Project #6

DIGITAL IMAGE PROCESSING

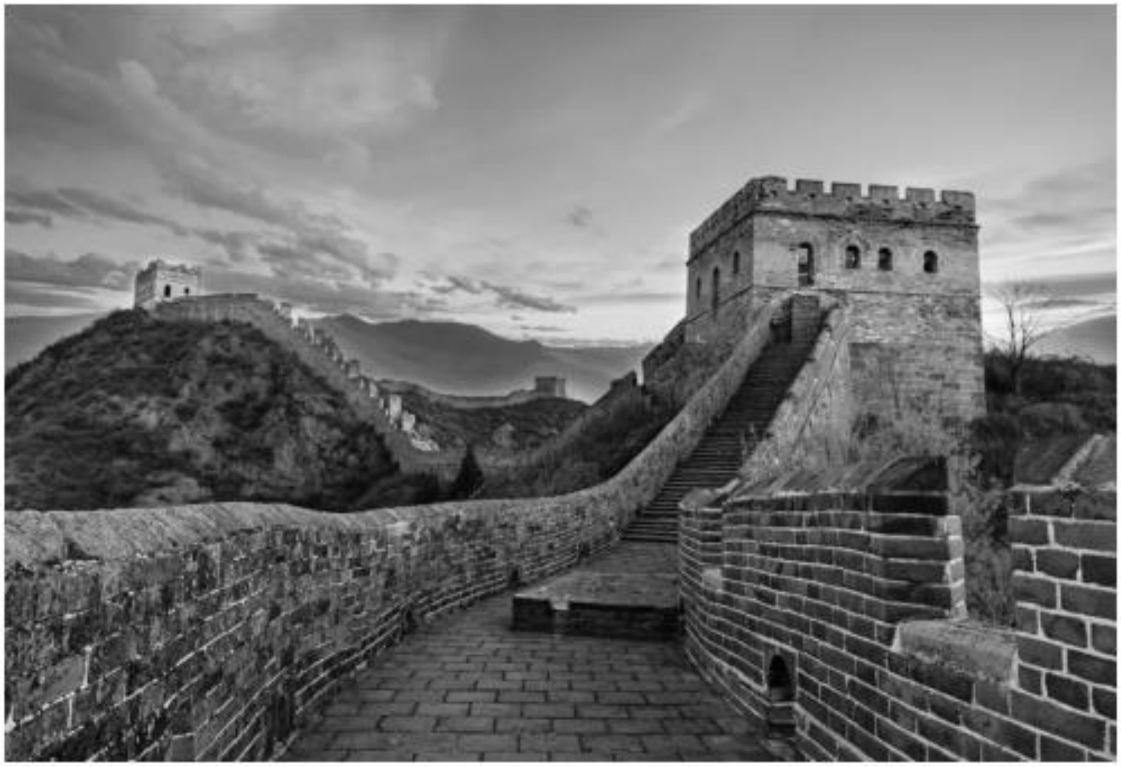
Student: 黎文雄 (Le Van Hung)

Student ID: 0860831

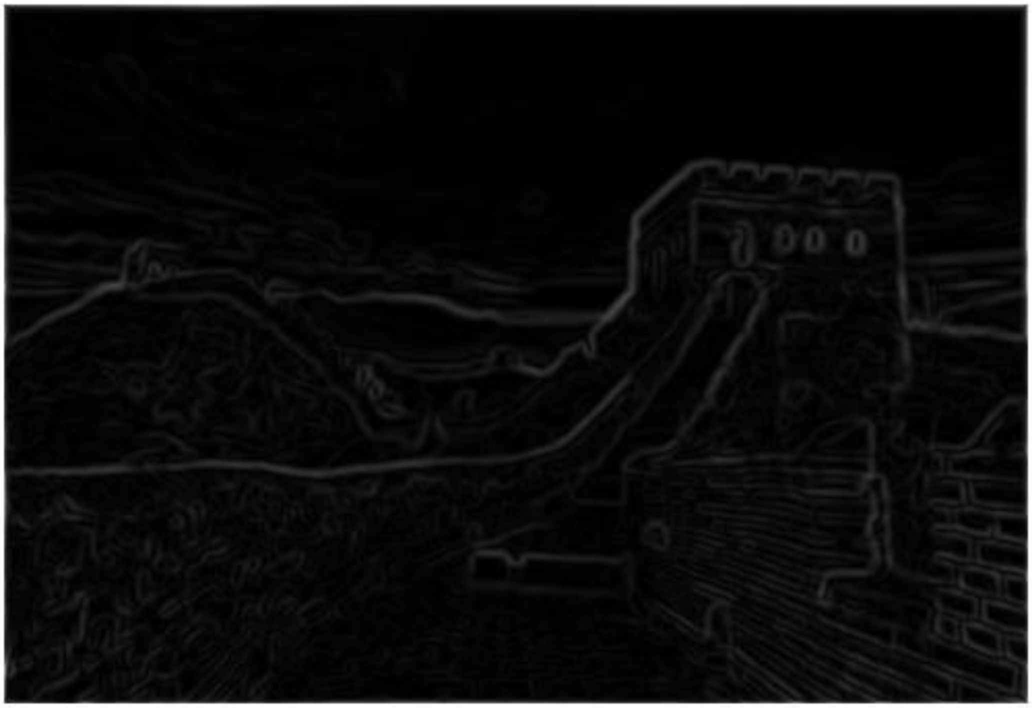
Requirement:

- Apply Canny edge detection algorithm to obtain the edge image by using the following setup and parameters:
- *My report contains:*
- Source codes
- Figures of H, S and I component images
- Figure of color – slicing image using a1
- Figure of color – slicing image using a2

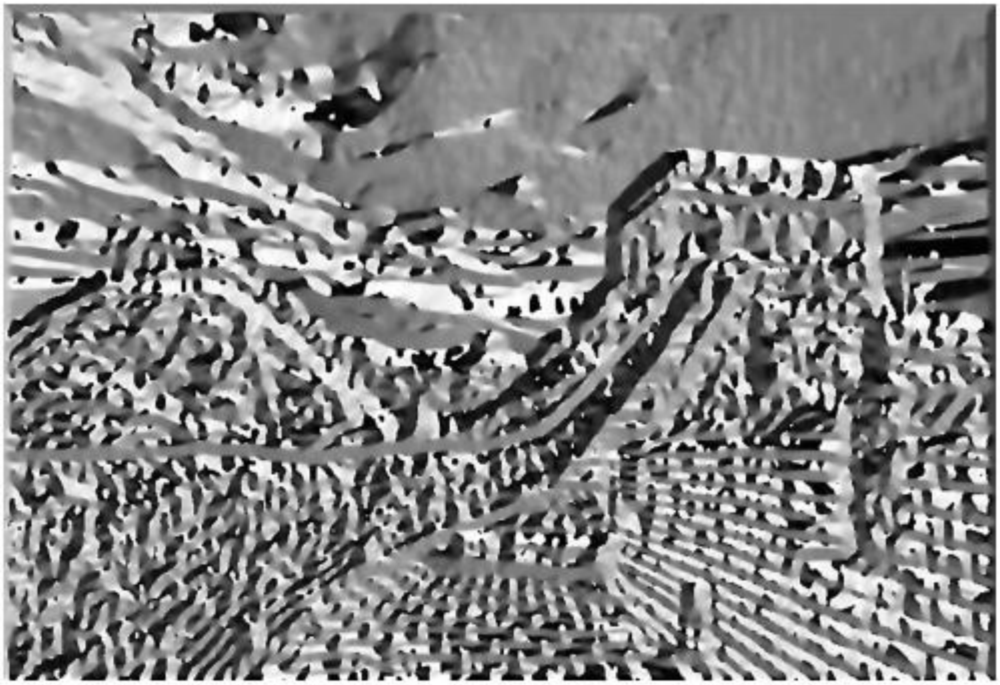
Solution



Original Image



Gradient magnitude image



Gradient angle image

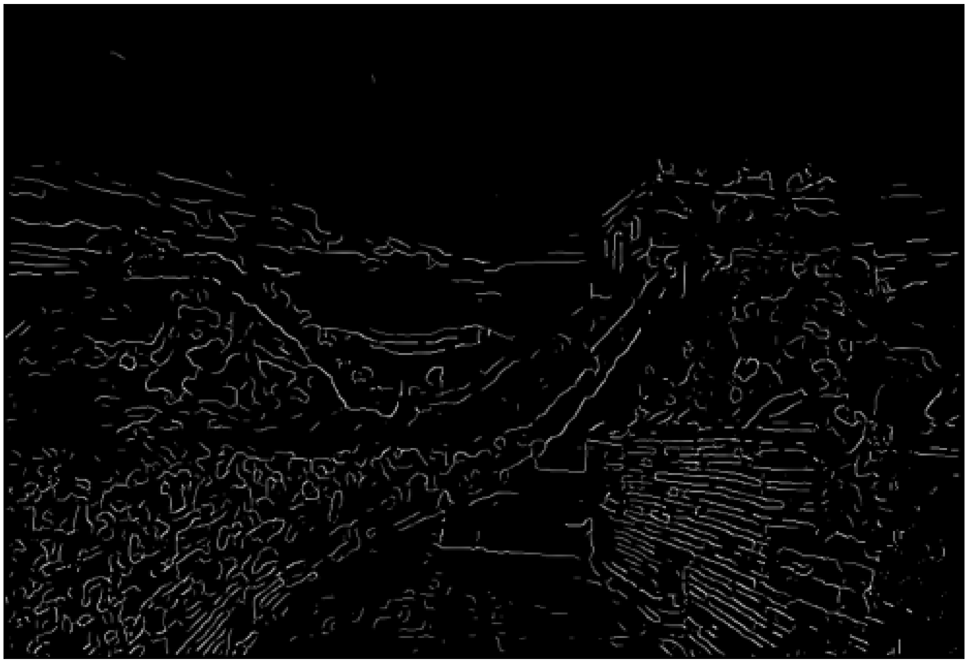


Figure of gNL(x,y)

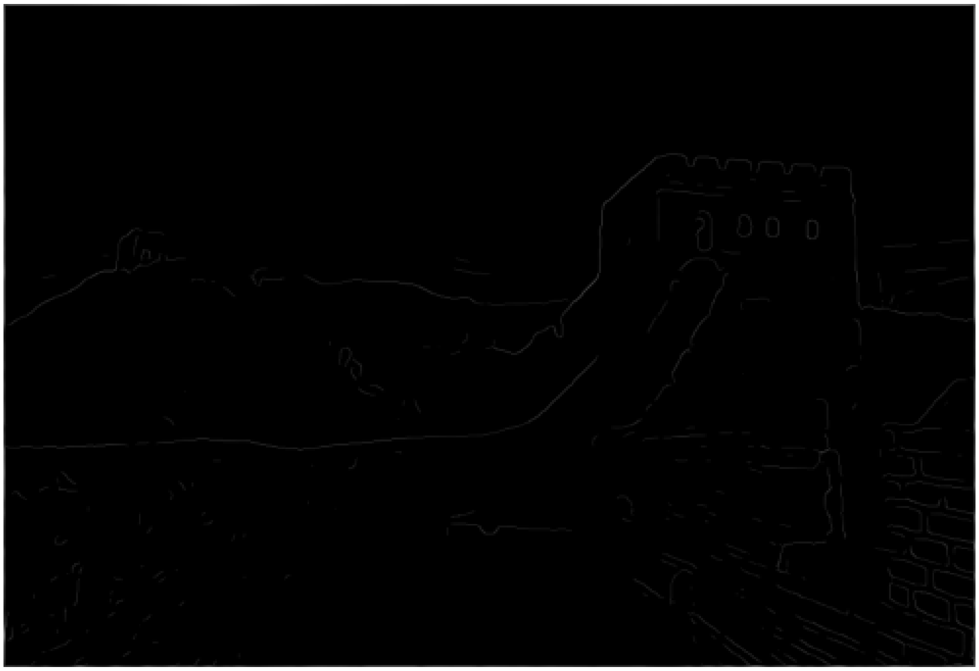


Figure of gNH(x,y)

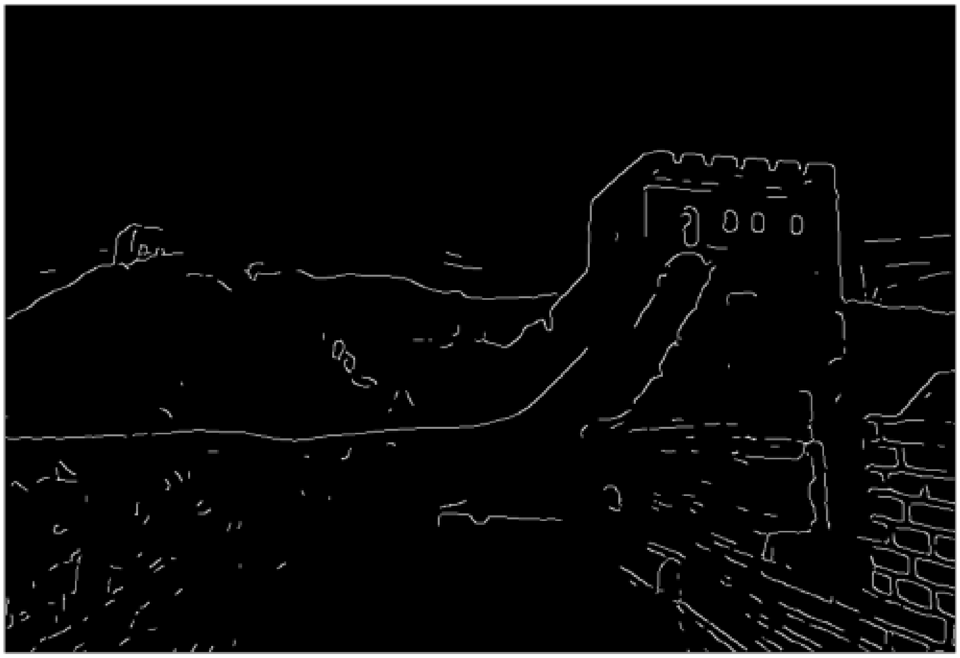
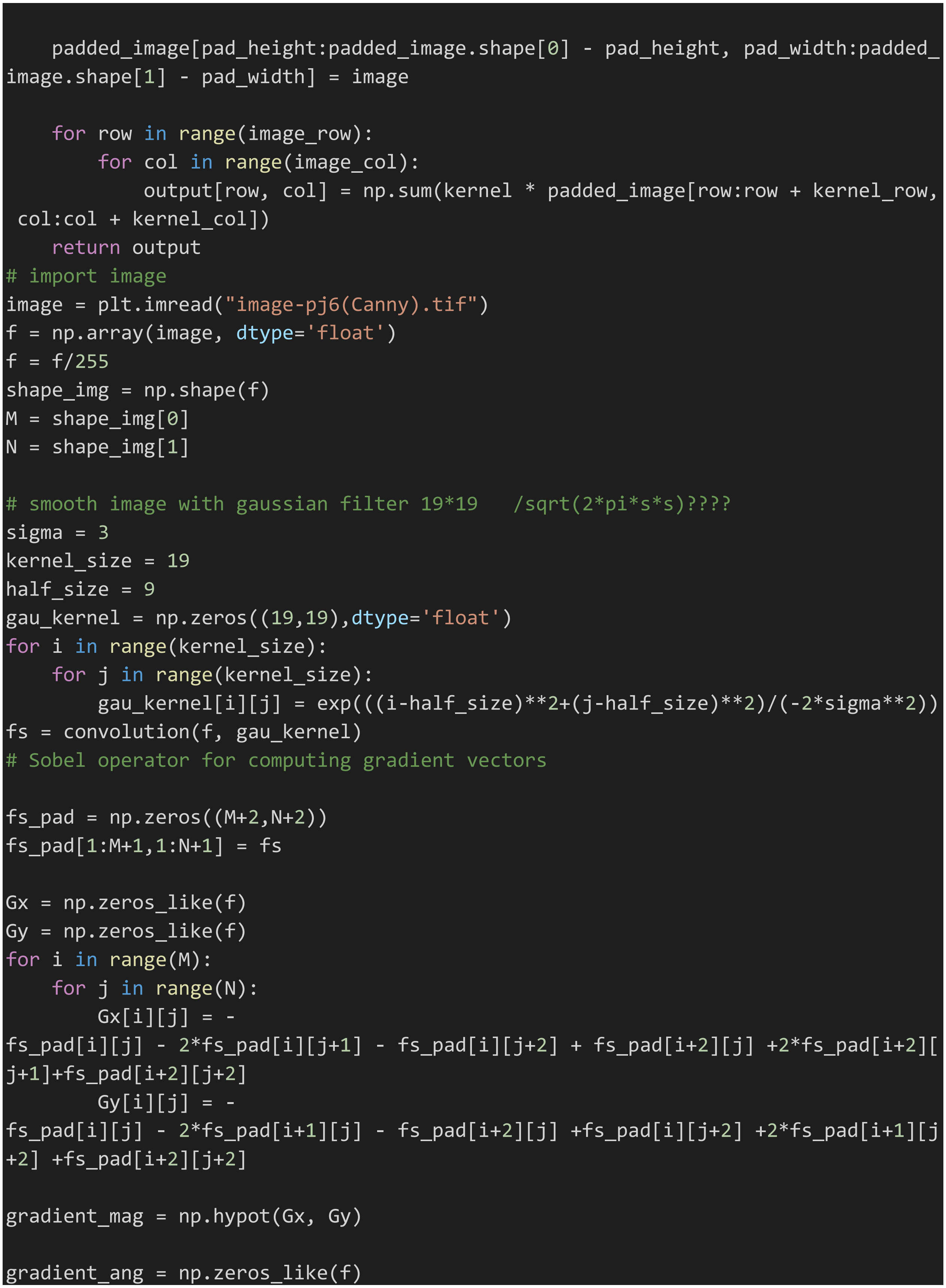


Figure of Final edge map

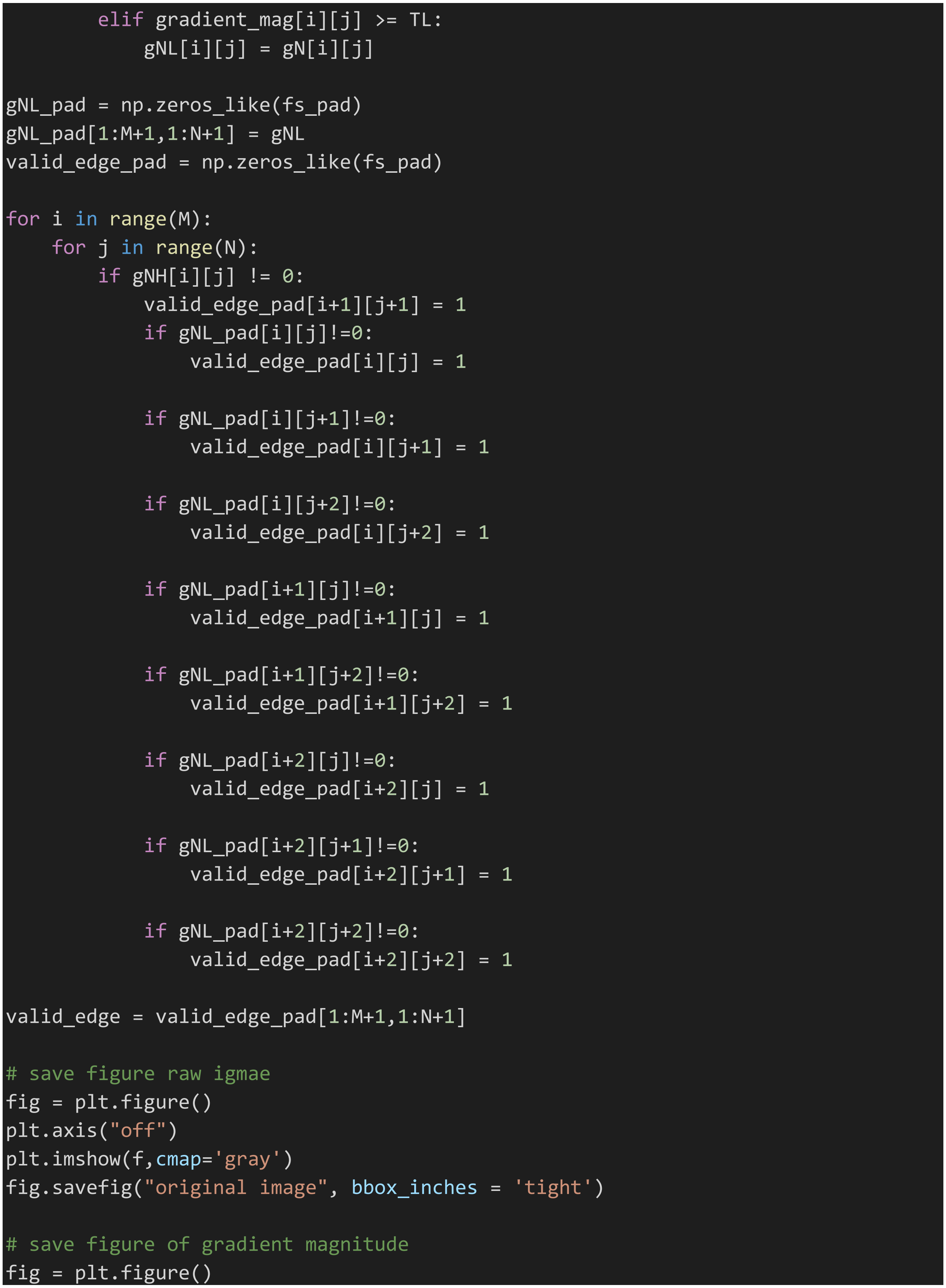
Source Code (python)

```
# import library
#%%
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from scipy import ndimage
from numpy import pi, exp, sqrt

def convolution(image, kernel):
    image_row, image_col = image.shape
    kernel_row, kernel_col = kernel.shape

    output = np.zeros(image.shape)

    pad_height = int((kernel_row - 1) / 2)
    pad_width = int((kernel_col - 1) / 2)

    padded_image = np.zeros((image_row + (2 * pad_height), image_col + (2 * pad_w
idth)))
```



```
    padded_image[pad_height:padded_image.shape[0] - pad_height, pad_width:padded_
image.shape[1] - pad_width] = image

    for row in range(image_row):
        for col in range(image_col):
            output[row, col] = np.sum(kernel * padded_image[row:row + kernel_row,
 col:col + kernel_col])
    return output
# import image
image = plt.imread("image-pj6(Canny).tif")
f = np.array(image, dtype='float')
f = f/255
shape_img = np.shape(f)
M = shape_img[0]
N = shape_img[1]

# smooth image with gaussian filter 19*19   /sqrt(2*pi*s*s)????
sigma = 3
kernel_size = 19
half_size = 9
gau_kernel = np.zeros((19,19),dtype='float')
for i in range(kernel_size):
    for j in range(kernel_size):
        gau_kernel[i][j] = exp(((i-half_size)**2+(j-half_size)**2)/(-2*sigma**2))
fs = convolution(f, gau_kernel)
# Sobel operator for computing gradient vectors

fs_pad = np.zeros((M+2,N+2))
fs_pad[1:M+1,1:N+1] = fs

Gx = np.zeros_like(f)
Gy = np.zeros_like(f)
for i in range(M):
    for j in range(N):
        Gx[i][j] = -
fs_pad[i][j] - 2*fs_pad[i][j+1] - fs_pad[i][j+2] + fs_pad[i+2][j] +2*fs_pad[i+2][
j+1]+fs_pad[i+2][j+2]
        Gy[i][j] = -
fs_pad[i][j] - 2*fs_pad[i+1][j] - fs_pad[i+2][j] +fs_pad[i][j+2] +2*fs_pad[i+1][j
+2] +fs_pad[i+2][j+2]

gradient_mag = np.hypot(Gx, Gy)

gradient_ang = np.zeros_like(f)
```

```
for i in range(M):
    for j in range(N):
        gradient_ang[i][j] = np.arctan(Gy[i][j]/Gx[i][j])

gradient_ang_plot = np.arctan2(Gy, Gx)
# Calculate gN

gN = np.zeros_like(f)

gradient_mag_pad = np.zeros_like(fs_pad)
gradient_mag_pad[1:M+1,1:N+1] = gradient_mag

for i in range(M):
    for j in range(N):
        K = gradient_mag_pad[i+1][j+1]
        if (gradient_ang[i][j] > -
22.5*pi/180 and gradient_ang[i][j] < 22.5*pi/180): #d1
            if K>gradient_mag_pad[i][j+1] and K>gradient_mag_pad[i+2][j+1]:
                gN[i][j] = K
        elif (gradient_ang[i][j] > 22.5*pi/180 and gradient_ang[i][j] < 67.5*pi/1
80): #d2
            if K>gradient_mag_pad[i][j] and K>gradient_mag_pad[i+2][j+2]:
                gN[i][j] = K
        elif (gradient_ang[i][j] < -22.5*pi/180 and gradient_ang[i][j] > -
67.5*pi/180): #d4
            if K>gradient_mag_pad[i+2][j] and K>gradient_mag_pad[i][j+2]:
                gN[i][j] = K
        else: #d3
            if K>gradient_mag_pad[i+1][j] and K>gradient_mag_pad[i+1][j+2]:
                gN[i][j] = K


# hysteretic thresholding
max_gradient_mag = np.amax(gradient_mag)

TH = 0.1*max_gradient_mag
TL = 0.04*max_gradient_mag

gNH = np.zeros_like(f)
gNL = np.zeros_like(f)

for i in range(M):
    for j in range(N):
        if gradient_mag[i][j] >= TH:
            gNH[i][j] = gN[i][j]
```



```
        elif gradient_mag[i][j] >= TL:
            gNL[i][j] = gN[i][j]

gNL_pad = np.zeros_like(fs_pad)
gNL_pad[1:M+1,1:N+1] = gNL
valid_edge_pad = np.zeros_like(fs_pad)

for i in range(M):
    for j in range(N):
        if gNH[i][j] != 0:
            valid_edge_pad[i+1][j+1] = 1
            if gNL_pad[i][j]!=0:
                valid_edge_pad[i][j] = 1

            if gNL_pad[i][j+1]!=0:
                valid_edge_pad[i][j+1] = 1

            if gNL_pad[i][j+2]!=0:
                valid_edge_pad[i][j+2] = 1

            if gNL_pad[i+1][j]!=0:
                valid_edge_pad[i+1][j] = 1

            if gNL_pad[i+1][j+2]!=0:
                valid_edge_pad[i+1][j+2] = 1

            if gNL_pad[i+2][j]!=0:
                valid_edge_pad[i+2][j] = 1

            if gNL_pad[i+2][j+1]!=0:
                valid_edge_pad[i+2][j+1] = 1

            if gNL_pad[i+2][j+2]!=0:
                valid_edge_pad[i+2][j+2] = 1

valid_edge = valid_edge_pad[1:M+1,1:N+1]

# save figure raw igmae
fig = plt.figure()
plt.axis("off")
plt.imshow(f,cmap='gray')
fig.savefig("original image", bbox_inches = 'tight')

# save figure of gradient magnitude
fig = plt.figure()
```

```
plt.axis("off")
plt.imshow(gradient_mag,cmap='gray')
fig.savefig("gradient_mag", bbox_inches = 'tight')


# save figure of gradient angle
fig = plt.figure()
plt.axis("off")
plt.imshow(gradient_ang_plot,cmap='gray')
fig.savefig("gradient_ang", bbox_inches = 'tight')

# save figure of gNH
fig = plt.figure()
plt.axis("off")
plt.imshow(gNH,cmap='gray')
fig.savefig("gNH", bbox_inches = 'tight')

# save figure of gNH
fig = plt.figure()
plt.axis("off")
plt.imshow(gNL,cmap='gray')
fig.savefig("gNL", bbox_inches = 'tight')

# save figure of valid edge
fig = plt.figure()
plt.axis("off")
plt.imshow(valid_edge,cmap='gray')
fig.savefig("valid_egde", bbox_inches = 'tight')
```