# Digital Image Processing Assignment

**Submitted by -**
**Gunjan Kumar**
**(Enroll.No - 211020012008)**

**Submitted to -**
**Dr. Raghvendra H. Bhalerao**
**Assistant Professor Electrical Engineering**


**Department of Electrical Engineering**
**Institute of Infrastructure, Technology, Research And Management (IITRAM),**
**Ahmedabad-380026, Gujarat, India**

# Contents

# Chapter 1

# Introduction

## 1.1 Overview

This project demonstrates various digital image processing techniques using Python, including:

- Image compression.
- Image transformations (logarithmic, gamma, etc.).
- Geometric transformations (rotation, shifting, etc.).
- Histogram analysis and histogram equalization.
- Filtering (low-pass and high-pass).
- Edge detection using gradients.
- Morphological operations.

# Chapter 2

# Methodology

## 2.1 Technologies Used

The following libraries and frameworks were used:

- OpenCV for image processing.
- NumPy for numerical computations.
- Matplotlib for visualization.
- scikit-image for morphological operations.

## 2.2 Steps

Each operation is applied to an input image (`input.png`), and the results are discussed in the subsequent sections.

# Chapter 3

# Results and Discussion

## 3.1 Image Compression

**Input Image:**


Figure 3.1: Original Image

**Compressed Image:**

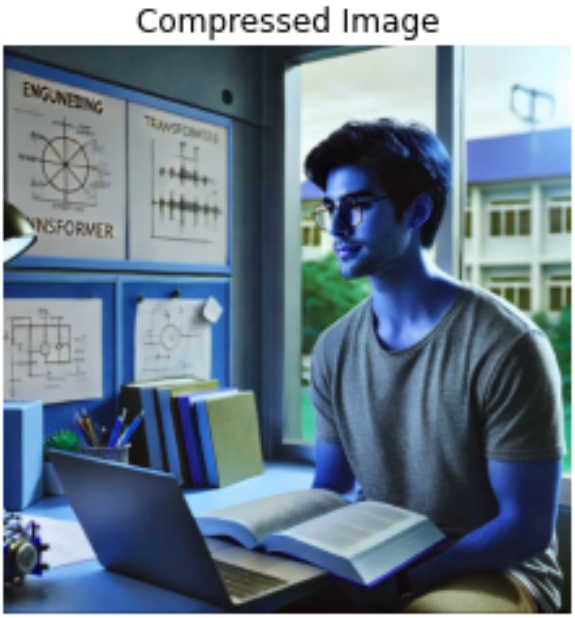


Figure 3.2: Compressed Image with Quality 50

## 3.2 Log and Gamma Transformations

**Log Transformation:**

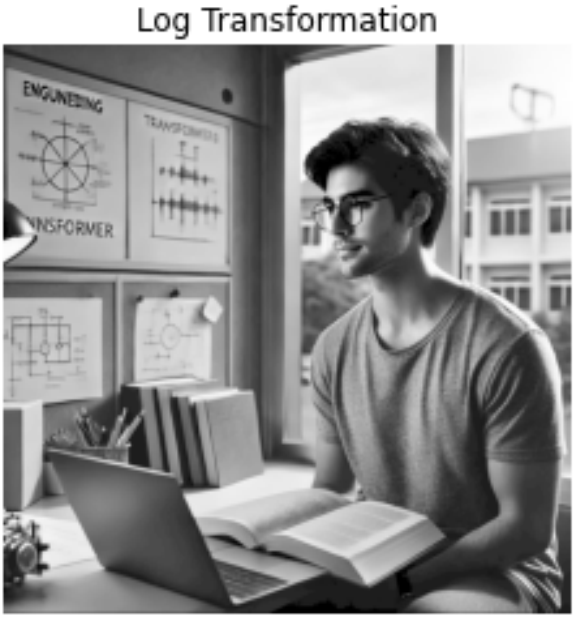


Figure 3.3: Log Transformation

**Gamma Transformation:**


Figure 3.4: Gamma Transformation (Gamma = 2.0)

## 3.3 Geometric Transformations

**Rotated Image (45 Degrees):**

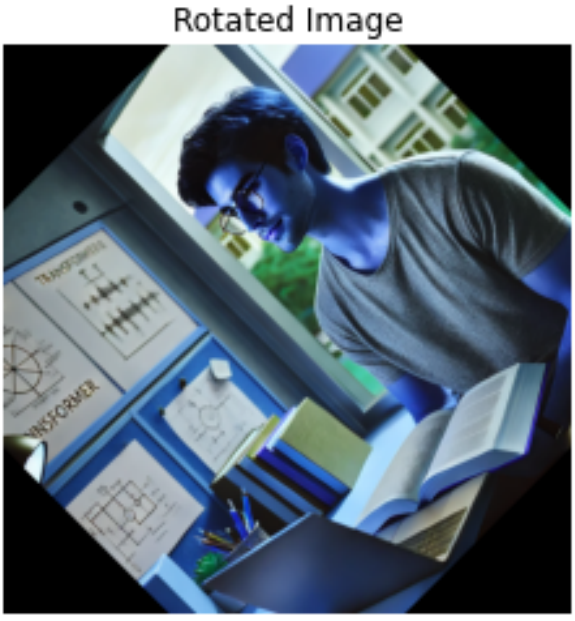


Figure 3.5: Image Rotated by 45 Degrees

**Translated Image:**


Figure 3.6: Image Translated (Tx=50, Ty=30)

## 3.4 Histogram and Equalization

**Histogram:**

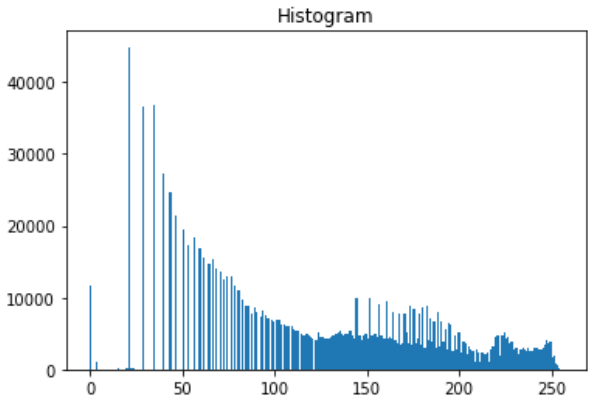


Figure 3.7: Histogram of Grayscale Image

**Histogram Equalized Image:**

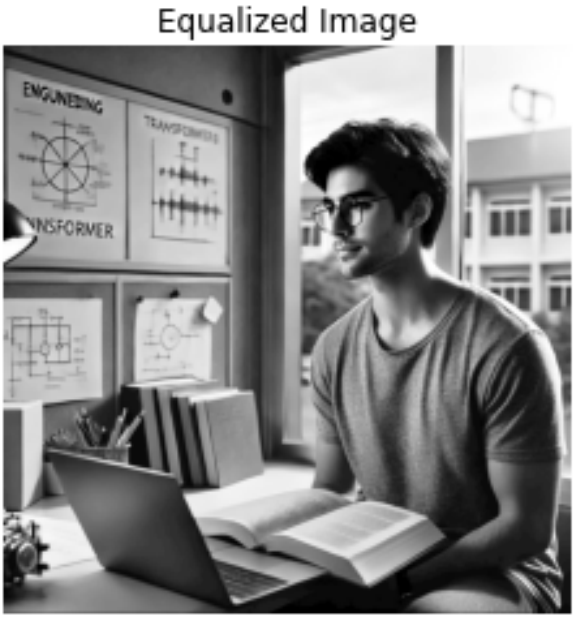


Figure 3.8: Histogram Equalized Image

## 3.5 Filtering

**Low-Pass Filtered Image:**

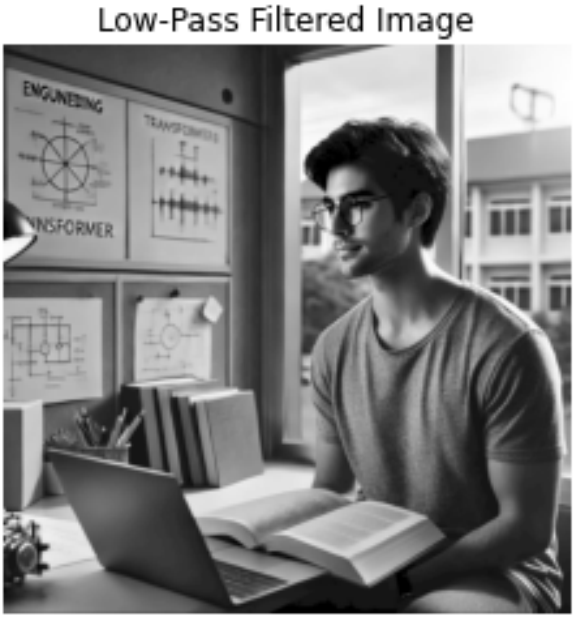


Figure 3.9: Low-Pass Filtered Image

**High-Pass Filtered Image:**


Figure 3.10: High-Pass Filtered Image

## 3.6 Edge Detection

**Gradient-Based Edge Detection:**


Figure 3.11: Gradient Magnitude for Edge Detection

## 3.7 Morphological Image Processing

**Eroded Image:**


Figure 3.12: Eroded Image

**Dilated Image:**


Figure 3.13: Dilated Image

**Opened Image:**


Figure 3.14: Opened Image

**Closed Image:**


Figure 3.15: Closed Image

# Chapter 4

# Code Implementation

The Python code used for implementing all digital image processing techniques is provided below:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.morphology import binary_erosion, binary_dilation,
   binary_opening, binary_closing, disk

image_path = r"C:\Users\Gunjan Anshuman\Desktop\DIP.png"

# Unified Image Display Function
def display_image_with_matplotlib(title, image, cmap='gray'):
    plt.imshow(image, cmap=cmap)
    plt.title(title)
    plt.axis('off')
    plt.show()

# 1. Image Compression
def compress_image(image_path, quality):
    img = cv2.imread(image_path)
    if img is None:
        print("Error: Image not loaded. Check the file path or
           format.")
        return None

    compressed_image_path = 'compressed.jpg'
    success = cv2.imwrite(compressed_image_path, img, [int(cv2.
       IMWRITE_JPEG_QUALITY), quality])
    if success:
        print(f"Image successfully compressed and saved at {
           compressed_image_path}")
        compressed_img = cv2.imread(compressed_image_path)
        display_image_with_matplotlib("Compressed Image",
           compressed_img)
    else:
        print("Error: Image compression failed.")
    return compressed_image_path
```

```
compress_image(image_path, 50)

# 2. Image Transformations (Log, Gamma)
def log_gamma_transformations(image_path, c, gamma):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) / 255.0
    if img is None:
        print("Error: Image not loaded.")
        return

    # Log Transformation
    log_transformed = c * np.log(1 + img)
    display_image_with_matplotlib('Log Transformation',
       log_transformed, cmap='gray')

    # Gamma Transformation
    gamma_transformed = c * (img ** gamma)
    display_image_with_matplotlib('Gamma Transformation',
       gamma_transformed, cmap='gray')

log_gamma_transformations(image_path, c=1.0, gamma=2.0)

# 3. Geometric Transformations (Rotation, Shifting)
def geometric_transformations(image_path, angle, tx, ty):
    img = cv2.imread(image_path)
    if img is None:
        print("Error: Image not loaded.")
        return

    rows, cols = img.shape[:2]
    # Rotation
    rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows /
       2), angle, 1)
    rotated_img = cv2.warpAffine(img, rotation_matrix, (cols,
       rows))
    display_image_with_matplotlib("Rotated Image", rotated_img)

    # Translation
    translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
    translated_img = cv2.warpAffine(img, translation_matrix, (
       cols, rows))
    display_image_with_matplotlib("Translated Image",
       translated_img)

geometric_transformations(image_path, angle=45, tx=50, ty=30)

# 4. Histogram and Histogram Equalization
def plot_histogram(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print("Error: Image not loaded.")
```

```
        return
    plt.hist(img.ravel(), bins=256, range=(0, 256))
    plt.title('Histogram')
    plt.show()

plot_histogram(image_path)

def histogram_equalization(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print("Error: Image not loaded.")
        return
    equalized_img = cv2.equalizeHist(img)
    display_image_with_matplotlib('Equalized Image',
       equalized_img, cmap='gray')

histogram_equalization(image_path)

# 5. Filtering (LPF & HPF) Using Mask
def filtered_images(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print("Error: Image not loaded.")
        return
    # Low-Pass Filter
    kernel_lpf = np.ones((3, 3), np.float32) / 9
    lpf_img = cv2.filter2D(img, -1, kernel_lpf)
    display_image_with_matplotlib('Low-Pass Filtered Image',
       lpf_img)

    # High-Pass Filter
    kernel_hpf = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1,
       -1]])
    hpf_img = cv2.filter2D(img, -1, kernel_hpf)
    display_image_with_matplotlib('High-Pass Filtered Image',
       hpf_img)

filtered_images(image_path)

# 6. Edge Detection Using Gradient
def edge_detection(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print("Error: Image not loaded.")
        return
    sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
    sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
    magnitude = np.sqrt(sobelx**2 + sobely**2)
    display_image_with_matplotlib('Edge Detection (Gradient
       Magnitude)', magnitude, cmap='gray')
```

```
edge_detection(image_path)

# 7. Morphological Image Processing
def morphological_processing(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print("Error: Image not loaded.")
        return
    binary_img = img > 127

    # Erosion
    eroded = binary_erosion(binary_img, disk(3))
    display_image_with_matplotlib('Eroded Image', eroded, cmap='
       gray')

    # Dilation
    dilated = binary_dilation(binary_img, disk(3))
    display_image_with_matplotlib('Dilated Image', dilated, cmap=
       'gray')

    # Opening
    opened = binary_opening(binary_img, disk(3))
    display_image_with_matplotlib('Opened Image', opened, cmap='
       gray')

    # Closing
    closed = binary_closing(binary_img, disk(3))
    display_image_with_matplotlib('Closed Image', closed, cmap='
       gray')

morphological_processing(image_path)
```

# Chapter 5

# Conclusion

The project successfully implemented various digital image processing techniques using Python. Each operation's results were displayed and analyzed. These techniques form the foundation for advanced image processing applications in fields like computer vision, machine learning, and image analysis.