

---

# DIGITAL IMAGE FORMATION AND ENHANCEMENT

---

## Objectives

1. Mathematically create images.
2. Save an image in an appropriate file format.
3. Open and capture images using Python or Matlab
4. Improve the appearance of graylevel and color images
5. Use the **backprojection** technique to transform the histogram of an image to a desired distribution

## Introduction

With cameras found nearly everywhere, (in your phones and laptops, on surveillance CCTV's, in cars, in laboratories, and in satellites) making objective sense out of images and video is a powerful skill to possess. In the following activities, we first learn how to “make” synthetic digital images which we can use for modeling, analysis, and experiments. Images can be saved in different formats or modes depending on the application. Lastly, we learn how to enhance digital images for further use.

My preferred programming language is MATLAB because it is easy to learn and use for visualization (you can get a license using your up.edu.ph account at [www.mathworks.com](http://www.mathworks.com)) but you can also use Python offline, so that you can use opencv2. (OpenCV works poorly in Google Collab). In addition, kindly install any of these free image processing software : GIMP ([www.gimp.org](http://www.gimp.org)) and ImageJ .

## Creating Synthetic Images

It is sometimes necessary to create synthetic images for masking, testing algorithms, or simulating optical elements in imaging systems. Here's how to create a centered circle in MATLAB, and PYTHON:

Matlab	Python
<pre>N = 200; x = linspace(-1,1,N); y = x; [X,Y] = meshgrid(x,y); R = sqrt(X.^2 + Y.^2); A = zeros(size(R)); A(find(R&lt;0.5)) = 1; figure (1); imshow(A); figure (2) ; mesh(x,y,A);</pre>	<pre>import numpy as np import matplotlib.pyplot as plt N = 200 #the higher num is the finer x = np.linspace(-1,1,num = N) y = x X,Y = np.meshgrid(x,y) R = np.sqrt(X**2 + Y**2) A = np.zeros(np.shape(R)) A[np.where(R&lt;0.5)] = 1.0  #display as an image</pre>

	<pre>plt.imshow(A, cmap = "gray")  #display as a 3D surface in Cartesian coordinates from mpl_toolkits.mplot3d import Axes3D fig = plt.figure() ax = Axes3D(fig) ax.plot_surface(X,Y,A)</pre>
--	---

In Figure 1 and 2 are the outputs of the Matlab script showing an image view and 3D view of the circle we just made. You'll get the same from Python. You might think, "Why not just draw it in Paint?". In simulations it is necessary to be accurate about the object parameters (e.g. diameter, length, distance between slits, etc.). A circle, for example, can simulate the aperture of lenses. The diameter of the aperture determines how much light the lens can collect which translates to how well the lens can resolve fine details. The 3D view allows you to inspect if you got the simulation correctly.



Figure 1. Circle displayed as image

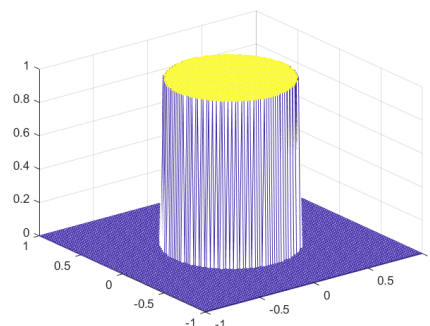
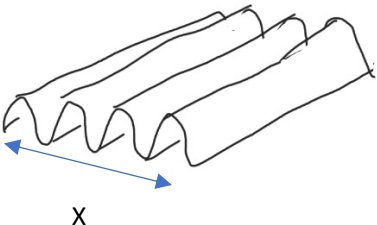
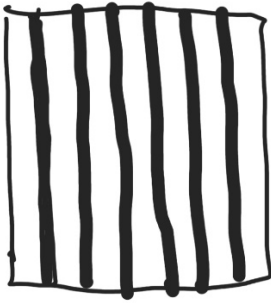
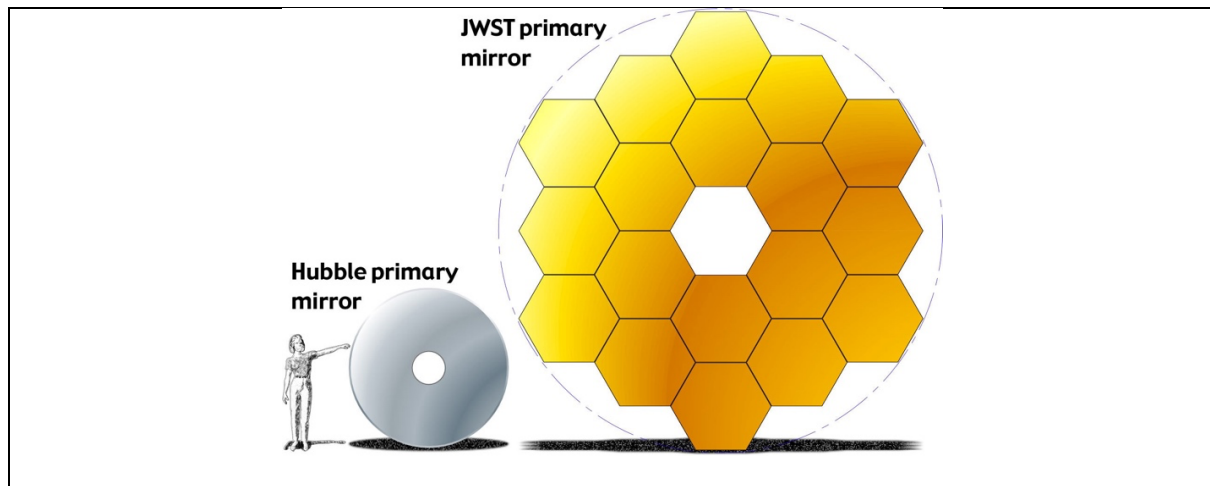


Figure 2. Circle displayed as a 3D surface in Cartesian coordinates

## Activity 1.1 Image DIY

<p><b>It's your turn!</b> Mathematically create the following images.</p> <p>For items 1 and 2, <math>X \in [-2 \text{ cm}, 2 \text{ cm}]</math>, <math>Y \in [-2 \text{ cm}, 2 \text{ cm}]</math>, we are simulating a 4cm x 4cm optical element. The image size should be 400x400 pixels.</p> <p><b>Hand-drawings are for reference and are NOT drawn to scale.</b></p>	<p>1. Sinusoid along the x-direction, frequency is 4 cycles/cm.</p>  <p>*Note: Physical images do not have negative values. What should you do such that the range of intensities of your sinusoid is between 0 and 255?</p>	<p>2. Grating, frequency is 5 line pairs/cm. A line pair is a strip of black (0) and white (1)</p> 
<p>For the following items you may choose your own image size, but not to exceed 2000x2000 pixels.</p> <p>3. Hubble's Primary Mirror – Circle with a hole in the middle (annulus).</p> <p>4. Hexagon Array, simulating the mirrors of the James Webb Space Telescope (image credit NASA)</p>		



Here's how to create color images. Due to our visions' trichromatism we only need to overlay three primary colors in different proportions to produce a color image. Study this code and try it out.

Matlab	Python
<pre> clear all ; close all; N = 500; x = linspace(-10,10,N); y = x; [X,Y] = meshgrid(x,y); Rd = zeros(N,N); Gn = A; Bl =A;  % draw colored circles Rt = 3; RC = 4; deg = 30 xt = Rt*cosd(deg); yt = Rt*sind(deg);  R = sqrt((X.^2) + (Y+Rt).^2); Rd (find (R&lt;RC)) =1;  R =sqrt((X-xt).^2 + (Y-yt).^2); Gn (find(R&lt;RC)) = 1;  R = sqrt((X+xt).^2 + (Y-yt).^2); Bl (find(R&lt;RC)) = 1;  I(:,:,1) = Rd; I(:,:,2) = Gn; I(:,:,3) = Bl;  figure; image(I); axis equal; </pre>	<pre> import numpy as np import matplotlib.pyplot as plt N = 500 x = np.linspace(-10,10,num = N) y = x X,Y = np.meshgrid(x,y) Rd, Gn, Bl = np.zeros((N,N)), np.zeros((N,N)), np.zeros((N,N))  #draw colored circles Rt, Rc, deg = 3, 4, 30 xt, yt = Rt*np.cos(deg*np.pi/180), Rt*np.sin(deg*np.pi/180)  R = np.sqrt((X)**2 + (Y+Rt)**2) Rd[np.where(R&lt;Rc)]=1.0  R = np.sqrt((X-xt)**2 + (Y-yt)**2) Gn[np.where(R&lt;Rc)]=1.0  R = np.sqrt((X+xt)**2 + (Y-yt)**2) Bl[np.where(R&lt;Rc)]=1.0  I = np.zeros((N,N,3))  I[...,0] = Rd I[...,1] = Gn I[...,2] = Bl  fig = plt.figure() plt.imshow(I) </pre>

Here's how it looks:

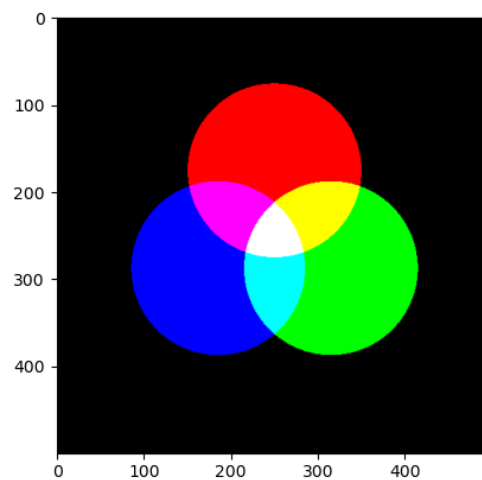


Figure 3. Synthetic colored image

## Image Modes

Before we can save the images we created, we need to convert it into a digitized form. Each pixel (“pixel” is short for “picture element”) in a digitized image is typically represented by an integer between 0 to 255 or an unsigned integer 8-bit number. In programming, this is the number type uint8. The information carried by the image dictates in what mode it should be presented. The mode affects the file size as well. In professional or research applications the bit resolution in research-grade cameras may go up to 16-bit. There are four image modes:

1. **Binary images** are images with pixels that are either ones (1's) or zeros (0's) or BITS. Examples are QR codes, bar codes, document text, and digitized line drawings such as rasterized architectural plans drawn in AutoCad. In the previous activity, the grating is an example of a binarized image.
2. **Grayscale images** have pixel values from black (0) to white (255) . These are also known as single channel or panchromatic images. Images are saved as grayscale if the information needed is embedded in gray tones. Examples include, medical or biological images (scanned x-rays, CT scans), faces for face recognition, and fingerprints. In the previous activity the sinusoid is an example of a grayscale image.
3. **Truecolor images** are images with three CHANNELS or BANDS. Each channel is the intensity of a red, green and blue primary light. Color is perceived when these three channels are overlaid. The colored circles in the previous example were assembled this way. The number of possible colors in a truecolor image is  $256^3$  or 1.6+ million colors. Images that need to be in truecolor include archival images, i.e. images whose color information need to be preserved as accurately as possible, e.g. images of paintings and other heritage objects. Commercial digital cameras capture images in truecolor.
4. **Indexed images** are colored images whose colors are represented by numbers which denote the index of the colors in a COLOR MAP. Two sets of data are stored in an indexed image, the image and its color map. An indexed image is generally smaller in size compared to a true color image but may have reduced color information. Infographics are examples of indexed images as they use only a few set of colors. Truecolor images that are saved as GIF become indexed images. The colored circles in the above example can be stored as an indexed image.

[Read about these image modes online.](#)

As more and more advanced imaging techniques and devices are developed, image types have become advanced as well. Some examples include:

1. **High dynamic range (HDR) images.** There are special scenes that require more than 8-bit grayscale recording in order to be appreciated. Examples are digital x-rays and very bright objects (nuclear explosion, sun or cloud images, plasma). HDR images can be stored in 10- to 16-bit grayscales.
2. **Multi or hyperspectral image.** These are images that have more bands than 3 (for Red, Green and Blue). A multispectral image has bands in the order of 10's while a hyperspectral image has bands in the order of 100. Example are satellite images.
3. **3D images.** With 3D printers becoming more affordable and popular, 3d imaging has also gotten a boost. There are many ways of storing spatial 3D information. 3D surfaces may be saved as point clouds (x,y,z), image stacks (2D images of several cross sections, such as MRI or CT SCAN images), stereopairs, (dual images of a scene taken a short distance apart).
4. **Temporal images or Videos.** Although moving pictures have been around since 1870, their digitization, compression and method of capture have advanced dramatically in the advent of smaller and smaller transistors. We now have cheap, high definition, high frame rate digital cameras that can show fast phenomena in slow motion.

## Activity 1.2 Color Image

1. Mathematically recreate the Olympics logo as an image. No need to make it as accurate as the logo below. It is fine if the colors overlap where the rings intersect.

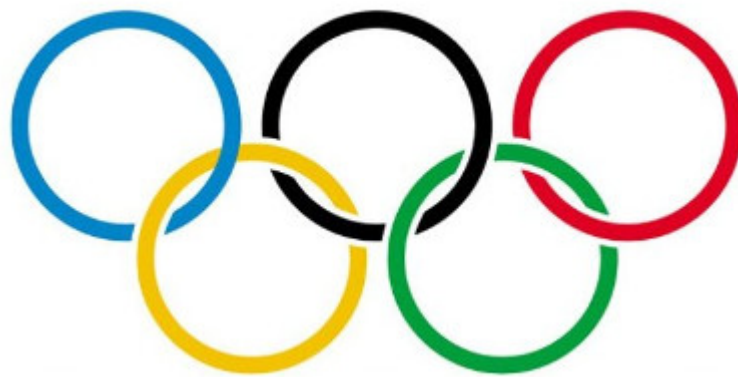


Figure 4 Olympic Logo

2. There are tons of image file formats but what will matter to us will only be 4. Research about the following file formats:
  - a. JPEG – file format developed by the Joint Photographic Experts Group
  - b. PNG – Portable Network Graphics
  - c. BMP – Bitmap
  - d. TIFF – Tagged Image File Format

3. Save all the images you created in Activity 2.1 and 2.2 into JPG, PNG, BMP, and TIFF format because we will use them in another activity. This is different from exporting the figure as an image since doing so will save the image with tick marks and indices of the matrix. To save an image from Matlab and Python here are some example codes:

Matlab	Python
<pre>#Let I be the image matrix I_img = uint8(I*255); imwrite(I_img, 'sinusoid.jpg'); imwrite(I_img, 'sinusoid.bmp'); imwrite(I_img, 'sinusoid.png'); imwrite(I_img, 'sinusoid.tif')</pre>	<pre>rgbim = rgb*255 img = rgbim.astype(np.uint8) plt.imsave("coloredcircle.jpg",img) plt.imsave("coloredcircle.bmp",img) plt.imsave("coloredcircle.png",img) plt.imsave("coloredcircle.tif",img)</pre>

## Image Enhancement

Digital cameras are designed to mimic properties of the human eye but there are some limitations in one that is not found in the other. For example, the sensor of a camera can record information under low light levels which our visual system can no longer perceive. On the other hand cameras will tend to saturate when there's an intense light in the scene while our eyes can manage bright scenes just fine.

Consider for example the image in Figure 5 below. Because of the bright background, the autoexposure control of the camera made the foreground dark.



Figure 5. Picture taken by a Canon D10 camera showing my research assistants on board MY Navorca. The bright sunny background caused the foreground image to appear dark.

It may seem that information is lost in the dark parts of the image. The lady on the left is wearing a black shirt so one might think there is no reflected light from her shirt especially so that it is in the shadows (Figure 6). But as you can see that by manipulation of the value input-output curve (done in GIMP), there is, in fact, some detail.





Figure 6. Left - close up of dark sections of the image. Right - after graylevel input-output curve manipulation using GIMP.

So do not trust your eyes. Clearly, the sensor of the camera picked up the information in the shadows except that our eyes are not sensitive enough to see them.

The **histogram** of an image is a plot of grayscale value or digital numbers (DN) versus the number of pixels having those values. Most image processing software allow you to view the histogram of an image in different **channels** (e.g. R,G,B) or in **color spaces** (e.g. hue, saturation, value or brightness). The **histogram** of a grayscale image is a low level property of an image because it is obtained from the pixel values only. High level properties such as texture or edges are obtained from areas of neighboring pixels. An image has poor contrast if its histogram is either skewed towards the brighter or the darker graylevels. A good contrast image will have an evenly distributed histogram across the grayscale.

In this activity we will learn different techniques to alter the histogram of an image for a human observer to see information even in seemingly dark or poor contrast images. So don't delete those dark images just yet.

## Activity 1.3 Altering the Input-Output Curve

1. Look for a dark-looking digital photo from your collection.
2. Open the image using GIMP (or any software that has histogram manipulation functions).
3. In GIMP, from the **Colors** drop-down menu select **Curves**. A separate interactive window will pop-up showing the histogram of the image. By default it shows the **VALUE** histogram. Value is the brightness of the pixel. See Figure 7.
4. The diagonal line across the histogram is the input-output curve of the value. Drag any point along this line to alter the I-O curve and play around with it until you see details in the dark portions of your image.

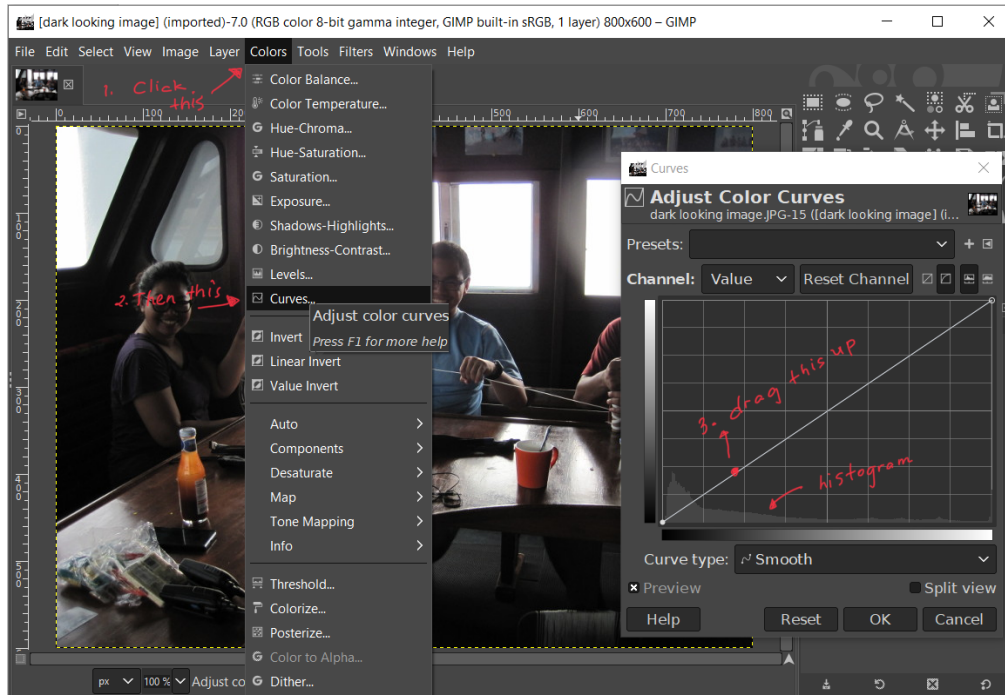


Figure 7. Illustration for Steps 3 and 4 or how to change the brightness of an image by altering the input-output curve of the brightness value.

## Histogram Manipulation by Backprojection

The histogram of an image can be converted into the probability distribution function (PDF) of its digital numbers if we normalize it by the total number of pixels (recall that the area under the PDF curve is equal to 1.0). Therefore, the grayscale PDF of an image can be modified in the same way one can modify the PDF of random numbers to follow a specified distribution. Histogram manipulation is one way of improving the quality of an image, enhancing certain image features, or mimicking the response of different imaging systems, such as the human eye.

Given the cumulative distribution function (CDF) of a desired PDF, the grayscale PDF of an image can be modified by **backprojecting** the grayscale values using the CDF. Suppose the graylevels  $r$  of an image has a probability distribution function (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.

We want to map the  $r$ 's to a different set of graylevel  $z$ 's such that the new image will have a CDF given by  $G(z) = \int_0^z p_2(t)dt$  where  $p_2(z)$  is the PDF of the transformed image and  $t$  is a dummy variable. If we let  $G(z) = T(r)$  then to get  $z$  we perform

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

To illustrate, let Figure 8 be the grayscale PDF of an image and Figure 9 its CDF.



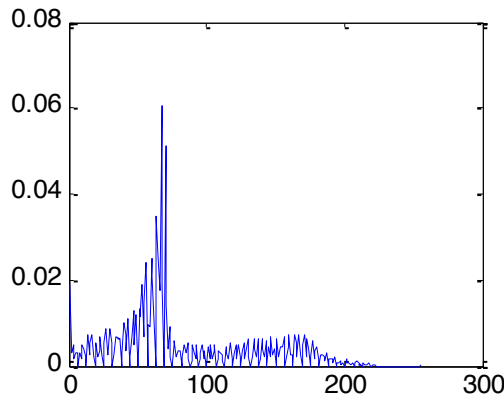


Figure 8. PDF (normalized histogram) of a grayscale image. The x-axis is grayscale digital number.

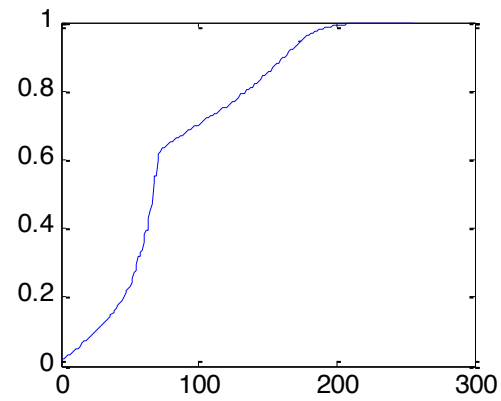


Figure 9. CDF of Figure 4.

Now suppose we want to remap the grayscales of the image such that the resulting CDF of the transformed image will look like that in Figure 10 below.

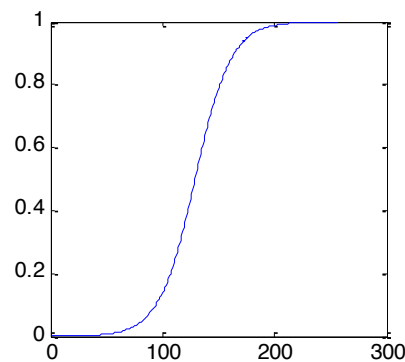


Figure 10. Desired CDF.

The steps are then:

1. Per pixel in the image having grayscale  $r$ , find its CDF value  $T(r)$ .
2. Find the value of  $T(r)$  in the y-axis of the CDF  $G(z)$ . Find the  $z$  value of this  $G(z)$ . See Figure 11. This is a graphical illustration of Equation 1 and it shows why the technique is called backprojection.

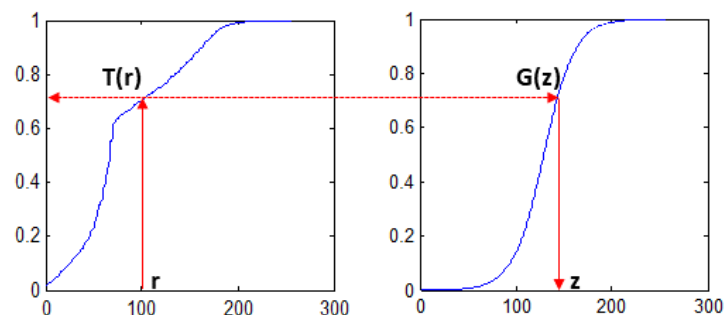


Figure 11. Illustration of  $z = G^{-1}T(r)$  or histogram backprojection.

## Activity 1.4 Histogram Backprojection on Grayscale Images

1. Open your dark-looking image in Matlab or Python and convert to grayscale.

How to open images

In Matlab	In Python
<pre>Igray = rgb2gray(imread("Dark picture.jpg")); imshow(Igray);</pre>	(find the equivalent of imread in Python packages)

2. Obtain the grayscale histogram of your image and normalize by the number of pixels to get its PDF. Compute the CDF from the PDF.

In Matlab	In Python
<pre>PDF = hist(Igray(:), [0:255])/numel(Igray); CDF = cumsum(PDF); figure; plot([0:255], CDF);</pre>	(find the equivalent of hist and cumsum)

3. For a uniform distribution, all graylevels,  $g$ , are equally probable. That is, if the pdf is  $p = 1$ , then  $cdf = \int_0^g pdg = g$ . Create the CDF function of a uniform distribution, with x values equal to 0 to 255 (for an 8-bit grayscale image) and y values equal to 0 to 1. Use histogram backprojection to pixel-per-pixel backproject the image pixel values by finding its corresponding y-value in the desired CDF. That is, replace the dark pixel values with the x-values from the desired CDF.

Tip : You can find the inverse of a function if the analytic form of the function is known. However it is easier and computationally simpler to do an inverse mapping by using a look up table and an interpolation routine. Most interpolation functions have the following syntax:

`newY = interp1(oldX, oldY, newX)`

Here, oldX and oldY are the representative points for the function  $f$  given as  $oldY = f(oldX)$  and `interp1` is the function that does 1-d interpolation. To invert, simply interchange the position of oldX and oldY like this:

`newX = interp1(oldY, oldX, newY).`

In Matlab	In Python
<pre>x = [0:255]; desiredCDF = (1/255) * x; newGS = interp1(desiredCDF, x, CDF(Igray(:)+1)); Igraynew = reshape(newGS, size(Igray)); figure; imshow(uint8(Igraynew));</pre>	(find the equivalent of interp1 and reshape)

4. Display the modified image. This image is known as a histogram equalized image. Plot the histogram and get the CDF from the histogram values to see if the desired CDF was achieved.
5. The human eye has a nonlinear response. It has a higher gain in dark light levels and lower gain in bright levels. Create a CDF that has a nonlinear shape and use this to manipulate the histogram of your image. Try out different PDFs as well.

FYI : You can use the same backprojection technique for changing the distribution of random numbers!

## Contrast Stretching

The CDF computed from the histogram of a grayscale image also gives us the grayscale at which the number of pixels reaches a certain percentile. For example, if at grayscale  $x = 20$  the  $CDF = 0.25$  this means 25% of the total number of pixels have grayscales 20 and below.

Suppose we have a set of numbers  $I_{old}$  (an array or a vector, for example). In that set the minimum value is  $I_{min}$  and the maximum value  $I_{max}$ . To contrast “stretch” the values of  $I_{old}$ , we normalize using

$$I_{new} = \frac{I_{old} - I_{min}}{I_{max} - I_{min}}. \quad (2)$$

Notice that now the range of the new numbers  $I_{new}$  is 0 to 1.0.

If for a grayscale image the minimum and maximum value of its grayscale is 0 and 255 respectively, equation (2) will not alter the image. In such a case, one selects as  $I_{min}$  and  $I_{max}$  the grayscale at a percentile of the dark and bright pixels. For example  $I_{min}$  can be the grayscale at  $CDF = 0.1$  and  $I_{max}$  can be the grayscale at  $CDF = 0.9$ .

## Activity 1.5 Contrast Enhancement

1. Contrast stretch your dark picture by applying Equation (2) to its grayscale. Try out different percentiles for the minimum and maximum intensities.

## White Balancing of Color Images

A colored digital image is formed from the overlay of three matrices representing the redness, greenness, and blueness of an image. Reading a colored image from your scientific software you will see that the size of the image is  $M \times N \times 3$  where  $M \times N$  is the size of the image, and 3 is because of the red, green, and blue channels of the image. Per pixel, the color captured by a digital color camera is an integral of the product of the spectral power distribution of the incident light source  $M_e(\lambda)$ , the surface reflectance  $\rho(\lambda)$  and the spectral sensitivity of the camera  $S_i(\lambda)$ . That is, if the color camera has spectral sensitivity  $S_R(\lambda)$ ,  $S_G(\lambda)$ ,  $S_B(\lambda)$  for the red, green and blue channels respectively, then the color of the pixel is given by

$$DN_R = K_R \sum M_e(\lambda) \rho(\lambda) S_R(\lambda) \quad (3)$$

$$DN_G = K_G \sum M_e(\lambda) \rho(\lambda) S_G(\lambda) \quad (4)$$

$$DN_B = K_B \sum M_e(\lambda) \rho(\lambda) S_B(\lambda) \quad (5)$$

where

$$K_i = \frac{1}{\sum M_e(\lambda) S_i(\lambda)} \quad (6)$$

is a balancing constant equal to the inverse of the camera output when shown a white object ( $\rho(\lambda) = 1.0$ ). Equations (3) to (5) explain why sometimes the colors captured by a digital camera appear unsatisfactory.

## White Balancing in Cameras

If you inspect your digital camera you will find that it has a setting called “WHITE BALANCE” or “WB”. This setting allows the user to select the white balancing constants appropriate for the capturing conditions. For example, you may have seen icons such as a sun (for sunny, bright outdoors), cloud (for cloudy day), light bulb (for incandescent light) and a fluorescent lamp (for fluorescent lighting). You may also have seen “AWB” which stands for Automatic White Balancing, and symbols such as “6500K” which stands for the color temperature of the ambient light. By setting your camera white balance to the illumination condition, you will obtain an image where white appears white and the rest of the colors are properly rendered.

White balancing algorithms can be used to restore faded color photographs or correct unbalanced images. How do you know if an image is unbalanced? If there is a known white object in the scene but it does not look white, it is unbalanced. Skin color is another indicator. If there are people in the picture and their skin color looks purple, bluish or even greenish (and you’re certain they’re not sick or dead- haha) then the image may be unbalanced, like in the picture shown on the left. Have you ever took a picture of your ID picture so that you can insert it on a PDF form? I’m pretty sure my background was white when this photo ID was taken. But I took the picture of this ID using my phone in a hall which had orange lights, the resulting image has an orange cast and my skin appears jaundiced.

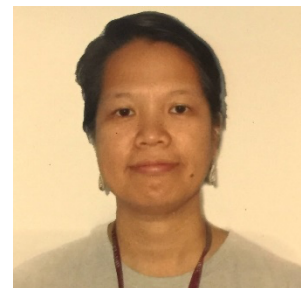


Figure 12. An unbalanced photo. Background was originally white.

## Automatic White Balancing Algorithms

There are three popular algorithms for achieving automatic white balance. These are **Contrast Stretching**, **White Patch Algorithm**, and the **Gray World Algorithm**.

### Contrast Stretching in RGB

In Activity 3.3 we applied contrast stretching in grayscale. In color images, white balancing can be achieved by contrast stretching each color channel separately using a set percentile for min and max. Check out the pseudo algorithm in the website : <https://adadevelopment.github.io/gdal/white-balance-gdal.html#:~:text=The%20white%20balance%20is%20an%20algorithm%20that%20adjust,idea.%20It%20is%20an%20simple%20and%20fast%20algorithm.>

### White Patch Algorithm

If you look carefully at equations (3) to (5) these are just the raw camera outputs divided by the camera output for a white object. So the key is to determine what the camera output is for a white object. In the White Patch Algorithm use the RGB values of a known white object as the denominator in Equation (6).

## Gray World Algorithm

In the Gray World Algorithm, it is assumed that the average color of the world is gray. Gray is part of the family of white, as is black. Therefore, if you know the RGB of a gray object, it is essentially the RGB of white times a constant factor. Thus, to get the balancing constants, you take the average red, green and blue value of the captured image and use them as the denominator in Equation (6).

### Activity 1.6 Restoring Faded Colored Photographs

1. Can white balancing restore faded or imperfect photographs? Scan a faded or unbalanced photograph or get one from your collection. Note: photograph must originally be in color. Sepia photos do not count as colored photographs.
2. Apply all three white balancing algorithms to your image.
  - a. Contrast Stretching
    - i. If you load the image as  $I$  let  $R = I(:, :, 1)$ ;  $G = I(:, :, 2)$ ;  $B = I(:, :, 3)$ .
    - ii. If you `imshow` these channels they look gray. The graylevel indicates how bright the red, green or blue light must be to reconstruct the color.
    - iii. Perform contrast stretching on each of these channels and overlay them into one matrix, e.g,  $I\_restored(:, :, 1) = R\_stretched$ ,  $I\_restored(:, :, 2) = G\_stretched$ ,  $I\_restored(:, :, 3) = B\_stretched$ , and then `imshow I_restored`.
  - b. Gray World Algorithm
    - i. Get the average value of the red, green and blue channel of your image,  $R_{ave}$ ,  $G_{ave}$  and  $B_{ave}$ , respectively.
    - ii. To get the white balanced image, divide each RGB channel by the respective averages, that is,  $R_{wb} = R/R_{ave}$ ,  $G_{wb} = G/G_{ave}$ ,  $B_{wb} = B/B_{ave}$ . Notice this is the same gist as Equations 3 to 6. Display the image.
  - c. White Patch Algorithm. This technique only works if there are white parts in the image. Skip this if there are none.
    - i. Get a region from the image which is known to be white (e.g. white sand or uniform, white sand, white paper, etc.) Average the RGB of the white pixels to get  $R_w$ ,  $G_w$ ,  $B_w$ , separately.
    - ii. Divide each channel with the whole original image with the respective white averages from each channel, that is,  $R_{wb} = R/R_w$ ,  $G_{wb} = G/G_w$ ,  $B_{wb} = B/B_w$ . Display the image.
3. Comment on how each algorithm fared in restoring your image.

#### **FUN FACT**

*Instagram filters that make your pictures look a certain colored way are nothing but Equations 3 to 4 with the balancing constants  $K_i$  set to some set of numbers. Try it! Set the  $K$ 's (a number from 0 to 1.0) to 3 arbitrary numbers for  $R$ ,  $G$  and  $B$  and see how your pictures look.*

## References

Gonzales, R. C., & Woods, R. E. (2008). Digital image processing.

[www.mathworks.com](http://www.mathworks.com) documentation

[www.gimp.org](http://www.gimp.org) documentation