CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Problem Set 01

Lucas Santos Zatta

oriented by

Flávio Luis Cardeal Pádua

Belo Horizonte

2021

Introduction

The goal of this work is to try to apply concepts seen in class as well as formulas in order to perform image analyzes. To solve this problem set, the codes implemented integrate functions created by the author with library functions(mostly, due to performance and time reasons).

For this first problem set we will be using Python3 with the libraries Numpy, OpenCV and Matplotlib. It's important to make sure that these libs are properly installed and fully working so that we can achieve the desired end results.

Questions

1. Basic Acquaintance with Programmed Imaging

In all of the following scripts, we will be using Numpy alongside OpenCv:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

1.1. Loading an RGB image in a lossless data format

This first example is a very simple one. For the next problems the following image (Figure 1) is going to be used as an example in the PNG format:

Figure 1: Example image

As stated, this first example is quite simple. In order to load the image we run:

```
img = cv2.imread('landscape.png')
```

This line will load all image channels as a matrix of pixels. Each element is represented as a vector that contains information of the RGB channels.

And to show image on screen:

```
cv.imshow("Display window", img)
cv.waitKey(0)
```

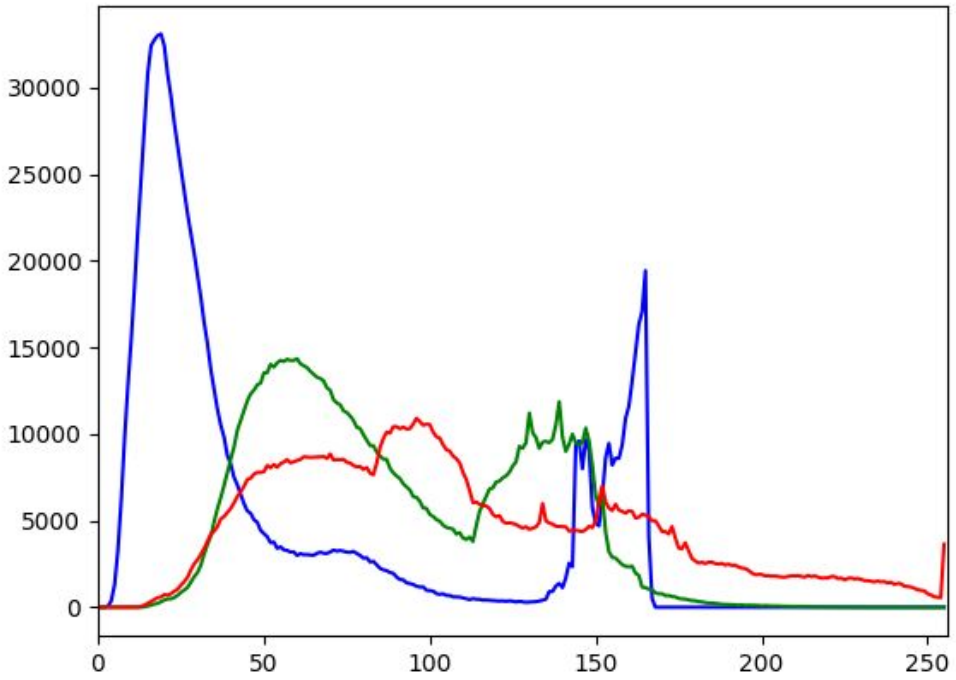
1.2. Displaying the histograms of all three color channels of Figure 1

Now that our image is loaded, we have the means to calculate its histogram. To do it, we will use OpenCV and Matplotlib:

```
color = ('b','g','r')
plt.figure()
#iterate through each channel
for i,col in enumerate(color):
    hist = cv.calcHist([img],[i],None,[256],[0,256])
    plt.plot(hist,color = col)
```

```
            plt.xlim([0,256])
        plt.show()
```

The result should be:



1.3. Getting some pixel information using the mouse and drawing an outer border

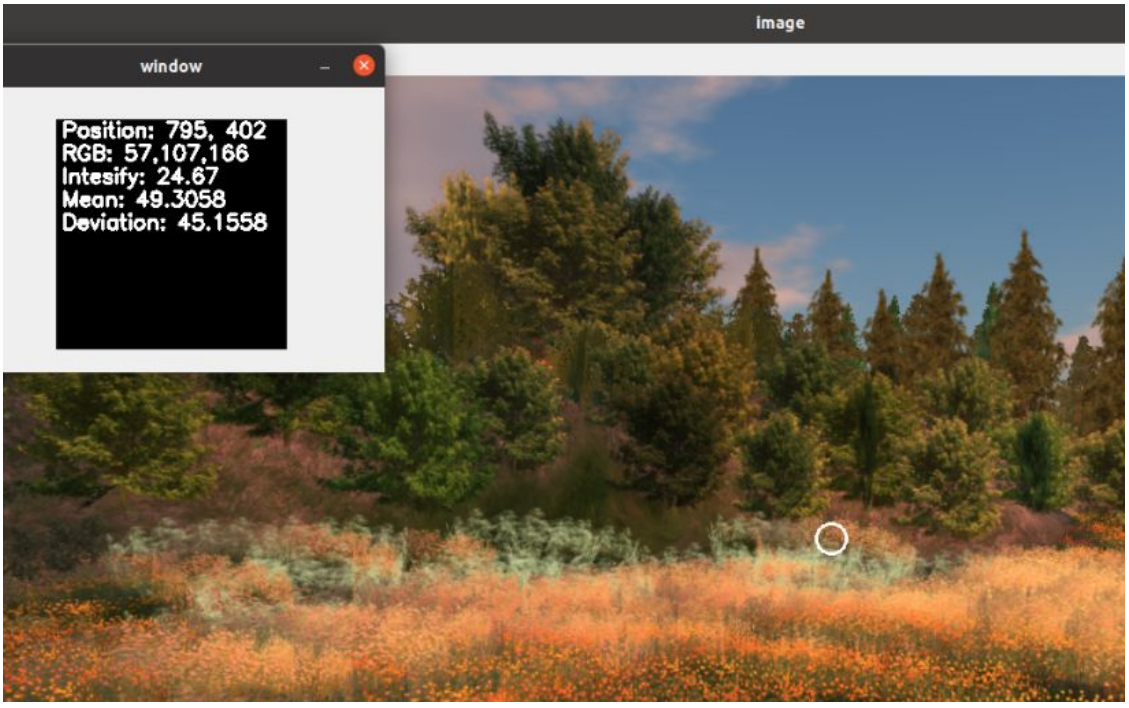
Now we need to get the mouse position (*p)* in our example FIgure 1, and draw an outer border to determine a window $W_p^{11,11}$ and return information about the pixel our mouse is currently on.

```
#Calculate mean to img parameter
def calculateMean(img):
    x,y,n = img.shape
    total = 0
    for a in range(0,y-1):
        for b in range(0,x-1):
            for k in range(0,n-1):
                total = total + img.item(b,a,k)
    calculatedMean = round(total/(x*y),4)
    return calculatedMean

#Calculate variance to img parameter
def calculateVariance(img,mean = -1):
    if mean == -1:
```

```
        mean = calculateMean(img)
    x,y,n = img.shape
    total = 0
    for a in range(0,y-1):
        for b in range(0,x-1):
            for k in range(0,n-1):
                total = total + pow((img.item(b,a,k)-mean),2)
    variance = round(total/(x*y),4)
    return variance

#Calculate deviation to img parameter
def calculateDeviation(img,variance = -1):
    if variance == -1:
        variance = calculateVariance(img)
    return round(math.sqrt(variance),4)
```



1.4 - Discuss

It can be observed that in images that have a very dominant color tone, the values are more homogeneous than when the image has a wide range of shades and colors overall. In our example, we can notice homogeneous behavior if we take pixels from atop of the trees and from the sky, and we can notice that homogeneity is not quite present when we analyze pixels that are between the trees and the grass for example

2. Data Measures of Image Sequences

First we read all the images and calculate mean, deviation and contrast for each one, after doing this, we create the functions that define this curve and calculate mean, deviation for this curves

The first step is to calculate mean, deviation and contrast for each of the images in the sequence. After doing that, we create functions that define curve and calculate the curves for mean, deviation and contrast.

```
#create functions
f2 = interp1d(times,deviations, kind='cubic')
g2 = interp1d(times,means, kind='cubic')
h2 = interp1d(times,contrasts, kind='cubic')
#get mean and deviation of functions
g_mean = statistics.mean(g2(times))
f_mean = statistics.mean(f2(times))
h_mean = statistics.mean(h2(times))
g_dev = statistics.stdev(g2(times))
f_dev = statistics.stdev(f2(times))
h_dev = statistics.stdev(h2(times))
```

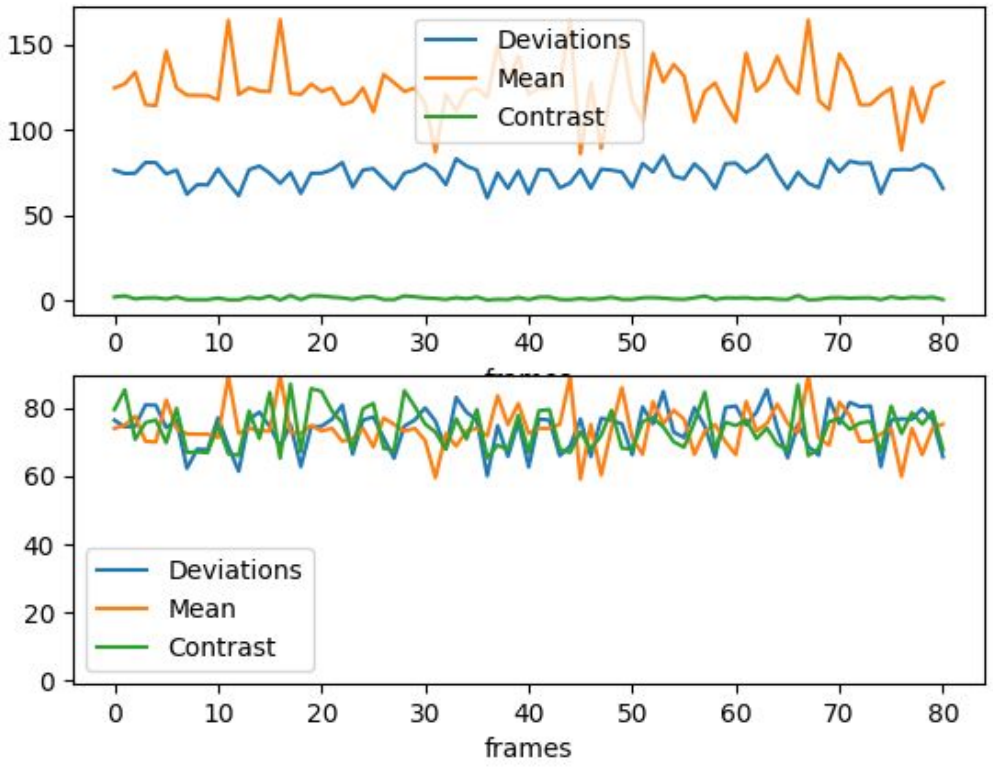
Now we calculate alpha and beta:

```
#calculate alpha and beta
alpha = (g_dev/f_dev)*f_mean-g_mean
beta = f_dev/g_dev
alpha2 = (h_dev/f_dev)*f_mean-h_mean
beta2 = f_dev/h_dev
```

Now, normalizinf the functions:

```
#normalize functions
for i in range(0,len(calcs)):
    g_new.append(beta*(y_new[i]+alpha))
    h_new.append(beta2*(a_new[i]+alpha2))
```

With the functions normalized we can plot the graphs and display the results



We can also calculate the distance between the functions

```
#calculate the distance between functions
for i in times:
    sum_d2 = pow((f[i]-g[i]),2)
    sum_new_d2 = pow((f[i]-g_new[i]),2)
    sum_new_h2 = pow((f[i]-h_new[i]),2)

d1_g=d*math.sqrt(sum_d2)
d1_g_new = d*math.sqrt(sum_new_d2)
d1_h_new = d*math.sqrt(sum_new_h2)
```

We can notice that after normalization, the distance between these functions was reduced to values very close to zero, which makes theses functions structurally similar.

3. Different Impacts of Amplitudes and Phase in Frequency Space

For this example, we need to use fourier transform, phase and amplitude calculation and inverse transform. Luckily we can use the existing functions for this operations from the OpenCV library.

```
#transform image to frequency domain
def transform(img):
    dft = cv.dft(np.float32(img),flags = cv.DFT_COMPLEX_OUTPUT)
    return dft

#get phases and amplitude
def phases_amplitude(matrix):
    dft_shift = np.fft.fftshift(matrix)
    amplitude,phases = cv.cartToPolar(dft_shift[:,:,0],dft_shift[:,:,1])
    return phases,amplitude


#transform image to spatial domain
def re_transform(phases,amplitude):
    x,y = cv.polarToCart(amplitude,phases)
    back = cv.merge([x, y])
    back_ishift = np.fft.ifftshift(back)
    img_back = cv.idft(back_ishift)
    img_back = cv.magnitude(img_back[:,:,0], img_back[:,:,1])
    return img_back
```

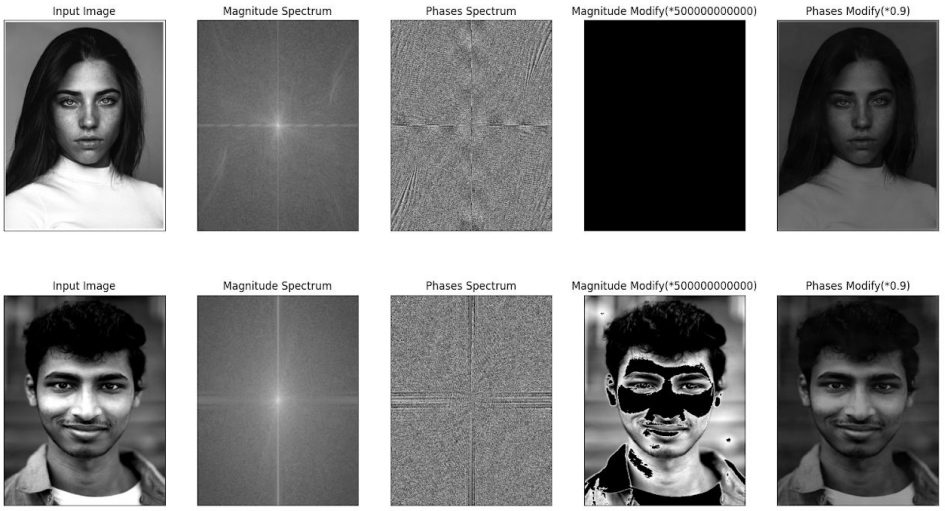
It is noticeable that one image interferes in the other when returning to the spatial domain and combining the phases of one with the amplitude of the other.

```
#transform images to back spatial domain
img_back = re_transform(phases_1,amplitudes_2)
new_img = cv.normalize(img_back, None, alpha=0, beta=255,
norm_type=cv.NORM_MINMAX, dtype=cv.CV_8U)
img_back_2 = re_transform(phases_2,amplitudes_1)
new_img_2 = cv.normalize(img_back_2, None, alpha=0, beta=255,
norm_type=cv.NORM_MINMAX, dtype=cv.CV_8U)
```

a. Overall, regardless of which image we send, the image phase contributes more to changes.

b. We can notice that a small change in amplitude doesn't interfere much with the image, but a small change in phase completely changes the final result of the image.

Even the smallest of changes in the phase of a given image will completely deform it. Changing the phase value will make the image get darker, but in order to make these changes visible, amplitude values must be high. With values of 500000000000 we can barely see the image.



4. Approximating the HSI Space by Planar Cuts

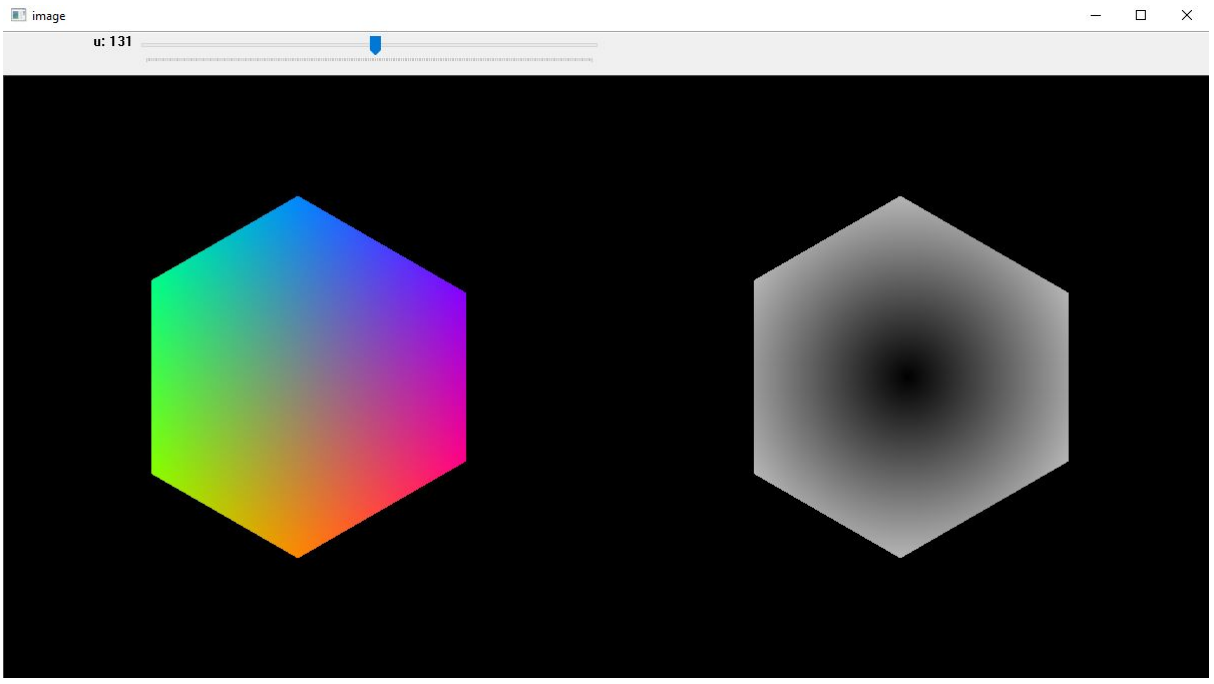
Now it's time to explore the HSI Space. HSI is a color model like RGB that represents colors using three very different components: Hue, Saturation and Intensity. To calculate RGB values based on u, we need to find values of x and y in the cartesian plane. After finding those values we need to find the nearest RGB value and add it to the point (x,y) to get the saturation level

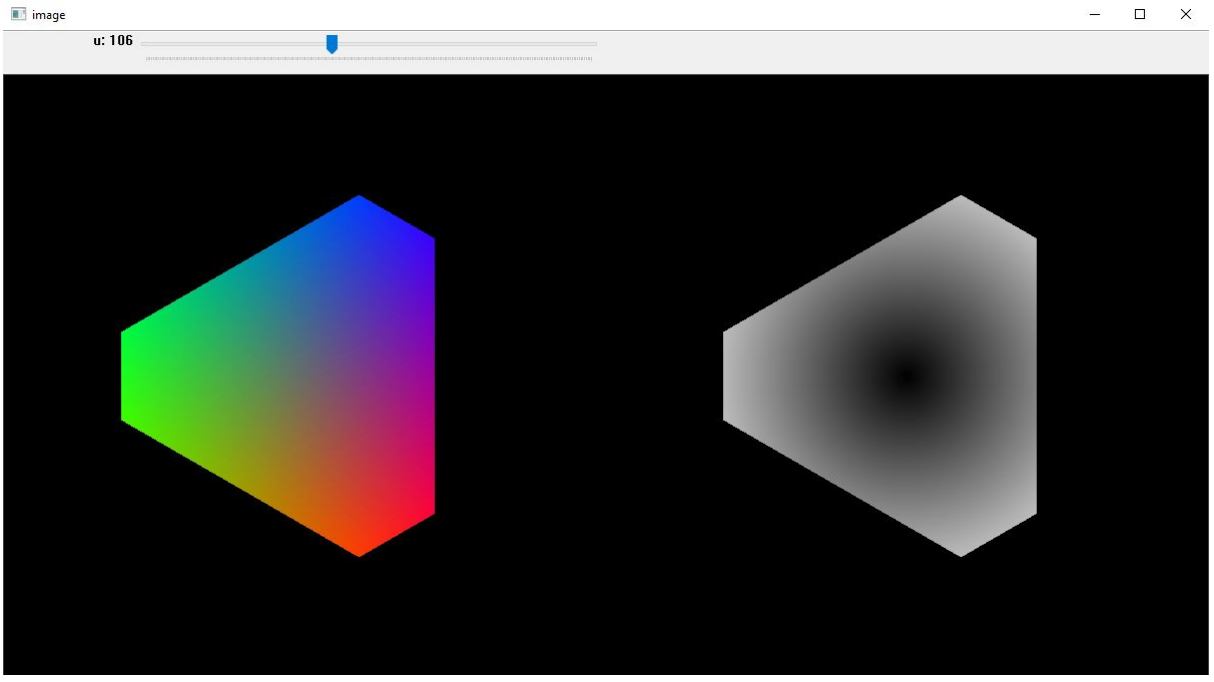
```
for h in range(0, H_MAX):
            for s in range(0,G_MAX):

                #calculate x and y
                x = int(s * (np.cos(np.deg2rad(h/10))))
                y = int(s * (np.sin(np.deg2rad(h/10))))

                #get nearby rgb values
                r = u + x/np.sqrt(2) + y/np.sqrt(6)
                g = u - 2*y/np.sqrt(6)
                b = u - x/np.sqrt(2) + y/np.sqrt(6)

                if r > 0 and g > 0 and b > 0:
                    if r < G_MAX and g < G_MAX and b < G_MAX:
                        img[x+C,y+C] = np.array([b,g,r], np.uint8)
                        img2[x+C,y+C] = np.array([s,s,s], np.uint8)
```





Conclusion

This work was very challenging and productive. It was a great way to put in practice what was exercised through the lectures of the subject. From a “developer” standpoint, it was really refreshing exploring new libraries and technologies as well as concepts not often present in what I’m used to work with.