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

Elective 3

Laboratory Activity No. 4
Image Restoration

Score

Submitted by:
Alambra, Joseph Nathaniel A.
Aragon, Patrick Laurence M.
Banal, Daryll L.
Sentasas, David Bryan L.
Tutanes, Allen Christopher O.
SAT 7:00AM – 4:00PM / CPE 0332.1-1

Date Submitted
09-08-2024
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

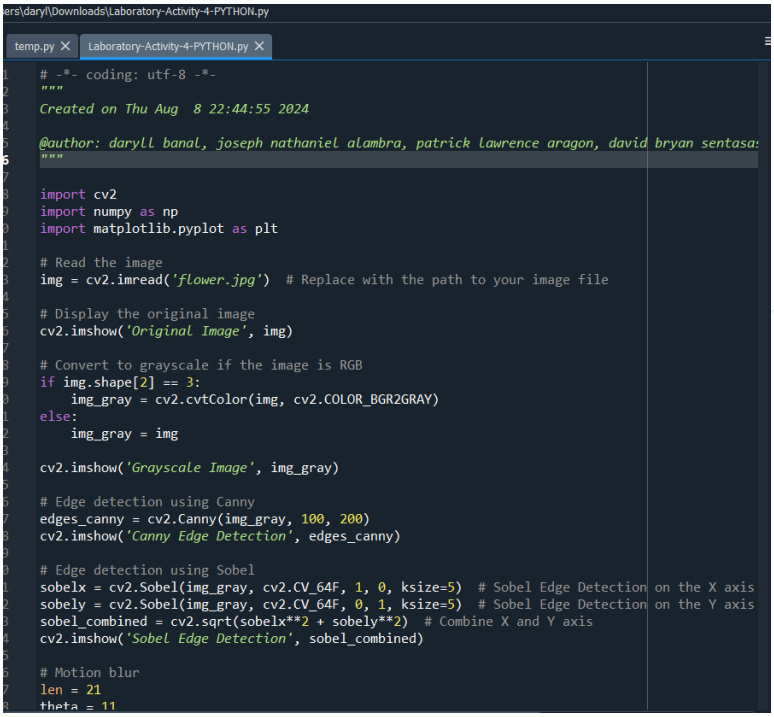
- Copy and paste your MATLAB code (use the original picture file: flower.jpg)

```
% 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');
```

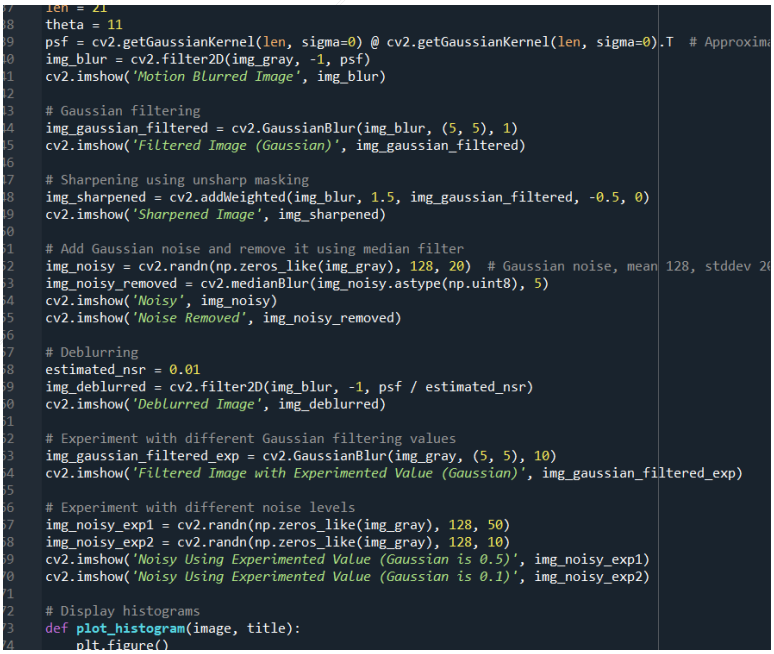
```
% Show the image
figure;
imshow(img_blur);
title('Motion Blurred Image');



% 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



```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug  8 22:44:55 2024

@author: daryll banal, joseph nathaniel alambra, patrick Lawrence aragon, david bryan sentasa
"""

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

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

# Display the original image
cv2.imshow('Original Image', img)

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

cv2.imshow('Grayscale Image', img_gray)

# Edge detection using Canny
edges_canny = cv2.Canny(img_gray, 100, 200)
cv2.imshow('Canny Edge Detection', edges_canny)

# Edge detection using Sobel
sobelx = cv2.Sobel(img_gray, cv2.CV_64F, 1, 0, ksize=5)  # Sobel Edge Detection on the X axis
sobely = cv2.Sobel(img_gray, cv2.CV_64F, 0, 1, ksize=5)  # Sobel Edge Detection on the Y axis
sobel_combined = cv2.sqrt(sobelx**2 + sobely**2)  # Combine X and Y axis
cv2.imshow('Sobel Edge Detection', sobel_combined)

# Motion blur
len = 21
theta = 11
```



```
len = 21
theta = 11
psf = cv2.getGaussianKernel(len, sigma=0) @ cv2.getGaussianKernel(len, sigma=0).T  # Approxima
img_blur = cv2.filter2D(img_gray, -1, psf)
cv2.imshow('Motion Blurred Image', img_blur)

# Gaussian filtering
img_gaussian_filtered = cv2.GaussianBlur(img_blur, (5, 5), 1)
cv2.imshow('Filtered Image (Gaussian)', img_gaussian_filtered)

# Sharpening using unsharp masking
img_sharpened = cv2.addWeighted(img_blur, 1.5, img_gaussian_filtered, -0.5, 0)
cv2.imshow('Sharpened Image', img_sharpened)

# Add Gaussian noise and remove it using median filter
img_noisy = cv2.randn(np.zeros_like(img_gray), 128, 20)  # Gaussian noise, mean 128, stddev 20
img_noisy_removed = cv2.medianBlur(img_noisy.astype(np.uint8), 5)
cv2.imshow('Noisy', img_noisy)
cv2.imshow('Noise Removed', img_noisy_removed)

# Deblurring
estimated_nsr = 0.01
img_deblurred = cv2.filter2D(img_blur, -1, psf / estimated_nsr)
cv2.imshow('Deblurred Image', img_deblurred)

# Experiment with different Gaussian filtering values
img_gaussian_filtered_exp = cv2.GaussianBlur(img_gray, (5, 5), 10)
cv2.imshow('Filtered Image with Experimented Value (Gaussian)', img_gaussian_filtered_exp)

# Experiment with different noise levels
img_noisy_exp1 = cv2.randn(np.zeros_like(img_gray), 128, 50)
img_noisy_exp2 = cv2.randn(np.zeros_like(img_gray), 128, 10)
cv2.imshow('Noisy Using Experimented Value (Gaussian is 0.5)', img_noisy_exp1)
cv2.imshow('Noisy Using Experimented Value (Gaussian is 0.1)', img_noisy_exp2)

# Display histograms
def plot_histogram(image, title):
    plt.figure()
```

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

PYTHON

Figure 1A: Acquire an Image of a Flower

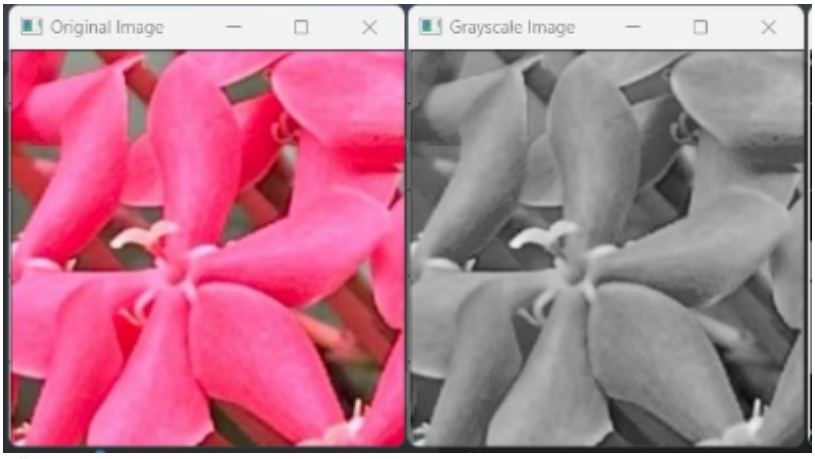


Figure 2A: Original and Grayscale Image

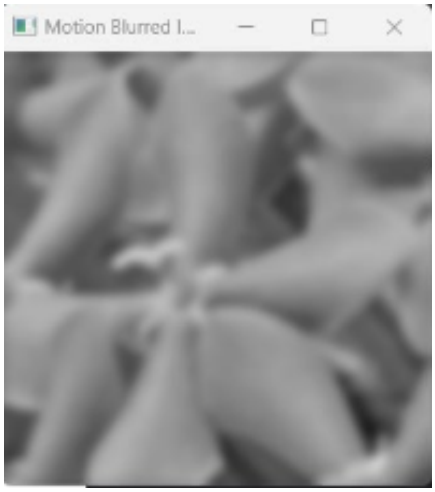


Figure 3A: Motion Blurred Image

Figure 4A: Gaussian-filtered Image

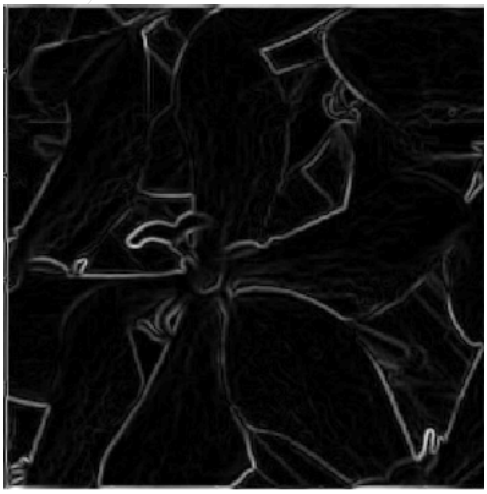


Figure 5A: Edge-detected Image

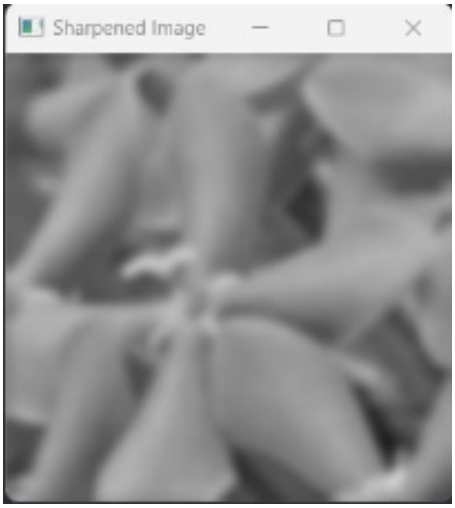


Figure 6A: Sharpen Image

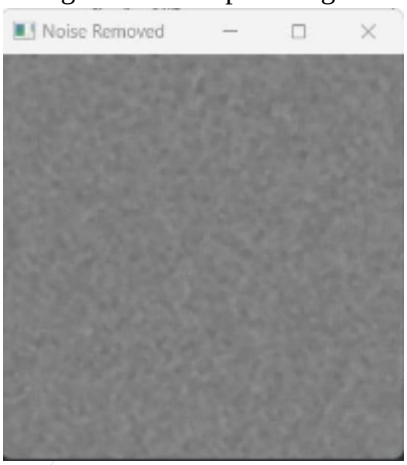
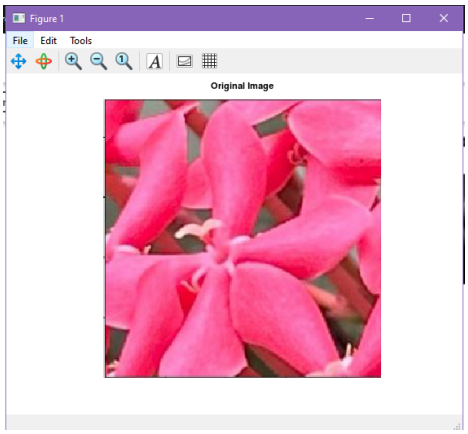


Figure 7A: Added Gaussian Noise and Removed Image

Figure 8A: Deblurred Image

OCTAVE

Figure 1A: Acquire an Image of a Flower



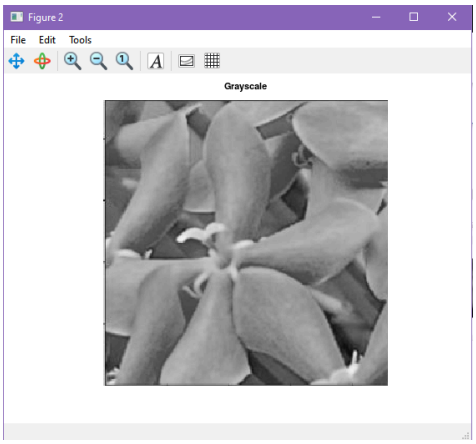


Figure 2B: Original and Grayscale Image

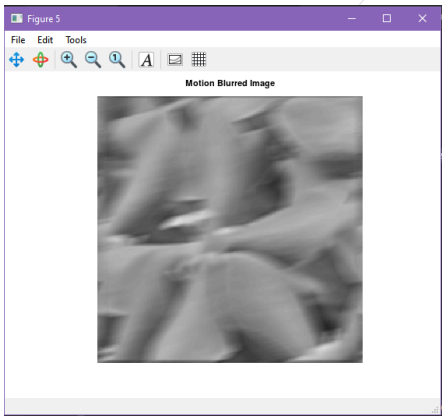


Figure 3B: Motion Blurred Image

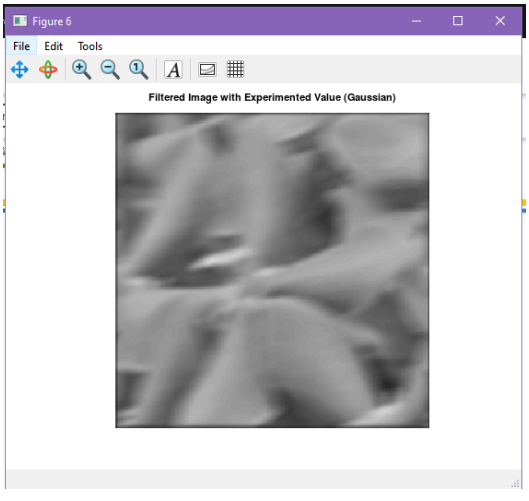


Figure 4B: Gaussian-filtered Image

Figure 5B: Edge-detected Image

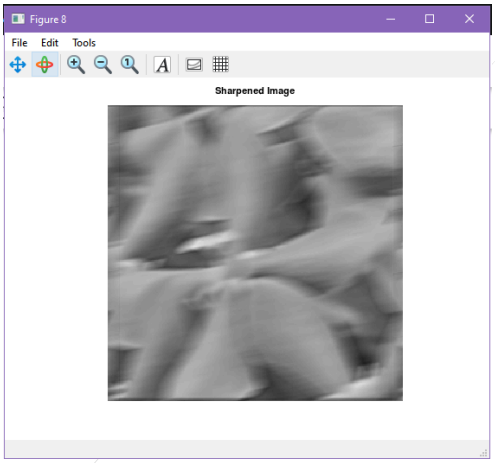


Figure 6B: Sharpen Image

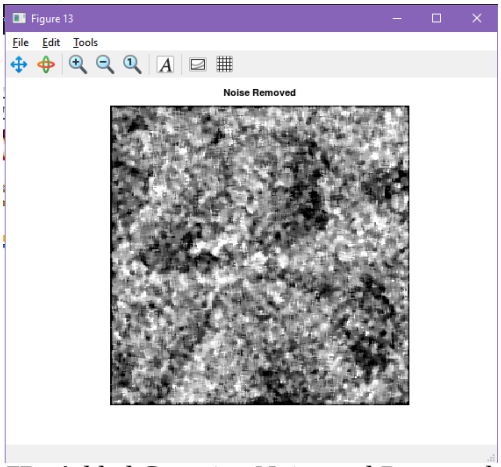


Figure 7B: Added Gaussian Noise and Removed Image

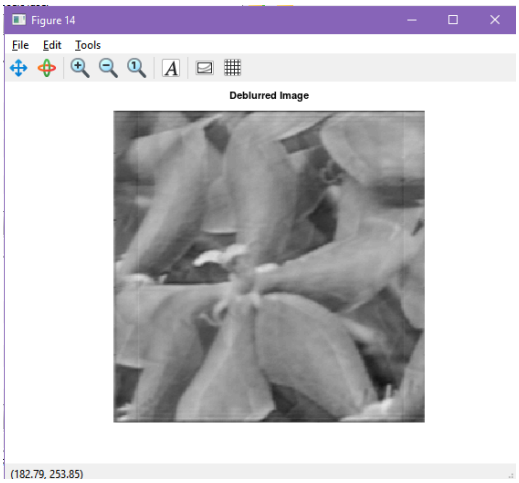


Figure 8B: Deblurred Image

MATLAB

Figure 1C: Acquire an Image of a Flower

Figure 2C: Original and Grayscale Image

Figure 3C: Motion Blurred Image

Figure 4C: Gaussian-filtered Image

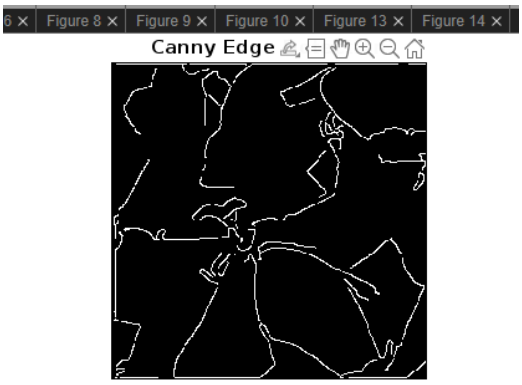


Figure 5C: Edge-detected Image

Figure 6C: Sharpen Image

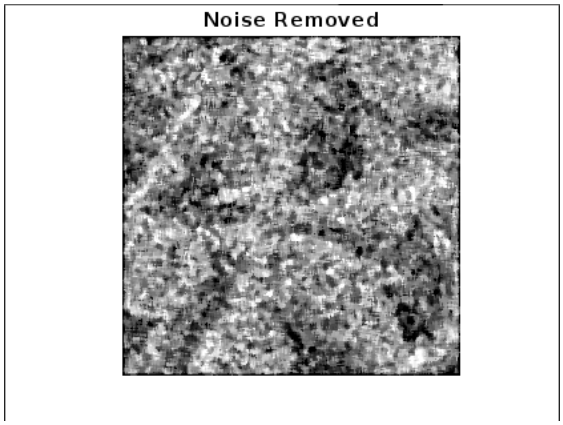


Figure 7C: Added Gaussian Noise and Removed Image

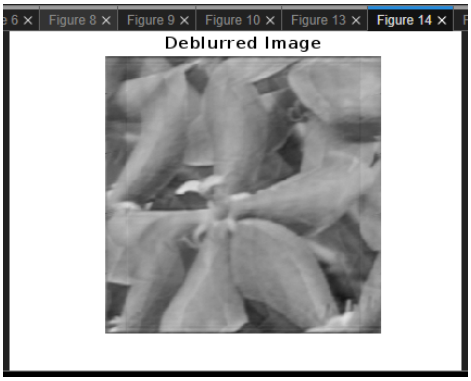


Figure 8C: Deblurred Image

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');
```

PYTHON PARAMETER MODIFICATION

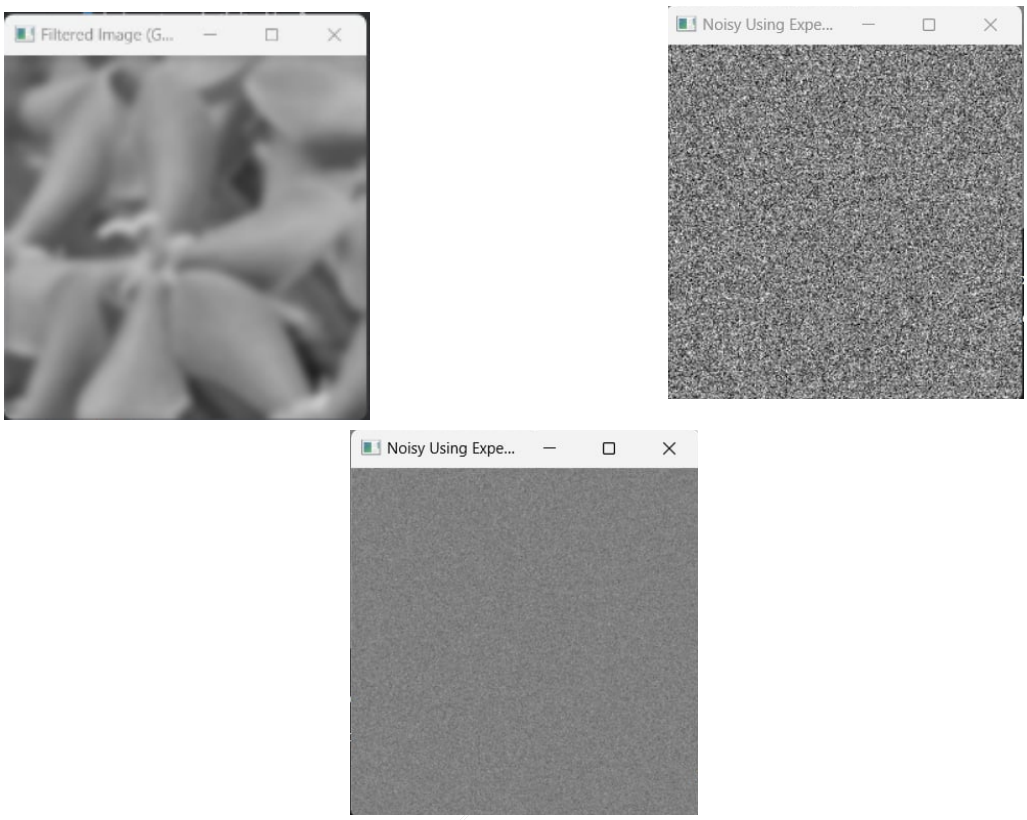
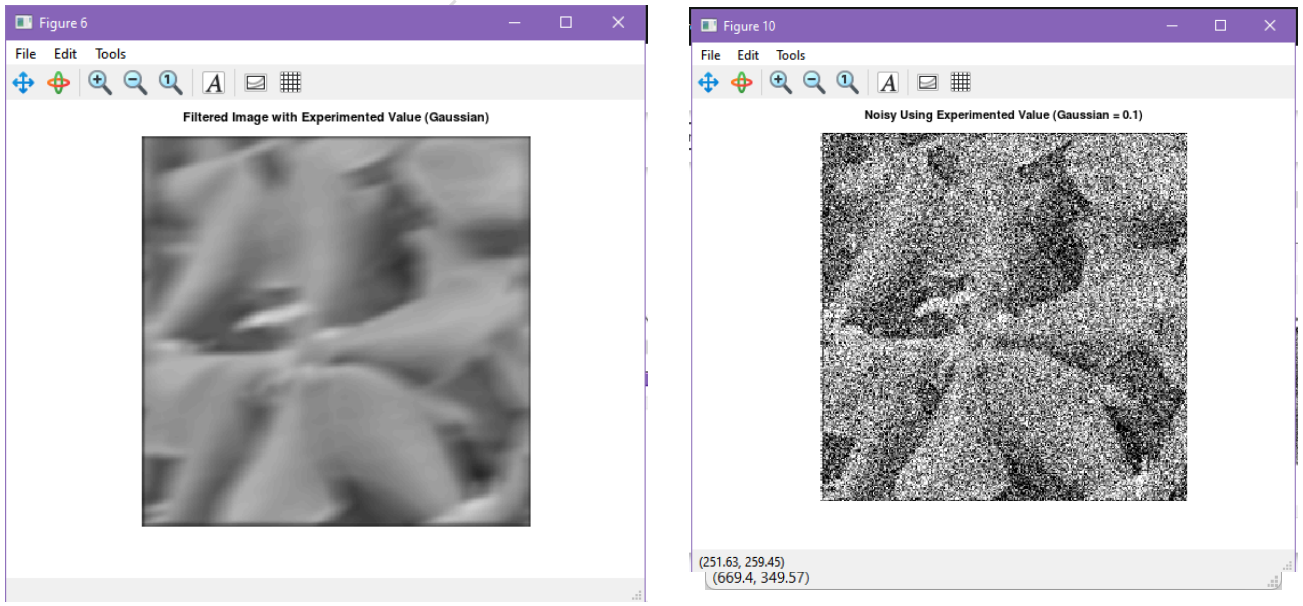


Figure 9: Parameters Modification (Filtered Image, Noisy Image 0.1, Noisy Image 0.5) PYTHON

OCTAVE PARAMETER MODIFICATION



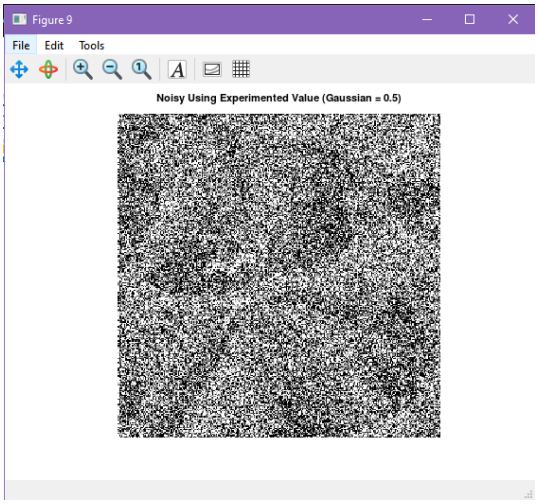


Figure 9: Parameters Modification (Filtered Image, Noisy Image 0.1, Noisy Image 0.5) PYTHON

MATLAB PARAMETER MODIFICATION

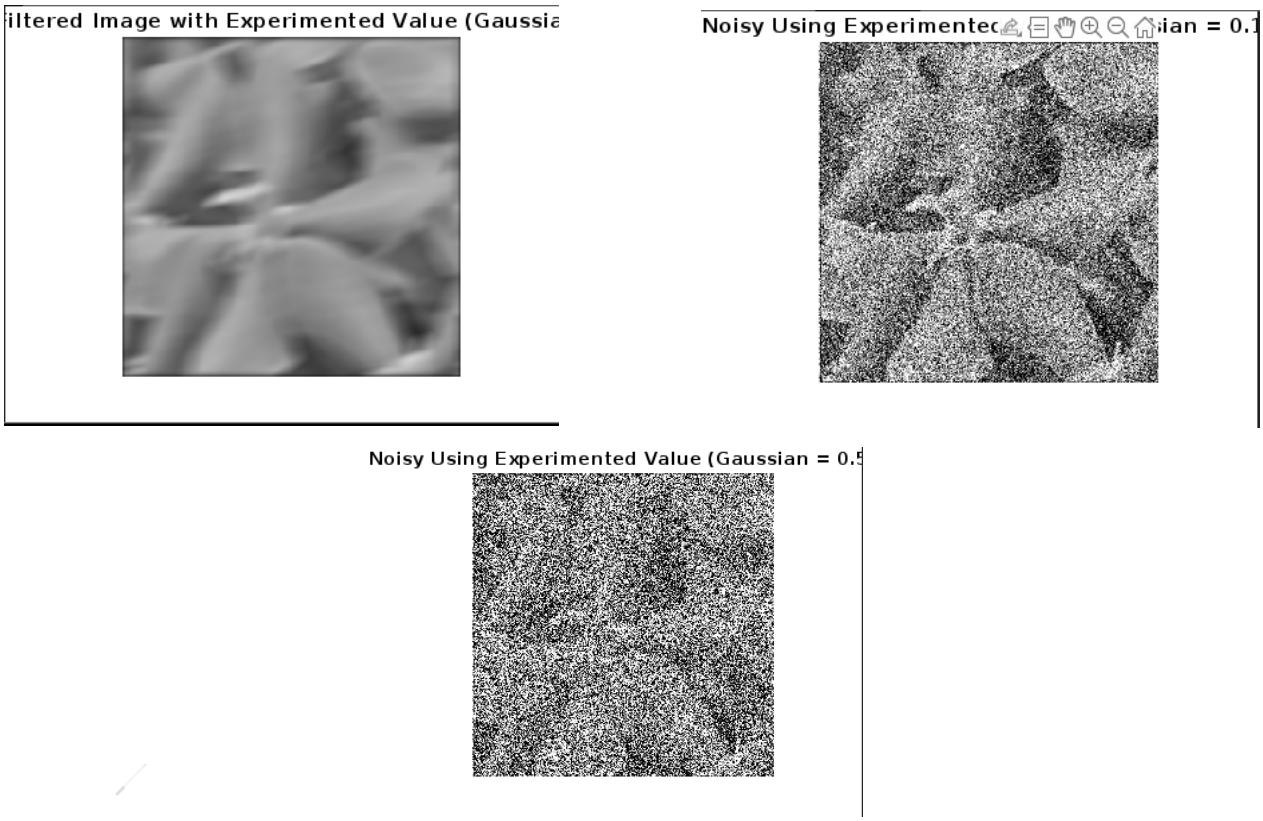


Figure 9: Parameters Modification (Filtered Image, Noisy Image 0.1, Noisy Image 0.5) PYTHON

2. Visualize the results, analyze and interpret:

The first action in the course of any picture processing process is the act of getting the picture from a file. This is an indispensable procedure as it is the basis of other operations. In OpenCV Python, the cv2.imread function is used to bring in an image from a specified file path, which admits different image types. This function, in turn, brings the image into memory for more handling. Likewise, in MATLAB, the imread function allows users to do this by merely selecting images from different types of formats. Octave does not differ in this respect as it uses its own imread function to load the image file in the program and then proceeds to the next step where it is to be pre-processed. The main thing to be achieved at the time of image acquisition is that the image data is the one available for the next analysis and enhancement.

As a matter of fact, Gaussian filtering is an important image restoration technique in which the image is smoothed, thus taking away the noise and improving the overall disposition of the image. In OpenCV Python, this is done via the use of the cv2.GaussianBlur function to apply a Gaussian filter to the picture. Smoothing can be regulated by changing the kernel size and standard deviation, variables typically set to (5, 5) and 1, respectively. The function fspecial of MATLAB is brought into play by computing a Gaussian filter. Once the resultant filter is created, the imfilter function applies it to the picture. Users can adjust the standard deviation and kernel size to get the noise and smoother image. Octave virtually follows the same procedure as MATLAB, where fspecial is used for the design of the Gaussian filter and imfilter is used for its application. A very clear and noise-reduced image appears as the final solution which is the proof of its powerful effect that the Gaussian blurring has on image restoration.

Deblurring is a process for eliminating the blur caused by moving objects during imaging, a problem which often arises in photography. The main purpose of this method is to restore the quality of the original image and remove the blur artifacts. In OpenCV Python, deblurring can be performed by using the cv2.filter2D function along with a specially developed kernel or by Wiener filtering under scipy.signal.convolve2d. MATLAB offers efficient tools through the deconvwnr function that performs Wiener filtering to estimate and correct motion blurs it requires knowledge of the point spread function PSF and the amount of noise-to-signal ratio(NSR) the image has. Likewise, Octave is equipped with a tool to eliminate motion blur using the deconvwnr function that applies the Wiener

filter, thus successfully achieving the results. Both have a wide range of motion and deblurring tools that are tailored towards correcting motion blur and restoring the image to its best quality. Each platform has developed its blurring removal facilities where some customized tools and methods for developers and users to solve the problem of blurring and restoration of image quality.

Every one of the procedures corresponds to various objectives for imaging such as restoration, enhancing, and collecting, which showcases the variety of capabilities these skills give to picture processing instruments.

IV. Conclusion

In this laboratory activity, we successfully demonstrated a comprehensive approach to image processing using OpenCV Python, MATLAB, and Octave. The primary objectives were to acquire an image, apply Gaussian filtering for image restoration, and perform deblurring to correct motion blur.

The laboratory activity highlighted the effectiveness of image processing techniques in improving image quality. By acquiring, restoring, and enhancing images through Gaussian filtering and deblurring, we demonstrated how these methods can significantly impact the clarity and usability of digital images. The successful application of these techniques across OpenCV Python, MATLAB, and Octave underscores the versatility and power of modern image processing tools in achieving high-quality results.

References

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

MATLAB. (n.d.). MATLAB documentation. Retrieved from https://www.mathworks.com/help/matlab/

OpenCV. (n.d.). OpenCV documentation. Retrieved from https://docs.opencv.org/

GNU Octave. (n.d.). GNU Octave documentation. Retrieved from https://www.gnu.org/software/octave/doc/interpreter/

Gonzalez, R. C., & Woods, R. E. (2018). Digital image processing (4th ed.). Pearson.