Name: Phạm Huỳnh Quý An

Class: AI1703 – FPT university HCM

ID Number: SE171139 ***Subject:*** Computer Vision – CPV301

Workshop #2: Image processing

Requirements:

In this exercise, students are asked to write a simple image processing program that has the following basic functions: performing color balance, calculating histogram- performing histogram equalization. Then apply filters like median filter, mean filter, and Gaussian smoothing. Details of the functions are described below:

Function 1: color balance, to perform this function, the user needs to enter the necessary parameters to perform color balance. (can use the slider to represent it visually)

Function 2: Show histogram and enter the necessary information to perform histogram equalization.

Function 3: implement the median filter to remove noise in the image(salt and pepper noise)

Function 4: implement the Mean filter to remove noise in image (salt and pepper noise)

Function 5: implement Gaussian smoothing to perform image smoothing.

Code – File Name: SE171139_Workshop2

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import random

def empty_function():
    pass

def color_balance(img):
    balanced_img = img.copy()

    cv.namedWindow('Balanced Image')
    cv.createTrackbar('Blue', 'Balanced Image', 0, 255, empty_function)
    cv.createTrackbar('Green', 'Balanced Image', 0, 255, empty_function)
    cv.createTrackbar('Red', 'Balanced Image', 0, 255, empty_function)

    while (True):


        blue = cv.getTrackbarPos('Blue','Balanced Image')
        green = cv.getTrackbarPos('Green', 'Balanced Image')
        red = cv.getTrackbarPos('Red', 'Balanced Image')

        img_new = np.zeros(img.shape, dtype=np.uint8)
        img_new[:, :] = [blue, green, red]
        img_result = cv.addWeighted(balanced_img, 0.7, img_new, 0.3, 0)

        cv.imshow('Balanced Image', img_result)
```

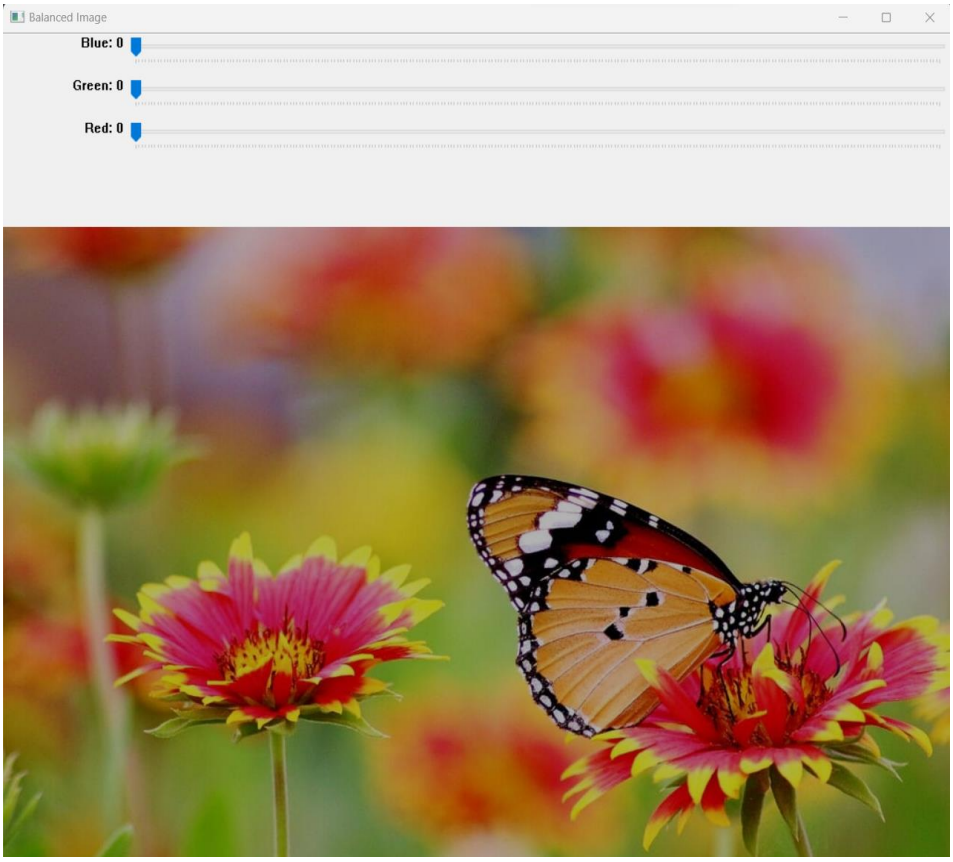
```
        if cv.waitKey(1) == 27:
            break

    cv.destroyAllWindows()


def drawHistogram(img,title='src'):
    """
    Draw histogram of given img
    :param img:
    :return: draw histogram of the img
    """

    fig, ax = plt.subplots()
    x = np.linspace(0,255,256)
    y = np.zeros(x.shape,np.float64)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            y[img[row,col]]+=1
    cum_hist = np.zeros(y.shape,np.float64)
    cum_hist[0]=y[0]
    for i in range(1,len(cum_hist)):
        cum_hist[i]=cum_hist[i-1]+y[i]
    max_hist = np.float64(np.max(y))
    max_cum_hist = np.float64(np.max(cum_hist))
    # Nomarlization for drawing
    y *= (255./max_hist)
    cum_hist *=(255./max_cum_hist)
    plt.bar(x,y,color='b',label="histogram")
    plt.plot(x,cum_hist,color='r',label="Cummulate histogram")
    plt.legend(["Cummulate histogram","histogram"])
    ax.set_xlim(0,256)
    ax.set_ylim(0,256)
    ax.set_title("Histogram "+title)
    #cv.imshow(title, img)
    plt.show()
    #cv.waitKey(0)

def histogram_equalization(img,bShow=False):
    """
    :param img: input image (gray scale)
    :param bShow: whether to show image or not
    :return: equalised histogram image
    """
    #assert len(img.shape)==2,"Must be a gray image"
    #1.Calculate hist of image
    hist = np.zeros(256,np.int16)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            hist[img[row,col]]+=1
    #print(hist)
    #2.Caculate cummulate sum his of the image
    cum_hist = np.zeros(256,np.int64)
    cum_hist[0]=hist[0]
    for i in range(1,len(hist)):
        cum_hist[i]=cum_hist[i-1]+hist[i]

    max_cum_hist = np.max(cum_hist)
    min_cum_hist = np.min(cum_hist)
    print(cum_hist)
    #3.create map for old pixel values
    hist_map = np.zeros(256, np.uint8)
    for i in range(len(cum_hist)):
        hist_map[i]=np.uint8((cum_hist[i]-min_cum_hist)/(max_cum_hist-
min_cum_hist)*255.)
```

```
    #4.create histogram equalised image
    new_img = np.zeros(img.shape,np.uint8)
    for row in range(new_img.shape[0]):  # traverse by row (y-axis)
        for col in range(new_img.shape[1]):  # traverse by column (x-axis)
            new_img[row, col] = hist_map[img[row, col]]

    if bShow:
        cv.imshow("source",img)
        drawHistogram(img)
        cv.imshow("Histogram equalization",new_img)
        drawHistogram(new_img)
        cv.waitKey(0)
    return new_img

def add_noise(img):
    # Getting the dimensions of the image
    row = img.shape[0]
    col = img.shape[1]

    # Randomly pick some pixels in the
    number_of_pixels = random.randint(300, 15000)
    for i in range(number_of_pixels):
        y_coord = random.randint(0, row - 1)
        x_coord = random.randint(0, col - 1)
        img[y_coord][x_coord] = 255

    # Randomly pick some pixels in
    number_of_pixels = random.randint(300, 15000)
    for i in range(number_of_pixels):
        y_coord = random.randint(0, row - 1)
        x_coord = random.randint(0, col - 1)
        img[y_coord][x_coord] = 0

    return img

def median_filter(image, kernel_size):
    height, width = image.shape

    result = np.zeros((height, width), dtype=np.uint8)

    # Xác định số phần tử của kernel
    k = kernel_size // 2
    for i in range(k, height - k):
        for j in range(k, width - k):
            # Lấy phần tử của kernel tương ứng với pixel hiện tại
            kernel = image[i - k:i + k + 1, j - k:j + k + 1]
            # Sắp xếp các giá trị trong kernel và lấy giá trị median
            median_value = np.median(kernel)
            # Gán giá trị median cho pixel tại vị trí hiện tại
            result[i][j] = median_value
    return result


def mean_filter(image, kernel_size):
    height, width = image.shape

    result = np.zeros((height, width), dtype=np.uint8)

    # Xác định số phần tử của kernel
    k = kernel_size // 2
    for i in range(k, height - k):
        for j in range(k, width - k):
            kernel = image[i - k:i + k + 1, j - k:j + k + 1]
            mean_value = np.mean(kernel)
            result[i][j] = mean_value
    return result
```

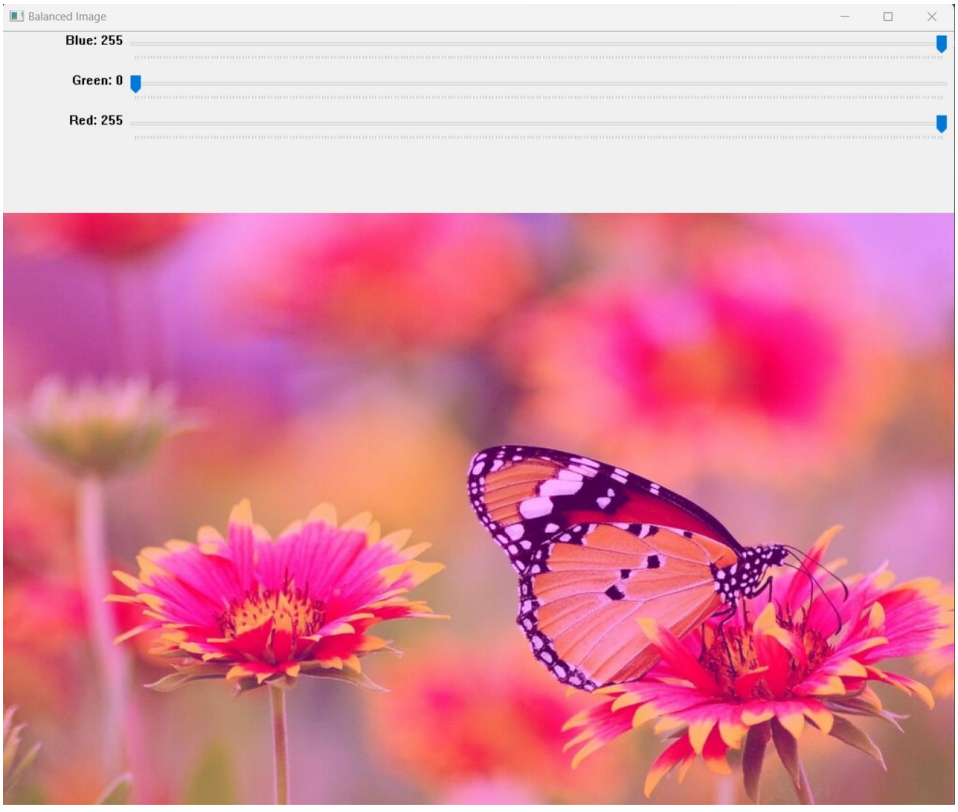
```python
def gaussian_filter(image, kernel_size=3, sigma=1):
    # Tạo kernel theo hàm Gaussian
    kernel = np.zeros((kernel_size, kernel_size))
    center = kernel_size // 2

    for i in range(kernel_size):
        for j in range(kernel_size):
            x = i - center
            y = j - center
            kernel[i, j] = np.exp(-(x ** 2 + y ** 2) / (2 * sigma ** 2))

    kernel = kernel / (2 * np.pi * sigma ** 2)
    kernel = kernel / kernel.sum()

    # Áp dụng filter
    filtered_image = cv.filter2D(image, -1, kernel)
    return filtered_image

def main():
    Buom = cv.imread('Buom.jpg')
    Rush = cv.imread('Rush.jpg')


    #Function 1
    color_balance(Buom)
    gray = cv.cvtColor(Buom, cv.COLOR_BGR2GRAY)

    #Function 2
    drawHistogram(gray)
    histogram_equalization(gray, True)

    #Function 3 and 4
    noise_gray = add_noise(gray)
    median_gray = median_filter(noise_gray,3)
    mean_gray = mean_filter(noise_gray,3)
    cv.imshow('Noise Image',noise_gray)
    cv.imshow('Median Image',median_gray)
    cv.imshow('Mean Image',mean_gray)
    cv.waitKey(0)

    #Function 5
    Gaussian_buom = gaussian_filter(Rush)
    cv.imshow('Original Image',Rush)
    cv.imshow('Gaussian Image',Gaussian_buom)
    cv.waitKey(0)

if (__name__)=='__main__':
    main()
```

Result:

Function 1: color balance, to perform this function, the user needs to enter the necessary parameters to perform color balance. (can use the slider to represent it visually)



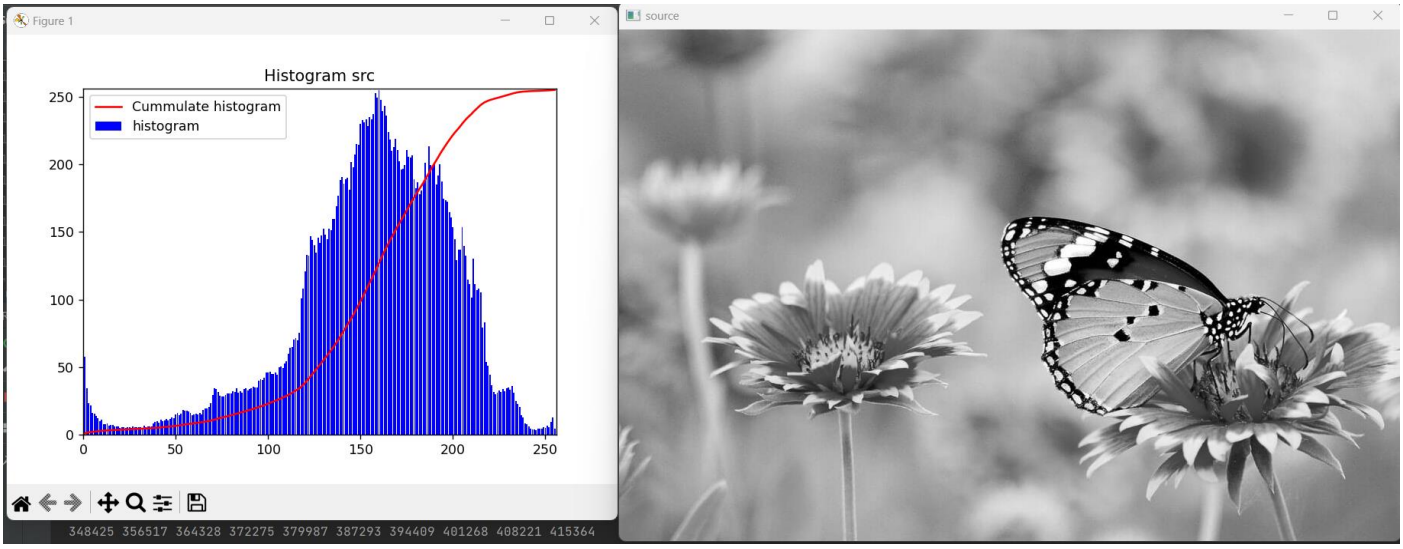
Before performing color balance



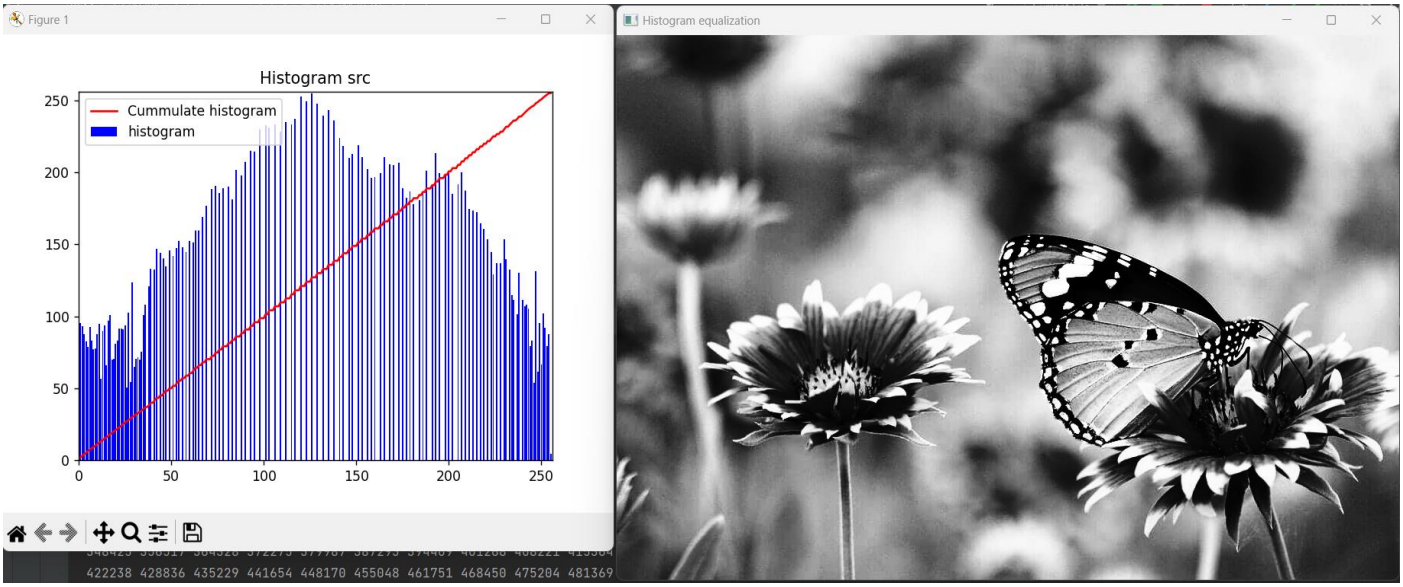
After performing color balanced: Blue (115), Green (136), Red (97)

Name Function: color_balance

Function 2: Show histogram and enter the necessary information to perform histogram equalization.



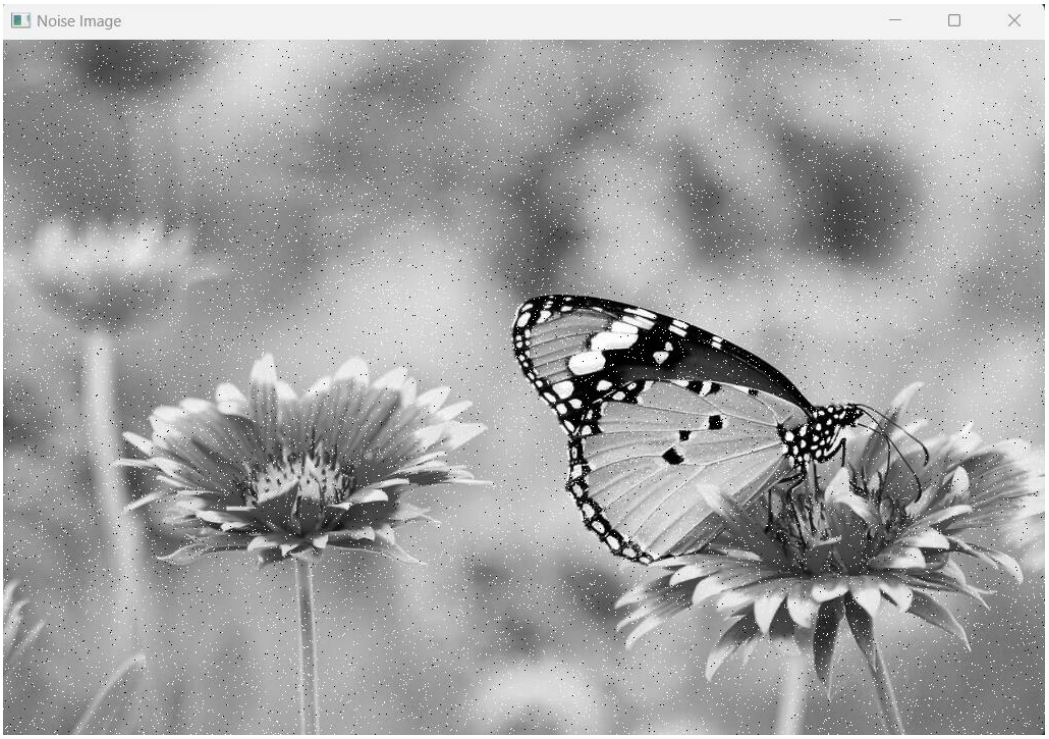
Histogram of Original Image (Figure 1 and source)



Histogram of the image after performing histogram equalization (Figure 1 and Histogram equalization windows)

Name Function: drawHistogram

histogram_equalization



Add salt and pepper noise to image.

Name Function:
add_noise

Function 3: implement the median filter to remove noise in the image(salt and pepper noise)

After implementing median filter

Name Function:
median_filter

Function 4: implement the Mean filter to remove noise in image (salt and pepper noise)

After implementing the Mean filter

Name Function:
mean_filter

Function 5: implement Gaussian smoothing to perform image smoothing.

Original Image

Gaussian Implemented

Name Function: gaussian_filter

Thank you for reviewing