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

**Elective 3**

Laboratory Activity No. 4

# Image Restoration

Score

*Submitted by:*
*Group 1*
**Antoni, Austine C.**
**Atencia, Dan Eric B.**
**Bauyon, Jared Randalle B.**
**Jao, Steven Rey C.**
**Leonardo, Aibel Renzo T.**
**Saturday, 7AM-4PM / CPE 0332.1-1**

*Date Submitted*
**08-09-2024**

*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');
```

```matlab
% 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.

```python
import cv2
import numpy as np

#Original
img = cv2.imread('flower.jpg')

# Convert to grayscale if the image is BGR
if img.shape[2] == 3:
        grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
else:
        grayImg = img

#Motion Blur
def motionBlurKernel(size):
    kernel = np.zeros((size, size))
    kernel[int((size - 1) / 2), :] = np.ones(size)
    kernel = kernel / size
    return kernel
motionBlurredImg = cv2.filter2D(grayImg, -2, motionBlurKernel(15))

#Gaussian Blur
gaussianImg = cv2.GaussianBlur(grayImg, (15, 15), 0)

#Sharpen
sharpen = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
sharpenedImg = cv2.filter2D(grayImg, -1, sharpen)

#Noisy
noise = np.random.normal(0, 1, grayImg.shape).astype('uint8')
noisyImg = cv2.add(grayImg, noise)

#Denoise
denoisedImg = cv2.medianBlur(noisyImg, 5)

cv2.imshow('Original Image', img)
cv2.imshow('Grayscale', grayImg)
cv2.imshow('Motion Blurred Image', motionBlurredImg)
cv2.imshow('Filtered Image (Gaussian)', gaussianImg)
cv2.imshow('Sharpened Image', sharpenedImg)
cv2.imshow('Noisy', noisyImg)
```

```
cv2.imshow('Noise Removed', denoisedImg)

#Deblurred
motionBlurredImg = cv2.filter2D(grayImg, -2, motionBlurKernel(8))
cv2.imshow('Deblurred Image', motionBlurredImg)

cv2.waitKey(0)
```

III. Results

**MATLAB:**

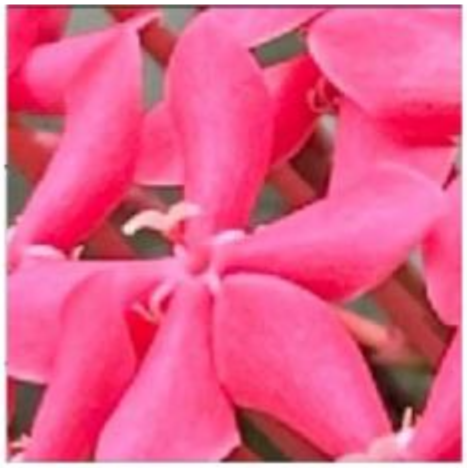

Figure 1: Acquire an Image of a Flower

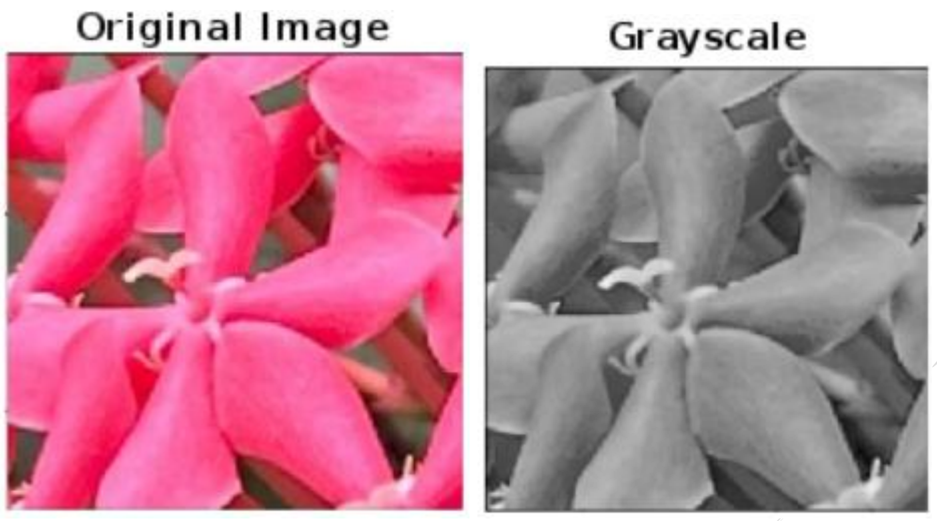

Figure 2: Original and Grayscale Image

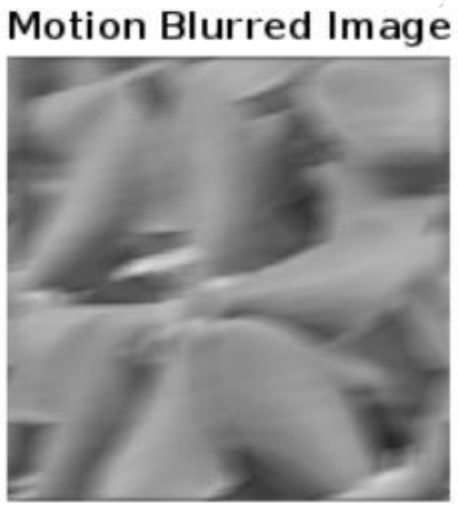

Figure 3: Motion Blurred Image

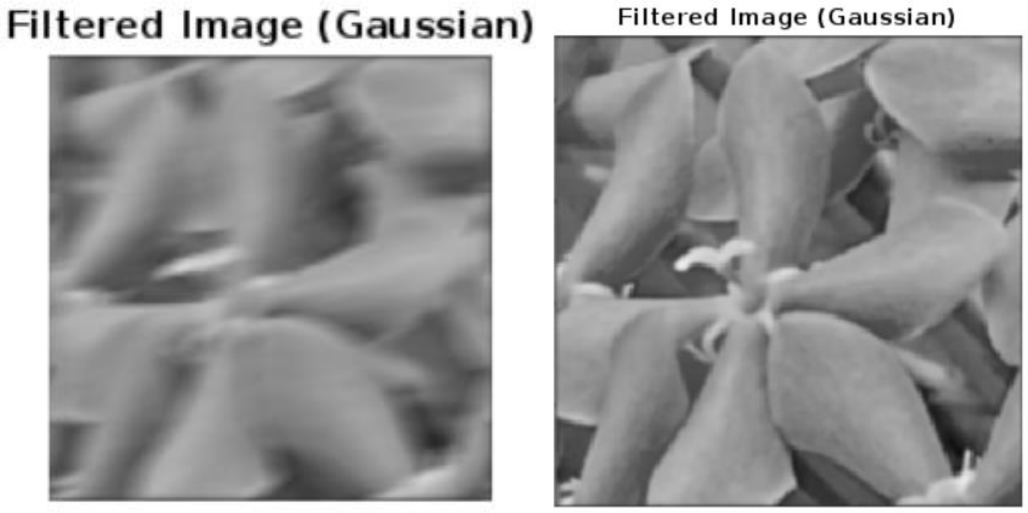

Figure 4: Gaussian-filtered Image using img_blur (based on provided code) and img_gray

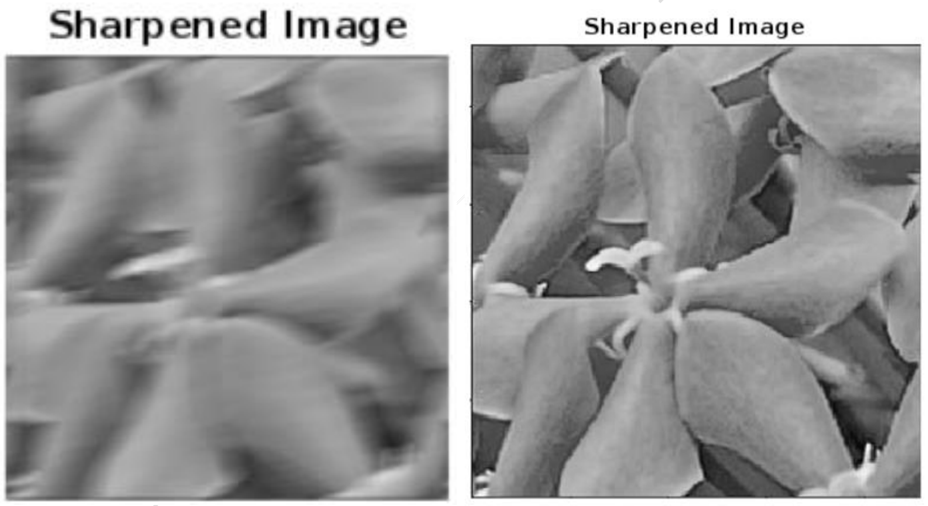

Figure 5: Sharpen Image using img_blur (based on provided code) and img_gray

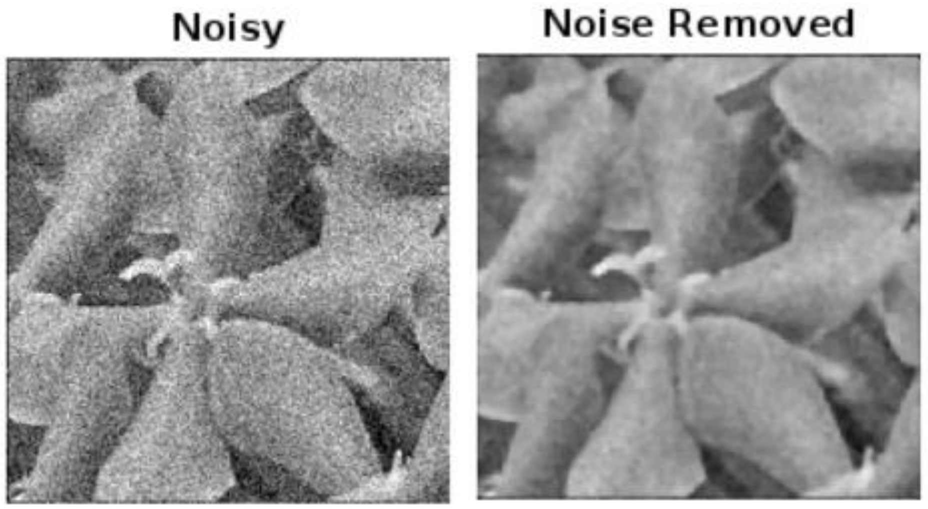

Figure 6: Added Gaussian Noise and Removed Image

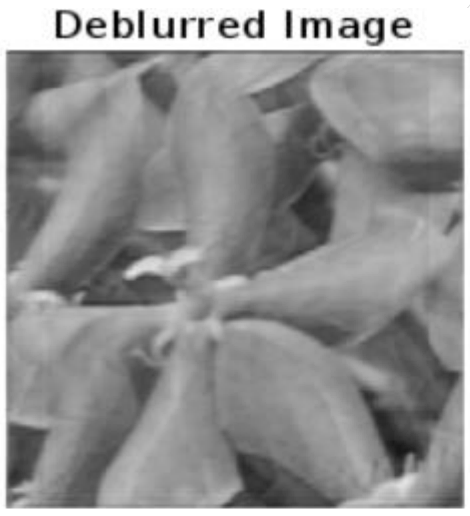

Figure 7: Deblurred Image

**OpenCV**:

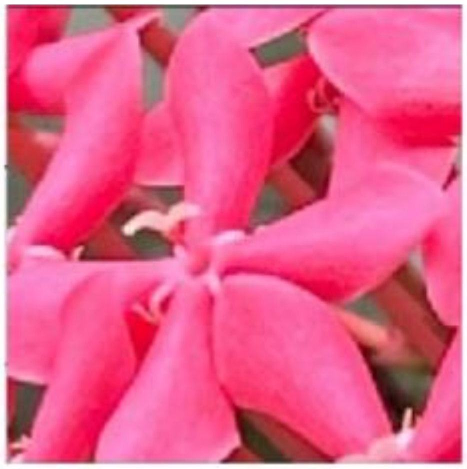

Figure 1: Acquire an Image of a Flower

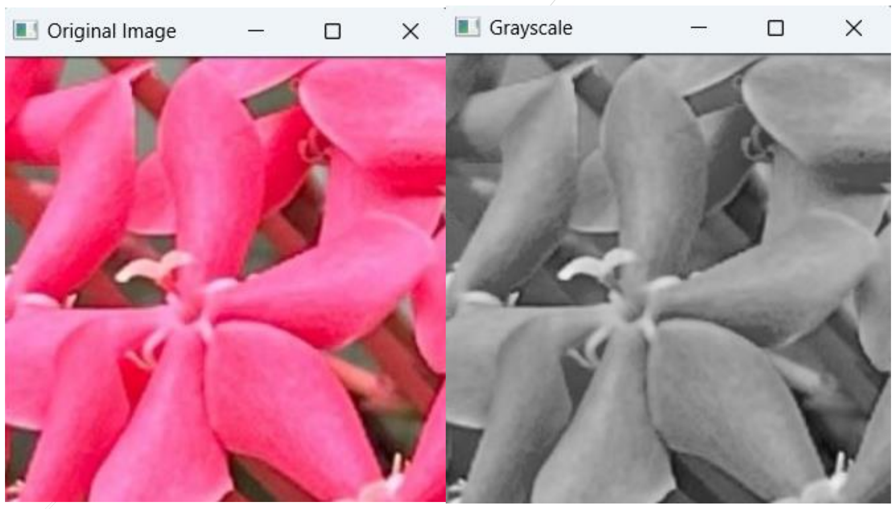

Figure 2: Original and Grayscale Image

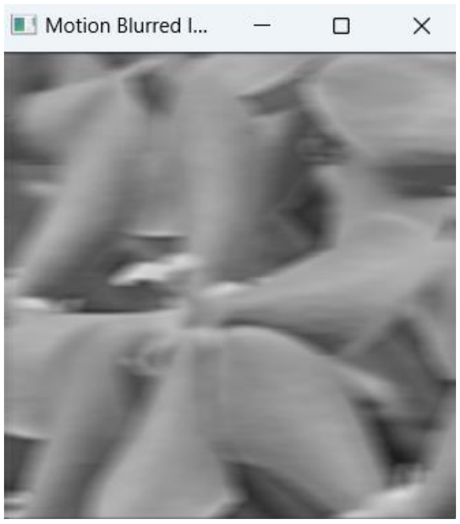

Figure 3: Motion Blurred Image

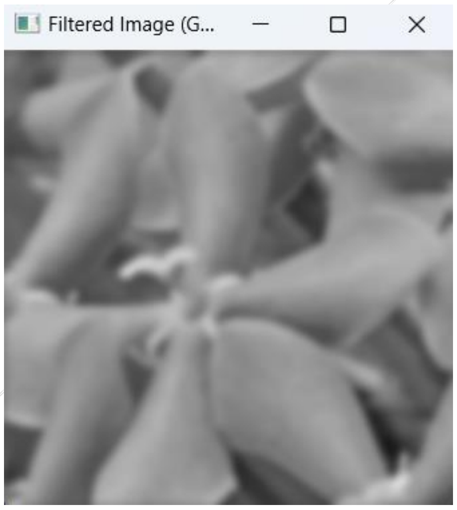

Figure 4: Gaussian-filtered Image

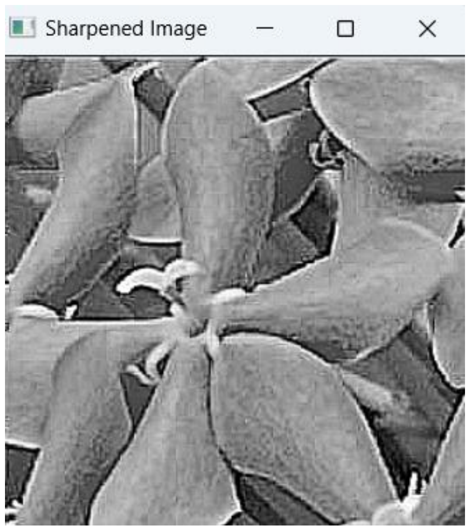

Figure 5: Sharpen Image

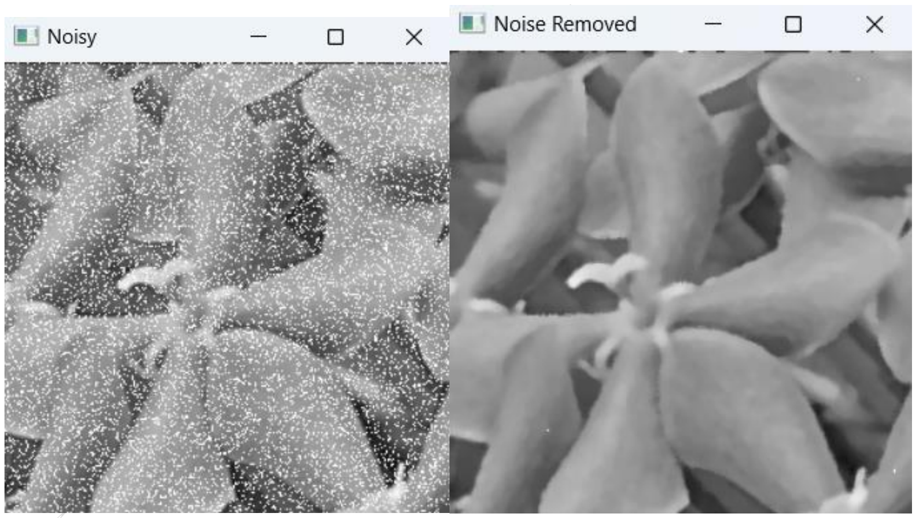

Figure 6: Added Gaussian Noise and Removed Image

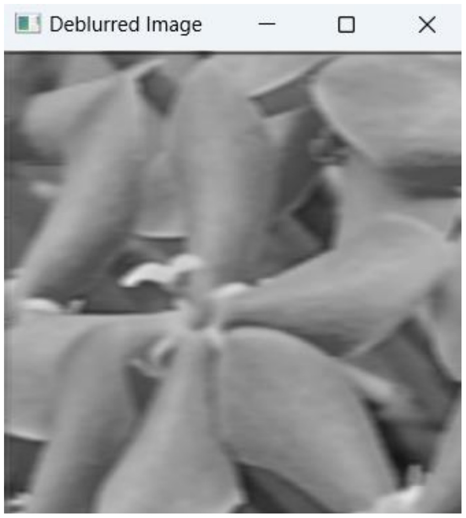

Figure 7: Deblurred Image

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

MATLAB

```
% 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');
```

OpenCV

```
import cv2
import numpy as np

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

# Convert to grayscale if the image is BGR
if img.shape[2] == 3:
    grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
else:
    grayImg = img

# Function to create histogram with white background, blue bars, border, and axis
labels
def create_histogram(hist):
    # Define the size of the histogram image
    histImageWidth = 600
    histImageHeight = 500
    histImage = np.ones((histImageHeight, histImageWidth, 3), dtype=np.uint8) *
255

    # Normalize the histogram
    max_val = np.max(hist)
    cv2.normalize(hist, hist, 0, histImageHeight - 40, cv2.NORM_MINMAX)

    # Draw the histogram with blue bars
    for i in range(256):
        height = int(hist[i])
        cv2.line(histImage, (i * 2 + 50, histImageHeight - 30),
                 (i * 2 + 50, histImageHeight - 30 - height), (255, 0, 0), 2)

    # Draw border
    cv2.rectangle(histImage, (0, 0), (histImageWidth - 1, histImageHeight - 1),
(0, 0, 0), 2)

    # Draw x-axis line
    cv2.line(histImage, (50, histImageHeight - 30), (histImageWidth - 10,
histImageHeight - 30), (0, 0, 0), 2)

    # Draw y-axis line
    cv2.line(histImage, (50, histImageHeight - 30), (50, 10), (0, 0, 0), 2)
```

```python
    # Add x-axis values and lines
    for i in range(0, 256, 50):  # x-axis ticks every 50 pixels
        x = i * 2 + 50
        if x < histImageWidth - 30:  # Ensure label fits within image width
            cv2.putText(histImage, str(i), (x, histImageHeight - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
            # Draw vertical lines for x-axis ticks
            cv2.line(histImage, (x, histImageHeight - 30),
                     (x, histImageHeight - 35), (0, 0, 0), 1)

    # Add y-axis values and lines
    for i in range(0, 1200, 200):  # y-axis ticks every 200 frequency values
        y = histImageHeight - 30 - int(i * (histImageHeight - 40) / 1000)
        if y > 20:  # Ensure label fits within image height
            cv2.putText(histImage, str(i), (10, y + 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
            # Draw horizontal lines for y-axis ticks
            cv2.line(histImage, (45, y),
                     (50, y), (0, 0, 0), 1)

    return histImage

# Gaussian Blur
# Define Gaussian kernel
kernel_size = (5, 5)
sigma = 10
h_gaussian = cv2.getGaussianKernel(kernel_size[0], sigma)
h_gaussian = h_gaussian * h_gaussian.T  # Convert to 2D Gaussian kernel

# Apply Gaussian filter
gaussianImg = cv2.filter2D(grayImg, -1, h_gaussian)

# Histogram of Gaussian Blur
gaussianHist = cv2.calcHist([gaussianImg], [0], None, [256], [0, 256])
gaussianHistImg = create_histogram(gaussianHist)

# Noisy Experiment 1
VarNoise1 = np.sqrt(0.5)
noise1 = np.random.normal(0, varNoise2, grayImg.shape).astype('uint8')
noisyImg1 = cv2.add(grayImg, noise1)

# Histogram of Noisy Experiment 1
noisy1Hist = cv2.calcHist([noisyImg1], [0], None, [256], [0, 256])
```

```python
noisy1HistImg = create_histogram(noisy1Hist)

# Noisy Experiment 2
varNoise2 = np.sqrt(0.1)
noise2 = np.random.normal(0, varNoise2, grayImg.shape).astype('uint8')
noisyImg2 = cv2.add(grayImg, noise2)

# Histogram of Noisy Experiment 2
noisy2Hist = cv2.calcHist([noisyImg2], [0], None, [256], [0, 256])
noisy2HistImg = create_histogram(noisy2Hist)

# Display images and histograms
cv2.imshow('Original Image', img)
cv2.imshow('Grayscale', grayImg)
cv2.imshow('Filtered Image with Experimented Value (Gaussian)', gaussianImg)
cv2.imshow('Histogram of the Experimented Value (Gaussian Filtered)',
gaussianHistImg)
cv2.imshow('Noisy Using Experimented Value (Gaussian is 0.5)', noisyImg1)
cv2.imshow('Noisy Using Experimented Value (Gaussian is 0.1)', noisyImg2)
cv2.imshow('Histogram of Noisy Image Experimented Value 1', noisy1HistImg)
cv2.imshow('Histogram of Noisy Image Experimented Value 2', noisy2HistImg)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

MATLAB:

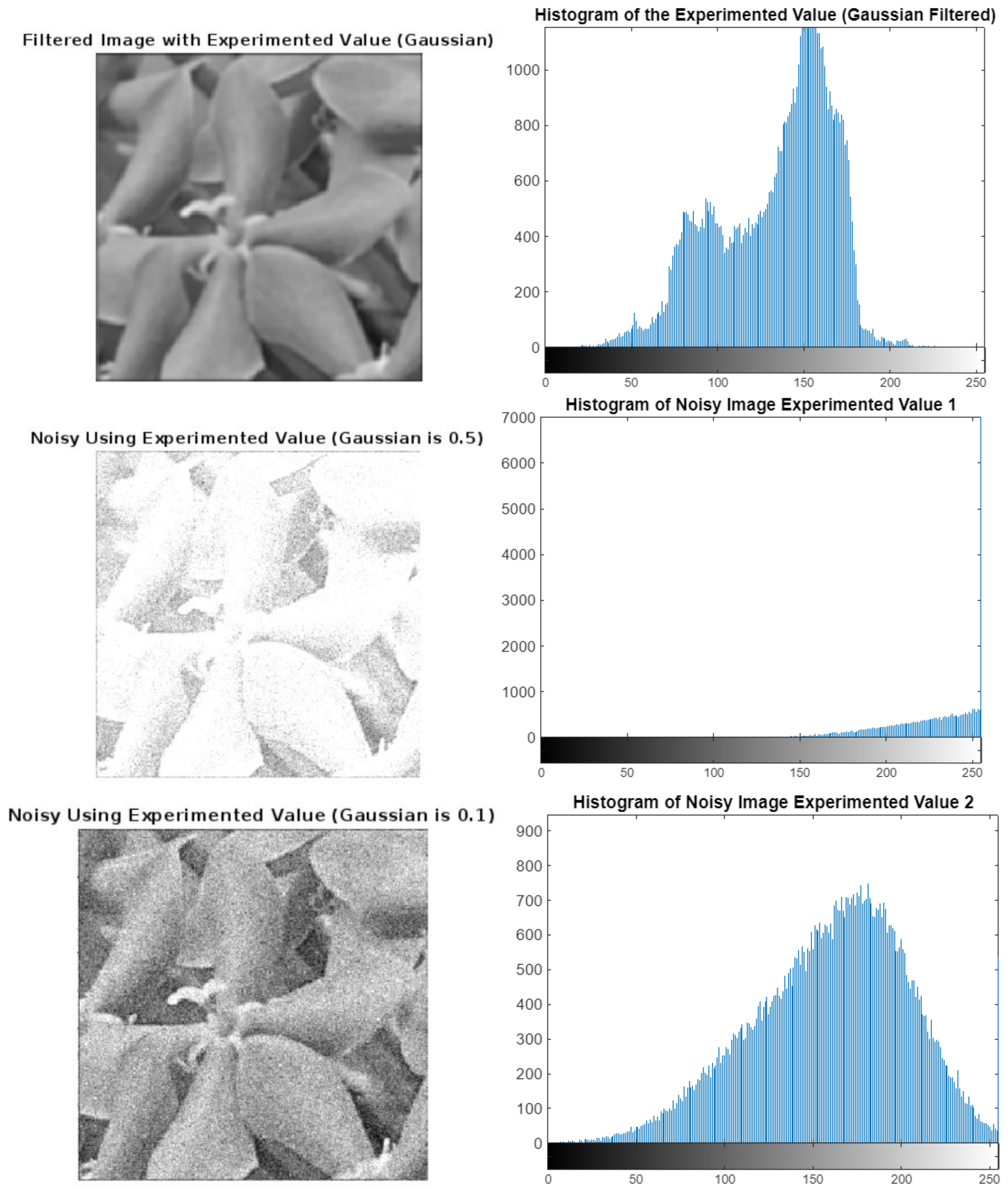

Figure 9: Parameters Modification

OpenCV:

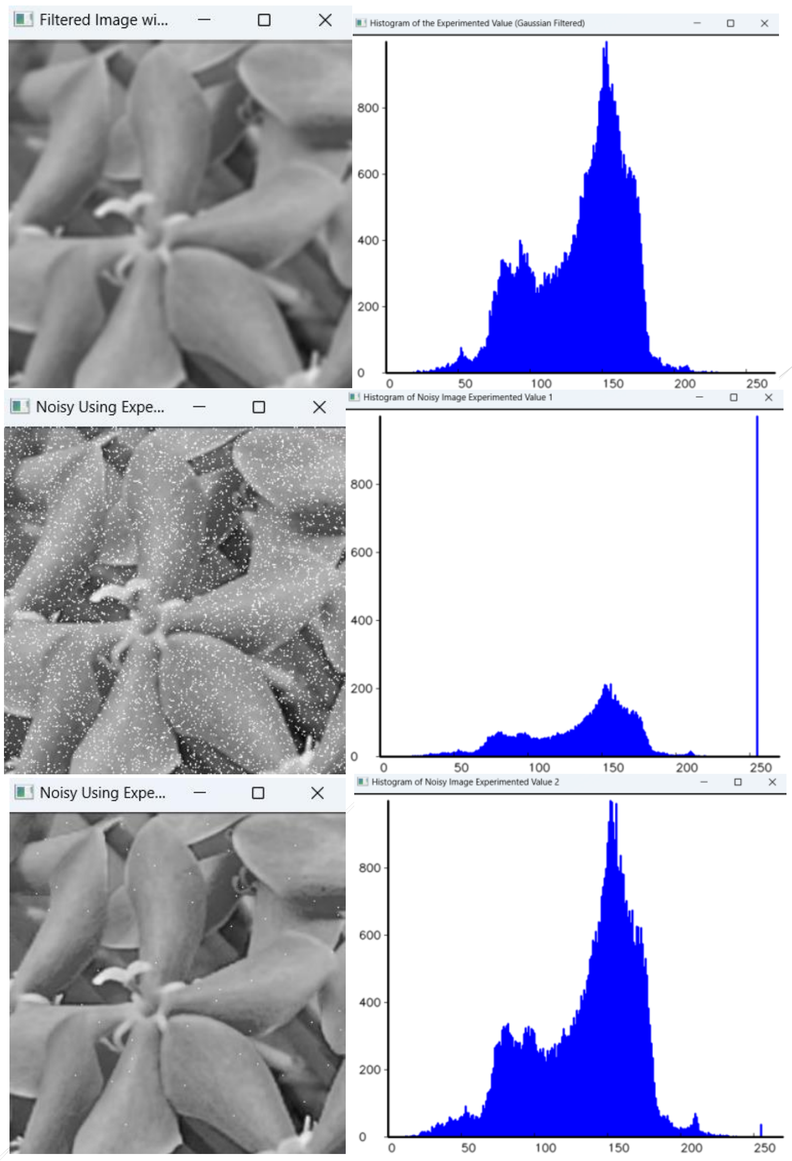
Filtered Image wi...
Histogram of the Experimented Value (Gaussian Filtered)
Noisy Using Expe...
Histogram of Noisy Image Experimented Value 1
Noisy Using Expe...
Histogram of Noisy Image Experimented Value 2
0
200
400
600
800
0
50
100
150
200
250

Figure 9: Parameters Modification

1. Visualize the results, analyze and interpret:

**A. MATLAB**

This activity shows how a picture can be manipulated step by step and undergo numerous changes. It begins by reading an image file 'flower.jpg' and presenting it in a figure titled 'Original Image.' If the image is in RGB format (has 3 channels), it is converted to grayscale and displayed in another figure. A motion blur effect is applied to the grayscale image using fspecial's point spread function (PSF) with a defined length and angle, and the resulting blurred image is presented. The blurred image is then filtered with a Gaussian filter, and the output is presented. The code sharpens the blurred image and displays it in another figure. Next, Gaussian noise is applied to the grayscale image, which is then presented. The noise is eliminated with a median filter, and the resulting image is displayed. Finally, the blurred image is deblurred using the Wiener deconvolution method with a given noise-to-signal ratio, and the output image is presented.

**B. MATLAB Modification Parameter**

This program manipulates an image and illustrates several transformations and histograms. It begins by reading the image file 'flower.jpg' and showing it as the 'Original Image.' The code determines whether the image is colored by checking if it has three channels, and if it is, it converts it to grayscale. The grayscale image is shown next. The grayscale image is filtered with a Gaussian filter of a chosen size and standard deviation (experimented value of 10) before being shown. The histogram of the filtered image is then shown in a separate figure. The code adds Gaussian noise to the grayscale image at two different experimental values (0.5 and 0.1), and then displays the noisy images separately. Finally, it displays the histograms of these noisy images in separate figures, allowing for a visual comparison of noise levels and their impact on image distribution.

## C. OpenCV

This Python program imports the OpenCV library to manipulate an image and display the results. It begins by loading the required libraries, including cv2 for image processing and numpy for numerical computations. The original image 'flower.jpg' is read and stored in the variable img. The image is subsequently converted to grayscale and saved as grayImg. The function motionBlurKernel generates a horizontal motion blur kernel, which is then applied to the grayscale image to create the motion-blurred image motionBlurredImg. A 15x15 Gaussian blur is applied to the grayscale image, resulting to gaussianImg. A sharpening filter is used and applied to the grayscale image to obtain a sharpened image (sharpenedImg). Random Gaussian noise is applied to the grayscale image to produce the noisy image noisyImg. The noisy image is then denoised with a median blur, resulting to denoisedImg. All of these photos, including the original, grayscale, motion-blurred, Gaussian-blurred, sharpened, noisy, and denoised versions, are displayed using OpenCV's imshow function. Finally, the grayscale image is motion-blurred using a variable kernel size and shown as the deblurred image. The waitKey(0) function call keeps the shown images open until a key is pushed.

## D. Python Parameter Modification

This program manipulates a picture with OpenCV and NumPy to do various image changes and displays the results, including histograms. It begins by importing OpenCV (cv2) and NumPy (np). The original image, 'flower.jpg', is read and stored to img. The image is converted to grayscale and saved as grayImg. gaussianImg is created by applying a Gaussian blur with a kernel size of 15x15 to a grayscale image. The histogram of this blurred image is produced and normalized before being drawn onto GaussianHistImg with lines to show the distribution of pixel intensities. The grayscale picture is then filled with Gaussian noise with a standard deviation of sqrt(0.5), resulting in noisyImg1. The histogram for this noisy image is computed, normalized, and drawn on noisy1HistImg. Similarly, noisyImg2 is created by adding Gaussian noise with a standard deviation of sqrt(0.1) to the grayscale image, after which its histogram is generated, normalized, and drawn onto noisy2HistImg. The original image, grayscale image, Gaussian blurred image, and all noisy images, together with their histograms, are displayed in separate windows via OpenCV's imshow function. The waitKey(0) function keeps the window open until a key is pressed.

## IV. Conclusion

In this activity, we have learned how to turn an image into grayscale if it is RGB, add blur to that grayscale image; Gaussian filter, sharpen, and deblur the blurred image; and add Gaussian noise to an image and then remove it using both MATLAB and Python. We have also analyzed the effects of these additional image manipulations using histogram presentations. MATLAB uses commands like rgb2gray, fspecial('motion') then imfilter, fspecial('gaussian') then imfilter, imsharpen, imnoise, medfilt2, and deconvwn to perform each image manipulation. Python on the other hand uses commands like cv2.cvtColor, motionBlurKernel then cv2.filter2D, cv2.GaussianBlur, np.array then cv2.filter2D, np.random.normal then cv2.add, and cv2.medianBlur to perform the same image manipulations as in MATLAB. Both languages were able to demonstrate how their respective commands can manipulate images by grayscaling, motion blurring, Gaussian filtering, sharpening, adding and removing noise, and deblurring accompanied by histograms that present each manipulations' effect on the image.

## References

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