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

Elective 3

Laboratory Activity No. 4
Image Restoration

Score

Submitted by:

Bolocon, Joniel R.

De Guzman, Jastine V.

De Juan, Lord Welchie P.

Mercado, Ed-Vir G.

Puno, Harold Dennis R.

Saturday 7:00 AM – 4:00 PM / CPE 0332.1-1

Date Submitted

08-08-2024

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

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

# Read the image
img = cv2.imread('flower.jpg')  # Replace with the path to your image file
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display the original image
plt.figure()
plt.imshow(img_rgb)
plt.title('Original Image')
plt.axis('off')

# Convert to grayscale if the image is RGB
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
plt.figure()
plt.imshow(img_gray, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')

# Add motion blur to the image
len = 21
psf = np.zeros((len, len))
center = len // 2
for i in range(len):
    psf[i, center] = 1
psf = psf / psf.sum()
img_blur = ndimage.convolve(img_gray.astype(float), psf, mode='wrap')

# Display the blurred image
plt.figure()
plt.imshow(img_blur, cmap='gray')
plt.title('Motion Blurred Image')
plt.axis('off')

# Gaussian filtering
h_gaussian = cv2.getGaussianKernel(5, 1)
h_gaussian = h_gaussian @ h_gaussian.T
img_gaussian_filtered = cv2.filter2D(img_blur, -1, h_gaussian)

# Display the Gaussian filtered image
plt.figure()
plt.imshow(img_gaussian_filtered, cmap='gray')
plt.title('Filtered Image (Gaussian)')
plt.axis('off')

# Edge Detection
img_edges = cv2.Canny(img_gray, 25, 75)
```

```
# Display the edge-detected image
plt.figure()
plt.imshow(img_edges, cmap='gray')
plt.title('Edge Detected Image')
plt.axis('off')

# Sharpening using unsharp masking
blurred = cv2.GaussianBlur(img_blur, (5, 5), 1.0)
img_sharpened = cv2.addWeighted(img_blur, 1.5, blurred, -0.5, 0)

# Display the sharpened image
plt.figure()
plt.imshow(img_sharpened, cmap='gray')
plt.title('Sharpened Image')
plt.axis('off')

# Add Gaussian noise and remove it using median filter
img_noisy = np.clip(img_gray + np.random.normal(0, 20, img_gray.shape), 0,
255).astype(np.uint8)
img_noisy_removed = cv2.medianBlur(img_noisy, 5)

# Display the noisy image
plt.figure()
plt.imshow(img_noisy, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')

# Display the noise-removed image
plt.figure()
plt.imshow(img_noisy_removed, cmap='gray')
plt.title('Noise Removed Image')
plt.axis('off')

# Deblurring
def wiener_deconvolution(img, kernel, noise_var):
# Compute the Fourier transform of the image and kernel
img_fft = np.fft.fft2(img)
kernel_fft = np.fft.fft2(kernel, s=img.shape)

# Compute Wiener filter
kernel_fft_conj = np.conj(kernel_fft)
wiener_filter = kernel_fft_conj / (np.abs(kernel_fft) ** 2 + noise_var)

# Apply Wiener filter in the frequency domain
deblurred_fft = img_fft * wiener_filter

# Convert back to spatial domain
deblurred_img = np.abs(np.fft.ifft2(deblurred_fft))

return np.uint8(np.clip(deblurred_img, 0, 255))

# Define the kernel (PSF) for deblurring
kernel_size = 21
kernel = np.zeros((kernel_size, kernel_size))
center = kernel_size // 2
```

```
for i in range(kernel_size):
   kernel[i, center] = 1
kernel = kernel / kernel.sum()  # Normalize kernel

# Set noise variance for Wiener filter
noise_var = 0.01

# Apply Wiener deconvolution
deblurred_img = wiener_deconvolution(img_blur, kernel, noise_var)

def remove_boundaries(image, padding_size):
return image[padding_size:-padding_size, padding_size:-padding_size]

padding_size = kernel_size // 2
deblurred_img = remove_boundaries(deblurred_img, padding_size)

# Display the deblurred image
plt.figure()
plt.imshow(deblurred_img, 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

Figure 1: Acquire an Image of a Flower (MATLAB)

Figure 1.1: Acquire an Image of a Flower (PYTHON)

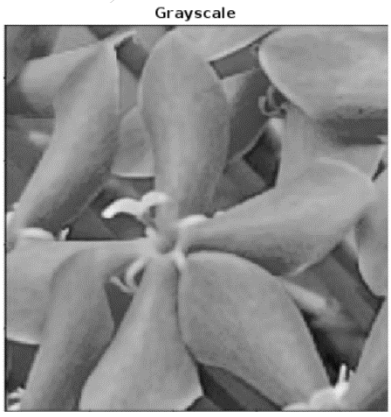


Figure 2: Original and Grayscale Image (MATLAB)

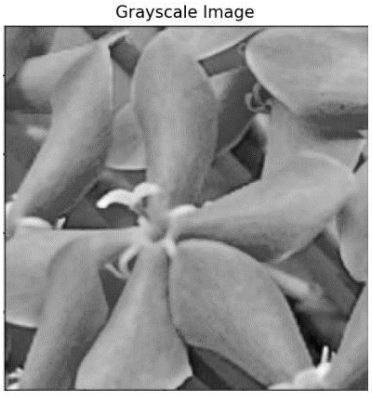


Figure 2.1: Original and Grayscale Image (PYTHON)

Figure 3: Motion Blurred Image (MATLAB)

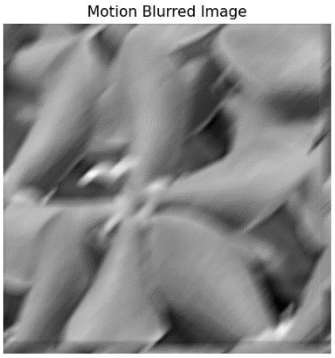


Figure 3.1: Motion Blurred Image (PYTHON)

Figure 4: Gaussian-filtered Image (MATLAB)

Figure 4.1: Gaussian-filtered Image (PYTHON)

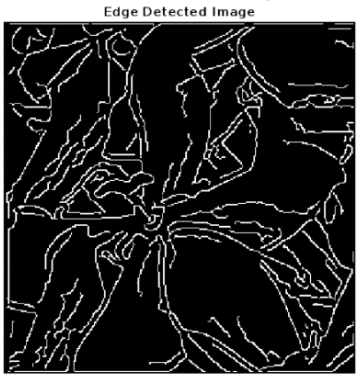


Figure 5: Edge-detected Image (MATLAB)

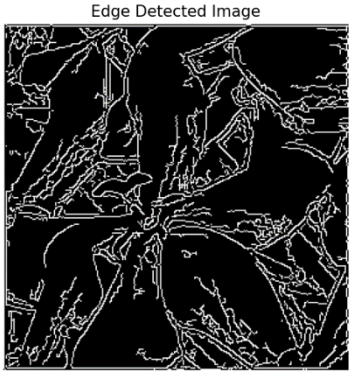


Figure 5.1: Edge-detected Image (PYTHON)

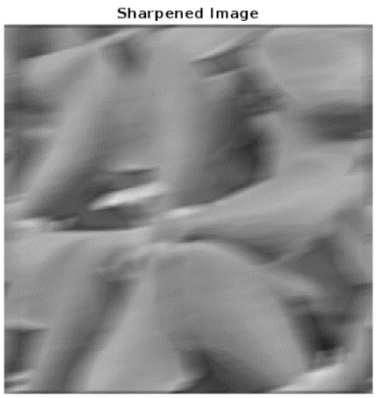
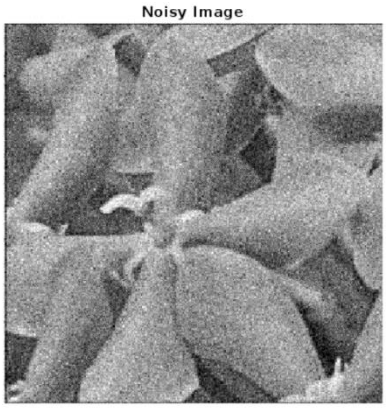


Figure 6: Sharpen Image (MATLAB)

Figure 6.1: Sharpen Image (PYTHON)



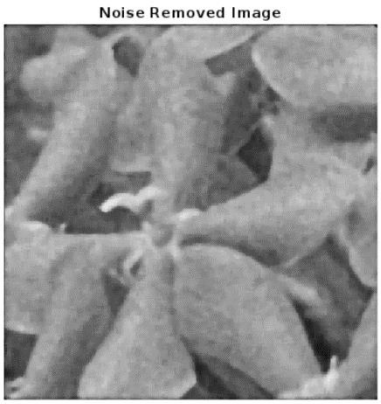
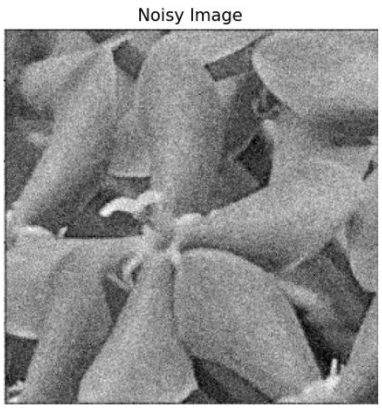


Figure 7: Added Gaussian Noise and Removed Image (MATLAB)



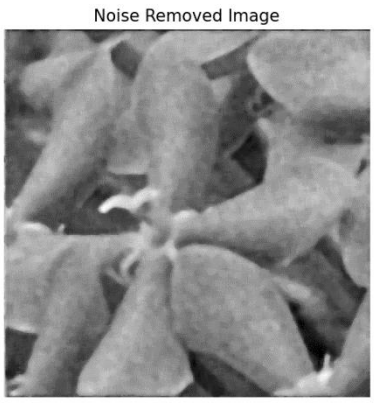


Figure 7.1: Added Gaussian Noise and Removed Image (PYTHON)

Figure 8: Deblurred Image (MATLAB)

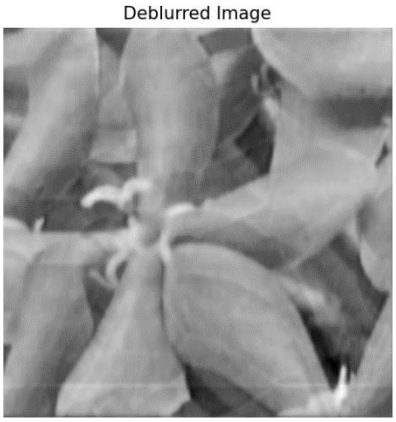


Figure 8.1: Deblurred Image (PYTHON)

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');
```

PYTHON

```
# Gaussian filtering
h_gaussian = cv2.getGaussianKernel(5, 10)
h_gaussian = h_gaussian @ h_gaussian.T
img_gaussian_filtered = cv2.filter2D(img_blur, -1, h_gaussian)

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

# Histogram (Gaussian Filtered)
plt.figure()
plt.hist(img_gaussian_filtered.ravel(), bins=256, range=[0,256])
plt.title('Histogram of the Experimented Value (Gaussian Filtered)')
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True)  # Add axis borders
plt.show(block=False)

# Add Gaussian noise (experimented values)
mean1, stddev1 = 0, 0.5
mean2, stddev2 = 0, 0.1
img_noisy_exp1 = np.clip(img_gray + np.random.normal(mean1, stddev1 * 100,
img_gray.shape), 0, 255).astype(np.uint8)
img_noisy_exp2 = np.clip(img_gray + np.random.normal(mean2, stddev2 * 100,
img_gray.shape), 0, 255).astype(np.uint8)

# 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])
plt.title('Histogram of Noisy Image Experimented Value 1')
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True)
plt.show(block=False)

plt.figure()
plt.hist(img_noisy_exp2.ravel(), bins=256, range=[0,256])
plt.title('Histogram of Noisy Image Experimented Value 2')
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True)
plt.show(block=False)
```

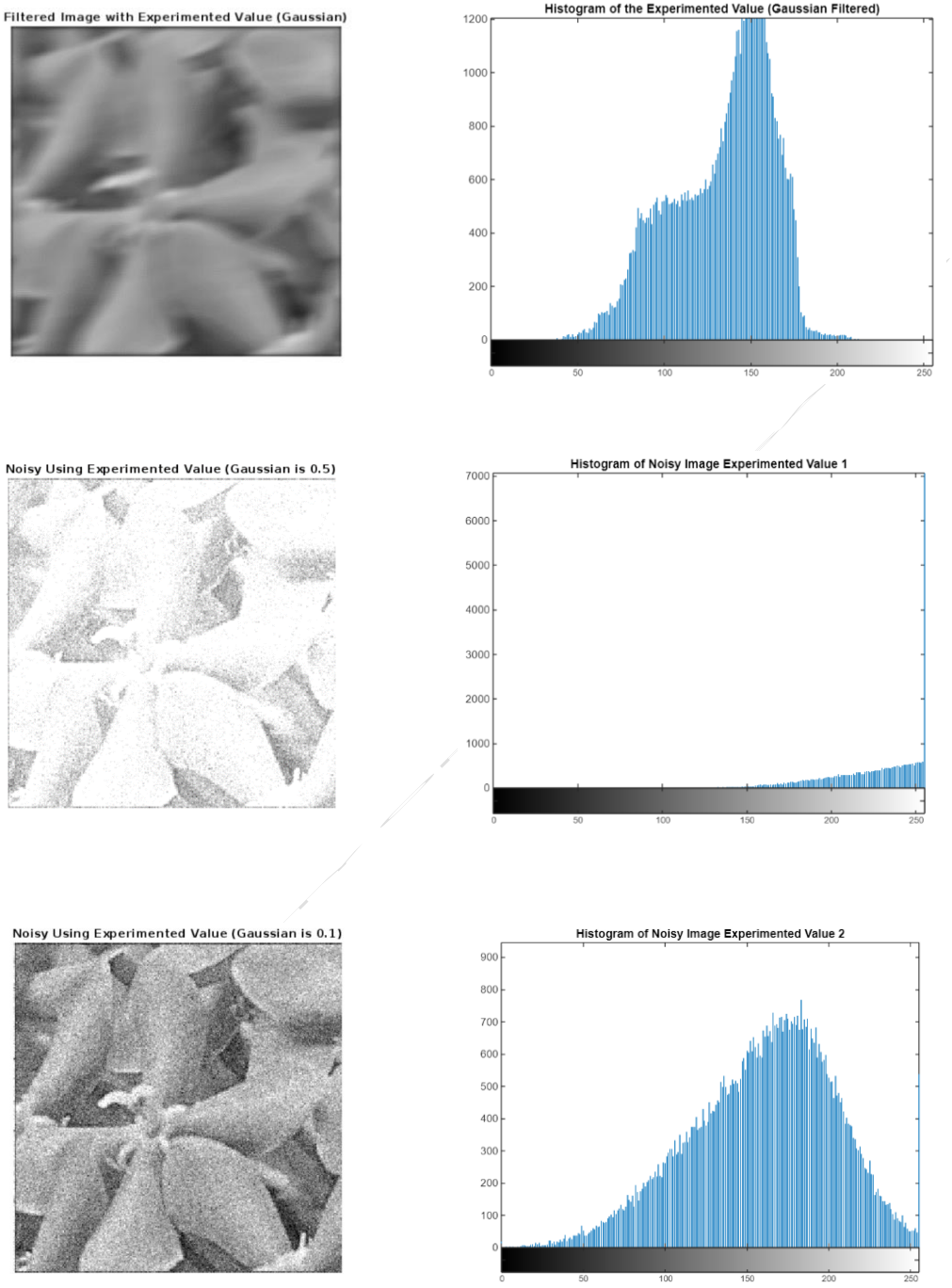


Figure 9: Parameters Modification (MATLAB)

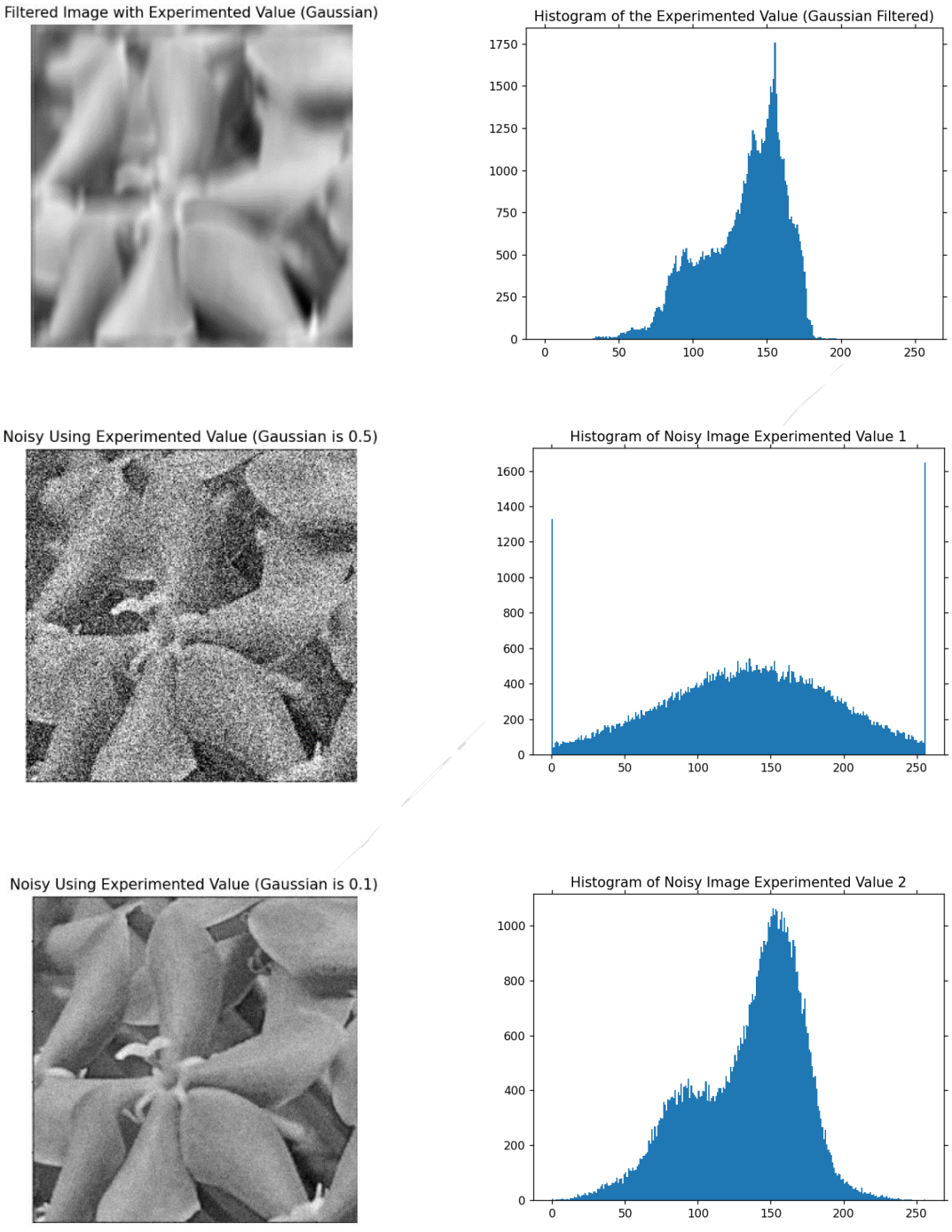


Figure 9: Parameters Modification (PYTHON)

2. Visualize the results, analyze and interpret:

The application of various image processing algorithms on the image resulted in distinct transformations that improved the overall quality and characteristics of the image. Converting the image to grayscale simplified processing by removing color information. Applying motion blur simulated camera movement, effectively blurring the image. The Gaussian filter smoothed the image, reducing noise but also slightly blurring details. Unsharp masking sharpened edges, enhancing clarity while introducing some artificial artifacts. Adding Gaussian noise simulated real-world imperfections, and the median filter effectively removed this noise, though with minor blurring of edges. Finally, deconvolution attempted to reverse the motion blur, with effectiveness dependent on accurate blur parameters and noise levels.

IV. Conclusion

In conclusion, after successfully completing this laboratory exercise, we have applied various image processing techniques, including image acquisition and conversion from RGB color to grayscale, like the previous laboratory exercises. While earlier activities focused on contrast enhancement, filtering in spatial (i.e., average and median) and image blurring, this exercise includes additional image processing techniques such as displaying histograms, motion blurring, Gaussian filtering, edge detection, image sharpening, adding and removing Gaussian noise, and lastly, deblurring images. Moreover, this activity utilized the same tools as the previous laboratory exercise, such as MATLAB and Python, and both displayed the image appropriately, with slight differences in their output, such as image clarity and color accuracy.

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>