IP - Mini-project:
Topic 6: Convert from RGB to HSI

Jannick Drews

November 9, 2018

1 Convert from RGB to HSI

The `RGB` and `HSI` refers to their respective color-spaces. These are used to display an image in a certain aspect, both color-spaces presented in a digital form will use a 3-dimensional array.
For `RGB`, the 3-dimensional array would look like the following; $[R, G, B]$, an example could be $[255, 10, 25]$ which is the value 255 for the red channel, 10 for the green channel and 25 for the blue channel. Note that RGB values follow the ruleset[2];

$$R, G, B \in [0, 255]$$

For the `HSI` color-space, represented digitaly it would look like; $[H, S, I]$, an example could be $[310, 0.3, 144]$ which is the value 310 for the hue in degrees, 0.3 for the saturation and 144 for the intensity. Note that HSI values follow the ruleset[2]:

$$H \in [0, 360[$$

$$S \in [0, 1]$$

$$I \in [0, 255]$$

Keep in mind that this color-space does have a inconsistent naming convention, it is also known as `HSB` and `HSL`, if not more.

This assignment will convert the RGB color-space to the HSI color-space, this is performed by utilizing the formulas found on Figure 1 [2]

$$H = \begin{cases} \cos^{-1}\left(1/2 \cdot \frac{(R-G)+(R-B)}{\sqrt{(R-G)(R-G)+(R-B)(G-B)}}\right), & \text{if } G \geq B; \\ 360^\circ - \cos^{-1}\left(1/2 \cdot \frac{(R-G)+(R-B)}{\sqrt{(R-G)(R-G)+(R-B)(G-B)}}\right), & \text{Otherwise} \end{cases} \tag{3.8}$$

$$H \in [0, 360[$$

$$S = 1 - 3 \cdot \frac{\min\{R, G, B\}}{R + G + B} \quad S \in [0, 1] \tag{3.9}$$

$$I = \frac{R + G + B}{3} \quad I \in [0, 255] \tag{3.10}$$

Figure 1: Formulas for converting RGB to HSI[2]

For converting the degrees to radians and vice versa;

$$D_{deg} = R_{rad} \cdot \frac{180}{\pi}, \; R_{rad} = (\frac{D_{deg}}{180}) \cdot \pi$$

Figure 2: Radians $\rightarrow$ Degrees and Degrees $\rightarrow$ Radians

And the description of the assignment is the following:
"Make a Python program that can convert from RGB to HSI.
Input: An RGB image
Output: Three images: a H-image, a S-image, and an I-image"

2 Implementation

2.1 Code & Description

Following is my implementation of the channel conversion from RGB to HSI:

```python
# Topic #6: Convert from RGB to HSI #
# Author: Jannick Drews             #
import cv2, numpy as np, matplotlib.pyplot as plt

def HSI_Conv(im):
    # Algorithm:
    original = im
    hsi_h = np.zeros((im.shape[0], im.shape[1]), np.float32)
    hsi_s = np.zeros((im.shape[0], im.shape[1]), np.float32)
    hsi_i = np.zeros((im.shape[0], im.shape[1]), np.float32)
    for x in range(im.shape[0]):
        for y in range(im.shape[1]):

            # Get RGB(Remember to use BGR_TO_RGB function further down for
                correct vals)
            red = int(im[x][y][0])
            green = int(im[x][y][1])
            blue = int(im[x][y][2])

            # Get HSI (Hue first)
            top = (red - green) + (red - blue)
            bot = 2 * np.sqrt((red - green)*(red - green) + (red - blue)*(
                green - blue))
            if (bot == 0): # Exception if R = G = B
                saturation = 0.0
                hue = 0.0
                intensity = 255.0
                hsi_h[x][y] = hue
                hsi_s[x][y] = saturation
                hsi_i[x][y] = intensity
                continue
            formula = np.arccos( np.divide(top,bot) )
            #print ("rgb: {},{},{}\nFormula return:{}".format(red, green,
                blue, formula))

            hue = 0.0
            if (green >= blue):
                hue = formula
            else:
                hue = 2*np.pi - formula
            #print("G>=B: {}".format(hue))
            hue = (hue * 180)/np.pi # rads to degrees
            #print("Degree conversion: {}\n".format(hue))

            # Saturation
            sum_rgb = red + green + blue
            division = np.divide(min(red,green,blue), float(sum_rgb))
            mult = 3.0 * division
            saturation = 1.0 - mult

            # Intensity
            intensity = np.divide(sum_rgb, 3)

```

```
51                # Output respective channels
52                hsi_h[x][y] = hue
53                hsi_s[x][y] = saturation
54                hsi_i[x][y] = intensity
55
56        # Show plot:
57        plt.subplot(141), plt.imshow(original), plt.title('Original')
58        plt.xticks([]), plt.yticks([])
59        plt.subplot(142), plt.imshow(hsi_h), plt.title('Hue')
60        plt.xticks([]), plt.yticks([])
61        plt.subplot(143), plt.imshow(hsi_s), plt.title('Saturation')
62        plt.xticks([]), plt.yticks([])
63        plt.subplot(144), plt.imshow(hsi_i), plt.title('Intensity')
64        plt.xticks([]), plt.yticks([])
65        plt.show()
66        return hsi_h, hsi_s, hsi_i
67
68    # Fast implementation since the real assignment is RGB -> HSI
69    def BGR_TO_RGB(im):
70        tmp = np.zeros((im.shape[0], im.shape[1]))
71        tmp[:,:] = im[:,:,2]
72        im[:,:,2] = im[:,:,0]
73        im[:,:,0] = tmp[:,:]
74        return im
75
76    img = cv2.imread("baboon.jpg", 3)
77    img = BGR_TO_RGB(img)
78    HSI_Conv(img)
```

Lines; 7 to 10 A copy of the image is made, to show later for comparison. 3 different channels(matrices) are made for h,s,v respectively, hence the naming convention. These 3 new channels are of the same shape of the original image, and has a depth value of a float32 since float numbers are going to be present in the Saturation channel.

Lines; 11 to 17 Inside a double for-loop, to go through each pixel in the image, we grab the respective colors from the red, green and blue channel of the input image, and store them as integers to be sure the calculations will be done properly.

Lines; 20 to 39 The different variables are assigned to the respective parts of the formula from Figure 1, and makes sure to catch the exception where $R = G = B$. And lastly the `formula` variable calculates the remaining part of the formula. The `top` variable calculates the top part of the formula, and the `bot` calculates the bottom part.
We then check if the statement from the formula is true, that if $G \geq B$ then we subtract 360 degrees, and remember to convert it first to $2 \cdot \pi$ for the radians. And the last instruction converts the radian back to degrees.

Lines; 43 to 54 From here we calculate the Saturation and the Intensity channels, still following the formula described at Figure 1. First the saturation, which is dividing the minimum value of the respective red green or blue channel, this formula is also split it for ease of reading.
Lastly the intensity channel is calculated by finding the mean of the R,G,B pixels. And lastly put into the respective HSI channels. Lines; 57 to 66 These lines just serve to subplot the images and display the different resulting HSI channels in comparison to the original RGB image.

Lines; 69 to 74 This function just convers the BGR to RGB image, since OpenCV loads as BGR first, I had to do a convert to RGB previously to inputting the image into the main function to convert to HSI. This function just creates a new temporary channel to store the red, and then it switches the blue and red channel with eachother to get RGB.
Lines; 76 to 78 These three lines just load the image and convert it to RGB and put it into the HSV conversion function.

2.2 Input/Output

The subplot illustrated on Figure 3, is displayed after the code has executed, where the *"Original"* is the original RGB image, of cause converted to RGB with the conversion function implemented at line `69`, and the rest of the titles are the Hue, Saturation and Intensity channels respectively.

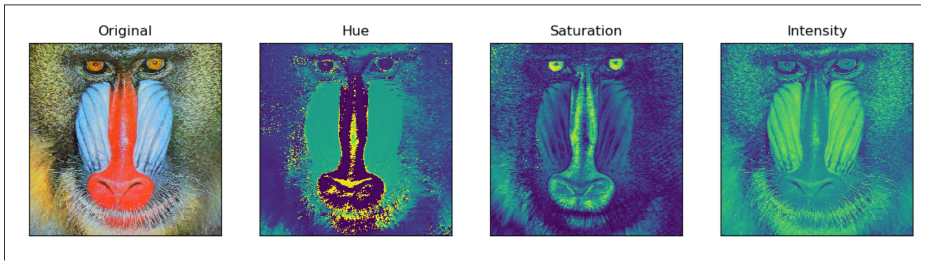
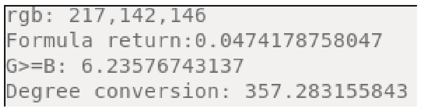


Figure 3: Subplot result from code

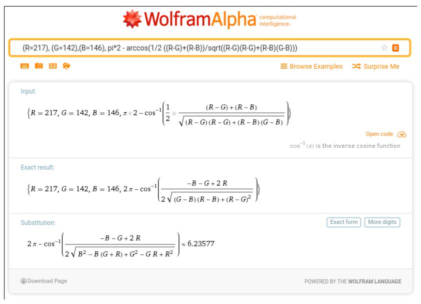
3 Points of note

As illustrated on Figure 3, the *"Hue"* channel may look weird, theoretically this could be a display error from matplotlib, since it potentially does not know how to handle the values from; $]255, 360[$. The reason for writing this section is to portray that the resulting calculations from the implementation are indeed correct.
The next figures will illistrate calculations from Wolfra Alpha, which is a huge computational system that has features such as; solving math equations. I will compare the results from Wolfram Alpha with my implementations results.



```
rgb: 217,142,146
Formula return:0.0474178758047
G>=B: 6.23576743137
Degree conversion: 357.283155843
```

(a) Result from implementation



(R=217), (G=142),(B=146), pi*2 - arccos(1/2 ((R-G)+(R-B))/sqrt((R-G)(R-G)+(R-B)(G-B)))

Input:

$$\left\{R = 217,\ G = 142,\ B = 146,\ \pi \times 2 - \cos^{-1}\left(\frac{1}{2} \times \frac{(R-G)+(R-B)}{\sqrt{(R-G)(R-G)+(R-B)(G-B)}}\right)\right\}$$

$\cos^{-1}(x)$ is the inverse cosine function

Exact result:

$$\left\{R = 217,\ G = 142,\ B = 146,\ 2\pi - \cos^{-1}\left(\frac{-B-G+2R}{2\sqrt{(G-B)(R-B)+(R-G)^2}}\right)\right\}$$

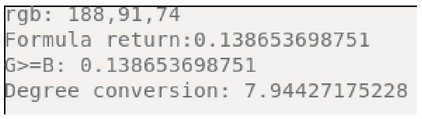
Substitution:

$$2\pi - \cos^{-1}\left(\frac{-B-G+2R}{2\sqrt{B^2 - B(G+R) + G^2 - GR + R^2}}\right) \approx 6.23577$$

WolframAlpha computational intelligence.
Browse Examples
Surprise Me
Open code
Exact form
More digits
Download Page
POWERED BY THE WOLFRAM LANGUAGE

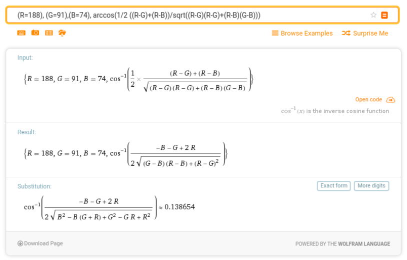
(b) Result from Wolfram alpha[1]

Figure 4: Comparison of implementation result and wolfram alphas result



```
rgb: 188,91,74
Formula return:0.138653698751
G>=B: 0.138653698751
Degree conversion: 7.94427175228
```

(a) Result from implementation



(R=188), (G=91),(B=74), arccos(1/2 ((R-G)+(R-B))/sqrt((R-G)(R-G)+(R-B)(G-B)))

Input:

$$\left\{R = 188,\ G = 91,\ B = 74,\ \cos^{-1}\left(\frac{1}{2} \times \frac{(R-G)+(R-B)}{\sqrt{(R-G)(R-G)+(R-B)(G-B)}}\right)\right\}$$

$\cos^{-1}(x)$ is the inverse cosine function

Result:

$$\left\{R = 188,\ G = 91,\ B = 74,\ \cos^{-1}\left(\frac{-B-G+2R}{2\sqrt{(G-B)(R-B)+(R-G)^2}}\right)\right\}$$

Substitution:

$$\cos^{-1}\left(\frac{-B-G+2R}{2\sqrt{B^2 - B(G+R) + G^2 - GR + R^2}}\right) \approx 0.138654$$

(b) Result from Wolfram alpha[1]

Figure 5: Comparison of implementation result and wolfram alphas[1] result

Figure 4 and Figure 5, shows the results from my implementations calculations and the calculations done by Wolfram alpha[1] for the Hue value, using the Hue formula from Figure 1, the difference of the two figures being the boolean $G \geq B$. Both get the same result for the Hue value, which stands to show that the calculations for the pixel values in the Hue channel are correct, but also stands to show that the display may be incorrect. As mentioned before, this could be a result of matplotlib displaying the channel incorrectly(Which seems improbable) or a bug.

4 Bibliography

References

[1] Wolfram Alpha LLC. “WolframAlpha: Computational Intelligence”. In: (2018). URL: `http://www.wolframalpha.com/`.

[2] Thomas B. Moeslund. *Introduction to Video and Image Processing.* Springer London Dordrecht Heidelberg New York, 1995. ISBN: 978-1-4471-2502-0.