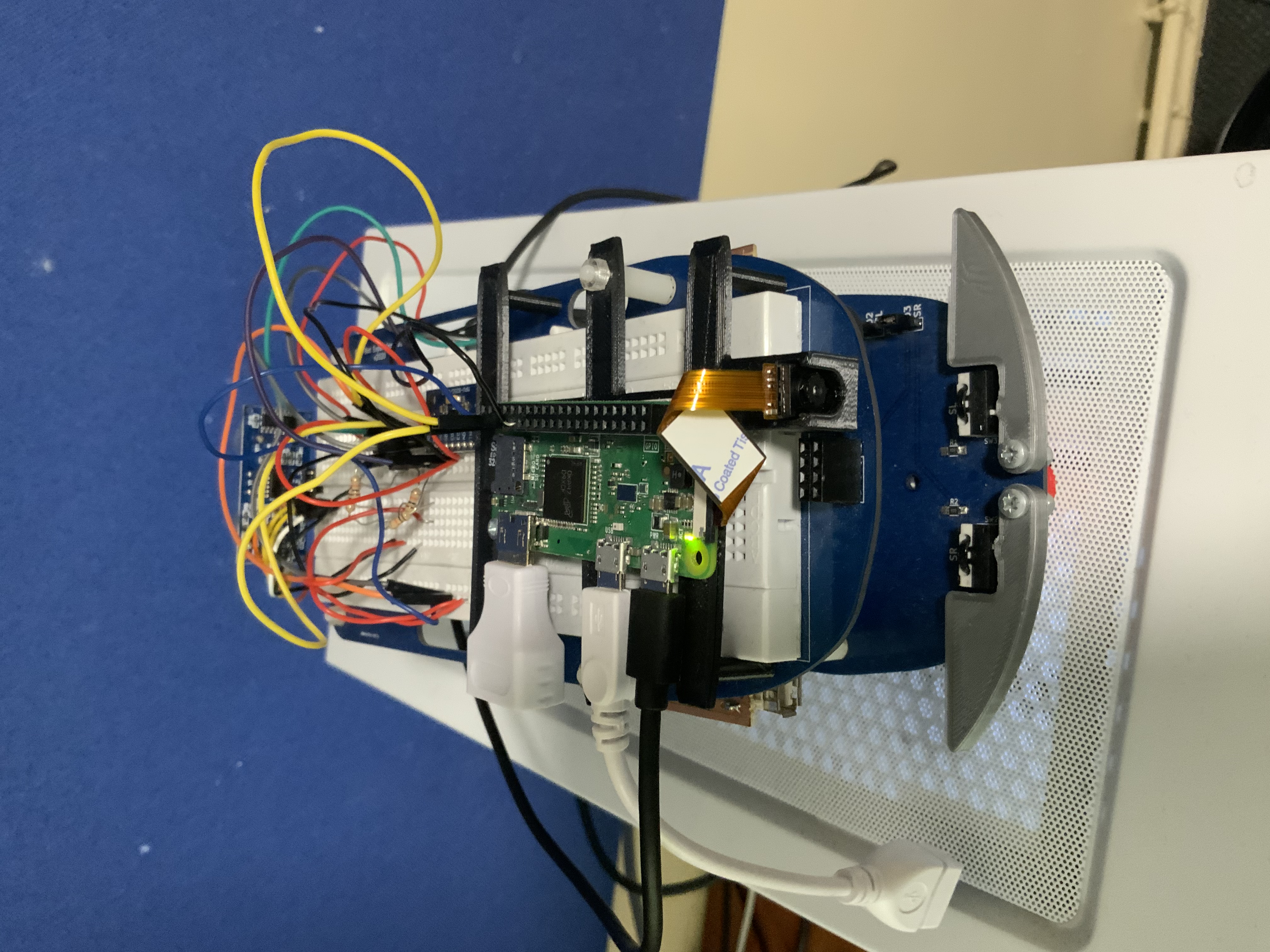
# Lab Report 3: Raspberry Pi and OpenCV



\

## Abstract

This document will explore and explain the implementation of image recognition technology using a raspberry pi zero with a camera module and OpenCV to isolate the key features on an image, resize the image then force the vehicle to perform an action based on the recognised image such as rotating a certain angle or moving forwards. It does this by communicating with the Arduino through i2c communications. However, since the Arduino and the raspberry pi zero operate on separate voltages, a level shifter is required for them to talk to each other without overloading one of the systems. Moreover, the system uses an MPU6050 to determine its current rotation compared to the angle it was facing previously. The MPU6050 as speaks to the Arduino through i2c communication which means that the Arduino needs to be able to recognise the different input from the different devices which created a unique issue which is also solved by the implementation of the level shifter. This technology is used in real life systems to avoid obstacles or even fully automate vehicles such as the latest tesla technologies that analyse the road and navigate whilst using computer imaging to avoid all possible obstacles and keep the passenger as safe as possible.

Contents

[Lab Report 3: Raspberry Pi and OpenCV 1](#_Toc72336020)

[Abstract 2](#_Toc72336021)

[Introduction and Background 2](#_Toc72336022)

[SAE level of autonomy 3](#_Toc72336023)

[Raspberry Pi 3](#_Toc72336024)

[Development and implementation of a voltage level shifter 3](#_Toc72336025)

[OpenCV 5](#_Toc72336026)

[OpenCV colour detection 5](#_Toc72336027)

[OpenCV shape counter 7](#_Toc72336028)

[OpenCV Image recognition 7](#_Toc72336029)

[Line following with computer vision 8](#_Toc72336030)

## Introduction and Background

OpenCV is an open-source software created for the intent of providing a library of real time computer vision processes for use in C++ and python. It originated in 1999 created by Intel corporation for free use under the open-source Apache 2 licence. For these reasons, this project will be utilizing it to allow a raspberry pi zero to recognise basic images and output data based on the input from the camera module. Furthermore, this the sake of simplicity functions created by Dan Fallows are utilized in the software.

The raspberry pi zero is a stripped-down raspberry pi developed in 2015 by the raspberry pi foundation in 2015. The raspberry pi foundation was originally started as a company dedicated to teaching basic computational electronics to school children but has since evolved into a company dedicated to creating microcomputers which are easy to access and can achieve a multitude of tasks from controlling robotics to providing ease of access of home servers. They are used by hobbyists worldwide as they are great for automating home systems due to the large number of programmers around the world creating a multitude of free libraries and working to improve on operating systems such as Raspbian which is the free to use Linux based operating system that this project uses to run code blocks and hence operate OpenCV.

Nowadays image recognition systems are used everywhere from airport security and phone security systems which make use of facial recognition to self-driving vehicles and stock inventory systems which can identify an object or objects and perform an action based on the recognised object. Usually, these systems would require more powerful computers to operate however by breaking down the image into its core assets we can use a much less powerful system such as the raspberry pi zero however there is a constant issue of making sure that the system is not overloaded.

### SAE level of autonomy

The EEE-Bot operates at two different stages. The first stage is where the robot can recognise images and turns either at different increments or moves in a direction for a period based on the quantity of shapes in the image. This stage operates at the second SAE level of autonomy referred to as partial automation. Partial automation is defined as a vehicle that is automated in steering and acceleration but requires human input. In this case the human input is the person holding up the image to the vehicle’s camera. The second stage is where the robot can follow the line based on image recognition software. This stage is the 5th level of SAE autonomy called Full Automation as it makes the car fully automatically follow the line without any need for human input.

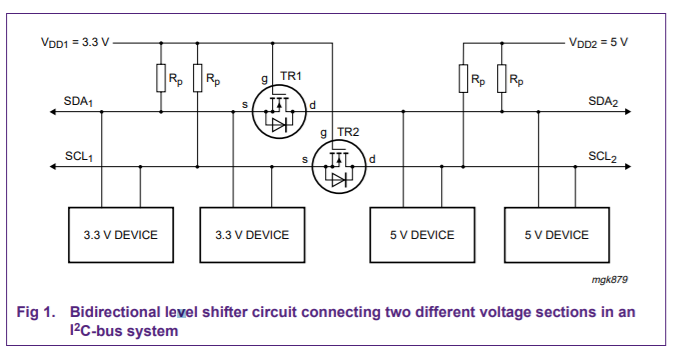
## Raspberry Pi

The raspberry pi is a microcomputer like the Arduino however, unlike the Arduino it is much more powerful and can be run separate from any other computer once it has been setup. Whilst the Arduino must be programmed from an external computer using a specific programming language, the raspberry pi zero has its own memory card for permanently storing information and as such can store its own operating system. The raspberry pi also comes with a micro-HDMI port and two USB type c ports one for power and another for external hardware such as a mouse or keyboard. Meanwhile, the Arduino micro only has one micro-USB slot that can be plugged into a computer. The Broadcom BCM2835 SoC used in the raspberry pi zero includes a 700 MHz ARM1176JZF-S processor, VideoCore IV graphics processing unit (GPU), and RAM whereas the Arduino nano only has an ATMega328 and is therefore a lot worse at processing higher level programs. Image recognition software requires a large amount of memory and processing power and hence an Arduino nano would not be capable of performing such a complex task. However, if we were creating a simpler program the Arduino would be a lot better at the task as it is much smaller, cheaper, and more physically durable than the raspberry pi.

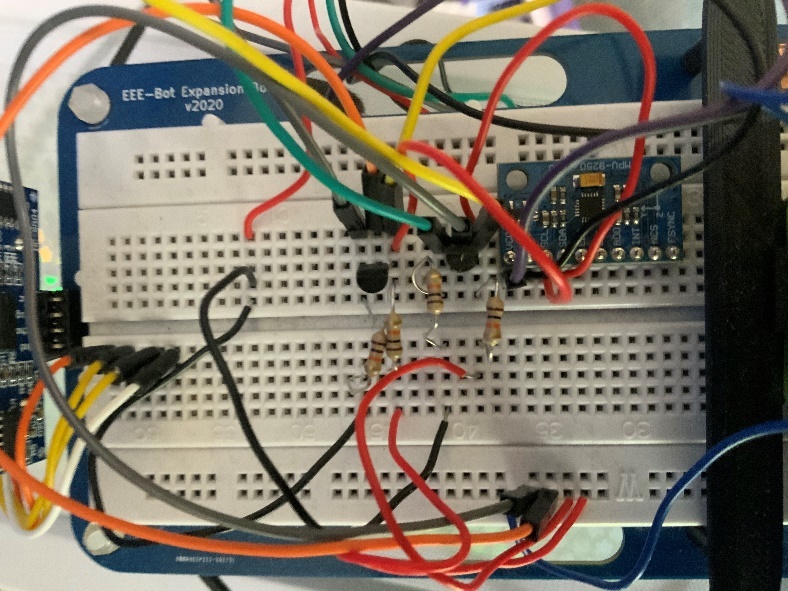
### Development and implementation of a voltage level shifter

In this project we use both a Raspberry pi and an Arduino to process the information from a camera and communicated between each other to send information. In this case the raspberry pi processes the information from the camera and the Arduino takes the output from the Raspberry pi and converts it into the appropriate instruction for the motors on the vehicle using information also communicated through i2c from the MPU6050. The voltage level shifter makes use of two N-channel enhancement MOSFETs and four pull up resistors connected such as **Figure 1**. This configuration allows up to two devices to be connected to the SDA and SCL lines so that they can communicate to each other through i2c communications without being overloaded. The voltage shifter is implemented in real life according to **Figure 2**.

**Figure 1**

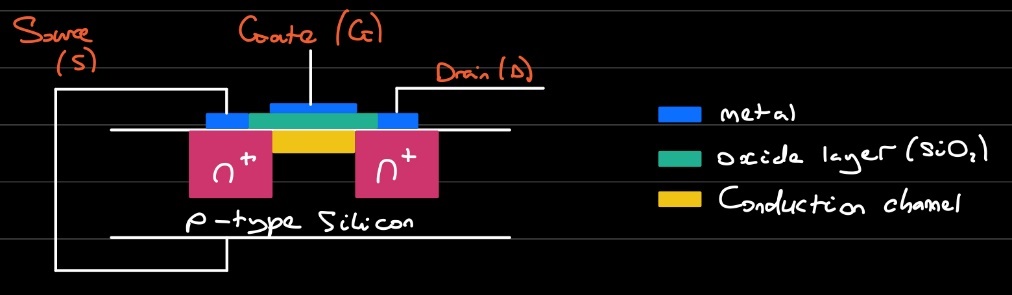


**Figure 2**



**Figure 3** shows an N-channel device formed between source and drain. Bother n and p-channel devices come in two forms which are distinguished by channel formation and operation. Enhancement mode devices has no ‘built in’ channel and no gate voltage whereas in depletion mode devices conduction occurs when the gate voltage is equal to zero which means that the channel is normally on. N-channel enhancement is the most common MOSFET as they are much faster than p-channel devices. The operate like JFET devices but the oxide layer but the oxide layer is insulating, and charge does not transfer from one side to the other. Just like the JFET, a gate voltage is used to create a pathway for current to flow from source to drain.

**Figure 3**



There are three states that should be considered during the operation of the voltage level shifter. The first is when no device is pulling down the bus line. In this scenario the bus line of the ‘lower-voltage’ section is pulled up by the pull up resistors to 3.3V. The source and the gate of the MOSFET are both at 3.3V so the Gate-Source voltage is below the threshold voltage and the MOSFET is not conducting. Therefore, the bus line at the ‘high voltage’ area can be pulled up by the pull up resistor to 5V. This means that both sections are ‘high’ but at a different voltage level. The second state to consider is when a 3.3V device pulls down the bus line to a low level. In this case the source of the MOSFET also becomes low whilst the gate stays at 3.3V. The Gate-Source voltage rises above the threshold and also pulled down to a low level by the 3.3V device via the conduction MOSFET. So, the bus lines of both sections go low to the same voltage level. The third and final state is where a 5V devices pulls down the bus line to a low level. In this situation the drain-substrate diode of the MOSFET (the ‘lower voltage’ section) is pulled down until the Gate-Source passes the threshold and the MOSFET starts to conduct. The bus line of the ‘lower-voltage’ section is then further pulled down to a low level by the 5V device via the conducting MOSFET. So, the bus lines of both sections go low to the same voltage level. These three states show that the logic levels are transferred in both directions of the bus system, independent of the driving section. State one performs the level shift function. States 2 and 3 perform the ‘wired-AND’ function between the bus lines of both sections as requires by the i2c-bus specification. **NOTE TO SELF INCLUDE CODE HERE**

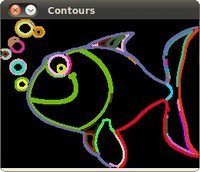
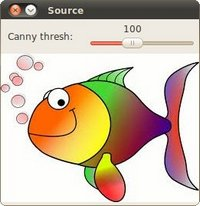
## OpenCV

OpenCV is an open-source software that was utilized in this project for the sake of providing libraries to all for easier image recognition. To access OpenCV the raspberry pi was set up to be able to use code blocks for Raspbian. For this to be possible the raspberry pi had to be set up with SPI And VNC protocols which allows the user to access the pi through the command window and the visual desktop respectfully. The programs Putty, VNC viewer and FileZilla were for ease of access of the raspberry pi.

### OpenCV colour detection

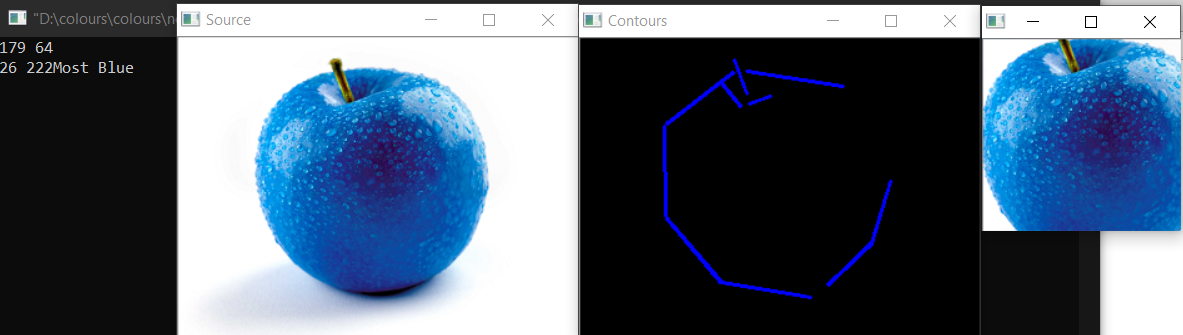
The first program designed in OpenCV on the pi was a program that could isolate the main colour of any image put into it. The first stage of this program was to isolate whether the main colour was red, green, or blue in the image. However, to achieve this the HSV values must be determined by either an alternative program or user detection. HSV stands for hue, saturation and value which are the three qualities that define every colour of light. Hue is defined by google as “the attribute of a colour by virtue of which it is discernible as red, green, etc., and which is dependent on its dominant wavelength and independent of intensity or lightness” which in general terms means how much of the total light wave is red, green, or blue light. Therefore, generally, the entire colour can usually be defined entirely by the hue. Saturation is defined by google as “the intensity of a colour, expressed as the degree to which it differs from white.” Therefore, this is more useful for determining the strength of the colour so the acceptable range of which OpenCV defines a colour. Higher saturation is good for isolating the focus of the image. However, a lower saturation is good for allowing a larger range of colour as most real-life colours are not a solid block of colour but are in fact made up of many varieties of the same colour. Finally, there’s value; value as defined by the New York time is “Value defines how light or dark a given colour or hue can be. Values are best understood when visualized as a scale or gradient, from dark to light. The more tonal variants in an image, the lower the contrast. When shades of similar value are used together, they also create a low contrast image.”. This is useful for deciding where in the colour spectrum you want to detect. If you want a light green you use a high value and if you want a darker green you use a low value. After the HSV values for the different colours to be identified are decided the next step is to simply make use of the “cvtColor” and “inRange” functions in OpenCV to create a copy the image and convert and apply the correct HSV filters to the image respectfully. If looped through once per colour option, we can count the number of different pixels for each converted image. The converted image with the largest number of white pixels must therefore be the image of the colour converted. By all logic, this system works however there is an issue that most images have backgrounds that could be a more prominent colour than the focus of the image. This means that while the program could be fully operational the system may still detect an image as the incorrect colour due to this fault. To solve this issue, we can identify the focus of the image using contours and crop the image automatically inside of the contours to find the colour of the focus of the image quickly and accurately without worrying about the colour of the background. Firstly, we blur the image using the “Canny” function between the original image and a grey version of the original image. Then we can use the “findContours” function to identify any lines in the images, the contours are stores as an array called Contours and applied to the image to display an image such as showing in **Figure 4**. The canny threshold controls the number of contours on the resulting image. This is as it determines how blurred the image is. The more blurred the image, the less obvious the contours hence only the larger contours remain on the image.

**Figure 4**



However, despite the few remaining contours being quite large there is still an issue which is that most contours are still mostly quite small due the jagged edged and therefore it is difficult to identify the largest one. A solution to this is to simplify the contours which converts all the contours into straight lines making it easier to identify the bounding edges of an image. Simplifying contours works through some complex use of vector functions but the result is several straight lines which surround and identify the image. The next step is the cropping of the image. The cropping function works by taking a few inputs including the coordinated of the upper left pixel of the image as well as the dimensions of square to be cropped out of the image. By finding the position of the furthest left, right, highest, and lower midpoints of the contours we can get a good idea of where the object of the image is by using the furthest left coordinate and the highest coordinate and the different between that point and the furthest right and lowest coordinate, we can crop a section of the inputted image to represent the colour of the focus of the image more accurately. This is demonstrated in **Figure 5**.

**Figure 5**



### OpenCV shape counter

The shape counter uses a lot of the same concepts as the cropping section of the colour detection program. Just like the colour detection program the contours of the shapes are found however instead of cropping the image it is required to count the number of shapes. This can be done by applying the “findContours” function to the matrix of the image. If the contours were then counter the program could hypothetically find the number of shapes in the image. However, since each shape does not count as one contour this is not possible. To this this issue you can **SOLUTION HERE**.

### OpenCV Image recognition

The next program is designed to recognise an image and send instructions to the vehicle to perform an action based on the image that matches with the appropriate image in the library. This programming solution can be broken down into individual problems. The first problem is that the image will not be necessarily shown to the camera at the correct angle to compare it. The second issue is recognising the image and processing to output the correct output. The third issue is communicating between the raspberry pi and the Arduino. Fourthly the solution must be able to read data from the MPU to recognise the angle it is currently facing. And finally, the solution must be able to combine data from the MPU and raspberry pi to send a signal to the motors to correctly perform the correct action.

The solution to the first problem as with the solution to many of the previous problems is contours. This time we must find the biggest contour from the image and assume it is the border of the image. We can use this border to rescale the image based on the size we know the picture to be. Finally, we can use a compare images function to see whether the images are more than 75% similar. The first step is to isolate the pink out of the video, so we are only left with the actual image. To do this we apply the HSV filter to filter out everything that is not pink based on predetermined values deemed to be pink. Next yet again, to find the contours we must first blur the image as so that only the lines surrounding the image remain. This allows us to compare the blurred image to the original and are left with an array of vectors that make up the contours. We can display these contours on an empty matrix to make it easy for a user to see however this is not necessary for the program to function so it is best not to so that we can save processing power for more complex functions. Now that we have our contours, we can simplify them then loop through the array, calculating the area for each contour to find the largest one. This contour is therefore assumed to be the border of the image. Then the “transformPerspective” function to change the perspective of the image so that it can be compared to the version we have stored on the raspberry pi already. To solve the second problem, we simply loop through all our predetermined images and compare them to the image generated by the “transformPerspective” function. If any of the images output a percentage higher than 75% then they are deemed to be the same image and the program outputs the correct identifying integer for a match with the respective image. This integer is then communicated to the Arduino which is where problem three is encountered. To communicate with the Arduino, the raspberry pi must use a i2c communication program. Normally for i2c to work it must send the data across one bit at a time however since the raspberry pi is only transferring one integer it only takes up one bit worth of data. The raspberry pi uses the “wire.h” library and connect to the Arduino over port 17. As the master the raspberry pi could be made to search for the port that the Arduino is connected to however as we programmed the Arduino to connect to port 17 there is no point in searching. On the other end of the connection the Arduino is constantly listening to receive any data from the raspberry pi. When it receives data, it will process it to do one of a multitude of things. If it receives an integer that makes it rotate 90 degrees, it contacts the MPU which is the fourth issue. The Arduino is already a slave to the raspberry pi, but it would have to be a master to the MPU to send it a signal to receive the rotation data. The solution to this is to make the Arduino a slave and a master intermittently. If the Arduino is a master to the MPU long enough to read and extract the data everything will function normally as the motors cannot perform two functions at once anyway. This may become an issue in the future where the Arduino does need to take two inputs at the same time from the raspberry pi but the solution to that is to simply make the raspberry pi the slave and have the Arduino constantly check it for potential inputs.

### Line following with computer vision

There are many potential ways to follow a line using computer vision. One potential way would be to recognise the image of the line by having a picture of a line a comparing the percentages of image and the stored image. Another way is using colour recognition to filter out the colour of the line and make it so that there is constantly a certain number of white pixels in the image; then if there are white pixels on left side of the screen then it would turn left and vice versa with the right. This would allow the car to accurately turn however it would take too long to count the pixels in the entire image every time the car needed to turn, and it would end up off the track. Another solution is to use contours to find the edges of the lines and determine the area of the contours. If one of the two major contours increases in area it must be because the line is curving in the opposite direction, as a result the car must also turn by an amount calculated by finding the angle between the simplified contour and the straight contour **Figure** then tell the motor on the line with the largest area to rotate at a speed equal the speed of the other wheel plus the angle divided by the total possible angle multiplied by the other wheel’s speed. While this method sounds good in theory it might not entirely work in practise as the time it takes to send information to the wheels, especially a value with multiple digits, would be significant due to the processing speed and speed of transmission of i2c. Due to the possibility it might not function, a new approach has to be taken.

**Figure**

