

# **EE 569 Homework #1**

Maroof Mohammed Farooq

[maroofmf@usc.edu](mailto:maroofmf@usc.edu)

213-663-7996

# Table of Contents

<u>1. Problem 1</u> .....	3
<u>1.1 Abstract and Motivation</u> .....	3
<u>1.2 Approach and Procedures</u> .....	3
<u>1.2.1 Cropping and Resizing</u> .....	3
<u>1.2.2 Color Space Transformation</u> .....	4
<u>1.3 Experiment Results</u> .....	4
<u>1.4 Discussion</u> .....	7
<u>2. Problem 2</u> .....	9
<u>2.1 Abstract and Motivation</u> .....	9
<u>2.2 Approach and Procedures</u> .....	9
<u>2.2.1 Histogram Equalization for grayscale images</u> .....	9
<u>2.2.2 Histogram equalization for color images</u> .....	9
<u>2.2.3 Special Effects using contrast manipulation.</u> .....	10
<u>2.2.4 Histogram Transform</u> .....	10
<u>2.3 Experiment Results</u> .....	12
<u>2.4 Discussion</u> .....	15
<u>3. Problem3</u> .....	17
<u>3.1Abstract and Motivation</u> .....	17
<u>3.2Approach and Procedures</u> .....	17
<u>3.2.1Mix noise color Image</u> .....	17
<u>3.2.2 NLM Filter</u> .....	17
<u>3.2.3 Block matching and 3D transform filter</u> .....	18
<u>3.3 Experiment Results</u> .....	18
<u>3.4 Discussion</u> .....	21
<u>4. References</u> .....	23

# 1. Problem 1

## 1.1 Abstract and Motivation:

Cropping and Resizing are one of the most frequently used operations in image processing. The ability to crop images digitally gives the flexibility to remove unnecessary contents from the image. A simple technique to crop an image is explored in this problem.

In Image processing, image resizing or scaling is often performed to up-sample or down-sample images. Down-sampling images decreases the file size and is utilized for storing more number of images on a web server.

We will explore bilinear interpolation technique to resize an image. When an image is up sampled it introduces pixels with zero value. This introduces impulse noise and hence reduces the picture quality. To solve this problem, we interpolate the padded pixels from its neighbors.

Color spaces are a way to define the complete sub-set of colors in the visible spectrum. This type of color reproduction in digital images is possible by mixing different components of a color space. Fortunately, we have a variety of color spaces such as RGB, HSL, YCbCr, CMY and so on. Each color space has its own advantages that makes image processing simpler. Therefore, it is necessary to have the ability to transform an image to a color space that favors a particular processing step.

To support the above statement, we will look into the advantages of having CMY in printing process. In printing, we need to determine what colors should be injected onto the paper that would produce the desired effect. Using CMY in printing is advantageous as it uses subtractive color mixing to reproduce the desired color. This means that adding more ink makes the printed image darker as opposed to getting lighter when RGB is used.

In this problem we will implement and discuss about a simple subroutine that converts an RGB image to CMY and HSL color spaces.

## 1.2 Approach and Procedures:

There are two parts to this section. The methodology applied for cropping and resizing is described in 1.2.1 and later in 1.2.2 the methodology for color space transformation is described.

### 1.2.1 Cropping and Resizing

In order to successfully crop an image, we need to specify the top left coordinates, desired height and width of the cropped image. We use these coordinate values to copy the bounded data into a new image file.

An image can be resized into sizes that are smaller or larger than the original images. This is also known as resampling. When an image is resized to a smaller size then we say that the original image has been down sampled. Similarly, when an image is resized to a larger size then we say that the original image has been up sampled.

To resize an image, we first determine if the original image is down-sampled or up-sampled. We then create an image with all its pixels initialized to zero. In the case of down sampling, we simply map the pixel values of the resized image to that of the original image. This is done by pre-calculating the ratio of the resized and original image sizes. The mapped row and column index on the original image can be obtained by multiplying the ratio. Generally, the ratio is not a whole number which strongly suggests that the mapped row and column index are not whole

numbers. This will cause an error as indices cannot be in fraction. To solve this problem, we use the ceil function to round the indices to the nearest whole number. In this way, each pixel in the resized image is mapped to pixel in the original image.

In order to reconstruct an image which is up sampled, we need to map the pixels in the resized image to the original image. The resized image contains missing values as not every pixel in the up sampled image can be mapped to the original image. Hence we need to interpolate the missing values in the resized image. This can be done in several ways. In this problem we will explore bilinear interpolation to fill up the missing values.

Mathematically speaking, bilinear interpolation is used for interpolating a missing/unknown value in a 2D grid. This method interpolates the missing value with the help of the following formula:

$$P'_{r+a,c+b} = (1 - a)*(1 - b)*P_{r,c} + (a)*(1 - b)*P_{r+1,c} + (1 - a)*(b)*P_{r,c+1} + (a)*(b)*P_{r+1,c+1} \quad (1)$$

In the above formula,  $P'$  is the interpolated pixel value derived from pixels  $P$  in the original image. By observing the above formula we see how four neighboring pixels  $P$  in the original image located at rows given by  $r$  and columns given by  $c$  is utilized to obtain the interpolated pixel value. Each pixel  $P$  in the original image is multiplied with a weight given by parameters  $a$  and  $b$ . These parameters define how far the interpolated pixel is from the four neighboring pixels in the original image.

In C++ the above equation was implemented directly to obtain the resized image. The values of  $a$  and  $b$  were obtained by extracting the fractional part of the indices.

## 1.2.2 Color Space Transformation

The implementation of color space transformation is very straightforward. Firstly, we read a raw image with RGB values and store it in a vector container. To convert an image from RGB to CMY we normalize all the pixel values to a range of 0 to 1. This was done by dividing all the pixels by 255. After normalizing the values of each channel, we subtract the pixel values by 1. By finding the differences of all the pixels, the image is said to be transformed from RGB to CMY.

Similarly, to convert from RGB to HSL we first normalize all the pixel values for all the channels by dividing them by 255. Then for each pixel location we compute the maximum minimum of the normalized R,G and B values. We then compute the difference between the maximum and minimum values obtained. For each pixel location we calculate the Hue, saturation and lightness value by directly applying the formulas. After applying the formulas, we get the hue value between 0 to 360, lightness and saturation range between 0 to 1. In order to convert each channel to a grayscale image, we renormalize the values such that they lie between 0 to 255. For the hue values, this is achieved by multiplying (255/360) to the original hue value. By multiplying 255 to the lightness and saturation, we can transform their range to lie between 0 to 255.

## 1.3 Experiment Results:

- [Figure 1 displays the solution to problem 1\(a\) part 1](#)
- [Figure 2 displays the solution to problem 1\(a\) part 1](#)
- [Figure 3 displays the solution to problem 1\(a\) part 2](#)
- [Figure 4 displays the solution to problem 1\(a\) part 2](#)
- [Figure 5 displays the solution to problem 1\(b\) part 1](#)
- [Figure 6 displays the solution to problem 1\(b\) part 1](#)
- [Figure 7 displays the solution to problem 1\(b\) part 2](#)
- [Figure 8 displays the solution to problem 1\(b\) part 2](#)



(a) Original Image



(b) Cropped to 150x150



(c) Cropped to 180x144

Figure 1: Cropped images of Anna.



(a) Original Image



(b) Cropped to 150x150



(c) Cropped to 153x116

Figure 2: Cropped images of Rebel.



(a) Resized to 100x100



(b) Resized to 200x200

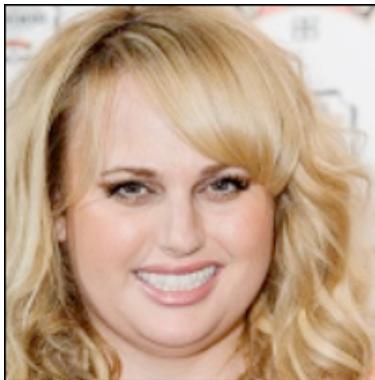


(c) Resized to 300x300

Figure 3: Resized images of Anna



(a) Resized to 100x100

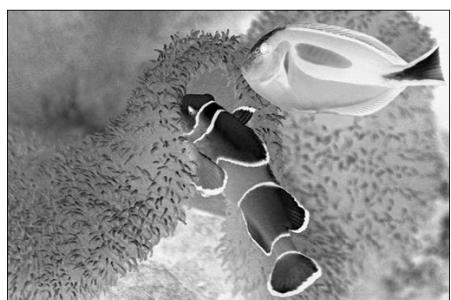


(b) Resized to 200x200

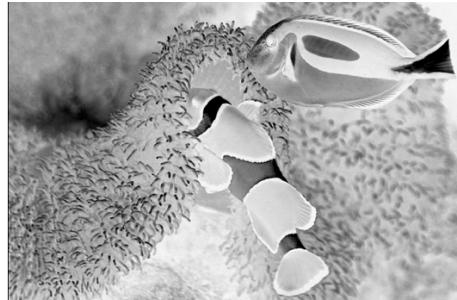


(c) Resized to 300x300

Figure 4: Resized images of Rebel



(a) Cyan Channel

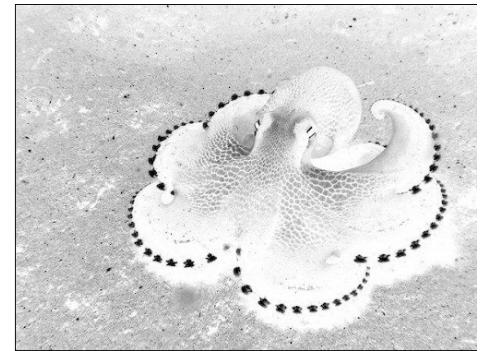
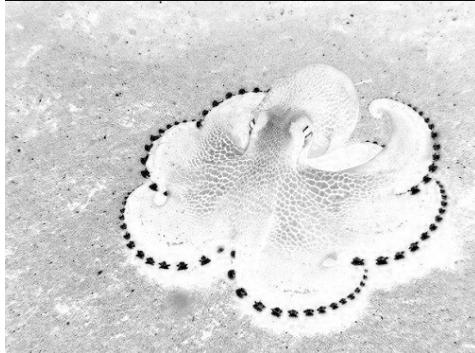
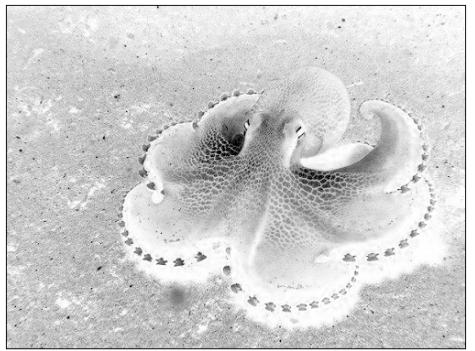


(b) Magenta Channel



(c) Yellow Channel

Figure 5: CMY Channels for clownfish image



(a) Cyan Channel

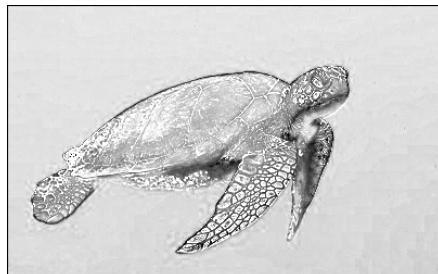
(b) Magenta Channel

(c) Yellow Channel

Figure 6: CMY Channels for octopus image



(a) Hue Channel

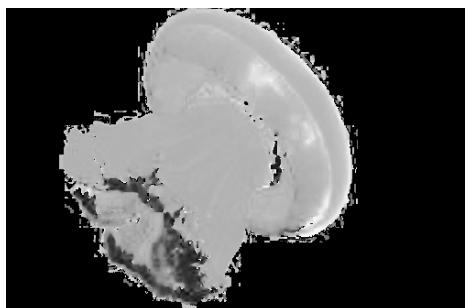


(b) Saturation Channel

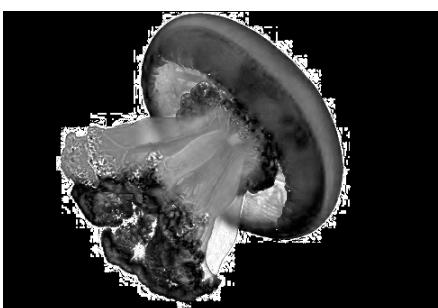


(c) Lightness Channel

Figure 7: HSL Channels for turtle image



(a) Hue Channel



(b) Saturation Channel



(c) Lightness Channel

Figure 8: HSL Channels for jellyfish image

## 1.4 Discussion:

**Cropping:** As seen in the [Figure 1](#) and [Figure 2](#), displays the output after cropping. As mentioned before, cropping is used to remove parts of the image that are not necessary. The quality of the image after cropping essentially remains the same but its size changes based on the crop height and width.

**Resizing:** In resizing the image, as seen in [Figure 3](#) and [Figure 4](#), we observe that there is a reduction in quality. When we down-sample the image, which is the case with 100x100 images, we are reducing a large number of samples that are essentially used to represent the image. In terms of signal processing, when we down sample a signal, we are essentially aliasing the high frequency component into the low frequency component. This makes the image to appear more pixelated. To solve this problem, we can pre-process the image to remove high frequency content and hence avoiding aliasing to get a better down sampled image.

When the image is up sampled then it introduces zero valued pixels that were interpolated by using bilinear interpolation. The bilinear interpolation method is simple to implement and it takes less time to compute the output. But this process deteriorates the sharpness of the image as bilinear interpolation removes high frequency components. Additionally, this process creates edge halos that can be seen clearly on 300x300 images. To improve the resizing process, we need to implement adaptive algorithms to perform resizing. These algorithms are designed to reduce artifacts that are caused due to resizing.

**Color Transformation from RGB to CMY:** The gray level images for each channel in the CMY image are given in [Figure 5](#) and [Figure 6](#). This color space is used in printing devices. The pixel intensity values range from 0 to 255 where 0 corresponds to black and 255 corresponds to white.

**Color Transformation from RGB to HSL:** The gray level images for each channel in the HSL image are given in [Figure 7](#) and [Figure 8](#). By closely inspecting each channel, we find out that they define different characteristics of the image. This color space is very popular in computer vision because it separates the color information from the image intensity.

The Lightness component of HSL captures the image intensity values. This value ranges from 0 to 100% with 0% as black and 100% as white.

The Hue component of HSL captures the color of the pixel. The value of hue ranges from 0 to 360 degrees. The black pixels in the hue components show that the pure color for the corresponding pixels is red. By slowly increasing the hue value we find that the corresponding pixels have a true color of orange, yellow, green and so on. In a nutshell, this channel contains the pure color information.

The saturation component of HSL captures the extent to which the hue component is diluted with white light. With values equal to zero suggesting that the color is not mixed with any white light and that pixel appears to be black. Incrementing the saturation value step by step, we can observe that the pure color becomes lighter as a result of dilution.

---

## 2. Problem 2

### 2.1 Abstract and Motivation

It is often the case that the pixel value distribution of an image gets concentrated to a narrow range. This can occur due to poor illumination and can deteriorate the quality of images. Fortunately for images that are shot in poor illumination can be fixed by contrast enhancement techniques.

Contrast enhancement changes the pixel value distribution to span a wider range. This can be achieved by various implementations. In this problem we will implement and discuss the transfer function based histogram equalization method and cumulative probability based histogram equalization method.

In the transfer function based implementation, our objective is to find a transfer function that maps the original pixel distribution to a desired distribution. In order to have the highest dynamic range we map the original pixel distribution to a uniform distribution.

Cumulative probability based histogram equalization method distributes the pixels are equally into 256 bins. The pixel values in each bin corresponds to the bin number. This implies that none of the bins are empty as long as number of pixels are greater than 256.

In this problem the above mentioned methodologies for contrast enhancement will be studied in detail. We will apply these techniques on greyscale and color images.

Histogram matching allows us to change the pixel value distribution to match a certain histogram. This operation is really powerful as it enables us to utilize the dynamic range of the image. In this problem we will study the algorithm for matching histograms and enhancing contrast.

### 2.2 Approach and Procedures

#### 2.2.1 Histogram Equalization for grayscale images

There were two methods implemented in this problem to equalize the histogram. First, we will discuss the implementation of transfer function based histogram equalization and then we will move on to cumulative probability based histogram equalization.

**Transfer function based:** The main purpose of the transfer function based histogram equalization is to transfer the gray levels so that the histogram is uniform. To achieve this, we map the values of the target image using a transfer function such that the pixels follow a uniform distribution. Firstly, the probability density function of pixel values is calculated. A hash map is used to store the probability of occurrence of a pixel value. The pdf of a pixel value is given by the following:

$$pdf[p] = \frac{\text{Total number of pixels with pixel value } p}{\text{Total number of pixels}} \quad (2)$$

The above equation gives the pdf of pixel with value  $p$  and is stored in the hash map. After this calculation, the pdf values are utilized to calculate the CDF values. The CDF values that are obtained is our transfer function. We utilize the obtained transfer function to map the pixel values to CDF of uniform distribution. This is done by implementing the following formulae:

$$I_{r,c} = CDF[I_{r,c}] * 255 \quad (3)$$

Here  $I_{r,c}$  is the pixel value corresponding to row r and column c of the image. The CDF values range between 0 to 1. To renormalize the pixel values, we multiply the obtained CDF value by 255.

**Cumulative probability based:** The transfer function based histogram equalization introduces artificial contours in the histogram as seen in [Figure 10 \(b\)](#) and [Figure 13\(b\)](#). To avoid such artificial contours, we implement the cumulative probability based histogram equalization.

To implement this method, we need to build a vector container that has all the pixel locations based on sorted pixel values. To realize this, we first initialize a hash table that maps the pixel values to its locations. The locations corresponding to a pixel value is stored in a vector container. Then we concatenate all the pixel locations into one vector container by looping through the hash table that we initialized earlier. Then we calculate the number of pixels per bin by using the following formula:

$$\# \text{pixels per bin} = \frac{\text{Total number of pixels}}{255} \quad (4)$$

In the final step, we distribute the pixels uniformly into each bin that were sorted based on pixel values. Finally, we set the pixel values equal to the bin number to obtain the equalized image.

## 2.2.2 Histogram equalization for color images

The concept of histogram equalization on color images are similar to that discussed in the previous section. But there is an additional processing step before implementing histogram equalization. Since we are dealing with color images, we need to separate each channel and equalize it separately. Each channel is converted into a gray scale image and the procedures followed in the previous section were applied. The equalized channels were then concatenated to get the equalized color image.

## 2.2.3 Special Effects using contrast manipulation

The flexibility to change the histogram distribution of a grayscale image allows us to implement special effects on colored images. By fine tuning the distribution of the color channels, we can obtain an image with a desired special effect. In this problem we will implement an algorithm that applies the special effect of the Skyline image to any original image.

Conceptually, we need to match the histogram of each channel of the original image with the corresponding channel of skyline image. The algorithm used for implementing this is very similar to the transfer function method discussed in the previous sections. We develop a transfer function that maps the pixel distribution of the original image to the pixel distribution of the skyline image.

Firstly, we calculate the pdf and CDF values for each channel of the images. These values are stored in a hash map which has pixel values as the keys that links to the corresponding CDF value. To match the pixel distribution, we map the pixel values based on its CDF. The CDF value of each original pixel is obtained from the hash table and replaced with the pixel value with the same CDF in the skyline image. Doing this for each channel separately produces the same effect as observed in the skyline image.

## 2.2.4 Histogram Transform

The histogram transformation is done by using the cumulative probability based approach. The CDF of the original image and Gaussian distribution are calculated and stored in hash maps. At this point it is important to note that the maximum CDF value doesn't equal to 1. This is due to the fact that Gaussian distribution is defined in the interval  $[-\infty, +\infty]$ . By observation it was found that the maximum CDF value was equal to 0.998596. According to the general rule of probability, the maximum CDF value should be equal to 1. Hence, we need to normalize the CDF values before applying any transformation. Normalization for this problem can be achieved by dividing CDF value for each pixel by the maximum CDF obtained. This ensures that we get maximum CDF value of 1.

The calculation for number of pixels per bin is done by first finding the pdf values from its CDF values. By performing the normalization stated above we can strongly conclude that the pdf values sums up to 1. To get the number of pixels per bin we multiply the pdf with the total number of pixels in the image.

Knowing the number of pixels per bin and the location of pixels sorted based on pixel values, we can follow the cumulative probability based approach as discussed in [Section 2.2.1](#).

## 2.3 Experiment Result



(a) Contrast Enhancement using Transfer Function

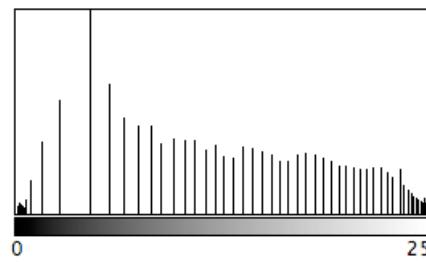


(b) Contrast Enhancement using CDF method

Figure 9: Contrast enhancement of beach bright image



(a) Original Image



(b) Transfer function method



(c) CDF method

Figure 10: Histogram of beach bright image

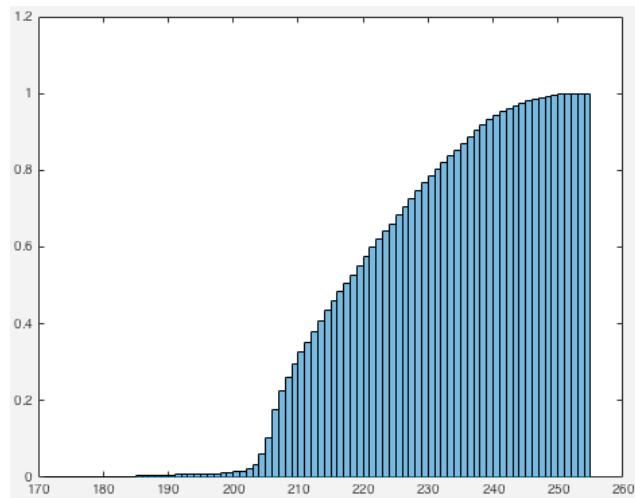


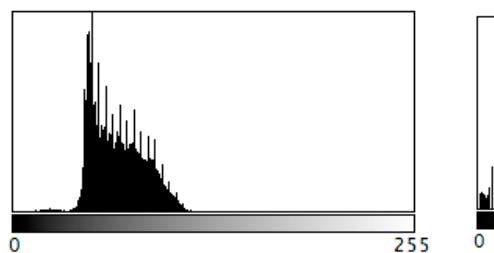
Figure 11: Transfer function of beach bright image



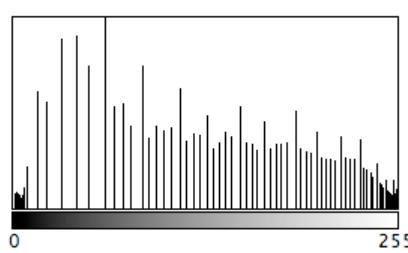
(a) Contrast Enhancement using Transfer Function

(b) Contrast Enhancement using CDF method

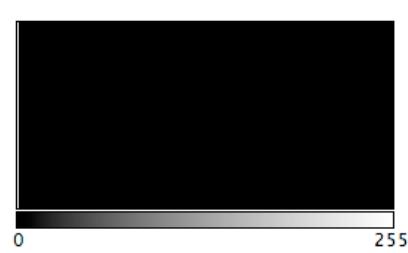
Figure 12: Contrast enhancement of beach dark image



(a) Original Image



(b) Transfer function method



(c) CDF method

Figure 13: Histogram of beach dark image

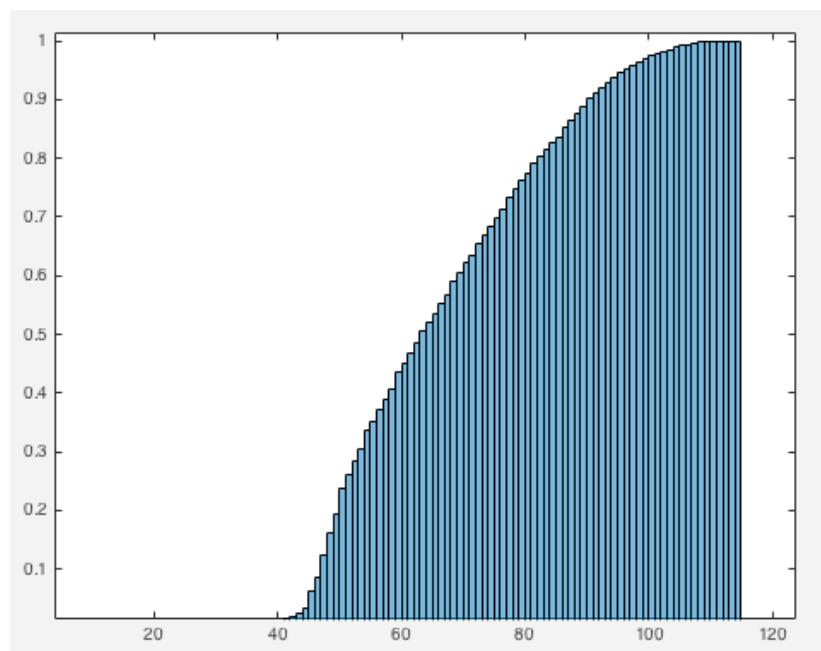


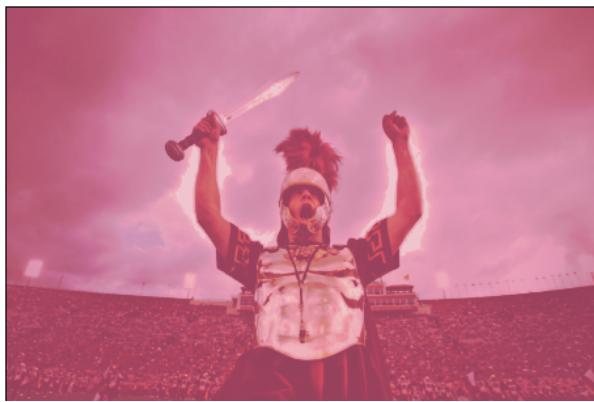
Figure 14: Transfer function of beach dark image



(a) Contrast Enhancement using Transfer Function

(b) Contrast Enhancement using CDF method

Figure 15: Contrast enhancement of skyline image



(a) Trojan

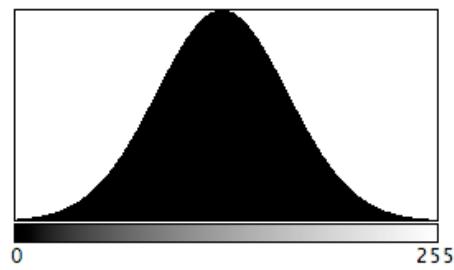


(b) Park

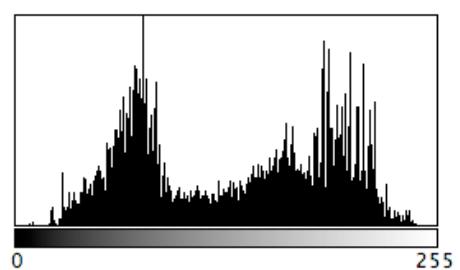
Figure 16: Applying special effects via contrast manipulation



(a) Output Image



(b) Histogram of the output image

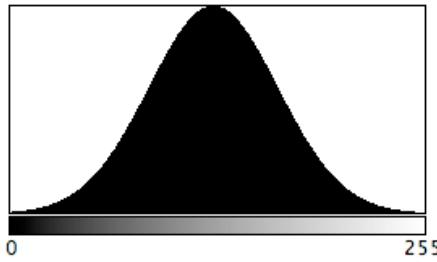


(c) Histogram of the input image

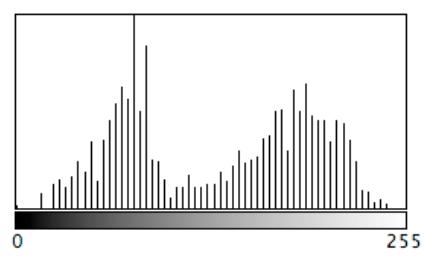
Figure 17: Histogram transformation for student 1



(a) Output Image



(b) Histogram of the output image



(c) Histogram of the input image

Figure 18: Histogram transformation for student 2

## 2.4 Discussion

In this problem we implemented methods to enhance the contrast and manipulate the histogram of an image to a pre defined distribution. These methods are simple but effective in achieving the task.

The main objective of the transfer function based histogram equalization was to improve the dynamic range of the image by uniformly distribution the pixels values. Figures [10\(b\)](#) and [13\(b\)](#) shows the histogram of the beach images after contrast enhancement. We can see that it has many pixels values that are not utilized. This causes the image to be pixelated as seen in [figure 9\(a\)](#) and [12\(a\)](#). Apart from the disadvantages, this method is simple to implement and is faster than other methods in terms of computational complexity. Additionally, it is a one to one mapping and hence it is consistent.

Applying the cumulative probability based histogram equalization forces all the pixel to accurately follow the desired distribution. The output image produced is generally smooth. The implementation of this method is slightly complicated as the location of pixels should be stored in order of their pixel values. The major disadvantage of this method is that the mapping is not consistent.

Both the methods suffer drawbacks which reduces the overall quality of images. The incapability to differentiate between various pixels causes distortion in the images. Methods like adaptive histogram equalization uses several histograms of various parts of the image and then enhances it.<sup>[\[1\]](#)</sup> A well rounded review of alternative histogram equalization algorithms are given in [\[2\]](#).

Applying histogram equalization to each channel of the color image independently causes distortion in the image as seen in [figure 15](#). This is because the distribution of all the channels are different as see in figure [19](#), [20](#) and [21](#). A better approach to this problem is to convert the image to HSL or YCbCr color spaces that separates the Luma and Chroma contents of an image. Then we can effectively apply histogram equalization to the luma/brightness channel to retain the color information but changing the saturation and brightness of the image.

When any special effects are desired on an image based on a reference image, we can perform histogram matching. Looking at the images after histogram matching in [figure 16](#), we can see that the special effects have not been captured accurately. The reddish white pixels near the Trojans arms are caused due to color distortion. We can solve this by processing the image in a different color space.

From the histogram of the output image in figures [17](#) and [18](#) we can see that the pixel density is centered around the desired mean. Due to normalization, it so happens that the pixel value of 255 in the input image gets mapped to 255 in the output regardless of the mean or standard deviation of the distribution. This gives less flexibility in changing the dynamic range of the image and the image loses its smoothness.

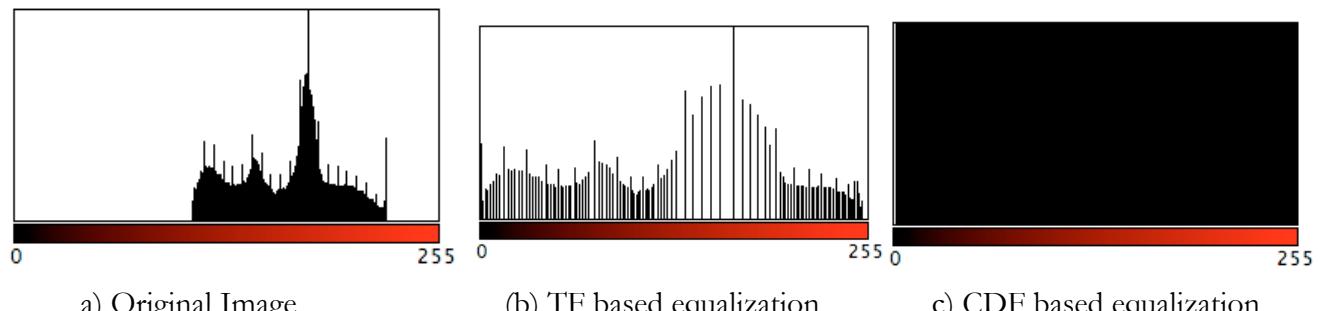


Figure 19: Histogram of red channel in skyline image

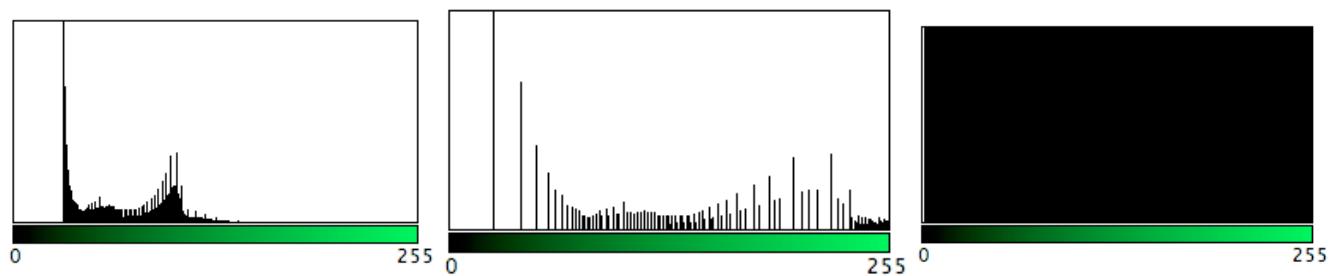


Figure 20: Histogram of green channel in skyline image

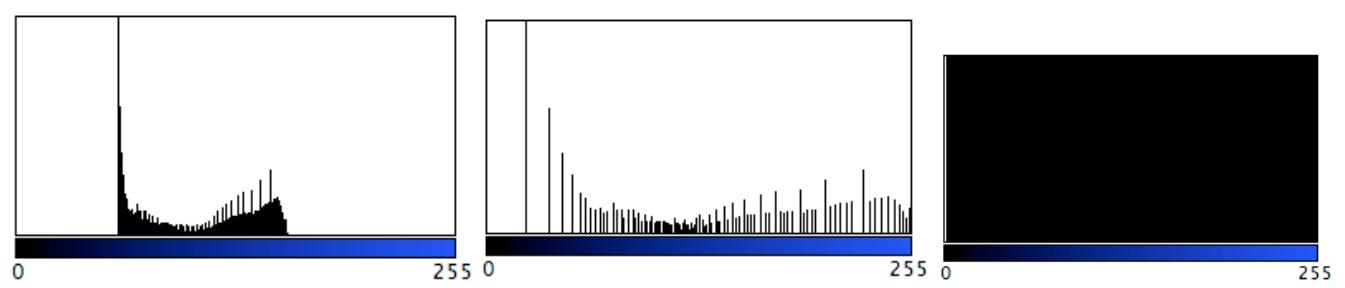


Figure 21: Histogram of blue channel in skyline image

# **Problem 3**

## **3.1 Abstract and Motivation**

All kinds of signals face the problem of noise. Noise is a random variations in magnitude of a signal that deteriorates the signal quality. In image processing, noise is induced due to various factors such as dead sensors, thermal noise, poor illumination, variation in color information and so on. These factors induce Gaussian noise, salt and pepper noise into the images.

In this problem, we will study and implement different methods for noise removal in images like mean filter, median filter, Non local means filter and BM3D filter. We will evaluate the performance of each filter and visualize the effects of changing filter parameters and cascading the abovementioned filters.

Filtering is performed on different channels separately. A moving 2D window is used to capture the pixel values and perform filtering operation. In median filter, the center value of the 2D window is replaced by the median of all the values in the window. Similarly for the mean filter, the center value is replaced with the mean of all the values in the window.

Since we consider the center value of the 2D window, it is convenient to choose an odd number for window size. Commonly used window sizes are 3x3, 5x5 and 7x7. In this problem, we will make an attempt to use analyze the effects on window sizes.

The performance of each filter or set of filters will be based on the Peak Signal to Noise Ratio (PSNR). Ideally, the PSNR ratio for a noise free image is  $\infty$  dB. Lower PSNR values indicate presence of high noise content.

## **3.2 Approach and Procedures**

### **3.2.1 Mix noise color Image**

The basic step of applying a filter to de-noise the image involves creating a window that moves through the image. The moving window captures a small part of the pixels and combines them to calculate the output. This output is written to the new image.

Before applying the mean or median filter, the image is extended to avoid indexing errors. The image is extended by a factor of  $\text{ceil}((\text{window size})/2)$ . The window is traversed through the image and it replaces the pixel value in the output image by calculating the mean/median of the window values.

### **3.2.2 NLM Filter**

The NLM filter is a powerful image de-noising tool. This technique is used to de noise images by taking three parameters. They are window size, search window size and sigma value (gives the rate of decay). Firstly, the image is extended by  $\text{ceil}((\text{search window size})/2)$  pixels. This is done to ensure that the indices don't go out of range.

Value of each pixel in the output image is determined by calculating the average of similar pixels surrounding the pixel of interest. Each surrounding window of pixels are assigned a weight that corresponds to the correlation with the window that contains the pixel of interest. The weights are normalized such that all the weights sum up to 1. The neighboring pixels contributes towards deciding the value of the pixel. This is because the random

noise component gets added to the window with pixel of interest. This reduces the standard deviation of the random noise up to 10 times.

### 3.2.3 Block matching and 3D transform filter

The block matching and 3D transform filter was studied using the code that is provided online<sup>[3]</sup>. The values of sigma were varied and the corresponding PSNR values were recorded in [Table 1](#).

## 3.3 Experiment Results

Filter	PSNR (Lena)	PSNR (Buildings)	PSNR(Trees)
No filter	18.56	27.06	22.41
Mean(3x3)	25.24	26.97	21.70
Mean(5x5)	25.65	23.70	19.78
Mean(7x7)	25.06	22.04	18.84
Median(3x3)	25.53	27.55	21.52
Median(5x5)	26.53	24.57	19.83
Median(7x7)	26.20	22.80	18.91
Mean(3x3) + Median(3x3)	22.0072	18.57	17.15
Median(3x3) + Mean(3x3)	26.78	26.02	21.18
Mean(5x5) + Median(3x3)	25.77	23.65	19.74
Median(5x5) + Mean(5x5)	26.20	23.01	19.41
NLM (Search Window:11, Window:3, h: 100)	25.93	27.76	23.59
Median(3x3) + NLM(11,3,100)	27.16	25.43	20.61
NLM (13,3,100)	26.043	27.55	23.42
BM3D (sigma = 10)	20.65	25.81	24.61
BM3D (sigma = 30)	26.17	30.97	25.22
BM3D (sigma = 50)	27.79	28.42	22.18

Table 1: PSNR values for different filters

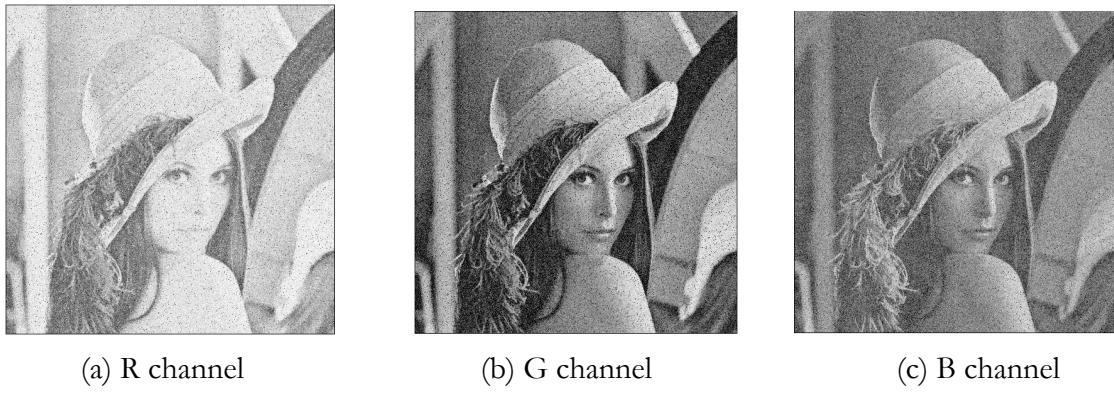


Figure 22: Noisy Lena image



(a) R channel



(b) G channel



(c) B channel

Figure 23: Noisy Building image



(a) R channel



(b) G channel



(c) B channel

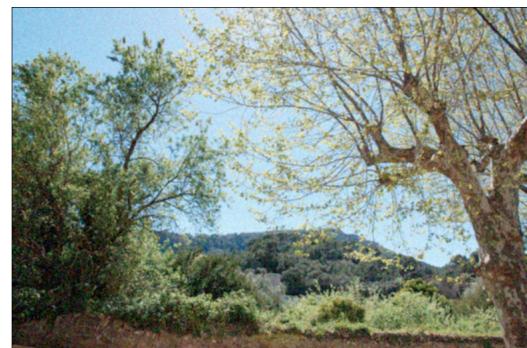
Figure 24: Noisy Tree image



(a) Lena

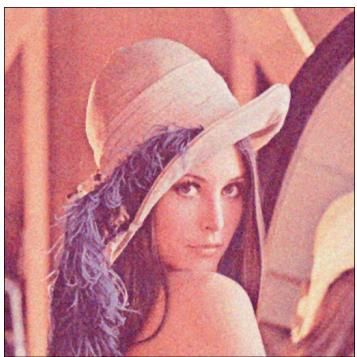


(b) Buildings



(c) Trees

Figure 25: Mean filter with window size 3x3



(a) Lena



(b) Buildings



(c) Trees

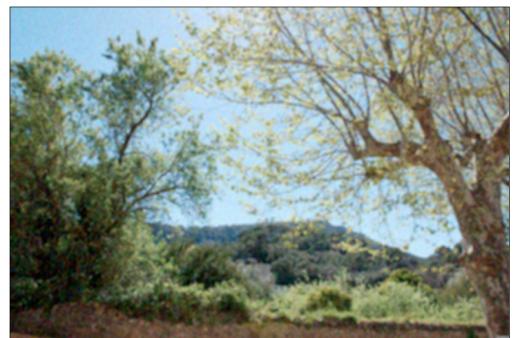
Figure 26: Median filter with window size 3x3



(a) Lena

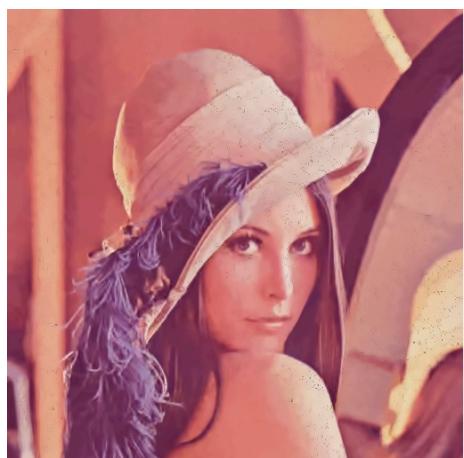


(b) Buildings



(c) Trees

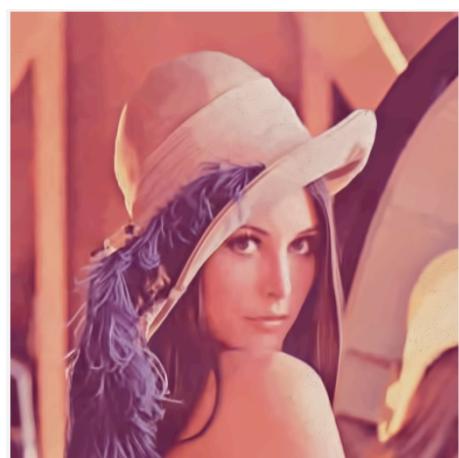
Figure 27: Median filter and Mean filter with window size 3x3



(a) sigma = 40



(b) sigma = 50



(c) sigma = 60

Figure 28: BL3D Filter on Lena image



(a) Lena image

(b) Buildings

(c) Trees

Figure 29: NLM Filter



(a) Lena image

(b) Buildings

(c) Trees

Figure 30: Median filter followed by NLM Filter

### 3.4 Discussion

As observed in figures Figures 22, 23 and 24, we observe that the individual channels are filled with salt and pepper noise. The red channel is filled with salt noise, green channel with pepper noise and blue channel with salt and pepper noise. In addition to salt and pepper noise, the images contain Gaussian noise.

Since the noise is random and independent of the channel or location of pixels, we can apply the filters to each channel individually. From table 1, we can see that the PSNR value improves when median filter is applied first and then the mean filter. This is because the image contains salt and pepper noise. Median filter is best at removing salt and pepper noise. Then we can apply a mean filter that will help remove Gaussian noise and not get affected by outliers caused by salt and pepper noise.

The window size alters the sharpness of the image. Larger window causes the image to lose its higher frequency content thus affecting the quality of the image.

NLM filter gives appreciable results for the test images. Random noise gets eliminated due to non local means. If we observe Lena's image carefully, we find that salt and pepper noise is still there. With the application of median filter and NLM filter, the PSNR value for Lena image boosted from 25.93 to 27.16.

The major disadvantage that NLM suffers is from its computational complexity. Looking at the advantages, NLM preserves the structure of the image and removes random noise without any compromise in quality of the image.

---

## **References :**

[1] [https://en.wikipedia.org/wiki/Adaptive\\_histogram\\_equalization](https://en.wikipedia.org/wiki/Adaptive_histogram_equalization)

[2] Nityananda CR, Ramachandra AC, Preethi “Review on Histogram Equalization based Image Enhancement Techniques.”

[3] BM3D: <http://www.cs.tut.fi/~foi/GCF-BM3D/>