Assignment 1 - IS2U

CV701 - Human and Computer Vision

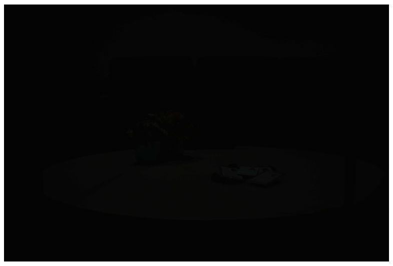
Khalil Alblooshi, Yevheniia Kryklyvets, Sebastian Cavada, Mohamed Insaf Ismithdeen

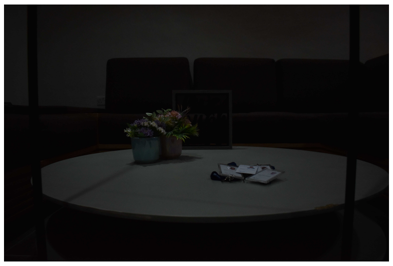
Task 1

To complete the requirements of the first task of the given assignment our group had to explore and experiment with various camera settings in order to portray how each specific one impacts the final image. Focal length, aperture size, shutter speed, and ISO are the parameters that vary from picture to picture. The setup itself included multiple objects located in different planes for a better demonstration of the depth of field and difference in focal length.

Below you can see the progression of our discoveries broken down for each parameter. The conclusion was that to achieve a better portrayal of the impact of each separate camera setting, it is important to keep all others fixed.

Effect of ISO


(a) ISO 100


(b) ISO 400

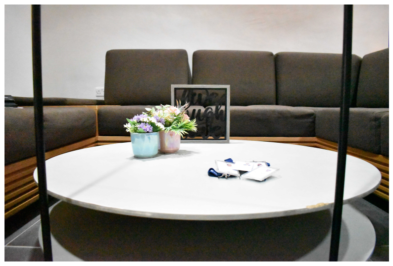
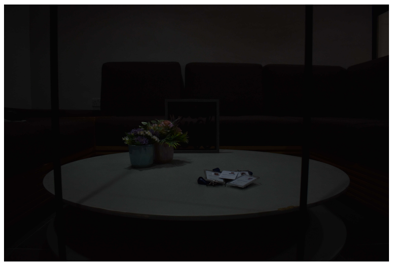
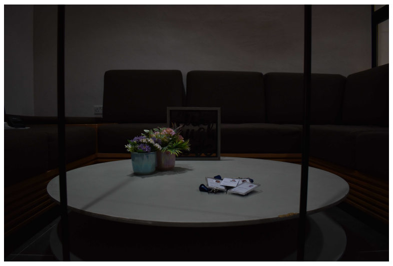

(c) ISO 25600

Figure 1: Fixed variables: Shutter speed: 1/400, Focal length: 18mm, Aperture: f/3.

The above images give a better understanding of how ISO, or the International Organization of Standards, impacts the resulting image. By definition, it defines the sensitivity of a digital sensor to light. All pictures were taken in the same light settings, but it is visible that by increasing the sensitivity a brighter image output was achieved. At the same time, it is important to mention that selecting a high ISO setting is not the best solution, because it creates noise. For better outcomes, it is important to consider other camera exposure parameters e.g. shutter speed.

Shutter speed


(a) Shutter speed: 1/250s


(b) Shutter speed: 1/100s

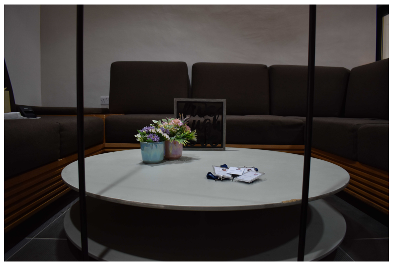
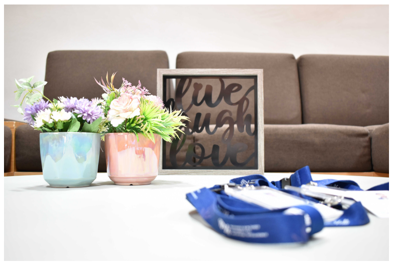
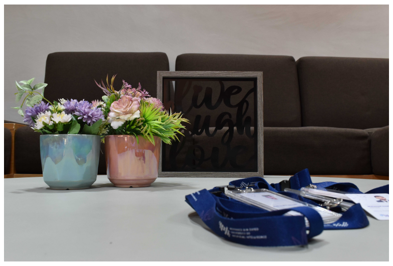

(c) Shutter speed: 1/40s

Figure 2: Fixed variables: ISO 400, Aperture 3.5, Focal length: 18mm.

Shutter speed directly impacts the exposure time of the sensor. It is a measurement of time (1/n seconds) that the camera's shutter is opened or powered, for the electronic ones. This is a more straightforward parameter that dictates a simple rule: the longer the shutter is open - the more light comes through. In the progression displayed above this linear dependency is seen clearly. By gradually increasing only shutter speed time it was possible to achieve a brighter image without any other parameter change.

Aperture


(a) Aperture: f/3.8


(b) Aperture: f/10

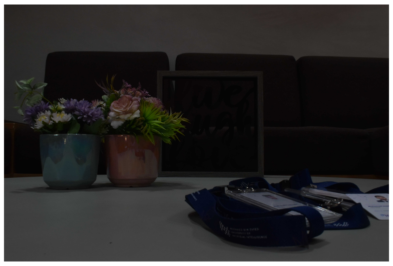
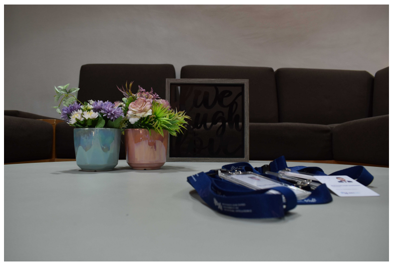
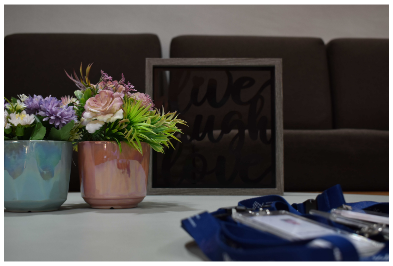

(c) Aperture: f/25

Figure 3: Fixed variables: ISO 800, Shutter speed 1/8, Focal length: 24mm.

The aperture defines the size of the lens opening. It is also the third parameter in the so-called "Exposure triangle" [1], with which we conclude the list of parameters that work together to regulate the effect of light on the final picture. The aperture works like the iris of the human eye by actually controlling the amount of photons that are passing through the lens. Using the provided images we can also see that it has linear dependency - a smaller f-number (bigger aperture size) allows more light to pass through. During the calibration process, an overly bright image was the result.

Focal Length


(a) Focal length: 18mm


(b) Focal length: 30mm

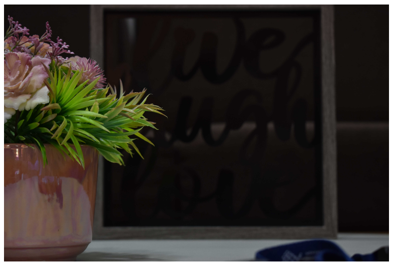
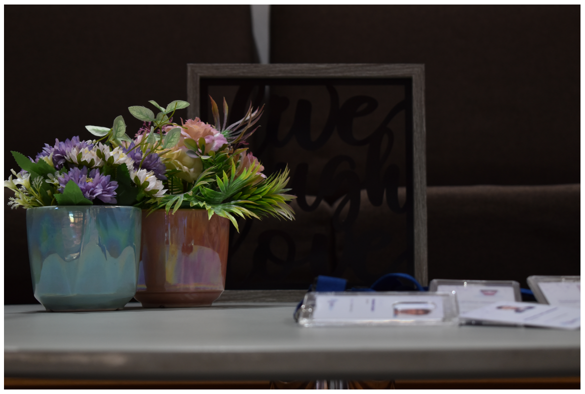

(c) Focal length: 55mm

Figure 4: Fixed variables: ISO 400, Shutter speed 1/20, Aperture: f/5.6.

This is a parameter that is not directly related to luminance, unlike previous ones, but its importance lies in defining the angle of view, which is how much of the scene will be captured. The effect of this parameter can also be easily defined: shorter focal length (in mm) results in a wider view angle that allows capturing more elements but with a cost of lower magnification value. It is important to note, that a bigger focal length requires a smaller aperture size, so for this experiment, to keep other constraints fixed, the minimum aperture value for the biggest focal length was chosen.

Summary

To summarise the impact and correlation between all four parameters described in previous sections it is important to look at them all together in different light settings. The below figure will serve as an example.


(a) Dark Environment: ISO 400, Shutter speed **1/15**, Aperture: **f/6.3**, Focal length: 35mm.

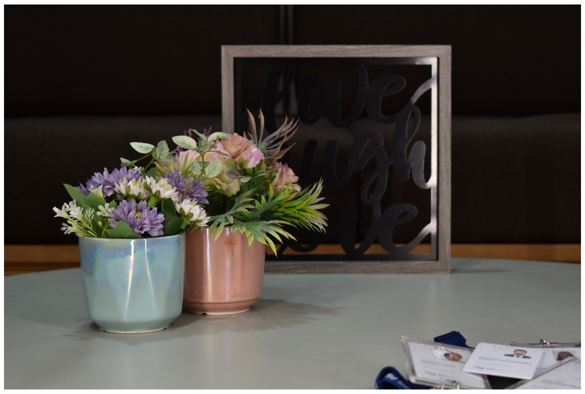

(b) Light Environment: ISO 400, Shutter speed **1/40**, Aperture: **f/5.6**, Focal length: 35mm.

Figure 5: Settings for Darker vs. Brighter Environment

The parameters highlighted in bold were changed depending on the amount of light in the room. For the experiment, two vertical LED lamps were placed in front of the captured objects. For the sake of this trial focal length was not changed. ISO value also remained unchanged due to a comparably minor luminescence increase. The difference between the displayed images lies in changes in aperture size and shutter speed. For an environment with more sources of light, the shutter speed was decreased and the aperture size was increased, to regulate the amount of natural light passing through. The opposite was applied to the environment with less natural light to give enough exposure time for the light-sensitive surface of the camera.

Task 2

Task 2.1

For this subsection of the second task, there were a couple of major parts that required attention. Implementation of white balancing algorithms for color correction (gray world and white patch) and unsharp masking algorithm. The resulting image of the mentioned sharpening technique then had to be used with previously noted color balancing methods to compare both outputs.

Gray World

The gray world is a white balancing algorithm that operates under the assumption that each color sensor averages gray throughout the entire image. This basically means that the entire scene has to be light gray - have a good distribution of colors.

Advantages:

- Easy to implement;
- Effective in specific environments;
- Widely used.

Disadvantages:

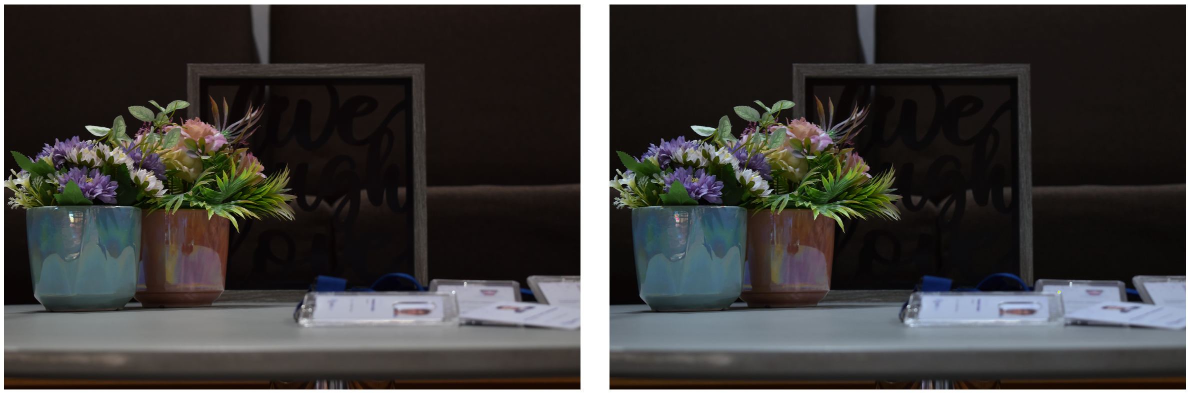
- Does not work well in color-imbalanced images.

For this task one of the most common normalization methods was used: producing an estimate of illuminant by computing the mean of each channel of the image.[10]

$$Si = \frac{avg}{avg_i} \tag{1}$$

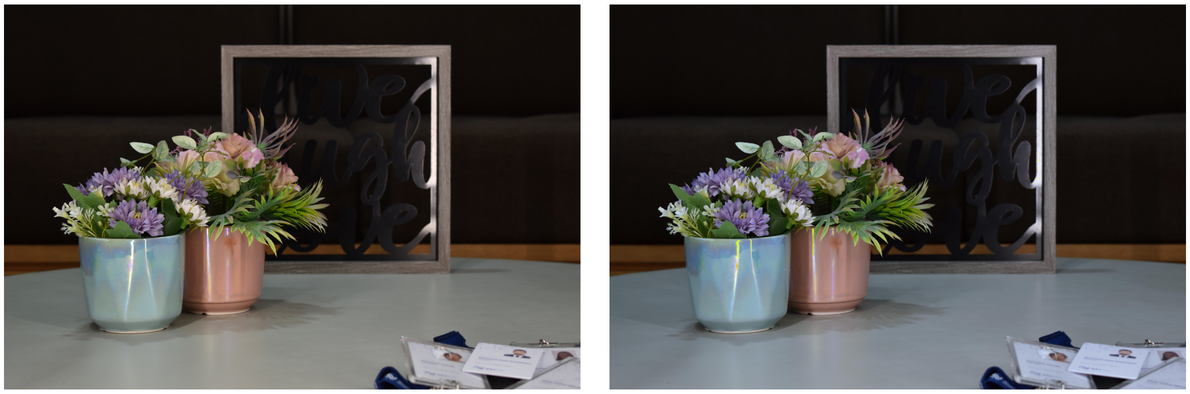
The above formula is used to calculate the value by which each pixel with respect to each color channel is being scaled in order to normalize the channels. Here avg is the illumination estimate (calculated as the average between all three colors) and avg_i - is the mean of a particular channel.

Based on these calculations and assumptions the gray world color correction on previously selected images was successfully implemented A.



(a) Dark environment image (b) Color corrected dark image

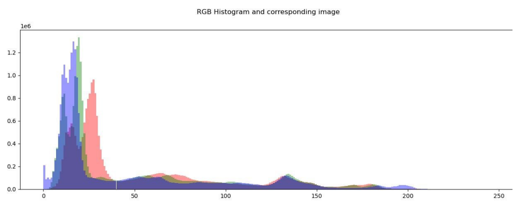
Figure 6: Comparison of a darker picture before and after gray world color correction



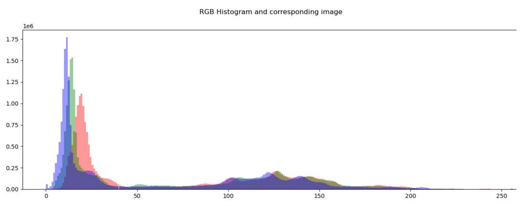
(a) Bright environment image (b) Color corrected bright image

Figure 7: Comparison of a brighter picture before and after gray world color correction

For both of the pictures, slight tinting changes are observed when looking at the table. After color correction reflected light is more white than yellow. This change is subtle due to the fact that all three color channels in this picture are indeed balanced. It can be verified by using the figure below that will show histograms C of the distribution of each color band in the original images.



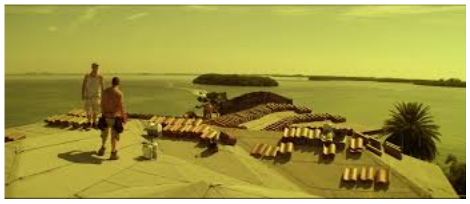
(a) Original dark image



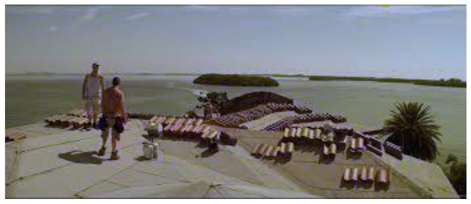
(b) Original bright image

Figure 8: View of RGB distribution in both original images

For the proof of concept, the algorithm was tested on the edge case - removal of specific color tint on the image A.



(a) Image with yellow tint



(b) Color corrected yellow image

Figure 9: Vivid example of gray world color correction usage

White Patch

White Patch is another example of white balancing algorithm. It operates under the assumption that White objects are supposed to have the same intensity values in all three color channels.

Advantages:

- Easy to implement;
- Effective in specific environments.

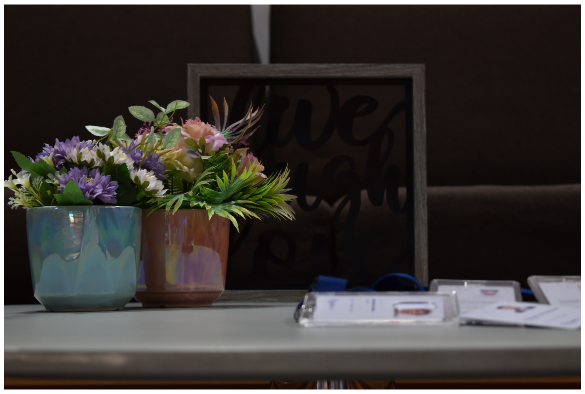
Disadvantage:

- Algorithm suffers when there is no or no easily identifiable white or neutral object in the picture.

Algorithm selected for this implementation assumes the highest value of each color channel as a white representation of the image. For this reason, it is important to first find the maximum pixel intensity per channel (illuminant) and then scale all pixels accordingly. [10]

$$o_i(x,y) = \frac{f_i(X,Y)}{I_{i_{max}}} \tag{2}$$

Above is the formula used for the computations. It is also worth mentioning that to add flexibility to the algorithm B and to be able to control the resulting brightness of the picture, a percentile parameter was added to it, which allowed avoidance of 255 color intensities as maximum in each channel.


(a) Dark environment image

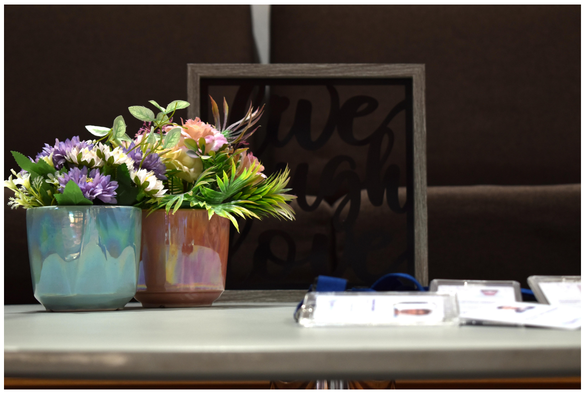
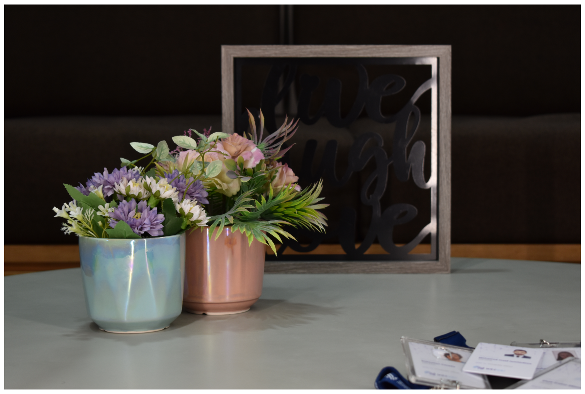

(b) Color corrected dark image

Figure 10: Comparison of a darker picture before and after white patch color correction


(a) Bright environment image

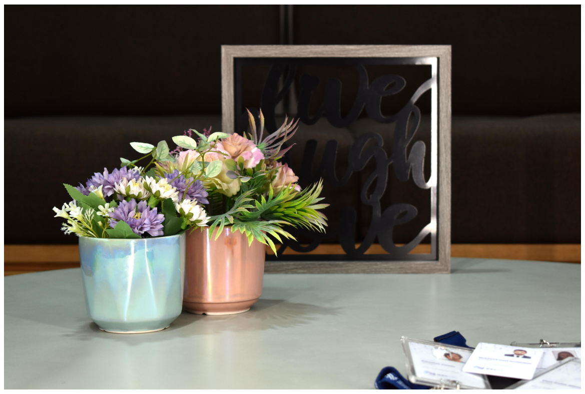

(b) Color corrected bright image

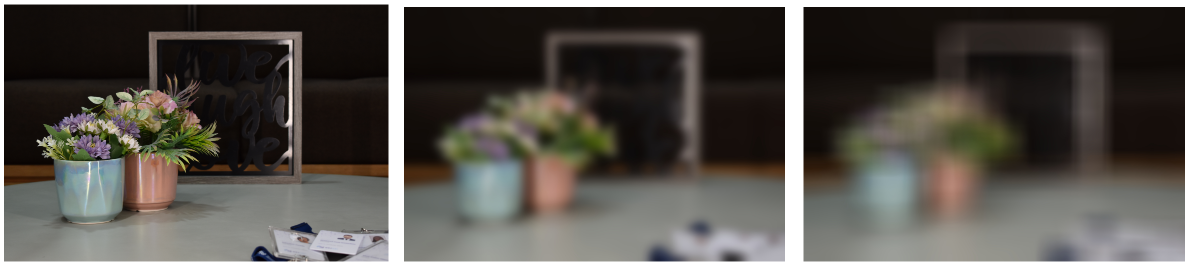
Figure 11: Comparison of a brighter picture before and after white patch color correction

For this particular set of pictures, the "White Patch" algorithm works better. It stretched all color bands in a way that gradually improved color quality and vibrancy, especially in the darker image set-up. Using different percentile values allowed the manipulation of maximum intensity values for each color band which resulted in the ability to see the change. This is due to the fact that both images have pixels with the highest possible intensities for each band. Using 100 percentile would have resulted in no adjustments for both pictures. Lowering this value resulted in visible color and light improvements for both output cases.

Task 2.1.2 - Unsharp masking

In image processing, unsharp masking is a process used to exaggerate the details of an image. This helps eliminate the noise and enhance focus on different objects within the scene. The process pertains to measuring the change quantity between different pixel values, later amplifying that change to highlight the region and make it obvious to the viewer.

Following is the way in which the unsharp masking was implemented:

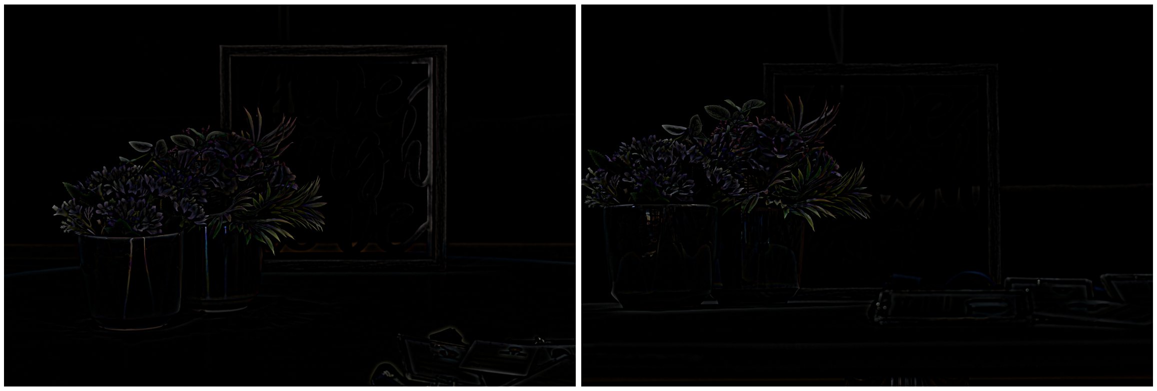


(a) Original image 1 (b) Gaussian smoothed image 1 (c) Average smoothed image 1

Figure 12: Differences between average and Gaussian smoothed filters

$gau\_smoothed = cv2.GaussianBlur(img, (501, 501), \sigma)$

As it is evident from the pictures, gaussian smoothing gives better blur results than averaging, hence it is likely to perform better when subtracting it from the original for a crispier image. Note that the high kernel window of (501, 501) was used for illustrating purposes, and it will be changed for optimum results in the coming steps. In both filters, we have used functions that are offered by the OpenCV library to perform the smoothing.

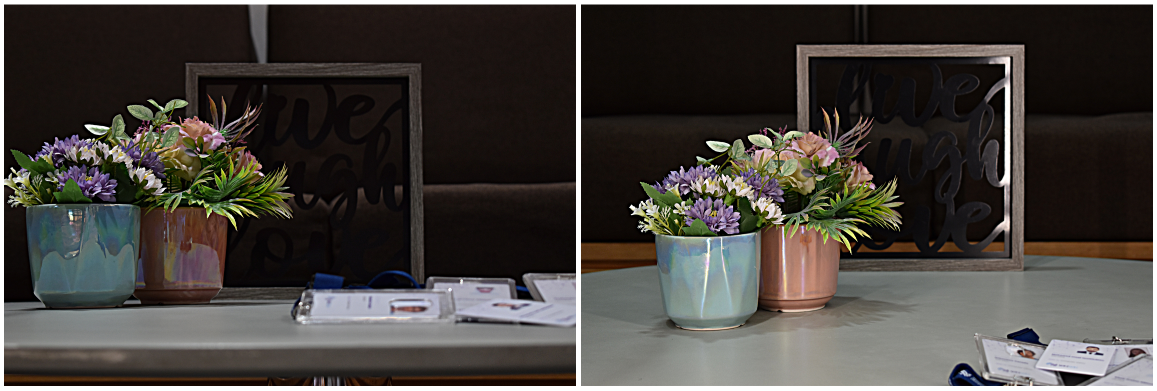


(a) Mask of dark image (b) Mask of bright image

Figure 13: Calculated mask for dark and bright image

$gau\_mask = cv2.subtract(img, gau\_smoothed)$

Figure 13 shows the result of subtracting the blurred image from the original to get the mask. The bright edges in the dark scene indicate the regions that will be added to the original image in order to sharpen it.



(a) Sharpened dark image (b) Sharpened bright image

Figure 14: Output images after unsharp masking

Above is the output of adding the mask to the original image (unsharp masking where sigma is equal to 20). The A multiplied by the mask decides how much of an effect that mask is to change on the original image (Higher A leads to more sharpening). High-boost filtering occurs on the occasion of increasing the A to greater values than one, yet it is not equally true that we get a better image since we'll lose many details, especially for intersected objects. The output images show better highlights in the regions where pixel values change drastically, thus allowing better observation of different color variations and objects within the image. In this example, some flowers appear to be easily distinguishable from the others when observed in the sharpened image than in the original. In addition, the words written in the wooden frame are much sharper and slightly easier to read.

To summarize this, the procedure of sharpening the image went as indicated arvis [4]:

- Blurred image = Gaussian blur filter (original image)
- Mask = original image − blurred image
- Enhanced image = original + (A * mask)

Task 2.1.3 - White balancing on sharpened image

For this part, it was decided to use a=1 when calculating for the unsharp masking image, which basically means that we just straight filter out the image using the selected Gaussian mask without any boost (high-boost filtering) because it gave us less extra noise when compared to other values. Input images are shown in Figure 14

When comparing results between the sharpened image and its color-corrected version in gray world 15 there is no particular correlation between how these two algorithms impact each other. At the same time, it does become obvious that performing these two algorithms together provides better output for further analysis due to the fact that it results in closer true color display and preservation of important edges and color features.

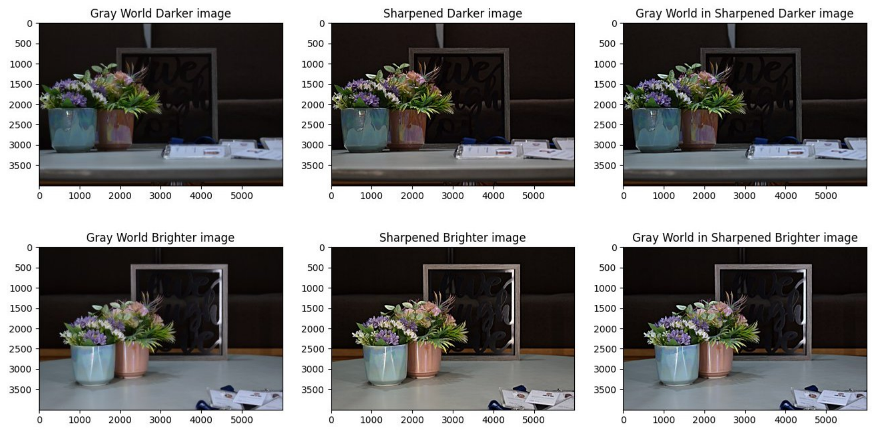

Gray World Darker image
Sharpened Darker image
Gray World in Sharpened Darker image
Gray World Brighter image
Sharpened Brighter image
Gray World in Sharpened Brighter image

Figure 15: Gray world on sharpened image

The plotted results 16 illustrate how applying white-patch results in blurring some of the edge details, such as how the edge of the badge diffused in the table. Yet, in the sharpened image this region was preserved to show clear isolation between the two objects. Hence, it's reasonable to consider applying the combination of both as it gives optimum output that provides both color improvement and crispier images.

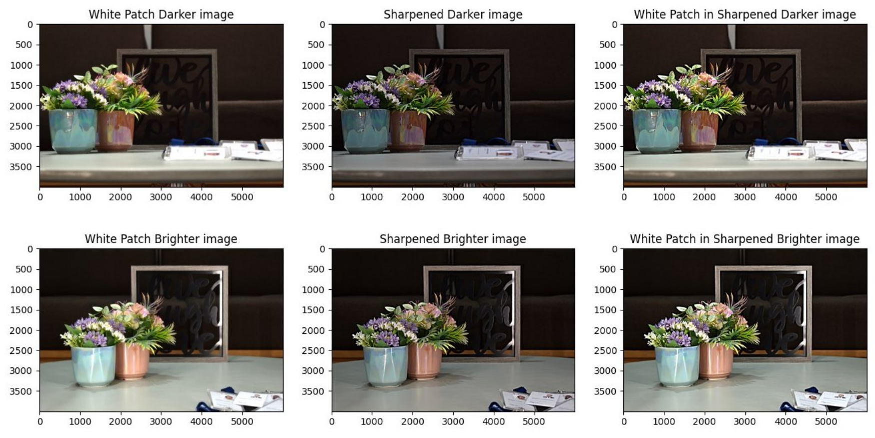

White Patch Darker image
Sharpened Darker image
White Patch in Sharpened Darker image
White Patch Brighter image
Sharpened Brighter image
White Patch in Sharpened Brighter image

Figure 16: White patch on sharpened image

Task 2.2

Image enhancement is a technique used to improve visual appeal of the digital image. There are two main goals associated with it - making the picture more aesthetically pleasing or improving effectiveness of message delivery, which is more relevant to the computer vision field. Variety of methods exist in order to enhance quality of the image making its features more distinguishable for the analysis. This part of the assignment contains detailed analysis of two of them: histogram equalization and contrast stretching.

Task 2.2.1 - Histogram equalization

Histogram equalization is a method that aims to improve the contrast of the image by stretching out its intensity range to the whole set of values, usually from 0 to 255. In other words, the algorithm tries to spread out values that are clustered around a particular intensity and "stretch" them out to use the full extent of the available values. In the following paragraph a more in-depth analysis will be performed on the aforementioned algorithm.

Histogram equalization can be achieved in many ways [5], below are the most common ones:

- Equalize every channel in the RGB image and then stack them together again
- Convert the image into a format where the luminance is isolated and perform histogram normalization on that channel, then convert the image back to RGB

Methods mentioned above were implemented and analyzed.

Both use the same technique to get the histogram equalized. The basic concept is that each histogram is treated as a probability mass function. The cumulative distribution function is also plotted and will be needed in the next step. The goal is to make the histogram follow, as close as possible, a uniform distribution. That entails the cumulative distribution function to approximate a line with a slope of 1. [7]. Furthermore, the basic idea of the technique can be summarized in the following steps [3]:

- Map one distribution (the histogram) to another distribution with intensity values spread over the whole range.
- The remapping function is the *cumulative distribution function (cdf)* of the original histogram $H(i)$. It's cumulative distribution $H'(i)$ is:

  $$H'(i) = \sum_{j=0}^{i} H(j)$$

  Note: To use this as a remapping function, the normalization of the maximum value to 255 is required.
- The final procedure is to remap the intensity values of the equalized image as follows:

  $$equalized(x, y) = H'(src(x, y))$$

  where *equalized* is the output and *src* is the input image.

Equalization on every RGB channel

The most common method to perform histogram equalization is to process every channel in the RGB image separately and then stack them together to form another RGB image [5]. Following this approach, the first histogram equalization was performed using the built-in function in OpenCV **cv::equalizeHist()**[2] which takes in input an array of values and outputs the equalized array of values. The input and output must be of type **np.uint8**.

In figure 17 it can be clearly seen that none of the histograms resemble a uniform distribution, on the contrary, the 3 histograms have a high spike on the left end of the graph. That means that a majority of dark colors and low-intensity are present, as indeed is expected from the big influence of the background. The code used to create the figure 17 can be found in appendix E.

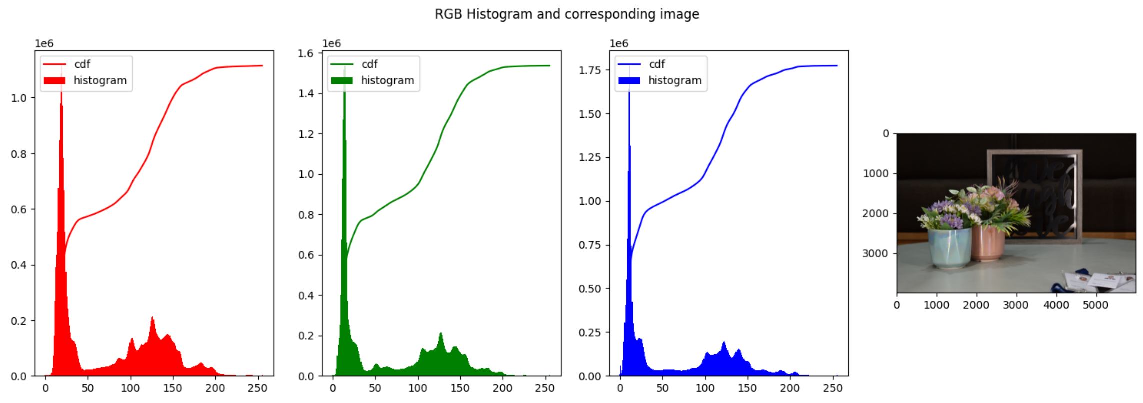


Figure 17: RGB channels with histogram and their respective CDFs

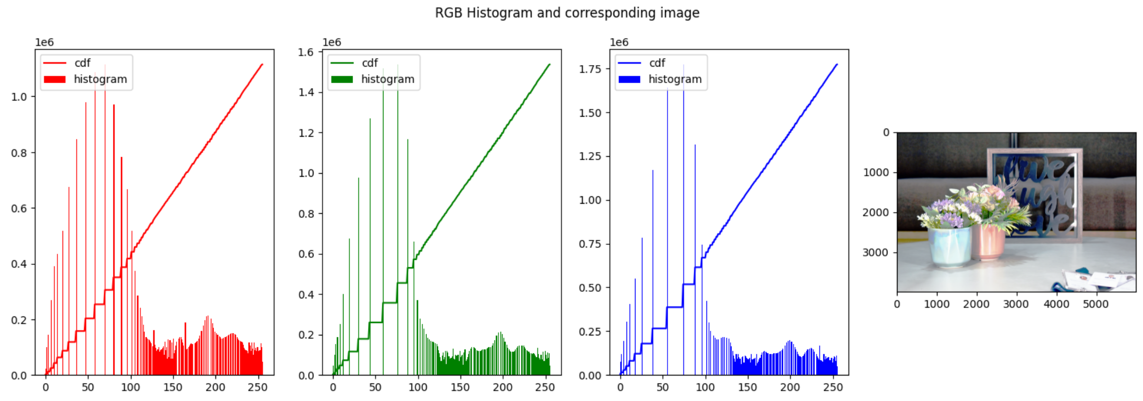


Figure 18: Equalized RGB channels with histogram and their respective CDFs

The second step of the procedure is to process 3 different channels performing histogram equalization on each of them. The results are shown in figure 18. After the equalization, every channel was ”stretched” to make it closer to a uniform distribution. That effect can be seen by looking at the cdf of the histogram that approximates the $x = y$ line.

Equalization on the I component in the HSI color space

The second technique foresees that the images have to be transformed into HSI color space first. In this new color space, the three channels are Hue, Saturation, and Intensity [8]. The intensity is not scattered among three different channels as it happens to be in RGB.

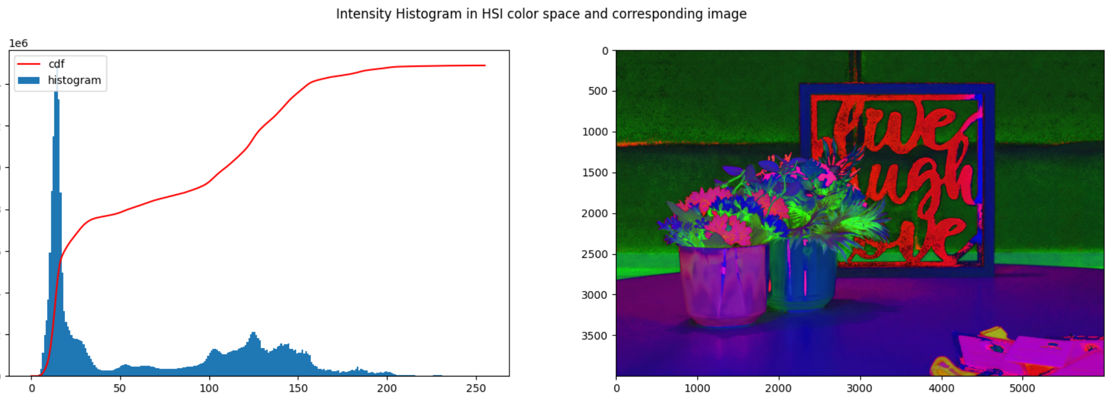


Figure 19: Non-equalized intensity channel with histogram and its CDF

Given the peculiar separation of the channels, histogram equalization can be performed just on the intensity. The result can thus be converted to RGB. To achieve this, the conversion of the image's color space using the function defined in appendix D is needed. The resulting image is plotted in figure 20.

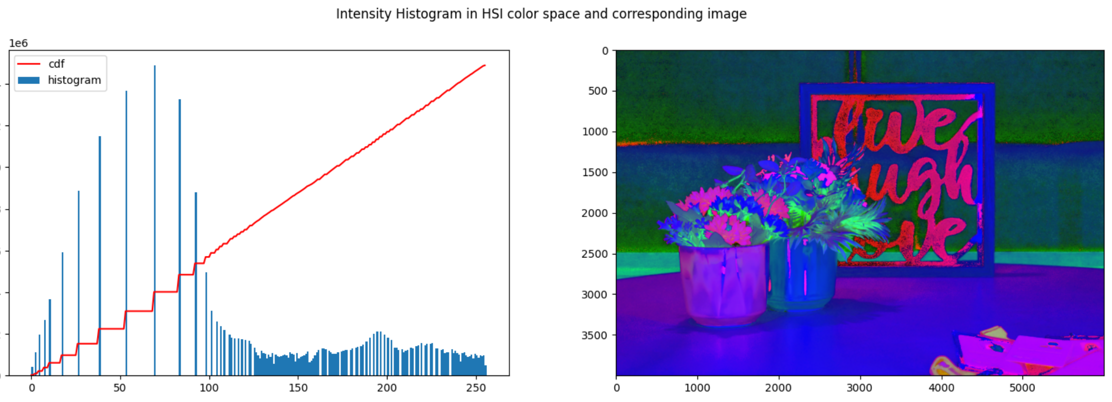


Figure 20: Equalized intensity channel with histogram and its CDF

At this point, the intensity of light is equalized in the HSI color space and can be converted back to the RGB color space, see figure 21.

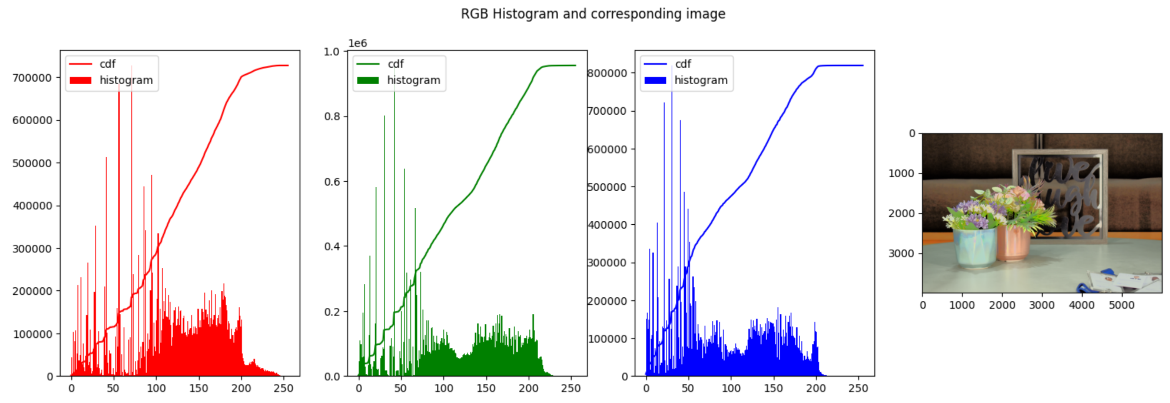


Figure 21: Equalized RGB channels with histogram and their respective CDFs

The result is similar to the previous technique, however, it has minor differences. In the right-most part of the histogram, it can be seen that the cdf does not approximate a uniform distribution, in the RGB color space.

Another experiment was performed on a different image to verify the correct implementation of the algorithm. Despite the phenomena decreased, it is still present in a minimal part like in figure 23. The image with its original histograms can be seen in figure 22.

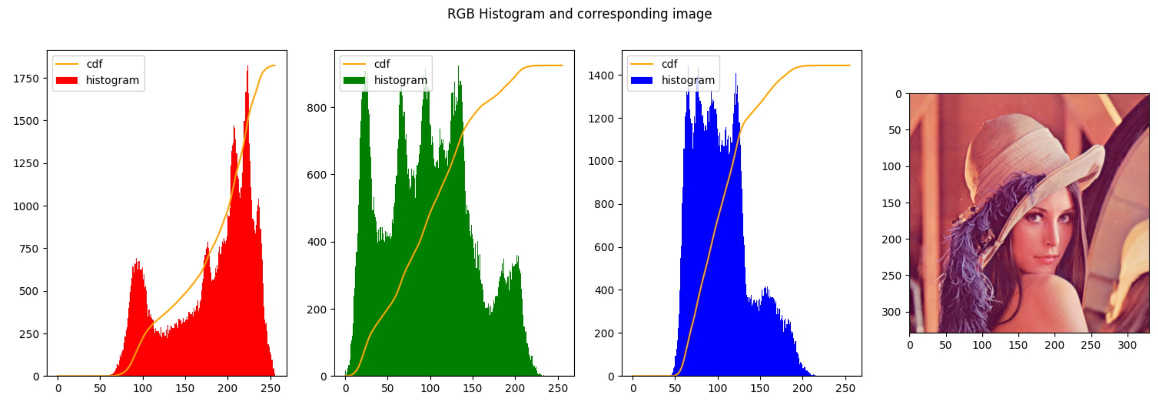


Figure 22: Other example of Non-equalized RGB channels with histogram and their respective CDFs

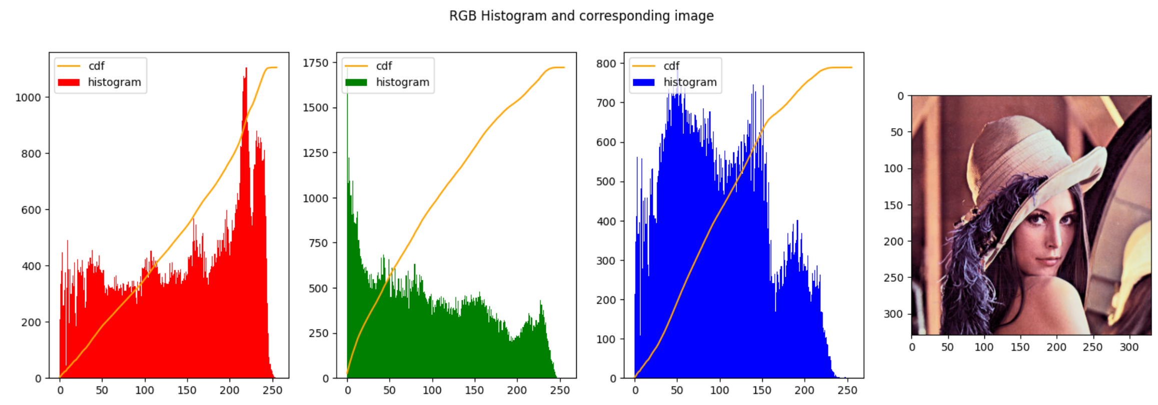


Figure 23: Other example of Equalized RGB channels with histogram and their respective CDFs

Task 2.2.2 - Contrast stretching

Effective contrast within an image plays a crucial role in delineating and highlighting intricate details, preventing them from blending into the background. An image lacks sufficient contrast, when there are suboptimal lighting conditions or nonlinear behavior of the equipment used to capture the image.

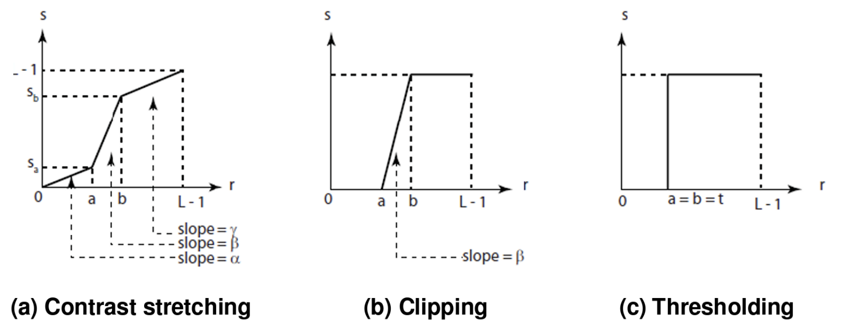


Figure 24: Contrast Stretching

Figure 24(a) shows the transformation used for Contrast Stretching [6]. This is a piece-wise linear mapping between the input image pixels and output image pixels. The degree of contrast enhancement or reduction is dictated by the slopes. If the slope exceeds one, it implies that the value levels are expanded because the levels are now spread across a different and wider range. Conversely, if the slope is less than one, it signifies a reduction in contrast. Clipping and Thresholding are two special cases of contrast stretching which are shown in figure 24(b) and 24(c) respectively.

In our case, α and β values are defined as shown in figure 25 [11]. Let the input intensity value be denoted by x. Three different functions are defined in terms of α and β values for x: $x < \alpha_1, \alpha_1 < x < \alpha_2, x > \alpha_2$ in order to map the input to the output intensity values. Reference to the code used for this is in Appendix F.

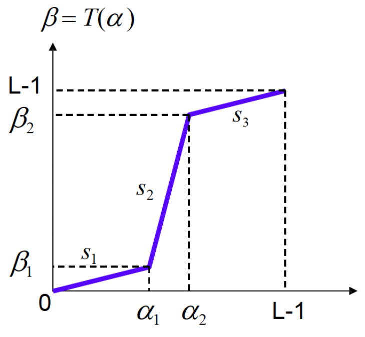


Figure 25: Contrast Stretching

Contrast Stretching can be achieved in two main ways [5]:

- Stretch every channel in the RGB image and then stack them together again
- Convert the image into a format where the luminance is isolated perform contrast stretching on that channel and then convert back the image to RGB

RGB Contrast Stretching

The methodology consisted of first dividing the image into different channels and then performing contrast stretching on each one. The final image is constructed by stacking together the three separate channels.

When performing contrast stretching on pictures taken within the scope of the assignment as well as the ones provided externally, it was vivid that all of them have relatively similar histograms for the three color channels, see figure 26. Thus major improvements after applying mentioned algorithm would be harder to observe. Thus the decision of using one of the most popular images in computer vision was made after extensive analysis of histograms of all originally available images.

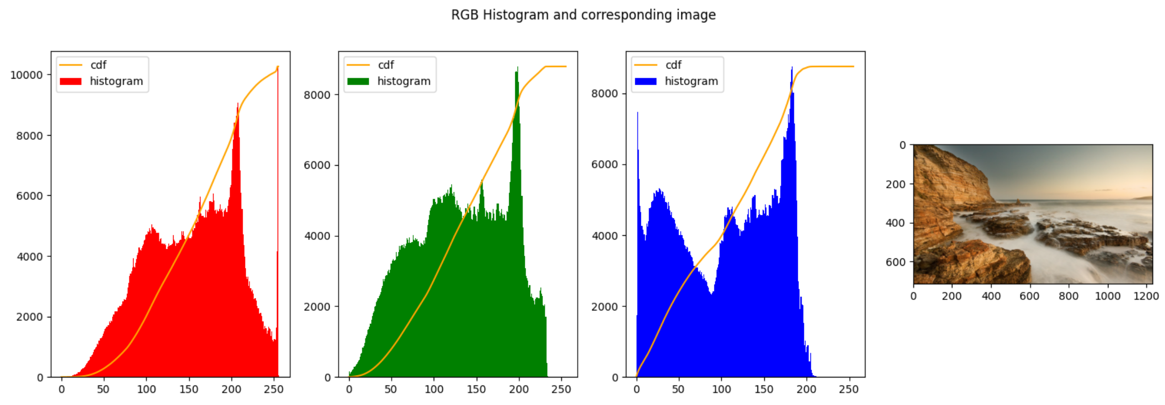


Figure 26: Image provided for the experiments with the histograms by each color channel

Figure 27 shows the Original image with the histograms of the R,G,B channels. In this case, a popular image was chosen to best show the effects of stretching.

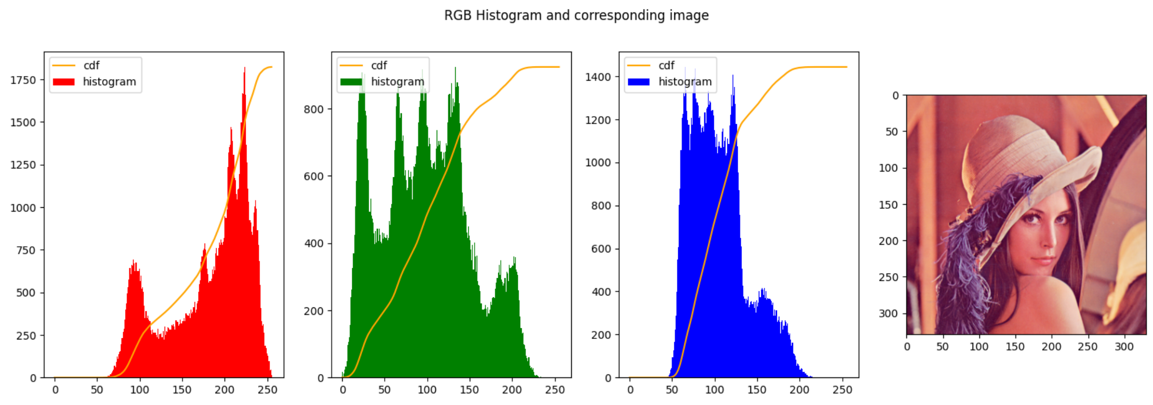


Figure 27: The original photo, with the histograms divided by channel

Different α and β values were tested to see the behavior of the intensity. In the first trial the α values were fixed to the range **0-255** and only the β values were changed.

When intensity values are mapped to higher values it makes the image lighter but does not improve contrast. (Figure 28)

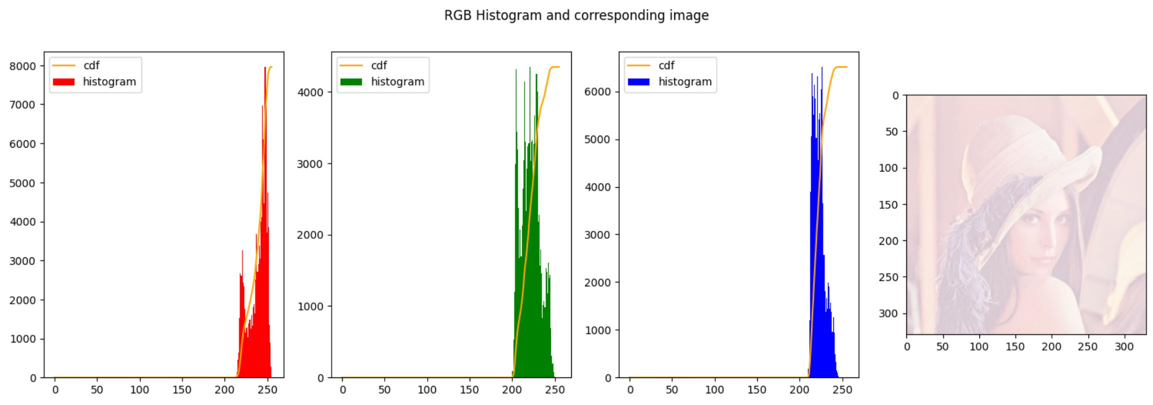


Figure 28: Contrast stretching with α = **(0,255)** β = **(200,255)**

On the contrary, when the intensity is mapped to lower values, the image becomes darker. This does not improve the contrast. 29

Figure 30 shows where there is a slight diminishing in contrast since the output intensity range is less, especially in the red and green channels.

After experimenting with different parameters, an image was produced in which the contrast did indeed increase 31. Some parameters search was done in order to find the ones that would fit best the need to increase the contrast of the image. While observing the figure it can be noted that in the red channel, there is a spike on the very end of the right side, this is given by our mapping,

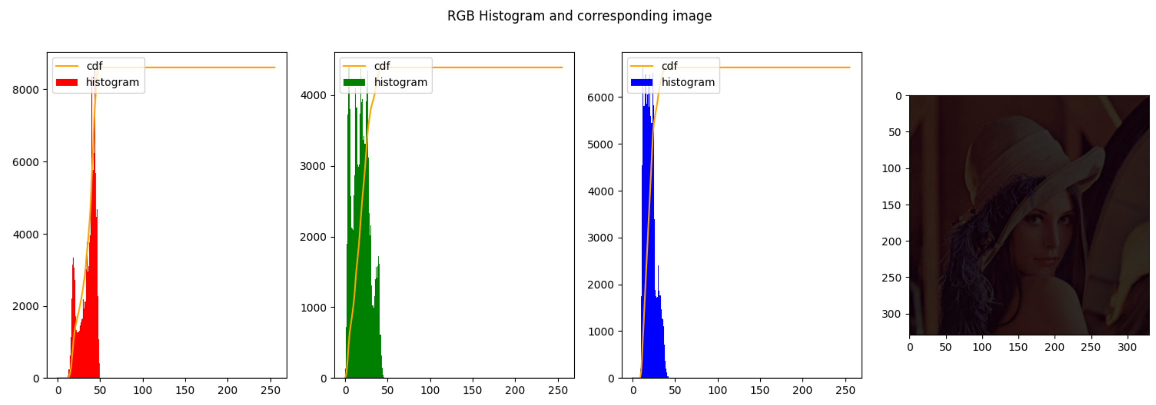


Figure 29: Contrast stretching with α = **(0,255)** β = **(0,50)**

which would map all the pixels with a red intensity value greater than α_2 to 255. Given that the red histogram is skewed towards the right end, this behavior was expected. The same thing happens with the green channel, this time at both ends. The blue channel is the one that gets more benefits from these particular values of α and β, as they are very close to the minimum and the maximum of the blue channel.

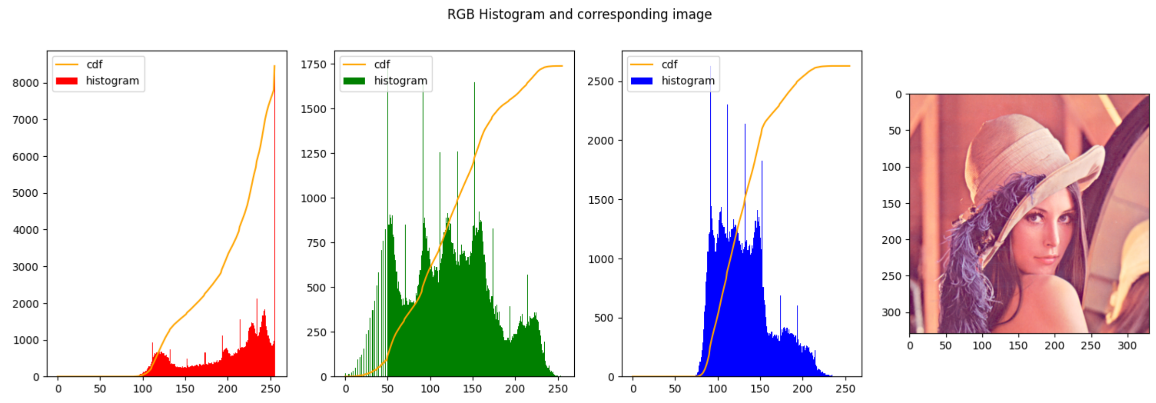


Figure 30: Contrast stretching with α = **(20,235)** β = **(50,255)**

For this reason, another function was implemented in which every channel could be stretched with different α and β values. In doing so an image with more contrast could be achieved. Fine-tuning every alpha to the min and max of every channel led to great improvements in the contrast as figure 32 demonstrates. β values were left unchanged to span the whole range (0-255), whether α were selected as follows: α_r = (60,225), α_g = (0,230), α_b = (45,220).

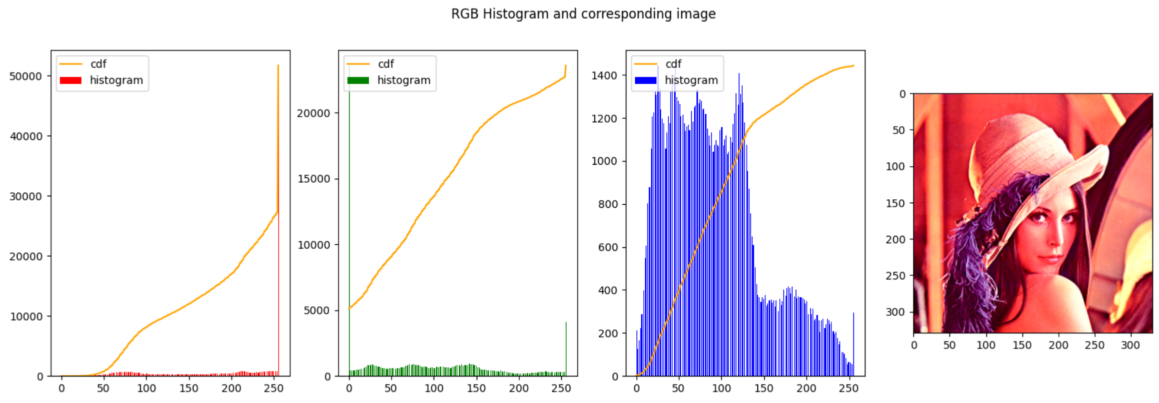


Figure 31: Contrast stretching with α = **(50,200)** β = **(0,255)**

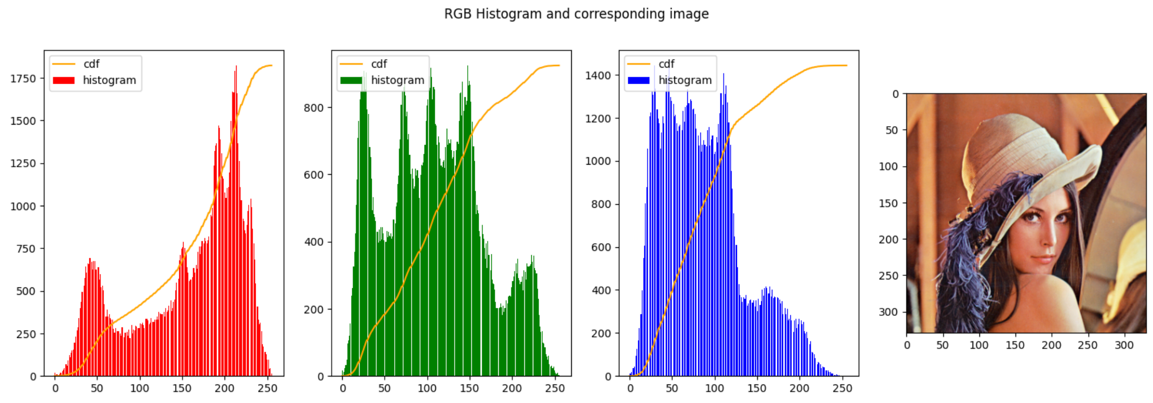


Figure 32: Contrast stretching with different α and β values for R,G,B channels namely: α_r = (60,225), α_g = (0,230), α_b = (45,220)

HSI Contrast Stretching

In the same way, that histogram equalization was performed, some experiments were carried out in the HSI color space to see if any differences or notable exceptions would occur. First of all the original image is displayed in figure 33. α = (0-150) and β = (0,240) were chosen for the first experiment. These values produced a satisfactory result in which the contrast was slightly increased while keeping the overall color fidelity as depicted in figure 35

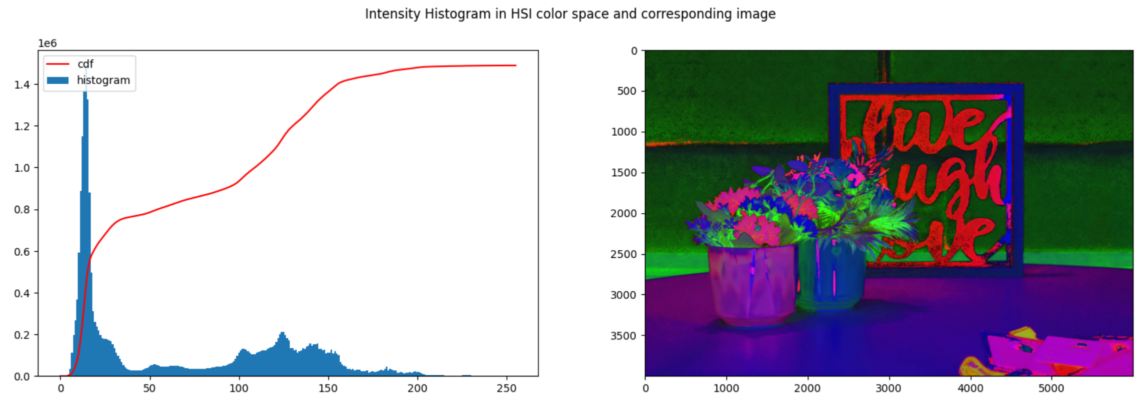


Figure 33: Original image mapped to HSI with its histogram

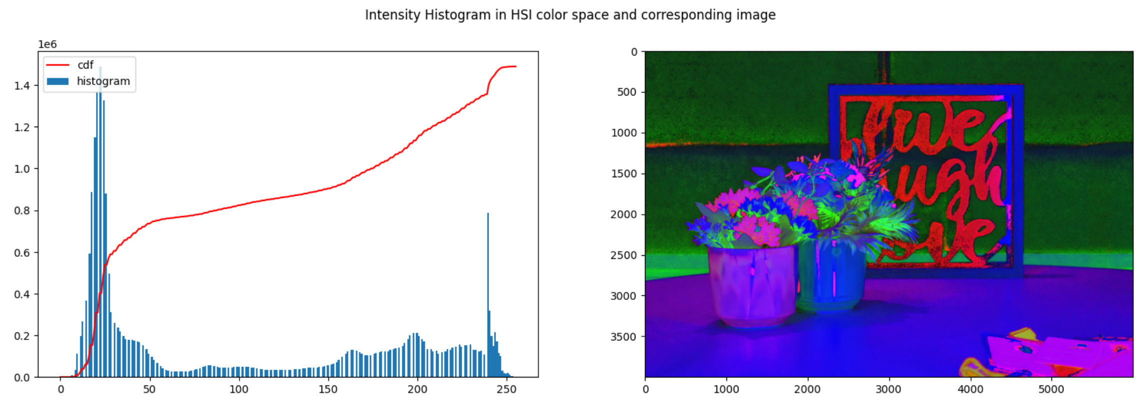


Figure 34: Image stretched in the HSI color space with α = (0,150) and β = (0,240)

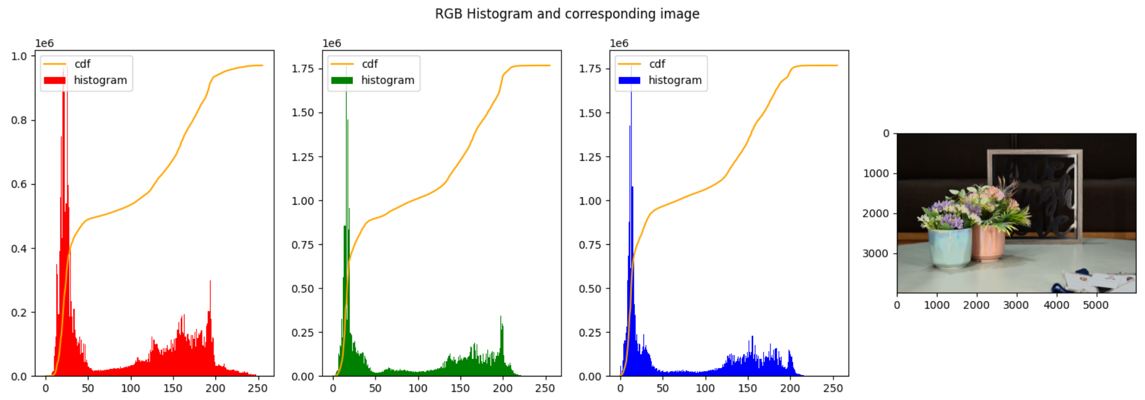


Figure 35: Image stretched converted back to RGB and its histograms

Another experiment was performed with different values of α being (0,50). This experiment stretched all the values between 0 and 50 to the range of 0 to 245. It can be noted in figure 36 that the values on the very right have a spike. That is due to the fact that the range between 50 and 255 of the input image is mapped to the interval of 245-255 to the output image. In other words, this means a bigger number of high-intensity pixels. The effect can be clearly seen in figure 37.

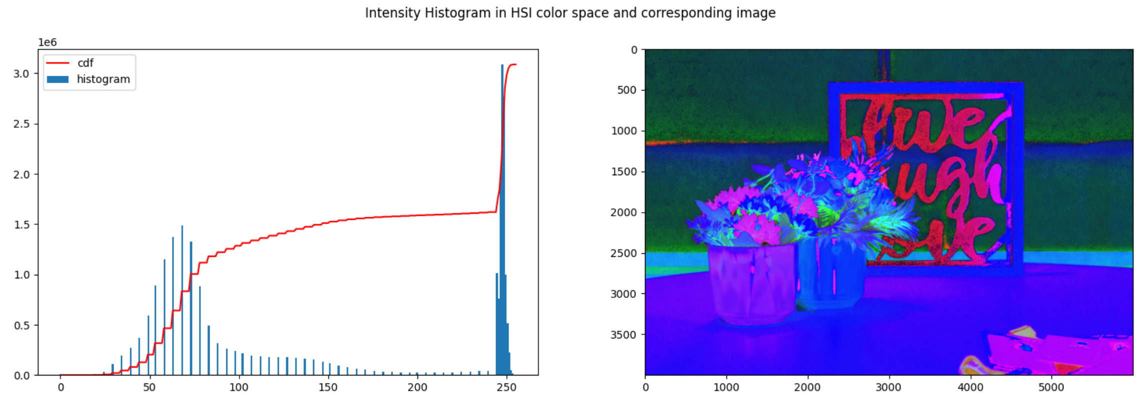


Figure 36: Contrast stretching in HSI with α = (0,150)

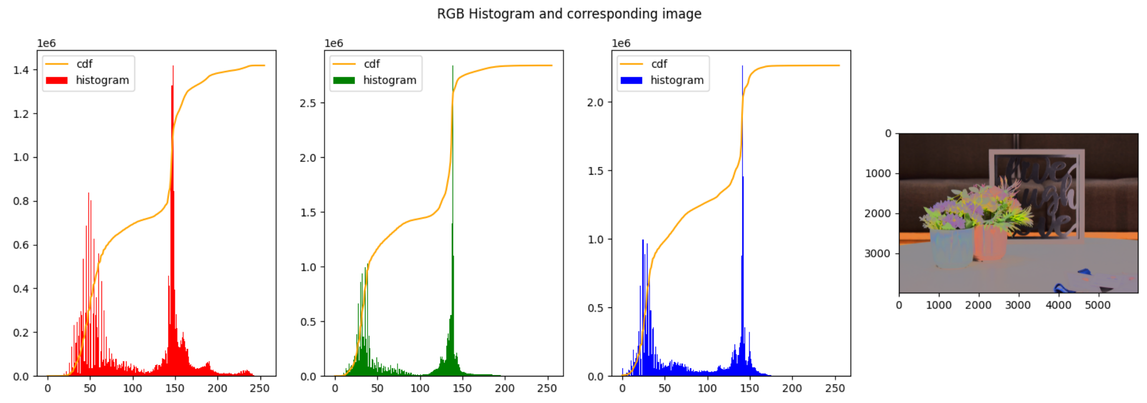


Figure 37: Image stretched converted back to RGB and its histograms

Task 2.2.3 - Analysis

In this section, the results are going to be discussed from the previous two sections, namely between Histogram Equalization and Contrast stretching.

From the previous experiments, it can be noted that histogram equalization is a subset of contrast stretching. Contrast stretching in general can be defined as: ”expanding the range of intensity levels in an image so that it spans the ideal full intensity range of the recording medium or display device” [7]. That means the expansion of the range is just a linear mapping in different intervals as discussed in section 2.2.2. In this linear transformation, the choice of alpha and beta is arbitrary (in the range [0, L-1]) and with some limitations: $\alpha_1 < \alpha_2$ and $\beta_1 < \beta_2$. The main difference is that the contrast stretching can be done on different regions of the image, while the histogram is stretched as a whole according to the transformation seen in section 2.2.1.

Different experiments were performed for both techniques to enhance the image. Both had successful results.

A comparison of the two different algorithms can be seen in figure 38 and 39. In the first pair of images, different algorithms were applied. Namely, the first image was applied histogram equalization. The second one instead of the contrast stretching was applied with α = (0, 150) β = (0, 240). These values were handpicked to show a good amount of contrast. Indeed in the first image, we can see higher contrast but also the noise was amplified, which is an undesired side-effect. In the second image, the contrast was slightly less and the noise is less as well.

The second pair of images was enhanced with contrast stretching by hand-picking the parameters to maximize the usage of all values of every channel, thus achieving a better and more natural result. Histogram equalization was performed on the HSI-converted second image. The result is a bit darker as can be seen in the histogram of figure 23 because a small range in the right-end of the histogram is not fully utilized.

Given an initial comparison of the data, further analysis of the algorithms’ impact can be inferred.

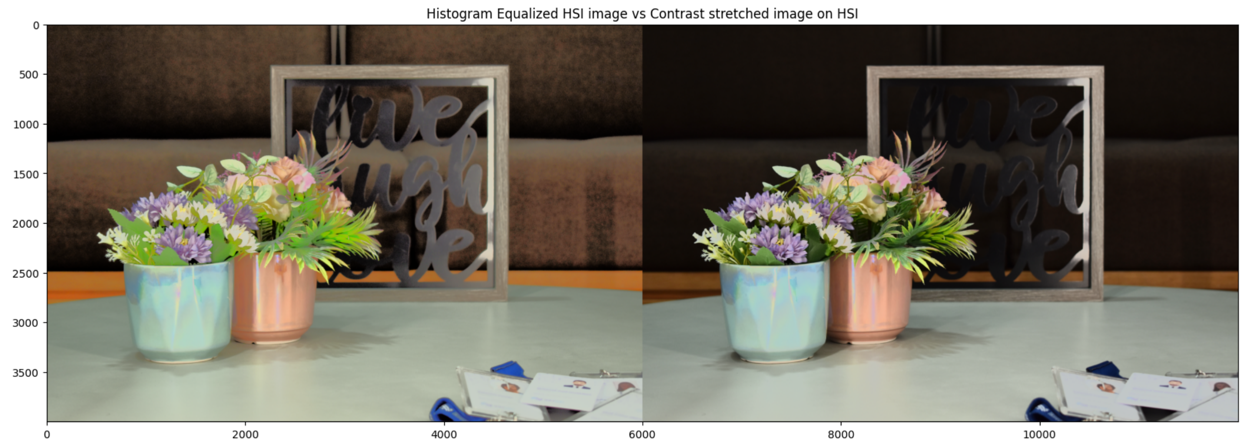

Histogram Equalized HSI image vs Contrast stretched image on HSI
0
500
1000
1500
2000
2500
3000
3500
0
2000
4000
6000
8000
10000

Figure 38: Histogram Equalized image on HSI vs Image stretched on HSI channel

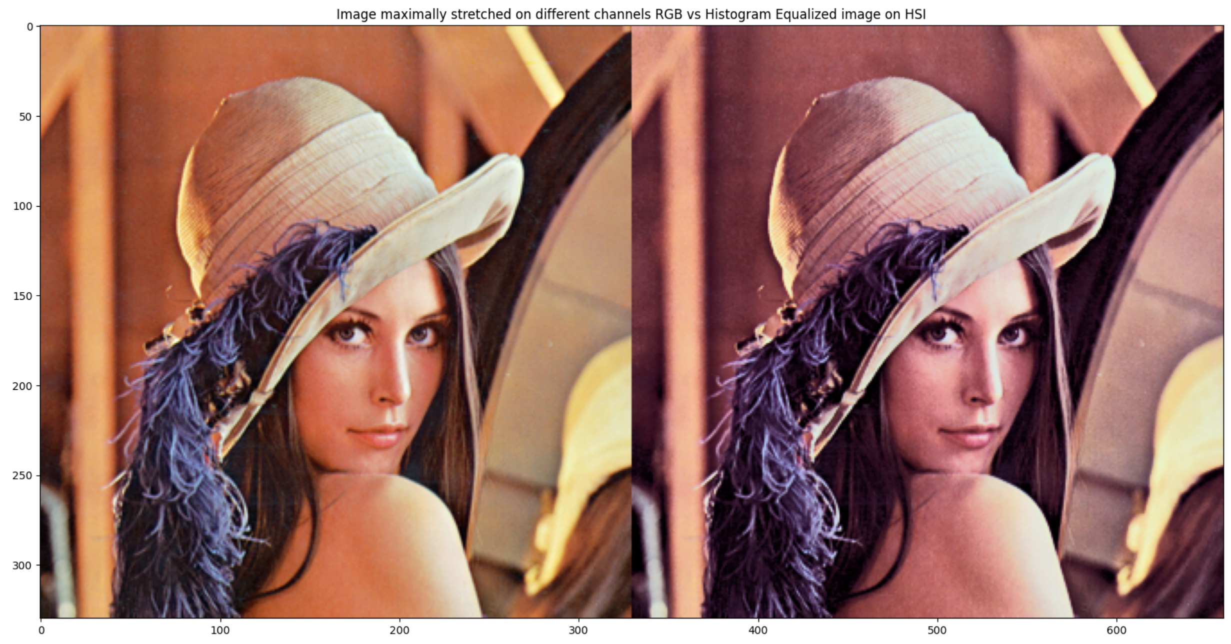


Image maximally stretched on different channels RGB vs Histogram Equalized image on HSI
0
50
100
150
200
250
300
0
100
200
300
400
500
600

Figure 39: Image maximally stretched on different channels RGB vs Histogram Equalized image on HSI

Advantages of Histogram equalization:

- The image would be enhanced on a specific mapping and over the entire range
- It is an automated way of enhancing the image: no need to tune any parameter

- Every channel can be equalized on its own.

Disadvantages of Histogram equalization:

- More computationally intensive than contrast stretching
- There is no way to explicitly change parameters and have a different outcome
- Tends to amplify noise - increase the contrast of its background and the distortion of the signal [9]

Advantages of image stretching:

- Simpler and straightforward algorithm
- Targets specific ranges of values where the concentration of intensity is higher and adjusts the light level in these ranges
- Possibility to increase or decrease a specific intensity of a channel and leave the other invariant
- Possibility to fine-tune the parameters according to the needs and the different use cases

Disadvantages of image stretching:

- Manual tuning of parameters can be difficult and requires additional steps/algorithms
- Potential loss of information - if not applied carefully some information could be lost due to over-saturation on some range of values

In conclusion, histogram equalization is more accurate and it works in most cases without the need to pre-compute some values that will adjust the image. However that brings the drawback of being less flexible than contrast stretching

Contrast stretching is more flexible and therefore requires the computation of the two values α and β. These can be computed by different algorithms. The catch is that the algorithm can be costly and sometimes a further analysis of the image metadata is needed to achieve enhanced results.

References

[1] URL https://www.bhphotovideo.com/explora/photography/tips-and-solutions/understanding-exposure-part-1-the-exposure-triangle.

[2] .

[3] . URL https://docs.opencv.org/3.4/d4/d1b/tutorial_histogram_equalization.html.

[4] Z. arvis. Image filtering in python - unsharp mask, 2020, May, 20. URL https://youtu.be/_p_36DIJMIw. Accessed on 17 Sun, 2023.

[5] I. M. Bockstein. Color equalization method and its application to color image processing. *J. Opt. Soc. Am. A*, 3(5):735–737, May 1986. doi: 10.1364/JOSAA.3.000735. URL https://opg.optica.org/josaa/abstract.cfm?URI=josaa-3-5-735.

[6] A. Choaudhary and S. Sharma. Unit-5 image transformations frequency domain, Jan 1970. URL http://egyankosh.ac.in//handle/123456789/90216.

[7] R. Gonzalez and R. Woods. *Histogram Equalization*, page 134–140. Pearson Education Limited, 2017.

[8] R. Gonzalez and R. Woods. *Histogram Equalization*, page 411–418. Pearson Education Limited, 2017.

[9] N. Longkumer, M. Kumar, and R. Saxena. Contrast enhancement techniques using histogram equalization: A survey. 2014. URL https://api.semanticscholar.org/CorpusID:54878553.

[10] M. Singh. Enhanced color correction using histogram stretching based on modified gray world and white patch algorithms. *IJCSIT*, 5(3):4762–4770, 2014. doi: ISSN:0975-9646. URL https://www.ijcsit.com/docs/Volume%205/vol5issue03/ijcsit20140503454.pdf.

[11] D. M. Yaqub. Cv701:lecture 3.2: Image filtering and image enhancements, 2023. URL https://mbzuaiac-my.sharepoint.com/:b:/g/personal/mohamed_ismithdeen_mbzuai_ac_ae/EVPAkFywjixDov8gDTBBqUABUbRPCQRs5GUNZOSmubMrjQ?e=f49oPl. Accessed on Sep 18, 2023.

Appendices

A Code for Gray World color correction

```
def gray_world_algorithm(img):
  # for the gray world we need to scale each channel based on illumination
  # estimate and channel mean to normalize values of all three color channels
  # s = avg(avg_R, avg_G, avg_B)/avg_i, where i = {R, G, B}
  rgb = np.float32(img)
  red = rgb[:, :, 0]
  green = rgb[:, :, 1]
  blue = rgb[:, :, 2]

  #let's calculate illumination estimate avg
  avg_R = np.mean(red)
  avg_G = np.mean(green)
  avg_B = np.mean(blue)

  avg = np.divide(sum([avg_R, avg_G, avg_B]), 3)

  red_scalar_avg, green_scalar_avg, blue_scalar_avg = 0, 0, 0

  #let's scale all three colors
  for indx, red_value in enumerate(red):
    red[indx] = np.multiply(red_value, np.divide(avg, avg_R))
    red_scalar_avg += np.divide(avg, avg_R)

  for indx, green_value in enumerate(green):
    green[indx] = np.multiply(green_value, np.divide(avg, avg_G))
    green_scalar_avg += np.divide(avg, avg_G)

  for indx, blue_value in enumerate(blue):
    blue[indx] = np.multiply(blue_value, np.divide(avg, avg_B))
    blue_scalar_avg += np.divide(avg, avg_B)

  # building modified image
  gray_world_image = np.copy(img)
  gray_world_image[:, :, 0] = red
  gray_world_image[:, :, 1] = green
  gray_world_image[:, :, 2] = blue

  return gray_world_image, [np.divide(red_scalar_avg, len(red)),
                            np.divide(green_scalar_avg, len(green)),
                            np.divide(blue_scalar_avg, len(blue))]
```

B Code for White Patch color correction

```
def white_patch_algorithm(img, percentile = 95):
  # We have to stretch each color band that way, so we are able to normalize colors usi
  # For this we will use intensity values of each color retrieved with the help of perc
  # Which will help with avoiding 'white' pixels in order not to make resulting image o

  img = img.copy()

  red = img[:, :, 0]
```

```
    green = img[:, :, 1]
    blue = img[:, :, 2]

    # Calculation of percentiles for each band
    red_perc = np.percentile(red, percentile)
    green_perc = np.percentile(green, percentile)
    blue_perc = np.percentile(blue, percentile)

    # let's normalize all color bands based on these values
    img = img / [red_perc, green_perc, blue_perc]

    return img.clip(0,1)
```

C Code for Joined Color Histogram

```
def create_histogram_rgb(img=None):
    #plotting histogram for all color bands to verify balancing
    fig, axs = plt.subplots(1, 2)
    fig.suptitle('RGB Histogram and corresponding image')
    fig.set_size_inches(16, 5)

    color = ('red','green','blue')
    for i,col in enumerate(color):
        axs[0].hist(img[:,:,i].ravel(),256,[0,256], color = col, alpha = 0.4)

    axs[1].imshow(img)
    plt.show()
```

D Code for RGB to HSI conversion and vice-versa

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

def HSI_to_rgb(img):
    h = img[:,:, 0]
    s = img[:,:, 1]
    i = img[:,:, 2]

    h1 = h < 1/3 * 2 * np.pi
    h2 = (1/3 * 2 * np.pi <= h) & (h < 2/3 * 2 * np.pi)
    h3 = 2/3 * 2 * np.pi <= h
    r = np.zeros_like(h)
    g = np.zeros_like(h)
    b = np.zeros_like(h)
    h_curr = np.zeros_like(h)

    # case 1
    h_curr[h1] = h[h1]
    b[h1] = i[h1] * (1 - s[h1])
    r[h1] = i[h1] * (1 + (s[h1] * np.cos(h_curr[h1])) / (np.cos(np.pi/3 - h_curr[h1])))
    g[h1] = 3 * i[h1] - (r[h1] + b[h1])

    # case 2
    h_curr[h2] = h[h2] - (2/3 * np.pi)

    r[h2] = i[h2] * (1 - s[h2])
    g[h2] = i[h2] * (1 + (s[h2] * np.cos(h_curr[h2])) / (np.cos(np.pi/3 - h_curr[h2])))
    b[h2] = 3 * i[h2] - (r[h2] + g[h2])

    # case 3
    h_curr[h3] = h[h3] - (4/3 * np.pi)

    g[h3] = i[h3] * (1 - s[h3])
    b[h3] = i[h3] * (1 + (s[h3] * np.cos(h_curr[h3])) / (np.cos(np.pi/3 - h_curr[h3])))
    r[h3] = 3 * i[h3] - (g[h3] + b[h3])

    return np.stack((r,g,b),axis=2)

def rgb_to_HSI(img):

    with np.errstate(divide="ignore", invalid="ignore"):

        # load image with 32 floating point type
        rgb = np.float32(img)/255

        # Separating RGB
        r = rgb[:,:,0]
        g = rgb[:,:,1]
        b = rgb[:,:,2]
        r = r.astype(np.float32)

        # Intensity
        intensity = (r+b+g)/3

        # Saturation
        min = np.minimum(np.minimum(r,g),b)
        sat = 1 - (3/(r+g+b+0.0000001)*min)

        # Hue
        hue = np.arccos((0.5*((r-g)+(r-b)))/((np.sqrt((r-g)**2+(r-b)*(g-b)))))
        hue[b>g] = 2*np.pi - hue[b>g]

        return np.stack((hue,sat,intensity),axis=2)
```

E Code for creating RGB histograms

```
def create_histogram_rgb(img=None):
    fig, axs = plt.subplots(1, 4)
    fig.suptitle('RGB Histogram and corresponding image')
    fig.set_size_inches(18.5, 5.5)

    color = ('red','green','blue')

    cdfs = np.empty(shape=(3,256))
    hists = []
    tot_max_cdf = 0

    for i,col in enumerate(color):
        hist,_ = np.histogram(img[:,:,i].flatten(),256,[0,256])
        cdf = hist.cumsum()
        hists.append(hist)
        cdfs[i] = cdf
        if(cdf.max() > tot_max_cdf):
            tot_max_cdf = cdf.max()

    for i,col in enumerate(color):
        cdf_normalized = cdfs[i] * float(hists[i].max()) / np.max(cdfs.flatten())
        axs[i].plot(cdf_normalized, color = col)
        axs[i].hist(img[:,:,i].ravel(),256,[0,256], color = col)

    axs[0].legend(('cdf','histogram'), loc = 'upper left')
    axs[1].legend(('cdf','histogram'), loc = 'upper left')
    axs[2].legend(('cdf','histogram'), loc = 'upper left')
    axs[3].imshow(img)

    plt.plot()
    plt.show()
```

F Code for stretching the histogram of one channel of an image with alpha and beta

```
def contrast_stretch_channel(image, alpha, beta):

    if len(image.shape) > 2:
        raise Exception("Image must be of only one channel")

    if alpha[0] < 0 or alpha[1] < 0 or beta[0] < 0 or beta[1] < 0:
        raise Exception("Alpha and beta must be positive")

    if alpha[0] > 255 or alpha[1] > 255 or beta[0] > 255 or beta[1] > 255:
        raise Exception("Alpha and beta must be smaller than 256")

    max_val = image.flatten().max()

    c = image.copy()
    cn = np.zeros_like(c)

    alpha1 = alpha[0]
    alpha2 = alpha[1]

    beta1 = beta[0]
    beta2 = beta[1]

    img_bit_mask_1 = c <= alpha1
    img_bit_mask_2 = (alpha1 < c) & (c < alpha2)
    img_bit_mask_3 = alpha2 <= c

    s1 = (beta1 / (alpha1 + 0.00001))
    s2 = ((beta2 - beta1) / (alpha2 - alpha1 + 0.00001))
    b2 = beta2 - alpha2 * s2
    s3 = ((max_val - beta2) / (max_val - alpha2 + 0.00001))
    b3 = max_val - max_val * s3

    cn[img_bit_mask_1] = c[img_bit_mask_1] * s1
    cn[img_bit_mask_2] = c[img_bit_mask_2] * s2 + b2
    cn[img_bit_mask_3] = c[img_bit_mask_3] * s3 + b3

    return cn
```