Master of Science (Five Year Integrated) in Computer Science (Artificial Intelligence & Data Science)

Seventh Semester

Laboratory Record

21-805-0705: Image and Video Processing Lab

Submitted in partial fulfillment
of the requirements for the award of degree in
Master of Science (Five Year Integrated)
in Computer Science (Artificial Intelligence & Data Science) of
Cochin University of Science and Technology (CUSAT)
Kochi

Submitted by

HIRA MOHAMMED K
(80521011)

DEPARTMENT OF COMPUTER SCIENCE
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)
KOCHI-682022

DECEMBER 2024

DEPARTMENT OF COMPUTER SCIENCE

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)

KOCHI, KERALA-682022

This is to certify that the software laboratory record for ***21-805-0705: Image and Video Processing Lab*** *is a record of work carried out by* ***HIRA MOHAMMED K (80521011)****, in partial fulfillment of the requirements for the award of degree in* ***Master of Science (Five Year Integrated)*** *in* ***Computer Science (Artificial Intelligence & Data Science)*** *of Cochin University of Science and Technology (CUSAT), Kochi. The lab record has been approved as it satisfies the academic requirements in respect of the seventh semester laboratory prescribed for the Master of Science (Five Year Integrated) in Computer Science degree.*

Faculty Member in-charge

Dr. Remya K Sasi
Assistant Professor
Department of Computer Science
CUSAT

Dr. Madhu S. Nair
Professor and Head
Department of Computer Science
CUSAT

OUR VISION

To globally excel in innovative research,teaching, and technology development inspired by social obligation.

OUR MISSION

- **To contribute to knowledge development and dissemination.**
- **To facilitate learning and innovative research in frontier areas of computer science.**
- **To drive students for technology development to solve problems of interest.**
- **To create socially responsible professionals.**

Table of Contents

Image Negative

AIM

- Implement the image negative transformation function and apply it to a grayscale image.
- Analyze the effect of image negative on different types of images (e.g., low contrast, high contrast).
- Compare the histogram of an original image with its negative. Explain the observed differences.

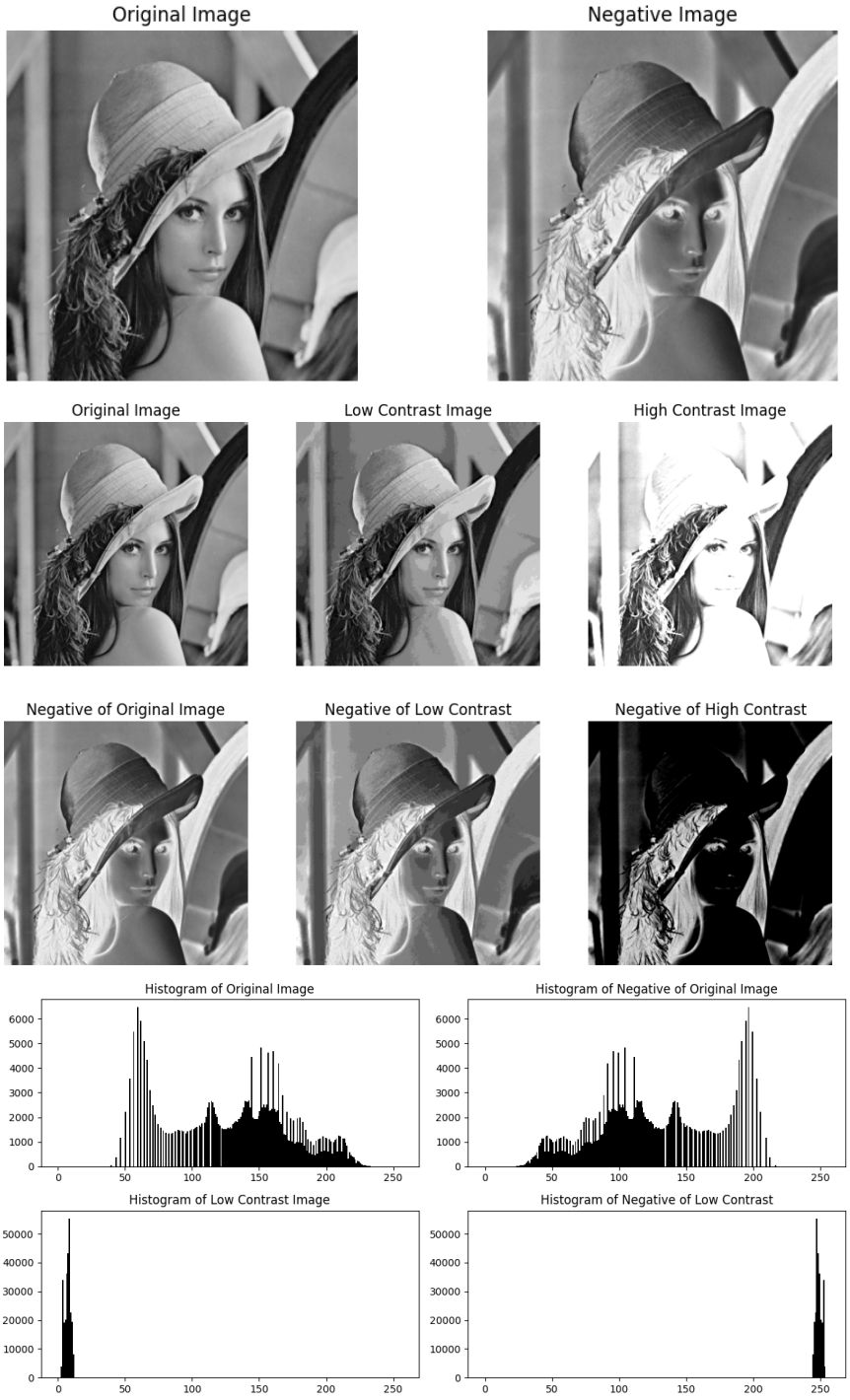
PROGRAM

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('/content/Lenna_(test_image).png', cv2.IMREAD_GRAYSCALE)

def negative_transformation(img):
    return 255 - img

negative_image = negative_transformation(image)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Negative Image')
plt.imshow(negative_image, cmap='gray')
plt.axis('off')

plt.show()

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Histogram of Original Image')
plt.hist(image.ravel(), bins=256, range=[0, 256], color='black')
```

```
plt.subplot(1, 2, 2)
plt.title('Histogram of Negative Image')
plt.hist(negative_image.ravel(), bins=256, range=[0, 256], color='black')

plt.show()
import cv2
import numpy as np
import matplotlib.pyplot as plt

def adjust_contrast(img, alpha, beta):
    return cv2.convertScaleAbs(img, alpha=alpha, beta=beta)

def negative_transformation(img):
    return 255 - img

low_contrast = adjust_contrast(image, alpha=0.05, beta=0)
high_contrast = adjust_contrast(image, alpha=2.0, beta=0)

negative_low_contrast = negative_transformation(low_contrast)
negative_high_contrast = negative_transformation(high_contrast)
plt.figure(figsize=(12, 8))

plt.subplot(2, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.title('Low Contrast Image')
plt.imshow(low_contrast, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.title('High Contrast Image')
plt.imshow(high_contrast, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.title('Negative of Original Image')
plt.imshow(negative_transformation(image), cmap='gray')
```

```
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title('Negative of Low Contrast')
plt.imshow(negative_low_contrast, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 6)
plt.title('Negative of High Contrast')
plt.imshow(negative_high_contrast, cmap='gray')
plt.axis('off')

plt.show()

# Plot the histograms
plt.figure(figsize=(12, 6))

# Histogram of Original Image and its Negative
plt.subplot(2, 2, 1)
plt.title('Histogram of Original Image')
plt.hist(image.ravel(), bins=256, range=[0, 256], color='black')

plt.subplot(2, 2, 2)
plt.title('Histogram of Negative of Original Image')
negative_image = negative_transformation(image)
plt.hist(negative_image.ravel(), bins=256, range=[0, 256], color='black')

# Histogram of Low Contrast Image and its Negative
plt.subplot(2, 2, 3)
plt.title('Histogram of Low Contrast Image')
plt.hist(low_contrast.ravel(), bins=256, range=[0, 256], color='black')

plt.subplot(2, 2, 4)
plt.title('Histogram of Negative of Low Contrast')
plt.hist(negative_low_contrast.ravel(), bins=256, range=[0, 256], color='black')

plt.tight_layout()
plt.show()
```

SAMPLE INPUT-OUTPUT



Log Transformation

AIM

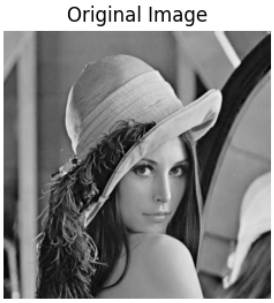
- Implement the log transformation function and apply it to an image with a narrow range of low gray-level values.
- Analyze the effect of the log transformation on enhancing details in dark regions of an image.
- Experiment with different values of the constant 'c' in the log transformation equation and observe the changes in output image.

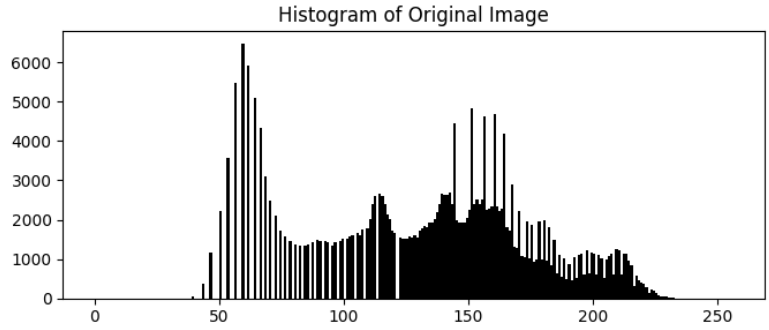
PROGRAM

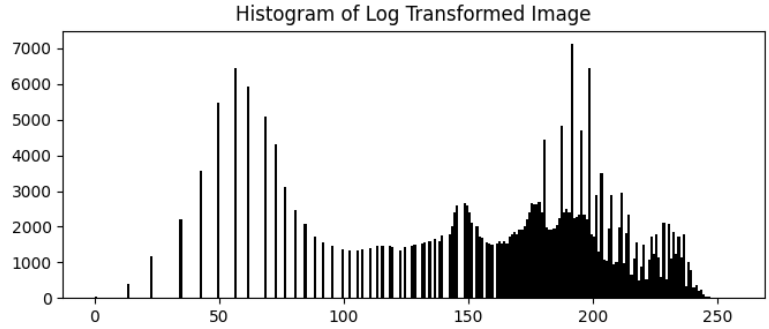
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def log_transform(image, c=1.0):
    image = np.float32(image)
    log_image = c * np.log(1 + image)
    log_image = np.uint8(cv2.normalize(log_image, None, 0, 255, cv2.NORM_MINMAX))
    return log_image
image = cv2.imread('/content/Lenna_(test_image).png', cv2.IMREAD_GRAYSCALE)

log_image = log_transform(image, c=1.0)
plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.hist(image.ravel(), bins=256, range=[0, 256], color='black')
plt.title('Histogram of Original Image')

plt.subplot(2, 2, 3)
plt.imshow(log_image, cmap='gray')
plt.title('Log Transformed Image')
plt.axis('off')
```

```
plt.subplot(2, 2, 4)
plt.hist(log_image.ravel(), bins=256, range=[0, 256], color='black')
plt.title('Histogram of Log Transformed Image')

plt.tight_layout()
plt.show()
c_values = [0.5, 1.0, 2.0]

plt.figure(figsize=(15, 5))

for i, c in enumerate(c_values, 1):
    log_image = log_transform(image, c)
    plt.subplot(1, 3, i)
    plt.imshow(log_image, cmap='gray')
    plt.title(f'Log Transform with c = {c}')
    plt.axis('off')

plt.show()
```

SAMPLE INPUT-OUTPUT







Power-Law Transformation

AIM

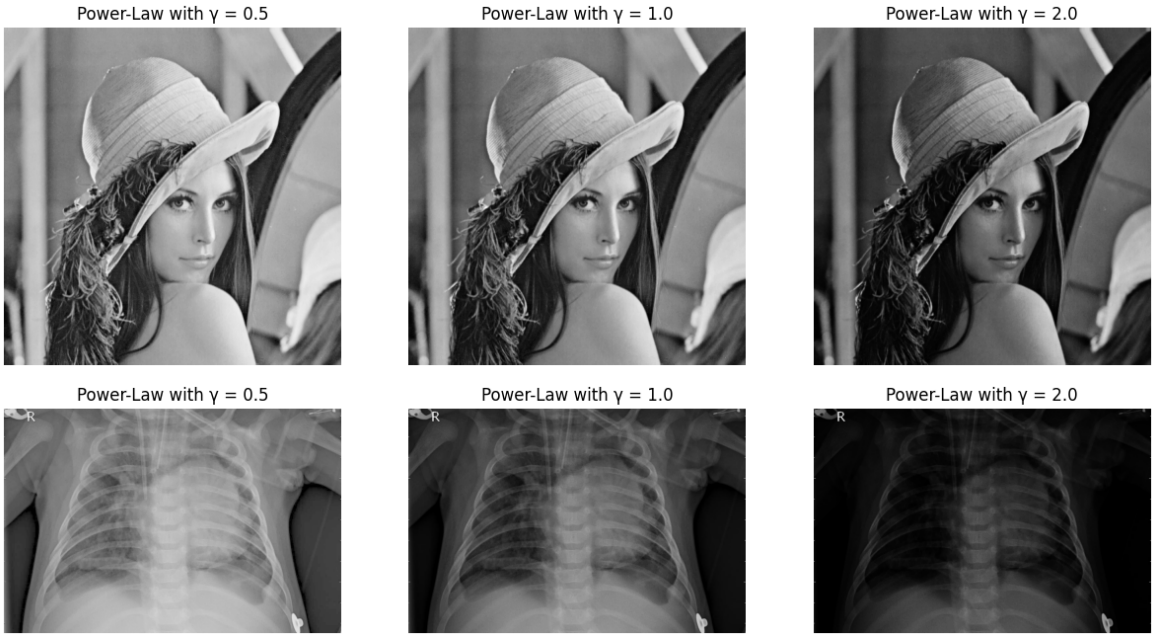
- Implement the power-law transformation function with different values of gamma.
- Apply the power-law transformation to enhance images with different contrast characteristics.
- Analyze the effect of gamma values on the image appearance, especially for values less than and greater than 1.
- Experiment with different image types (e.g., medical, satellite, natural) to observe the impact of transformations.

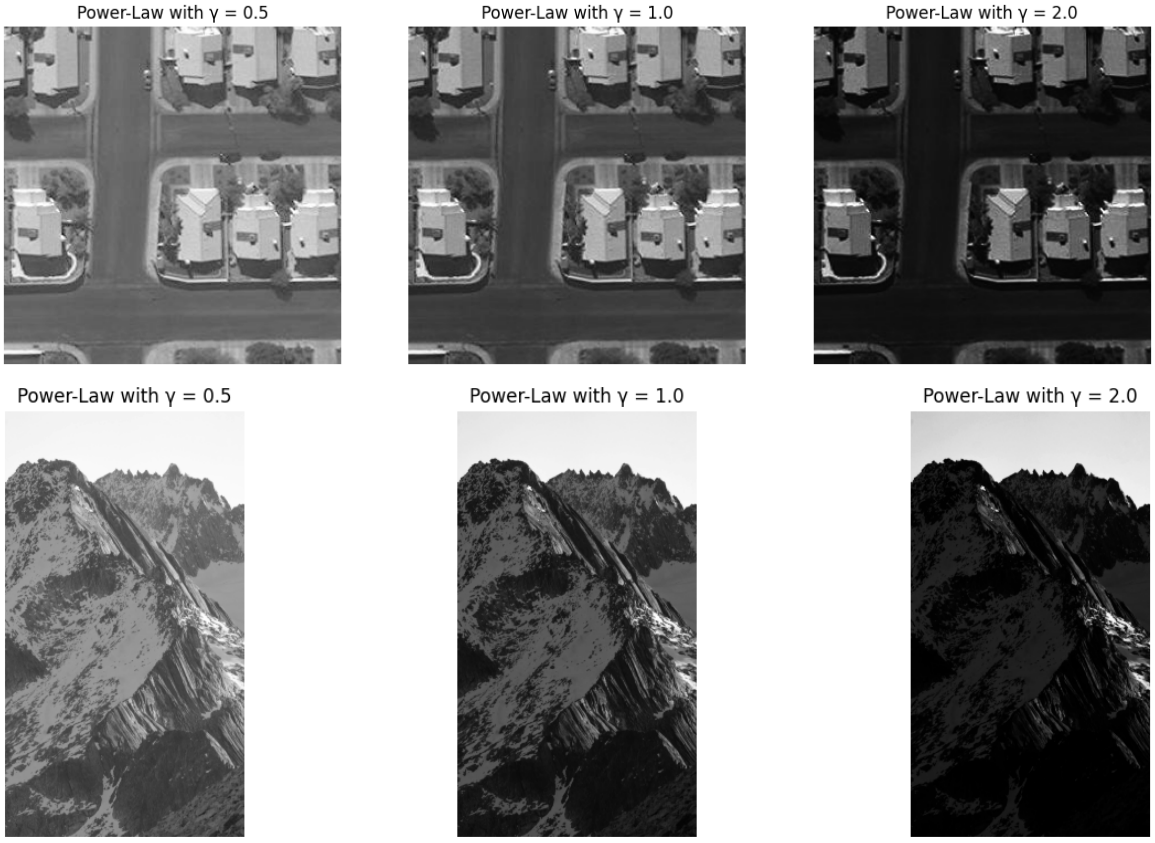
PROGRAM

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def power_law_transform(image, gamma=1.0, c=1.0):
    image = np.float32(image)
    power_law_image = c * (image ** gamma)
    power_law_image = np.uint8(cv2.normalize(power_law_image,
    None, 0, 255, cv2.NORM_MINMAX))
    return power_law_image
image = cv2.imread('/content/Lenna_(test_image).png', cv2.IMREAD_GRAYSCALE)

gamma_values = [0.5, 1.0, 2.0]

plt.figure(figsize=(15, 5))

for i, gamma in enumerate(gamma_values, 1):
    power_law_image = power_law_transform(image, gamma)

    plt.subplot(1, 3, i)
    plt.imshow(power_law_image, cmap='gray')
    plt.title(f'Power-Law with  = {gamma}')
    plt.axis('off')

plt.show()
def experiment_with_images(image_path, gamma_values):
```

```
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    plt.figure(figsize=(15, 5))

    for i, gamma in enumerate(gamma_values, 1):
        power_law_image = power_law_transform(image, gamma)

        plt.subplot(1, 3, i)
        plt.imshow(power_law_image, cmap='gray')
        plt.title(f'Power-Law with  = {gamma}')
        plt.axis('off')

    plt.show()

# Example for testing with different image types
experiment_with_images('/content/medical-image.jpeg', gamma_values)
experiment_with_images('/content/satellite-image.jpg', gamma_values)
experiment_with_images('/content/natural-image.jpg', gamma_values)
```

SAMPLE INPUT-OUTPUT




Power-Law with γ = 0.5
Power-Law with γ = 1.0
Power-Law with γ = 2.0
Power-Law with γ = 0.5
Power-Law with γ = 1.0
Power-Law with γ = 2.0

Spatial Filtering

AIM

- Implement mean, median, and Gaussian filters. Apply them to images with different noise types (salt-and-pepper, Gaussian) and compare the results.
- Design a custom filter for sharpening edges while preserving image details. Apply it to a natural image and evaluate its performance.
- Experiment with different Laplacian operators (4-connected, 8-connected) and compare their edge detection capabilities.

PROGRAM

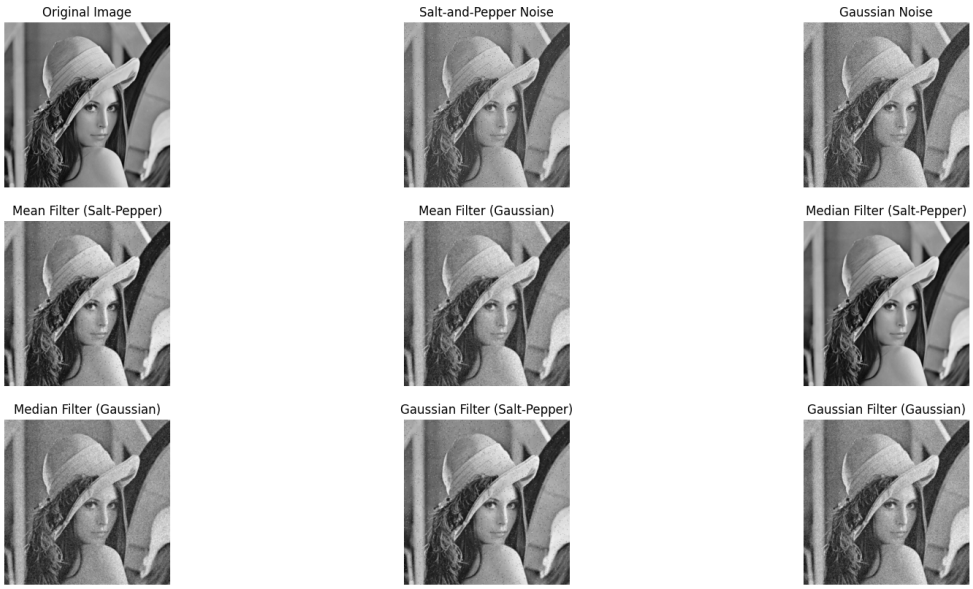
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def apply_mean_filter(image, kernel_size=3):
    return cv2.blur(image, (kernel_size, kernel_size))

def apply_median_filter(image, kernel_size=3):
    return cv2.medianBlur(image, kernel_size)

def apply_gaussian_filter(image, kernel_size=3, sigma=1):
    return cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)

def add_salt_and_pepper_noise(image, amount=0.02):
    noisy_image = image.copy()
    total_pixels = noisy_image.size
    num_salt = int(total_pixels * amount / 2)
    num_pepper = int(total_pixels * amount / 2)

    salt_coords = [np.random.randint(0, i-1, num_salt) for i in noisy_image.shape]
    noisy_image[salt_coords[0], salt_coords[1]] = 255

    pepper_coords = [np.random.randint(0, i-1, num_pepper) for i in noisy_image.shape]
    noisy_image[pepper_coords[0], pepper_coords[1]] = 0

    return noisy_image
```

```
def add_gaussian_noise(image, mean=0, sigma=25):
    row, col = image.shape
    gauss = np.random.normal(mean, sigma, (row, col))
    noisy_image = np.add(image, gauss)
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image.astype(np.uint8)

image_path = '/content/Lenna_(test_image).png'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

salt_pepper_image = add_salt_and_pepper_noise(image)
gaussian_image = add_gaussian_noise(image)

mean_filtered_salt_pepper = apply_mean_filter(salt_pepper_image)
median_filtered_salt_pepper = apply_median_filter(salt_pepper_image)
gaussian_filtered_salt_pepper = apply_gaussian_filter(salt_pepper_image)

mean_filtered_gaussian = apply_mean_filter(gaussian_image)
median_filtered_gaussian = apply_median_filter(gaussian_image)
gaussian_filtered_gaussian = apply_gaussian_filter(gaussian_image)

plt.figure(figsize=(20, 10))

plt.subplot(3, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(3, 3, 2)
plt.imshow(salt_pepper_image, cmap='gray')
plt.title('Salt-and-Pepper Noise')
plt.axis('off')

plt.subplot(3, 3, 3)
plt.imshow(gaussian_image, cmap='gray')
plt.title('Gaussian Noise')
plt.axis('off')

plt.subplot(3, 3, 4)
plt.imshow(mean_filtered_salt_pepper, cmap='gray')
plt.title('Mean Filter (Salt-Pepper)')
```

```
plt.axis('off')

plt.subplot(3, 3, 5)
plt.imshow(mean_filtered_gaussian, cmap='gray')
plt.title('Mean Filter (Gaussian)')
plt.axis('off')

plt.subplot(3, 3, 6)
plt.imshow(median_filtered_salt_pepper, cmap='gray')
plt.title('Median Filter (Salt-Pepper)')
plt.axis('off')

plt.subplot(3, 3, 7)
plt.imshow(median_filtered_gaussian, cmap='gray')
plt.title('Median Filter (Gaussian)')
plt.axis('off')

plt.subplot(3, 3, 8)
plt.imshow(gaussian_filtered_salt_pepper, cmap='gray')
plt.title('Gaussian Filter (Salt-Pepper)')
plt.axis('off')

plt.subplot(3, 3, 9)
plt.imshow(gaussian_filtered_gaussian, cmap='gray')
plt.title('Gaussian Filter (Gaussian)')
plt.axis('off')

plt.show()
def apply_sharpening_filter(image):
    kernel = np.array([[ 0, -1,  0],
                       [-1,  5, -1],
                       [ 0, -1,  0]], dtype=np.float32)
    return cv2.filter2D(image, -1, kernel)

sharpened_image = apply_sharpening_filter(image)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
```

```
plt.subplot(1, 2, 2)
plt.imshow(sharpened_image, cmap='gray')
plt.title('Sharpened Image')
plt.axis('off')

plt.show()
def laplacian_4_connected(image):
    kernel = np.array([[ 0,  1,  0],
                       [ 1, -4,  1],
                       [ 0,  1,  0]], dtype=np.float32)
    return cv2.filter2D(image, -1, kernel)

def laplacian_8_connected(image):
    kernel = np.array([[ 1,  1,  1],
                       [ 1, -8,  1],
                       [ 1,  1,  1]], dtype=np.float32)
    return cv2.filter2D(image, -1, kernel)

laplacian_4_image = laplacian_4_connected(image)
laplacian_8_image = laplacian_8_connected(image)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(laplacian_4_image, cmap='gray')
plt.title('Laplacian (4-Connected)')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(laplacian_8_image, cmap='gray')
plt.title('Laplacian (8-Connected)')
plt.axis('off')

plt.show()
```

SAMPLE INPUT-OUTPUT



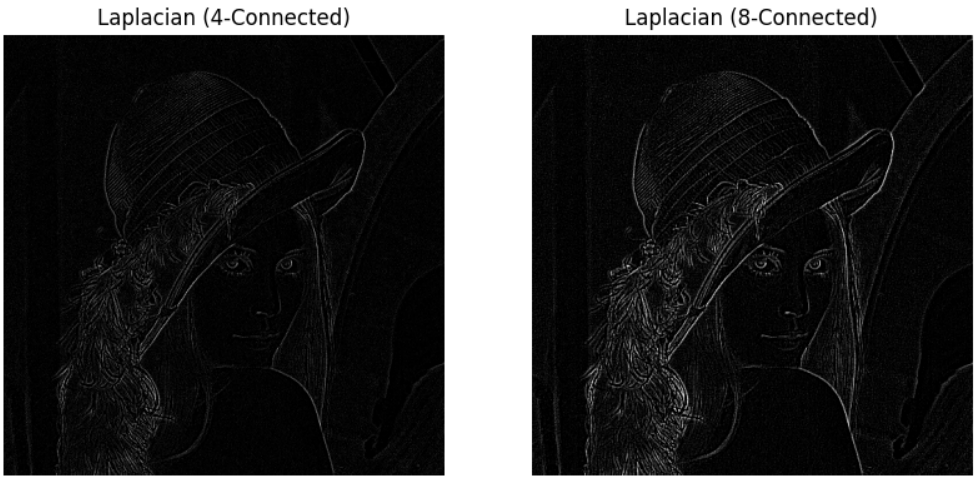


Image Enhancement: Arithmetic/Logic Operations

AIM

- Implement image subtraction to detect changes between two images (e.g, before and after an event).
- Create a simple image watermarking system using image addition and subtraction.
- Experiment with image averaging to reduce noise in a sequence of images.

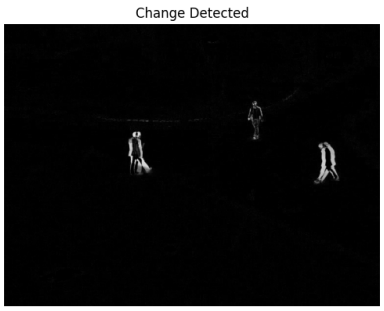
PROGRAM

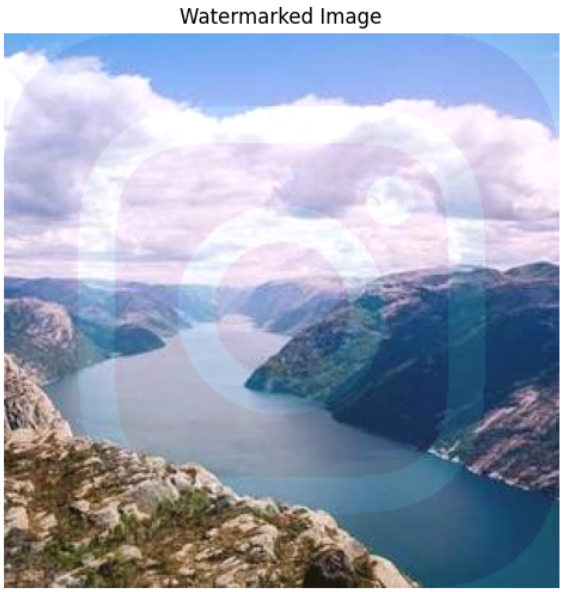
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

video = cv2.VideoCapture("/content/sample_video.mp4")
success, frame1 = video.read()
success, frame2 = video.read()
video.release()

frame1_gray = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
frame2_gray = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)

change_detected = cv2.absdiff(frame1_gray, frame2_gray)

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.title("Frame 1 (Before)")
plt.imshow(frame1_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Frame 2 (After)")
plt.imshow(frame2_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title("Change Detected")
plt.imshow(change_detected, cmap='gray')
plt.axis('off')
```

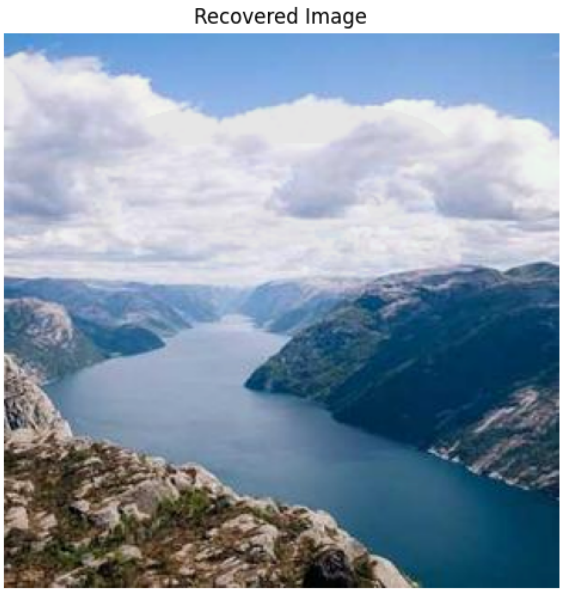
```
plt.tight_layout()
plt.show()

import requests
import numpy as np

def load_image_from_url(url):
    response = requests.get(url)
    image_array = np.asarray(bytearray(response.content), dtype=np.uint8)
    img = cv2.imdecode(image_array, cv2.IMREAD_COLOR)
    return img

base_image_url = 'https://picsum.photos/300/300'
watermark_image_url = 'https://upload.wikimedia.org/wikipedia/commons/a/a5/
                    Instagram_icon.png'

base_image = load_image_from_url(base_image_url)
watermark_image = load_image_from_url(watermark_image_url)

watermark_image = cv2.resize(watermark_image, (base_image.shape[1], base_image.shape[0]))

base_image = np.array(base_image, dtype=np.uint8)
watermark_image = np.array(watermark_image, dtype=np.uint8)

alpha = 1
beta = 0.1
gamma = 0

watermarked_image = cv2.addWeighted(base_image, alpha, watermark_image, beta, gamma)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(watermarked_image, cv2.COLOR_BGR2RGB))
plt.title('Watermarked Image')
plt.axis('off')

recovered_image = cv2.addWeighted(watermarked_image, 1/alpha,
watermark_image, -beta/alpha, -gamma/alpha)

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(recovered_image, cv2.COLOR_BGR2RGB))
```

```
plt.title('Recovered Image')
plt.axis('off')

plt.tight_layout()
plt.show()

noisy_images_files = [
    '/content/Gaussian_noisy_image.png',
    '/content/Salt_and_Pepper_noisy_image.png',
    '/content/Speckle_noisy_image.png'
]

def load_images(file_paths):
    images = [cv2.imread(file_path) for file_path in file_paths]
    return images

noisy_images = load_images(noisy_images_files)

def average_images(images):
    return np.mean(images, axis=0).astype(np.uint8)

average_image = average_images(noisy_images)

plt.figure(figsize=(8, 6))
plt.title('Averaged Image')
plt.imshow(cv2.cvtColor(average_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

SAMPLE INPUT-OUTPUT







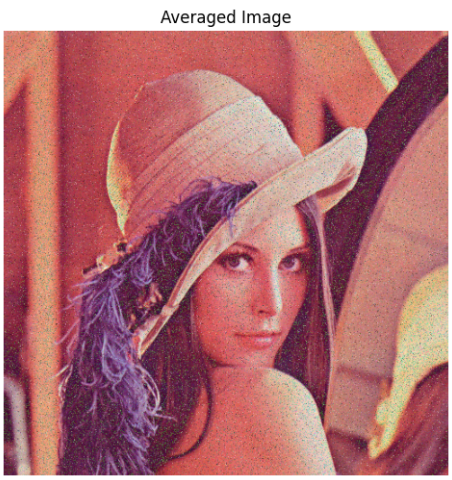


Image Transform

AIM

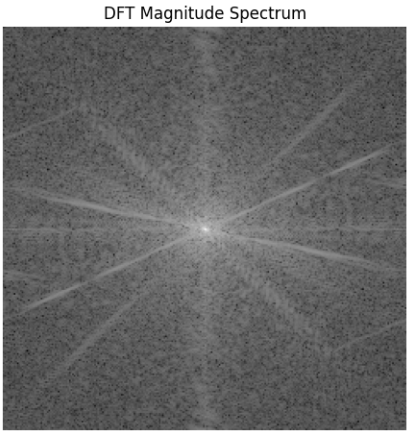
Perform Discrete Fourier Transform, Z- transform KL Transform on a gray scale image.

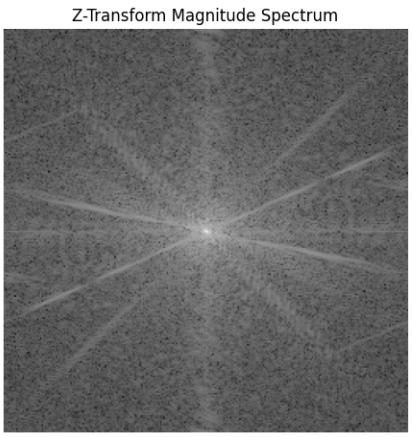
PROGRAM

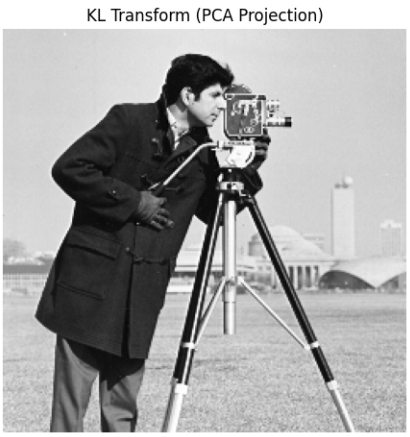
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

image_path = '/content/cameraman-testimage.png'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

if image is None:
    print(f"Error: Unable to load image at {image_path}")
else:
    print(f"Image loaded successfully with shape: {image.shape}")

def dft_transform(image):
    dft = cv2.dft(np.float32(image), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    magnitude_spectrum = cv2.magnitude(dft_shift[:, :, 0], dft_shift[:, :, 1])
    return magnitude_spectrum

def z_transform(image):
    z_transform_result = np.fft.fft2(image)
    z_shifted = np.fft.fftshift(z_transform_result)
    magnitude_spectrum = np.abs(z_shifted)
    return magnitude_spectrum

def kl_transform(image, n_components=1):
    image_reshaped = image.reshape(-1, 1)
    pca = PCA(n_components=n_components)
    pca_result = pca.fit_transform(image_reshaped)
    image_reconstructed = pca.inverse_transform(pca_result)
    kl_transformed = image_reconstructed.reshape(image.shape)
    return kl_transformed

dft_result = dft_transform(image)
```

```
z_result = z_transform(image)

if image is not None:
    kl_result = kl_transform(image)

plt.figure(figsize=(12, 12))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(np.log(dft_result + 1), cmap='gray')
plt.title("DFT Magnitude Spectrum")
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(np.log(z_result + 1), cmap='gray')
plt.title("Z-Transform Magnitude Spectrum")
plt.axis('off')

plt.subplot(2, 2, 4)
if image is not None:
    plt.imshow(kl_result, cmap='gray')
    plt.title("KL Transform (PCA Projection)")
    plt.axis('off')
else:
    plt.text(0.5, 0.5, "Image not loaded correctly", ha='center',
    va='center', fontsize=12)
    plt.axis('off')

plt.show()
```

SAMPLE INPUT-OUTPUT







Intensity Transformation and Histogram Processing

AIM

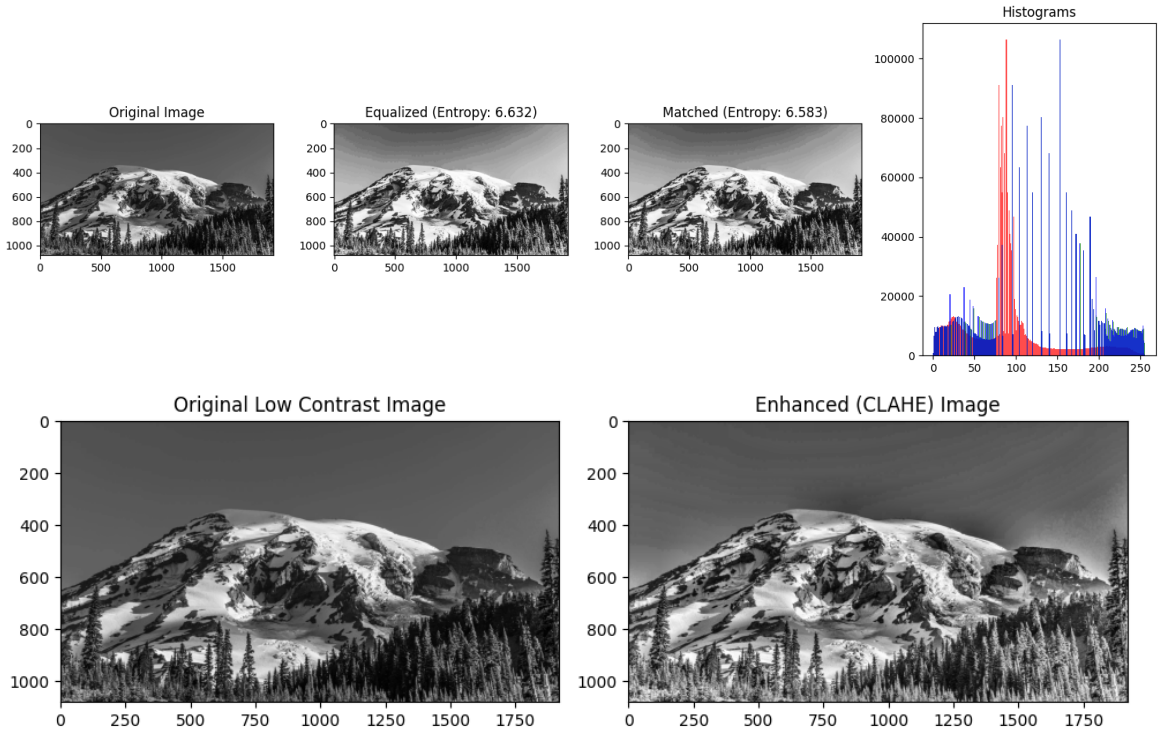
- Implement histogram equalization and matching on a grayscale image. Compare the results visually and quantitatively using metrics like entropy.
- Design a contrast enhancement technique for images with low contrast. Apply it to a real-world image and evaluate its effectiveness.

PROGRAM

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import exposure

image = cv2.imread('/content/Rainier.png', cv2.IMREAD_GRAYSCALE)
equalized_image = cv2.equalizeHist(image)

reference_image = cv2.imread('/content/reference_image.png', cv2.IMREAD_GRAYSCALE)
matched_image = exposure.match_histograms(image, reference_image, channel_axis=None)

def compute_entropy(img):
    hist, _ = np.histogram(img, bins=256, range=(0, 256))
    hist = hist / hist.sum()
    entropy = -np.sum(hist * np.log2(hist + 1e-6))
    return entropy

entropy_original = compute_entropy(image)
entropy_equalized = compute_entropy(equalized_image)
entropy_matched = compute_entropy(matched_image)

fig, axes = plt.subplots(1, 4, figsize=(15, 5))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image')
axes[1].imshow(equalized_image, cmap='gray')
axes[1].set_title(f'Equalized (Entropy: {entropy_equalized:.3f})')
axes[2].imshow(matched_image, cmap='gray')
axes[2].set_title(f'Matched (Entropy: {entropy_matched:.3f})')
axes[3].hist(image.ravel(), bins=256, range=(0, 256), color='r', alpha=0.7)
axes[3].hist(equalized_image.ravel(), bins=256, range=(0, 256), color='g', alpha=0.7)
```

```
axes[3].hist(matched_image.ravel(), bins=256, range=(0, 256), color='b', alpha=0.7)
axes[3].set_title('Histograms')

plt.tight_layout()
plt.show()

clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
clahe_image = clahe.apply(image)

fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Low Contrast Image')
axes[1].imshow(clahe_image, cmap='gray')
axes[1].set_title('Enhanced (CLAHE) Image')

plt.tight_layout()
plt.show()
```

SAMPLE INPUT-OUTPUT



Frequency Domain Processing

AIM

- Implement the 2D Discrete Fourier Transform (DFT) and its inverse.
- Design low-pass, high-pass, and band-pass filters in the frequency domain. Apply them to an image and analyze the results.
- Implement homomorphic filtering and apply it to an image with uneven illumination.

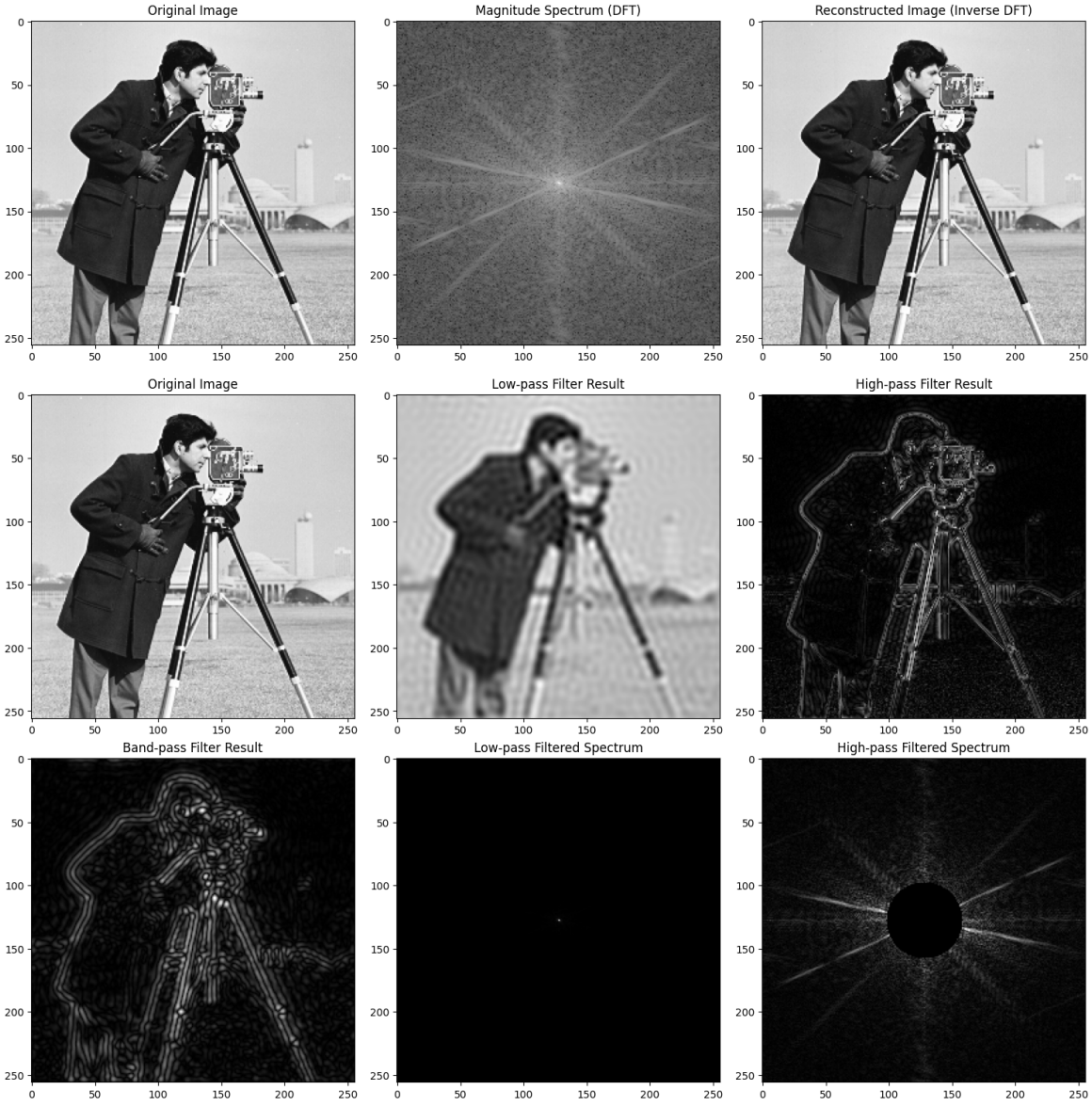
PROGRAM

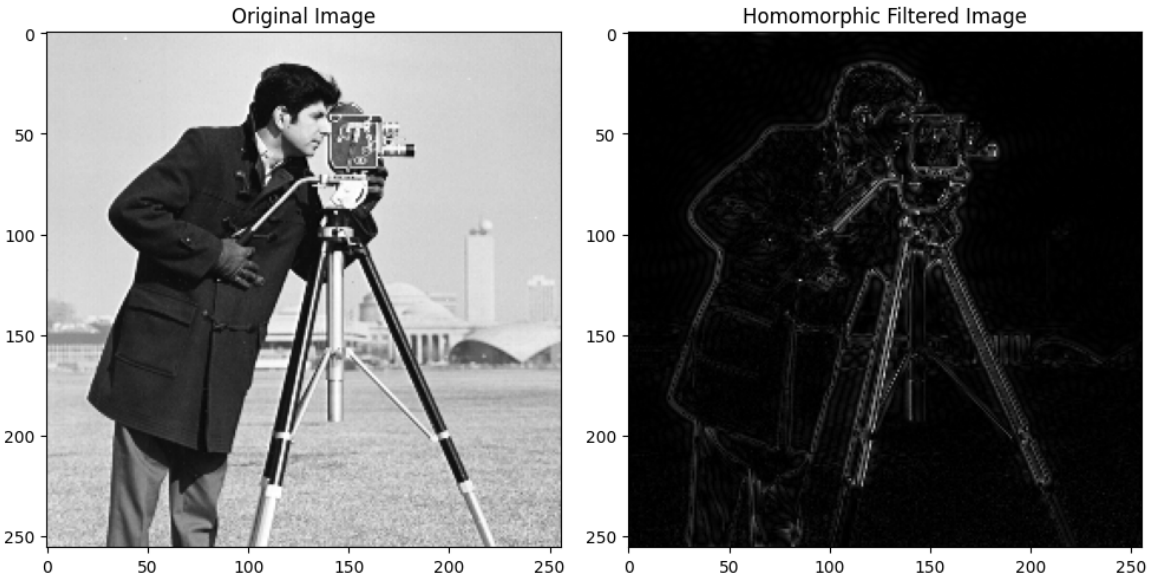
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

image_path = '/content/cameraman-testimage.png'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

dft = np.fft.fft2(image)
dft_shifted = np.fft.fftshift(dft)

magnitude_spectrum = np.abs(dft_shifted)
magnitude_spectrum = np.log(magnitude_spectrum + 1)

dft_ishifted = np.fft.ifftshift(dft_shifted)
image_reconstructed = np.fft.ifft2(dft_ishifted)
image_reconstructed = np.abs(image_reconstructed)

fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image')
axes[1].imshow(magnitude_spectrum, cmap='gray')
axes[1].set_title('Magnitude Spectrum (DFT)')
axes[2].imshow(image_reconstructed, cmap='gray')
axes[2].set_title('Reconstructed Image (Inverse DFT)')

plt.tight_layout()
plt.show()
rows, cols = image.shape
crow, ccol = rows // 2, cols // 2
```

```
low_pass_filter = np.zeros((rows, cols))
cv2.circle(low_pass_filter, (ccol, crow), 30, 1, -1)

high_pass_filter = np.ones((rows, cols))
cv2.circle(high_pass_filter, (ccol, crow), 30, 0, -1)

band_pass_filter = np.zeros((rows, cols))
cv2.circle(band_pass_filter, (ccol, crow), 50, 1, -1)
cv2.circle(band_pass_filter, (ccol, crow), 20, 0, -1)

low_pass_result = np.fft.ifft2(np.fft.ifftshift(dft_shifted * low_pass_filter))
high_pass_result = np.fft.ifft2(np.fft.ifftshift(dft_shifted * high_pass_filter))
band_pass_result = np.fft.ifft2(np.fft.ifftshift(dft_shifted * band_pass_filter))

fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes[0, 0].imshow(image, cmap='gray')
axes[0, 0].set_title('Original Image')
axes[0, 1].imshow(np.abs(low_pass_result), cmap='gray')
axes[0, 1].set_title('Low-pass Filter Result')
axes[0, 2].imshow(np.abs(high_pass_result), cmap='gray')
axes[0, 2].set_title('High-pass Filter Result')

axes[1, 0].imshow(np.abs(band_pass_result), cmap='gray')
axes[1, 0].set_title('Band-pass Filter Result')
axes[1, 1].imshow(np.abs(dft_shifted * low_pass_filter), cmap='gray')
axes[1, 1].set_title('Low-pass Filtered Spectrum')
axes[1, 2].imshow(np.abs(dft_shifted * high_pass_filter), cmap='gray')
axes[1, 2].set_title('High-pass Filtered Spectrum')

plt.tight_layout()
plt.show()
image_log = np.log1p(image)

dft_log = np.fft.fft2(image_log)
dft_log_shifted = np.fft.fftshift(dft_log)

high_pass_filter = np.ones((rows, cols))
cv2.circle(high_pass_filter, (ccol, crow), 30, 0, -1)

dft_filtered = dft_log_shifted * high_pass_filter
dft_filtered_ishifted = np.fft.ifftshift(dft_filtered)
```

```
image_filtered_log = np.fft.ifft2(dft_filtered_ishifted)
image_filtered = np.expm1(np.abs(image_filtered_log))

fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image')

axes[1].imshow(image_filtered, cmap='gray')
axes[1].set_title('Homomorphic Filtered Image')

plt.tight_layout()
plt.show()
```

SAMPLE INPUT-OUTPUT




Original Image
Homomorphic Filtered Image
0
50
100
150
200
250

Color Image Processing

AIM

- Implement color space conversions between RGB, HSI, and YCbCr color models.
- Perform color histogram equalization on a color image and analyze the results.
- Implement color edge detection using Sobel or Canny operators.

PROGRAM

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('/content/flowers.png')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

def rgb_to_hsi(rgb):
    rgb = rgb / 255.0
    R, G, B = rgb[:, :, 0], rgb[:, :, 1], rgb[:, :, 2]
    I = np.mean(rgb, axis=2)
    S = 1 - 3 * np.min(rgb, axis=2) / (R + G + B + 1e-6)
    numerator = 0.5 * ((R - G) + (R - B))
    denominator = np.sqrt((R - G)**2 + (R - B) * (G - B))
    H = np.arccos(numerator / (denominator + 1e-6))
    H[B > G] = 2 * np.pi - H[B > G]
    H = H / (2 * np.pi)
    return np.stack([H, S, I], axis=2)

def rgb_to_ycbcr(rgb):
    ycbcr = cv2.cvtColor(rgb.astype(np.uint8), cv2.COLOR_RGB2YCrCb)
    return ycbcr

image_hsi = rgb_to_hsi(image_rgb)
image_ycbcr = rgb_to_ycbcr(image_rgb)

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(image_rgb)
plt.title("Original RGB Image")
```

```
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(image_hsi)
plt.title("Converted to HSI")
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(image_ycbcr)
plt.title("Converted to YCbCr")
plt.axis('off')

plt.show()

r, g, b = cv2.split(image_rgb)
r_eq = cv2.equalizeHist(r)
g_eq = cv2.equalizeHist(g)
b_eq = cv2.equalizeHist(b)
image_eq = cv2.merge([r_eq, g_eq, b_eq])

plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(image_eq)
plt.title("Equalized Image")
plt.axis('off')
plt.show()

image_gray = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)
sobel_x = cv2.Sobel(image_gray, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(image_gray, cv2.CV_64F, 0, 1, ksize=3)
sobel_edges = np.hypot(sobel_x, sobel_y)
sobel_edges = np.uint8(np.absolute(sobel_edges))
canny_edges = cv2.Canny(image_gray, 100, 200)

plt.subplot(1, 2, 1)
plt.imshow(sobel_edges, cmap='gray')
plt.title("Sobel Edge Detection")
```

```
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(canny_edges, cmap='gray')
plt.title("Canny Edge Detection")
plt.axis('off')
plt.show()
```

SAMPLE INPUT-OUTPUT

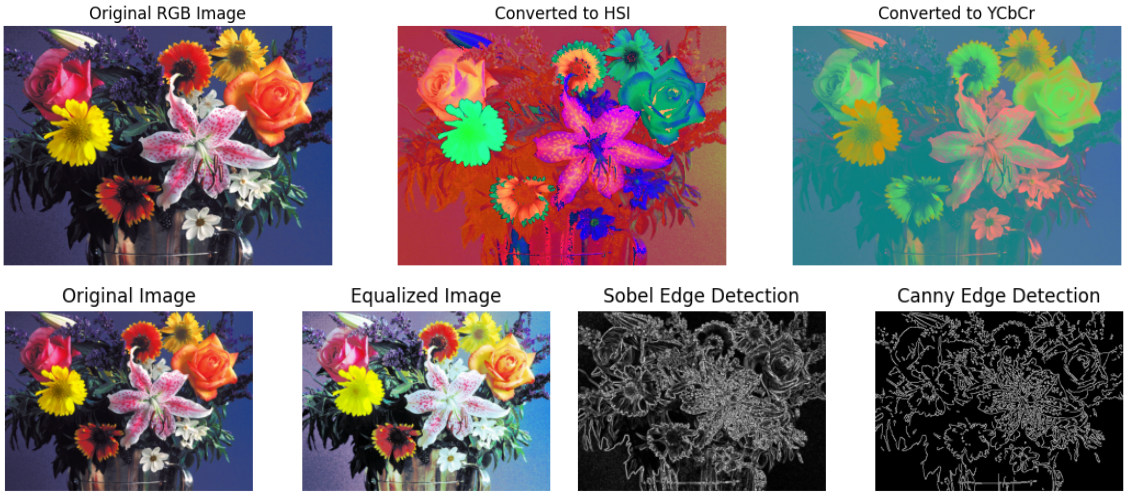


Image Segmentation

AIM

Implement thresholding, region-based, and edge-based segmentation techniques.

PROGRAM

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

image_path = '/content/cameraman-testimage.png'
if not os.path.exists(image_path):
    raise ValueError(f"File not found at {image_path}.
    Please check the file path or upload the image.")
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
if image is None:
    raise ValueError("Unable to load image. Check the path.")

_, thresholded_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
ret, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
if binary_image is None or binary_image.size == 0:
    raise ValueError("Thresholding failed, binary image is empty.")
kernel = np.ones((3, 3), np.uint8)
binary_image = cv2.dilate(binary_image, kernel, iterations=3)

dist_transform = cv2.distanceTransform(binary_image, cv2.DIST_L2, 5)
_, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)
sure_fg = np.uint8(sure_fg)
sure_bg = cv2.dilate(binary_image, kernel, iterations=3)
unknown = cv2.subtract(sure_bg, sure_fg)

_, markers = cv2.connectedComponents(sure_fg)
markers = markers + 1
markers[unknown == 255] = 0
image_colored = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
cv2.watershed(image_colored, markers)
image_colored[markers == -1] = [0, 0, 255]

sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
```

```
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
sobel_edges = cv2.magnitude(sobel_x, sobel_y)
sobel_edges = np.uint8(np.absolute(sobel_edges))

robert_kernel_x = np.array([[1, 0], [0, -1]], dtype=np.float32)
robert_kernel_y = np.array([[0, 1], [-1, 0]], dtype=np.float32)
robert_x = cv2.filter2D(image, -1, robert_kernel_x)
robert_y = cv2.filter2D(image, -1, robert_kernel_y)
robert_edges = np.sqrt(np.square(robert_x) + np.square(robert_y))
robert_edges = np.uint8(np.absolute(robert_edges))

prewitt_kernel_x = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]], dtype=np.float32)
prewitt_kernel_y = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]], dtype=np.float32)
prewitt_x = cv2.filter2D(image, -1, prewitt_kernel_x)
prewitt_y = cv2.filter2D(image, -1, prewitt_kernel_y)
prewitt_edges = np.sqrt(np.square(prewitt_x) + np.square(prewitt_y))
prewitt_edges = np.uint8(np.absolute(prewitt_edges))
edges_canny = cv2.Canny(image, 100, 200)
plt.figure(figsize=(12, 12))
plt.subplot(3, 3, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis('off')

plt.subplot(3, 3, 2)
plt.imshow(thresholded_image, cmap='gray')
plt.title("Thresholding Segmentation")
plt.axis('off')

plt.subplot(3, 3, 3)
plt.imshow(image_colored)
plt.title("Region-based Segmentation (Watershed)")
plt.axis('off')

plt.subplot(3, 3, 4)
plt.imshow(sobel_edges, cmap='gray')
plt.title("Sobel Edge Detection")
plt.axis('off')

plt.subplot(3, 3, 5)
plt.imshow(robert_edges, cmap='gray')
```

```
plt.title("Robert Edge Detection")
plt.axis('off')

plt.subplot(3, 3, 6)
plt.imshow(prewitt_edges, cmap='gray')
plt.title("Prewitt Edge Detection")
plt.axis('off')

plt.subplot(3, 3, 7)
plt.imshow(edges_canny, cmap='gray')
plt.title("Canny Edge Detection")
plt.axis('off')
plt.tight_layout()
plt.show()
```

SAMPLE INPUT-OUTPUT

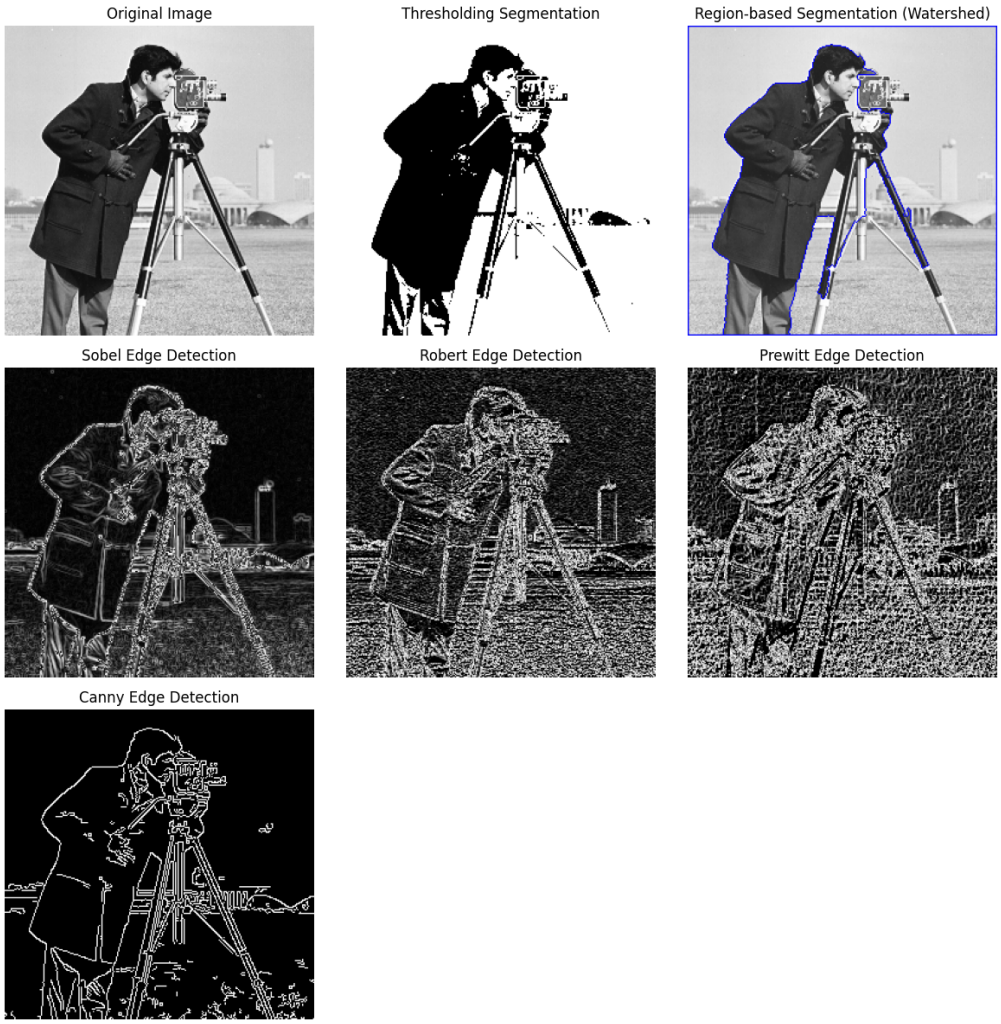


Image Morphological Processing

AIM

Perform erosion, dilation, opening, and closing operations on binary images.

PROGRAM

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path = '/content/Lenna_(test_image).png'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

if image is None:
    raise ValueError(f"Unable to load image. Check the file path: {image_path}")

kernel = np.ones((5, 5), np.uint8)

erosion = cv2.erode(image, kernel, iterations=1)
dilation = cv2.dilate(image, kernel, iterations=1)
opening = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
closing = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)

plt.figure(figsize=(10, 6))

plt.subplot(2, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(erosion, cmap='gray')
plt.title('Erosion')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(dilation, cmap='gray')
plt.title('Dilation')
plt.axis('off')
```

```
plt.subplot(2, 3, 4)
plt.imshow(opening, cmap='gray')
plt.title('Opening')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(closing, cmap='gray')
plt.title('Closing')
plt.axis('off')

plt.tight_layout()
plt.show()
```

SAMPLE INPUT-OUTPUT

Image Registration

AIM

Implement image registration techniques for aligning multiple images.

PROGRAM

```
import cv2
from matplotlib import pyplot as plt

def split_collage(image):
    height, width = image.shape[:2]
    mid_width = width // 2
    img1 = image[:, :mid_width]
    img2 = image[:, mid_width:]
    return img1, img2

collage = cv2.imread('/content/The-same-face-under-two-different-lighting-conditions.png',
cv2.IMREAD_COLOR)

image1, image2 = split_collage(collage)

image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)

orb = cv2.ORB_create()

kp1, des1 = orb.detectAndCompute(image1, None)
kp2, des2 = orb.detectAndCompute(image2, None)

bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1, des2)

matches = sorted(matches, key=lambda x: x.distance)

line_thickness = 0.0001

image3 = cv2.drawMatches(
    image1, kp1, image2, kp2, matches[:10], None,
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
)
```

```
plt.figure(figsize=(10, 5))
plt.imshow(image3)
plt.axis('off')
plt.show()
```

SAMPLE INPUT-OUTPUT