ASSIGNMENT - 1

An assignment

Submitted in fulfillment of the Completion of

the course 19CCE447 Image Processing under the

Faculty of Electronics and Communication

Engineering

By:

SANTOSH [CB.EN.U4CCE20053]

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE CAMPUS

FACULTY IN-CHARGE: SUGUNA G.

ASSIGNMENT 1 – IMAGE ENHANCEMENT

AIM:

To explore different methods for improving the visual quality of digital images.

SOFTWARE:

Product version: Anaconda3 2021.11	Qt version: 5.9.7
Spyder version: 5.1.5	PyQt5 version: 5.9.2
Python version: 3.9.7 64-bit	Operating System: Windows 10

THEORY AND PYTHON CODES:

I. Spatial Filters – Averaging Filter and Median Filter in Image Processing

Spatial filters are used in image processing to modify the pixel values of an image based on its spatial properties. Averaging filter and median filter are two widely used spatial filters in image processing.

The averaging filter is a linear filter that replaces the pixel value with the average value of its neighbouring pixels within a specified window size. The averaging filter is a low-pass filter that is used to smooth an image, reducing the noise present in the image. A larger window size leads to more smoothing, while a smaller window size leads to less smoothing. The averaging filter is especially useful for removing high-frequency noise, such as Gaussian noise, which affects the high-frequency components of the image.

The median filter, on the other hand, is a non-linear filter that replaces the pixel value with the median value of its neighbouring pixels within a specified window size. The median filter is particularly useful for removing impulse noise, such as salt and pepper noise, which corrupts the pixel values randomly. The median filter preserves the edges and fine details in an image, making it a popular choice for denoising images.

Both filters can be implemented using convolution, which involves sliding a filter window over each pixel in the image and computing the weighted sum of the pixel values in the window. The averaging filter and the median filter can be implemented using a box filter and a median filter respectively. A box filter computes the average value of pixels in the filter window, while a median filter computes the median value of the pixels. The box filter and the median filter can be implemented efficiently using spatial domain techniques, such as integral images, which can speed up the computation time significantly.

In summary, the averaging filter and the median filter are two popular spatial filters used in image processing. The choice of filter depends on the type of noise present in the image and the desired level of smoothing required. The filters can be implemented using convolution and spatial domain techniques, such as integral images, to improve computational efficiency.

Q. Write a program in Python to implement a spatial domain averaging filter and to observe its blurring effect on the image.

This Python program is implementing a spatial domain averaging filter to observe its blurring effect on an image. It uses the OpenCV package to read an image and the NumPy package to create a 3X3 averaging filter mask. The program then convolves this mask over the image using nested loops and stores the output in a new image. Finally, the output image is converted to 8-bit unsigned integers and saved to a storage device using the cv2.imwrite() function.

```
# Low Pass Spatial Domain Filtering to observe the blurring effect

import cv2 # Wrapper package for OpenCV python bindings
import numpy as np # General-purpose array-processing package

img = cv2.imread('Lighthouse.jpg', 0) # Read the image
m, n = img.shape # Obtain the number of rows and columns of the image
img_new = np.zeros([m, n]) # Convolve the 3X3 mask over the image

# Develop Averaging filter (3, 3) mask
mask = np.ones([3, 3], dtype = int)
mask = mask/9

for i in range(1, m-1):
    for j in range(1, n-1):

        temp = img[i-1, j-1]*mask[0, 0] + img[i-1, j]*mask[0, 1] +
img[i-1, j + 1]*mask[0, 2] + img[i, j-1]*mask[1, 0] + img[i, j]*mask[1,
1] + img[i, j + 1]*mask[1, 2] + img[i + 1, j-1]*mask[2, 0] + img[i + 1,
j]*mask[2, 1] + img[i + 1, j + 1]*mask[2, 2]

        img_new[i, j]= temp

img_new = img_new.astype(np.uint8) # Convert the data type of the
output image to uint8
cv2.imwrite('Averaging Filter - Lighthouse.jpg', img_new) # Save an
image to any storage device
```

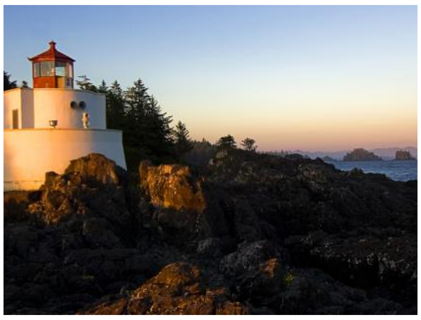
Figure 1. Original Image

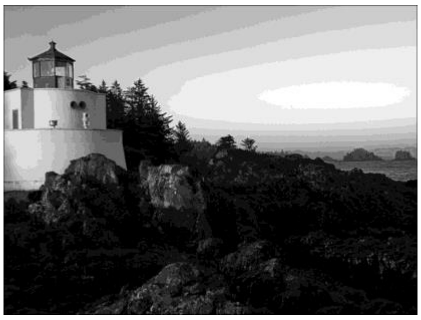
Figure 2. Image After Averaging Filter

Q. Write a program in Python to implement a spatial domain median filter to remove salt and pepper noise.

```
import cv2 # Wrapper package for OpenCV python bindings
import numpy as np # General-purpose array-processing package

img_noisy = cv2.imread('Lighthouse.jpg', 0) # Read the input image
m, n = img_noisy.shape # Obtain the number of rows and columns of the
image
img_new = np.zeros([m, n]) # Initialize a new image with zeros

# Traverse the image and apply the median filter to each 3x3
neighborhood
for i in range(1, m-1):
    for j in range(1, n-1):

        # Get the nine pixels in the 3x3 neighborhood
        temp = [img_noisy[i-1, j-1],
            img_noisy[i-1, j],
            img_noisy[i-1, j + 1],
            img_noisy[i, j-1],
            img_noisy[i, j],
            img_noisy[i, j + 1],
            img_noisy[i + 1, j-1],
            img_noisy[i + 1, j],
            img_noisy[i + 1, j + 1]]

        temp = sorted(temp) # Returns a sorted list of the specified
iterable object
        img_new[i, j]= temp[4] # Select the median value

img_new = img_new.astype(np.uint8) # Convert the data type of the
output image to uint8
cv2.imwrite('Median Spatial Domain Filtering - Lighthouse.jpg',
img_new) # Save an image to any storage device
```

Figure 3. Original Image

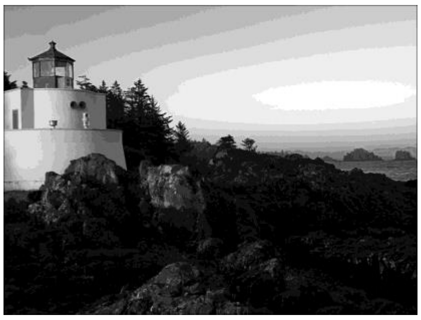
Figure 4. Image After Median Spatial Domain Filtering

II. Image Masking

```
import cv2 # Wrapper package for OpenCV python bindings
import numpy as np # General-purpose array-processing package

img = cv2.imread('Lighthouse.jpg') # Read the input image
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # Convert BGR to HSV

# Define range of yellow color in HSV
lower_yellow = np.array([15,50,180])
upper_yellow = np.array([40,255,255])

# Create a mask
rectangular_mask = np.zeros(img.shape[:2], np.uint8)
rectangular_mask[100:250, 150:450] = 255

mask_track = cv2.inRange(hsv, lower_yellow, upper_yellow) # Create a
mask; Threshold the HSV image to get only yellow colors
yellow_result = cv2.bitwise_and(img, img, mask= mask_track) # Bitwise-
AND mask and original image
rectangular_masked_img = cv2.bitwise_and(img, img, mask =
rectangular_mask) # Compute the bitwise AND using the mask

# Save an image to any storage device
cv2.imwrite('Mask Tracking.jpg', mask_track)
cv2.imwrite('Yellow Masked Image.jpg', yellow_result)
cv2.imwrite('Rectangular Mask.jpg', rectangular_mask)
cv2.imwrite('Rectangular Masked Image.jpg', rectangular_masked_img)
```

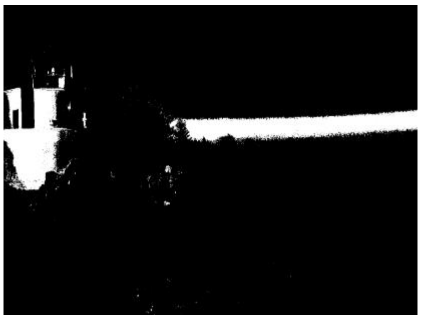
Figure 5. Mask Tracking

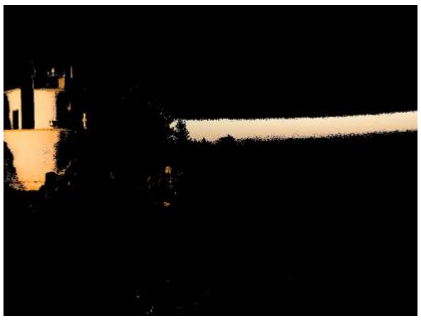
Figure 6. Yellow Masked Image

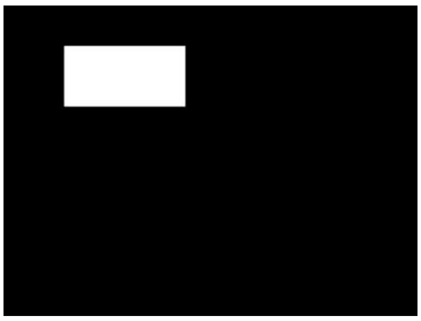
Figure 7. Rectangular Mask

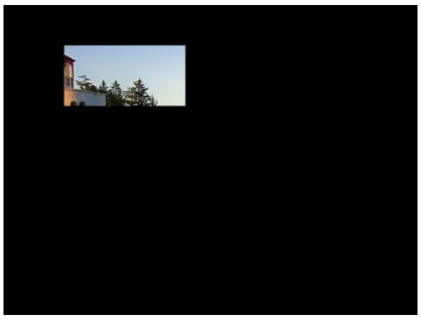
Figure 8. Rectangular Masked Image

III. Point Processing

Point Processing is a fundamental technique in digital image processing that involves manipulating individual pixels in an image. The three-point processing techniques we will discuss are Image Negative, Image with Thresholding, and Image with Grey Level Slicing with Background.

Image Negative is a point processing technique that involves inverting the intensity values of an image. This technique is useful for enhancing the details of an image that may be difficult to see in the original image. By inverting the intensity values, the darker areas of the image become lighter, and the lighter areas become darker.

Image Thresholding is a point processing technique that involves setting a threshold value to separate the pixels of an image into two groups: those that have intensity values above the threshold and those that have intensity values below the threshold. This technique is useful for separating the foreground and background of an image or for identifying specific objects in an image.

Image Grey Level Slicing with Background is a point processing technique that involves highlighting a specific range of intensity values in an image. This technique is useful for highlighting specific features in an image that fall within a certain intensity range. The background of the image is usually left unchanged to provide context.

All these techniques are important in image processing as they help in improving the quality of images by making them more readable, identifying specific objects, and highlighting specific features. These techniques are widely used in various applications, including medical imaging, surveillance, and pattern recognition.

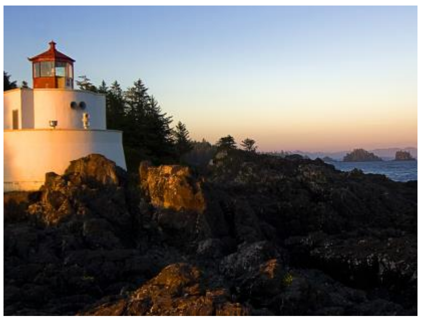
Figure 9. Original Image

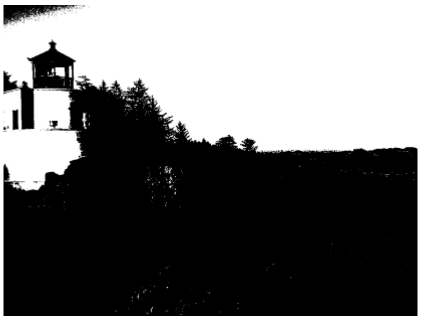
Figure 10. Image with Thresholding

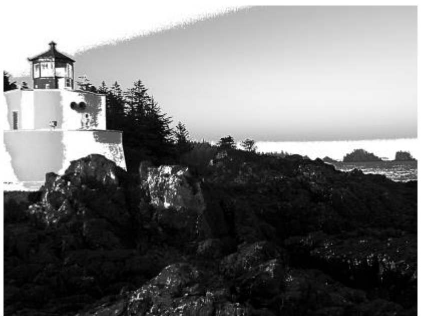
Figure 11. Image with Grey Level Slicing with Background

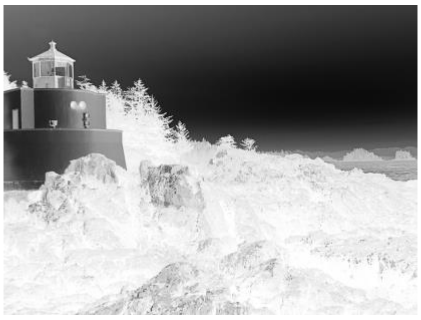
Figure 12. Image Negative - Lighthouse

```
import cv2 # Wrapper package for OpenCV python bindings
import numpy as np # General-purpose array-processing package

img = cv2.imread('Lighthouse.jpg', 0) # Read the input image
m,n = img.shape # To ascertain total numbers of rows and columns of the
image, size of the image
L = img.max() # To find the maximum grey level value in the image
img_neg = L-img # Maximum grey level value minus the original image
gives the negative image
cv2.imwrite('Image Negative - Lighthouse.png', img_neg) # Convert the
np array img_neg to a png image

# Thresholding without background
# Let threshold =T
# Let pixel value in the original be denoted by r
# Let pixel value in the new image be denoted by s
# If r<T, s= 0
# If r>T, s=255

T = 150
img_thresh = np.zeros((m,n), dtype = int) # Create an array of zeros

for i in range(m):
    for j in range(n):

        if img[i,j] < T:
            img_thresh[i,j]= 0
        else:
            img_thresh[i,j] = 255

cv2.imwrite('Image with Thresholding.png', img_thresh) # Convert array
to png image

T1 = 100 # Lower threshold value
T2 = 180 # Upper threshold value
img_thresh_back = np.zeros((m,n), dtype = int) # Create an array of
zeros

for i in range(m):
    for j in range(n):

        if T1 < img[i,j] < T2:
            img_thresh_back[i,j]= 255
        else:
            img_thresh_back[i,j] = img[i,j]

cv2.imwrite('Image with Grey Level Slicing with Background.png',
img_thresh_back) # Convert array to png image
```

IV. *Contrast Stretching*

```
from PIL import Image # Provides a class with the same name which is
used to represent a PIL image
from matplotlib import pyplot as plt # Collection of command style
functions that make matplotlib work like MATLAB

# Define methods to process the red, green, and blue bands of the image
def normalize(intensity, minI, maxI, minO, maxO):
    # Scale pixel values from input range (minI, maxI) to output range
(minO, maxO)
    return (intensity - minI) * ((maxO - minO) / (maxI - minI)) + minO

# Read the input image and split it into red, green, and blue bands
image = Image.open("Lighthouse.jpg")
bands = image.split()

# Define the minimum and maximum intensity values for each color band
red_min, red_max = 86, 230
green_min, green_max = 90, 225
blue_min, blue_max = 100, 210

# Apply contrast stretching on each color band
normalized_red = bands[0].point(lambda i: normalize(i, red_min,
red_max, 0, 255))
normalized_green = bands[1].point(lambda i: normalize(i, green_min,
green_max, 0, 255))
normalized_blue = bands[2].point(lambda i: normalize(i, blue_min,
blue_max, 0, 255))

# Merge the resulting red, green, and blue bands into a new image
normalized_image = Image.merge("RGB", (normalized_red,
normalized_green, normalized_blue))

# Display the resulting image
plt.imshow(normalized_image)
plt.axis('off')
plt.show()
```

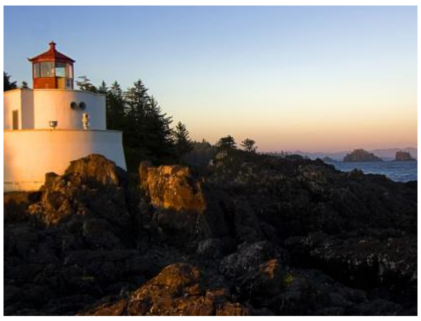
Figure 13. Original Image

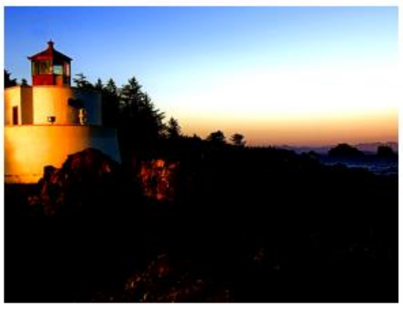
Figure 14. Contrast Stretching – Lighthouse

V. *Intensity Transformation Operations*

Log transformation and power-law (gamma) transformations are commonly used intensity transformation operations in image processing. Log transformation maps a narrow range of high-intensity values to a wider range of low-intensity values, while power-law transformation alters the contrast of an image by mapping input pixel values to output values using a power function.

```
import cv2 # Wrapper package for OpenCV python bindings
import numpy as np # General-purpose array-processing package
import warnings; warnings.filterwarnings("ignore")

img = cv2.imread('Lighthouse.jpg') # Read the input image

# Log Transformation
c = 255/(np.log(1 + np.max(img))) # Calculate the constant c
log_transformed = c * np.log(1 + img) # Apply the log transform
log_transformed = np.array(log_transformed, dtype = np.uint8) # Convert
the data type to unit8
cv2.imwrite('Log Transformed - Lighthouse.jpg', log_transformed) # Save
the log-transformed image

# Power-Law (Gamma) Transformations
for gamma in [0.1, 0.5, 1.2]: # Iterate over different gamma values
    gamma_corrected = np.array(255*(img / 255) ** gamma, dtype =
'uint8') # Apply gamma correction
    cv2.imwrite('Gamma Transformed ('+str(gamma)+').jpg',
gamma_corrected) # Save the gamma corrected image
```

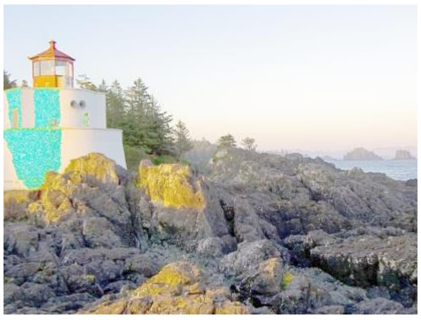
Figure 15. Log Transformed – Lighthouse

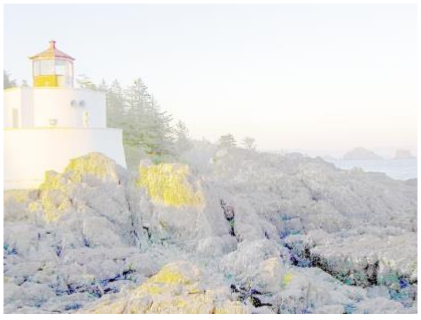
Figure 16. Gamma Transformed (0.1)

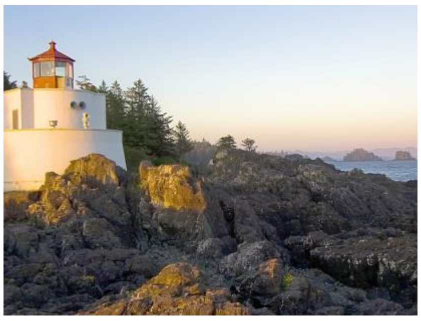
Figure 17. Gamma Transformed (0.5)

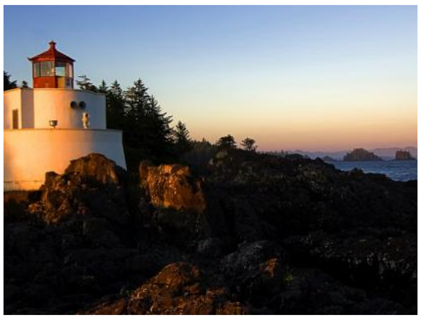
Figure 18. Gamma Transformed (1.2)

VI. *Bit-Plane Slicing*

```
import cv2  # Wrapper package for OpenCV python bindings
import numpy as np  # General-purpose array-processing package

# Read the input image
img = cv2.imread('Lighthouse.jpg', 0)

# Convert each pixel value to binary and store it in a list
lst = [np.binary_repr(pixel, width=8) for pixel in img.flatten()]

# Initialize empty lists for each bit plane
bit_planes = [[] for _ in range(8)]

# Store the bit value of each pixel in its corresponding bit plane
for i in range(8):
    for j in range(len(lst)):
        bit_planes[i].append(int(lst[j][i]))

    # Multiply with 2^(n-1) and reshape to reconstruct the bit image
    bit_planes[i] = (np.array(bit_planes[i], dtype=np.uint8) * 2**(7-
i)).reshape(img.shape)

# Concatenate the bit planes horizontally and vertically for ease of
display
finalr = cv2.hconcat(bit_planes[:4])
finalv = cv2.hconcat(bit_planes[4:])
final = cv2.vconcat([finalr, finalv])

# Write the bit planes and the combined 4-bit plane images
cv2.imwrite('8 Bit Planes - Lighthouse.jpg', final)
cv2.imwrite('Image Using 4 Bit Planes - Lighthouse.jpg', bit_planes[0]
+ bit_planes[1] + bit_planes[2] + bit_planes[3])
```

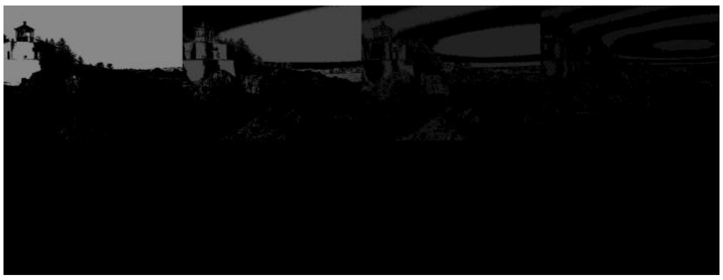
Figure 19. 8 Bit Planes – Lighthouse

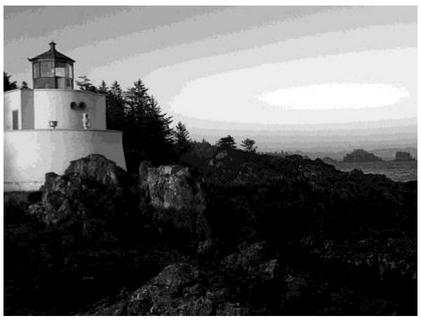
Figure 20. Image Using 4 Bit Planes – Lighthouse

VII. Histogram Processing

Histogram analysis is a commonly used technique in image processing for analysing the distribution of pixel intensities in an image. It involves plotting the frequency distribution of pixel values in an image as a histogram, where the x-axis represents the intensity values and the y-axis represents the number of pixels with that intensity value.

In image processing, the histogram is typically calculated using grey-level images, where each pixel is represented by a single intensity value. However, it can also be extended to colour images by computing separate histograms for each colour channel (RGB or HSV).

Histogram analysis is a powerful and widely used technique in image processing, and can be implemented using various programming languages and libraries such as Python, MATLAB, and OpenCV.

```python
import cv2  # Wrapper package for OpenCV python bindings
import numpy as np  # General-purpose array-processing package
from matplotlib import pyplot as plt # Collection of command style
functions that make matplotlib work like MATLAB

# Load the input image in grayscale
img = cv2.imread('Lighthouse.jpg', cv2.IMREAD_GRAYSCALE)

# Calculate the histogram using the OpenCV function cv2.calcHist()
hist, bins = np.histogram(img.flatten(), 256, [0, 256])

# Plot the histogram using Matplotlib
plt.hist(img.flatten(), 256, [0, 256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.title('Grayscale Image Histogram')

# Show the histogram
plt.show()
```

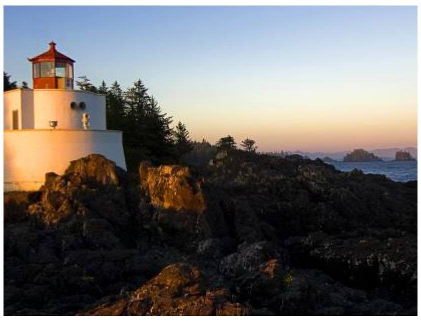
Figure 21. Original Image

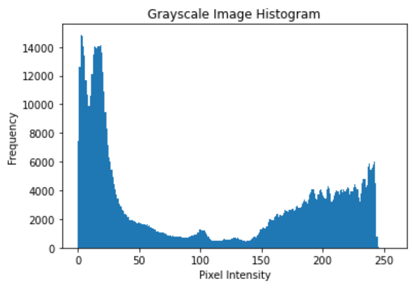

Figure 22. Grayscale Image Histogram

INFERENCE:

- The image processing assignment on image enhancement has been completed, and several methods for improving the visual quality of digital images have been explored.
- The theoretical background research and practical implementation of image enhancement techniques using Python programming have allowed for a comprehensive understanding of the different methods and their applications.
- The labelled images of input and output figures included in the documentation demonstrate the effectiveness of the methods and the improvements achieved in image quality.
- The references provided also allow for further exploration of the topic. Overall, the assignment has contributed to the development of skills in image processing and the use of Python programming for image enhancement.

REFERENCES:

Serial Number	Online Portal	Content	URL
1	Geeks For Geeks	Intensity Transformation Operations on Images	https://www.geeksforgeeks.org/python-intensity-transformation-operations-on-images/
2		OpenCV Python Program to Analyse an Image using Histogram	https://www.geeksforgeeks.org/opencv-python-program-analyze-image-using-histogram/
3		Point Processing in Image Processing using Python-OpenCV	https://www.geeksforgeeks.org/point-processing-in-image-processing-using-python-opencv/
4		Spatial Filters – Averaging Filter and Median Filter in Image Processing	https://www.geeksforgeeks.org/spatial-filters-averaging-filter-and-median-filter-in-image-processing/
5	Pythontic.com	Contrast Stretching Using Python and Pillow	https://pythontic.com/image-processing/pillow/contrast%20stretching#:~:text=Contrast%20stretching%20of%20an%20image,the%20value%20of%20original%20pixel
6	The AI Learner – Mastering Artificial Intelligence	Bit-Plane Slicing	https://theailearner.com/2019/01/25/bit-plane-slicing/
7	Tutorials Point - Simply Easy Learning	How to mask an image in OpenCV Python?	https://www.tutorialspoint.com/how-to-mask-an-image-in-opencv-python

Textbooks:

1. Digital Image Processing, Fourth Edition, Rafael C. Gonzalez, Richard E. Woods, Pearson, 2018
2. Digital Image Processing, S Jayaraman, S Esakkirajan, T Veerakumar, Tata McGraw Hill, 2009
3. Fundamentals of Digital Image Processing, Anil K. Jain, Pearson Education, Prentice Hall

References:

1. Digital Image Processing - PIKS Scientific Inside, Fourth Edition, Willian K Pratt, Wiley-Interscience, 2007
2. Multidimensional Signal, Image, and Video Processing and Coding, Second Edition, John W. Woods, Elsevier, 2012