

## 1 Input images

We consider 4 input images of different sizes.



(a) Input p0-1-0.jpg, 207x236 pixels



(b) Input p0-1-1.jpg, 183x275 pixels



(c) Input p0-1-2.jpg, 272x384 pixels



(d) Input p0-1-3.jpg, 390x502 pixels

Figure 1: Input images

## 2 Color planes

a) Swap the red and blue channels of the input image.



(a) Output for the input p0-1-0.jpg



(b) Output for the input p0-1-1.jpg



(c) Output for the input p0-1-2.jpg



(d) Output for the input p0-1-3.jpg

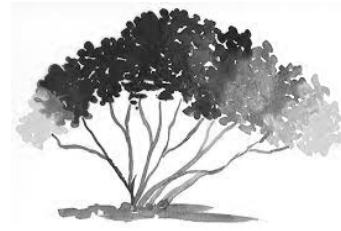
Figure 2: Output images for question 2.a

The `imread()` instruction of OpenCV reads images in format BGR, then we have to swap the channel blue(first channel) with the channel red(third channel), generating a new image in RGB format. We can see that the red colored regions become blue colored regions and the blue colored regions become red colored regions

- b) Create a monochrome image (`img-green`) by selecting the green channel of the input image.



(a) Output for the input p0-1-0.jpg



(b) Output for the input p0-1-1.jpg



(c) Output for the input p0-1-2.jpg



(d) Output for the input p0-1-3.jpg

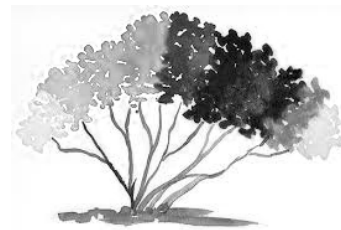
Figure 3: Output images for question 2.b

As we extract the green channel from the original image to generate a monochromatic image, then, the regions of the image that depend more on green color will have values close to 255 in the channel green, which means that in the output image those regions will have a color close to white, while the other regions of the image will have a color close to black.

- c) Create a monochrome image (`img-red`) by selecting the red channel of the first input image.



(a) Figura 13. Output for the input p0-1-0.jpg



(b) Figura 14. Output for the input p0-1-1.jpg



(c) Figura 15. Output for the input p0-1-0.jpg



(d) Figura 16. Output for the input p0-1-1.jpg

Figure 4: Output images for question 2.c

Similar to the previous interpretation, we say that the light colored regions in the output image are red colored regions in the original image.

d) Which image looks more like what you would expect a monochrome image to look like? Would you expect a computer vision algorithm to work on one better than the other? Why?

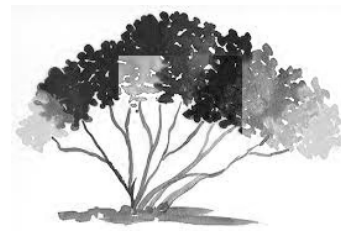
- 1) The monochrome image generated from the green channel is the image that most closely resembles the monochrome image that I would expect from the original image.
- 2) I think that in computer vision, the computing power of a computer is more important than the optimization of an algorithm, because in computer vision we work on the representation of an image, which is a matrix, then to perform any type of operation in a Matrix we will need to go through this in any way. In conclusion one algorithm works better than another depending on the environment in which it is executed.

### 3 Replacements of pixels.

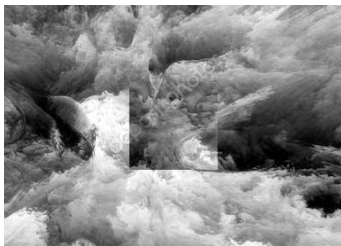
a) Take the center square region of 100 x 100 pixels of A and insert it into the center of B.



(a) Output for the input p0-1-0.jpg



(b) Output for the input p0-1-1.jpg



(c) Output for the input p0-1-2.jpg



(d) Output for the input p0-1-3.jpg

Figure 5: Output images for question 3.a

b) Replace the respective channel of B into the original image



(a) Output for the input p0-1-0.jpg



(b) Output for the input p0-1-1.jpg



(c) Output for the input p0-1-2.jpg



(d) Output for the input p0-1-3.jpg

Figure 6: Output images for question 3.b

In 3.a the central region of the image B take the values of the central region of the image A. The differences are easily visible.

In 3.b the output image is the same as the original image, except for the central region, the central region replaced its pixels of the green channel by pixels of the red channel, generating a new color.

## 4 Arithmetic and geometric operations.

- a) What is the min and max of the values of  $img - green$ ? What is the mean? What is the standard deviation? How do you compute them? Provide code snippets in your report for these questions.

The results for each  $img - green$  is summarized on table 1

Img. name	max	min	mean	std. deviation
p0-2-b-0.jpg	255	0	137.844325718	61.710497328
p0-2-b-1.jpg	255	0	198.553740686	76.377641762
p0-2-b-2.jpg	255	0	137.885818781	61.294456676
p0-2-b-3.jpg	255	0	138.209035652	77.913647393

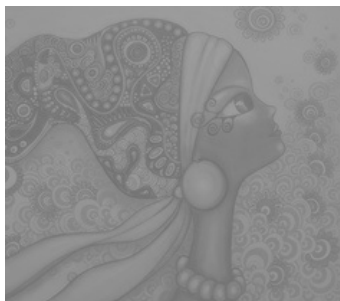
Table 1: Summary for image statistics

The code snippet for this problem is relative small using python:

```
import numpy as np
def question_A(img):
    img_max = img.max() #image max
    img_min = img.min() #image min
    img_mean = img.sum()/(1. * img.shape[0] * img.shape[1]) #image mean
    img_std = np.std(img.ravel()) #image standard deviation
    print ('max:', img_max, 'min:', img_min, 'mean:', img_mean, 'std:', img_std)
```

- b) Normalize the values of  $img - green$ . How the image looks like? Do you have an idea of what happened?

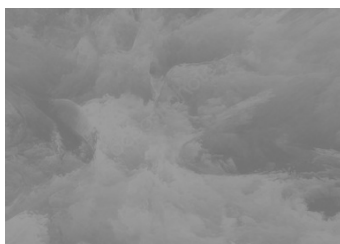
The results of the normalization are shown in figure 7.



(a) Output for  $img - green$  of p0-1-0.jpg



(b) Output for  $img - green$  of p0-1-1.jpg



(c) Output for  $img - green$  of p0-1-2.jpg



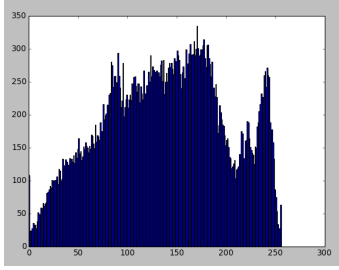
(d) Output for  $img - green$  of p0-1-3.jpg

Figure 7: Output images for question 4.b

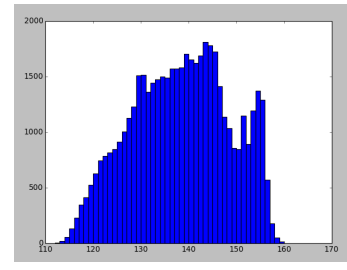
Comparing this images with the originals, it seems that a gray mask has been applied, this turns the image less visually-readable. The explanation for this effect can be done analysing the histograms that the images generate (figure 8). In the normalized image, the color are centered and go from a range of  $[0, 255]$  to a smaller one, note that the shape of the histograms (visual relation between bins size) remains the same, hence, the normalization works as a operation that push the bins to the center values.

- c) Shift  $img - green$  to the left by 2 pixels. Store the resulting image in the output folder. Subtract the shifted version to the original, and save the difference image. Make sure that all the values you are saving are legal so you can see all the differences. What do the negative values mean? What do you see in the result?





(a) Histogram for *img - green* of p0-1-0.jpg



(b) Histogram for *norm. img - green* of p0-1-1.jpg

Figure 8: Histograms for *img - green* and its normalized image

The results for the shifted image, and the original image subtracted with the shifted are shown in figure 9, the images shifted to left by two pixel present a black line which actually are the parts of the image that were replaced with 0 as part of the shifting. In the case of the subtracted image, an interesting effects accours: it presents approximation of the edges from the original images. This is really interesting and the min, max, mean and standard deviation (table 2) of the subtracted image can give an intuition of this effect.

Comparing the image statistics, we can see a big effect in the mean: it has decreased widely, since in RGB system a value 0 is equal to black, all the pixel that has negative value are represented with black, and they may represent no-edges, this is because edges are consider as elements with high frequency, thus, when subtracting with an adjacent pixel (shifted image) its value is usually bigger and the result is positive (not black).

Img. name	max	min	mean	std. deviation
p0-2-b-0.jpg	232	-211	1.00700073692	46.3247164932
p0-2-b-1.jpg	255	-253	1.83093889717	44.5053782822
p0-2-b-2.jpg	216	-168	0.64044309129	19.2998234779
p0-2-b-3.jpg	239	-234	0.60832567167	43.4115692626

Table 2: Summary for image statistics

## 5 Noise.

- a) Take the original colored image (*img*) and add Gaussian noise to the pixels in the green channel. Increase the sigma until the noise is visible in the image. Store the image into the output folder. What is the value of sigma you used? How did the noise in the resulting images behave? What did you observe?

The results of adding gaussian noise to the blue channel are shown in figure 10. The value of sigma used is 30, the noise tends to affect black/white-like areas, this is more clean in figures 10.b and 10.d, the noise does affect selective parts of the image.

- b) Add the noise using the same sigma to the blue channel instead. Store the output.

The results of adding gaussian noise to the blue channel are shown in figure 11.

- c) Which image looks better? Why?

The noise in the blue channel looks better because in the final image it generated points of color yellow that do no have a big contrast with the color around them, on the other hand, purple points are more notorious.



(a) Shifted image for input p0-1-0.jpg



(b) Original subtracted with shifted image for p0-1-0.jpg



(c) Shifted image for input p0-1-1.jpg



(d) Original subtracted with shifted image for p0-1-1.jpg



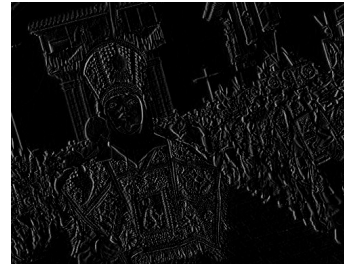
(e) Shifted image for input p0-1-2.jpg



(f) Original subtracted with shifted image for p0-1-2.jpg



(g) Shifted image for input p0-1-3.jpg



(h) Original subtracted with shifted image for p0-1-3.jpg

Figure 9: Output images for question 4.c



(a) Gaussian noise on p0-1-0.jpg blue channel



(b) Gaussian noise on p0-1-1.jpg blue channel



(c) Gaussian noise on p0-1-2.jpg blue channel



(d) Gaussian noise on p0-1-3.jpg blue channel

Figure 10: Gaussian noise on blue channel



(a) Gaussian noise on p0-1-0.jpg blue channel



(c) Gaussian noise on p0-1-2.jpg blue channel



(b) Gaussian noise on p0-1-1.jpg blue channel



(d) Gaussian noise on p0-1-3.jpg blue channel

Figure 11: Gaussian noise on blue channel