Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ENCS5343 Computer Vision

Assignment # 1

Student name: Lama Naser

Student ID: 1200190

Notes:

1. 1Use this page as a cover for your home work.
2. Late home works will not be accepted (the system will not allow it).
3. Due date is December 18th, 2023 at 11:59 pm on ritaj.
4. Report including Input and Output images (Soft Copy). Include all the code you write to implement/solve this assignment with your submission.
5. Organize your output files (images) as well as used codes to be in a folder for each question (Q1, Q2, etc.). Then add the solution of all questions along with his report in one folder named Assign1. Compress the Assign1 folder and name it as (Assign1_LastName_FirstName_StudetnsID.Zip).
6. Please write some lines about how to run your code.
7. You might need to use OpenCV Python in your implementation.

Table of Contents

List of Figures

Question 1

Look for an image from the internet with the following properties: 8-bit gray-level, 256x256 pixels in size.

1- Show this image. Don't use your friends' ones.
2- Apply a power law transformation with gamma=0.4 to the image and show the image after the transformation.
3- Add a zero-mean Gaussian noise (with variance =40 gray-levels) to the original image and show the resulting image.
4- Apply a 5 by 5 mean filter to the noisy-image you obtained in point 3 above and show the result. Discuss the results in your report.
5- Add salt and pepper noise (noise-density=0.1) to the original image and then apply a 7 by 7 median filter to the noisy-image and show both images.
6- Apply a 7 by 7 mean filter to the salt and pepper noisy-image and show the result. Discuss the results in your report.
7- Apply a Sobel filter to the original image and show the response (don't use ready functions to do this part).

For this question the following figure was used with 256x256 pixels in size.

Figure 1: Question 1 input image

Part one

For this part, a power law transformation was applied with gama = 0.4 as shown in the following code.

```python
import cv2
import numpy as np

image = cv2.imread("image.jpg",cv2.IMREAD_GRAYSCALE)
gama = 0.4
newImage = np.power(image/255.0,gama) * 255.0
#convert pixel values to 8-bit unsigned integers
newImage = np.uint8(newImage)
cv2.imwrite("powered_Image.jpg",newImage)

```

Figure 2: Question 1 part 1 code

The following image was obtained after running the above code.

Figure 3: power transformed image

Since the used gama was 0.4 (less than one), the power low transformation functions acts as the log transformation function that increases the intensity value of the pixels in the original image shown in figure 1 above. This increasing of the intensity made the image more light as noticed in the resulting image in figure 3.

Part Two

A zero-mean Gaussian noise was added to the input image with variance = 40 as shown in the following code.

```python
import cv2
import numpy as np

image = cv2.imread("image.jpg",cv2.IMREAD_GRAYSCALE)
mean = 0
variance = 40
segma = np.sqrt(variance)
noisy_image = image + np.random.normal(mean,segma,image.shape)
cv2.imwrite("noisy_image.jpg",noisy_image)

```

Figure 4: Question 1 part 2 code

The following image was obtained after running the above code.

Figure 5: Zero-mean Gaussian noise image

Part Three

A 5x5 mean filter was applied to the noisy image that was obtained from the previous part in figure 5 as shown in the following code.

```
1   import cv2
2   import numpy as np
3
4   image = cv2.imread("noisy_image.jpg",cv2.IMREAD_GRAYSCALE)
5   filter = np.ones((5,5),np.float32) / 25
6   filtered_image = cv2.filter2D(image,-1,filter)
7   cv2.imwrite("filtered_image.jpg",filtered_image)
8
```

Figure 6: Question 1 part 3 code

The following image was obtained after running the above code.

Figure 7: filtered noisy image with 5x5 mean filter

The mean filter acts as a low-pass filter and has a blurring effect on the image. It averages out the high-frequency components, which include the high-frequency noise introduced by the Gaussian noise. As a result, the image became smoother, and the noise was reduced. While the mean filter helps in reducing noise, it also has a blurring effect on the image. Fine details and edges were smoothed out, and the overall image sharpness was reduced.

Part Four

For this part, a salt and pepper noise with noise density = 0.1 was added to the input image then a 7x7 median filter was applied to the noisy image obtained from the salt and pepper noise.

```python
import cv2
import numpy as np

image = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)
density = 0.1
noisy_image = image.copy()
num_of_pixels = int(density * image.size)
# Add salt noise
salt_rows, salt_cols = np.random.randint(0, image.shape[0] - 1,
                                         num_of_pixels), np.random.randint(0, image.shape[1] - 1, num_of_pixels)
noisy_image[salt_rows, salt_cols] = 255
# Add pepper noise
pepper_rows, pepper_cols = np.random.randint(0, image.shape[0] - 1,
                                             num_of_pixels), np.random.randint(0, image.shape[1] - 1, num_of_pixels)
noisy_image[pepper_rows, pepper_cols] = 0
# Apply 7x7 Median filter to the noisy image
filtered_image = cv2.medianBlur(noisy_image, 7)
cv2.imwrite("noisy_image.jpg",noisy_image)
cv2.imwrite("filtered_image.jpg", filtered_image)
```

Figure 8: Question 1 part 4 code

The following images were obtained after running the above code.

Figure 9: Salt and pepper noisy image

Figure 10: 7x7 Midean filtered image

It was noticed from the figures above that the salt and pepper noise were removed after using the Median filter since it removes the out layers (white or black).

Part Five

A 7x7 mean filter was applied to the salt and pepper noisy image that was got from the previous part as shown in the following code.

```
1   import cv2
2   import numpy as np
3
4   image = cv2.imread("noisy_image.jpg", cv2.IMREAD_GRAYSCALE)
5   filter = np.ones((7,7),np.float32) / 49
6   filtered_image = cv2.filter2D(image,-1, filter)
7   cv2.imwrite("filtered_image.jpg",filtered_image)
```

Figure 11: Question 1 part 5 code

The following filtered image was got after running the above code.

Figure 12: 7x7 Mean filtered image

Here using the Mean filter for an image that is noisy by salt and pepper noise was not optimal as in the Median filter before, since the Mean filter reduces the noise but it does not remove it.

Part Six

For this part, the input image that was shown in figure 1 was convolved with Sobel kernals as shown in the following code.

```python
import cv2
import numpy as np
#loading the image
image = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)
#do zero padding to the image
image = cv2.copyMakeBorder(image,1,1,1,1,cv2.BORDER_CONSTANT,value=0)
# Defining Sobel kernels
sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
# Get image height and width
height, width = image.shape
# Initialize output arrays
sobel_x = np.zeros((height, width), dtype=np.float32)
sobel_y = np.zeros((height, width), dtype=np.float32)

# Perform convolution with Sobel kernels
for i in range(1, height - 1):
    for j in range(1, width - 1):
        sobel_x[i-1, j-1] = np.sum(image[i - 1:i + 2, j - 1:j + 2] * sobel_kernel_x)
        sobel_y[i-1, j-1] = np.sum(image[i - 1:i + 2, j - 1:j + 2] * sobel_kernel_y)

output = np.sqrt(sobel_x ** 2 + sobel_y ** 2)
cv2.imwrite("image_sobel.jpg",output)

```

Figure 13: Question 1 part 6 code

The following image was obtained after running the above code.

Figure 14: resulted image after sobel operators applied

The edges were detected by using the Sobel operators as seen in the above figure.

Question 2

Write a function that convolves an image with a given convolution filter

function [output_Image]= myImageFilter(Input_image, filter)

Your function should output image of the same size as that of input Image (use padding). Test your function (on attached images "House1.jpg" and "House2.jpg") and show results on the following Kernels.

1- Averaging Kernel (3×3 and 5×5)
2- Gaussian Kernel (σ = 1,2,3) Use $(2\sigma+1)\times(2\sigma+1)$ as size of Kernel (You may write a separate function to generate Gaussian Kernels for different values of σ). Discuss the results in your report.
3- Sobel Edge Operators:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

4- Prewitt Edge Operators:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & 1 \end{bmatrix}$$

The two input images for this question are shown below.

Figure 15: House1.jpg

Figure 16: House2.jpg

The solution of this question was divided into four parts as following

Part one

The following code was applied for the above two images to achieve the convolution using Average filter with kernel of sizes 3 and 5.

```python
import cv2
import numpy as np

def myImageFilter(input_image,kernal_size):
    #loading the image
    image = cv2.imread(input_image, cv2.IMREAD_GRAYSCALE)
    #define the Averaging kernal
    kernal = np.ones((kernal_size,kernal_size), np.float32) / (kernal_size*kernal_size)
    # Do the convolution with the kernel using padding
    height, width = image.shape
    output = np.zeros((height, width), dtype=np.float32)
    padding = kernal_size // 2
    for i in range(padding, height - padding):
        for j in range(padding, width - padding):
            region = image[i - padding:i + padding + 1
            , j - padding:j + padding + 1]
            output[i - padding, j - padding] = np.sum(region * kernal)

    return output.astype(np.uint8)

#convolve the first image
House1_3x3 = myImageFilter("House1.jpg",3)
House1_5x5 = myImageFilter("House1.jpg",5)
#convolve the second image
House2_3x3 = myImageFilter("House2.jpg",3)
House2_5x5 = myImageFilter("House2.jpg",5)
#saving the results
```

Figure 17: Question 2 part 1 code

The following are the results of using an Average filter with size 3x3 for the input images.

Figure 18: House1 image convolved with 3x3 kernal

Figure 19: House2 image convolved with 3x3 kernal

The following are the results of using an Average filter with size 5x5 for the input images.

Figure 20: House1 image convolved with 5x5 kernal

Figure 21: House2 image convolved with 5x5 kernal

The input image "House1.jpg" is more smoothing than the second image. Due to this, using the Averaging filter with 3x3 kernal was better then using 5x5 kernal as appears from the above results that using 5x5 kernal for the first image made it blurry. However, the second input image has less smoothness than the first one, so using the Averaging filter with 5x5 kernal was better.

Part Two

The following code was used to convolve the input images with Gaussian kernel for σ = 1,2 and 3, such that the size of the kernel is (2 σ + 1)x(2σ + 1).

```python
import cv2
import numpy as np

def generate_gaussian_kernel(sigma):
    size = int(2 * sigma + 1)
    kernel = np.fromfunction(
        lambda x, y: (1 / (2 * np.pi * sigma ** 2)) *
                     np.exp(-((x - size // 2) ** 2 + (y - size // 2) ** 2) /
                            (2 * sigma ** 2)),
        (size, size)
    )
    return kernel / np.sum(kernel)

def gaussian_convolve_func(input_image, sigma):
    # Load the image
    image = cv2.imread(input_image, cv2.IMREAD_GRAYSCALE)
    # Do zero padding to the image
    image = cv2.copyMakeBorder\
        (image, sigma,sigma,sigma,sigma, cv2.BORDER_CONSTANT, value=0)
    # Generate Gaussian kernel
    gaussian_kernel = generate_gaussian_kernel(sigma)
    # Get image height and width
    height, width = image.shape
    # Initialize output array
    gaussian_output = np.zeros((height, width), dtype=np.float32)
    # Perform convolution with Gaussian kernel
    for i in range(sigma, height-sigma):
        for j in range(sigma, width-sigma):
            region = image[i - sigma: i + sigma + 1,
                           j - sigma: j + sigma + 1]
            gaussian_output[i - sigma, j - sigma] \
                = np.sum(region * gaussian_kernel)

    return gaussian_output
```

Figure 22: Question 2 part 2 code

The following are the results of using Gaussian kernel with σ = 1.

Figure 24: House1 using Gaussian kernal with σ = 1

Figure 23: Hous2 using Gaussian kernal with σ = 1

The following are the results of using Gaussian kernel with σ = 2.

Figure 26: House1 using Gaussian kernal with σ = 2

Figure 25: House2 using Gaussian kernal with σ = 2

The following are the results of using Gaussian kernel with σ = 3.

Figure 28: House1 using Gaussian kernal with σ = 2

Figure 27: House2 using Gaussian kernal with σ = 3

It was noticed that increasing σ will result in a broader and smoother Gaussian curve which means that this will increase the smoothing in the image. Also it was noticed that using Gaussian is better than using box filter since the process of blurring is uniform not leaner.

Part Three

The following code was applied to convolve the input images with Sobel edge operators using the given kernels.

```python
import cv2
import numpy as np

def concolv_Func(input_image):
    #loading the image
    image = cv2.imread(input_image, cv2.IMREAD_GRAYSCALE)
    #do zero padding to the image
    image = cv2.copyMakeBorder(image,1,1,1,1,cv2.BORDER_CONSTANT,value=0)
    # Defining Sobel kernels
    sobel_kernel_x = np.array([[-1, -2, -1],[0, 0, 0],[1, 2, 1]])
    sobel_kernel_y = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
    # Get image height and width
    height, width = image.shape
    # Initialize output arrays
    sobel_x = np.zeros((height, width), dtype=np.float32)
    sobel_y = np.zeros((height, width), dtype=np.float32)

    # Perform convolution with Sobel kernels
    for i in range(1, height - 1):
        for j in range(1, width - 1):
            sobel_x[i-1, j-1] = np.sum(image[i - 1:i + 2, j - 1:j + 2] * sobel_kernel_x)
            sobel_y[i-1, j-1] = np.sum(image[i - 1:i + 2, j - 1:j + 2] * sobel_kernel_y)

    output = np.sqrt(sobel_x ** 2 + sobel_y ** 2)
    return output
######################################################################

#convolve the first image
House1_sobel = concolv_Func("House1.jpg")
#convolve the second image
House2_sobel = concolv_Func("House2.jpg")
#saving the results
cv2.imwrite("House1_sobel.jpg",House1_sobel)
cv2.imwrite("House2_sobel.jpg",House2_sobel)

```

Figure 29: Question 2 part 3 code

The following two images got after applying the above code.

Figure 31: House1 after applying sobel operators

Figure 30: House2 after applying sobel operators

Part Four

The following code was applied to convolve the input images with Prewitt Edge Operators using the given kernels.

```python
import cv2
import numpy as np

def concolv_Func(input_image):
    #loading the image
    image = cv2.imread(input_image, cv2.IMREAD_GRAYSCALE)
    #do zero padding
    image = cv2.copyMakeBorder(image,1,1,1,1,cv2.BORDER_CONSTANT,value=0)
    # Defining Sobel kernels
    prewitt_kernel_x = np.array([[-1, 0, 1],[-1, 0, 1],[-1, 0, 1]])
    prewitt_kernel_y = np.array([[1, 1, 1],[0, 0, 0],[-1, -1, 1]])
    # Get image height and width
    height, width = image.shape
    # Initialize output arrays
    prewitt_x = np.zeros((height, width), dtype=np.float32)
    prewitt_y = np.zeros((height, width), dtype=np.float32)

    # Perform convolution with Prewitt kernels
    for i in range(1, height - 1):
        for j in range(1, width - 1):
            prewitt_x[i-1, j-1] = np.sum(image[i - 1:i + 2, j - 1:j + 2] * prewitt_kernel_x)
            prewitt_y[i-1, j-1] = np.sum(image[i - 1:i + 2, j - 1:j + 2] * prewitt_kernel_y)


    output = np.sqrt(prewitt_x ** 2 + prewitt_y ** 2)
    return output
######################################################################

#convolve the first image
House1_prewitt = concolv_Func("House1.jpg")
#convolve the second image
House2_prewitt = concolv_Func("House2.jpg")
#saving the results
cv2.imwrite("House1_prewitt.jpg",House1_prewitt)
cv2.imwrite("House2_prewitt.jpg",House2_prewitt)
```

Figure 32: Question 2 part 4 code

The following two images got after applying the above code.

Figure 33: House1 after applying prewitt

Figure 34: House1 after applying prewitt

It was noticed that using Sobel operators was better than Prewitt since Sobel include diagonal gradient in the convolution kernel. Also Sobel operators tend to be less sensitive to uniform regions than Prewitt operators.

Question 3

Attached “Noisyimage1” and “Noisyimage2” are corrupted by salt and paper noise. Apply 5 by 5 Averaging and Median filter and show your outputs. Why Median filter works better than averaging filter?

The first and second noisy images were as shown below.

Figure 35: Noisyimage1.jpg

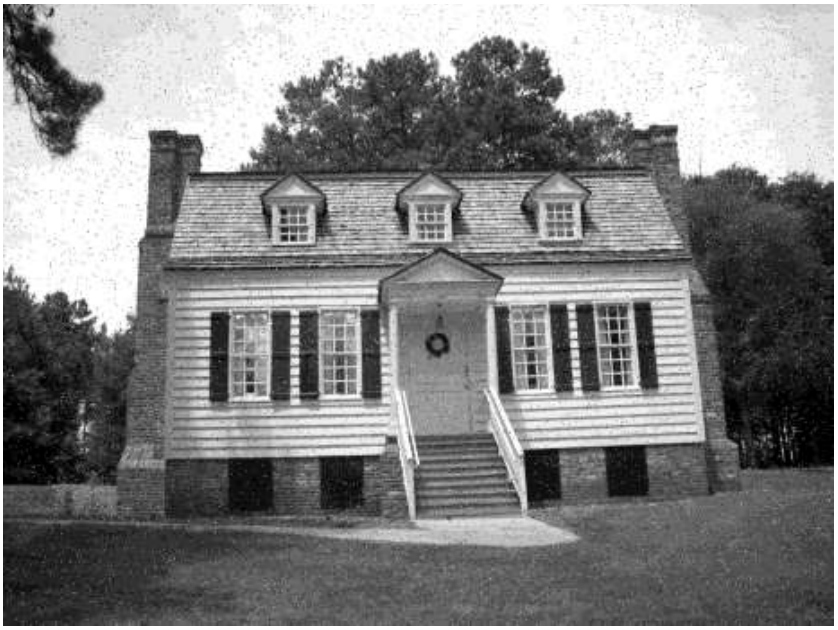

Figure 36: Noisyimage2.jpg

The following code was applied to the two input images with 5x5 Averaging and Median filters.

```python
import cv2
import numpy as np

#loading the noisy images
image1 = cv2.imread('Noisyimage1.jpg', cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread('Noisyimage2.jpg', cv2.IMREAD_GRAYSCALE)
#define a 5x5 kernel for averaging
kernel = np.ones((5, 5), np.float32) / 25
#apply averaging filter
averaged_image1 = cv2.filter2D(image1, -1, kernel)
averaged_image2 = cv2.filter2D(image2, -1, kernel)
#apply median filter
median_image1 = cv2.medianBlur(image1, 5)
median_image2 = cv2.medianBlur(image2, 5)
#save results
cv2.imwrite('averaged_image1.jpg',averaged_image1)
cv2.imwrite('averaged_image2.jpg',averaged_image2)
cv2.imwrite('median_image1.jpg',median_image1)
cv2.imwrite('median_image2.jpg',median_image2)
```

Figure 37: Question 3 code

The following two images were got from the Average filter.

Figure 38: Noisy image 1 after using 5x5 Averaging filter

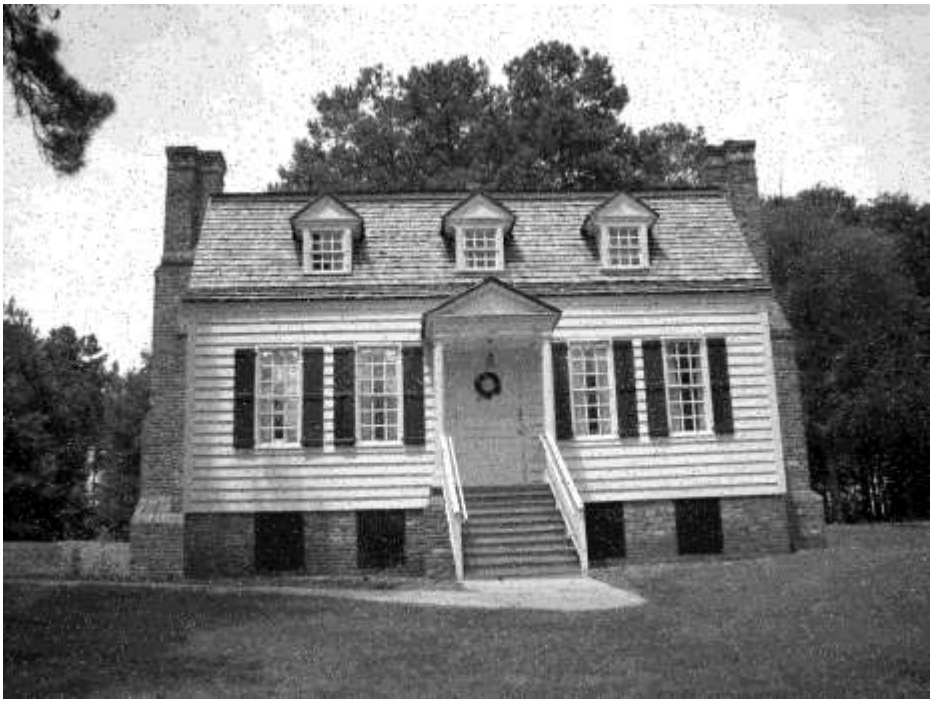

Figure 39: Noisy image 2 after using 5x5 Averaging filter

Also, the following two images were got from applying 5x5 Median filter.

Figure 40: Noisy image 1 after using 5x5 Median filter

Figure 41: Noisy image 2 after using 5x5 Median filter

As expected, the Midean filter worked better than the Average filter. This is because Median filter removes the out layer (white and black) by applying the filter on each pixel in the original image such that it orders them in ascending then it replace the value in the center with the median value. However, using the Average filter did not remove the noise but reduced it.

Question 4

Compute gradient magnitude for attached image “Q4_Image” (using built-in sobel gradients function).

1. Stretch the resulting magnitude (between 0 to 255) for better visualization

2. Compute the histogram of gradient magnitude

3. Compute gradient orientation (the angle of gradient vector)

4. Compute histogram of gradient orientation (angle between 0 and 2*pi)

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
#loading the image
image = cv2.imread('Q_4.jpg', cv2.IMREAD_GRAYSCALE)
#computing gradient magnitude using Sobel gradients
gradient_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
gradient_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
#computing gradient magnitude
gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)
gradient_magnitude_stretched = cv2.normalize(gradient_magnitude, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
#computing histogram of gradient magnitude
hist_gradient_magnitude = cv2.calcHist([gradient_magnitude.astype(np.uint8)],[0],None,[256],[0,256])
#computing gradient orientation (angle of gradient vector)
gradient_orientation = np.arctan2(gradient_y, gradient_x)
#converting angles to degrees
gradient_orientation_degrees = (np.degrees(gradient_orientation) + 360) % 360
#computing histogram of gradient orientation
hist_gradient_orientation = cv2.calcHist([gradient_orientation_degrees.astype(np.uint8)],[0],None,[256],[0,256])
plt.plot(hist_gradient_magnitude)
plt.title('Histogram of Gradient Magnitude')
plt.show()

plt.plot(hist_gradient_orientation)
plt.title('Histogram of Gradient Orientation')
plt.show()

plt.imshow(gradient_magnitude_stretched, cmap='gray')
plt.title('Stretched Gradient Magnitude')
plt.savefig('gradient_magnitude_stretched.jpg')
plt.imshow(gradient_orientation_degrees, cmap='hsv')
plt.title('Gradient Orientation (Degrees)')
plt.savefig('gradient_orientation_degrees.jpg')
```

Figure 42: Question 4 code

From the above python code, the following results were obtained.

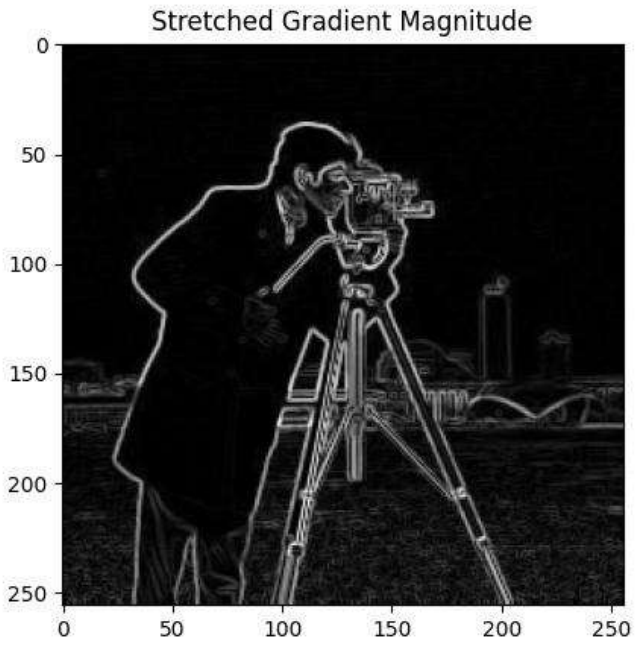

Figure 44: gradient magnitude stretched

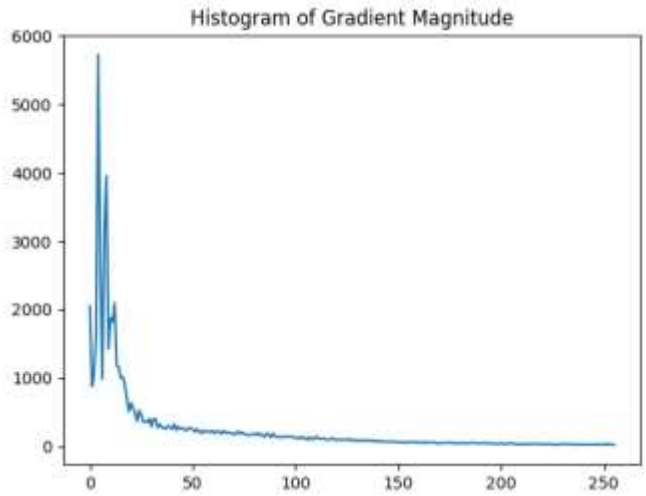

Figure 43: Histogram of gradient magnitude

As noticed from the above result for the gradient algorthim algorthim to detect the edges of the figure and the histogram showing the distribution of gradient magnitude in the image. While the below two figures show the result of using gradient orientation algorthim such that it takes the angle of the gradient vector using arctangent function. Thus the pixels value did not exceed 90 degrees.

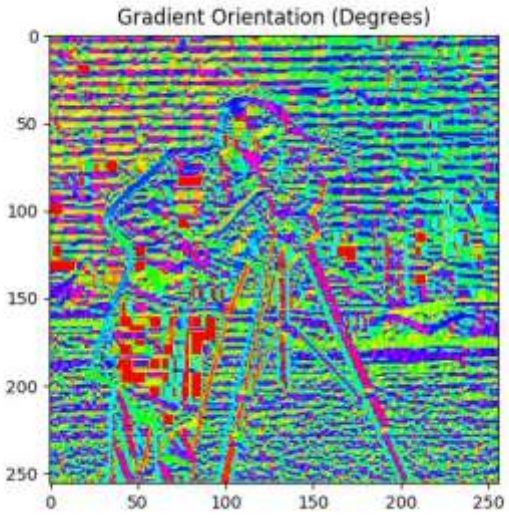

Figure 45: gradient orientation degrees

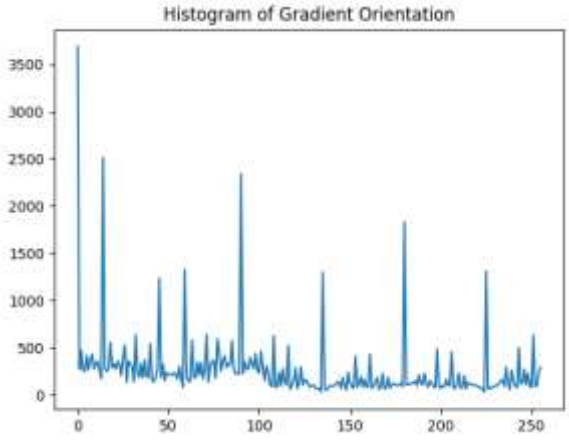

Figure 46: Histogram of gradient orientation degrees

Question 5

Load walk_1.jpg and walk_2.jpg images in Python. Convert them to gray scale and subtract walk_2.jpg from walk_1.jpg. What is the result? Why?

The input images are shown below.

Figure 47: walk_1.jpg

Figure 48: walk_2.jpg

First, the two above images were converted to gray scal. Then the image “walk_2.jpg” was subtracted from the image “walk_1.jpg” as showing in the python code below.

```
1   import cv2
2   #loading the images
3   image1 = cv2.imread('walk_1.jpg')
4   image2 = cv2.imread('walk_2.jpg')
5   #convert them to grey scale
6   image1_grey = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
7   image2_grey = cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)
8   #subtract image 2 from image 1
9   image_Q5 = image1_grey - image2_grey
10  #show the resul
11  cv2.imshow('Image', image_Q5)
12  cv2.waitKey(0)
13  cv2.destroyAllWindows()
14  #save the result image
15  cv2.imwrite("image_Q4.jpg",image_Q5)
16
```

Figure 49: Question 5 code

The result of the subtraction operator was found as in the figure below.

Figure 50: Result image, question 5

It was noticed that the resulting image in figure 50 holds the details in both of the first image shown in figure 47 and the image shown in figure 48. Such that when the intensity value of a pixel in the first image is close to the intensity value of the corresponding pixel in the second image, then the new value in the output image will be close to zero which means that it will have a low intensity value. Also, when the intensity value of the pixel in the first image is much larger than the corrsponding one in the second image, then the details in the first image will appear like the people walking on the right corner of the first image. The last case is that when the intensity value of a pixel in the first image is much less than the one in the second image, then the details appears in the second image will be seen.

Question 6

Apply canny edge detector on the "Q_4.jpg" using OpenCV function "Canny". Test different values of 'Threshold'.

The same input image that was used in question 4 was again used for this part as showing below.

Figure 51: Q_4.jpg

In the following code, 'cv2.Canny' function was applied to the input images of threshold value 10, 100 and 255. Then the edges detected by the Canny algorithm were obtained as showing in the python code below.

```
1   import cv2
2   #loading the image
3   image = cv2.imread('Q_4.jpg',cv2.IMREAD_GRAYSCALE)
4   #apply canny edge detection for diffrent threshoul valu
5   edge1 = cv2.Canny(image,10,50*2)
6   edge2 = cv2.Canny(image,100,100*2)
7   edge3 = cv2.Canny(image,255,255)
8   #save the results
9   cv2.imwrite('edge1.jpg',edge1)
10  cv2.imwrite('edge2.jpg',edge2)
11  cv2.imwrite('edge3.jpg',edge3)
12  cv2.waitKey(0)
13  cv2.destroyAllWindows()
```

Figure 52: Question 6 code

As a result of the above code, the following results were found.

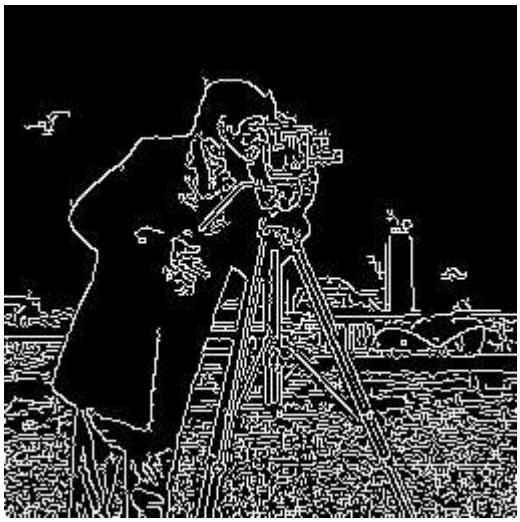

Figure 53: Question 6 result with threshold value = 10

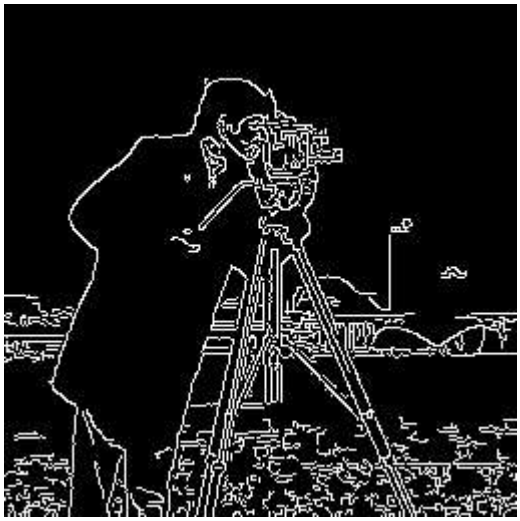

Figure 54: Question 6 result with threshold value = 100

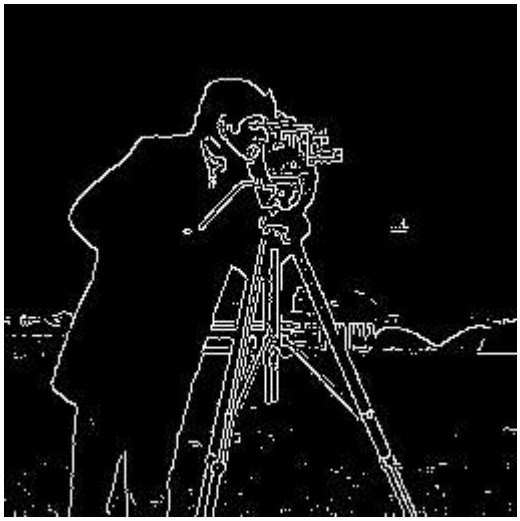

Figure 55: Question 6 result with threshold value = 255

Canny algorthim follows five steps for edge swrwction, first, it do smoothing on the original image to get rid of the weak edges. Second, it computes Gx and Gy using Sobel operators to detect the horizontal and vertical edges. Third, it reduces the angle of gradient to one of the four sectors, such that Any edge direction falling within the yellow range (0 to 22.5 & 157.5 to 180 degrees) is set to 0 degrees. Any edge direction falling in the green range (22.5 to 67.5 degrees) is set to 45 degrees. Any edge direction falling in the blue range (67.5 to 112.5 degrees) is set to 90 degrees. And finally, any edge direction falling within the red range (112.5 to 157.5 degrees) is set to 135 degrees. Fourth, it slims the edges. The final step is hysteresis thresholding based on two threshold values (low value and high value).

It was noticed that using the Canny function with a threshold value equals 255 gave an edge detection with broken edges since it used a high threshold value. However, using the function with threshold value equals 10 gave a bad detection since it shows the small details (weak edges) leading to a kind of noise on the object's edges. As a compromise, using the function with threshould value equals 100 gave a good detection.