ECE 438 - Laboratory
Image Processing Basics

In [ ]:
```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt


# refer to https://matplotlib.org/3.1.0/gallery/mplot3d/surface3d.html
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

In [ ]:
```
# make sure the plot is displayed in this notebook
%matplotlib inline
# specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 6)

# for auto-reloading extenrnal modules
%load_ext autoreload
%autoreload 2
```

1. Introduction

This is the first part of a two week experiment in image processing. During this week, we will cover the fundamentals of digital monochrome images, intensity histograms, pointwise transformations, gamma correction, and image enhancement based on filtering.

In the second week , we will cover some fundamental concepts of color images. This will include a brief description on how humans perceive color, followed by descriptions of two standard color spaces. The second week will also discuss an application known as image halftoning.

2. Introduction to Monochrome Images

An image is the optical representation of objects illuminated by a light source. Since we want to process images using a computer, we represent them as functions of discrete spatial variables. For monochrome (black-and-white) images, a scalar function $f[i, j]$ can be used to represent the light intensity at each spatial coordinate (i, j). Figure 1 illustrates the convention we will use for spatial coordinates to represent images.

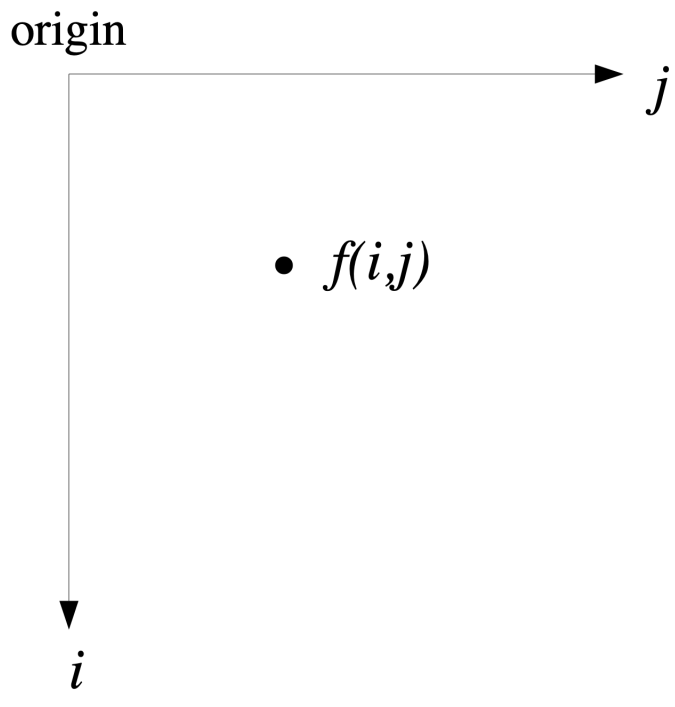

Figure 1: Spatial coordinates used in digital image representation

If we assume the coordinates to be a set of positive integers, for example $i = 0, \ldots, M-1$ and $j = 0, \ldots, N-1$, then an image can be conveniently represented by a matrix.

$$f[i, j] = \begin{bmatrix} f[0,0] & f[0,1] & \cdots & f[0,N-1] \\ f[1,0] & f[1,1] & \cdots & f[1,N-1] \\ \vdots & \vdots & \ddots & \vdots \\ f[M-1,0] & f[M-1,1] & \cdots & f[M-1,N-1] \end{bmatrix} \tag{1}$$

We call this an $M \times N$ image, and the elements of the matrix are known as *pixels*.

The pixels in digital images usually take on integer values in the finite range,

$$0 \leq f[i, j] \leq L_{\max} \tag{2}$$

where 0 represents the minimum intensity level (black), and $L_{\max}$ is the maximum intensity level (white) that the digital image can take on. The interval $[0, L_{\max}]$ is known as a gray scale.

In this lab, we will concentrate on 8-bit images, meaning that each pixel is represented by a single byte. Since a byte can take on 256 distinct values, $L_{\max}$ is 255 for an 8-bit image.

Exercise 2.1

In order to process images within Python, we need to first understand their numerical representation.

1. Load the image file `yacht.tif`, which contains an 8-bit monochrome image, using `plt.imread()` (https://matplotlib.org/3.5.0/api/_as_gen/matplotlib.pyplot.imread.html).

In [ ]:
```
# insert your code here
```

2. Display the type of this variable by printing its attribute `dtype` (https://numpy.org/doc/stable/reference/generated/numpy.ndarray.dtype.html).

In [ ]:
```
# insert your code here
```

Notice that the `A` matrix elements are of type `uint8` (unsigned integer, 8 bits). This means that Python is using a single byte to represent each pixel. Be careful that we should not perform numerical computation on numbers of type `uint8`, so we usually need to convert the matrix to a floating point representation, using the method `astype` (https://numpy.org/doc/stable/reference/generated/numpy.ndarray.astype.html).

3. Create a double precision representation of the image. Display the type of the matrix.

In [ ]:
```
# insert your code here
```

In future sections, we will be performing computations on our images, so we need to remember to convert them to type double before processing them.

4. Display `yacht.tif` using the following commands:

```
plt.imshow(B, cmap='gray', vmin=0, vmax=255)
plt.show()
```

In [ ]:
```
# insert your code here
```

The `plt.imshow()` command works for both type `uint8` and `double` images. The `cmap` argument specifies how the image is displayed. It is important to note that if any pixel values are outside the range `vmin` to `vmax` (after processing), they will be clipped to `vmin` or `vmax` respectively in the displayed image. `vmin` and `vmax` are two arguments of the function `plt.imshow()`. **It is necessary to set `vmin=0` and `vmax=255` to display a grayscale image. Without setting these two arguments, the `imshow` function tends to normalize the input matrix first, thus outputing a wrong grayscale image.**

It is also important to note that a floating point pixel value will be rounded down ("floored") to an integer before it is displayed. Therefore the maximum number of gray levels that will be displayed on the monitor is 255, even if the image values take on a continuous range.

Now we will practice some simple operations on the `yacht.tif` image.

5. Print the value of $f[35, 79]$ for this `yacht.tif` image.

In [ ]:
```
# insert your code here
```

6. Downsample this image by selecting every other row and column. Then, display it.

In [ ]:
```
# insert your code here
```

7. Make a horizontally flipped version of the image by reversing the order of each column. Then, display it.

In [ ]:
```
# insert your code here
```

8. Similarly, create a vertically flipped image. Display it.

In [ ]:
```
# insert your code here
```

9. Create a "negative" of the image by subtracting each pixel from 255. Then, display it.

In [ ]:
```
# insert your code here
```

10. Multiply each pixel of the original image by 1.5. Display it.

In [ ]:
```
# insert your code here
```

11. What effect did multiplying each pixel of the original image by 1.5 have?

insert your answer here

3. Pixel Distributions

3.1. Histogram of an Image

The histogram of a digital image shows how its pixel intensities are distributed. The pixel intensities vary along the horizontal axis, and the number of pixels at each intensity is plotted vertically, usually as a bar graph. A typical histogram of an 8-bit image is shown in Figure 2.

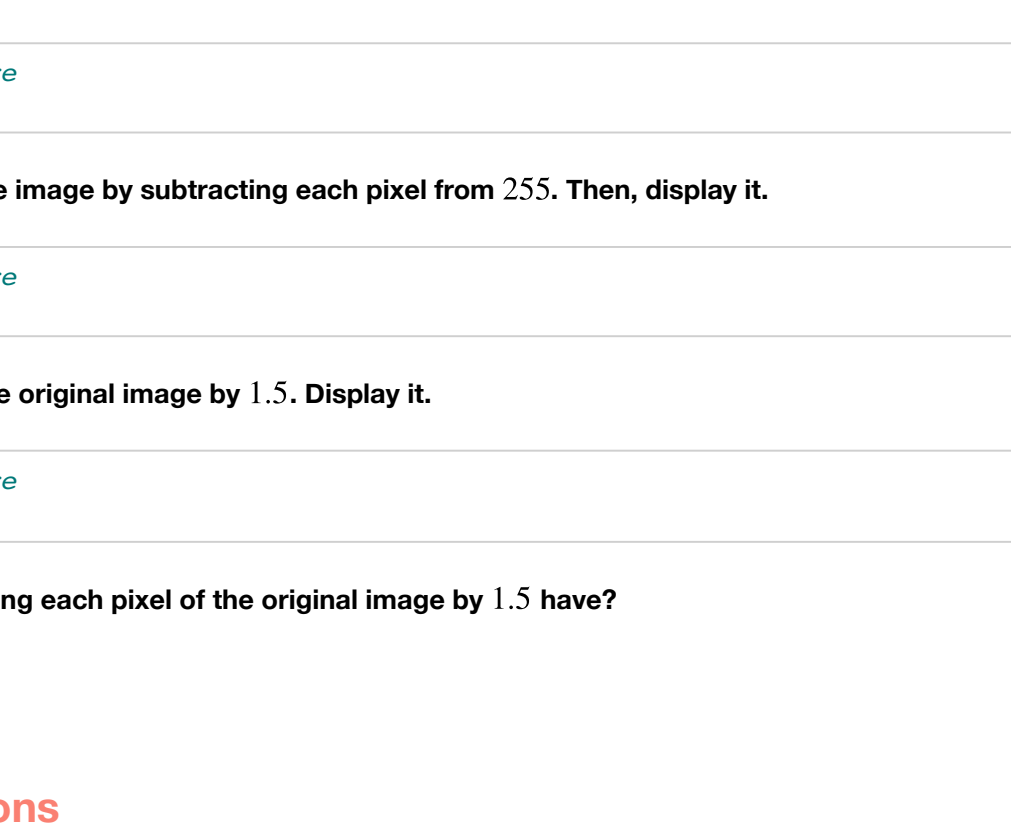

Figure 2: Histogram of an 8-bit image

Exercise 3.2: Histogram of an Image

1. Load the grayscale image `house.tif` and display it.

In [ ]:
```
# insert your code here
```

2. Plot the histogram of the image. Lable the axes of the histogram and give it a title.

Note: You may use `plt.hist()` (https://matplotlib.org/3.5.0/api/_as_gen/matplotlib.pyplot.hist.html) function. However, this function requires a vector as input. An example of using `plt.hist()` to plot a histogram of a matrix would be

```
plt.hist(x.reshape(-1), bins=np.arange(256))  # reshape(-1) reshapes the original multi-dimension a
rray to 1D
plt.title("title")
plt.xlabel("xlabel")
plt.ylabel("ylabel")
plt.show()
```

In [ ]:
```
# insert your code here
```

3.3. Pointwise Transformations

A pointwise transformation is a function that maps pixels from one intensity to another. An example is shown in Figure 3. The horizontal axis shows all possible intensities of the original image, and the vertical axis shows the intensities of the transformed image. This particular transformation maps the "darker" pixels in the range $[0, T_1]$ to a level of zero (black), and similarly maps the "lighter" pixels in $[T_2, 255]$ to

white. Then the pixels in the range $[T_1, T_2]$ are "stretched out" to use the full scale of $[0, 255]$. This can have the effect of increasing the contrast in an image.

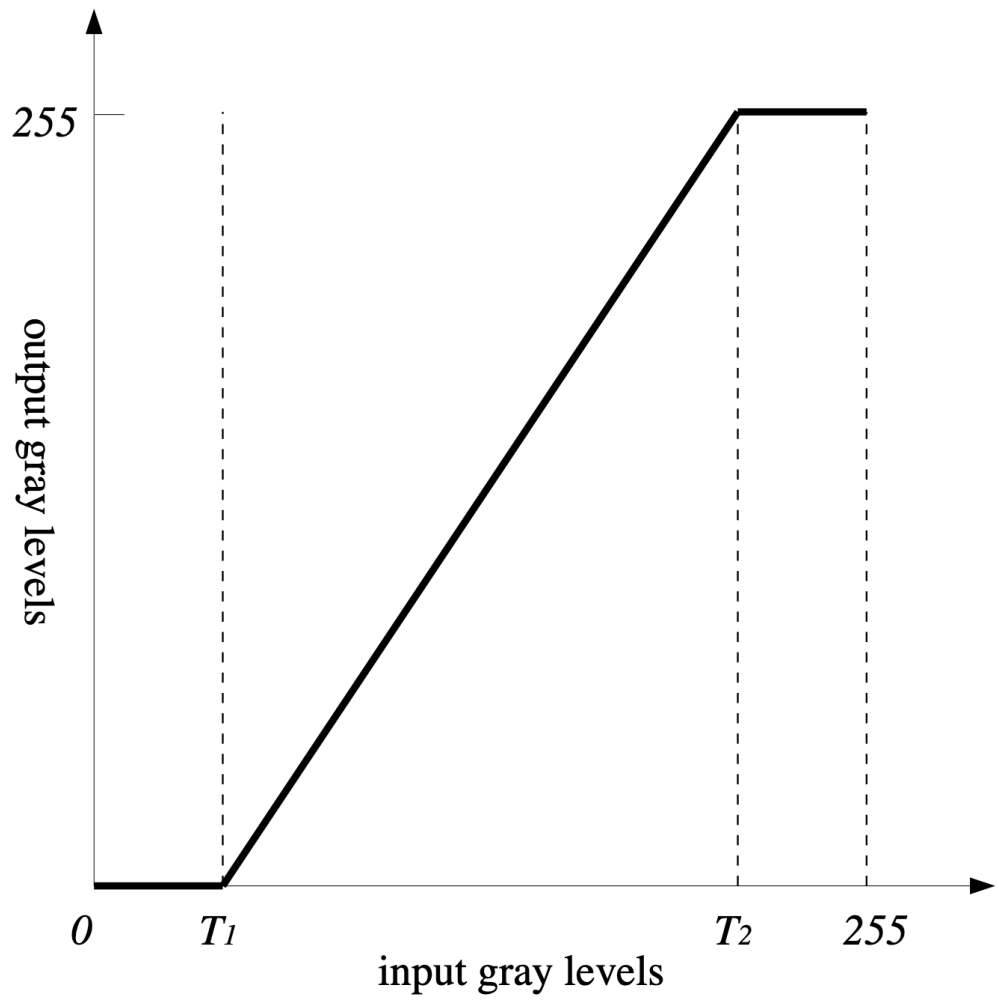

Figure 3: Pointwise transformation of image

Pointwise transformations will obviously affect the pixel distribution, hence they will change the shape of the histogram. If a pixel transformation can be described by a oneto-one function, $y = f(x)$, then it can be shown that the input and output histograms are approximately related by the following:

$$H_{\text{out}}(y) \approx \left. \frac{H_{\text{in}}(x)}{|f'(x)|} \right|_{x=f^{-1}(y)} \tag{3}$$

Since x and y need to be integers in (3), the evaluation of $x = f^{-1}(y)$ needs to be rounded to the nearest integer.

The pixel transformation shown in Figure 3 is not a one-to-one function. However, equation (3) still may be used to give insight into the effect of the transformation. Since the regions $[0, T_1]$ and $[T_2, 255]$ map to the single points 0 and 255, we might expect "spikes" at the points 0 and 255 in the output histogram. The region $[1, 254]$ of the output histogram will be directly related to the input histogram through equation (3).

First, notice from $x = f^{-1}(y)$ that the region $[1, 254]$ of the output is being mapped from the region $[T_1, T_2]$ of the input. Then notice that $f'(x)$ will be a constant scaling factor throughout the entire region of interest. Therefore, the output histogram should approximately be a stretched and rescaled version of the input histogram, with possible spikes at the endpoints.

Exercise 3.4: Pointwise Transformations

1. Complete the function below that will perform the pixel transformation shown in Figure 3.

Hints:

- Determine an equation for the graph in Fig. 3, and use this in your function. Notice you have three input regions to consider. You may want to create a separate function to apply this equation.
- If your function performs the transformation one pixel at a time, be sure to allocate the space for the output image at the beginning to speed things up.

In [ ]:
```
def pointTrans(x, T1, T2):
    """
    Parameters
    ---
    x: the input
    T1: the lower threshold
    T2: the upper threshold

    Returns
    ---
    y: the output
    """

    y = None
    return y
```

2. Load the image file `narrow.tif`. Display the image and its histogram.

In [ ]:
```
# insert your code here
```

3. Use your `pointTrans()` function to spread out the histogram using `T1 = 70` and `T2 = 180`. Display the new image and its

histogram.

```
In [ ]: # insert your code here
```

4. What qualitative effect did the transformation have on the original image? Do you observe any negative effects of the transformation?

insert your answer here

5. Compare the histograms of the original and transformed images. Why are there zeros in the output histogram?

insert your answer here

4. Gamma(γ) Correction

The light intensity generated by a physical device is usually a nonlinear function of the original signal. For example, a pixel that has a gray level of 200 will not be twice as bright as a pixel with a level of 100. Almost all computer monitors have a power law response to their applied voltage. For a typical cathode ray tube (CRT), the brightness of the illuminated phosphors is approximately equal to the applied voltage raised to a power of 2.5. The numerical value of this exponent is known as the gamma (γ) of the CRT. Therefore the power law is expressed as

$$I = V^{\gamma} \tag{4}$$

where I is the pixel intensity and V is the voltage applied to the device.

If we relate equation (4) to the pixel values for an 8-bit image, we get the following relationship,

$$y = 255\left(\frac{x}{255}\right)^{\gamma} \tag{5}$$

where x is the original pixel value, and y is the pixel intensity as it appears on the display. This relationship is illustrated in Figure 4.

In order to achieve the correct reproduction of intensity, this nonlinearity must be compensated by a process known as γ correction. Images that are not properly corrected usually appear too light or too dark. If the value of γ is available, then the correction process consists of applying the inverse of equation (5). This is a straightforward pixel transformation, as we discussed in Section 3.2.

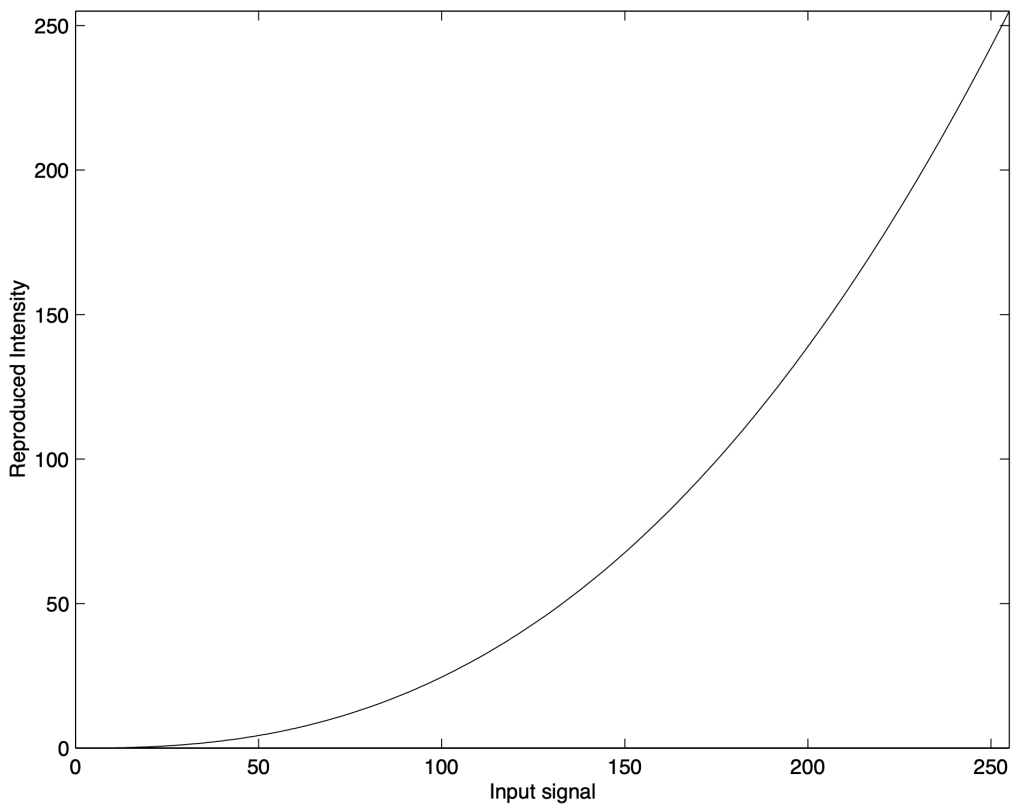

Figure 4: Nonlinear behavior of a display device having a γ of 2.2.

Exercise 4.1: Gamma(γ) Correction

1. Complete the function below that will γ correct an image by applying the inverse of equation (5).

```
In [ ]: def gammCorr(A, gamma):
            """
            Parameters
            ---
            A: the uncorrected image
            gamma: the gamma of the device

            Returns
            ---
            the corrected image
            """

            B = None
            return B
```

2. Load the image file `dark.tif` , which is an image that has not been γ corrected for your monitor. Display it and observe the quality of the image.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# make sure the plot is displayed in this notebook
%matplotlib inline
# specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 10)

# for auto-reloading extenrnal modules
%load_ext autoreload
%autoreload 2
```

1. Introduction

This is the second part of a two week experiment in image processing. In the first week , we covered the fundamentals of digital monochrome images, intensity histograms, pointwise transformations, gamma correction, and image enhancement based on filtering.

During this week, we will cover some fundamental concepts of color images. This will include a brief description on how humans perceive color, followed by descriptions of two standard **color spaces**. We will also discuss an application known as **halftoning**, which is the process of converting a gray scale image into a binary image.

2. Color Images

2.1. Background on Color

Color is a perceptual phenomenon related to the human response to different wavelengths of light, mainly in the region of 400 to 700 nanometers (nm). The perception of color arises from the sensitivities of three types of neurochemical sensors in the retina, known as the long (L), medium (M), and short (S) cones. The response of these sensors to photons is shown in Figure 1. Note that each sensor responds to a range of wavelengths.

Due to this property of the human visual system, all colors can be modeled as combinations of the three primary color components: red (R), green (G), and blue (B). For the purpose of standardization, the CIE (Commission International de l'Eclairage — the International Commission on Illumination) designated the following wavelength values for the three primary colors: $\text{blue} = 435.8\ \text{nm}$, $\text{green} = 546.1\ \text{nm}$, and $\text{red} = 700\ \text{nm}$.

The relative amounts of the three primary colors of light required to produce a color of a given wavelength are called *tristimulus values*. Figure 2 shows the plot of tristimulus values using the CIE primary colors. Notice that some of the tristimulus values are negative, which indicates that colors at those wavelengths cannot be reproduced by the CIE primary colors.

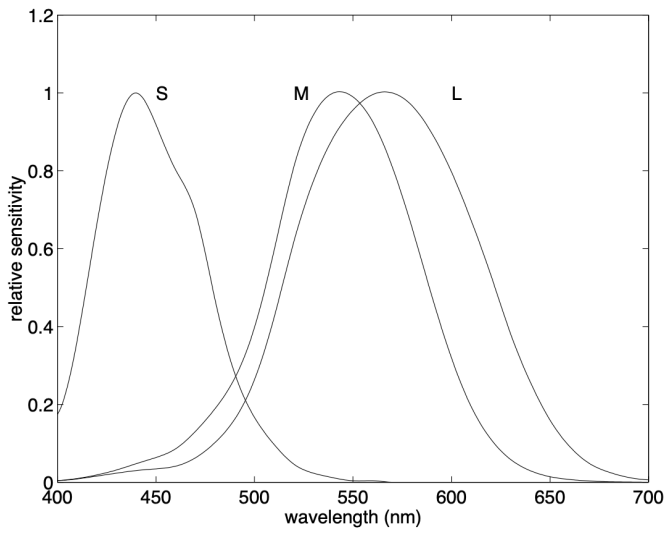

Figure 1: Relative photon sensitivity of long (L), medium (M), and short (S) cones.

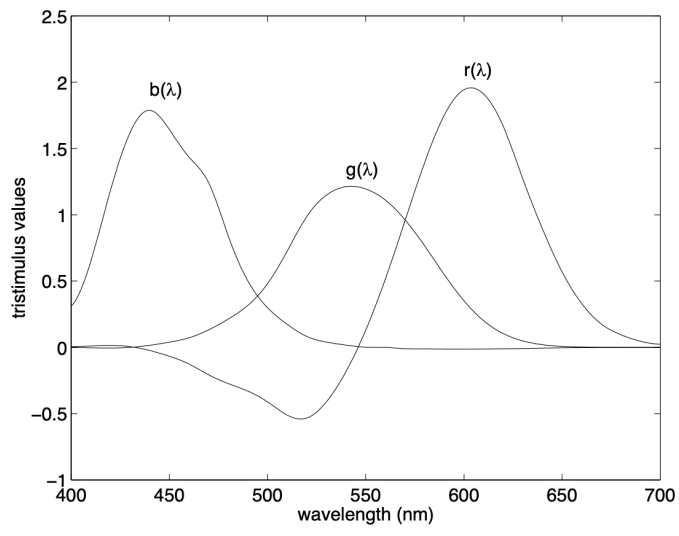

Figure 2: Plot of tristimulus values using CIE primary colors.

2.2. Color Spaces

A color space allows us to represent all the colors perceived by human beings. We previously noted that weighted combinations of stimuli at three

wavelengths are sufficient to describe all the colors we perceive. These wavelengths form a natural basis, or coordinate system, from which the color measurement process can be described. In this lab, we will examine two common color spaces: RGB and YC_bC_r. For more information, refer to [1].

- RGB space is one of the most popular color spaces, and is based on the tristimulus theory of human vision, as described above. The RGB space is a hardware-oriented model, and is thus primarily used in computer monitors and other raster devices. Based upon this color space, each pixel of a digital color image has three components: red, green, and blue.
- YC_bC_r space is another important color space model. This is a gamma corrected space defined by the CCIR (International Radio Consultative Committee), and is mainly used in the digital video paradigm. This space consists of luminance (Y) and chrominance (C_bC_r) components. The importance of the YC_bC_r space comes from the fact that the human visual system perceives a color stimulus in terms of luminance and chrominance attributes, rather than in terms of R, G, and B values. The relation between YC_bC_r space and gamma corrected RGB space is given by the following linear transformation:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= 0.564(B - Y) + 128 \\ C_r &= 0.713(R - Y) + 128 \end{aligned} \tag{1}$$

In YC_bC_r, the luminance parameter is related to an overall intensity of the image. The chrominance components are a measure of the relative intensities of the blue and red components. The inverse of the transformation in equation (1) can easily be shown to be the following:

$$\begin{aligned} R &= Y + 1.4025(C_r - 128) \\ G &= Y - 0.3443(C_b - 128) - 0.7144(C_r - 128) \\ B &= Y + 1.7730(C_b - 128) \end{aligned} \tag{2}$$

Exercise 2.3: Color

1. Load the image file `girl.tif`. Check the size of array for this image by using the command `print(image.shape)`, where `image` is the image matrix. Also, print the data type of this matrix.

Notice that this is a three dimensional array of type `uint8`. It contains three gray scale image planes corresponding to the red, green, and blue components for each pixel. Since each color pixel is represented by three bytes, this is commonly known as a 24-bit image.

In [3]:
```
# insert your code here
```

2. Display the image. Note that `cmap`, `vmin`, `vmax` arguments are not needed.

In [4]:
```
# insert your code here
```

3. Extract each of the color components, then plot each color component.

Note that while the original is a color image, each color component separately is a monochrome image, so plotting each color component requires `cmap`, `vmin`, `vmax` arguments.

In [5]:
```
# insert your code here
```

4. Load the files `ycbcr.npy` using `np.load()` (https://numpy.org/doc/stable/reference/generated/numpy.load.html), and print its type and data shape `dtype`.

This file contains a NumPy array for a color image in YC_bC_r format. The array contains three gray scale image planes that correspond to the luminance (Y) and two chrominance (C_bC_r) components.

In [6]:
```
# insert your code here
```

5. Plot each of the components.

In [7]:
```
# insert your code here
```

In order to properly display this color image, we need to convert it to RGB format.

6. Complete the function below that will perform the transformation of equation (2). It should accept a 3-D YC_bC_r image array as input, and return a 3-D RGB image array.

- Make sure `ycbcr` is in `double` or `float` before any processing.
- After conversion, to make sure the values of `rgb` are in $[0, 255]$, use `np.clip()` (https://numpy.org/doc/stable/reference/generated/numpy.clip.html).