KINGSTON UNIVERSITY
Faculty of Science, Engineering and Computing

COLOUR RECOGNITION TOOL

Arturas Bulavko

March 2017

Table of Contents

1 Introduction & Scope

The idea behind this project was to create a colour recognition tool. The application will allow the user to take an image using the Raspberry Pi camera module, pick a pixel within the image and the Sense HAT will output the colour which was selected. Furthermore, the colour of the scrolling text on the Sense HAT matrix will also match the selected colour on the image.

```
%Clearing the workspace and define raspi
clear
mypi=raspi
mysensehat = sensehat(mypi);
%Setting up Raspi camera with required resolution
cam = cameraboard(mypi, 'Resolution', '1024x768');
%Looping for 20 ticks to ensure the camera adjusts the brightness correctly
for i = 1:20
    A= snapshot(cam);
    A_array(:,:,:,i)=A;
end
%Getting the last image of the loop
Image=A_array(:,:,:,20);
%Displaying the image and allowing the user to select a coordinate
imshow(Image)
[x,y] = ginput(1)
%Converting the X and Y coordinates into integers
x = floor(x);
y = floor(y);
%Converting RGB image into HSV to simplify the process
ImageHSV = rgb2hsv(Image);
%Getting the Hue, converting it into Degrees and rounding to integer
Hue = ImageHSV(y, x, 1)
HueDegrees = Hue*360;
HueDegreesRounded = floor(HueDegrees)
%Getting Saturation
Saturation = ImageHSV(y, x, 2)
%Getting Value
Value = ImageHSV(y, x, 3)
%Starting IF statements to program each color and change the scrolling text
%colour on the Sense HAT
%White Colour
if (Saturation < 0.4)
    displayMessage(mysensehat,'Colour: White', 'TextColor', [255,255,255]);
else
    %Black Colour
    if (Value < 0.15)
        displayMessage(mysensehat,'Colour: Black', 'TextColor', [0,255,255]);
    else
        %Green Colour
        if (HueDegreesRounded < 170) && (HueDegreesRounded > 70)
            displayMessage(mysensehat,'Colour: Green', 'TextColor', [0,255,0]);
        %Yellow Colour
        elseif (HueDegreesRounded < 69) && (HueDegreesRounded > 30)
            displayMessage(mysensehat,'Colour: Yellow', 'TextColor', [255,255,0]);
        %Red Colour
        elseif (HueDegreesRounded < 360) && (HueDegreesRounded > 270)
            displayMessage(mysensehat,'Colour: Red', 'TextColor', [255,0,0]);
        elseif (HueDegreesRounded < 29) && (HueDegreesRounded > 0)
            displayMessage(mysensehat,'Colour: Red', 'TextColor', [255,0,0]);
        %Blue Colour
        elseif (HueDegreesRounded < 269) && (HueDegreesRounded > 171)
            displayMessage(mysensehat,'Colour: Blue', 'TextColor', [0,0,255]);
        %Print error message if the colour was not recognised
        else
            displayMessage(mysensehat,'Cound not define!', 'TextColor', [102,0,204]);
        end
    end
end
```

Figure 1 – System Code

Figure 1 shows the code which was developed. The code begins by clearing the workspace and removing the items from the system memory followed by a connection to the Raspberry Pi board. It is then necessary to connect to the Sense HAT board through a provided function which will allow to gather the necessary readings. It is also essential to connect to the Raspberry Pi's camera module by supplying the three parameters. Those parameters are; the earlier defined *mypi* Raspberry Pi device, argument name and the resolution at which the images will be taken. In order to take an image, it was necessary to record a video for 20 ticks, this is because the Raspberry Pi needs to configure the brightness levels and if the image is taken instantly, it is rather dark and the application loses all interest. Consequently, a 'for loop' is required which was programmed to last 20 ticks to ensure the camera is automatically configured to capture the best quality image. During the loop, an image is taken every tick and is stored in a 4-D array, normally the image is stored in a 3-D array, but as it is necessary to obtain the last image, another dimension has been added which recorded ticks. Once the loop has ended, a new variable is created for the required image. This is done by targeting specific frame of an array created earlier.

Having obtained the image which will be processed, it is displayed to the user through graphical user input interface (*ginput*) which allows the user to select an exact pixel for processing. The *ginput* return the x and y coordinates of the position which was clicked, the (1) tells the software that only one input/click is needed. This number can be increased to obtain multitude of coordinates. The x and y coordinates are then converted into an integer through the use of *floor()* function and are stored in appropriate variables. The image is converted from Red Green Blue (RGB) to Hue Saturation Value (HSV) format through the use of *rgb2hsv()* MATLAB function. This simplifies the process of colour recognition as primarily only the Hue will have to be analysed to determine the colour as oppose to the three values of RGB. *Figure 2* demonstrates the Hue values in degrees which clearly outlines that majority of the colours can be estimated using this. The only complication with this process is determining the white and black colours, however this will be discussed in more detail further down the report. The Saturation figure tells the intensity of a specific colour, whereas the Value illustrates how dark or light the image is.

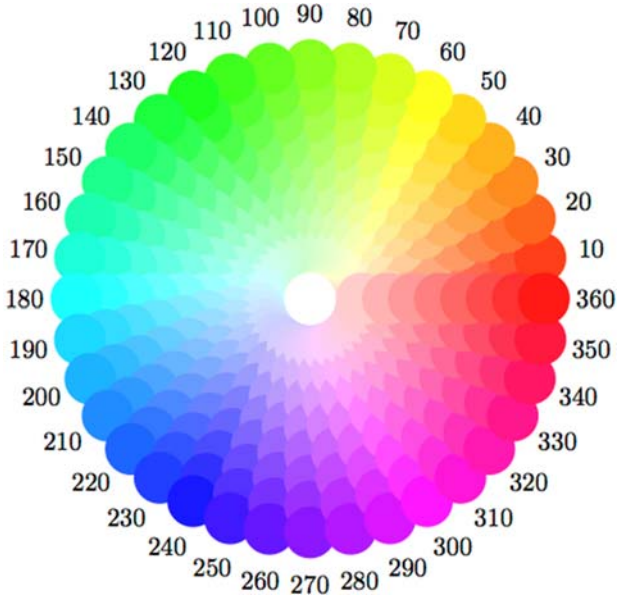


Figure 2 – Hue Colour Wheel (Thurston, 2014)

Once the image has been converted into HSV format, it is necessary to gather each value separately. To be able to do that, three variables are created, namely; Hue, Saturation and Value which will store each respective value. In order to put the right value into each variable, the x and y coordinated gathered earlier are used on the HSV image which obtains the necessary HSV value at a specific coordinate. The HSV format stores the values from 0 to 1 and therefore, to make it easier, the Hue value was converted into degrees by multiplying the figure by 360. The Saturation and value were left in the range of 0 to 1 as they are not particularly required at this stage.

Having obtained the necessary data, it was required to develop IF statements which would determine the colour of the pixel using the data provided within the statement. The IF statement begins by identifying White colour through the Saturation value, it is necessary to use Saturation because the intensity of each colour must be as close to 0 as possible for any colour to become white, as such the Hue could not be used. If the Saturation was above the 0.4 it meant that the image could either be black or colourful, this is known because the Saturation of a black image tend to go towards 1. However, to determine black the Value was used which ensured that all dark colours below 0.15 threshold result in the output of black colour. Similarly, if the Value is above 0.15 and Saturation is above 0.4 the image can be classified as colourful. At this stage, the Hue figures begin determining the colour of the supplied pixel. Using the *Figure 2*, each colour was programmed using Hue degrees and the necessary text was added which will be shown on the Sense HAT LED matrix. In order to show text on the LED matrix, a *displayMessage()* function was used with the necessary parameters. The four supplied parameters told where to display the text, what text to display and the remaining two established the colour of the text. The text colour was supplied in the RGB format, as required by the function.

The program partitioned the Hue colour wheel into 4 quadrants because due to scope of the project, it was only necessary to identify White Black, Blue, Red, Yellow and Green colours. To make the program reliable, an error checking feature was added which ensured that if the colour was not recognised, the user would be made aware of that, instead of the application crashing. Furthermore, due to technical limitation, the colour black was not allowed to be used as scrolling text to represent colour black. The black colour results in the text being unseen because LED matrix simply does not light up, to overcome this issue it was decided to use cyan text to represent black.

2 Testing & Results

The *Table 1* demonstrates the testing process of the application. A card with four required colours was printed and put in front of the Raspberry Pi camera. The black image was achieved by putting a finger onto the camera to prevent any light source from accessing the lens. To get pure white colour, a white image on the phone was found and directed towards the camera. As demonstrated in each picture, the *ginput* interface was used to select the appropriate colour of the pixel.

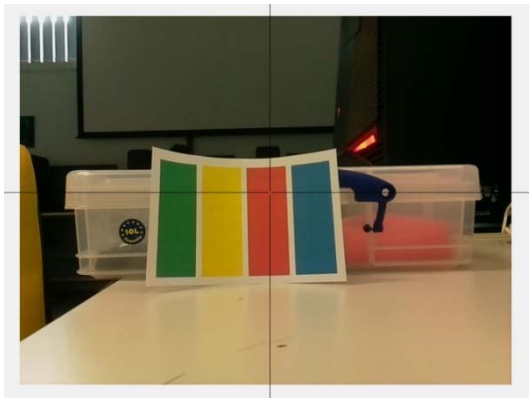
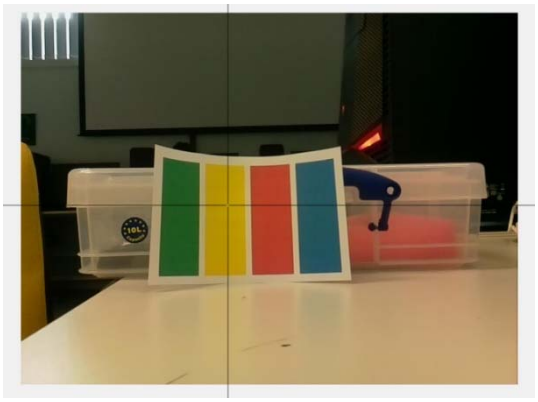
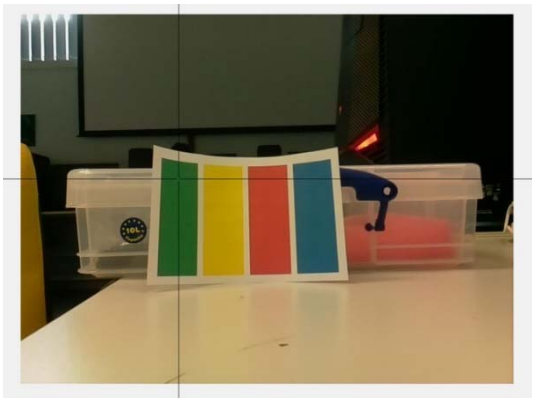
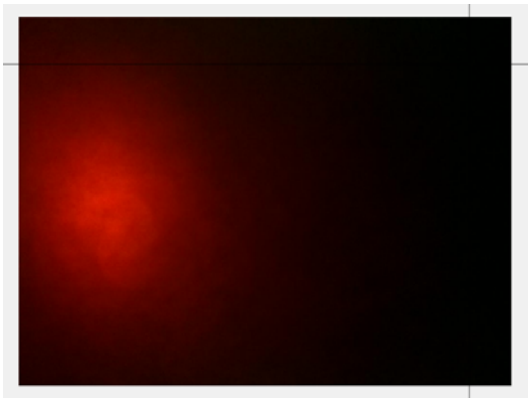
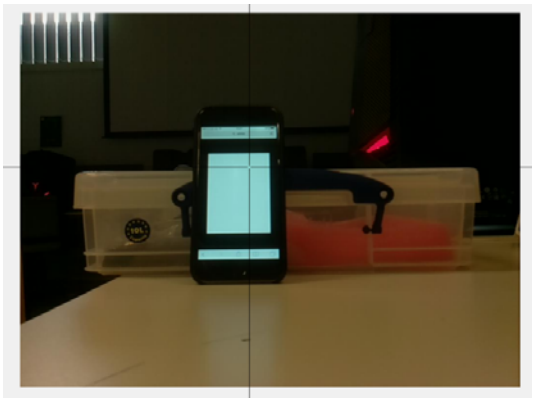
	
Selecting Blue colour	Selecting Red colour
	
Selecting Yellow colour	Selecting Green colour
	
Selecting Black colour	Selecting White colour

Table 1 – Capturing the image

Considering that the room had poor lighting, the images were of a decent quality. Once the exact pixel coordinate was selected, the application would process the algorithm and output the value of

the colour onto the Sense HAT LED matrix, shown in *Table 2*. During this test, it was important to ensure that the selected colour resulted in the correct colour of the text and the text itself.

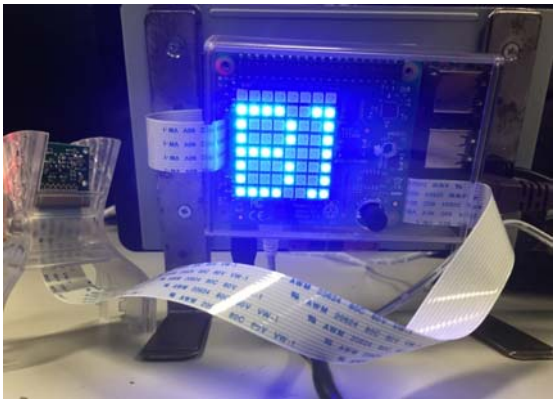
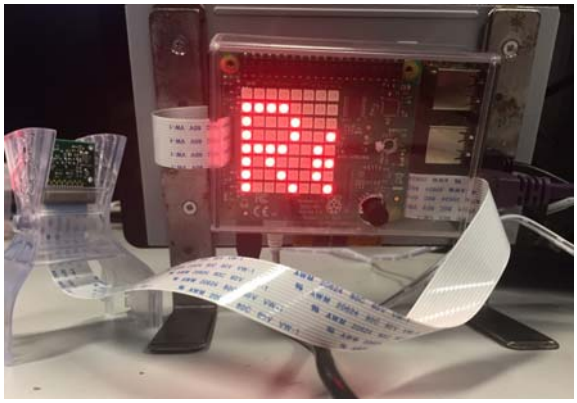
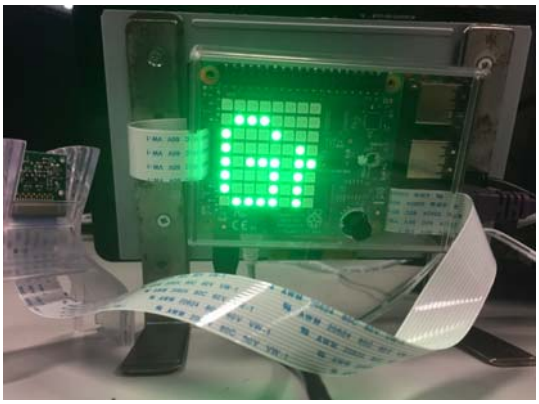
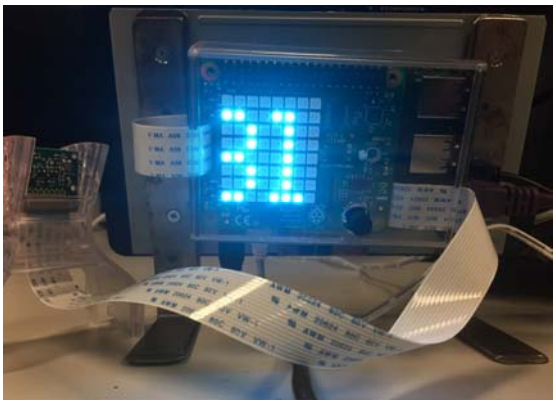
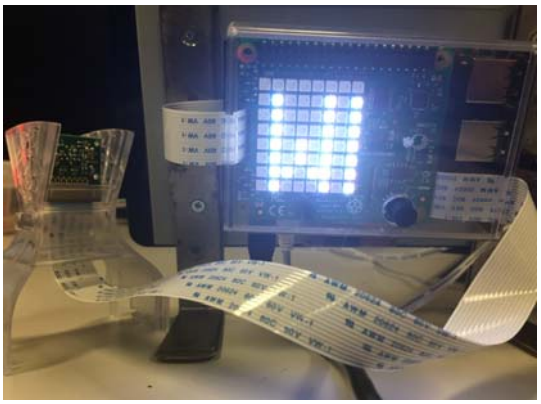
	
Blue colour output	Red colour output
	
Yellow colour output	Green colour output
	
Black colour output	White colour output

Table 2 – Displaying the output

Based on the input from *Table 1* it is clear that the output shown in *Table 2* fully met the expectations. Each colour supplied by the image was correctly mapped to the correct matrix output which constitutes this application a success. The testing has not revealed any bugs and by testing the application several times it was deducted that the colour determining algorithm works flawlessly. In order to test this, the developer simply took a picture, selected the pixel of the image and observed whether the application has correctly interpreted it.

3 Conclusion

Having carried out detailed testing, it is reasonable to say that the application aims have been fully achieved. The application is able to recognise six colours and display the colour name on the Sense HAT matrix. Nevertheless, the RGB image could be used as an alternative to the HSV which would achieve the same outcome, however, the code would have to be more complicated as each IF statement would have to compare three values instead of one. By using the HSV approach, the workload on the machine has been greatly reduced and thus, the application works efficiently. The RGB approach would also allow to find white colour better as all values would have to be 0.

To improve the current application, it is possible to get the RGB and HSV values of an image, and supply the RGB values to the *TextColor* in order to match the colour on the image with the colour on the Sense HAT LED matrix. It is also possible to move this application further by introducing more colours, however, current camera and lighting in the rooms would be a limitation because for the best colour recognition, the room must be well lit and the image must be of high quality.

References

Thruston (2014) Available at: http://tex.stackexchange.com/questions/136570/metapost-adjust-colour-value-brightness-luminosity. (Accessed: 9 March 2017).