## Introduction

This report consists of three tasks designed to introduce the concept of computer vision using relatable contextualised scenarios. The tasks all use OpenCV with C++ and use predefined datasets which have been analysed. To extend the analysis additional datasets have been created where required and these are clearly identified. The three tasks are based on three key computer vision elements, colour identification and sorting, colour tracking, and cross correlation.

Contents

[Introduction 1](#_Toc129277359)

[List of Figures 1](#_Toc129277360)

[Task 1 – Colour Sorter 2](#_Toc129277361)

[Task 2 – Colour Tracker 6](#_Toc129277362)

[Task 3 – Cross Correlation 11](#_Toc129277363)

[Appendix 14](#_Toc129277364)

[Flowcharts 14](#_Toc129277365)

[Video Links 17](#_Toc129277366)

[References 17](#_Toc129277367)

## List of Figures

[Figure 1 - Code extract showing the analysis of a pixel colour 2](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277203)

[Figure 2 - Flow chart showing the for loop through each pixel. 3](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277204)

[Figure 3 - Original Dataset for Task 1 3](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277205)

[Figure 4 - Additional Dataset for Task 1 4](#_Toc129277206)

[Figure 5 – An incorrect analysis, original image and pure RGB reconstruction. 4](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277207)

[Figure 6 - no blur vs Gaussian blur 5](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277208)

[Figure 7 - False Positives 5](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277209)

[Figure 8 – Showing the difference between the original frame (left), the HSV frame (right) and the filtered frame (middle). 7](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277210)

[Figure 9 - Diagram showing the HSV colour scale [3]. 7](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277211)

[Figure 10 - The results of the initial limits testing with the inRange() function. 8](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277212)

[Figure 11 - A colour picker showing the target colour 8](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277213)

[Figure 12 - Showing the use of circle() and line() 9](#_Toc129277214)

[Figure 13 - Pythagoras to find the angle. 10](#_Toc129277215)

[Figure 14 - Pendulum swing over time (original data) 10](#_Toc129277216)

[Figure 15 - Recreated Video 10](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277217)

[Figure 16 - Showing pivot location also detected. 11](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277218)

[Figure 17 - Pendulum swing over time (original data) 11](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277219)

[Figure 18 - Showing the rectangles drawn when an error is detected, or a component is found. 13](file:///C:\git\AINT308\C1%20Writeup.docx#_Toc129277220)

## Task 1 – Colour Sorter

**Context Statement: An online car dealership has a database of cars for sale and would like to give customers the ability to search based on the cars colour. To do this, all cars in the database will have to be tagged with a colour. The database is too big to do this manually, so can you create a computer vision solution to identify the colour of each car?**

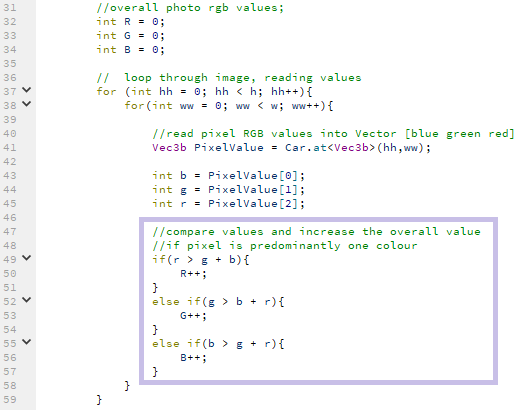


Figure - Code extract showing the analysis of a pixel colour

Following the three main requirements in the Colour Sorter task, this solution has a simple colour sorter model that displayed each car and could correctly identify the colour of the car, printing this value to the terminal. The full code and a video of the full solution for this initial working attempt can be found in Appendix 1, specific sections have also been highlighted in the report below. A video showing the working solution can be found in the appendix as well.

Each image is analysed by looping through each pixel by using two for loops, stepping through the width and height values that are stored in variables w and h. This iterative process is shown in Figure 2 and ensures every pixel is examined in the same way.

To perform the analysis, the RGB values are read with the line Vec3b PixelValue = Car.at<Vec3b>(hh,ww); which saves the values in PixelValue. These values are then copied into local variables b, g and r to make the code more readable. Although the colour space used is colloquially called RGB, OpenCV reads the channel order as BGR and so by renaming the values it became easier not to make accidental errors reading the values.

Using these three values, the next step was checking which, if any, of the colours is most prominent. By comparing the three values and choosing which value is the highest, it is possible to identify the most prominent colour. This method can be seen in the code extract below, for each pixel, the red value is compared to the sum of the blue and the green values and if it is higher, the R variable is incremented. If the red value isn’t the dominant colour, then the green value is checked in the same manner and then finally blue as well.

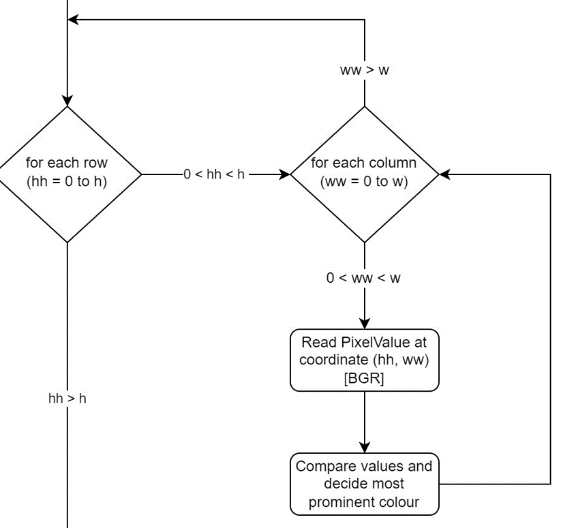


Figure - Flow chart showing the for loop through each pixel.

If one of the colours was dominant, the relevant value R, G or B would be increased. This method is used so that it is possible to keep track of how many times each colour was identified across the entire image as it was processed.

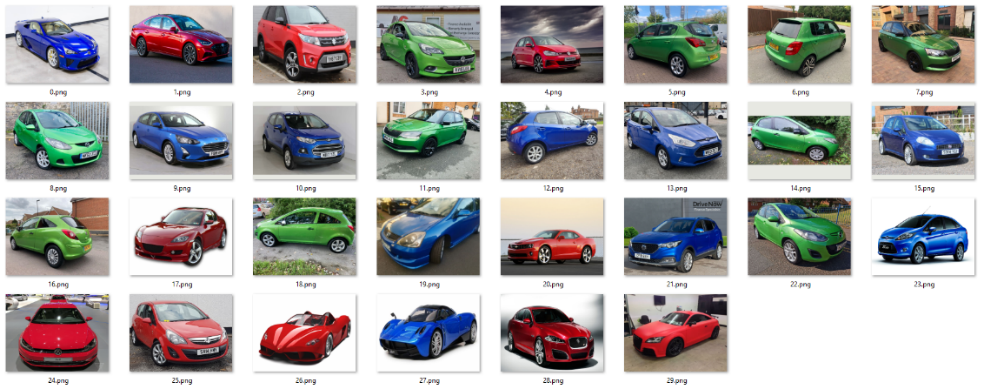
After each pixel had been analysed, the R, G and B values are then compared to each other to find out which was the largest. Finally, a string variable colour is defined according to this largest value and printed the output to terminal.

Text, application, chat or text message

Description automatically generated

This method worked effectively and correctly analysed all thirty cars from the initial dataset however these images were carefully handpicked to simplify the computer vision requirements. They were all taken in good lighting, with simple backgrounds and were all bright red, green or blue cars.

Figure 3 - Original Dataset for Task 1



The images had also been cropped to ensure the car was the main element in the picture. This means that although this dataset can be analysed accurately with this model, it might not work so well with different images, for example an image with a blue car on grass might have more green pixels than blue. To test this, a second dataset is generated with images of red, blue, and green cars downloaded from the internet. This dataset purposefully did not crop all the photos quite as close and introduced more varied lighting and background options but still only contained red, green, and blue cars.

Graphical user interface, website

Description automatically generated

Figure - Additional Dataset for Task 1

Text

Description automatically generatedThis time, the analysis calculated 19 out of 30 cars incorrectly and displayed the wrong colours. For example, car 3 in the second dataset is identified as green due to the grass and hedgerow surrounding the car. This example is one of the brighter and more cropped images in dataset two so this shows how easily fooled the initial model was. To see which areas of the images were being defined as dominant colours, a few lines of code are added to the classification technique. This produced the pure RGB image as shown in Figure 5.

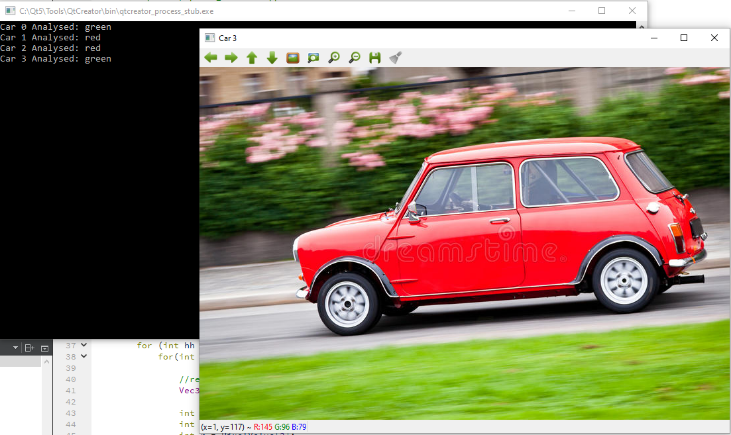
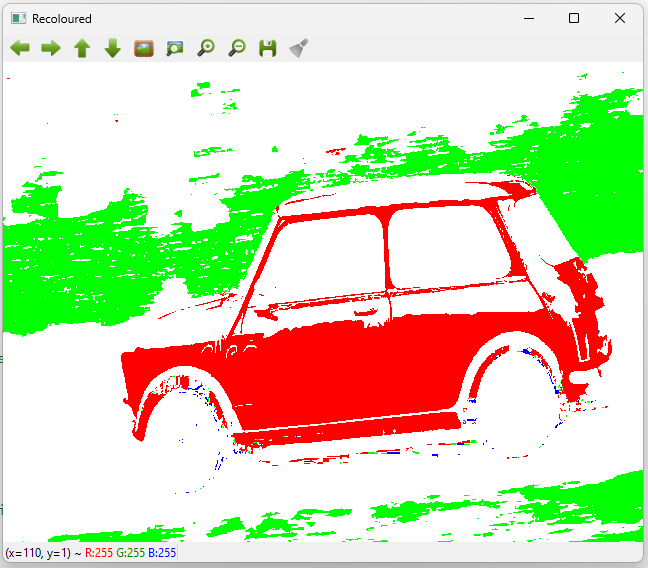


Figure 5 – An incorrect analysis, original image and pure RGB reconstruction.

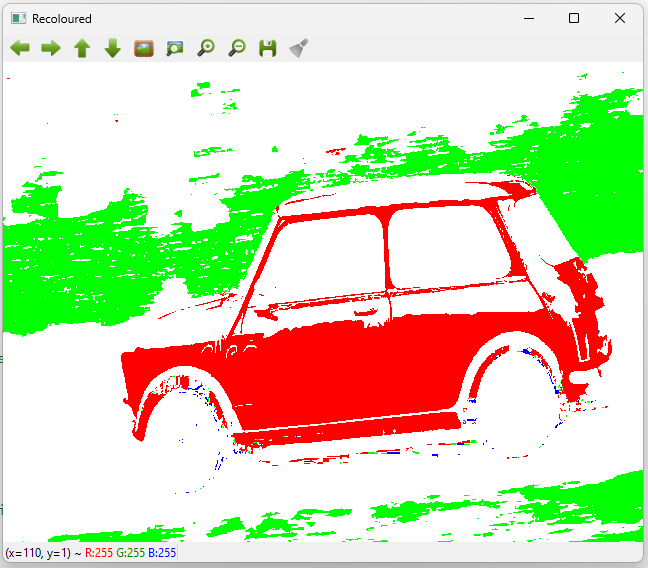
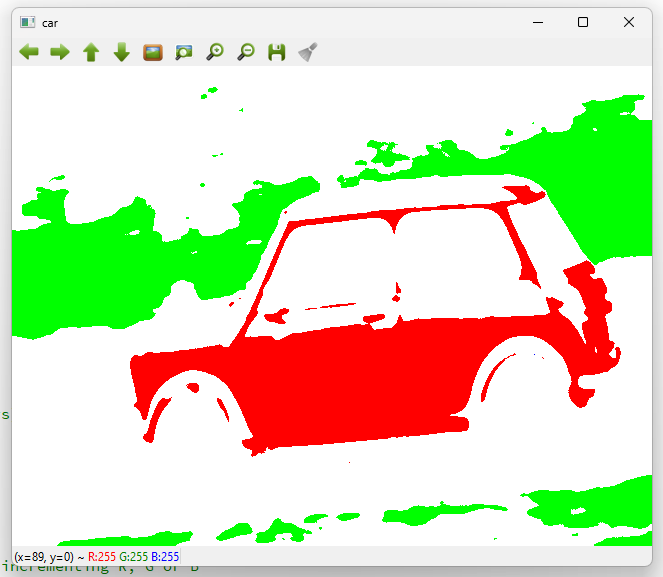
It is possible there were also some false positive identifications in this dataset as in scenarios such as a green car in front of grass, or a blue car near water, the background might have occupied more pixels than the car object and therefore with an alternative car in the same scenario, the model might have identified the colour incorrectly. The premise behind this false positive identification has been demonstrated below with a simplified image.

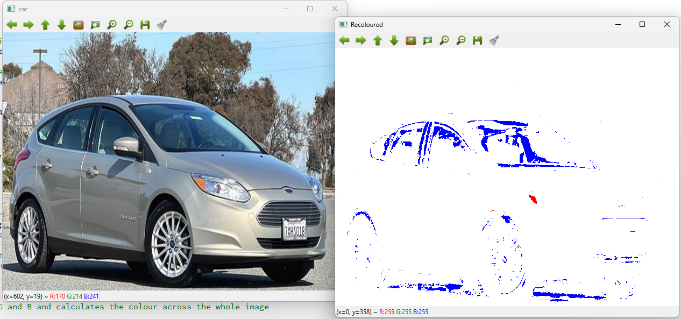
Figure - False Positives



To improve this computer vision model to work with more complicated scenarios implementing some filtering into the solution was necessary. By adding a gaussian blur to the original image, the amount of variation in a small area is reduced and the image is smoothed out. A gaussian blur is more computationally expensive than a simple mean blur, however it produces a more natural result than a mean blur, much like defocusing a camera. Each pixel is replaced with the average of surrounding pixels, each weighted based on a gaussian distribution. The results from the pure RGB reconstruction make it obvious how much difference to a human eye but it wasn’t a drastic enough improvement to show in the analysis results.

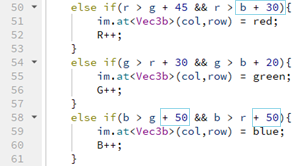
Figure - no blur vs Gaussian blur



The next improvement to make was to try and improve the classification of colours. One of the images in the secondary dataset was included on purpose to see if the result “unclassified” was possible as it wasn’t a red, blue or green car and is instead grey. In the first run through, this car was returning a blue value as the edges of the car were identifying as blue. This suggested the method either needed a minimum number of pixels to identify the colour or needed a minimum value to stop a colour close to black being identified as either red, green or blue.

Text

Description automatically generatedThe first attempt to create an undefined answer worked but highlighted a different issue. This time, whilst the grey car was correctly identified, two dark green cars were identified as undefined as well. This seemed to be because the dark green colour has almost as much blue in it as it had green.

At this point it made more sense to adjust the thresholds for each colour independently because RGB and human vision don’t correlate well. The blue channel was causing the most errors so the amount of blue that the red and green channels had to be larger by was reduced, and amount that blue had to be larger by was increased.

After this final set of adjustments, running the task gave 100% correct answers. This once again would need to be tested with a larger dataset as the background of the image could still cause errors but is a good improvement from the 19 out of 30 original cars identified correctly. It is likely that more filtering could help make this task easier as well but, in the end, the best system would likely be one that could remove the background or detect the car and only compare these pixels.

### Task 2 – Colour Tracker

**Context Statement: A researcher wishes to use a pendulum for an experiment but needs to know the friction of its pivot. One way to measure this is to swing the pendulum and observe the decay rate of the oscillation. To do so, the angle of the pendulum must be measured at small time intervals. A camera was placed in front of the setup, and a bright colour target was attached to the end of pendulum. Write a program to detect the colour target and measure the angle of the pendulum over time.**

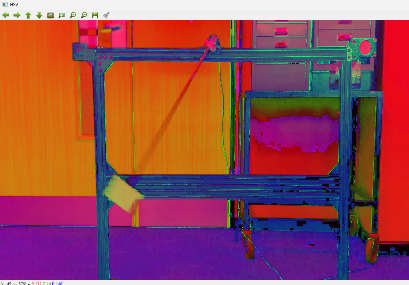
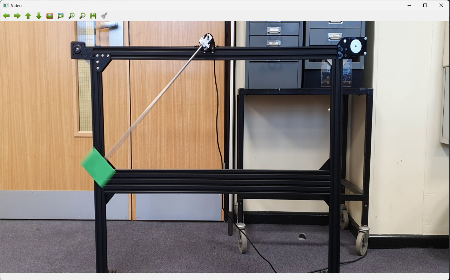
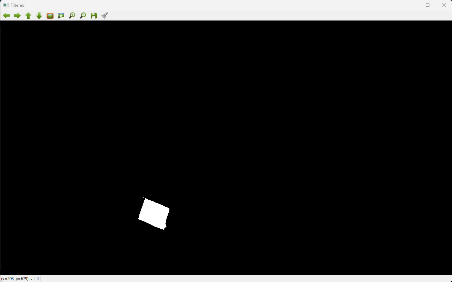


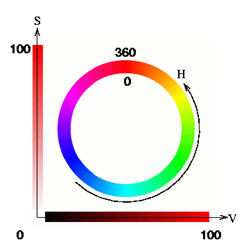
Figure 8 – Showing the difference between the original frame (left), the HSV frame (right) and the filtered frame (middle).



The first part of this task was to convert the colour space to the HSV colour space. As this task relies on colour detection frame-by-frame, HSV makes it easier to identify the object by separating hue from the luminance. By loading each frame of the video individually, the frame can be filtered to identify the target area and highlight this section of the frame. This conversion uses the cvtColor function and to convert from RGB in OpenCV to HSV colour space the COLOR\_BGR2HSV code is specified.

Text

Description automatically generated

Text

Description automatically generated

Unlike RGB which is designed for computers to display colours, HSV is a model that is designed to be easier for humans to understand. These dimensions are hue, saturation, and value.

Hue specifies the angle of the colour on the RGB colour circle. A 0° hue results in red, 120° results in green, and 240° results in blue. In OpenCV, the scale is only 0-180° so 0 is red, 60 is green and 120 is blue.

Saturation controls the amount of colour used. A colour with 100% saturation will be the purest colour possible, whilst 0% saturation has no pigment and will be on grey scale, depending on the value.

Figure - Diagram showing the HSV colour scale [3].

Value controls the brightness of the colour. A colour with 0% brightness is pure black while a colour with 100% brightness has no black mixed into the colour. In OpenCV, both saturation and value percentages are scaled to fit between 0 and 255.

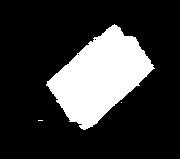
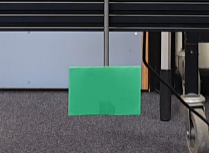
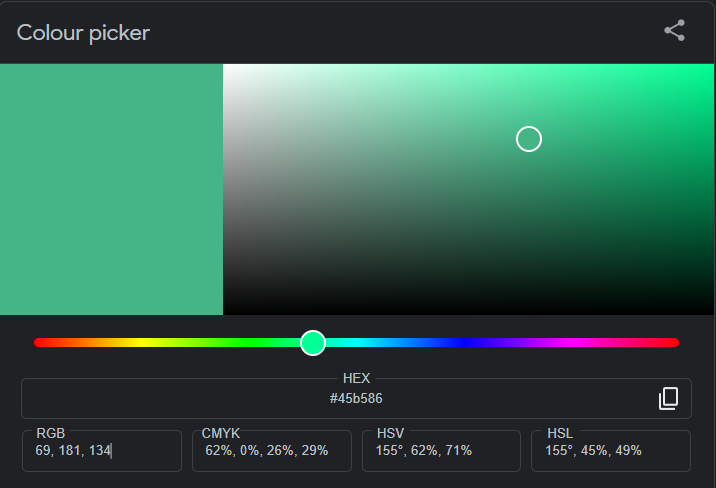
In this scenario, the green target is going to fall between 30 and 90 degrees on the Hue scale, the saturation is fairly strong and there is a lot of colour and not much black. This means that the first limits tested were lower (30, 50, 50) and upper (90, 255, 255). Using these limits alongside the inRange() function, it is possible to output a frame showing any pixels that are within the target range in white and the any areas not within this range in black. This is called thresholding and is a method used to select certain sections of an image. These initial limits provided a good result with only the target showing in the filtered frame. There were however still occasional spots from the background which flickered so more adjustments were made. By using a colour picker, it was possible to determine the RGB value of the target and translate this value into HSV value. As can be seen below, there was blue as well as green in the target so the limits needed to be adjusted to lower (65, 100, 100) and upper (85, 255, 255) which reduced the flickering to a minimum.

Figure - The results of the initial limits testing with the inRange() function.

Text

Description automatically generated

Figure - A colour picker showing the target colour



To make use of this filtered target, an image moment is calculated. Image moment is a particular weighted average of image pixel intensities, with the help of which we can find some specific properties of an image, like radius, area, centroid etc. In simple terms, image moments are a set of statistical parameters to measure the distribution of where the pixels are and their intensities. Mathematically, the image moment Mij of order (i,j) for a greyscale image with pixel intensities I(x,y) is calculated as

Here, x, y refers to the row and column index and I(x,y) refers to the intensity at that location (x,y). [2]

In this situation, because the filtered frame is a binary image (and therefore all the pixels have the same intensity), the zeroth order moment will correspond to the area and therefore the number of pixels. In the below code, m.m10 represents the sum of the x-coordinates and m.m01 represents the sum of all the y-coordinates and these are both divided by the total number of pixels to calculate the centroid.

A screenshot of a computer

Description automatically generated with medium confidence

The next section of the task was to draw some markers on the image to show the position of the pendulum. Using the recently calculated centrePoint this becomes simple using the OpenCV functions circle and line. For line, two coordinates are needed so the Pivot location is used. This co-ordinate was pre-defined in the demo code but was adjusted to be more accurate when a visual check showed it to be slightly off.







Figure - Showing the use of circle() and line()

To calculate the angle of the pendulum, the pivot and the centrePoint positions of the pendulum are used once again. Using Pythagoras, the angle can be calculated that the from the horizontal and vertical distances which is shown in Figure 9. These distances are calculated using the pivot point and the centre of the pendulum.

Text, letter

Description automatically generated

A picture containing line chart

Description automatically generated

Figure - Pythagoras to find the angle.

To demonstrate how successfully the angle was calculated, the angles were printed to the terminal and to a .csv file at the end of each frame analysis. The results were then analysed in excel and plotted in a graph to show the swings over time. The context of the task was a researcher studying the decay in oscillation so it was important to check that the code and results created were still applicable to the original context. As can be seen below in Figure 13, there is obvious decay over time and the initial 45-degree swings reduce at a steady rate until they reach 0 at the end.

A picture containing text

Description automatically generated

Figure - Pendulum swing over time (original data)

To ensure repeatability, a second pendulum video was created. As the original setup was not available for this recreation, an alternative was created using a shoelace and a colourful belt. This however meant the target colour was different from the original video. This meant that the filtering theory could be tested once again with different values which means any unnoticed errors might be spotted. Another difference between the two videos is that in the original video the pivot point does not move. This could be because it was taken using a tripod or another method of stabilisation, however the second video was taken handheld and had a bit of movement in it. After an initial external stabilisation method was added to reduce the movement, the pivot position was still not completely stable which could cause errors in the calculations if not corrected for. To continue the colour tracking theme of the task whilst correcting this, a second target was created, this time targeting the pivot location. It was beneficial that in the video, the fingers holding the shoelace are visible as this created an obvious colour target to find as can be seen in Figure 14.

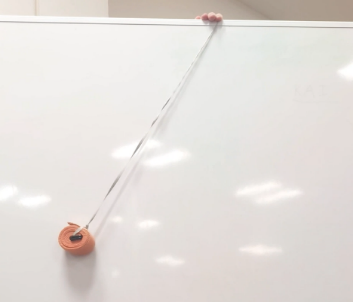


Figure - Recreated Video

Although all the methods used were the same as in the original dataset, the additional requirement to filter the frame for the pivot point created many changes to the code. This was unhelpful as ideally as improvements continued to be made, it should be possible to test both videos and analyse the benefits. To counter-act this, a variable video was created that was used to decide which code to run, this worked by having an if statement which checked the current status of the variable and then set certain parameters, such as upper and lower bounds for the filtering, based on this value. Throughout the code, the same technique was used again to separate the different methods required to calculate the angle as well. This method is shown in the flowchart and code snippet below, the two videos require different techniques to draw the line between the pivot and Diagram

Description automatically generatedText

Description automatically generatedthe centre point.

Once the issue of the pivot point was fixed, it was possible to check the results in the same way as the original video. The graph created once again shows a neat decay in oscillation from an original 45 degrees to finishing just above 0. This slight deviation is likely due to the accuracy of the pivot point being reduced. One of the issues noticed was that the HSV range chosen for the pivot point occasionally has specks appearing across the frame due to the reflective nature of the background. Another issue is that the shoelace actually is pivoting from the bottom corner of the hand whereas the centre is what has been used to define the pivot point. This is something that should be easily adjustable if the task was taken further.

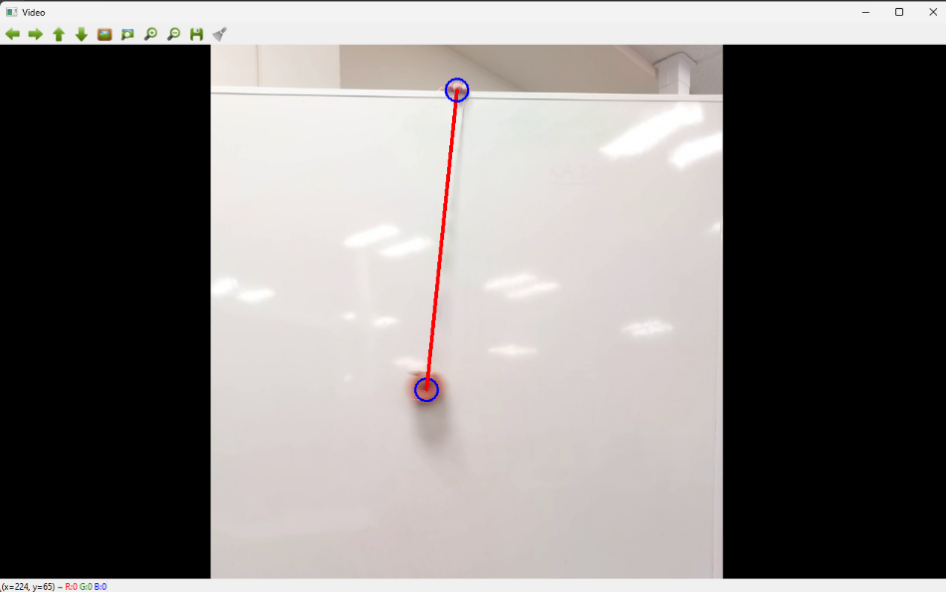


Figure - Showing pivot location also detected.

Chart

Description automatically generated

Figure - Pendulum swing over time (original data)

### Task 3 – Cross Correlation

**Context Statement: A PCB manufacture wants to automate part of its quality control, as a few key components have a high failure rate during the pick-and-place phase. Assure that all these components are in place by locating them on the PCB.**

Correlation is the process of moving a filter mask often referred to as kernel over the image and checking the outcome to identify the similarities. For this task, the requirement is to identify the area of the image that matches the kernel which can be performed by template matching. Template matching is a technique for finding areas of an image that match (are similar) to a template image and is simple to implement in OpenCV.

Text

Description automatically generatedFor this task, the initial element of the solution is to load both the PCB image and the target Component image so that they can be compared with the matchTemplate() function. This function is provided by OpenCV and requires arguments of the image, the kernel, the output, and the flag identifying the matching technique. For this task the chosen flag was TM\_SQDIFF\_NORMED to compare the images which stands for template matching square difference normalised. The next step was to use the minMaxLoc() function which checks a given image and identifies the minimum and maximum values in the image and returns the values and the locations of these values. For this task this is very useful as the location given by minimum value returned by the template matching is the most likely location for a successful match between image and kernel.

It is important to note that if an alternative flag was used, it could identify the most successful match with the maximum value in the image.

Text

Description automatically generated with medium confidenceOnce the minimum value has been identified, it was multiplied it by 100 to create a variable called error. This error value could then specify a specific threshold and a decision could be made to see if this was a small enough error and if this would indicate the component was placed on the board or not. After a few tests to see what the values returned were, a suitable threshold was decided to be 1. This gave the best results and within a few tests the solution was correctly identifying that 9 out of the 10 components tested were found on the PCB and one was missing.

To draw the rectangle on the PCB, the minLoc value determined by minMaxLoc() and the Component image size became very useful as ways to identify the size and location of the rectangle to be drawn. To make it easy to do this, a function called drawAroundComponent() was created to be called when needed.

Text

Description automatically generated

Text

Description automatically generatedTo confirm that this threshold was justified and wasn’t only working with the given dataset, the dataset was expanded, and the code edited to fit. An additional three PCBs, each with 10 components were added to the options and so the dataset was quadrupled in total. To ensure this would not cause errors, each dataset was in an individual folder which contained the PCB image and 10 Component images. This meant that the main change to the code was to add an additional for loop which introduced the variable m to indicate the folder number.

Without changing the threshold or any of the other code except the print to terminal commands, the first attempt to test this worked with a 100% success rate, identifying 6 components that weren’t found on the PCB. When these components were identified, the error value was also printed, and it can be confirmed that each of these values were over the threshold of 1.

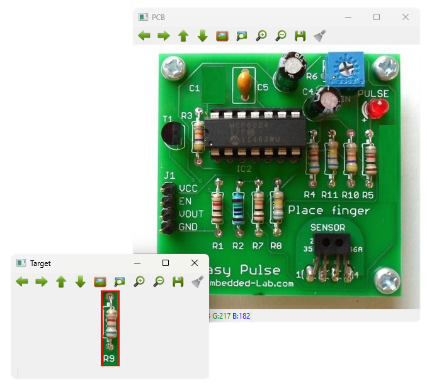
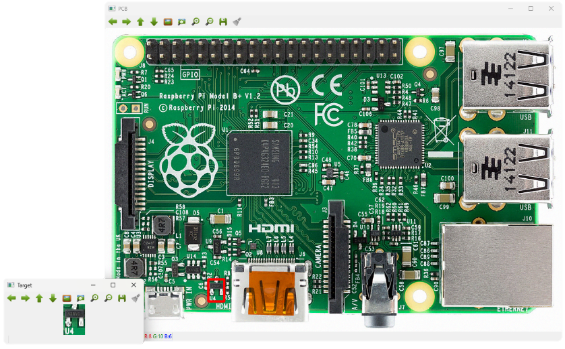


Figure - Showing the rectangles drawn when an error is detected, or a component is found.

The task also asked that the components be identified on the image by drawing around the found component. This is another way to check visually that the solution is working correctly. To expand on this, when there was an error, a rectangle was instead drawn on the component window.

To continue developing further, it would be good to expand the capabilities by looking at what happens if the component image and the component on the PCB are at different orientations, or if there are multiple of the same components on the same board. With the current solution, only the most identical solution would be identified.

Appendix:

### Flowcharts

Task 1:

Diagram

Description automatically generated

Task 2:

Diagram

Description automatically generated

Task 3:

Diagram

Description automatically generated

### Video Links

Task 1 - <https://youtu.be/VcIHozemFCA>   
Task 2 - <https://youtu.be/0qytRDvQkA0>   
Task 3 - <https://youtu.be/G8OPEAd_CAo>

### References

|  |  |
| --- | --- |
| [1] | N. a. R. D. Arshi Prabhakar, “Different Color Detection in an RGB Image,” 2017. [Online]. Available: https://www.journalijdr.com/sites/default/files/issue-pdf/9748.pdf. |
| [2] | “Image Moments,” 05 03 2023. [Online]. Available: https://theailearner.com/2020/10/16/image-moments/ . |
| [3] | Z. Gao, T.-R. Chiang, C.-F. Lin, Y.-C. Tsai and C. Fuh, “REALTIME PEDESTRIAN DETECTION SYSTEM USING COMBINATIONS OF MULTIPLE FEATURES AND OBJECT TRACKING METHOD,” in *IPPR Conference on Computer Vision, Graphics, and Image Processing*, 2014. |
| [4] | OpenCV, *Code Documentation,* 2015. |