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

Elective 3

Laboratory Activity No. 3
Image Enhancement

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
02-08-2024

Submitted to:

Engr. Maria Rizette H. Sayo

I. Objectives

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

1. Acquire the image.
2. Show histogram equalization.
3. Show contrast enhancement.
4. Show filtering in the spatial domain (average and median).

II. Methods

A. Perform a task given in the presentation

- Copy and paste your MATLAB code

```
% Read an image
img = imread('flower.jpg');
% 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 Image');
% Contrast enhancement using imadjust
img_contrast_enhanced = imadjust(img_gray);
% Display the contrast-enhanced image
figure;
imshow(img_contrast_enhanced);
title('Contrast Enhanced Image (imadjust)');
% Histogram equalization
img_histeq = histeq(img_gray);
% Display the histogram equalized image
figure;
imshow(img_histeq);
title('Equalized Image');
```

```
% Filtering using average filter
h_avg = fspecial('average', [5, 5]);
img_avg_filtered = imfilter(img_gray, h_avg);
% Display the average filtered image
figure;
imshow(img_avg_filtered);
title('Filtered Image (Average)');
% Filtering using median filter
img_median_filtered = medfilt2(img_gray, [5, 5]);
% Display the median filtered image
figure;
imshow(img_median_filtered);
title('Filtered Image (Median)');
% Display histograms for comparison
% Grayscale histogram
figure;
imhist(img_gray);
title('Histogram of Grayscale');
% Enhanced histogram (imadjust)
figure;
imhist(img_contrast_enhanced);
title('Histogram of Enhanced Image');
% Equalized histogram
figure;
imhist(img_histeq);
title('Histogram of Equalized Image');
% Histogram (Average Filtered)
figure;
imhist(img_avg_filtered);
title('Histogram of Average Filtered)');
% Histogram (Median Filtered)
figure;
imhist(img_median_filtered);
title('Histogram of Median Filtered)');
```

B. Supplementary Activity

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

```
import cv2
import numpy as np

# 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

# Contrast Enhanced Image
enhance = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))  # CLAHE: Contrast
Limited Adaptive Histogram Equalization
contrastEnhImg = enhance.apply(grayImg)

# Equalized Image
equalizedImg = cv2.equalizeHist(grayImg)

# Filtering using average filter
# Apply average Filter
kernelSize = (5, 5)  # 5 pixels wide and 5 pixels tall
aveFilImg = cv2.blur(grayImg, kernelSize)

# Median Filter
# Apply median Filter
kernelSizeMF = 5
medFilImg = cv2.medianBlur(grayImg, kernelSizeMF)

# 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)

    # 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

# Calculation of Histograms
histGrayImg = cv2.calcHist([grayImg], [0], None, [256], [0, 256])
histEnhImg = cv2.calcHist([contrastEnhImg], [0], None, [256], [0, 256])
histEqualizedImg = cv2.calcHist([equalizedImg], [0], None, [256], [0, 256])
histAveFilImg = cv2.calcHist([aveFilImg], [0], None, [256], [0, 256])
histMedFilImg = cv2.calcHist([medFilImg], [0], None, [256], [0, 256])

# Create histogram images
histogramOfGrayImg = create_histogram(histGrayImg)
histogramOfEnhImg = create_histogram(histEnhImg)
histogramOfEqualImg = create_histogram(histEqualizedImg)
histogramOfAveFilImg = create_histogram(histAveFilImg)
histogramOfMedFilImg = create_histogram(histMedFilImg)

# Display images and histograms
cv2.imshow('Original Image', img)
cv2.imshow('Grayscale Image', grayImg)
cv2.imshow('Contrast Enhanced Image', contrastEnhImg)
cv2.imshow('Equalized Image', equalizedImg)
cv2.imshow('Filtered Image (Average)', aveFilImg)
cv2.imshow('Filtered Image (Median)', medFilImg)
cv2.imshow('Histogram of Grayscale Image', histogramOfGrayImg)
cv2.imshow('Histogram of Enhanced Image', histogramOfEnhImg)
cv2.imshow('Histogram of Equalized Image', histogramOfEqualImg)
cv2.imshow('Histogram of Average Filtered', histogramOfAveFilImg)
cv2.imshow('Histogram of Median Filtered', histogramOfMedFilImg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

III. Results

a. MATLAB results

picture file: flower.jpg

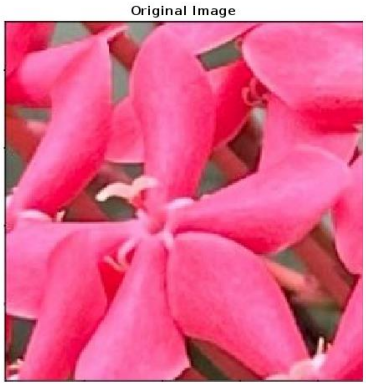

Figure 1: Acquire an Image of a Flower

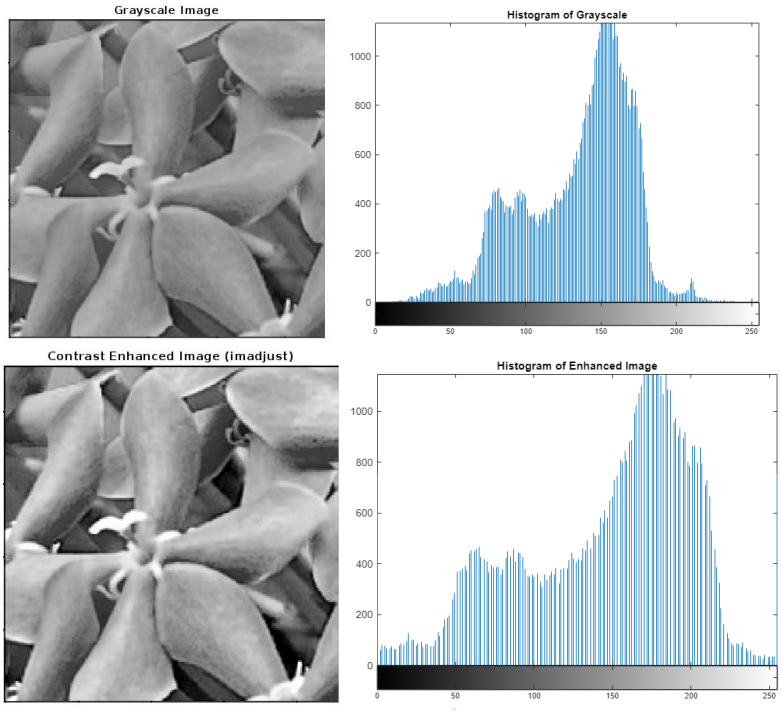

Figure 2: Grayscale, Contrast Enhancement, and its Histogram

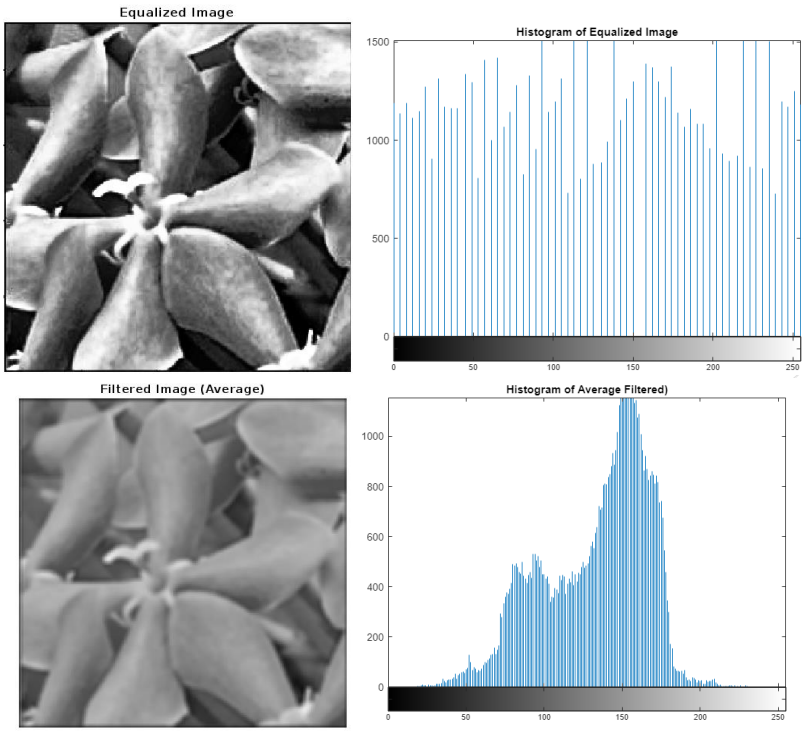

Figure 3: Histogram Equalized and Average Filtered Image and Its Histogram

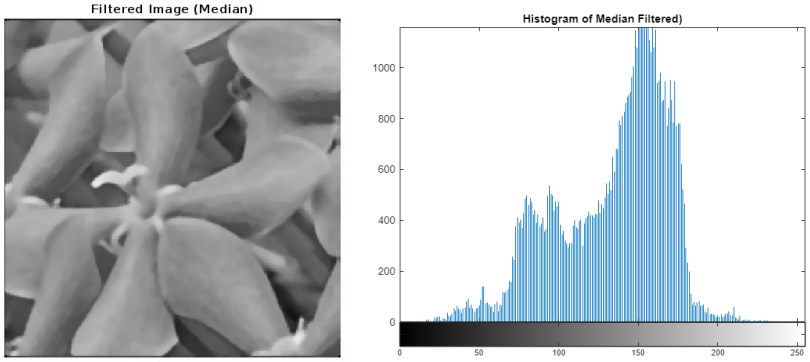

Figure 4: Median Filtered Image and Its Histogram

b. OpenCV results

picture file: flower.jpg

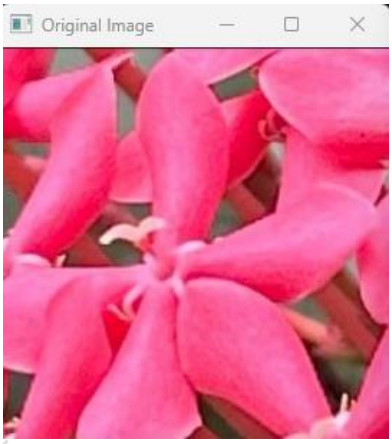

Figure 1: Acquire an Image of a Flower

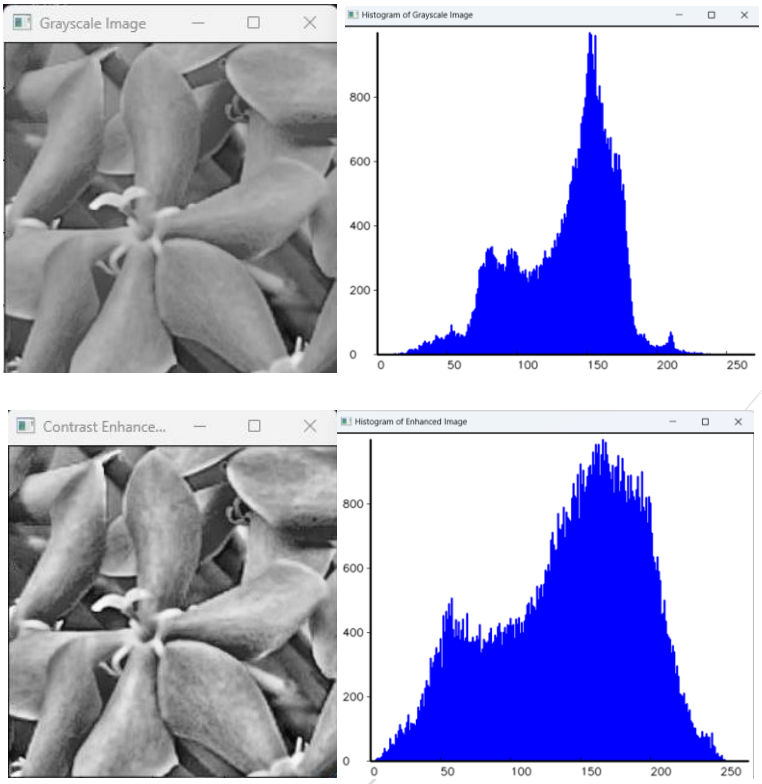

Figure 2: Grayscale, Contrast Enhancement, and its Histogram

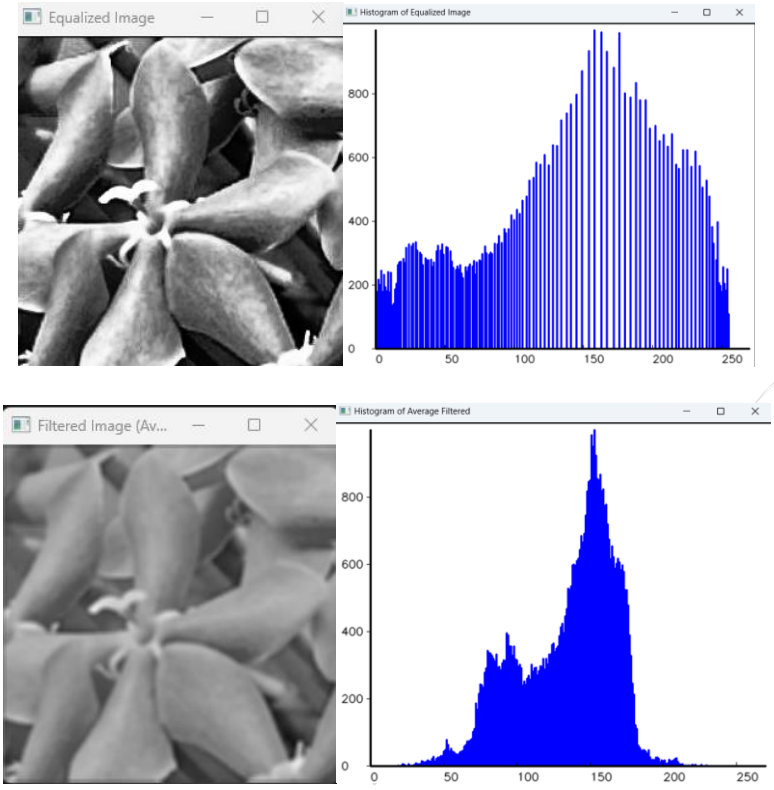

Figure 3: Histogram Equalized and Average Filtered Image and Its Histogram

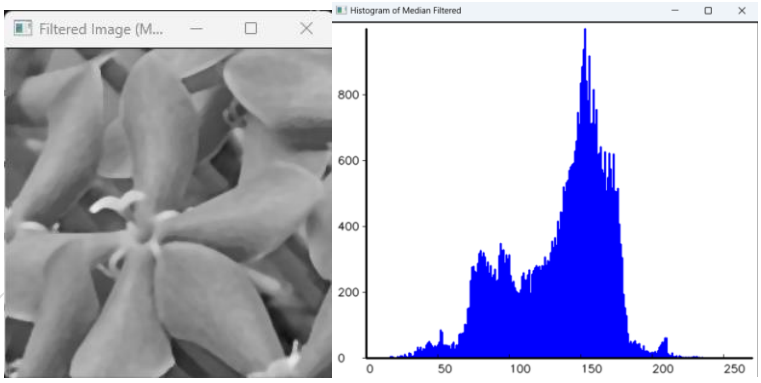

Figure 4: Median Filtered Image and Its Histogram

These codes perform the following:

1. Grayscale conversion, which converts a color image (RGB) to a single-channel grayscale image. Colors are lost, but information about brightness is preserved. This depends on the desired outcome. If color information isn't crucial and you want to focus on brightness variations or prepare the image for further processing, grayscale conversion is effective. So in our image our original image is bright hence using the grayscale conversion is effective for our image that will be applied to other functions.
2. The Contrast Enhancement, which uses the function imadjust, stretches the contrast of the image by adjusting pixel values. Darker pixels become darker, and brighter pixels become brighter. This can make details in low-contrast areas more visible. The imadjust is effective for improving the visibility of features in images with low contrast. However, it can sometimes create an unnatural appearance or exaggerate noise in the image.
3. The Histogram Equalization uses the function histeq, which redistributes the pixel intensities in the image to create a flat histogram. This aims to achieve a more even distribution of brightness across the image. It is effective for images with uneven lighting or where specific features are obscured due to a concentration of pixels in a certain brightness range. It can enhance overall contrast and detail. However, it may sometimes create an overly artificial look or introduce artifacts.
4. Average filtering uses the function imfilter which replaces each pixel with the average value of its surrounding pixels which reduces noise in the image by blurring sharp edges and details. The average filter is effective for reducing random noise but can also blur important image features. It's good for removing minor noise while preserving larger structures.
5. Median filtering uses the function medfilt2 which replaces each pixel with the median value of its surrounding pixels. Similar to the average filter, it reduces noise but is less prone to blurring edges. It's particularly effective for removing salt-and-pepper noise (random black and white pixels). The median filter offers a good balance between noise reduction and edge preservation.

And lastly, each image uses a histogram through a function hist, which helps visualize the distribution of pixel intensities. Visualizing histograms allows you to understand the original contrast distribution (grayscale) and how it's affected by the applied algorithms (contrast enhancement, equalization, filtering). This helps assess the effectiveness of each step.

Parameter Modification

<You can modify it to explore other functionalities>

```
% Convert to grayscale if the image is RGB
if size(img, 3) == 3
    img_gray = rgb2gray(img);
else
    img_gray = img;
end
% Filtering using average filter but different values
h_avg = fspecial('average', [10, 10]); % Original is [5,5]
img_avg_filtered = imfilter(img_gray, h_avg);

% Show the experimented image
figure;
imshow(img_avg_filtered);
title('Filtered Image (Using Average but Different values)');

% Filtering using median filter
img_median_filtered = medfilt2(img_gray, [1, 10]); % Original is [5,5]

% Display the median filtered image
figure; imshow(img_median_filtered);
title('Experimented Filtered Image (Median)');

% Show the Histogram
figure;
imhist(img_median_filtered);
title('Histogram of Experimented Median Filtered)');
```

b. MATLAB (parameters modified)

Code:

```
% Convert to grayscale if the image is RGB
if size(img, 3) == 3
img_gray = rgb2gray(img);
else
img_gray = img;
end
% Filtering using average filter but different values
h_avg = fspecial('average', [10, 10]); % Original is [5,5]
img_avg_filtered = imfilter(img_gray, h_avg);
% Show the experimented image
figure;
imshow(img_avg_filtered);
title('Filtered Image (Using Average but Different values)');
% Filtering using median filter
img_median_filtered = medfilt2(img_gray, [1, 10]); % Original is [5,5]
% Display the median filtered image
figure; imshow(img_median_filtered);
title('Experimented Filtered Image (Median)');
% Show the Histogram
figure;
imhist(img_median_filtered);
title('Histogram of Experimented Median Filtered)');
```

Results:

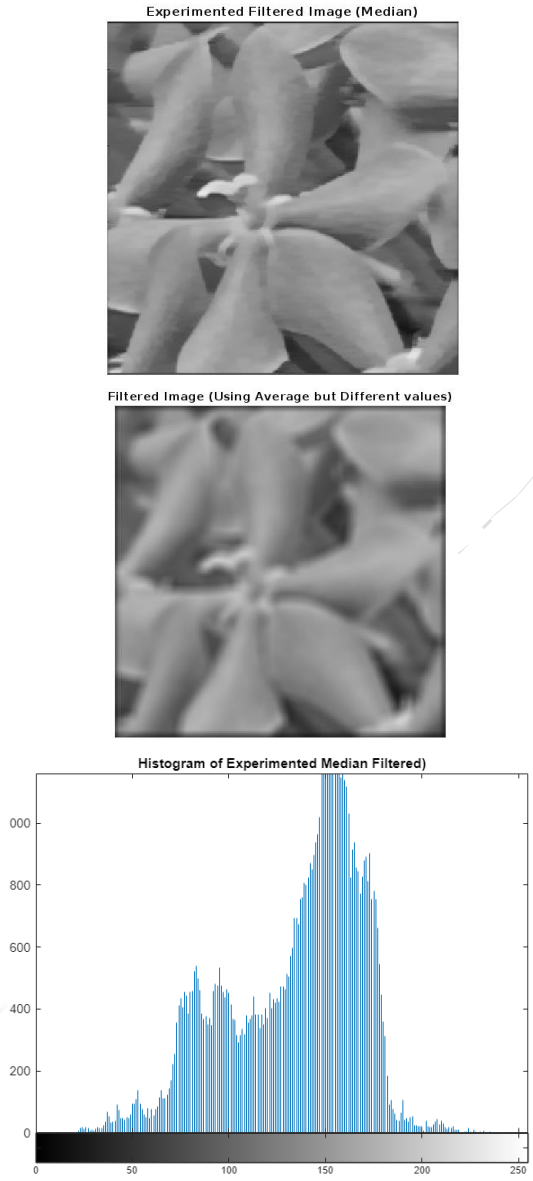

Figure 5 (MATLAB): Parameters Modification and Its Histogram

c. OpenCV (parameters modified)

Code:

```
import cv2
import numpy as np

# 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

# Contrast Enhanced Image
enhance = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))  # CLAHE: Contrast
Limited Adaptive Histogram Equalization
contrastEnhImg = enhance.apply(grayImg)

# Equalized Image
equalizedImg = cv2.equalizeHist(grayImg)

# Filtering using average filter with updated kernel size
kernelSize = (10, 10)  # Updated to 10 pixels wide and 10 pixels tall
aveFilImg = cv2.blur(grayImg, kernelSize)

# Median Filter with an odd kernel size
kernelSizeMF = 11  # Must be an odd number
medFilImg = cv2.medianBlur(grayImg, kernelSizeMF)

# 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)

    # 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

# Calculation of Histograms
histGrayImg = cv2.calcHist([grayImg], [0], None, [256], [0, 256])
histEnhImg = cv2.calcHist([contrastEnhImg], [0], None, [256], [0, 256])
histEqualizedImg = cv2.calcHist([equalizedImg], [0], None, [256], [0, 256])
histAveFilImg = cv2.calcHist([aveFilImg], [0], None, [256], [0, 256])
histMedFilImg = cv2.calcHist([medFilImg], [0], None, [256], [0, 256])

# Create histogram images
histogramOfGrayImg = create_histogram(histGrayImg)
histogramOfEnhImg = create_histogram(histEnhImg)
histogramOfEqualImg = create_histogram(histEqualizedImg)
histogramOfAveFilImg = create_histogram(histAveFilImg)
histogramOfMedFilImg = create_histogram(histMedFilImg)

# Display images and histograms
cv2.imshow('Original Image', img)
cv2.imshow('Grayscale Image', grayImg)
cv2.imshow('Contrast Enhanced Image', contrastEnhImg)
cv2.imshow('Equalized Image', equalizedImg)
cv2.imshow('Filtered Image (Using Average but Different values)', aveFilImg)
cv2.imshow('Experimented Filtered Image (Median)', medFilImg)
cv2.imshow('Histogram of Grayscale Image', histogramOfGrayImg)
cv2.imshow('Histogram of Enhanced Image', histogramOfEnhImg)
cv2.imshow('Histogram of Equalized Image', histogramOfEqualImg)
cv2.imshow('Histogram of Average Filtered', histogramOfAveFilImg)
cv2.imshow('Histogram of Experimented Median Filtered', histogramOfMedFilImg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Results:

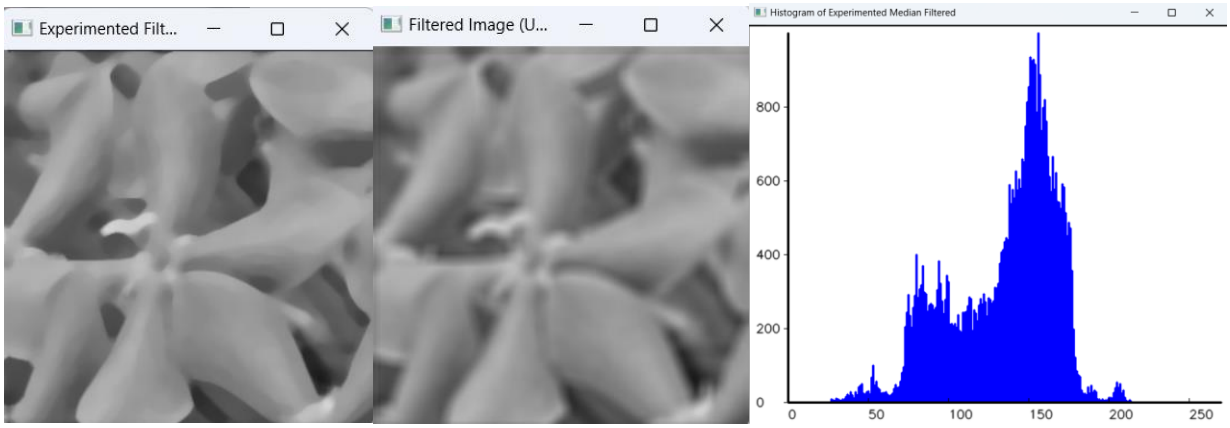

Figure 5 (OpenCV): Parameters Modification and Its Histogram

1. Visualize the results, analyze and interpret:

a. MATLAB:

The MATLAB code begins by reading an image file named 'flower.jpg' and displaying it with the title 'Original Image'. It then checks if the image is in RGB format and converts it to grayscale if necessary, displaying the result with the title 'Grayscale Image'. The grayscale image undergoes contrast enhancement using the imadjust function, and the enhanced image is displayed with the title 'Contrast Enhanced Image (imadjust)'. Next, histogram equalization is performed on the grayscale image using histeq, and the resulting image is shown with the title 'Equalized Image'. The code also applies an average filter using a 5x5 filter kernel created by fspecial, and the filtered image is displayed with the title 'Filtered Image (Average)'. Additionally, a median filter is applied using medfilt2 with a 5x5 neighborhood, and the median filtered image is displayed with the title 'Filtered Image (Median)'. Finally, histograms of the grayscale image and each processed version are displayed for comparison, with each histogram shown in a separate figure window and appropriately titled.

b. MATLAB Parameter Modification

The code snippet performs a series of image processing operations, starting with the conversion of an image to grayscale, followed by applying different filters, and finally displaying the processed images along with their histograms. First, the code checks if the input image, stored in the variable img, is an RGB image by evaluating the size of its third dimension using size(img, 3). If the image has three color channels, it is converted to grayscale using the rgb2gray function. Next is if the image is already grayscale, it remains unchanged. The grayscale image is then assigned to the variable img_gray. Next, the code creates an average filter with a kernel size of 10x10 using the fspecial function, stored in the variable h_avg. This filter is different from the previously used 5x5 filter. The grayscale image is filtered using this average filter through the imfilter function, and the resulting filtered image is stored in img_avg_filtered. This filtered image is then displayed in a new figure window with the title 'Filtered Image (Using Average but Different values)'. Afterward, the code applies a median filter to the grayscale image using the medfilt2 function with a kernel size of 1x10, which is a variation from the previous 5x5 kernel. The median filtered image is stored in img_median_filtered. This processed image is displayed in another figure window with the title 'Experimented Filtered Image (Median)'. Finally, the code generates and displays the histogram of the median filtered image using the imhist function. The histogram provides a visual representation of the pixel intensity distribution in the median filtered image, and it is shown in a new figure window titled 'Histogram of Experimented Median Filtered'.

c. **Python:**

The given Python code uses OpenCV to perform image processing on 'flower.jpg', including converting it to grayscale, enhancing contrast, applying filters, and displaying the results with their histograms. Initially, the necessary libraries (OpenCV and NumPy) are imported. Numpy was imported because it supports multi-dimensional arrays, which are essential for manipulating image data, calculating histograms, and normalizing values. The image is read into the variable img and converted to grayscale as grayImg if the image is in BGR format. Contrast enhancement is achieved using CLAHE with specified clip limit and tile grid size, producing contrastEnhImg. Histogram equalization is applied to the grayscale image, resulting in equalizedImg. An average filter with a 5x5 kernel is applied, generating aveFilImg, while a median filter with a kernel size of 5 produces medFilImg. Histograms for each processed image are calculated using cv2.calcHist, normalized, and displayed. For each histogram, a blank image is created, and lines are drawn to represent histogram values. *cv2.calcHist([image], [0], None, [256], [0, 256])* calculates the histogram of pixel intensities, which is then normalized with *cv2.normalize* to fit the display size for effective

visualization of the histogram. This process is repeated for the grayscale image (histogramOfGrayImg), contrast-enhanced image (histogramOfEnhImg), equalized image (histogramOfEqualImg), average filtered image (histogramOfAveFilImg), and median filtered image (histogramOfMedFilImg). Finally, the original image and all processed images along with their histograms are displayed in separate windows using cv2.imshow, each labeled appropriately. The script concludes with cv2.waitKey(0), keeping the windows open until a key is pressed.

d. **Python Parameter Modification:**

With the parameters modified, an average filter with a kernel size of 10x10 is applied to the grayscale image using cv2.blur, and the filtered image is stored in aveFilImg. Similarly, a median filter with a kernel size of 11 (*cv2 only accepts odd value for median blur*) is applied using cv2.medianBlur, producing medFilImg. For the average filtered image, the histogram is calculated using cv2.calcHist, and a blank image histogramOfAveFilImg is created to display it. The same steps are repeated for the median filtered image, resulting in histogramOfMedFilImg. Finally, the average filtered image, median filtered image, and the histogram of the median filtered image are displayed in separate windows using cv2.imshow, each labeled appropriately. The cv2.waitKey(0) function is called to keep the windows open until a key is pressed, allowing the user to view the images and histograms.

IV. Conclusion

In both the MATLAB and Python code, essential functions for image processing are used to convert images to grayscale, apply various filtering techniques, and analyze their effects through the illustration of histograms. The MATLAB code employs functions like rgb2gray, imadjust, histeq, fspecial, and medfilt2 to process the image, each contributing to different aspects of enhancement and noise reduction. Similarly, the Python code utilizes cv2.cvtColor, cv2.createCLAHE, cv2.equalizeHist, cv2.blur, and cv2.medianBlur to achieve comparable results. Both approaches demonstrate how these functions allow for image manipulation by adjusting contrast, smoothing, and visualizing pixel intensity distributions, with histograms serving as a tool to visualize each difference.

References

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