Lab Report 7 : Dilation, Erosion, Opening and Closing Operation, FFT, Homomorphic Transform of an Image

Course Title: Digital Image Processing Lab

Course Code: CSE-406

Date of Performance: October 06, 2024
Date of Submission: October 20, 2024

Submitted to:

Dr. Morium Akter
Professor

Dr. Md. Golam Moazzam
Professor

Department of Computer Science and Engineering
Jahangirnagar University

Submitted by:

Class Roll	Exam Roll	Name
370	202182	Rubayed All Islam

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka, Bangladesh

Experiment Name

a) **Dilation of an image**

b) **Erosion operation of an image**

c) **Opening and closing operation of an image**

d) **Fast Fourier Transform (FFT) operation of an image**

e) **Homomorphic operation of an image**

Objectives

The objectives of the experiments performed in this lab are as follows:

1. **Dilation of an Image**
   - To enhance the boundaries of objects in an image by expanding the white regions, allowing for better visibility of object contours and connectivity of broken parts.
2. **Erosion Operation of an Image**
   - To shrink the boundaries of objects by removing small-scale noise and detaching objects that are connected by thin lines or points, reducing overall noise in the image.
3. **Opening and Closing Operation of an Image**
   - To perform noise reduction and smoothing of image structures:
     - **Opening**: To remove small noise elements while maintaining the shape and size of the larger structures.
     - **Closing**: To fill small holes and gaps within objects, improving the image's structure and continuity.
4. **Fast Fourier Transform (FFT) Operation of an Image**
   - To transform an image into its frequency domain representation, allowing the analysis of its frequency components and facilitating operations like filtering and noise reduction.
5. **Homomorphic Operation of an Image**
   - To enhance the contrast and brightness of an image by applying a homomorphic filter, which allows for simultaneous dynamic range compression and contrast enhancement, particularly useful for improving illumination variations in an image.

Methodology

This section outlines the methodology followed in each experiment.

1. **Dilation of an Image**
   - Read the input image using the OpenCV `imread()` function.
   - Convert the image to grayscale if required.
   - Define a structuring element (kernel) for dilation using the `getStructuringElement()` function.
   - Apply the dilation operation using `cv2.dilate()`.
   - Display the original and dilated images side by side using Matplotlib.
2. **Erosion Operation of an Image**
   - Load the input image using OpenCV.
   - Define a structuring element (kernel) similar to dilation.
   - Perform the erosion operation using `cv2.erode()` to shrink the objects in the image.
   - Visualize the original and eroded images side by side for comparison.
3. **Opening and Closing Operation of an Image**
   - Use the same image and kernel as used for dilation and erosion.
   - For opening:
     - Perform erosion followed by dilation using `cv2.morphologyEx()` with the `MORPH_OPEN` option.
   - For closing:
     - Apply dilation followed by erosion using `cv2.morphologyEx()` with the `MORPH_CLOSE` option.
   - Display the original, opened, and closed images in a single plot.
4. **Fast Fourier Transform (FFT) Operation of an Image**
   - Load the input image using OpenCV.
   - Convert the image to grayscale or keep it in RGB.
   - Apply FFT to each color channel using the `np.fft.fft2()` function.
   - Shift the zero-frequency component to the center using `np.fft.fftshift()`.
   - Compute the magnitude spectrum and visualize it using Matplotlib.
   - Plot the original image and its frequency spectrum side by side.
5. **Homomorphic Operation of an Image**
   - Convert the image to the frequency domain using FFT.
   - Apply a high-pass filter to enhance high-frequency details while suppressing low-frequency components.

- Apply the inverse FFT to convert the image back to the spatial domain.
- Adjust the contrast and brightness of the result to enhance the image appearance.
- Display the original and processed images.

Experiment 1: Dilation of an image

MATLAB Code:

Listing 1: Dilation_Operation.m

```
% Read the original image
originalImage = imread('baby2.jpg');
% Define a structuring element for dilation
se = strel('disk', 5); % You can adjust the size of the disk
% Initialize an output image
dilatedImage = zeros(size(originalImage), 'like', originalImage);
% Perform dilation on each channel
for channel = 1:size(originalImage, 3)
    dilatedImage(:, :, channel) = imdilate(originalImage(:, :,
       channel), se);
end
% Create a figure to display the images
figure;
% Display the original image
subplot(1, 2, 1);
imshow(originalImage);
title('Original Image');
% Display the dilated image
subplot(1, 2, 2);
imshow(dilatedImage);
title('Dilated Image');
```

Python Code:

Listing 2: Dilation_Operation.py

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

# Read the original image
original_image = cv2.imread('baby2.jpg')
original_image_rgb = cv2.cvtColor(original_image,
   cv2.COLOR_BGR2RGB)

# Define a structuring element for dilation
se = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))

```

```
# Perform dilation on each channel
dilated_image = np.zeros_like(original_image_rgb)
for channel in range(3):
    dilated_image[:, :, channel] =
        cv2.dilate(original_image_rgb[:, :, channel], se)

# Display the images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(original_image_rgb)
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(dilated_image)
plt.title('Dilated Image')
plt.show()
```

Output:

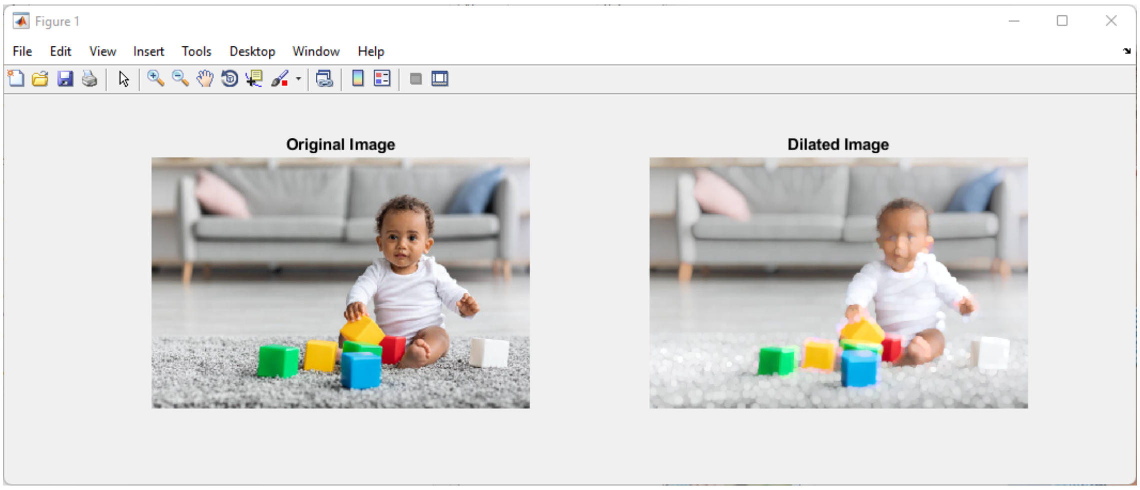

Figure 1: Output of Dilation of an Image

Experiment 2: Erosion operation of an image

MATLAB Code:

Listing 3: Erosion_Operation.m

```
% Read the original image
originalImage = imread('baby2.jpg');
% Define a structuring element for erosion
se = strel('disk', 5); % You can adjust the size of the disk
% Initialize an output image
erodedImage = zeros(size(originalImage), 'like', originalImage);
% Perform erosion on each channel
for channel = 1:size(originalImage, 3)
    erodedImage(:, :, channel) = imerode(originalImage(:, :,
        channel), se);
end
% Create a figure to display the images
figure;
% Display the original image
subplot(1, 2, 1);
imshow(originalImage);
title('Original Image');
% Display the eroded image
subplot(1, 2, 2);
imshow(erodedImage);
title('Eroded Image')
```

Python Code:

Listing 4: Erosion_Operation.py

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

# Load the image
original_image_rgb = cv2.imread('baby2.jpg')

# Check if the image is loaded properly
if original_image_rgb is None:
    print("Error: Could not read the image. Please check the
        file path.")
    exit()

# Convert the image from BGR (OpenCV format) to RGB (matplotlib
  format)
original_image_rgb = cv2.cvtColor(original_image_rgb,
  cv2.COLOR_BGR2RGB)

# Define the structuring element (disk-shaped) for erosion
```

```
se = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))

# Perform erosion on each channel
eroded_image = np.zeros_like(original_image_rgb)
for channel in range(3):
    eroded_image[:, :, channel] = cv2.erode(original_image_rgb[:, :, channel], se)

# Display the images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(original_image_rgb)
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(eroded_image)
plt.title('Eroded Image')
plt.show()
```

Output:

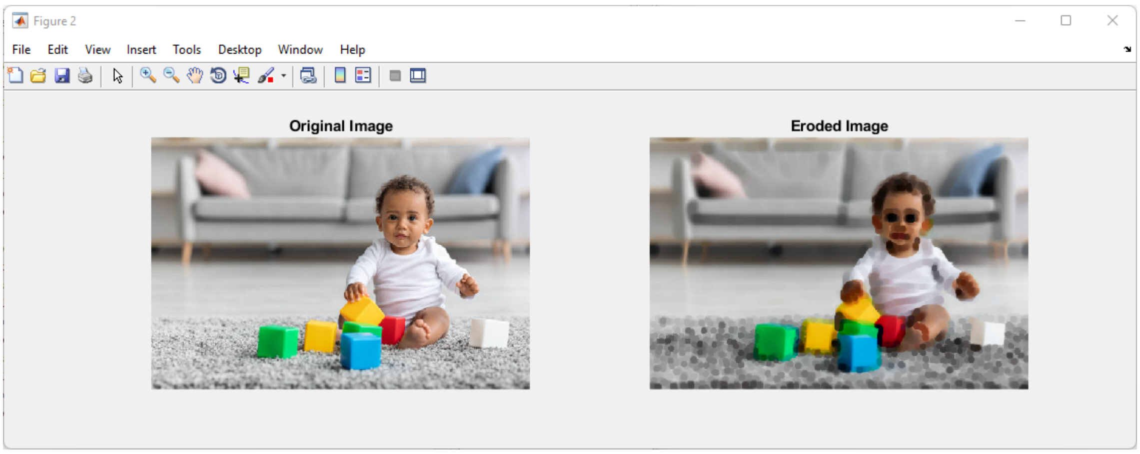

Figure 2: Output of Erosion operation of an Image

Experiment 3: Opening and closing operation of an image

MATLAB Code:

Listing 5: Opening_Closing.m

```
% Read the original image
originalImage = imread('baby2.jpg');
% Define a structuring element
se = strel('disk', 5); % You can adjust the size of the disk
% Initialize output images
openedImage = zeros(size(originalImage), 'like', originalImage);
closedImage = zeros(size(originalImage), 'like', originalImage);
% Perform Opening and Closing on each channel
for channel = 1:size(originalImage, 3)
    openedImage(:, :, channel) = imopen(originalImage(:, :,
        channel), se);
    closedImage(:, :, channel) = imclose(originalImage(:, :,
        channel), se);
end
% Create a figure to display the images
figure;
% Display the original image
subplot(1, 3, 1);
imshow(originalImage);
title('Original Image');
% Display the opened image
subplot(1, 3, 2);
imshow(openedImage);
title('Opened Image');
% Display the closed image
subplot(1, 3, 3);
imshow(closedImage);
title('Closed Image');
```

Python Code:

Listing 6: Opening_Closing.py

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

# Load the image
original_image_rgb = cv2.imread('baby2.jpg')
# Convert the image from BGR (OpenCV format) to RGB (matplotlib
    format)
original_image_rgb = cv2.cvtColor(original_image_rgb,
    cv2.COLOR_BGR2RGB)
```

```python

# Define the structuring element (disk-shaped)
se = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))

# Perform Opening and Closing on each channel
opened_image = np.zeros_like(original_image_rgb)
closed_image = np.zeros_like(original_image_rgb)
for channel in range(3):
    opened_image[:, :, channel] =
       cv2.morphologyEx(original_image_rgb[:, :, channel],
       cv2.MORPH_OPEN, se)
    closed_image[:, :, channel] =
       cv2.morphologyEx(original_image_rgb[:, :, channel],
       cv2.MORPH_CLOSE, se)

# Display the images
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(original_image_rgb)
plt.title('Original Image')
plt.subplot(1, 3, 2)
plt.imshow(opened_image)
plt.title('Opened Image')
plt.subplot(1, 3, 3)
plt.imshow(closed_image)
plt.title('Closed Image')
plt.show()
```

Output:

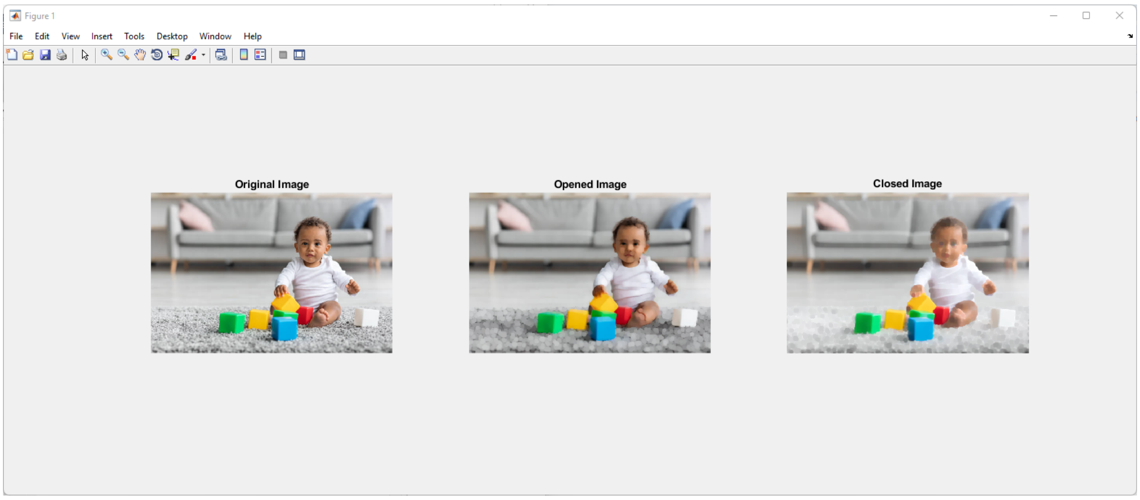

Figure 3: Output of Opening and Closing Operation of an Image

Experiment 4: Fast Fourier Transform (FFT) operation of an image

MATLAB Code:

Listing 7: FFT.m

```
% List of image files to process
imageFiles = {'moody.jpg', 'baby2.jpg', 'flower.jpg'}; % Add
    more image file names if needed
numImages = length(imageFiles);
% Create a figure to display the results
figure;
% Loop through each image file
for i = 1:numImages
    % Read the image
    originalImage = imread(imageFiles{i});

    % Check if the image has multiple color channels
    if size(originalImage, 3) == 3
        % Perform 2D FFT for each channel individually
        fftImageR = fft2(double(originalImage(:,:,1)));
        fftImageG = fft2(double(originalImage(:,:,2)));
        fftImageB = fft2(double(originalImage(:,:,3)));
        % Shift the zero frequency component to the center for
           each channel
        fftImageShiftedR = fftshift(fftImageR);
        fftImageShiftedG = fftshift(fftImageG);
        fftImageShiftedB = fftshift(fftImageB);
        % Compute the magnitude spectrum for each channel
        magnitudeSpectrumR = log(1 + abs(fftImageShiftedR)); %
           Red channel
        magnitudeSpectrumG = log(1 + abs(fftImageShiftedG)); %
           Green channel
        magnitudeSpectrumB = log(1 + abs(fftImageShiftedB)); %
           Blue channel
        % Compute the average magnitude spectrum for RGB channels
        magnitudeSpectrumAvg = (magnitudeSpectrumR +
           magnitudeSpectrumG + magnitudeSpectrumB) / 3;

    else
        % Perform FFT on the grayscale image
        fftImage = fft2(double(originalImage));
        fftImageShifted = fftshift(fftImage);
        magnitudeSpectrumAvg = log(1 + abs(fftImageShifted));
    end
    % Display the original image
    subplot(numImages, 2, 2*i-1);
    imshow(originalImage);
    title(['Original Image ', num2str(i)]);

```

```
    % Display the average magnitude spectrum (log scale)
    subplot(numImages, 2, 2*i);
    imshow(magnitudeSpectrumAvg, []);
    title(['Magnitude Spectrum ', num2str(i)]);
end
```

Python Code:

Listing 8: FFT.py

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

# List of image files to process
image_files = ['moody.jpg', 'baby2.jpg', 'flower.jpg']

# Create a figure to hold all the subplots
plt.figure(figsize=(15, 10))

# Loop through each image file
for i, image_file in enumerate(image_files):
    # Load the image
    original_image = cv2.imread(image_file)
    original_image_rgb = cv2.cvtColor(original_image,
       cv2.COLOR_BGR2RGB)

    # Check if it's a color image
    if original_image_rgb.shape[2] == 3:
        # Perform FFT for each channel
        fft_images = [np.fft.fft2(original_image_rgb[:, :,
           channel]) for channel in range(3)]
        fft_images_shifted = [np.fft.fftshift(fft_image) for
           fft_image in fft_images]
        magnitude_spectrum = [np.log(1 + np.abs(fft_shifted))
           for fft_shifted in fft_images_shifted]
        magnitude_spectrum_avg = sum(magnitude_spectrum) / 3
    else:
        fft_image = np.fft.fft2(original_image_rgb)
        fft_image_shifted = np.fft.fftshift(fft_image)
        magnitude_spectrum_avg = np.log(1 +
           np.abs(fft_image_shifted))

    # Plot the original image
    plt.subplot(len(image_files), 2, 2 * i + 1)  # Row:
       len(image_files), Col: 2, Pos: i*2+1
    plt.imshow(original_image_rgb)
    plt.title(f'Original Image {i+1}')
    plt.axis('off')  # Hide axis

    # Plot the magnitude spectrum
```

```
    plt.subplot(len(image_files), 2, 2 * i + 2)  # Row:
        len(image_files), Col: 2, Pos: i*2+2
    plt.imshow(magnitude_spectrum_avg, cmap='gray')
    plt.title(f'Magnitude Spectrum {i+1}')
    plt.axis('off')  # Hide axis

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```

Output:

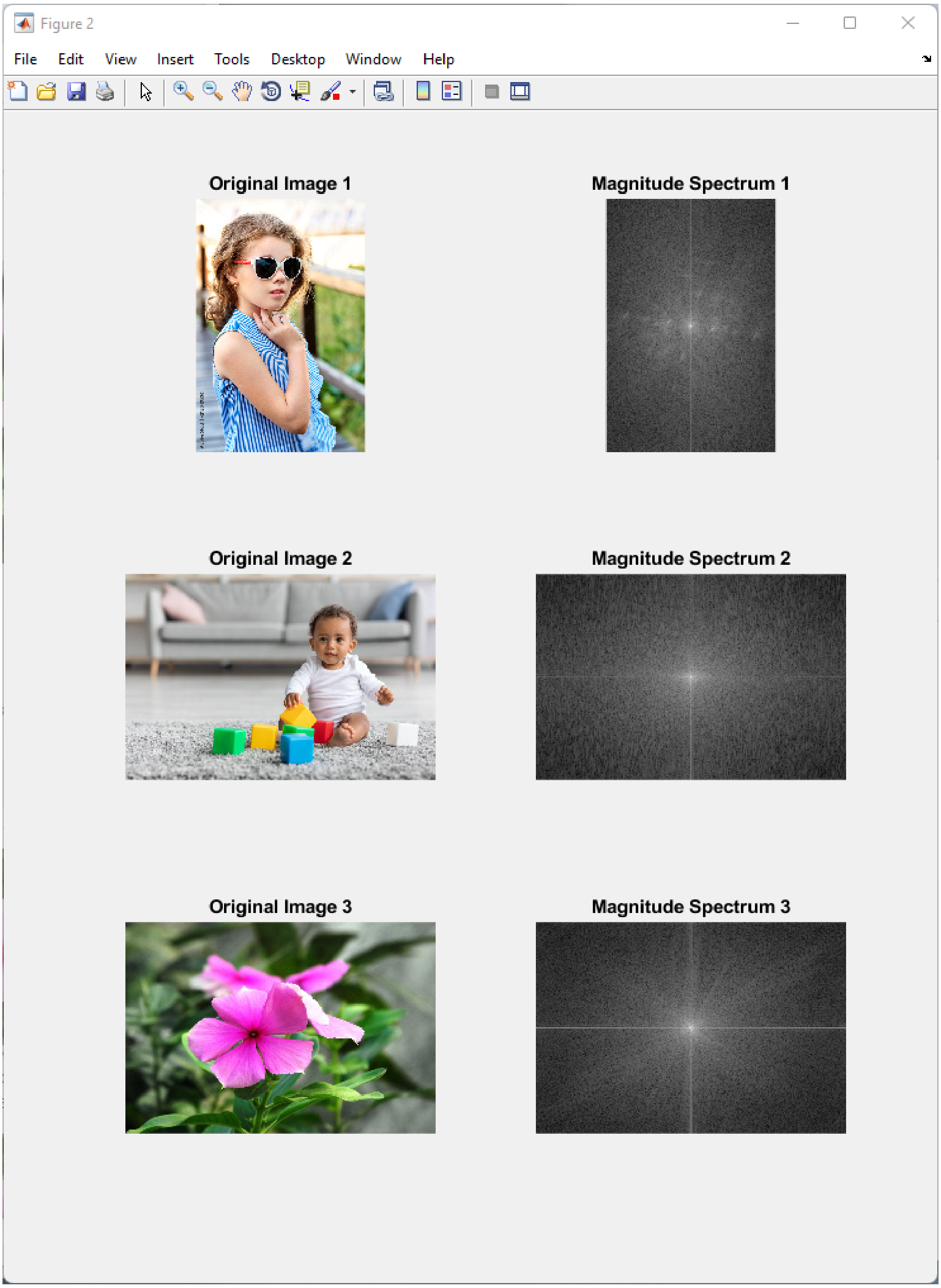

Figure 4: Output of FFT operation of an image

Experiment 5: Homomorphic operation of an image

MATLAB Code:

Listing 9: Homomorphic_Transform.m

```
% Read the original image
originalImage = imread('moody.jpg');
% Convert the image to double for processing
originalImage = double(originalImage) + 1; % Add 1 to avoid
    log(0)
% Get the dimensions of the image
[rows, cols, channels] = size(originalImage);
% Initialize output image
outputImage = zeros(size(originalImage));
% Perform homomorphic filtering on each channel
for channel = 1:channels
    % Perform the FFT
    fftImage = fft2(originalImage(:, :, channel));
    fftImageShifted = fftshift(fftImage);

    % Get the magnitude and phase
    magnitude = abs(fftImageShifted);
    phase = angle(fftImageShifted);

    % Define a Gaussian filter in the frequency domain
    crow = round(rows/2);
    ccol = round(cols/2);
    [x, y] = meshgrid(1:cols, 1:rows);
    sigma = 30; % Standard deviation for Gaussian filter
    gaussianFilter = exp(-((x - ccol).^2 + (y - crow).^2) / (2 *
        sigma^2));

    % Apply the filter to the magnitude
    filteredMagnitude = magnitude .* gaussianFilter;

    % Create a new complex image with filtered magnitude and
        original phase
    homomorphicImage = filteredMagnitude .* exp(1i * phase);

    % Perform the inverse FFT
    homomorphicImageShifted = ifftshift(homomorphicImage);
    outputImage(:, :, channel) = ifft2(homomorphicImageShifted);
end
% Take the real part and normalize
outputImage = real(outputImage);
outputImage = mat2gray(outputImage); % Normalize to [0, 1]
% Create a figure to display the images
figure;
% Display the original image
subplot(1, 2, 1);
imshow(uint8(originalImage)); % Convert back to uint8 for display
```

```
title('Original Image');
% Display the homomorphic filtered image
subplot(1, 2, 2);
imshow(outputImage);
title('Homomorphic Filtered Image');
```

Python Code:

Listing 10: Homomorphic_Transform.py

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

# Read the original image
original_image = cv2.imread('moody.jpg')
original_image_rgb = cv2.cvtColor(original_image,
    cv2.COLOR_BGR2RGB)
original_image_double = np.float64(original_image_rgb) + 1  #
    Convert to double and avoid log(0)

# Perform homomorphic filtering on each channel
rows, cols, channels = original_image_double.shape
output_image = np.zeros_like(original_image_double)

for channel in range(3):
    # Perform FFT
    fft_image = np.fft.fft2(original_image_double[:, :, channel])
    fft_image_shifted = np.fft.fftshift(fft_image)

    # Get the magnitude and phase
    magnitude = np.abs(fft_image_shifted)
    phase = np.angle(fft_image_shifted)

    # Define a Gaussian filter in the frequency domain
    crow, ccol = rows // 2, cols // 2
    x, y = np.meshgrid(np.arange(cols), np.arange(rows))
    sigma = 30  # Standard deviation for Gaussian filter
    gaussian_filter = np.exp(-((x - ccol) ** 2 + (y - crow) **
        2) / (2 * sigma ** 2))

    # Apply the filter to the magnitude
    filtered_magnitude = magnitude * gaussian_filter

    # Create a new complex image with filtered magnitude and
        original phase
    homomorphic_image = filtered_magnitude * np.exp(1j * phase)

    # Perform inverse FFT
    homomorphic_image_shifted =
        np.fft.ifftshift(homomorphic_image)
```

```
    output_image[:, :, channel] =
       np.fft.ifft2(homomorphic_image_shifted).real

# Normalize to [0, 1]
output_image_normalized = cv2.normalize(output_image, None, 0,
   1, cv2.NORM_MINMAX)

# Display the images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(original_image_rgb)
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(output_image_normalized)
plt.title('Homomorphic Filtered Image')
plt.show()
```

Output:

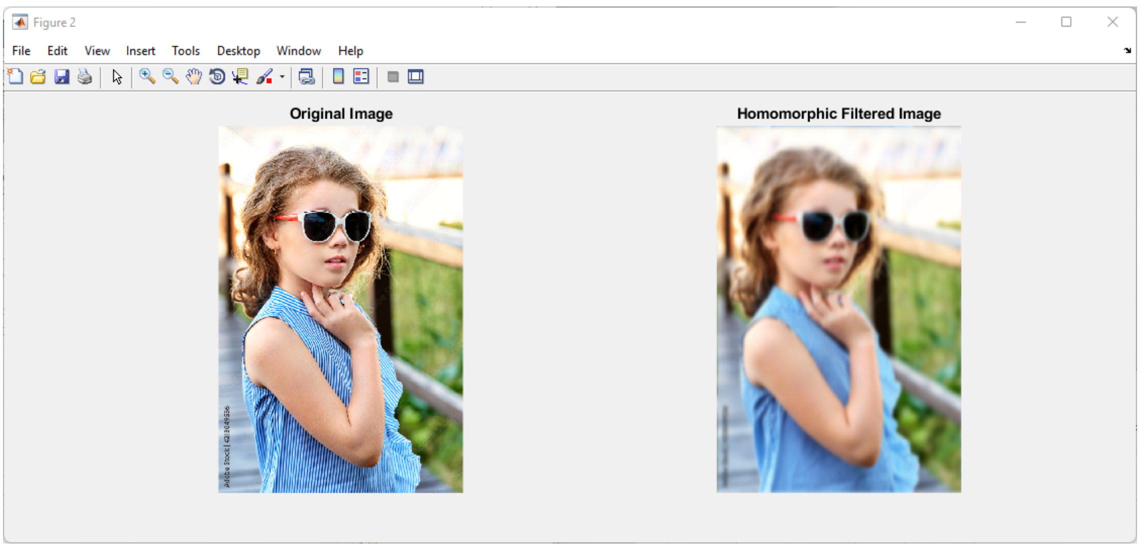

Figure 5: Output of Homomorphic operation of an image

Results

This section presents the results of the experiments performed during the lab. Each image was processed using the techniques discussed earlier, and the outcomes are displayed below.

1. **Dilation of an Image**
    - The dilation operation successfully expanded the bright areas in the image. Objects in the image appeared larger, with gaps between them getting filled. This is evident from the comparison between the original image and the dilated image.
    - The results demonstrate how dilation can enhance the prominence of certain features, making them more visible.
2. **Erosion Operation of an Image**
    - In the erosion operation, the boundaries of the objects in the image were shrunk. This operation reduced the size of the bright regions, effectively removing small noise and disconnecting weakly connected regions.
    - The result image shows reduced thickness of objects, highlighting how erosion is useful in minimizing small details and noise.
3. **Opening and Closing Operation of an Image**
    - The opening operation, which consists of erosion followed by dilation, removed small noise and detached unconnected parts of the image. The result was a smoother and cleaner image, particularly effective in eliminating noise.
    - The closing operation, where dilation was followed by erosion, successfully filled small holes and gaps in the objects. The outcome maintained the size of the objects while closing minor gaps in the image.
    - Both operations demonstrated their utility in image preprocessing, with opening cleaning up noise and closing preserving the object structure while filling gaps.
4. **Fast Fourier Transform (FFT) Operation of an Image**
    - The FFT operation transformed the image into the frequency domain, revealing the frequency components of the image. The magnitude spectrum provided insight into the distribution of frequencies, with low frequencies concentrated in the center.
    - The results illustrate how high-frequency details correspond to edges and sharp transitions in the image, while low-frequency components contribute to overall image structure.
    - The magnitude spectrum of each image displayed a distinct pattern, highlighting the distribution of spatial frequencies.
5. **Homomorphic Operation of an Image**

- The homomorphic filtering operation enhanced the image by reducing low-frequency components (illumination) and emphasizing high-frequency components (details). The resulting image had improved contrast and highlighted important details.
- The result demonstrated how homomorphic filtering can improve the visibility of fine details in images with non-uniform lighting, making it a powerful tool for image enhancement.

Conclusion

This lab explored essential image processing techniques, including dilation, erosion, opening and closing operations, Fast Fourier Transform (FFT), and homomorphic filtering. Each method demonstrated its effectiveness in enhancing image quality and extracting relevant features.

Dilation and erosion were used to modify object shapes, while opening and closing operations effectively removed noise and preserved essential details. The FFT allowed for frequency domain analysis, revealing underlying structures within images. Lastly, homomorphic filtering improved contrast and visibility, particularly in unevenly lit images.

Overall, these techniques are foundational for advanced image processing applications in various fields, emphasizing their significance in enhancing visual data for analysis.