DIGITAL IMAGE FORMATION AND ENHANCEMENT

Mary June A. Ricaña

SN: 2019-01551

Section: WFU-FX

Table of Contents

Side note:

All of my files may be found in this drive.

Introduction

Digital image formation using programming software allows us to create accurate images that may be used for modelling or simulations. This is especially useful in today's day and age when almost everything can be computerized.

This report consists of different imaging techniques performed using MATLAB. For the first part, we will be creating synthetic images. Next, we will tackle image enhancement, and lastly, we will try image restoration.

Objectives

The following are the goals of this activity:

- Mathematically create images.
- Save an image in an appropriate file format.
- Open and capture images using Python or Matlab
- Improve the appearance of gray level and color images
- Use the **backprojection** technique to transform the histogram of an image to a desired distribution

Creating Synthetic Images

Activity 1.1 Image DIY

For this part, we recreate some familiar forms or shapes using MATLAB. These can be easily recreated using paint, however, it would be more accurate using the software. It also gives us the convenience of manipulating the parameters and directly seeing how our actions affected the figures.

We will be creating a sinusoid, a grating, some oversized pixels on a sensor, an annulus, and a circular aperture with graded transmittance. Each image must have the dimensions X ϵ [-2 cm, 2cm], Y ϵ [-2 cm,2 cm] and the image size should be 400x400 pixels.

Sinusoid

Create a sinusoid along the x-direction with amplitude ϵ [0,1] and frequency of 4 cycles/cm.

MATLAB Code

```
% The code below generates 200 points between
% x1,y1 and x2,y2. x1,y1 and x2,y2 are set
% from -1 to 1 to account for the amplitude
nx = 200; ny = 200;
x = linspace(-1,1,nx);
y = linspace(-1,1,ny);
% Generated 2D grid coordinates based on x and y
[X,Y] = meshgrid(x,y);
f = 4; %frequency of the wave
Z = sin(2*pi*f*X);
mesh(Z);
```

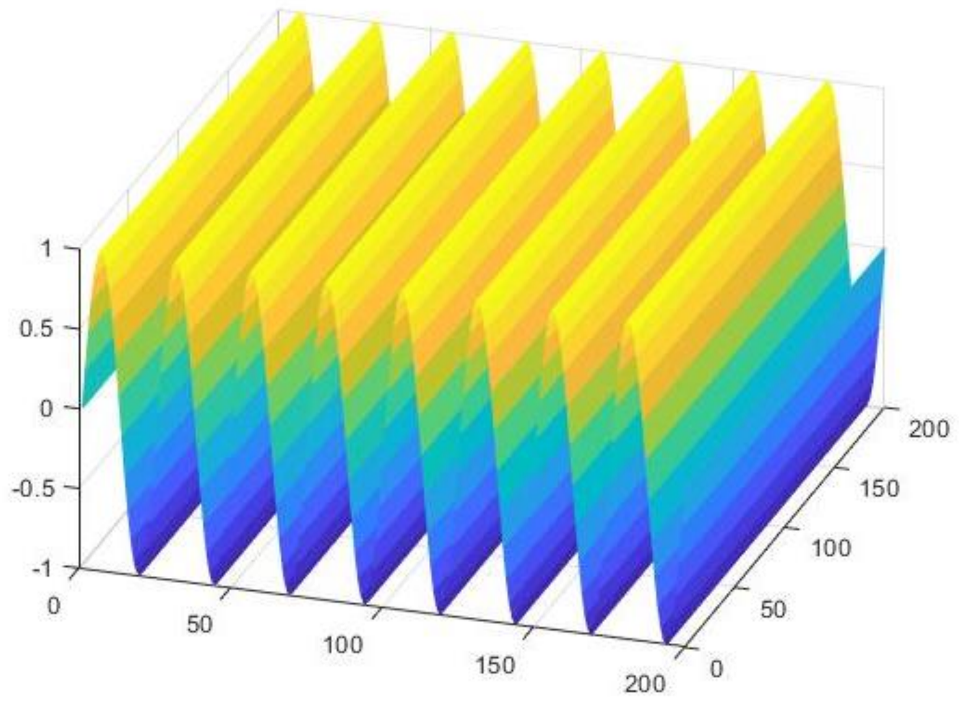


Figure 1. Synthetic Image of a 2D sinusoid along the x-direction, with amplitude ϵ [0,1] and frequency at 4 cycles/cm generated using MATLAB.

Analysis

The general formula for a sine function is given by

$$y = A\sin(2\pi f x + B)$$

where A is the amplitude, f is the frequency and B is the phase. The amplitude dictates the height of the wave, which can be observed in Figure 1 to be 1 cm. Furthermore, since we set the frequency to be 4 cycles per cm, and our image is 2 cm by 2 cm in size, we can see 8 cycles in the figure.

We can also generate other functions like the cosine or tan functions via MATLAB:

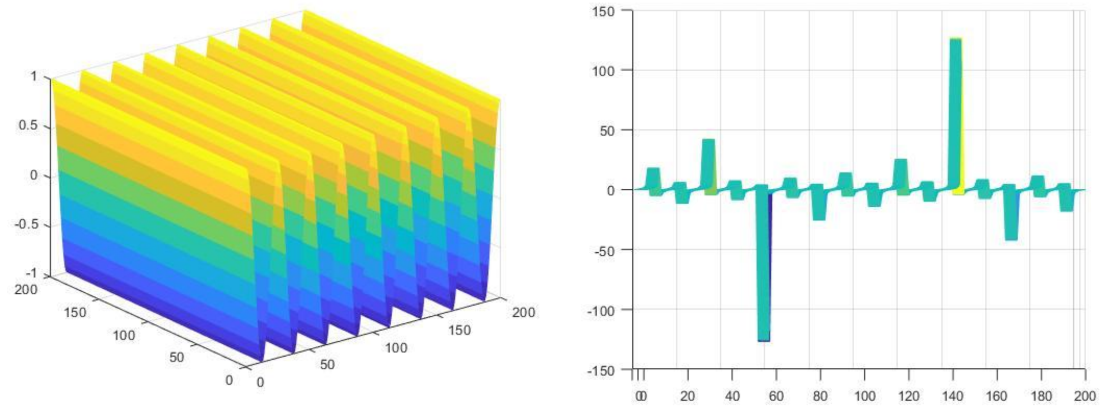


Figure 2. Synthetic Image of a 2D cosine and tan wave

Many systems in physics are sinusoidal in motion like the simple harmonic motion. Learning how to create these images will allow us to understand their behavior more.

Grating

Create a grating with frequency of 5 line pairs/cm. A line pair is a strip of black (0) and white (1)

MATLAB Code

```
frequency = 5; %frequency is 5 line pairs/cm
phase = 0;
amplitude = 1;
% Image size is 2 cm by 2 cm
[X,Y] = meshgrid(-2:0.001:2,-2:0.001:2);
% We generate the grating by using the top view of a sinusoid
Z = amplitude * sign(sin((2*pi*frequency.*X) + (phase)));
mesh(X,Y,Z);
shading interp % To vary the color of each line segment
view(0,90)
colormap gray % To make the colors black and white
```

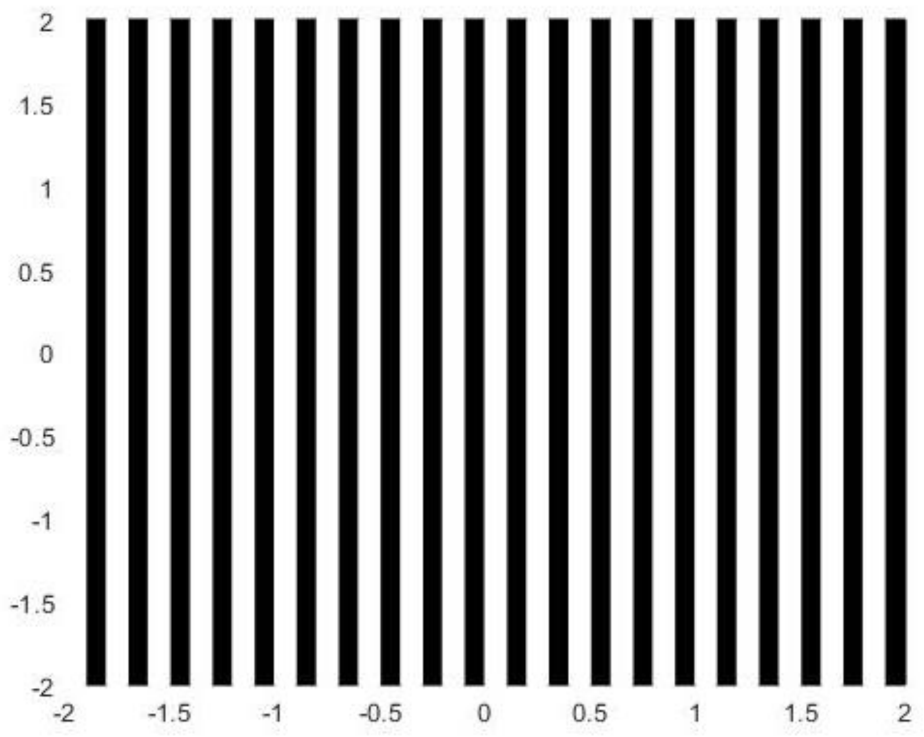


Figure 3. Synthetic image of a grating with frequency at 5 line pairs/cm, made by manipulating the view of a sinusoid (top view).

Analysis

We utilize the same formula as the sine function in the previous part in generating the gratings. We just manipulated the view and the colors of the plot to generate our desired image (top view, and black and white gratings). Through this, we created evenly spaced identical lines or a block of rectangle. Changing the frequency of the equation would change the number of line pairs.

Gratings are very useful in physics, especially in optics, as we see it being used for studying the spectra of light, as in diffraction gratings. This also reminds me of a discussion we had in Physics 165 on the Moire pattern by overlaying two gratings with one tilted at a different angle.

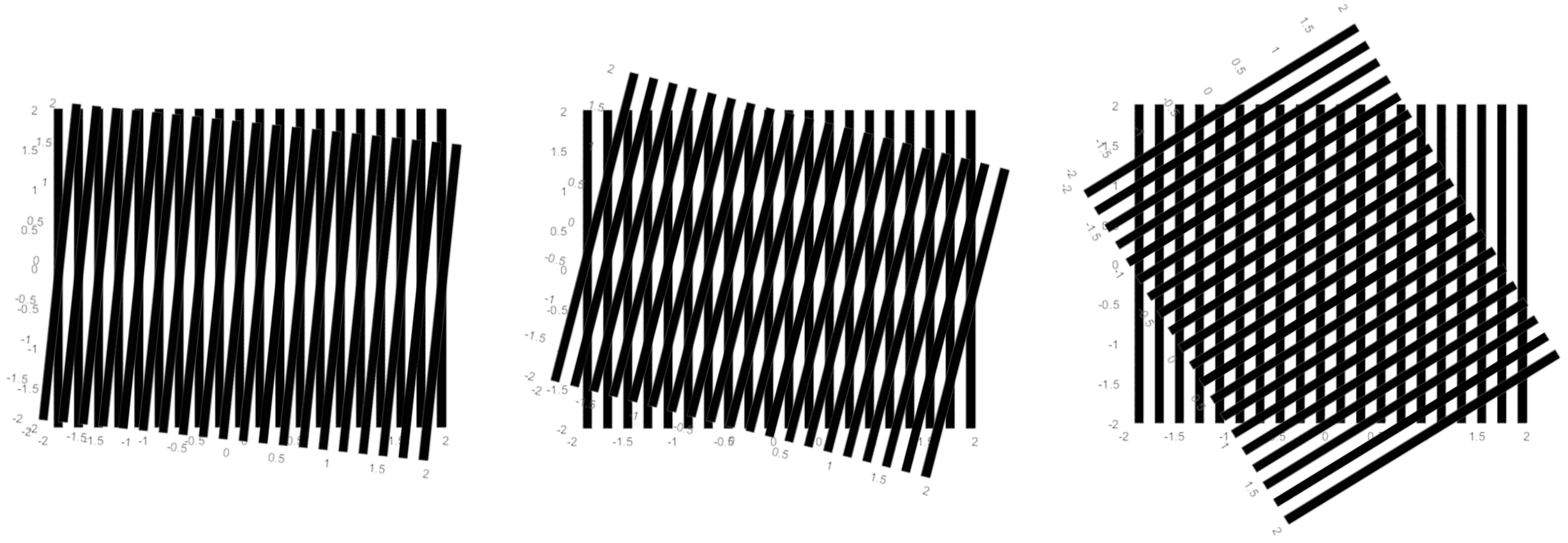


Figure 3. Moire pattern generated by overlaying two images of the gratings in Figure 2.

Oversized Pixels

Create oversized pixels on a sensor. Each pixel is 5mm x 5mm and there’s a 1mm gap between each pixel

MATLAB Code

```
% We set the axes to be from -20 to +20 mm for the image size
for x = [-20:6:20]
for y = [-20:6:20]
% x+1 and y+1 for the 1 mm gap per pixel and
% 5 by 5 mm for the pixel size
rectangle('Position', [x+1 y+1 5 5]);
axis([-20 20 -20 20])
end
end
```

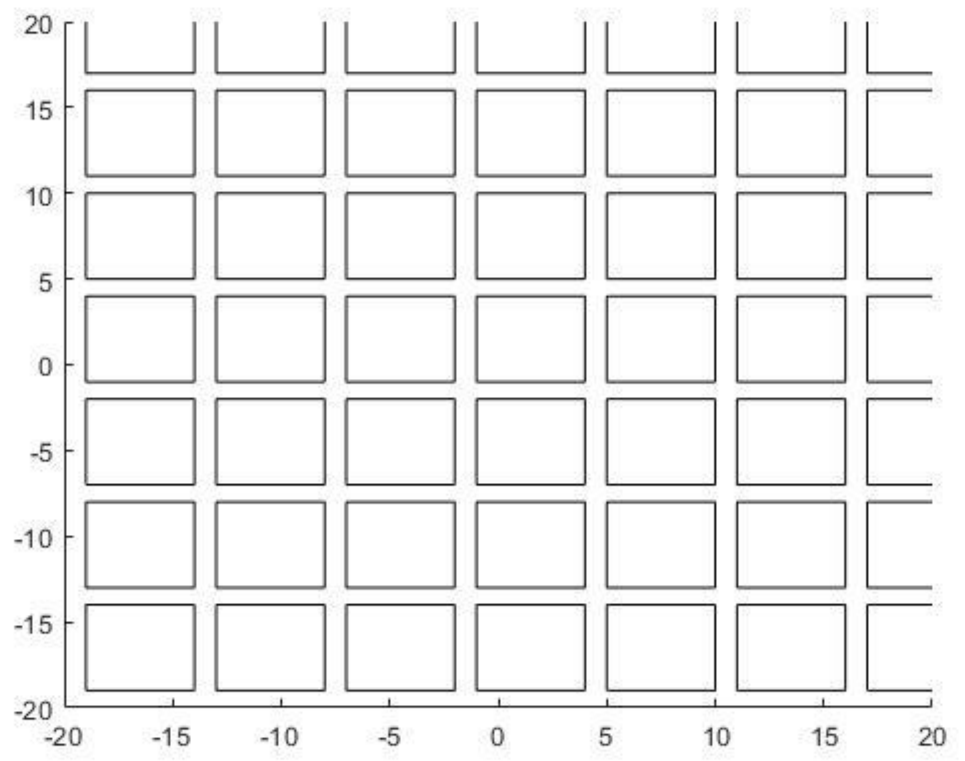


Figure 4. Synthetic image of oversized pixels on a sensor. Each pixel is 5mm x 5mm and there’s a 1 mm gap between each pixel.

Circular Annulus

Create an annulus with an outer radius of 2 cm and with thickness of 0.25cm.

MATLAB Code

```
% Make the annulus and set the size of the radii
t = linspace(0, 2*pi);
rout = 2;
rin = 1.75;
xin =  rin*cos(t);
xout = rout*cos(t);
yin =  rin*sin(t);
yout = rout*sin(t);
% We use patch to plot the inner and outer rings
patch([xin,xout],[yin,yout],'w','linestyle','none','facealpha', 1);
% These are just to customize the plot appearance
axis equal
ax = gca;
ax.Color = 'black';
xlim tight;
ylim tight;
```

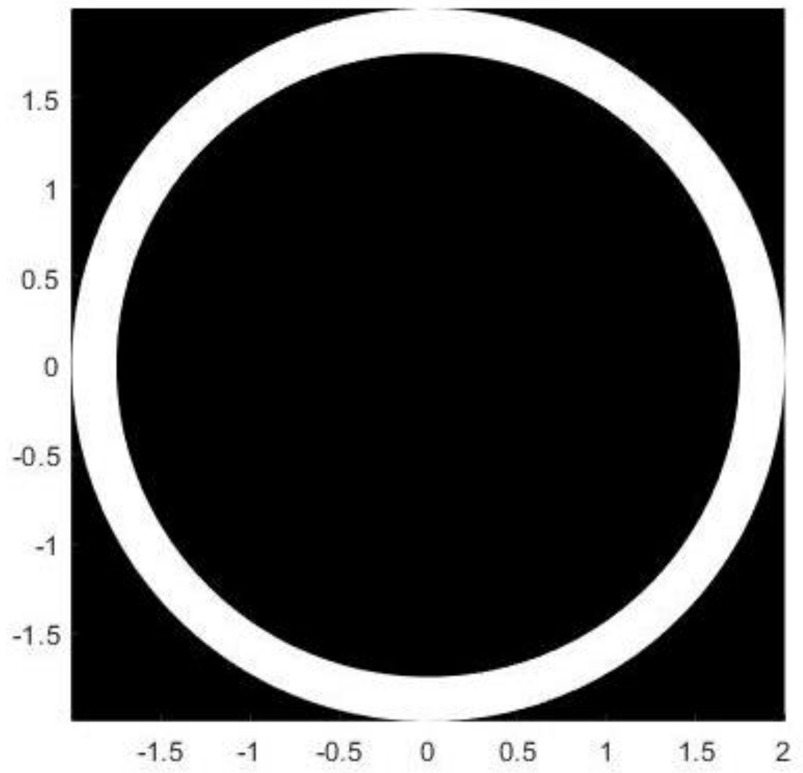


Figure 5. Synthetic image of a circular annulus with an outer radius of 2 cm and with thickness of 0.25 cm.

Analysis

Figure 4 is a synthetic image of oversized pixels on a sensor. The code is straightforward and we just made use of MATLAB's rectangle function to generate the pixels. We also used the for loop to generate as many rectangles that would fit our 2 by 2 cm image.

Meanwhile, for Figure 5, we made use of MATLAB's patch to synthesize the inner and outer radii of the annulus into one plot. We just used the equation for a circle to generate the rings. In the image, I set the area between the two rings to be transparent and the areas outside the outer radius and inside the inner radius to be black.

Both of the images generated are useful in modeling and understanding real life objects. For example, learning how to generate an image of pixels on a sensor would aid in making camera sensors and the likes. Meanwhile, there are many real life things in the form of a circular annulus like a ring or even pipes where water will pass through.

Circular Aperture w/ graded transmittance

Create a circular aperture with graded transmittance, a zero centered Gaussian profile, R = 1.75 cm, and s = 1cm.

MATLAB Code

```
% Set up the circular aperture
x=linspace(-2,2,1000);
y=x;
[X,Y]=meshgrid(x,y);
omega = 1;
R = sqrt(X.^2 + Y.^2);
% Equation for the Gaussian profile
z=exp(-(X.^2+Y.^2)/(2* omega^2));
z(find(R > 1.75)) = 0; %#ok<FNDSB>
figure(1); imshow(z);
figure(2); mesh(x,y,z); colormap(jet);
```

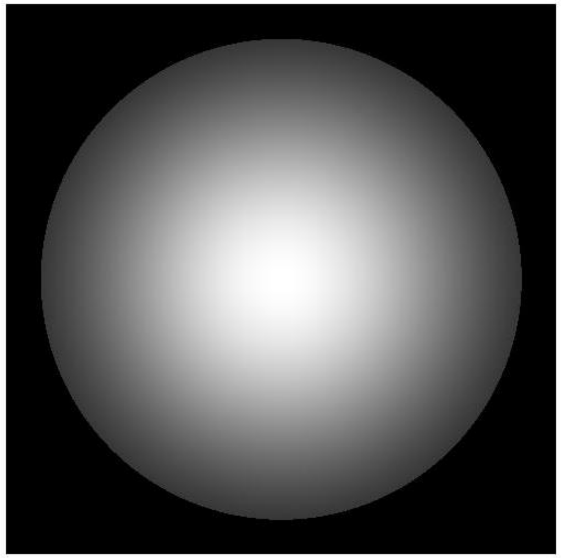
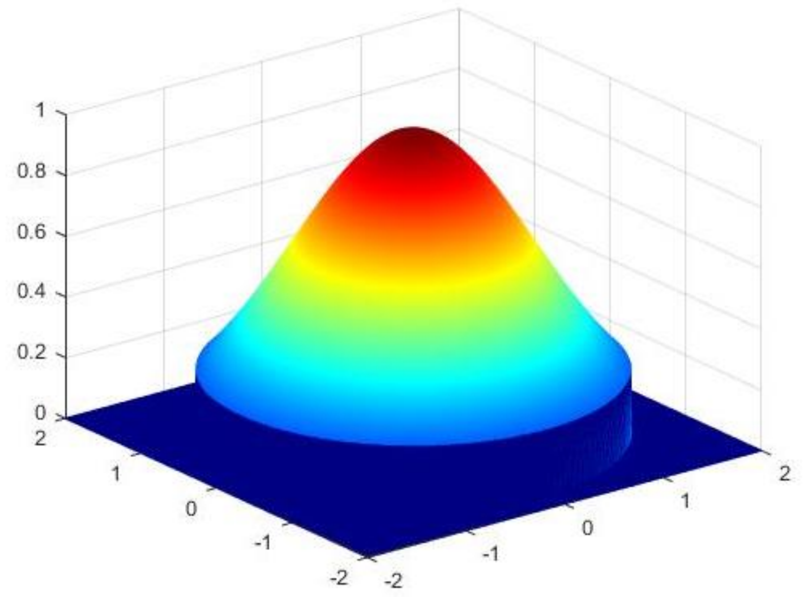


Figure 6. Synthetic image of a circular aperture with graded transmittance, a zero-centered Gaussian profile, R = 1.75 cm, and s = 1cm (left: 2D and right: 3D



Analysis

The general formula for a Gaussian function is given by

$$f(x,y) = A\exp\left(-\frac{x^2+y^2}{2\Omega^2}\right)$$

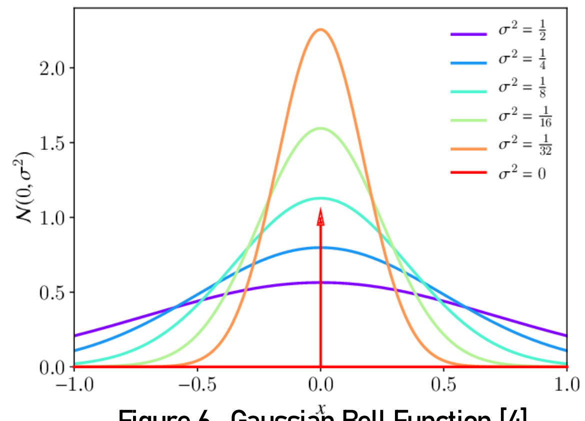
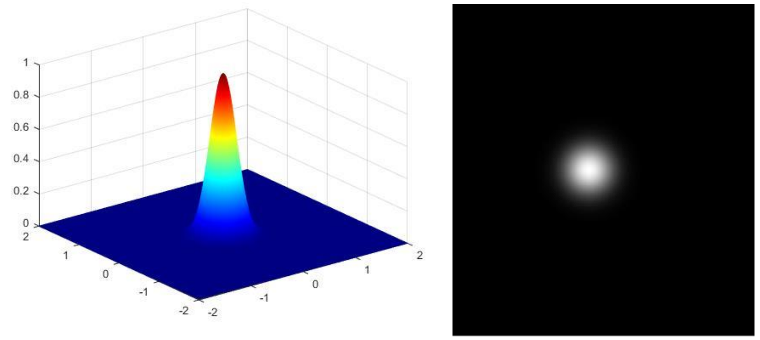
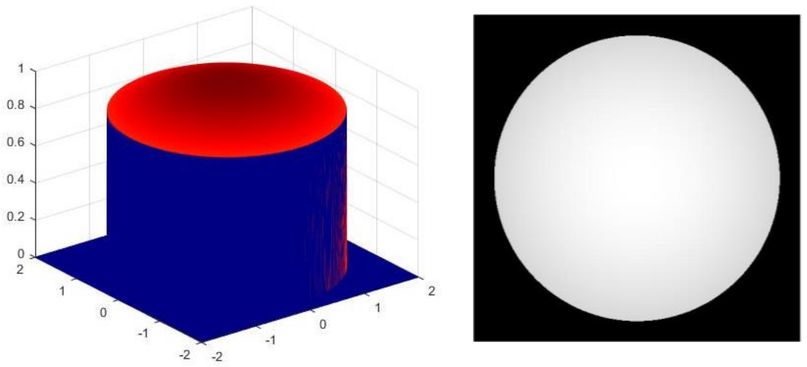


Figure 6. Gaussian Bell Function [4]

where A is the height of the curve's peak and the sigma dictates the variation surrounding the mean. This means that when the sigma is large, then the variation from the mean value would be greater and the plot would appear wider, while if the sigma is small, the curve will be concentrated in the mean value. We model this behavior by changing the sigma of our circular aperture to 0.2, 1, and 3. We can see that the curves generated reflect the behavior of the Gaussian curve.



a. Sigma = 0.2



b. Sigma = 3

Figure 7. Synthetic image of a circular aperture with graded transmittance, a zero-centered Gaussian profile, R = 1.75 cm.

Creating Synthetic Images

Activity 1.2 Color Image

For this part, we combine our previous learnings in creating an image mathematically, and add some challenge by including colors and overlapping layers. In this activity, the aim is to recreate the Olympics logo and save the generated image in different file formats using MATLAB.

Recreate the Olympics logo (MATLAB Code)

```
% Set up the rings and their positions
N = 1000;
angle = linspace(pi/4, 9*pi/4,N);

xb = cos(angle) * 0.9;
yb = sin(angle) * 0.9;
% To customze the colors of the rings, we turn their
% hexcode to a 1 by 3 RGB array
strb = '#0085C7';
cb = sscanf(strb(2:end),'%2x%2x%2x',[1 3])/255;

xy = cos(angle) * 0.9 + 1;
yy = sin(angle) * 0.9 - 1;
stry = '#F4C300';
cy = sscanf(stry(2:end),'%2x%2x%2x',[1 3])/255;

xk = cos(angle) * 0.9 + 2;
yk = sin(angle) * 0.9;
strk = '#000000';
ck = sscanf(strk(2:end),'%2x%2x%2x',[1 3])/255;

xg = cos(angle) * 0.9 + 3;
yg = sin(angle) * 0.9 - 1;
strg = '#009F3D';
cg = sscanf(strg(2:end),'%2x%2x%2x',[1 3])/255;

xr = cos(angle) * 0.9 + 4;
yr = sin(angle) * 0.9;
strr = '#DF0024';
```

```
cr = sscanf(strr(2:end),'%2x%2x%2x',[1 3])/255;

h1 = figure;
hold on
% Plot the rings
plot(xb(1:3*N/4),yb(1:3*N/4),'Color', cb,'linewidth',5);
plot(xy(N/4:N),yy(N/4:N),'Color', cy,'linewidth',5)

plot(xk(1:3*N/4),yk(1:3*N/4),'Color', ck,'linewidth',5);
plot(xy(1:N/4),yy(1:N/4),'Color', cy,'linewidth',5);
plot(xb(3*N/4:end),yb(3*N/4:end),'Color', cb,'linewidth',5);

plot(xr(1:N/2),yr(1:N/2),'Color', cr,'linewidth',5);
plot(xg(1:N),yg(1:N),'Color', cg,'linewidth',5);

plot(xk(3*N/4:N),yk(3*N/4:N),'Color', ck,'linewidth',5);
plot(xr(N/2:N),yr(N/2:N),'Color', cr,'linewidth',5);

% Customize the axis
axis equal
axis off
xlim([-1.2 5.2])
set(h1,'Color',[1 1 1])
hold off
% Save the images
saveas(h1, 'olympicrings.jpg');
saveas(h1, 'olympicrings.bmp');
saveas(h1, 'olympicrings.png');
saveas(h1, 'olympicrings.tif');
```

My code is inspired by Michael Katz's code in MathWorks [2]

Recreate the Olympics logo

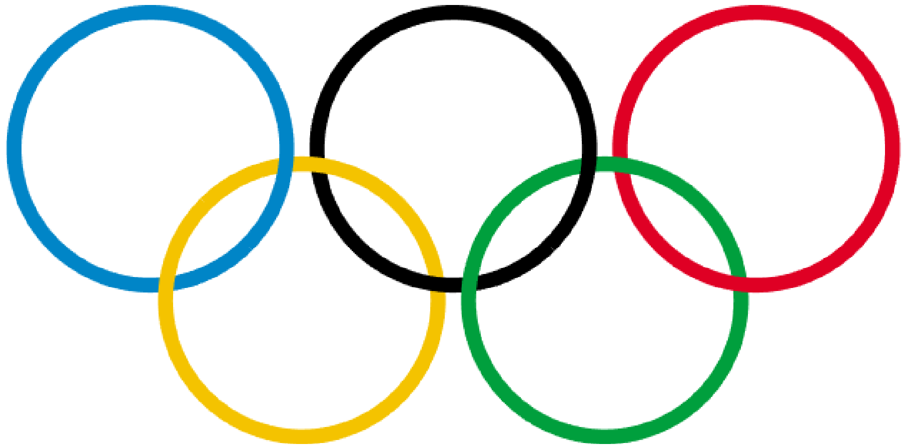
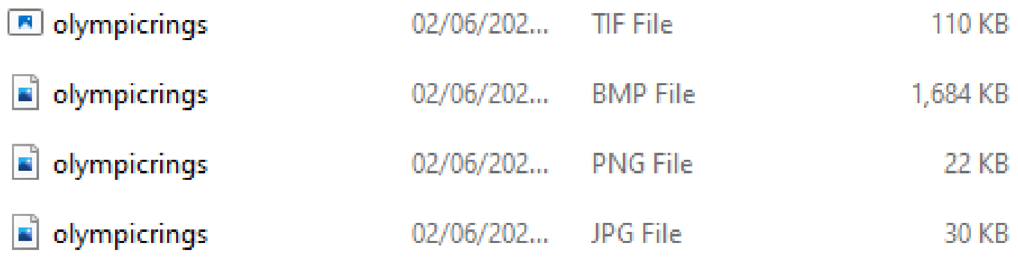


Figure 8. Recreated logo of the Olympic rings based on using MATLAB based on the code of Michael Katz on MathWorks.

Saved the Olympics logo image into different file formats: PNG, TIF, BMP, and JPG using MATLAB.



olympicrings	02/06/202...	TIF File	110 KB
olympicrings	02/06/202...	BMP File	1,684 KB
olympicrings	02/06/202...	PNG File	22 KB
olympicrings	02/06/202...	JPG File	30 KB

Discussion

The Olympics logo consists of five circles of the same sizes but of different positions. To recreate it using MATLAB, we generate the five circles and plot it in different locations and with different colors. The plot for the circles is similar to our process for the circular annulus. Meanwhile, to customize the color of each ring, we take the hexadecimal code of each color and turn them into a 1 by 3 RGB array.

We also saved the images into different file formats using MATLAB's saveas function. We generated an image in the PNG, JPG, TIF, and BIMP formats.

Below are their differences [3]:

JPG – a JPG format compresses the size of the photo, which means that it will be more considerate for your storage but the quality will also be decreased. This is best for web images since it will not be difficult to load.

Discussion

PNG – a PNG format has a lossless compression, meaning that its size and quality will not be reduced. This is useful in photography portfolios where the photo needs to be exact.

BMP – BMP or bitmap is just a combination of pixels or small dots to create an image. This is very useful in editing because it retains the details of individual pixels.

TIFF – TIFF does not compress the images thus, its original size and quality will be kept intact. This is useful for graphic designers and people working with high resolution images.

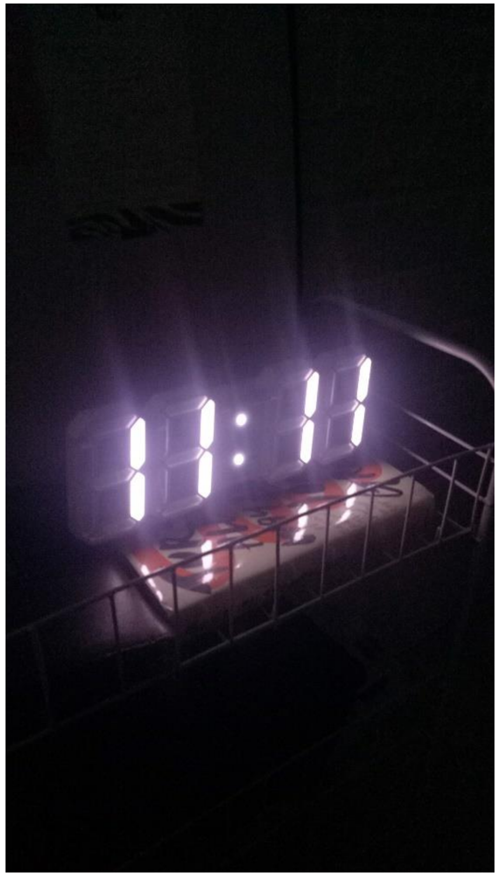
Both TIFF and BMP are “wrapper” formats, which means that they can house compressed or uncompressed images. In the screenshot of the details of the olympicrings file, it can be seen that the TIFF and BMP files are significantly larger in size than the JPEG and PNG files, with the BMP reaching around 1600 KB.

Image Enhancement

Activity 1.3 Altering the Input-Output Curve

Cameras can only capture so much. It has limitations on the details of the photos that it takes, especially in making out details to a dark image. In this case, we tap on image editors to enhance our photos and make it more detailed. In this part, we enhance a photo using the software GIMP. We take a dark photo and play with its input-output curves to obtain the most detailed photo we can get.

Altering the Input-Output Curve



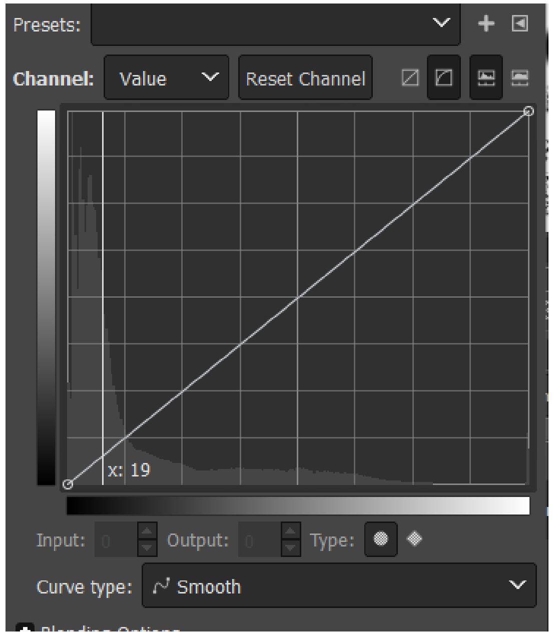
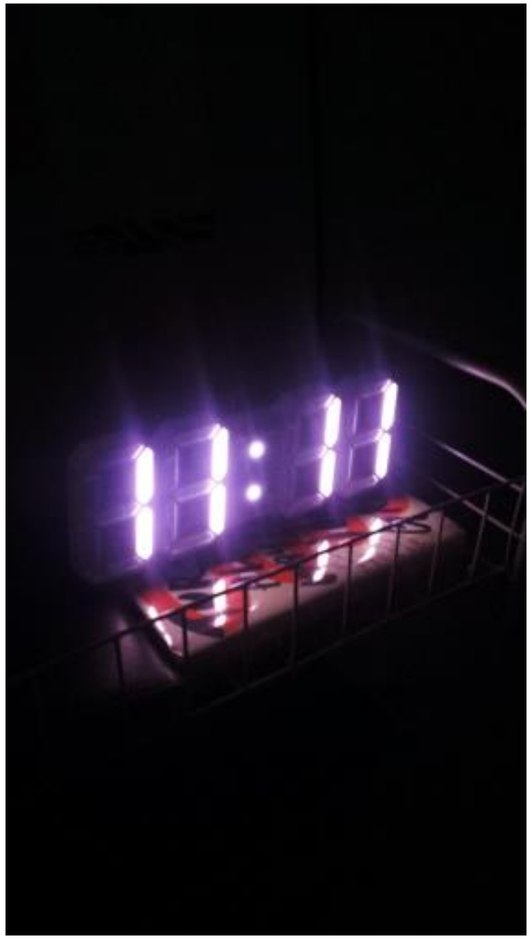


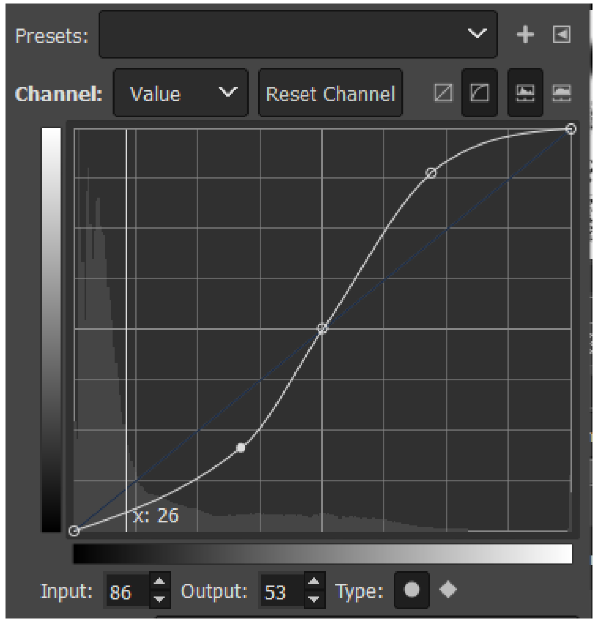
Figure 9. Original dark image and its I-O curve taken from GIMP.

The input-output curve of an image corresponds to its tonal range, which means that it controls the shadows and the highlights of an image.

In Figure 9, we see a dark photo and its I-O curve. For now, the curve is a diagonal line since this is still the raw, unedited photo. Because of the darkness of an image, the only things that can be observed are the LED digital clock, the book below it, and part of the shelf in which is it located.

Altering the Input-Output Curve





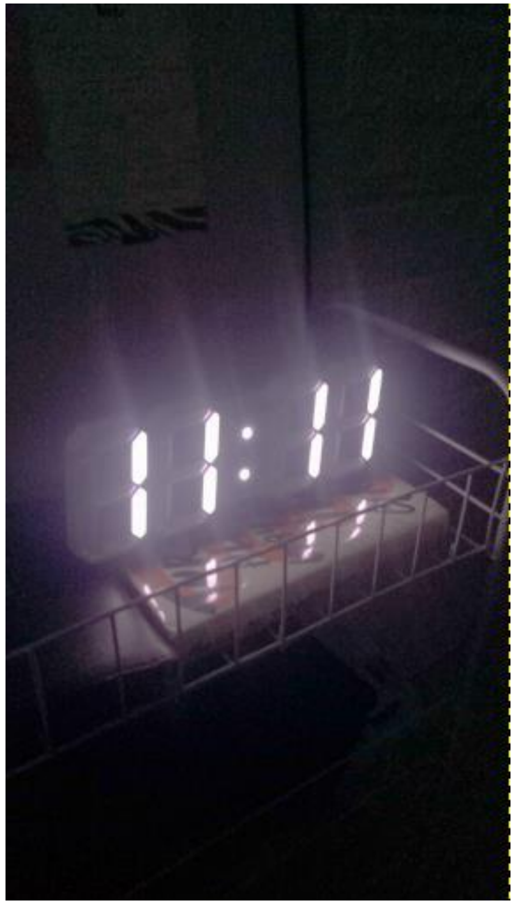
Input: 86 Output: 53

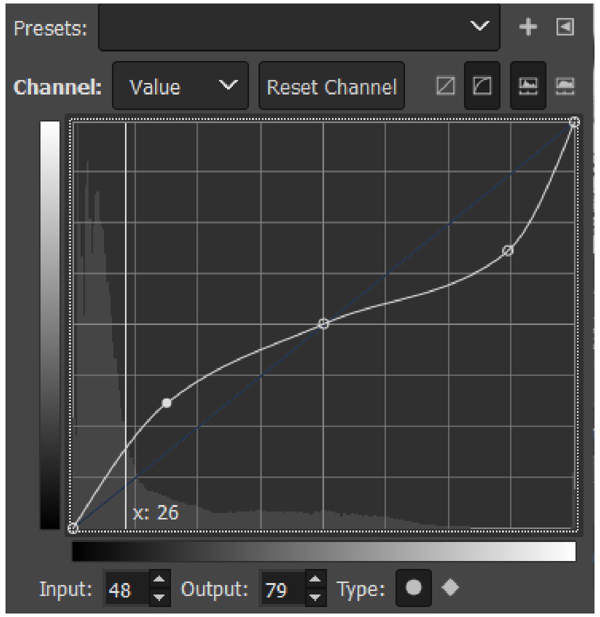
Figure 10. Adjusted dark image and its I-O curve taken from GIMP. The input is 86 and the output is 53.

In this figure, we set the shadows to be darker and the highlights to be brighter, thus we create a more contrasted and vibrant photo.

Its brightest parts, which is shown in the LED clock appears more purple and more defined. Meanwhile, we do not see any of the details in the background anymore since the dark parts are made darker.

Altering the Input-Output Curve





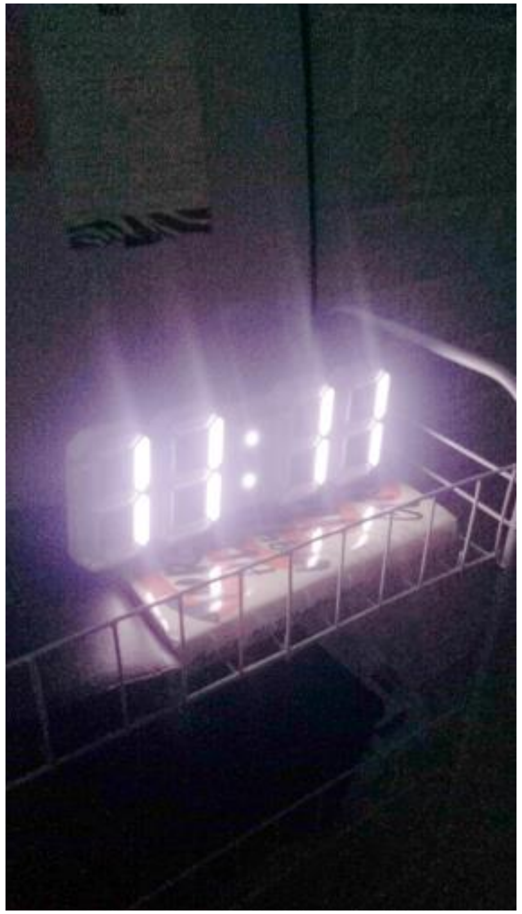
Input: 48 Output: 79

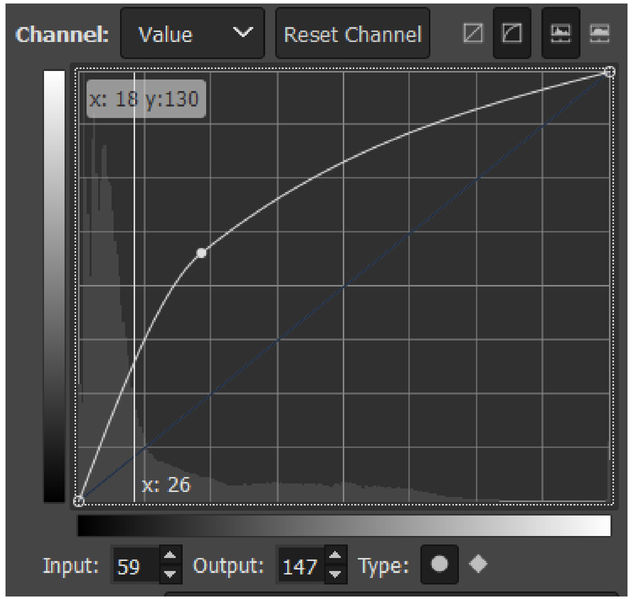
Figure 11. Adjusted dark image and its I-O curve taken from GIMP. The input is 48 and the output is 79.

We try to reverse the action in the previous figure and we decrease the highlights and we make the shadows brighter.

Here, we can observe a photo with less contrast that it almost appears to be in grayscale. We can see that the clock is not as vibrant anymore. Additionally, we can see that there are details below the clock and at the back of it.

Altering the Input-Output Curve





Input: 59 Output: 147

Figure 13. Adjusted dark image and its I-O curve taken from GIMP. The input is 59 and the output is 147.

In this photo, we adjust the curve to be above the diagonal for both the shadows and highlights. We can see that the vibrancy of the LED light is still there as compared to Figure 11, but the details of the photo can be observed.

Analysis

Playing with the input-output curve allows us to adjust the image into the desired outcome. If we want to increase the contrast, we make the shadows darker and the highlights brighter. Meanwhile, we do the reverse if we want to decrease the contrast.

In our dark image, we found that the most satisfactory adjustment is made when both the shadows and the highlights are adjusted above the diagonal. This helps us make out the details of the image as seen in Figure 14.

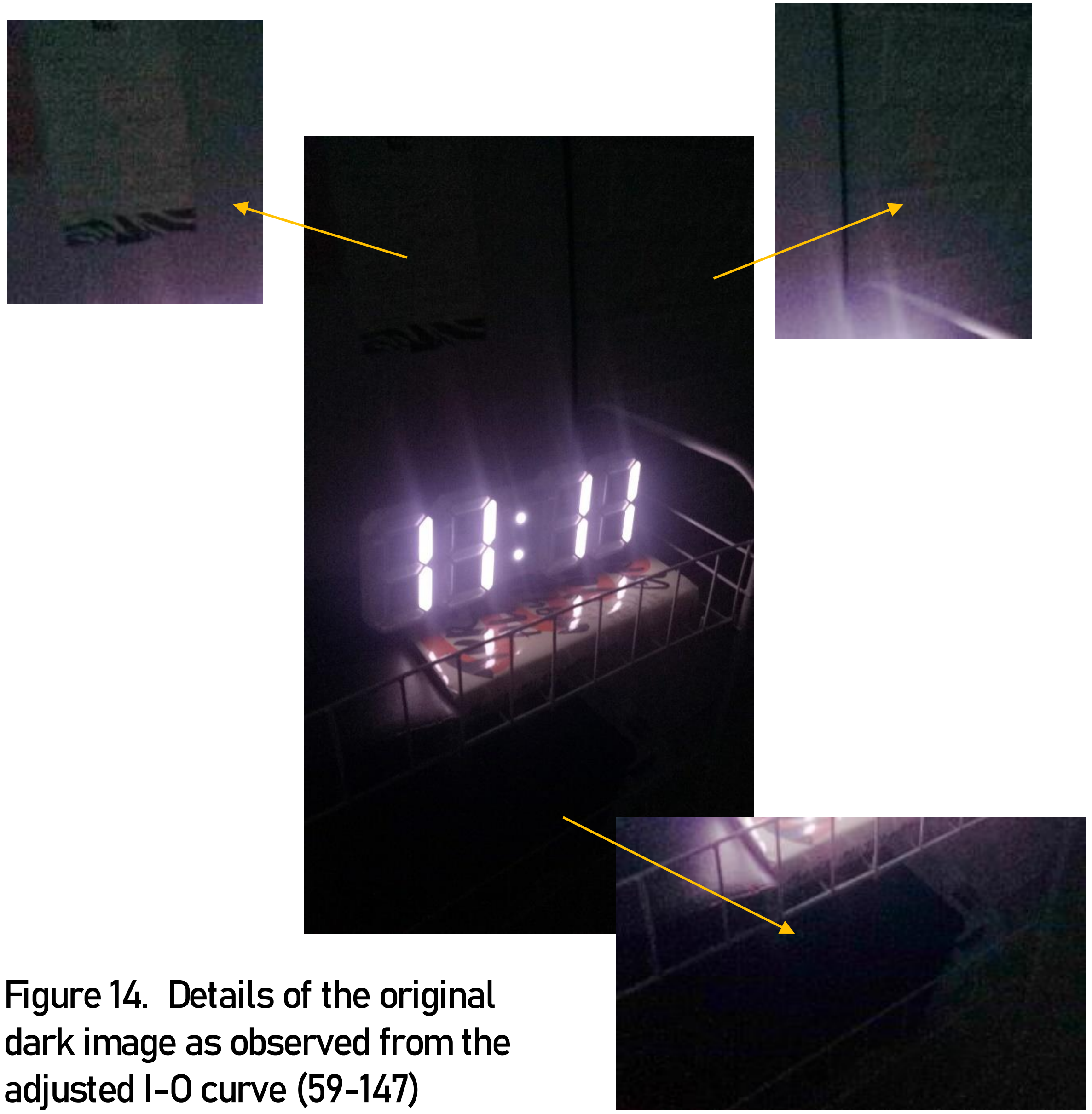


Figure 14. Details of the original dark image as observed from the adjusted I-O curve (59-147)

Image Enhancement

Activity 1.4 Histogram Backprojection on Grayscale Images

A histogram of an image can be viewed as a probability distribution function (PDF) that we can alter to follow a certain distribution to obtain the image that we want. In this part, we use backprojection to enhance the grayscale image of the dark image in Activity 1.5.

Image Enhancement

Activity 1.4 Histogram Backprojection on Grayscale Images

We first obtain the gray levels r of an image with PDF given by $p_1(r)$ and a cumulative distribution function $T(r) = \int_0^r p_1(g)dg$ where g is a dummy variable. Next, we will manipulate the image so that it would have a CDF in the form of

$$G(z) = \int_0^z p_2(t)dt$$

where $p_2(z)$ is the PDF of the transformed image. We can also get the z values by letting $G(z) = T(r)$ and this would give us

$$z = G^{-1}T(r)$$

Histogram Backprojection on Grayscale Images

MATLAB Code

```
% Open your dark-looking image in Matlab convert to grayscale.
image=imread('darkpicture.jpg');
igray=rgb2gray(image);
imshow(igray);

% Obtain the grayscale histogram of your image
[counts, grayLevels] = imhist(image);
figure;bar(grayLevels, counts, 1); title('Histogram');
hold on;
```

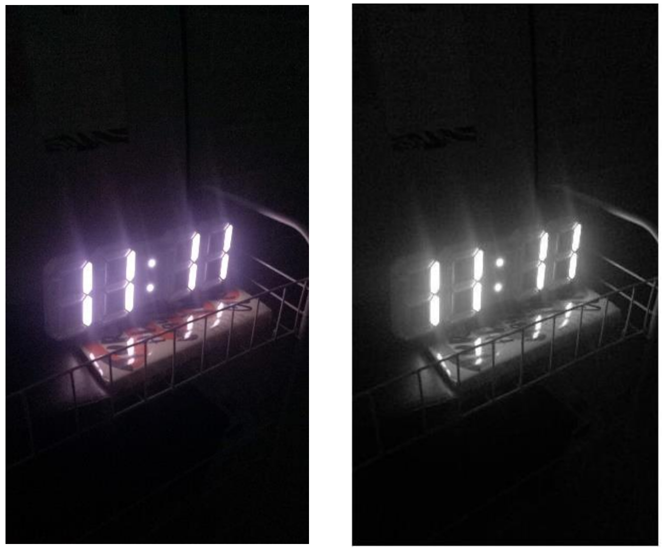


Figure 15. Dark image and the adjusted grayscale counterpart edited using MATLAB.

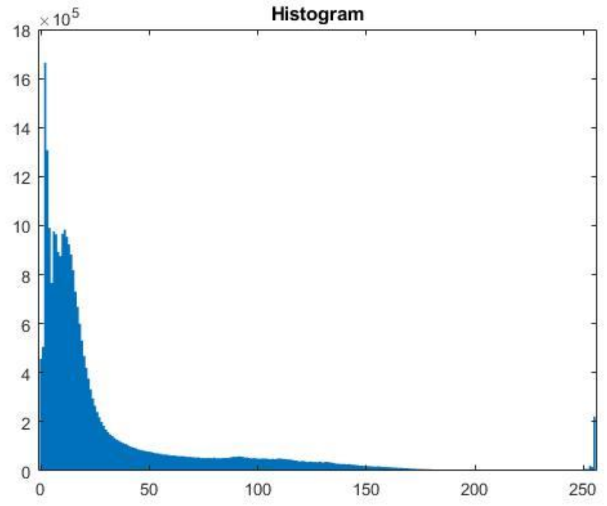


Figure 16. Histogram of the dark image.

From the histogram, we can see that the majority of the pixels are dark, which can be expected since we purposely used a dark image. We can also see that the smallest x value is at 0, while there is a spike at x = 255 to account for the brightest pixels in the image.

Histogram Backprojection on Grayscale Images

MATLAB Code

```
% Normalize by the number of pixels to get its PDF.
% Compute the CDF from the PDF.
PDF = hist(igray(:),[0:255])/numel(igray);
CDF = cumsum(PDF);
figure; plot((0:255), CDF); title('CDF');
```

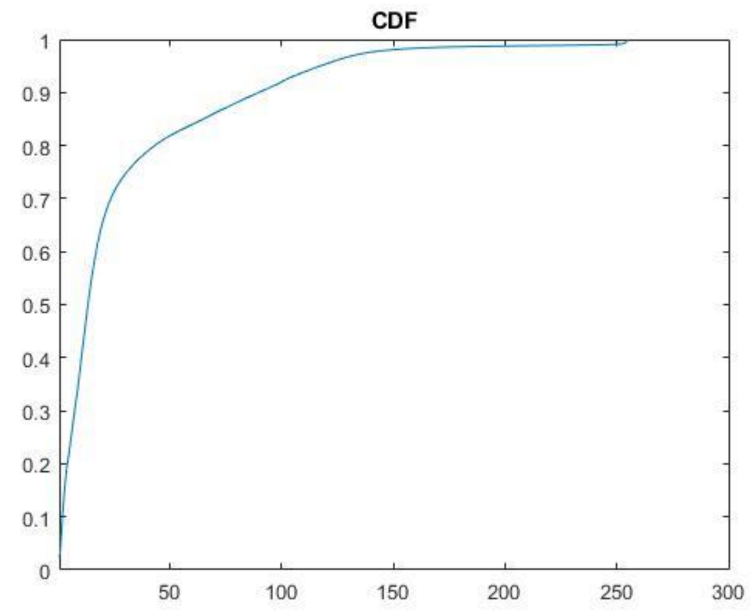


Figure 17. CDF the dark image.

The CDF or the cumulative distribution function shows the fraction of pixels at a certain intensity. For example, in the figure on the left, we see that the range of about 50 to 255 all belongs to the 80th percentile, while the majority of the pixels are below 50.

Understanding the CDF will also allow us to work with backprojection and contrast stretching easier as it will serve as a guide in choosing the minimum and maximum values of our pixel intensities.

Histogram Backprojection on Grayscale Images

MATLAB Code

```
%Use histogram backproject ion to pixel-per-pixel backproject the image
%pixel values by finding its corresponding y-value in the desired CDF.
x = [0:255];
desiredCDF = (1/255)*x;
newGS = interp1(desiredCDF, x, CDF(igray(:)+1));
Igraynew = reshape(newGS, size(igray));
Iold = figure; imshow(uint8(Igraynew));
hold on;
saveas(Iold, 'Iold.jpg');
```

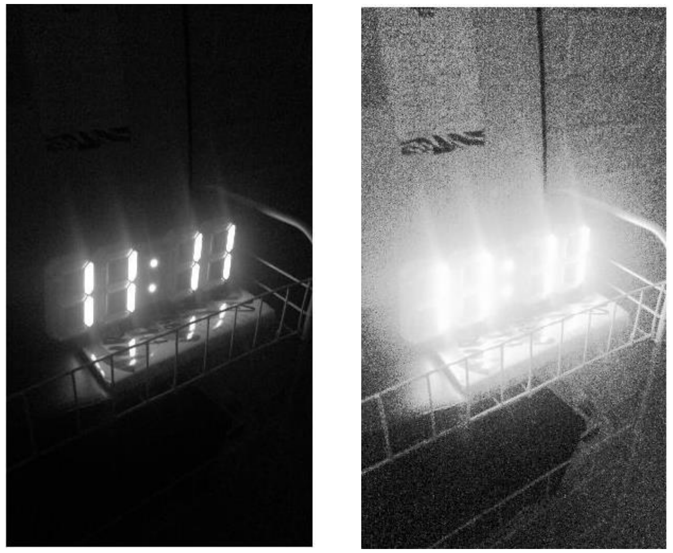


Figure 17. The grayscale of the dark image and the image obtained through backprojection.

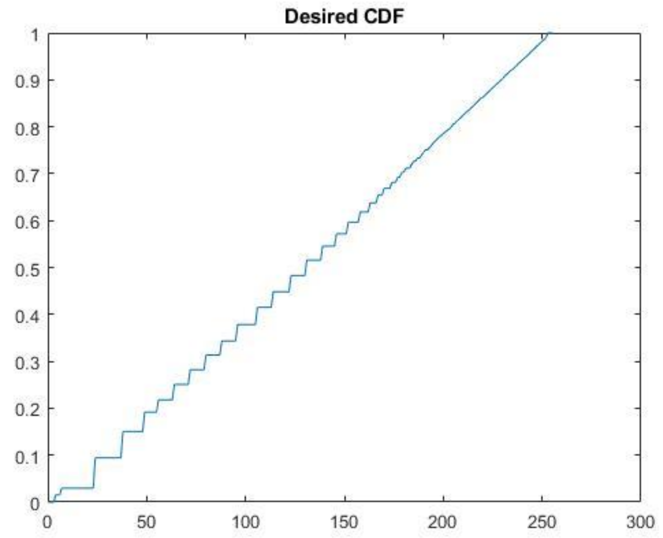


Figure 18. Desired CDF the dark image.

In order to find our desired CDF, we interpolate a linear function with our CDF in the previous slide. Applying the CDF above to our image, we get a more enhanced photo, where we can see the details much clearly. This is because now, the intensities of the pixels are linearized rather than logarithmic.

Backprojection VS I-O Curve Manipulation

Both processes aim to display more details of an image, which cannot be seen due to the photo lighting or the image quality. In GIMP, we tweaked the Input-Output curve of the image until it was to out satisfaction. In doing so, we see the details of the image behind and below the clock. For backprojection, we tweaked the CDF of the photo to be linear so that more details can be exposed.

The main difference between the two processes is that GIMP, of course, is already an editing software, while for the backprojection, we needed to code the process. For GIMP, we can tweak the image while it is colorized, while for backprojection in MATLAB, we had to analyze the image using its grayscale values to simplify the algorithm and make our computations more economical.

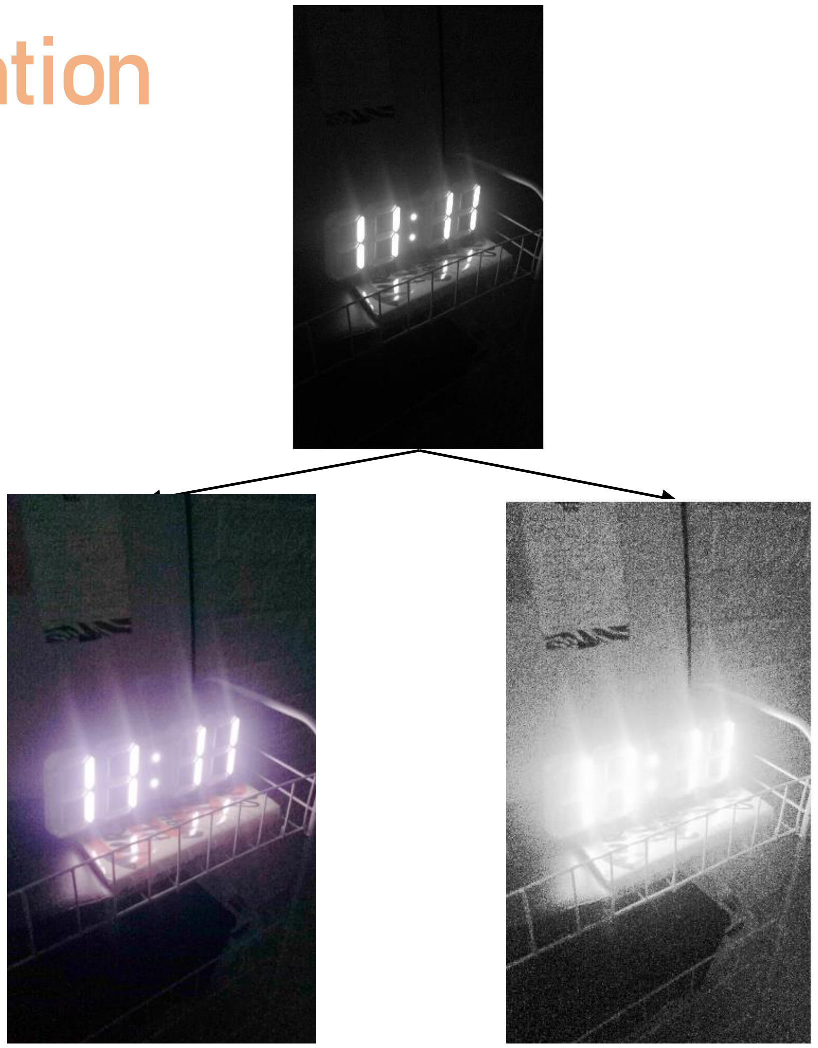


Figure 19. Comparison between backprojection (right) and manipulation of I-O curve (right)

Image Enhancement
Activity 1.5 Contrast Stretching

Contrast stretching is another way of enhancing an image by altering its histogram. Here, we stretch the minimum and maximum values of the intensity range of an 8-bit photo to enhance how it looks like. This is done using the equation

$$I_{new} = \frac{I_{old} - I_{min}}{I_{max} - I_{min}}$$

where I_{old} is the original intensity of the photo, I_{min} is the minimum, and I_{max} is the maximum.

Contrast stretching

MATLAB Code

```
image=imread('darkpicture.jpg');
igray = rgb2gray(image);
Inew = 255*((igray-min(igray))./(max(igray)-min(igray)));
imshow(Inew);
```

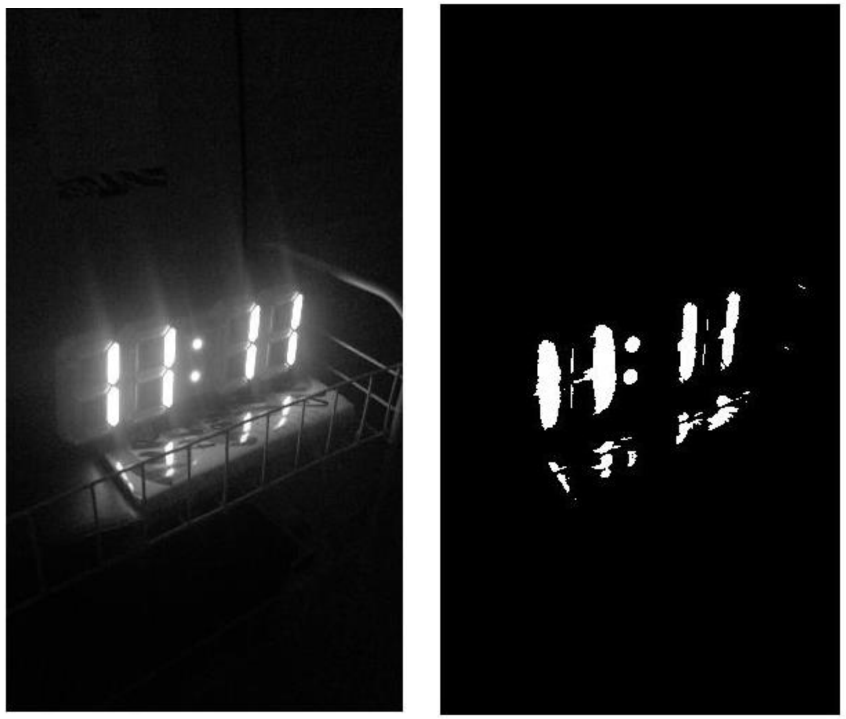


Figure 20. The grayscale of the dark image and the image obtained through contrast stretching.

For our first trial, we make use of the equation and apply it to the grayscale of our dark image. Here, we let MATLAB's min and max functions to determine the intensities that we need.

As we can see in the stretched image, only the light from the LED clock and its reflection can be seen. The rest looks black. We expect this to happen since what our equation does is to find the maximum and minimum values and intensify them, thus the contrast of our image is very high.

We can also tweak the values and base it on the CDF that we got for the backprojection and base our intensities on the projectiles. We will do this in the next slides.

Contrast Enhancement

The photos on the right are products of contrast stretching, but with different minimum and maximum intensities used. For the leftmost image, we use MATLAB's functions, for the one in the middle, we take the values at 0.9 and 0.1, and for the rightmost one, we made use of the values at 0.8 and 0.2. We can see that the light coming from the LED is more contained in (a) than the other photos. This is because we used the actual minimum and maximum values in our contrast stretching.

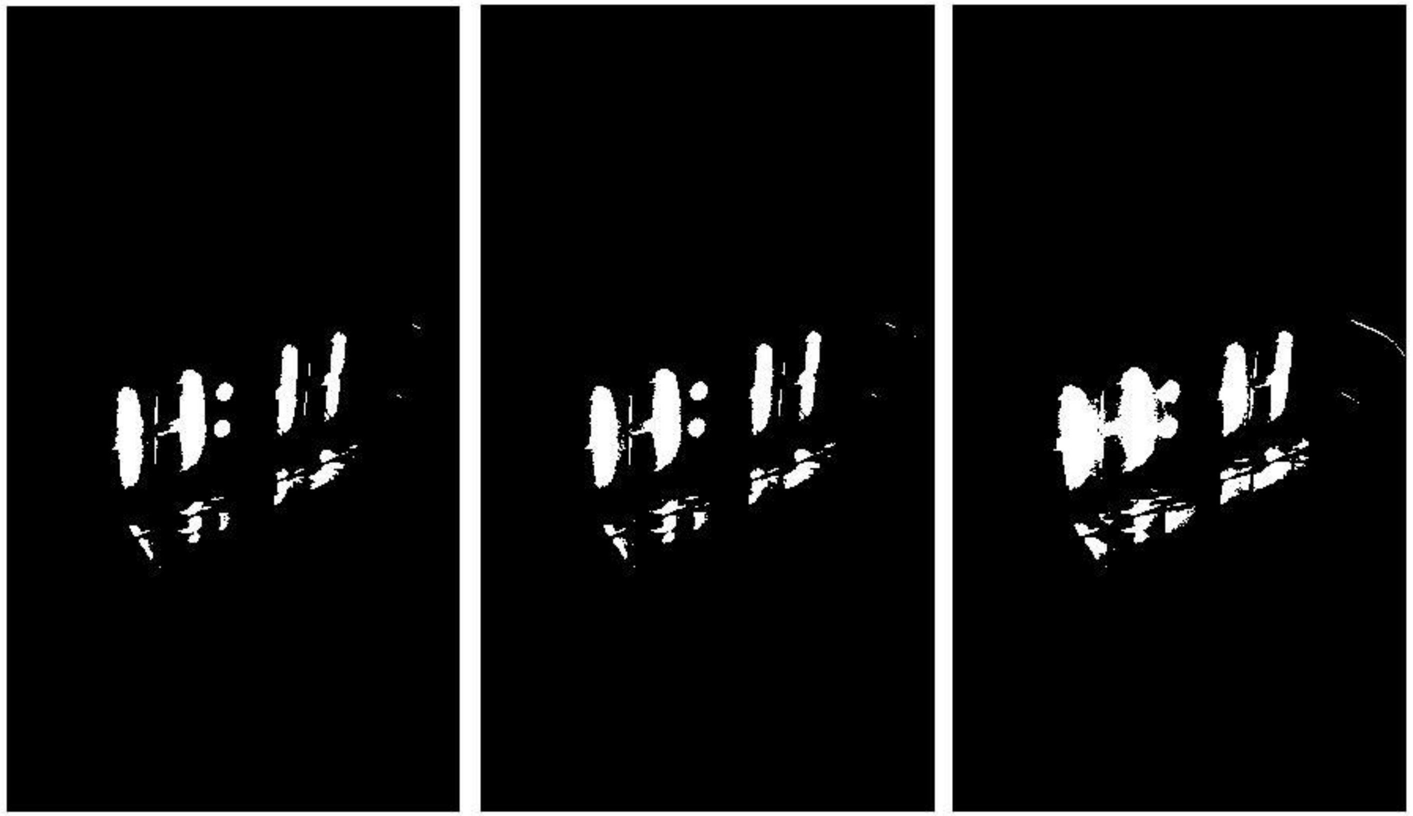


Figure 21. The contrast stretched grayscale dark image using (a, leftmost) MATLAB's min, max functions, (b, center) intensities at the 90th and 10th percentile, and (c, rightmost) intensities at the 80th and 20th percentiles.

Contrast Enhancement

Meanwhile, as the range of values we take increases, the light from the clock becomes more dispersed. For (c), it can be observed that the white pixels are very scattered, which shows the scattering of the light waves from the clock. This is because, we adjusted our minimum and maximum intensities to cater to more values, at the 80th and 20th percentile, thus, we expect that more details can be seen.

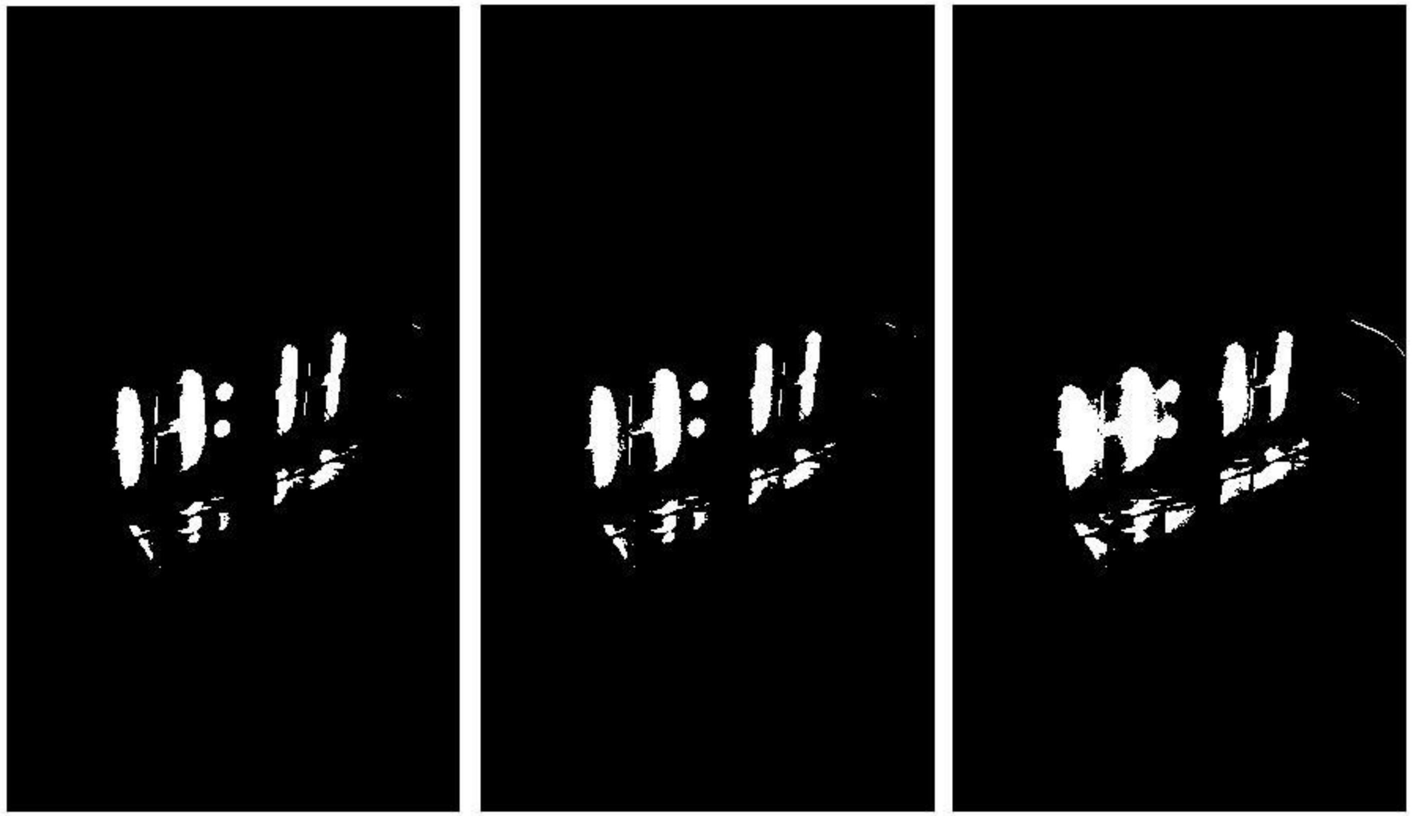


Figure 21. The contrast stretched grayscale dark image using (a, leftmost) MATLAB's min, max functions, (b, center) intensities at the 90th and 10th percentile, and (c, rightmost) intensities at the 80th and 20th percentiles.

Contrast Enhancement VS Backprojection

Both processes are geared towards image enhancement. Backprojection enhances an image by tweaking its distribution function to be linearized so as to enhance all the pixels and show more details. On the other hand, contrast stretching stretches the range of values of an image's intensity and uses it to enhance the image.

The difference in the outputs between the two process is that backprojection reveals more details of the image as it *evens out* the pixels, making them appear brighter. Meanwhile, for the contrast stretching, we cannot see the details of the picture anymore, instead, the highlights are made lighter and the shadows darker. Both are essential imaging tools that may serve different purposes for the users.

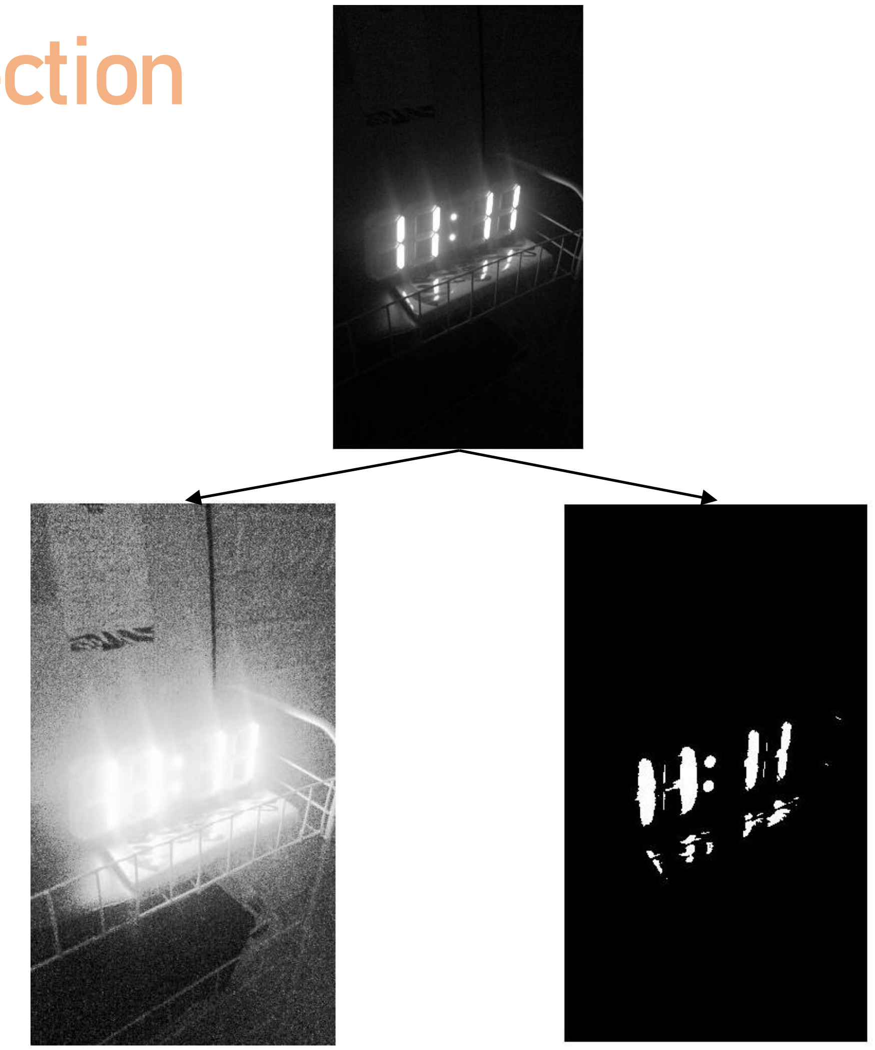


Figure 22. Comparison between backprojection (left) and contrast stretching (right)

Image Restoration

Activity 1.6 Restoring Faded Colored Photographs

For this activity, we aim to restore a faded colored image using different image restoration methods. I used this scanned image from my wallet, which is the closest thing I have to a faded colored image.

Because of the oldness of the photo and the filter added when this was taken, the then white background now appears gray. To restore this, we make use of three white balancing algorithms in MATLAB: Contrast stretching, gray world algorithm, and white patch algorithm.

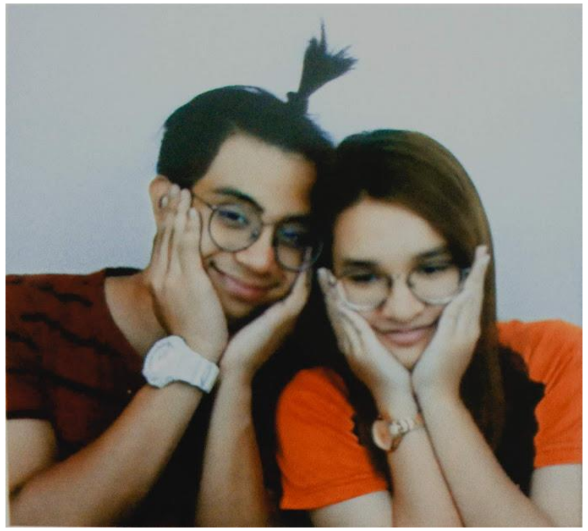


Figure 23. Faded Colored Image

Image Restoration

Activity 1.6 Restoring Faded Colored Photographs

White balancing is the process of adjusting a photo such that its white parts actually appear white in the photo, while the rest of the colors are rendered. One of the white balancing algorithm is contrast stretching. The process here is the same as what we did in the previous slides. However, here, we have to stretch for all the RGB channels and then overlay them.

Meanwhile, for white patch algorithm, we take a patch of the photo that is known to be white, and use it in adjusting the image. Lastly, Gray world algorithm makes use of the grayscale values aside from the RGB channels to restore the image.

In this portion, we will try out all three and compare their results.

Contrast Stretching

MATLAB Code

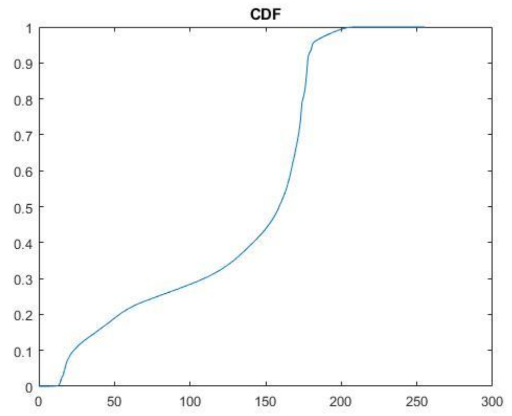
```
I = imread("pic.jpg");
imshow(I);
% Load the RGB channels of the image
R = I(:,:,1);
G = I(:,:,2);
B = I(:,:,3);

% To get the CDF of the image per color channel so we can use it as
% the minimum and maximum values for contrast stretching
PDFR = hist(R(:), [0:255])/numel(R);
CDFR = cumsum(PDFR);
R_min = find(CDFR > 0.02, 1, 'first'); % We use the 2nd and 98th
R_max = find(CDFR > 0.98, 1, 'first'); % percentile in taking our values
R_stretched = (R- R_min)*(255/(R_max - R_min));
figure(2); imshow(R_stretched); % Display the stretched channel
```

First, we read the image and load its RGB channels into an array. This is essential for later when we would overlay the results of the RGB channels after stretching.

Next, we stretch one channel, in this case, the R channel. We do this by using the CDF of the red channel and finding the minimum and maximum values at the 2nd and 98th percentile. We use these values in our contrast stretching equation.

As you can see, the resulting stretched image appears grayscale. We repeat contrast stretching for all the channels.



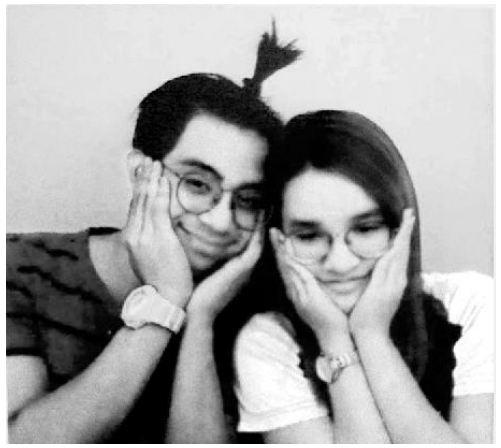


Figure 24. Stretched R channel of the faded colored image and its CDF

Contrast Stretching

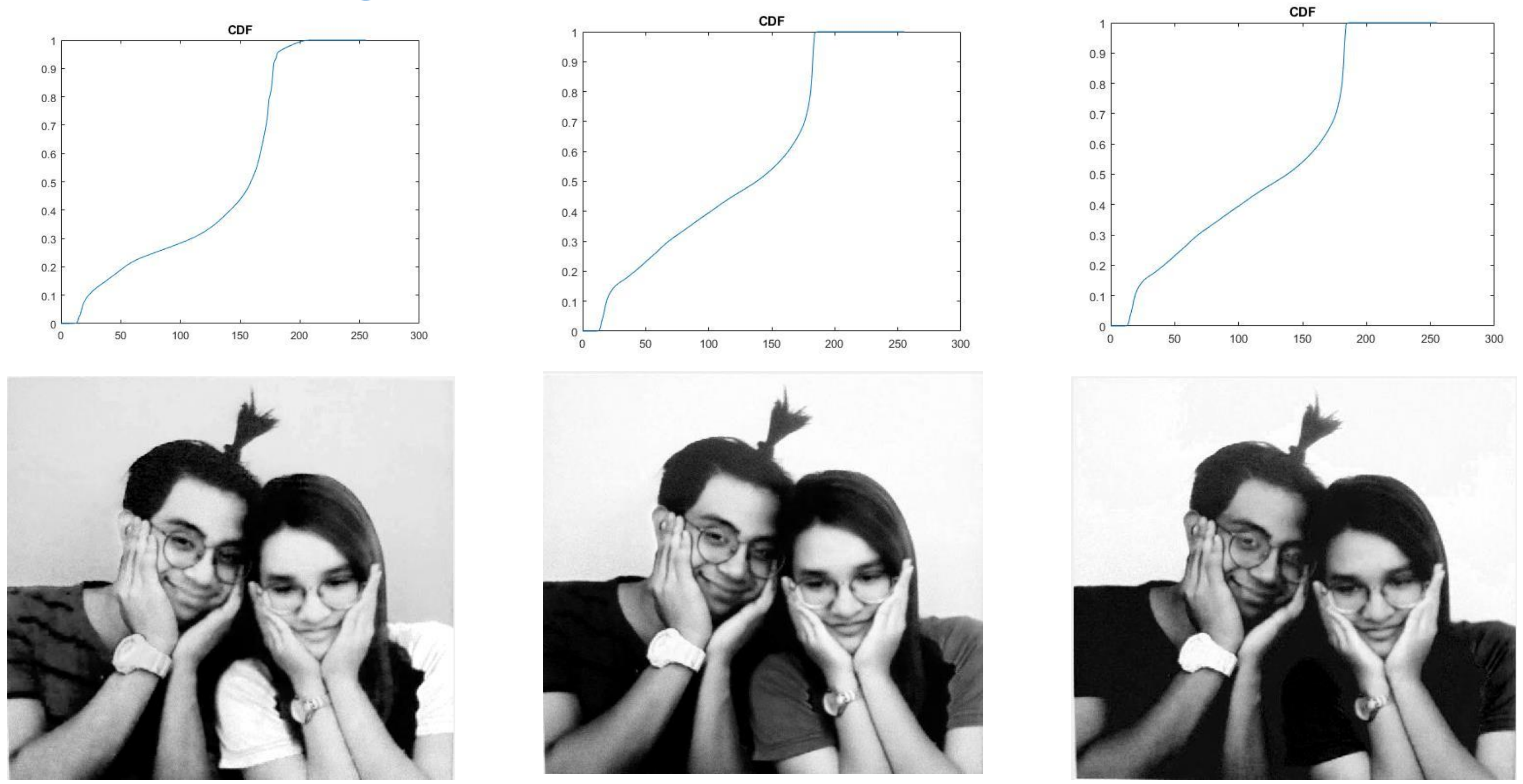
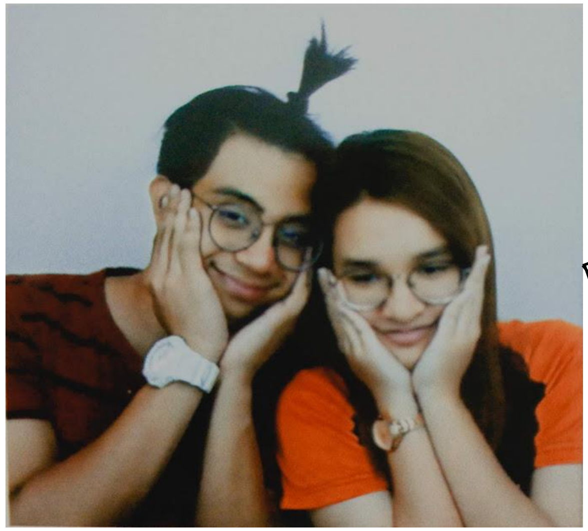
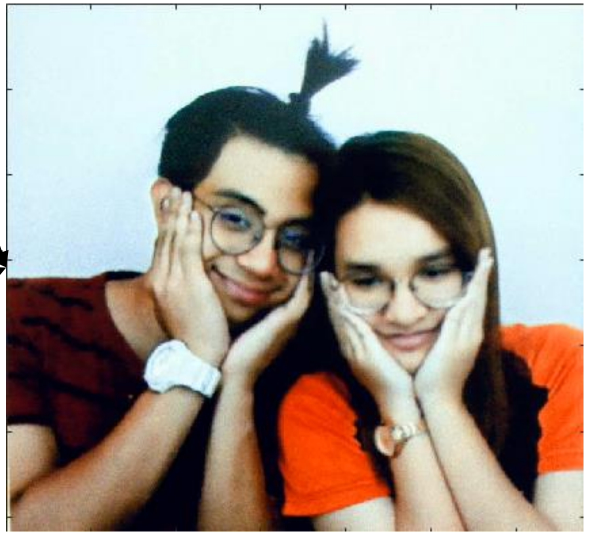


Figure 25. Stretched RGB channels of the faded colored image and their corresponding CDF. (Left to right: red, green, blue)



In the resulting photo, we see that the background is now closer to white than before. The photo subjects have also lightened and the skin tone is now more accurate than the original photo.



We overlay the stretched color channels to generate our whitebalanced image.

```
% Overlay each stretched color channel into one matrix
I_restored(:,:,1) = R_stretched;
I_restored(:,:,2) = G_stretched;
I_restored(:,:,3) = B_stretched;

figure(5);
image(I_restored);
```

Figure 26. Overlay of stretched RGB channels of the faded colored image and the resulting whitebalanced image.

Gray World Algorithm

MATLAB Code

```
I=imread('pic.jpg');
grayImage = rgb2gray(I);
% Extract the individual red, green, and blue color channels.
R = I(:, :, 1);
G = I(:, :, 2);
B = I(:, :, 3);
% Take the mean of the RGB and gray channels
Rave = mean2(R);
Gave = mean2(G);
Bave = mean2(B);
GYave =mean2(grayImage);
% Make all channels have the same mean
Rwb = uint8(double(R)*GYave/Rave);
Gwb = uint8(double(G)*GYave/Gave);
Bwb = uint8(double(B)*GYave/Bave);
% Recombine separate color channels into a single, true color RGB image.
K = cat(3, Rwb, Gwb, Bwb);
imshow(K);
```

For the gray world algorithm, it assumes that the colors in an image average to a neutral gray [1].

To do this, we follow the same first steps as the contrast stretching. However, here, we take the mean of the RGB and the gray channels.

We multiply the mean of the gray channel to the RGB channel and divide them by their respective means so as to equalize each channel.

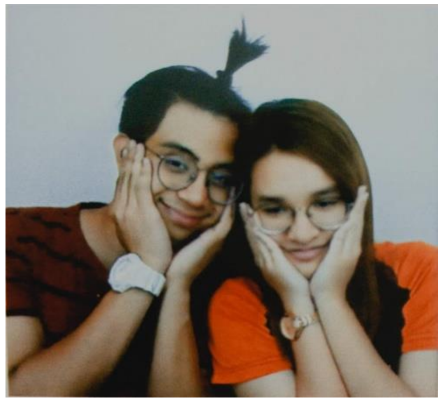
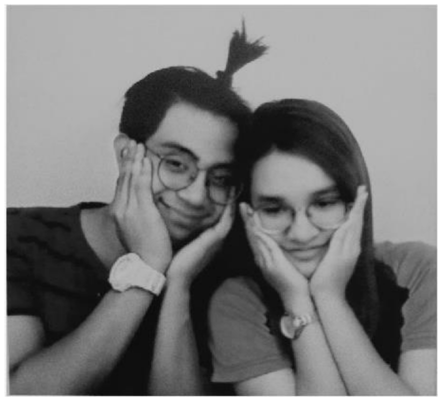



Figure 27. Original faded image and its grayscale version to be used for gray world algorithm.

Gray World Algorithm

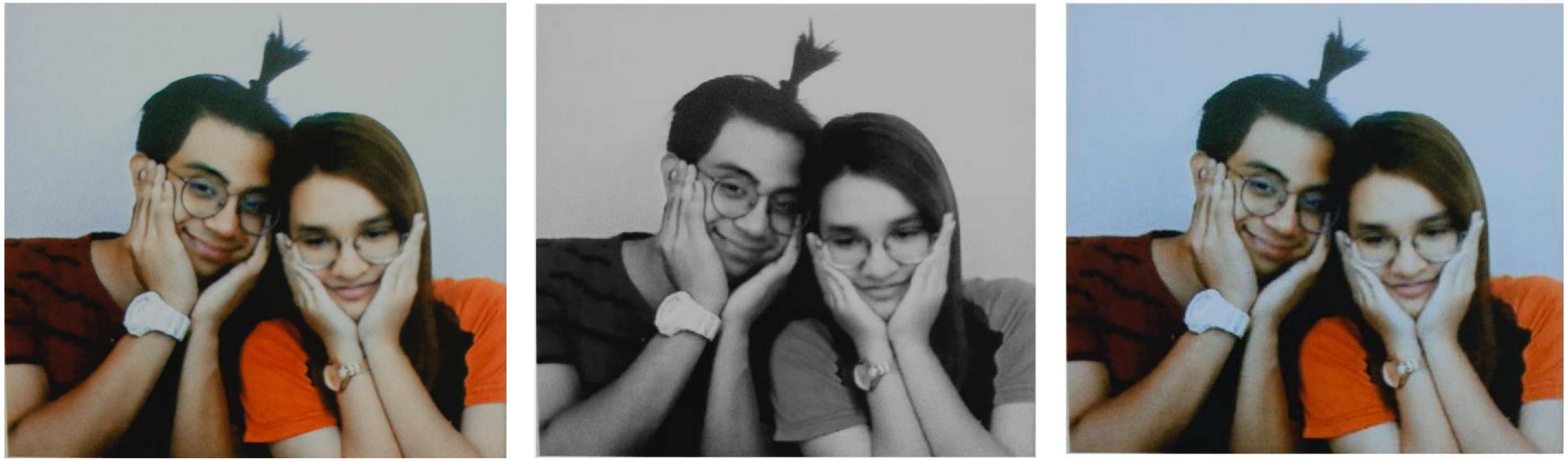


Figure 28. Original faded colored image, its grayscaled version, and the resulting image after the gray world algorithm (left to right)

Our resulting image can be seen on the right. We can observe that the final photo has a blue tint. This means that the image may have a little too much blue tint to reconstruct a whitebalanced image.

Here, we say that the gray world algorithm fails in restoring the image.

White Patch Algorithm

MATLAB Code

```
I = imread('pic.jpg');
figure; imshow(I); title('Original Image');
% We turn on the axis to see the patch area to take
axis on
patch = I(1:2, 1:2);
%imshow(patch)
% Extract the individual red, green, and blue color channels.
R = I(:, :, 1);
G = I(:, :, 2);
B = I(:, :, 3);
% Take the mean of the RGB channels and the patch
Rave = mean2(R);
Gave = mean2(G);
Bave = mean2(B);
patch_ave = mean2(patch);
% Make all channels have the same mean
Rwb = uint8(double(R)*patch_ave/Rave);
Gwb = uint8(double(G)*patch_ave/Gave);
Bwb = uint8(double(B)*patch_ave/Bave);
% Recombine separate color channels into a single,
% true color RGB image.
K = cat(3, Rwb, Gwb, Bwb);
figure; imshow(K); title('Whitebalanced Image');
axis on
```

For the white patch algorithm, we take a patch of the photo, which is known to be originally white. For my photo, I took the patch with coordinates (1:2, 1:2). In the figure below, the red patch is an enlarged example of the patch I took.

The process is similar to that of the gray world algorithm, except, here, we take the mean of the white patch and use it to equalize the means of the RGB channels. I just added axes to locate the patch.

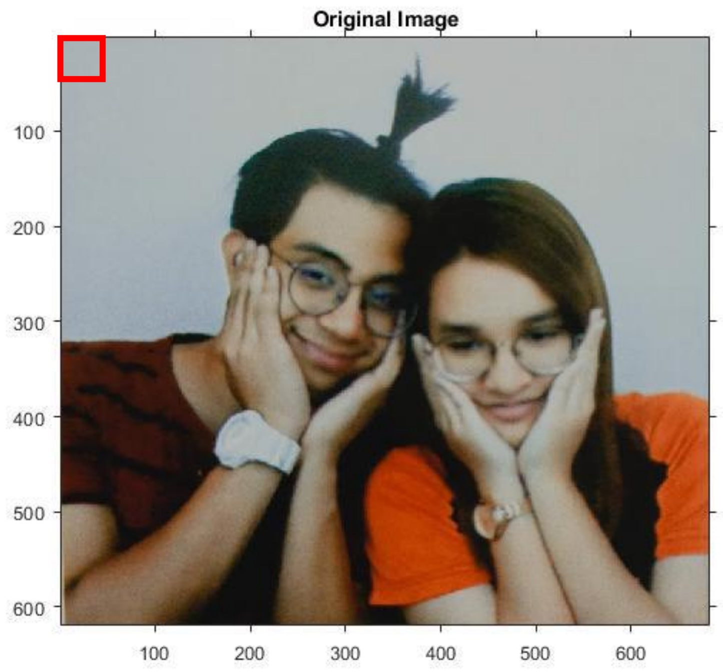


Figure 29. Original faded image and an enlarged patch for the white patch algorithm.

White Patch Algorithm

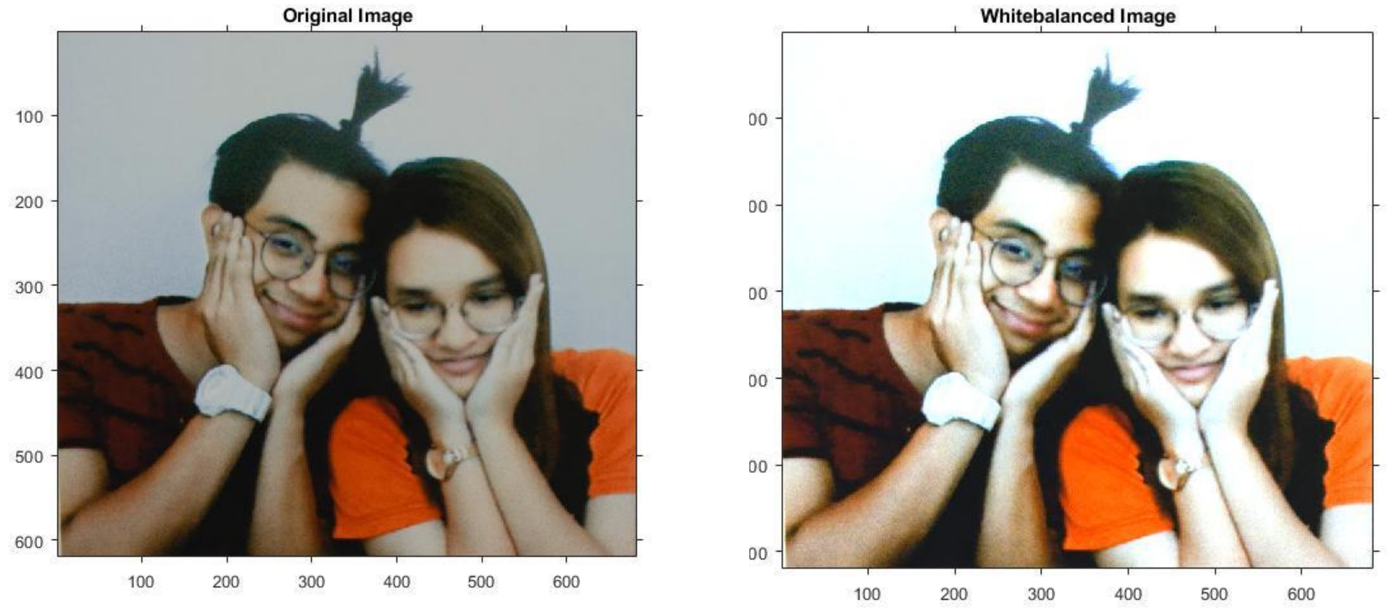


Figure 30. Original faded colored image, its whitebalanced version done using white patch algorithm.

Here, we see that the image actually improved, and some of the background has been restored to white. However, we see a bluish gradient in the background near the subjects. This suggests that the color of the background was not even in the scanned image. Despite the improvement in the background, the skin tones of the subjects are not reflective of the true tones, which may have been affected by the whitebalancing algorithm.

Whitebalancing Algorithms Comparison

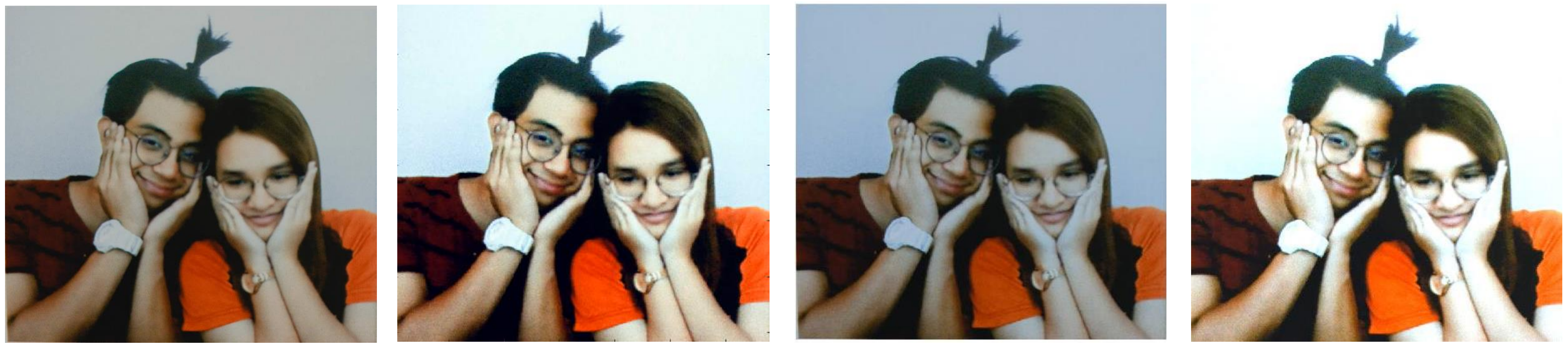


Figure 31. Left to right: Original faded image, contrast stretched, gray world, and white patch algorithm

Looking at the images side-by-side, we see that each algorithm has a different effect on the photo. For the contrast stretched version, the overall image was restored, however not fully since the background still does not look white. Meanwhile, in the gray world algorithm, the image shows a bluish tint. In the white patch algorithm, the background did turn into white, however, the skin tone of the subjects got affected.

Judging by the results, the contrast stretching version delivered the best image of the three because it was able to reconstruct the image including the background and the skin tones relatively well.

Reflection

Personal Experience

Performing this activity was very interesting because it is not something that I am familiar with. Through this, I was able to learn about MATLAB and the different functions available there since I am used to coding in Python.

I was also really excited to see my codes come to fruition, which made me appreciate programming more. I was amazed that you could construct accurate images by just inputting a few lines of codes. I enjoyed reconstructing the Olympics logo. I thought of doing the logo of one of my favorite albums, Wings by BTS, because it was also composed of circles. But, I did not have enough time anymore. Perhaps, I will do this when I have more time after the semester.

It did take me a long time to figure out how to perform the activities. To overcome this, I asked for help from my friends under the VIP lab, and also looked at tutorials and forums online like in MathWorks and StackExchange to understand the processes better.

Reflection

Technicalities

I had the most difficulty in restoring the faded colored image. I tried out different variations of the contrast stretching using different percentiles, however, my resulting image would look like the figure on the right. I resolved it by letting the algorithm choose the minimum and maximum values with the percentile I chose, instead of manually choosing the values myself.

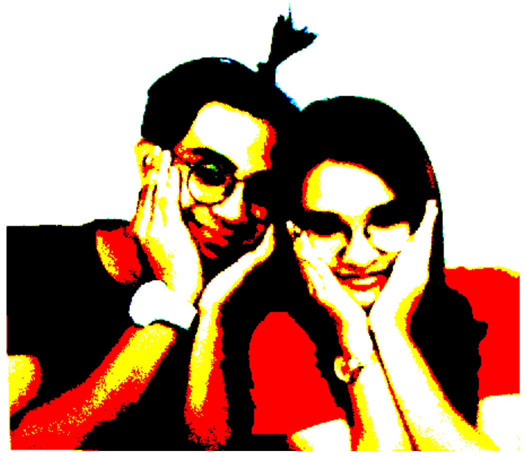


Figure 32. Failed image restoration using contrast stretching

Validity

For the first activity, my outputs met all of the objectives set. I also tried to play with the code to see how it would affect the images generated, such as changing the function used for the sinusoid plot, varying the sigma in the Gaussian transmittance, and more. My results were consistent of the theories we have learned in our math and physics classes, like the form of the graphs and the behavior of the functions.

Reflection

Validity

For the second part, which is the image enhancement, I was able to use the methods available to me to enhance my dark image. For the alteration of the input-output curves, I tried out using different curve shapes and found out that the best one for my image is altering the curve to be above the diagonal. Since my image was very dark and had a pop of light, it was important to increase the shadows and highlights.

Meanwhile, for contrast stretching, I was able to produce a well-contrasted grayscale image by using the 90th and 10th percentile for my maximum and minimum values. Here, the details were not visible anymore, which is to be expected since we intensified the shadows and highlights. Lastly, for backprojection, the details in my final image are now observable because the distribution has already been linearized. All of the results for this part coincide with the expected outcomes.

Reflection

Validity

Lastly, for the image restoration part, I was able to utilize the white balancing algorithms available and displayed the resulting images. The algorithms did unique changes to the faded image such that for the contrast stretching, the background was lightened and the skin tones were improved to look natural. For gray world, the resulting photo has a bluish tint, while for the white patch, the background became white, but the skin tones got affected such that they look unnatural.

This goes to show that the different algorithms have their strengths and weaknesses, and it depends on the composition of the photo that you will restore which of the three will be the most effective. For my case, it was contrast stretching.

References

[1] Abeln, M. (2011). White Balance, Part 2: The Gray World Assumption and the Retinex Theory. Obtained from http://therefractedlight.blogspot.com/2011/09/white-balance-part-2-gray-world.html

[2] Katz, M. (2008) Olympic Fever. MathWorks. Obtained from https://blogs.mathworks.com/community/2008/08/11/olympic-fever/

[3] Schenker, M. (2021). What Is the Difference Between JPG, PNG, BMP, and TIFF Images? Brush Up. Obtained from https://creativemarket.com/blog/difference-between-jpg-png-bmp-tiff-images#:~:text=While%20both%20PNG%20and%20JPG,sheer%20quality%20of%20your%20prints.

[4] Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Gaussian-bell-function-normal-distribution-N-0-s-2-with-varying-variance-s-2-For_fig1_334535945 [accessed 4 Jun, 2022]