AINT308 - Machine Vision and Behavioural Computing Coursework 1 Report

Student No. 10613591
School of Engineering,
Computing and Mathematics
University of Plymouth
Plymouth, Devon

***Abstract*—Machine Vision is field of study who's applications are becoming rapidly more prevalent amongst contemporary technology, with advancements having large implications in a wide variety of fields. This report details the use of a popular open-source machine vision library, OpenCV, in three different applications: Identifying the most common colour in an image, tracking a moving object within a video, and matching sub-components of an image to a larger image.**

Keywords:

Machine Vision, OpenCV, Object Tracking, C++

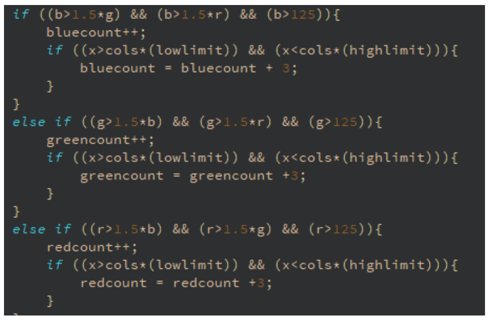
I. Task 1 - Colour Sorter

A. Introduction

The first task involves creating a system able to label a dataset of car colours red, green, or blue.

B. Solution

As part of its suite of features, **OpenCV** includes the ability to retrieve individual pixel's values. The specific values available will vary based on the colour space used. For example, working in the *RGB* colour space enables a pixel's *Red,Green* and *Blue* values to be retrieved. Using this information, it is possible to calculate the modal colour of an image.



```
if ((b>1.5*g) && (b>1.5*r) && (b>125)){
    bluecount++;
    if ((x>cols*(lowlimit)) && (x<cols*(highlimit))){
        bluecount = bluecount + 3;
    }
}
else if ((g>1.5*b) && (g>1.5*r) && (g>125)){
    greencount++;
    if ((x>cols*(lowlimit)) && (x<cols*(highlimit))){
        greencount = greencount +3;
    }
}
else if ((r>1.5*b) && (r>1.5*g) && (r>125)){
    redcount++;
    if ((x>cols*(lowlimit)) && (x<cols*(highlimit))){
        redcount = redcount +3;
    }
```

Fig. 1: Pixel testing code

The code in figure 1 performs the checks needed to assertain a pixel's primary colour. Initial attempts simply checked which of three component values were highest. In practice, this method was inaccurate as it had no way between differentiating between a near-white cell (such as a background cloud) which would have near equal *RGB* values, and one that belonged to the subject (in this case, the car).

To allow the algorithm to differentiate between background elements and the subject, two methods were employed. Firstly, a pixel was only counted if that colour's value was a significant margin higher than the other values. In figure 1 this margin is 50%. Additionally, the pixel's value must be greater than half the colour space's maximum value, in this case 125. This prevents background objects from being counted.

Additionally, extra weighting is applied to pixels within a central rectangle on the screen. This area can be changed by editing the values $highlimit$ and $lowlimit$. By adding this additional weighting to central areas, background areas are further discounted, helping to prevent false positives by valuing true positives more.

With the provided testing dataset of thirty red, blue, or green cars this solution is 100% accurate, correctly identifying the subject's colour in every image. This is a marked improvement from the simple method used in the initial attempt, which would often get confused at reflections in windows (these would be reflecting the sky, which would add bias towards blue), background walls (especially brick, which would bias the results towards red), and any natural backgrounds (adding green bias).

To further demonstrate the improvements performance, an additional dataset was sourced of ninety images, thirty of each colour. The results of this testing are visible in table I.

TABLE I: Expanded dataset testing results

Colour	Result
Red	30/30
Green	28/30
Blue	30/30

C. Limitations

This solution is extremely limited in its current format. The solution has no form of object recognition, meaning that it cannot differentiate between the car that it is trying to find the colour of, and any other objects inside the image. The middle-area weighting attempts to control for this, however it assumes that the subject of the image is centre-frame, which is something that cannot be said for all images of cars. This solution also has no way to deal with multiple cars in one image, especially ones of different colours.

Additionally, using *RGB* values is extremely limiting for colour recognition. The algorithm could not be expanded to detect additional colours (such as secondary colours made of a combination of red, green, and blue) without introducing a large amount of error. A major disadvantage of *RGB* values is that they can vary greatly based on the lighting conditions present in the image. Other colour spaces isolate the brightness (or luma) value, allowing for changes in lighting to be compensated for. A further problem with *RGB*'s representation of colour is that the hue is described as a combination of all three values, making it hard to isolate specific colours, as it is extremely rare for any physical colour to exist in purely one colour channel.

D. Further Improvements

A major improvement that could be made is switching colour space to *HSV* instead of *RGB*. *HSV* has numerous benefits for colour recognition when compared to *RGB*. *HSV* uses three values to describe a colour: *hue*, *saturation*, and *value*. Light changes will only affect two of these values; the *hue* value remains constant[1]. This is perfect for colour recognition applications as it means changes in lighting, as well as natural minor differences in colours, can be compensated for with much tighter ranges than in any other colour space.

Additionally, changing to *HSV* would allow for more colours to be recognised, as only the colour's *hue* value would need to be programmed in for the algorithm to recognise it. This would allow for theoretically any colour to be recognised, so long as its value is programmed in. In comparison to *RGB*, where as each colour is a mixture of all three channels, this is much simpler to implement.

E. Conclusion

This solution is adequate when the dataset is tightly controlled and consists only of red, green, or blue cars, in centre frame, with colour-neutral lighting. Due to the limitations of both the approach and colour space, anything more than marginal performance increases would require significant changes to the methodology. Changing colour space to *HSV* would provide a large increase to accuracy, as well as allow for the solution to be expanded to cover more colours.

II. Task 2 - Object Tracking

The second task involves tracking a pendulum, and using the observed displacement to calculate the pendulum's angular deviation from it's vertical rest position.

A. Introduction

In order for the angle to be calculated, the pendulum's motion must be tracked. Fortunately, the pendulum arm has a green target affixed to the end. By isolating this colour from the background, and calculating the centre of mass using OpenCV's *moments* functionality, the angle can be calculated using trigonometry.

B. Solution

The first stage of the solution is to find all the pixels that match the colour of the target on the pendulum arm. To do this, the colourspace is first converted to *HSV,* ***H**ue,* ***S**aturation,* ***V**alue.* As discussed in section I-D, *HSV* has numerous benefits over *RGB* for colour recognition.

C. Limitations

D. Further Improvements

E. Conclusion

III. Task 3 - Image Mapping

A. Introduction

B. Solution

C. Limitations

D. Further Improvements

E. Conclusion

Appendix

References

[1] Nathan Glover. *HSV vs RGB*. 2016. URL: https://handmap.github.io/hsv-vs-rgb/.