PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Intramuros, Manila

Elective 3

Laboratory Activity No. 4
Image Restoration

Score

Submitted by:
Mayor, Ann Jossan D. – Leader
Biol, Lorenz Jay Von P.
Lumane, Kyla L.
Sison, Dan Jedrick S.
Tenoria, Karl John Dave C.

SAT 7AM – 4PM / CPE 0332.1-1

Date Submitted
10-08-2024

Submitted to:

Engr. Maria Rizette H. Sayo

I. Objectives

This laboratory activity aims to implement the principles and techniques of image restoration through MATLAB/Octave and open CV using Python

1. Acquire the image.
2. Show Gaussian filter for Image Restoration.
3. Show Deblurring (motion blur removal).

II. Methods

A. Perform a task given in the presentation

- Copy and paste your MATLAB code (use the original picture file: flower.jpg)

```
% Read the image
img = imread('original image'); % Replace with the path to your image file

% Display the original image
figure;
imshow(img);
title('Original Image');


% Convert to grayscale if the image is RGB
if size(img, 3) == 3
    img_gray = rgb2gray(img);
else
    img_gray = img;
end


% Display the grayscale image
figure;
imshow(img_gray);
title('Grayscale');


% Add blur to the image
len = 21;
theta = 11;
psf = fspecial('motion', len, theta);
img_blur = imfilter(img_gray, psf, 'conv', 'circular');




% Show the image
```

```
figure;
imshow(img_blur);
title('Motion Blurred Image');



% Filtering Techniques

% Gaussian filtering
h_gaussian = fspecial('gaussian', [5, 5], 1);
img_gaussian_filtered = imfilter(img_blur, h_gaussian);

% Display the Gaussian filtered image
figure;
imshow(img_gaussian_filtered);
title('Filtered Image (Gaussian)');


% Sharpening using unsharp masking
img_sharpened = imsharpen(img_blur);

% Display the sharpened image
figure;
imshow(img_sharpened);
title('Sharpened Image');

% Add Gaussian noise and remove it using median filter
img_noisy = imnoise(img_gray, 'gaussian', 0.02);
img_noisy_removed = medfilt2(img_noisy, [5, ]);

% Display the noise image
figure;
imshow(img_noisy);
title('Noisy');

% Display the noise-removed images
figure;
imshow(img_noisy_removed);
title('Noise Removed');



% Deblurring

estimated_nsr = 0.01;
img_deblurred = deconvwnr(img_blur, psf, estimated_nsr);
figure;
imshow(img_deblurred);
title("Deblurred Image");
```

B. Supplementary Activity

- Write a Python program that will implement the output in Method A.

```
- import cv2
- import numpy as np
- from scipy.ndimage import convolve, gaussian_filter
- from skimage import restoration
- import matplotlib.pyplot as plt
-
- # Read the image
- img = cv2.imread('flower.jpg')
-
- # Convert the image from BGR to RGB (since OpenCV reads in BGR format)
- img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
-
- # Display the original image
- plt.figure(1)
- plt.imshow(img)
- plt.title('Original Image')
- plt.axis('off')
-
- # Convert to grayscale if the image is RGB
- if len(img.shape) == 3:
-     img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
- else:
-     img_gray = img
-
- # Display the grayscale image
- plt.figure(2)
- plt.imshow(img_gray, cmap='gray')
- plt.title('Grayscale')
- plt.axis('off')
-
- # Add blur to the image (motion blur)
- len = 21
- theta = 11
- psf = np.zeros((len, len))
- center = len // 2
- for i in range(len):
-     offset = int(np.round((i - center)  np.tan(np.radians(theta))))
-     if 0 <= center + offset < len:
-         psf[i, center + offset] = 1
- psf /= psf.sum()
-
- img_blur = convolve(img_gray, psf)
```

```
-
-   # Show the motion blurred image
-   plt.figure(3)
-   plt.imshow(img_blur, cmap='gray')
-   plt.title('Motion Blurred Image')
-   plt.axis('off')
-
-   # Gaussian filtering
-   img_gaussian_filtered = gaussian_filter(img_blur, sigma=1)
-
-   # Display the Gaussian filtered image
-   plt.figure(4)
-   plt.imshow(img_gaussian_filtered, cmap='gray')
-   plt.title('Filtered Image (Gaussian)')
-   plt.axis('off')
-
-   # Sharpening using unsharp masking
-   img_sharpened = cv2.addWeighted(img_blur, 1.5, gaussian_filter(img_blur,
    sigma=1), -0.5, 0)
-
-   # Display the sharpened image
-   plt.figure(5)
-   plt.imshow(img_sharpened, cmap='gray')
-   plt.title('Sharpened Image')
-   plt.axis('off')
-
-   # Add Gaussian noise and remove it using a median filter
-   img_noisy = img_gray + np.random.normal(0, 25,
    img_gray.shape).astype(np.uint8)
-   img_noisy = np.clip(img_noisy, 0, 255)
-   img_noisy_removed = cv2.medianBlur(img_noisy, 5)
-
-   # Display the noisy image
-   plt.figure(6)
-   plt.imshow(img_noisy, cmap='gray')
-   plt.title('Noisy')
-   plt.axis('off')
-
-   # Display the noise-removed image
-   plt.figure(7)
-   plt.imshow(img_noisy_removed, cmap='gray')
-   plt.title('Noise Removed')
-   plt.axis('off')
```

```
-
-   # Deblurring using Wiener filter
-   estimated_nsr = 0.01
-   img_deblurred = restoration.wiener(img_blur, psf, estimated_nsr)
-
-   # Display the deblurred image
-   plt.figure(8)
-   plt.imshow(img_deblurred, cmap='gray')
-   plt.title('Deblurred Image')
-   plt.axis('off')
-
-   plt.show()
-
```

III. Results

Steps:

1. Copy/crop and paste your results. Label each output (Figure1, Figure2, Figure3, Figure 4, and Figure 5)

picture file: flower.jpg

MATLAB OUTPUT:

Figure 1: Acquire an Image of a Flower

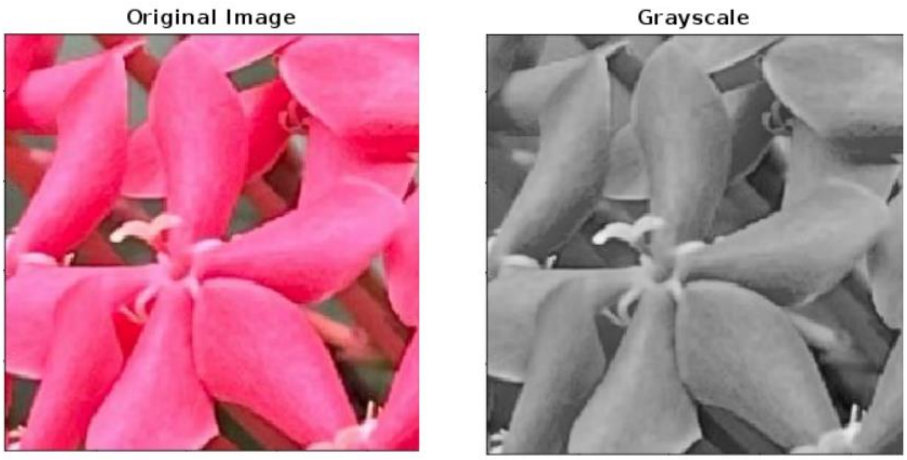


Figure 2: Original and Grayscale Image

Figure 3: Motion Blurred Image

Figure 4: Gaussian-filtered Image

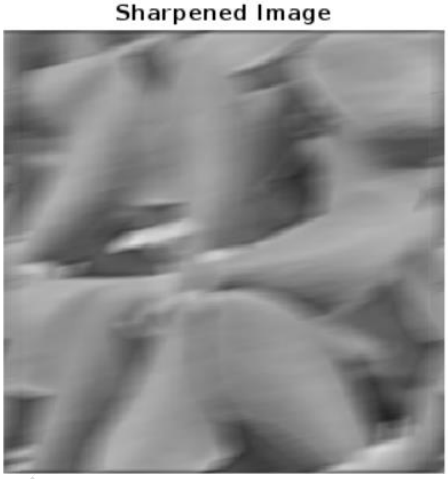


Figure 5: Sharpen Image

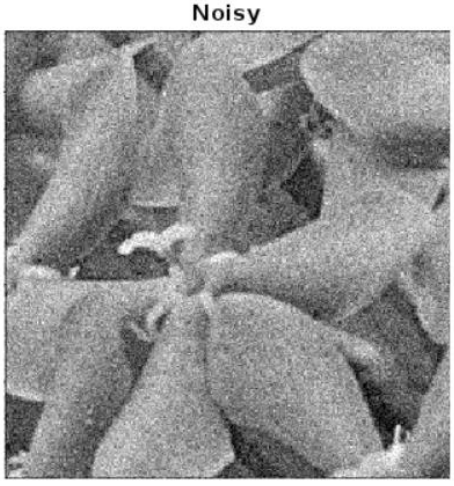


Figure 6: Noisy

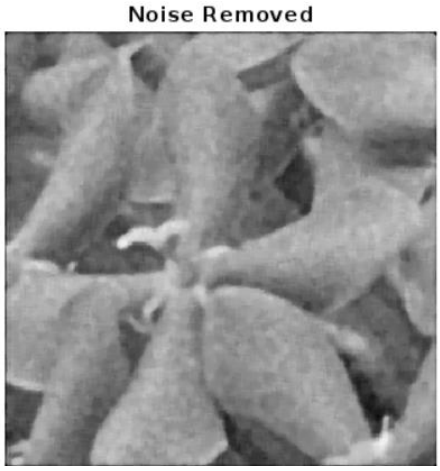


Figure 7: Added Gaussian Noise and Removed Image

Figure 8: Deblurred Image

PYTHON OPENCV OUTPUT:

Figure 1: Acquire an Image of a Flower

Figure 2: Original and Grayscale Image

Figure 3: Motion Blurred Image

Figure 4: Gaussian-filtered Image

Figure 5: Sharpen Image

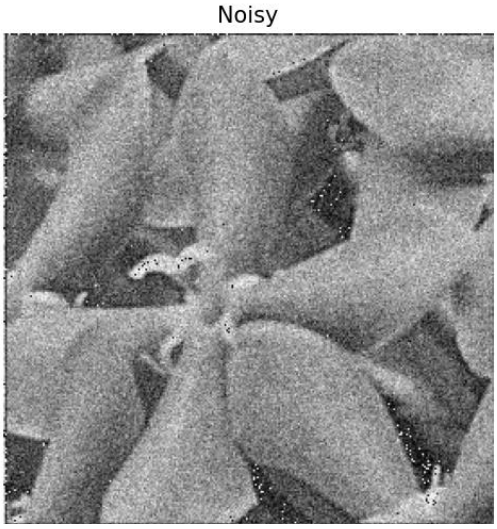


Figure 6: Noisy

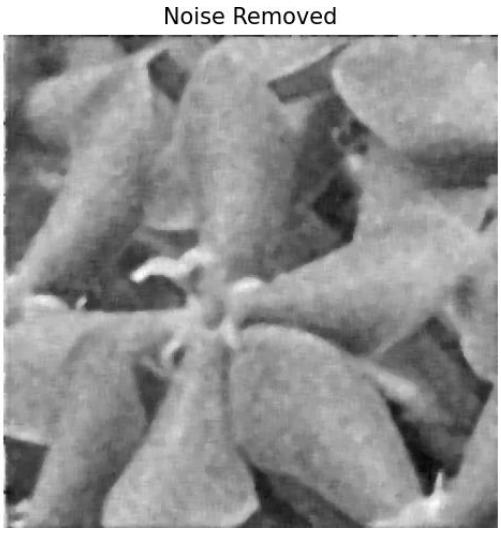


Figure 7: Added Gaussian Noise and Removed Image

Figure 8: Deblurred Image

These codes perform the following:

- Grayscale Conversion: The code first converts the image to grayscale if it's colored (RGB format). This simplifies the image by removing color information, making it easier for subsequent algorithms to process.
- Motion Blur: A motion blur filter is applied, simulating the effect of camera movement during image capture. This can blur sharp edges and details in the original image.
- Gaussian Filtering: A Gaussian filter is used to smooth out the image further. This reduces noise introduced by the motion blur but can also blur sharp details remaining from the original image.
- Sharpening: Unsharp masking is applied to enhance edges in the image. This counteracts the blurring effect but might introduce some artificial sharpening artifacts.
- Noise Addition and Removal: Gaussian noise is artificially added to the grayscale image, simulating imperfections that might occur during image capture. A median filter is then used to remove this noise. Median filters effectively remove impulsive noise but can slightly blur sharp edges.
- Deblurring: Finally, an attempt is made to reverse the motion blur using deconvolution. This process aims to recover the original sharp image, but its effectiveness depends on the accuracy of the estimated blur parameters and the amount of noise present.

Parameter Modification

<You can modify it to explore other functionalities>

```
% Gaussian filtering
h_gaussian = fspecial('gaussian', [5, 5], 10); % Original [5,5], 1
img_gaussian_filtered = imfilter(img_gray, h_gaussian);

% Display the Gaussian filtered
image figure;
imshow(img_gaussian_filtered);
title('Filtered Image with Experimented Value (Gaussian)');

% Histogram (Gaussian
Filtered) figure;
imhist(img_gaussian_filtered)
;
title('Histogram of the Experimented Value (Gaussian Filtered)');


% Add Gaussian noise
img_noisy_exp1 = imnoise(img_gray, 'gaussian', 0.5);
img_noisy_exp2 = imnoise(img_gray, 'gaussian', 0.1);

% Display the noisy
figure;
imshow(img_noisy_exp1)
;
title('Noisy Using Experimented Value (Gaussian is 0.5)');

figure;
imshow(img_noisy_exp2)
;
title('Noisy Using Experimented Value (Gaussian is 0.1)');

% Display the histogram for Noisy
figure;
imhist(img_noisy_exp1);
title('Histogram of Noisy Image Experimented Value 1');

figure;
imhist(img_noisy_exp2)
;
title('Histogram of Noisy Image Experimented Value 2');
```

MATLAB OUTPUT:

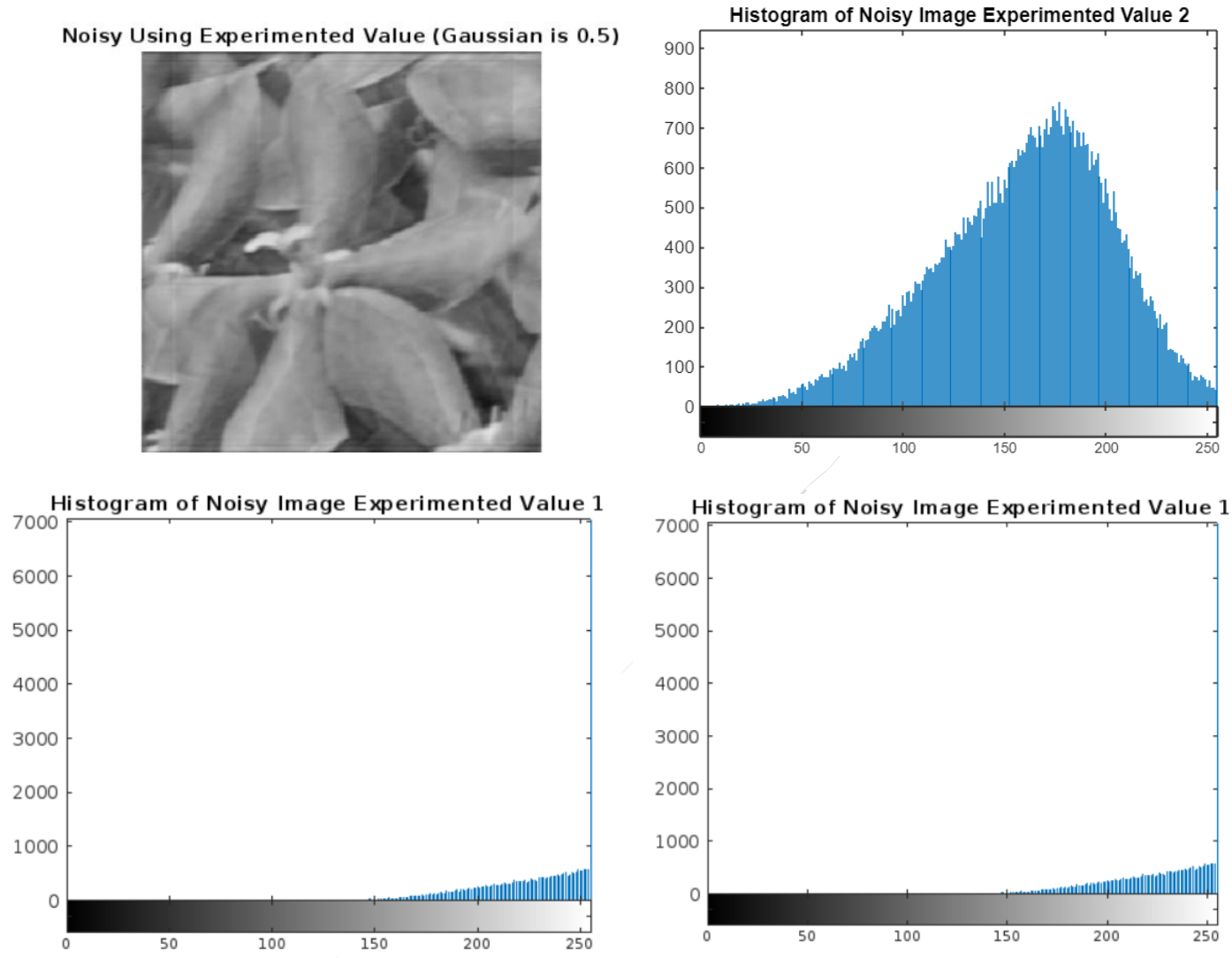
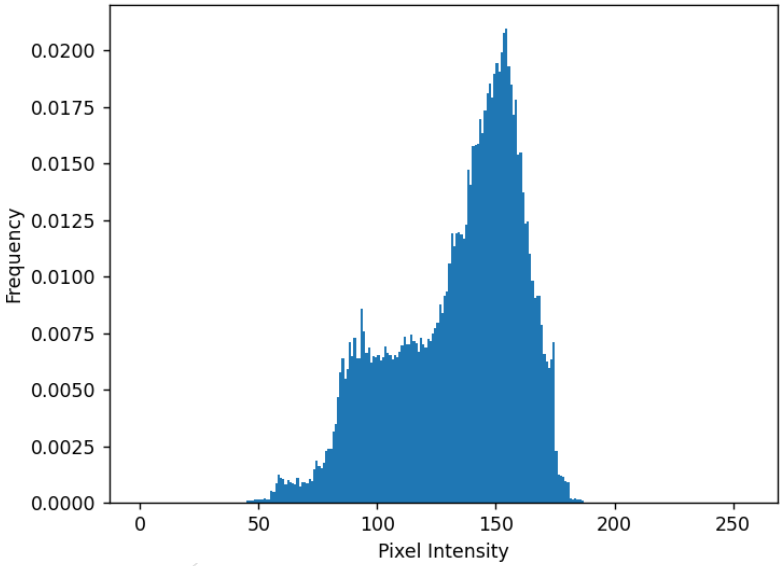


Figure 9: Parameters Modification

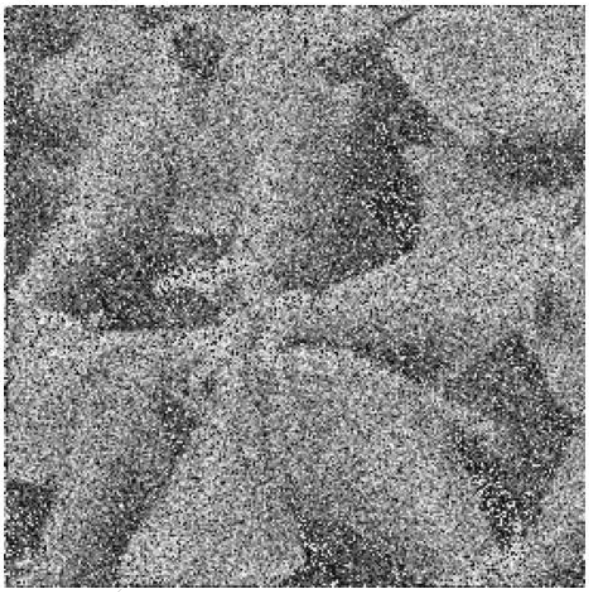
PYTHON OPENCV OUTPUT:

Filtered Image with Experimented Value (Gaussian)

Histogram of the Experimented Value (Gaussian Filtered)



Noisy Using Experimented Value (Gaussian is 0.5)



Noisy Using Experimented Value (Gaussian is 0.1)

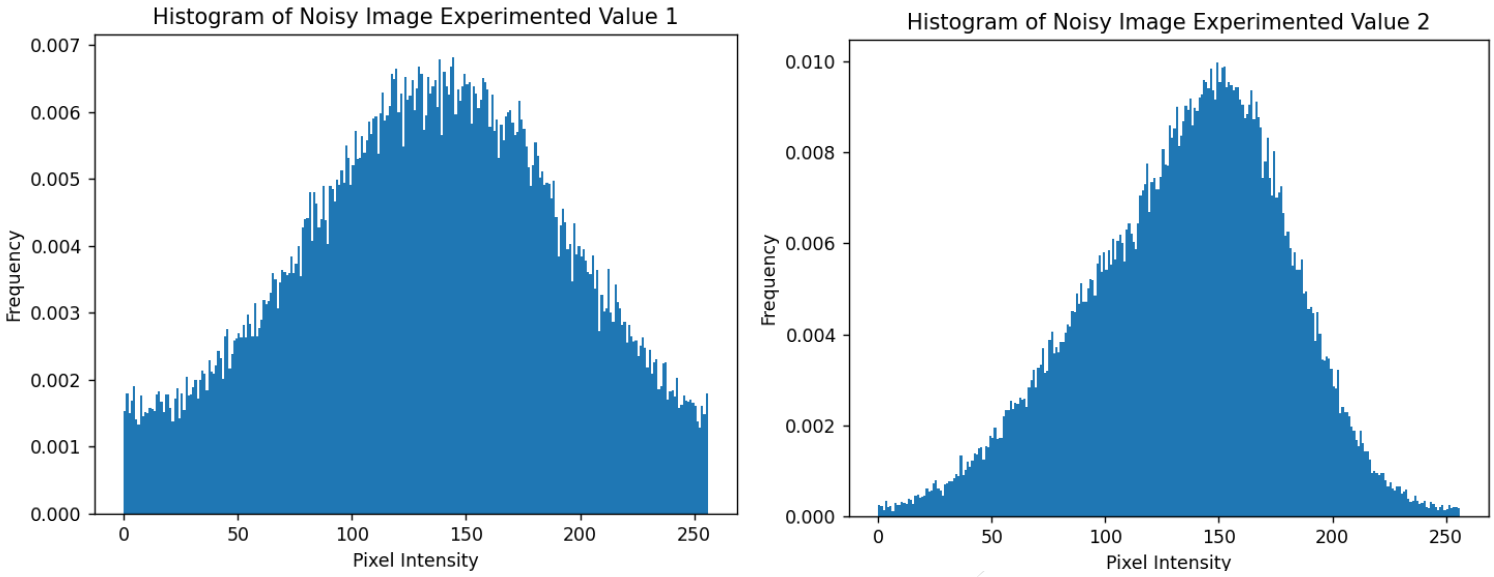


Figure 10: Parameters Modification

CODE PYTHON OPENCV:

```
import cv2
import numpy as np
from scipy.ndimage import gaussian_filter
import matplotlib.pyplot as plt

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

# Convert the image from BGR to RGB (since OpenCV reads in BGR format)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Convert to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

# Gaussian filtering with experimented value
h_gaussian = gaussian_filter(img_gray, sigma=5)

# Display the Gaussian filtered image
plt.figure()
plt.imshow(h_gaussian, cmap='gray')
plt.title('Filtered Image with Experimented Value (Gaussian)')
plt.axis('off')
```

```
# Display the histogram of the Gaussian filtered image
plt.figure()
plt.hist(h_gaussian.ravel(), bins=256, range=(0, 256), density=True)
plt.title('Histogram of the Experimented Value (Gaussian Filtered)')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

# Add Gaussian noise with experimented values
mean = 0
img_noisy_exp1 = img_gray + np.random.normal(mean, 0.2  255, img_gray.shape).astype(np.uint8)
img_noisy_exp2 = img_gray + np.random.normal(mean, 0.1  255, img_gray.shape).astype(np.uint8)

# Clip pixel values to be in the correct range
img_noisy_exp1 = np.clip(img_noisy_exp1, 0, 255)
img_noisy_exp2 = np.clip(img_noisy_exp2, 0, 255)

# Display the noisy images
plt.figure()
plt.imshow(img_noisy_exp1, cmap='gray')
plt.title('Noisy Using Experimented Value (Gaussian is 0.5)')
plt.axis('off')

plt.figure()
plt.imshow(img_noisy_exp2, cmap='gray')
plt.title('Noisy Using Experimented Value (Gaussian is 0.1)')
plt.axis('off')

# Display the histogram for noisy images
plt.figure()
plt.hist(img_noisy_exp1.ravel(), bins=256, range=(0, 256), density=True)
plt.title('Histogram of Noisy Image Experimented Value 1')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.figure()
plt.hist(img_noisy_exp2.ravel(), bins=256, range=(0, 256), density=True)
plt.title('Histogram of Noisy Image Experimented Value 2')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.show()
```

2. Visualize the results, analyze and interpret:

In this laboratory activity, various image restoration techniques were applied, and the resulting images were analyzed to assess the effectiveness of each technique in achieving the desired outcome.

1. Gaussian Filtering:
- Effect: The application of Gaussian filtering effectively smooths the image by reducing noise and minor details, which can be beneficial in pre-processing stages before applying more complex algorithms. The smoothing effect is controlled by the standard deviation parameter of the Gaussian function.
- Effectiveness: This method is effective in reducing noise; however, it also blurs the image, leading to a loss of finer details. The filter is useful when the primary objective is to reduce high-frequency noise.

2. Motion Blur and Deblurring:
- Effect: Motion blur was artificially introduced to the image to simulate the effect of camera movement. The motion blur degrades the image quality by spreading out image details in the direction of the blur.
- Effectiveness: The deblurring process, which attempted to reverse the motion blur using deconvolution, partially restores the image quality. The effectiveness of deblurring depends on the accuracy of the point spread function (PSF) used and the level of noise present. Complete restoration is often challenging, particularly if the blur parameters are not accurately estimated.

3. Sharpening:
- Effect: Unsharp masking was applied to enhance the edges in the blurred image, making the details more pronounced.
- Effectiveness: This technique effectively enhances the image's edges but can also introduce artifacts if over-applied. It's a useful method to counteract the blurring effect, though it must be balanced to avoid over-sharpening.

4. Noise Addition and Removal:
- Effect: Gaussian noise was added to the grayscale image to simulate real-world imperfections during image capture. A median filter was then used to remove this noise.
- Effectiveness: The median filter is effective at removing impulsive noise (e.g., salt-and-pepper noise) without significantly blurring edges, making it suitable for scenarios where preserving edge information is crucial.

5. Parameter Modifications:
- Effect: Experimenting with different parameters, such as increasing the standard deviation in Gaussian filtering, changes the extent of smoothing applied. Similarly, adjusting the noise level alters the image's graininess.
- Effectiveness: Modifying parameters allows for fine-tuning the balance between noise reduction and detail preservation. These experiments help in understanding how different settings impact image quality and guide the selection of optimal values for specific applications.

IV. Conclusion

The laboratory activity successfully demonstrated the application of image restoration techniques using MATLAB and Python. Each technique's effectiveness varied depending on the desired outcome, such as noise reduction or detail enhancement. Gaussian filtering and deblurring were particularly effective in their respective tasks, though they also highlighted the trade-offs between smoothing and preserving image details. The experiments with different parameters provided insight into optimizing these algorithms for various use cases, emphasizing the importance of parameter selection in achieving high-quality image restoration.

References

[1] D.J.D. Sayo. “University of the City of Manila Computer Engineering Department Honor Code,” PLM-CpE Departmental Policies, 2020.

<This is in a separate page>