Activity 1: DIGITAL IMAGE FORMATION AND ENHANCEMENT

Genesis Vertudez – 202003099

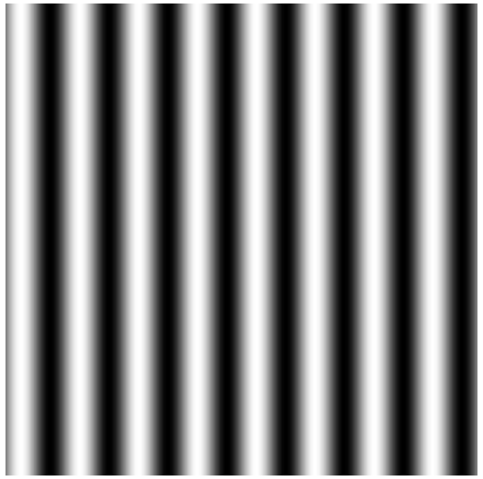
App Physics 157 - Computational Analysis and Modeling in Physics

Submitted to Dr. Maricor Soriano; Mx. Rene Principe Jr.

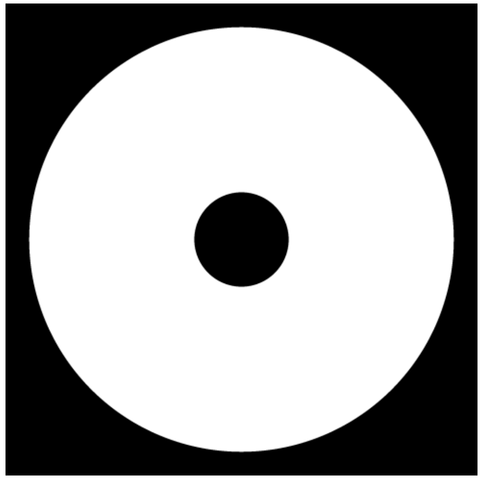
OBJECTIVES

- Mathematically create images
- Compare different file types for images
- Enhance grayscale and colored images
- Use backprojection to manipulate the histogram of an image to a desired distribution
- Compare contrast stretching, gray world algorithm, and white patch algorithm for white balancing of an image

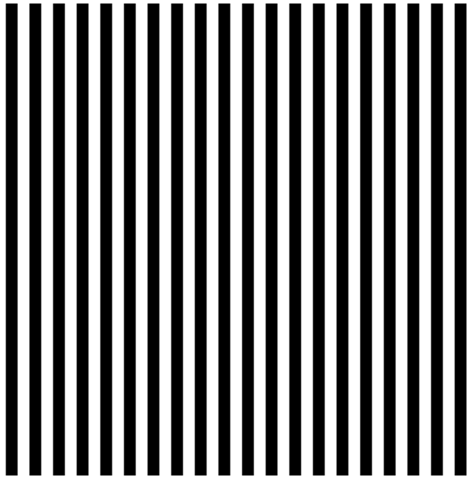
MATHEMATICALLY CREATE IMAGES



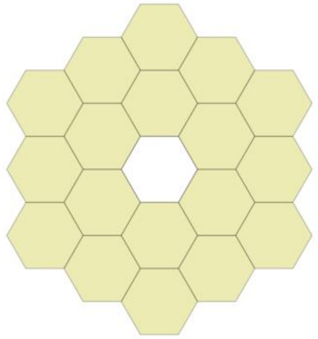
Sinusoid
This image of the sinusoid is in the top view. This is a 400x400 pixel grayscale image, with 200 pixel per cm, where the blacks and whites represent the crests and troughs of the sinusoid, respectively. The function I used here is $y = \sin 8\pi x$, so its frequency is 4 cycles per cm (which we can see there are four white bands in half the image). The x and y values were stored in a meshgrid.



Hubble's primary mirror (annulus)
This image is a binary image composed of 1's (white) and 0's (black). I generated this by creating a 2000x2000 meshgrid of 0's with the center as the origin. Then, I calculated each pixel's distance from the origin. From this, I assigned a pixel value 1 when the distance is less than a desired radius, hence creating a white circle. For the inner hole, I just switched the value to back to 0.



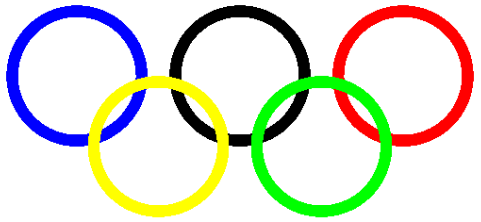
Grating
This image of the grating can be thought of a square wave in the top view. This is a 400x400 pixel grayscale image, where the blacks and whites represent the crests and troughs of the square wave, respectively. This was created by storing alternating 1's and -1's (or any set of binary numbers) where the higher value correspond to white. The x and y values were stored in a meshgrid.



JWST primary mirror (hexagon array)
This image is a colored simulation of the JWST mirror. The hexagons were created by solving the apothem (line segment starting from the center that is perpendicular to the side), along with a specified horizontal coordinate for the center of a singular hexagon shape.

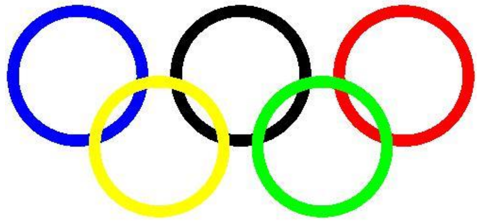
Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

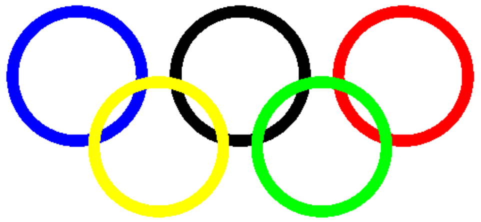
IMAGE TYPES

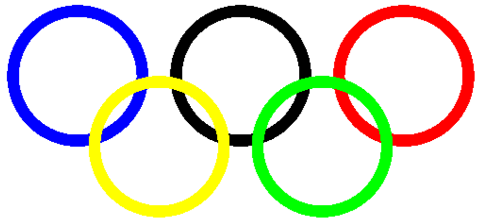

PNG file

Olympic logo

This image of the Olympic logo was created with the same idea as the annulus image. Only this time, the different colors are obtained by stacking RGB arrays, i.e., yellow color has pixel RGB value [1,1,0], and the different rings' origins were translated accordingly. The image was saved as PNG, JPG, TIFF, and BMP. BMP and TIFF files are usually large and used when the image is needed to be printed on large papers such as posters. The. PNG and JPG are the ideal image type for usual use and are the most common image file type.


JPG file


TIFF file


BMP file

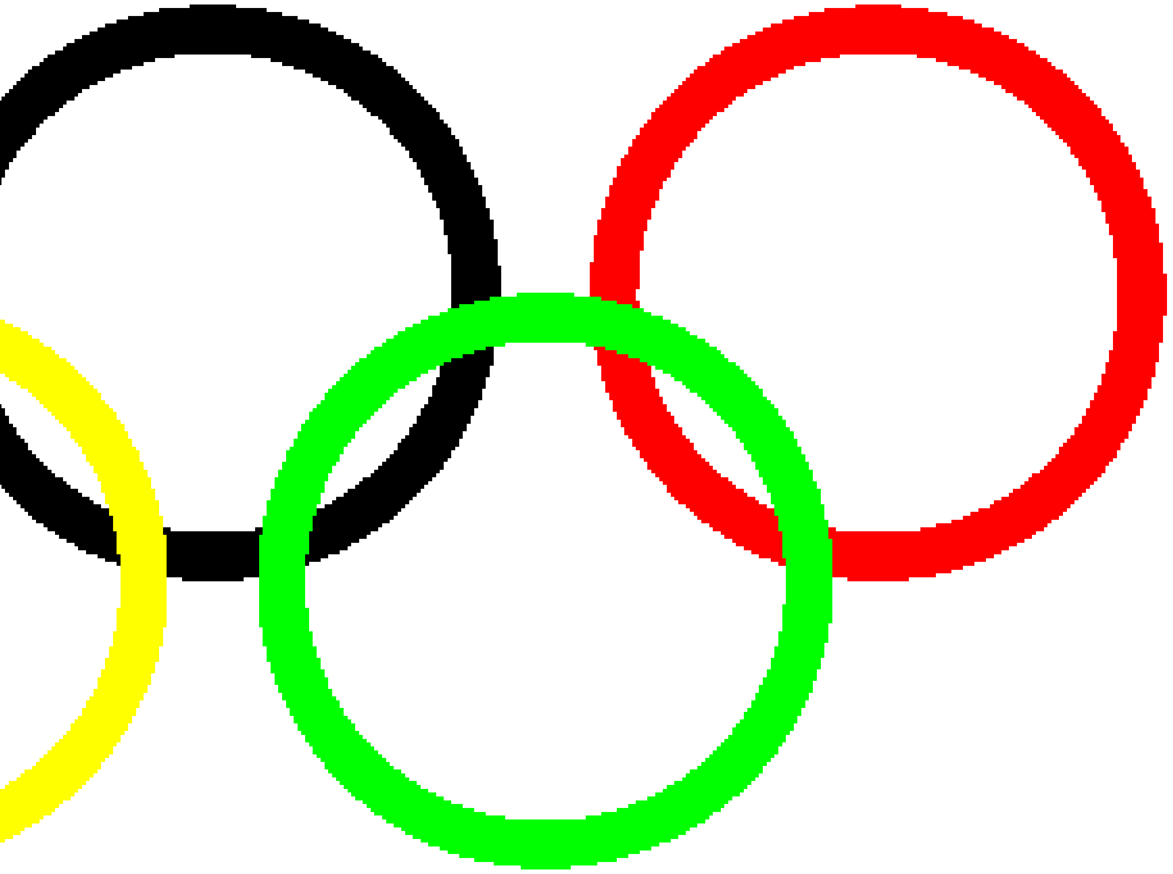
Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

IMAGE TYPES

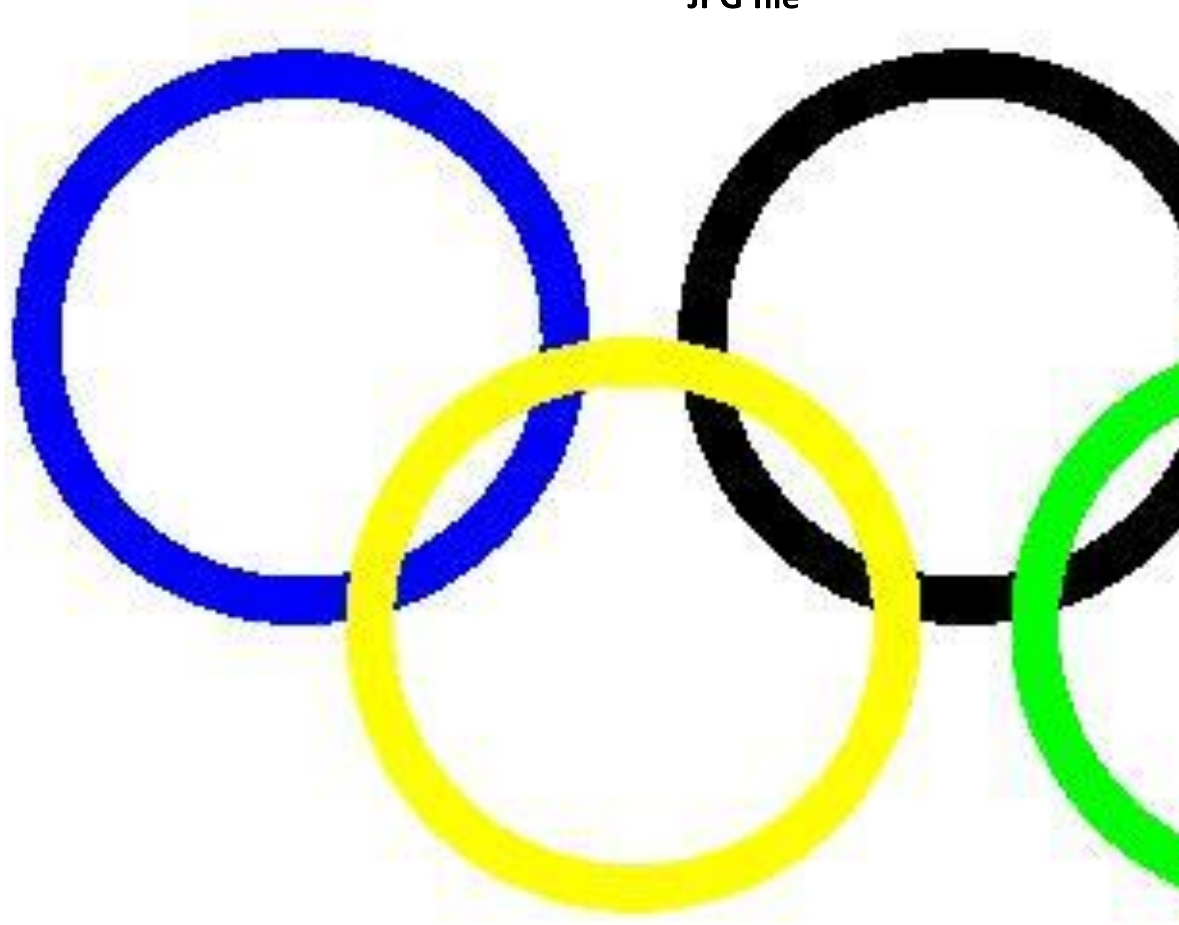
PNG VS JPG

The main difference between PNG and JPG is the way they compress the image. Yes, they both have much smaller file sizes compared to TIFF and BMP, but between PNG and JPG, JPG usually still has the lower size of the two. This is because JPG uses lossy compression, as opposed to PNG's lossless compression. As the terms suggest, JPG loses more information when compressed, hence having lower size. However, this makes the image more 'pixelated', as seen in the zoomed in version of the Olympic logo images.

PNG file

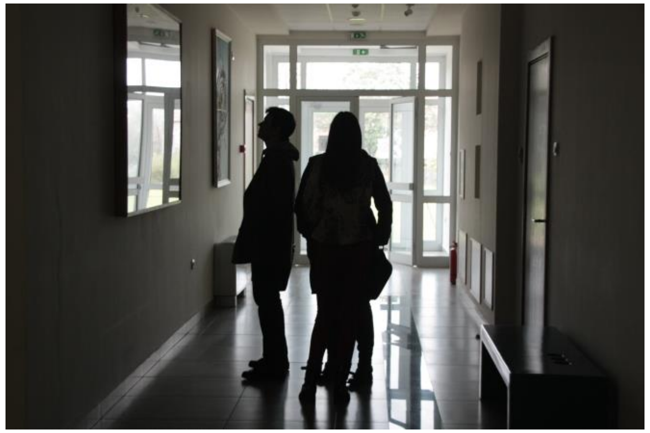


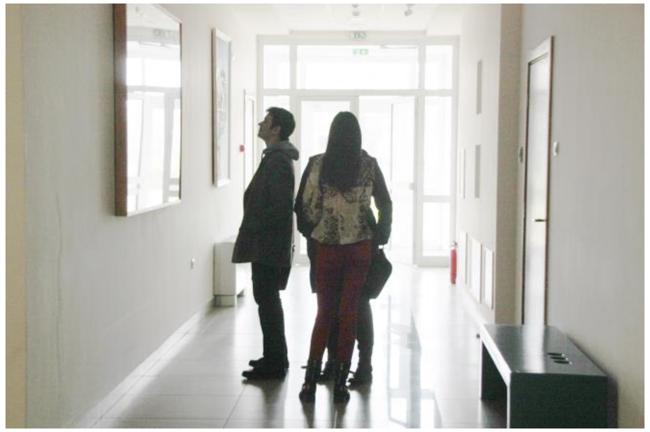
JPG file



Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

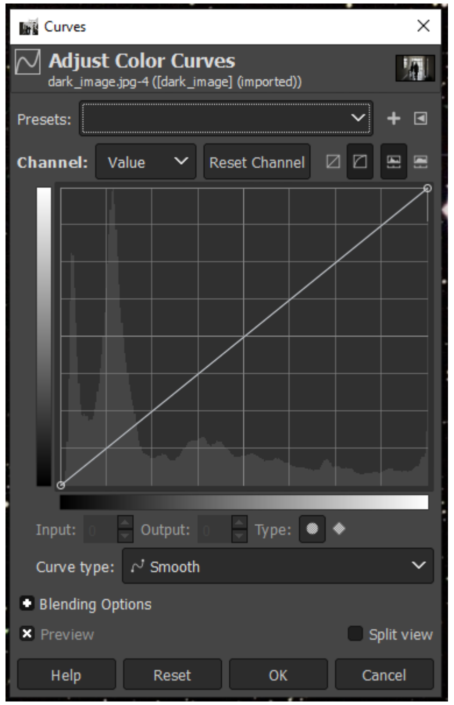
IMAGE ENHANCEMENT USING GIMP





Dark image

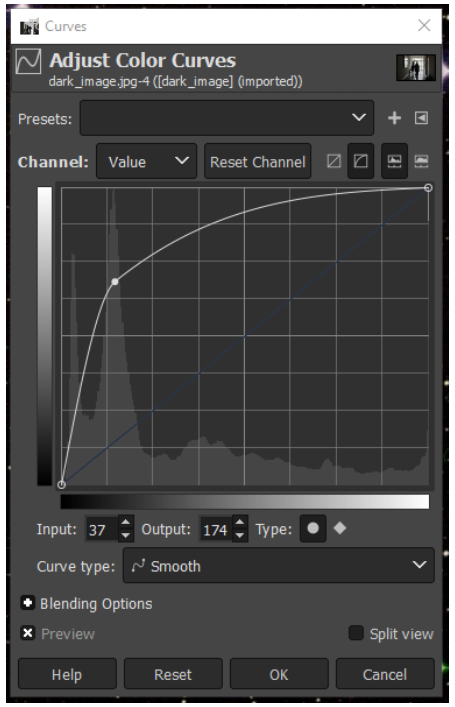
As seen from the left, the image contains silhouettes of people because of the bright light behind. It may seem that information on the person are lost, but in reality they are still encoded in the image. With GIMP, we are able to tweak its input-output curve of the image's histogram.



Channel: Value

Input: Output:

Curve type: Smooth



Channel: Value

Input: 37 Output: 174

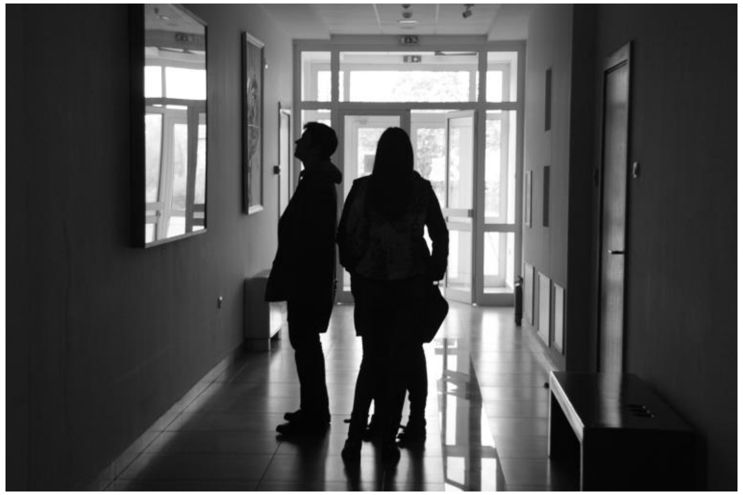
Curve type: Smooth

IO-curve

As seen from the left, the image's IO-curve does not properly scale with the peak values in the histogram, where the left side corresponding to dark pixels and the right side corresponding to bright pixels. By properly adjusting the curve and dragging it along the peak (which are the dark areas) in the histogram, we obtain an image with better information on the dark areas in the original image.

Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

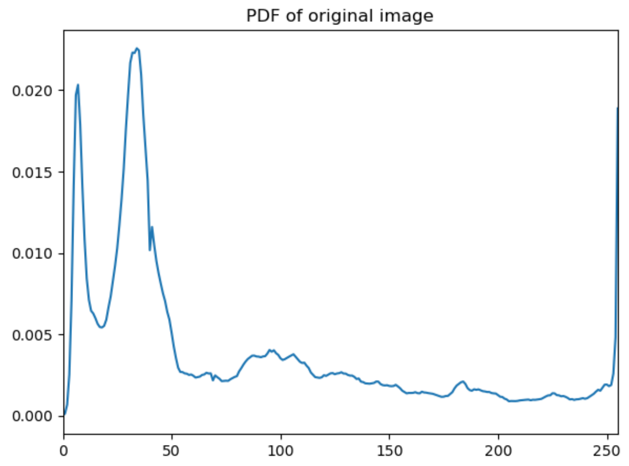
HISTOGRAM BACKPROJECTION

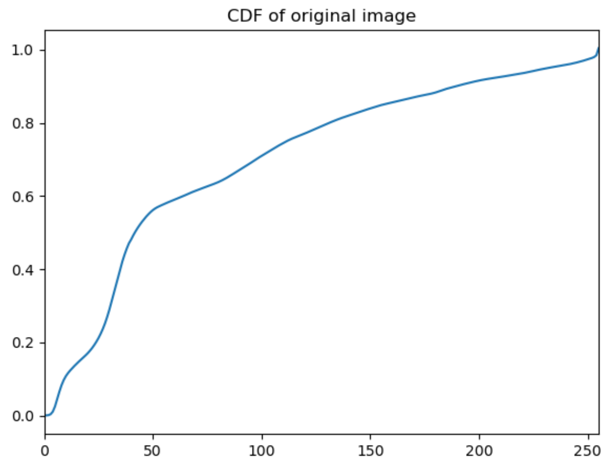


Grayscale image

In this section, the same dark image was used as a sample for doing histogram backprojection to enhance the image.

First, the image was converted to grayscale, which is shown on the left. Then, we obtain the histogram of the 'gray' values of each pixel ranging from 0 to 1, where 0 is black and 1 is white. By normalizing the histogram, we obtain the image's Probability Distribution Function (PDF), as shown below.





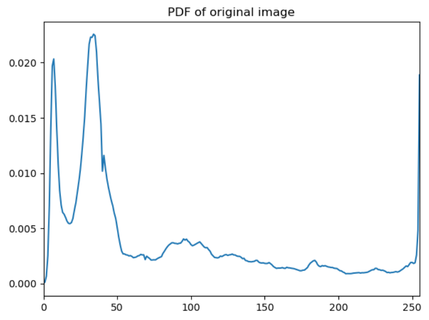
PDF and CDF

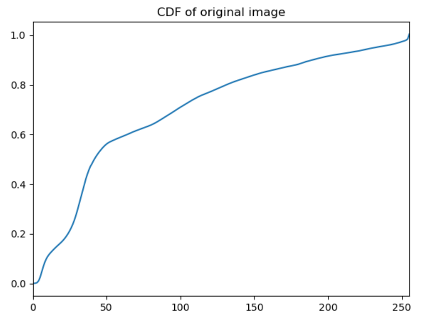
As can be seen from the GIMP IO-curve, the PDF obtained in the image is the same distribution shown in GIMP. When the value of the function includes the values previous it, then you obtain the Cumulative Distribution Function (CDF) of the image.

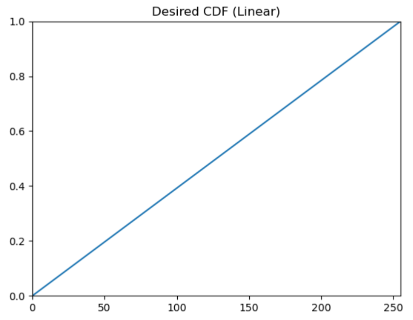
Manipulating the CDF to ideal distributions will enhance the grayscale accordingly by again scaling the curve to the appropriate target curve, as will be seen in the next parts.

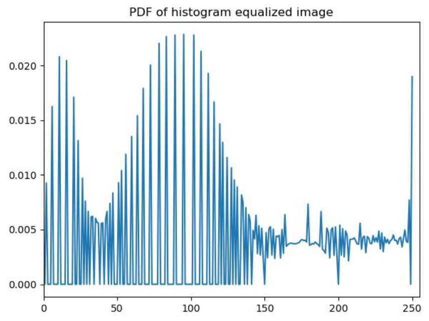
Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

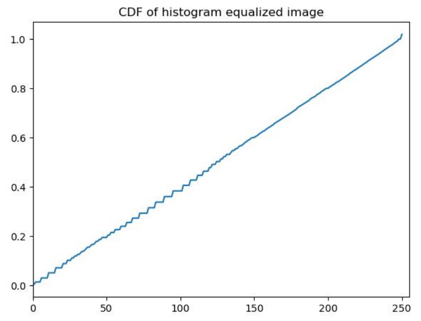
HISTOGRAM BACKPROJECTION: LINEAR CDF

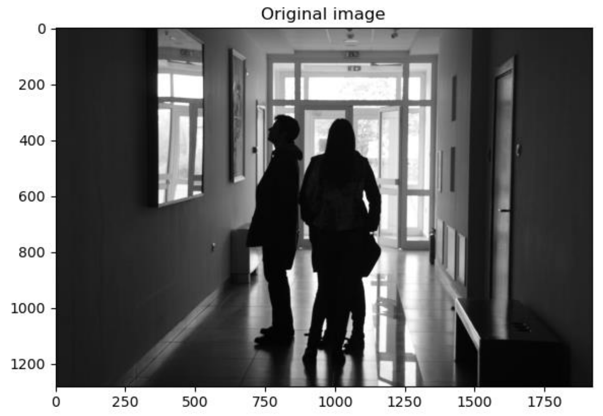


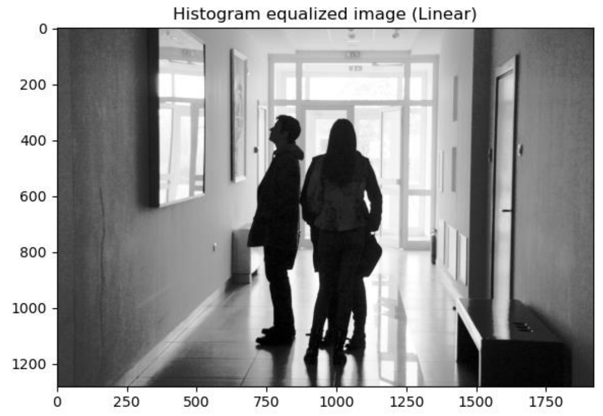












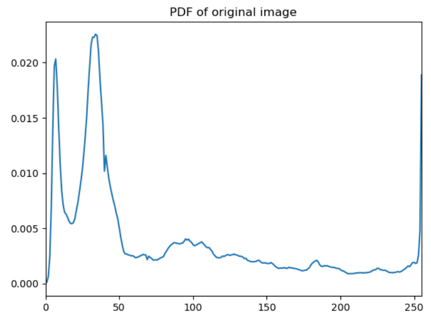
Linear desired CDF

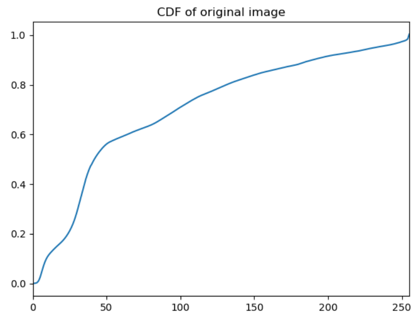
If our desired CDF is a linear function as shown on the left, then we must map the original CDF to look like it, which is called backprojection.

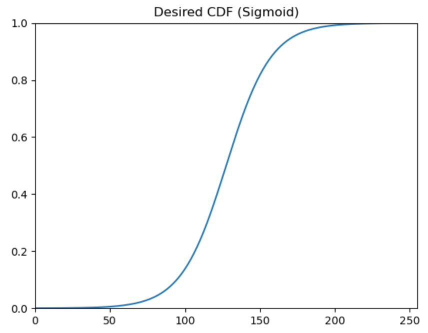
Doing so results in a rigged CDF that assumes a linear form. The resulting image is shown above. Though subtle, it can be seen that the image has been slightly enhanced, especially the dark areas which are now slightly brightened. This tells us that a linear CDF is not optimal for this image. We try another in the next part.

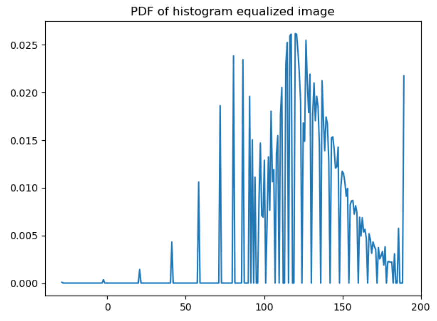
Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

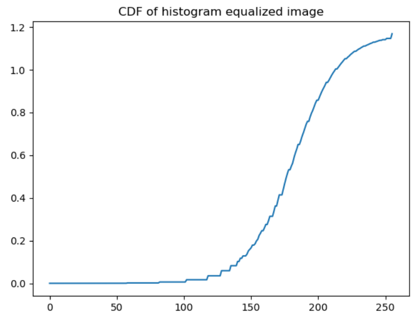
HISTOGRAM BACKPROJECTION: SIGMOID CDF

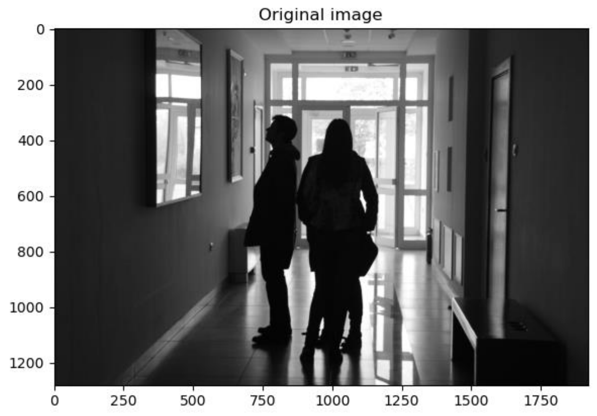


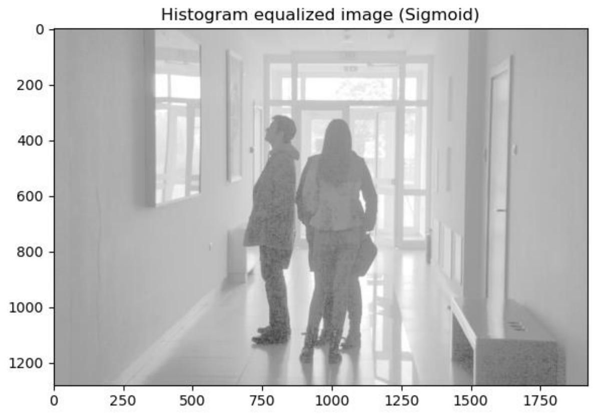












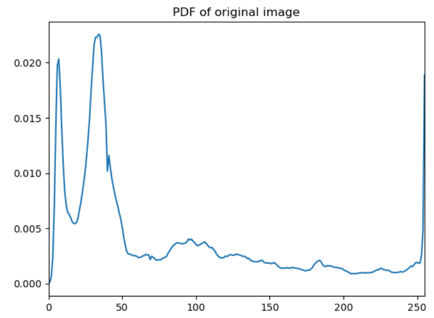
Sigmoid desired CDF
Here, our desired CDF is a sigmoid function as shown on the left. Then, we must map the original CDF to look like it, as before.

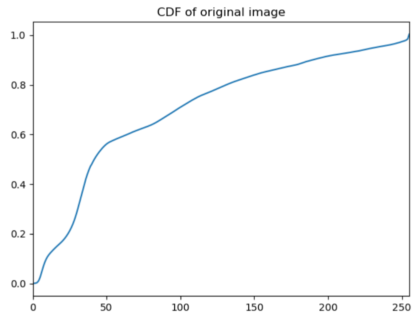
Doing so results in a rigged CDF that assumes a sigmoid form. The resulting image is shown above. It is clear that the image has been enhanced significantly, especially the dark areas which are now brightened and has more information visible.

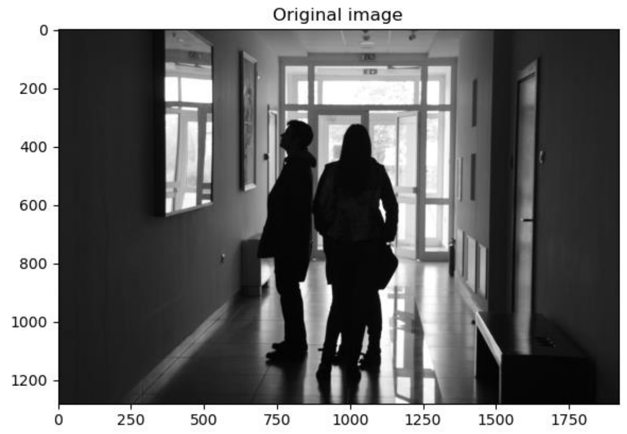
Choosing the right CDF to backproject is vital as different ones can result to different levels of enhancement.

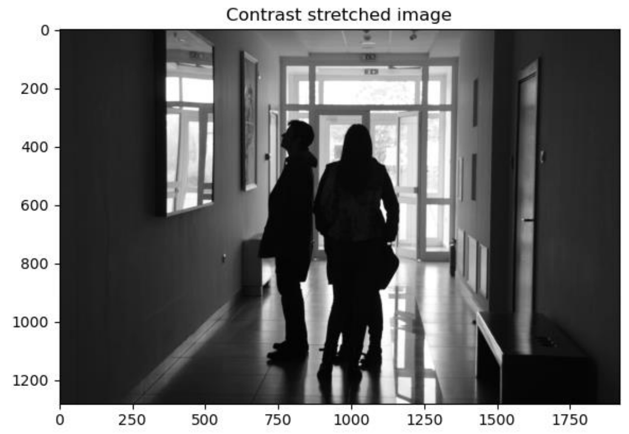
Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

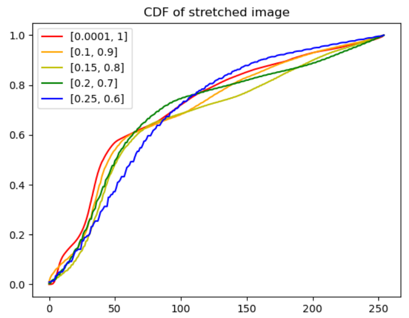
CONTRAST STRETCHING

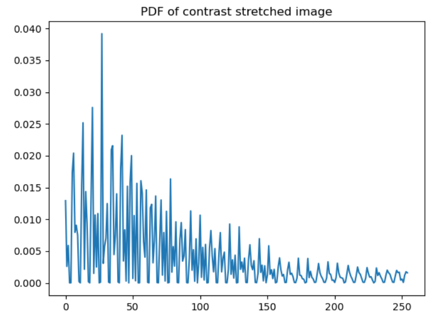


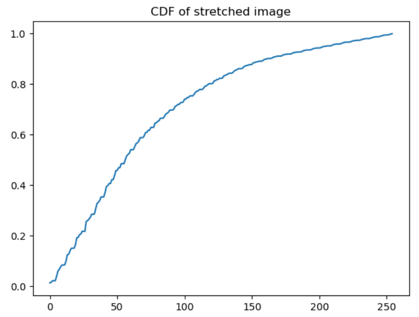












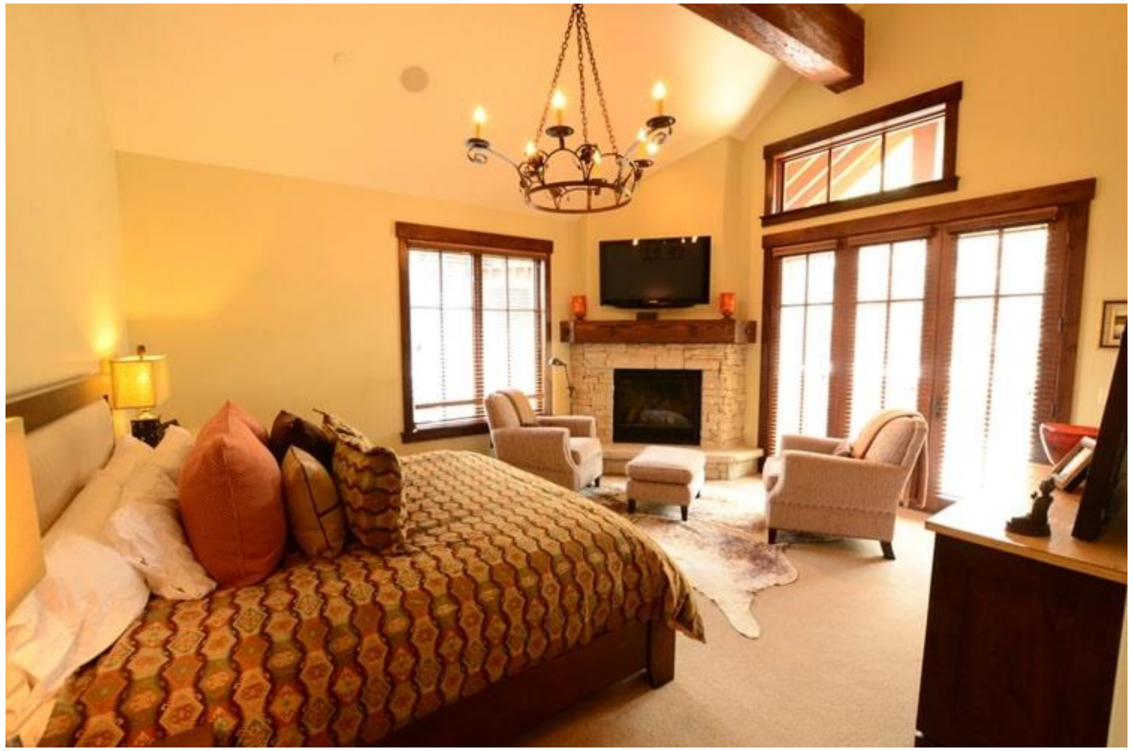
Contrast stretching is another technique to enhance an image. The idea is to manipulate the pixel value of an image based on the minimum and maximum value from all the grayscale pixels. But, if the minimum is 0 and maximum is 255 already, contrast stretching does nothing to the image.

So, choosing and appropriate percentile of the CDF as the minimum and maximum thresholds does the trick. However, this still does very little to the image as shown in the result above.

Still, we can look at the CDF of the image. Shown on the left is the CDF of the image using different percentiles of the original CDF as minimum and maximum thresholds. We see that as we “compress” the threshold, the CDF “smooths”.

Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

WHITE BALANCING



White balancing is a great tool to manipulate an image where the colors of the objects are affected by different colored light source. By white balancing the image, the colors are "corrected" and enhances the image. This can be used to restore faded or old photos, correct colors in images, and remove "color filters" on an image.

By doing white balancing, objects that are known to be white, which aren't in the image, are corrected to be white again, hence the name. There are three techniques to do this—contrast stretching, gray world algorithm, and white patching algorithm

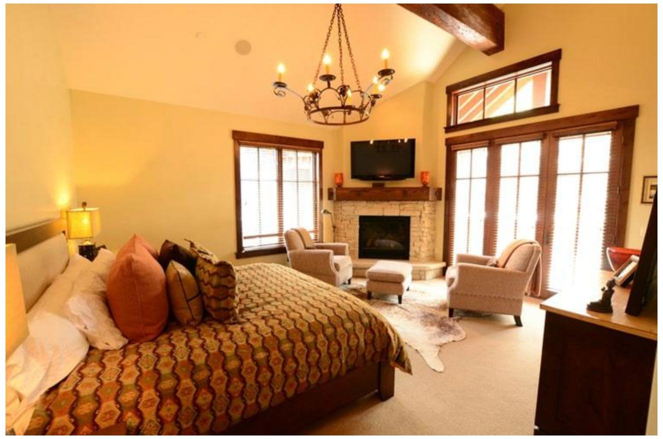
Take a look at the image on the left. Because of the chandelier's warm color, the room appears to be orange. Let us white balance the image using the three teqchniques.

Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

WHITE BALANCING

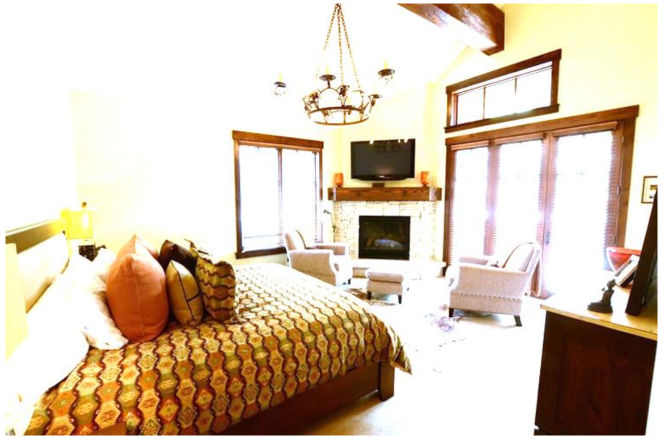
Contrast stretching
The idea here is to use the contrast stretching we did for grayscale and apply it to each RGB channel of the image.

The image below shows the result of contrast stretching. As before, we see that it has a very small effect on the image and is not very good at enhancing the image.



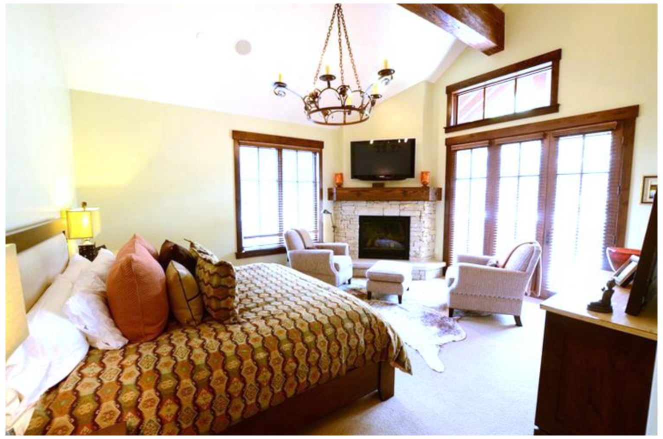
Gray world algorithm
The idea here is to treat the image as if it has a gray average. Thus, it is done by dividing each RGB channel by their average.

The image below shows the result of this technique. As can be seen, it does enhance the colors of the objects. However, it seems to overdo it and the image is now very bright. This technique works well if the assumption of the image being gray is somewhat true.



White patch algorithm
The idea here is to take a region in the image that is known to be white and divide its average from each RGB channel.

In the image, I assumed the ceiling around the upper left part of the image to be white. This technique has the best result as seen bellow. The “orange” color of the image is now gone, and the image brightness looks better.



Codes and files: https://github.com/genvert/AP_157_FX-2_Vertudez/tree/main/Activity%201

REFLECTION

It was fun to construct images mathematically. It really tested your knowledge of functions and how you can translate them to image processing. I had a significantly hard time understanding how to generate the hexagons in Python. I was trying to do them manually as I did the earlier images so that was the main reason I got stuck in this activity, and I have a bad habit of not proceeding with the activity without finishing the previous part even though they are not related. This is the main factor of why I get so long in finishing my activities.

However, this activity strengthened my foundations in image processing since here, the basic principles were covered. The ideas and techniques in this activity were useful in the succeeding activities.

SELF-GRADE

- Technical correctness: 35/35
  - I am confident that I understood the concepts behind the tasks in this activity. They were essential in the succeeding activities.
- Quality of presentation: 35/35
  - I have explained each step and idea, and images are clear and concise.
- Self-reflection: 30/30
  - Even though I was slacking in this subject, I managed to understand the fundamentals of image processing. The topics are really fun and interesting.
- Initiative: 10/10
  - I went beyond the expected output.

REFERENCES

[1] Soriano, M. (2023). AP 157 module. Digital Image Formation and Enhancement.

[2] Displaying a hexagonal grid with matplotlib. Stack Overflow. (2021, May 17). https://stackoverflow.com/questions/67563362/displaying-a-hexagonal-grid-with-matplotlib

[3] JPG vs PNG. https://www.adobe.com/ph_en/creativecloud/file-types/image/comparison/jpeg-vs-png.html

[4] Dark image source. https://www.hippopx.com/en/silhouette-shadow-dark-gallery-corridor-against-the-light-146688