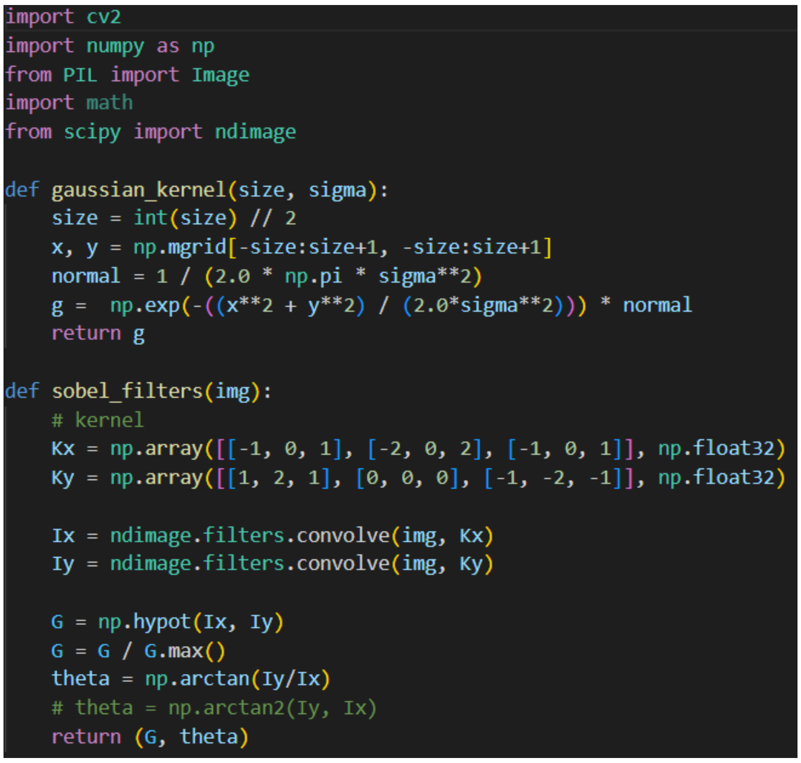
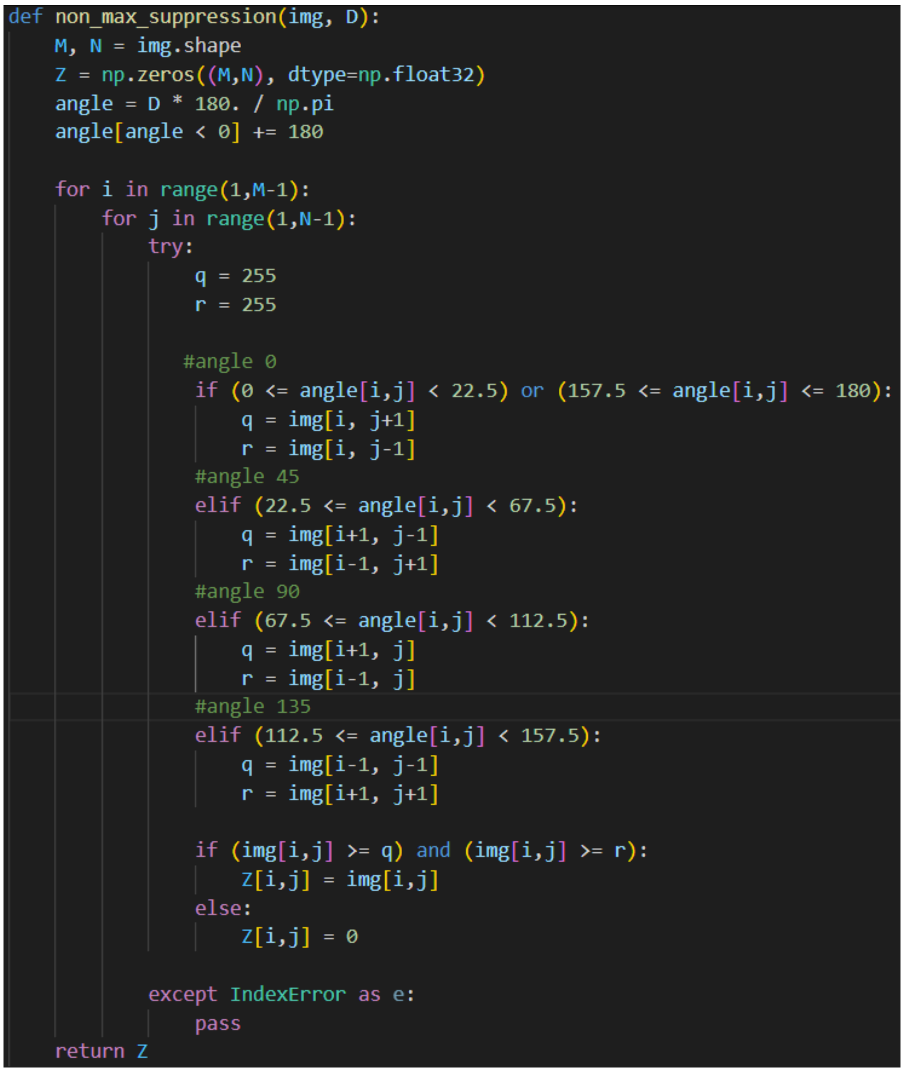
Project 5

ID：311512015　　name：謝元碩

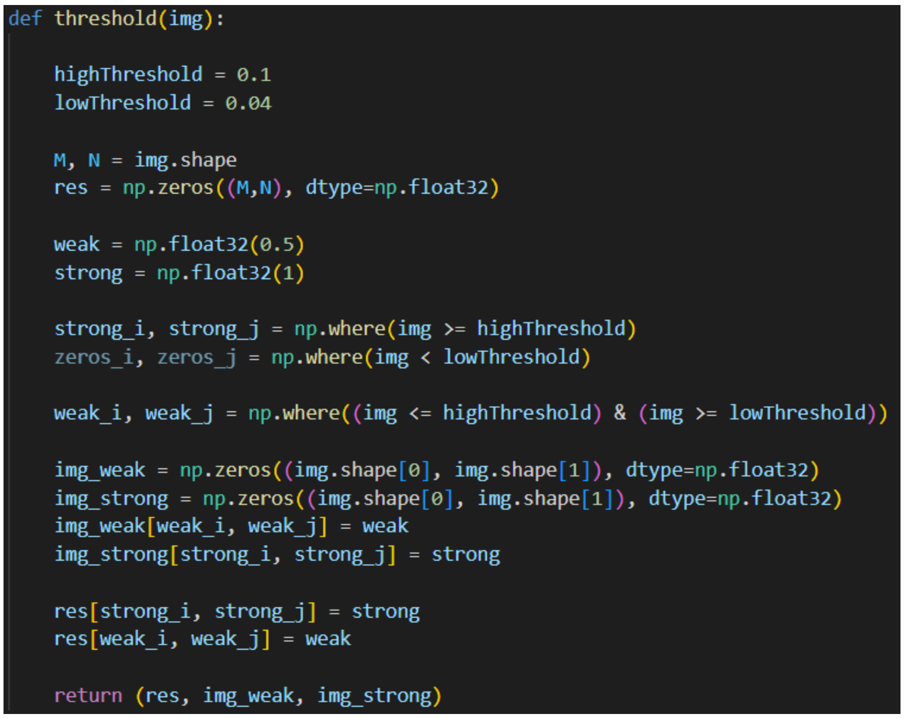
(a) Source codes:



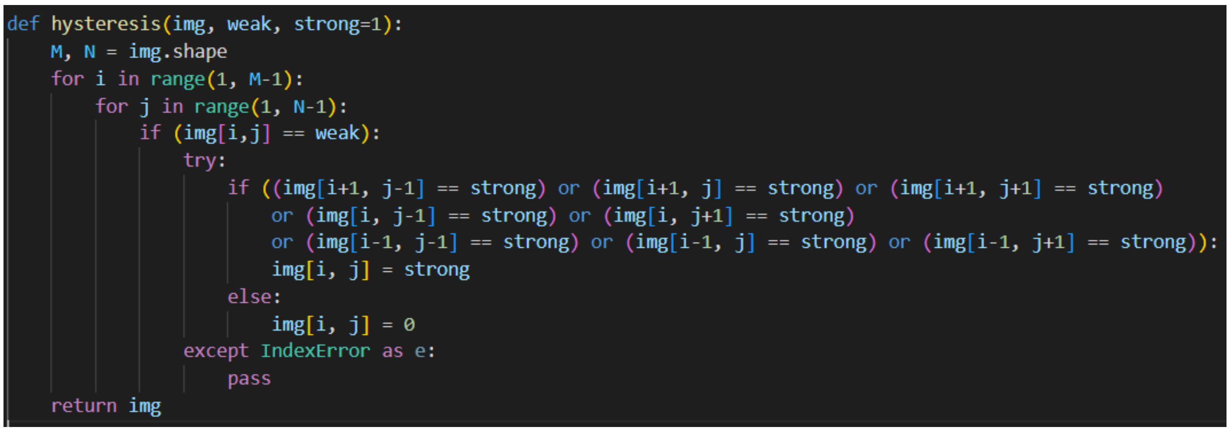
```python
import cv2
import numpy as np
from PIL import Image
import math
from scipy import ndimage

def gaussian_kernel(size, sigma):
    size = int(size) // 2
    x, y = np.mgrid[-size:size+1, -size:size+1]
    normal = 1 / (2.0 * np.pi * sigma**2)
    g =  np.exp(-((x**2 + y**2) / (2.0*sigma**2))) * normal
    return g

def sobel_filters(img):
    # kernel
    Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
    Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)

    Ix = ndimage.filters.convolve(img, Kx)
    Iy = ndimage.filters.convolve(img, Ky)

    G = np.hypot(Ix, Iy)
    G = G / G.max()
    theta = np.arctan(Iy/Ix)
    # theta = np.arctan2(Iy, Ix)
    return (G, theta)
```



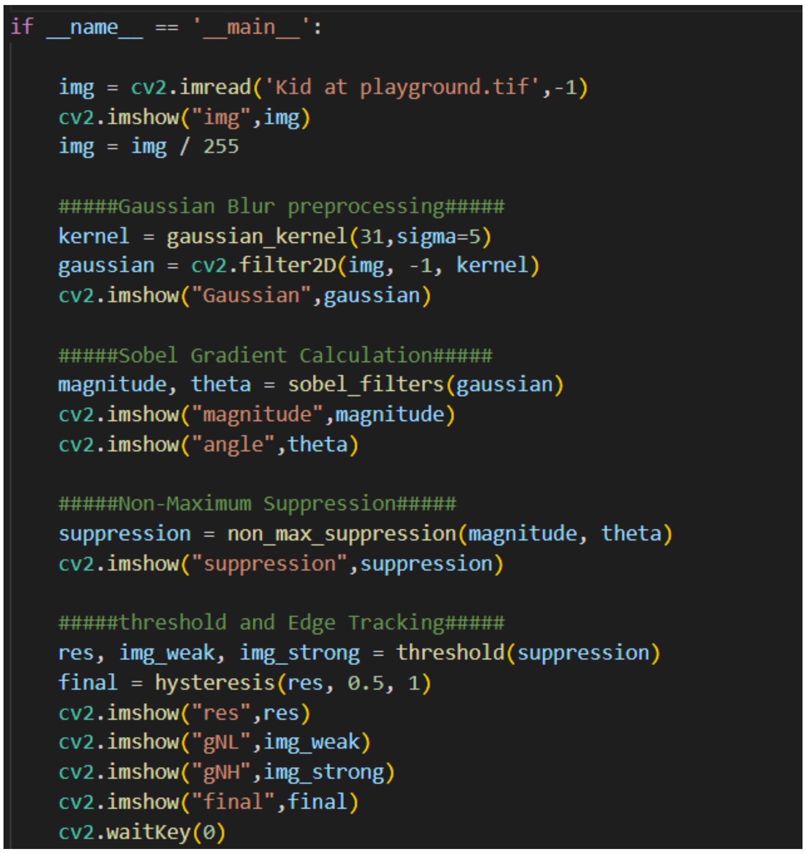
```python
def non_max_suppression(img, D):
    M, N = img.shape
    Z = np.zeros((M,N), dtype=np.float32)
    angle = D * 180. / np.pi
    angle[angle < 0] += 180

    for i in range(1,M-1):
        for j in range(1,N-1):
            try:
                q = 255
                r = 255

               #angle 0
                if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                    q = img[i, j+1]
                    r = img[i, j-1]
                #angle 45
                elif (22.5 <= angle[i,j] < 67.5):
                    q = img[i+1, j-1]
                    r = img[i-1, j+1]
                #angle 90
                elif (67.5 <= angle[i,j] < 112.5):
                    q = img[i+1, j]
                    r = img[i-1, j]
                #angle 135
                elif (112.5 <= angle[i,j] < 157.5):
                    q = img[i-1, j-1]
                    r = img[i+1, j+1]

                if (img[i,j] >= q) and (img[i,j] >= r):
                    Z[i,j] = img[i,j]
                else:
                    Z[i,j] = 0

            except IndexError as e:
                pass
    return Z
```



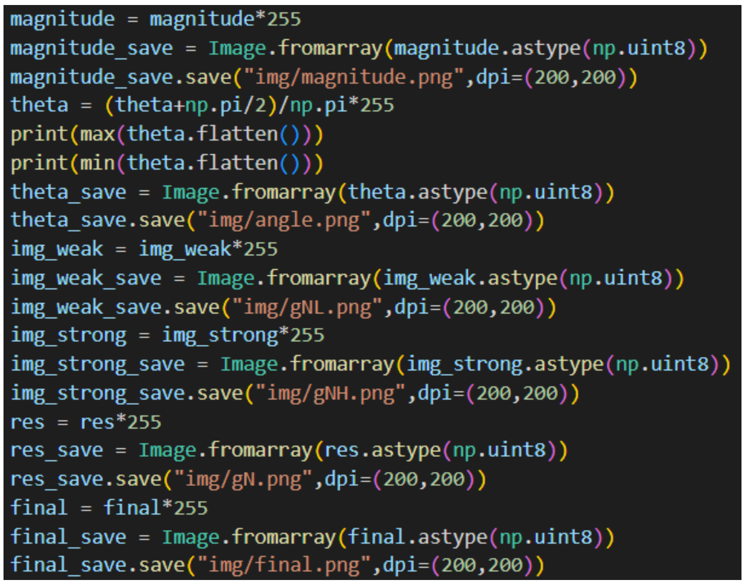
```python
def threshold(img):

    highThreshold = 0.1
    lowThreshold = 0.04

    M, N = img.shape
    res = np.zeros((M,N), dtype=np.float32)

    weak = np.float32(0.5)
    strong = np.float32(1)

    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)

    weak_i, weak_j = np.where((img <= highThreshold) & (img >= lowThreshold))

    img_weak = np.zeros((img.shape[0], img.shape[1]), dtype=np.float32)
    img_strong = np.zeros((img.shape[0], img.shape[1]), dtype=np.float32)
    img_weak[weak_i, weak_j] = weak
    img_strong[strong_i, strong_j] = strong

    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak

    return (res, img_weak, img_strong)
```

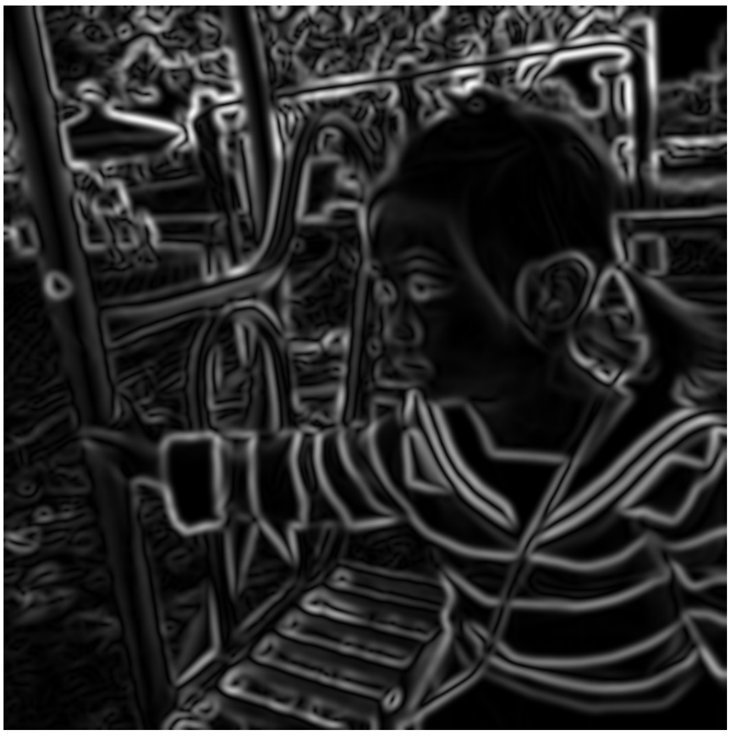
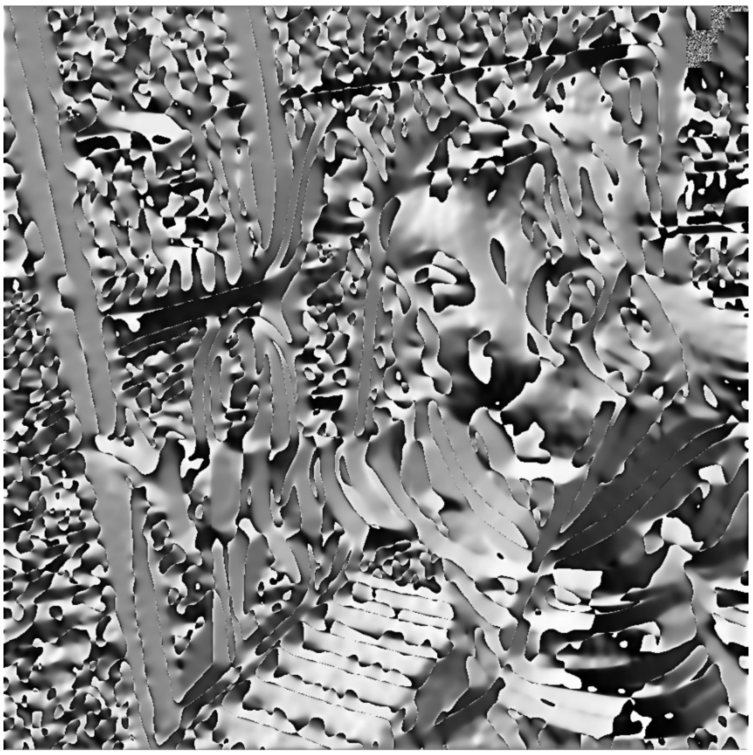


```python
def hysteresis(img, weak, strong=1):
    M, N = img.shape
    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1, j+1] == strong)
                        or (img[i, j-1] == strong) or (img[i, j+1] == strong)
                        or (img[i-1, j-1] == strong) or (img[i-1, j] == strong) or (img[i-1, j+1] == strong)):
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img
```

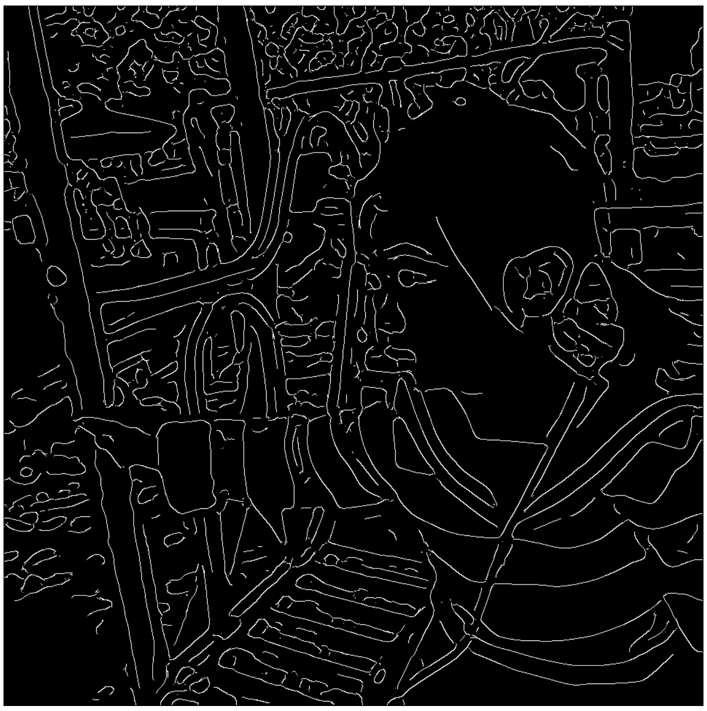
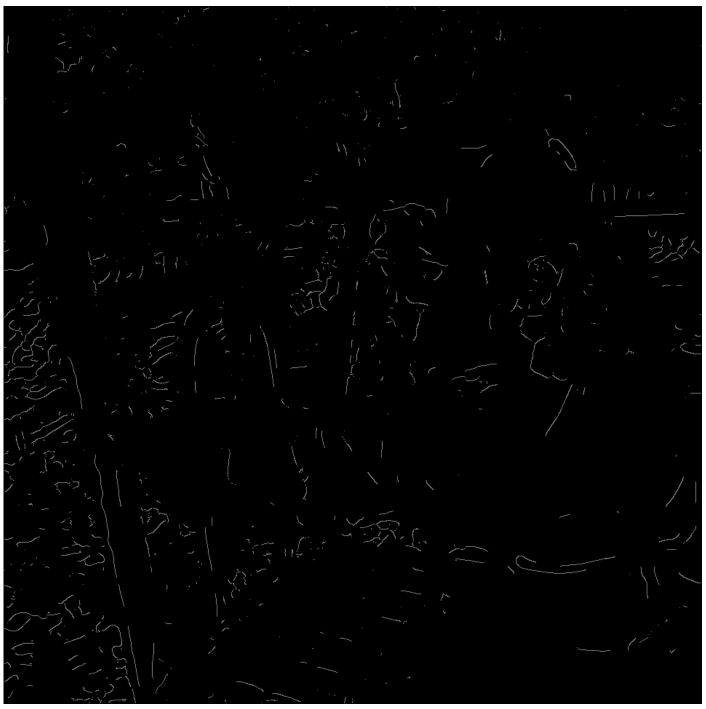
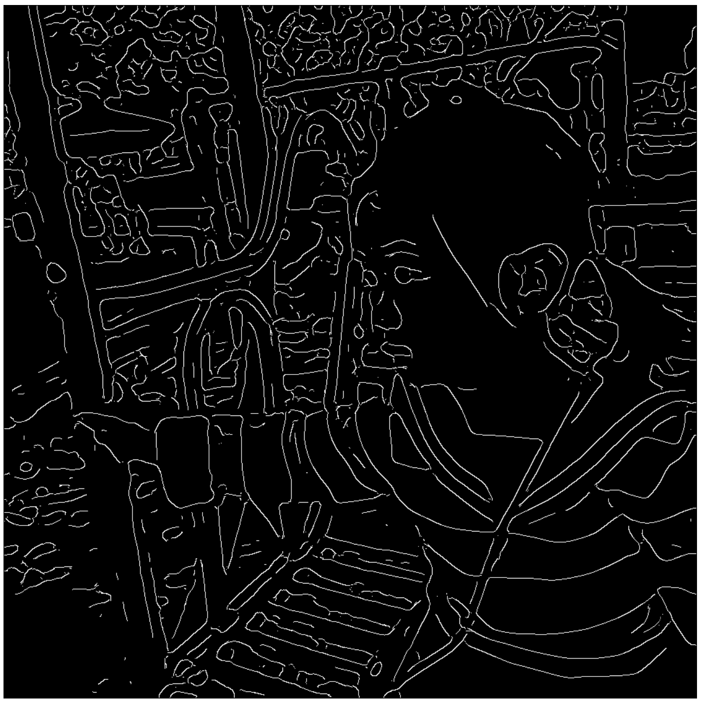


```python
if __name__ == '__main__':

    img = cv2.imread('Kid at playground.tif',-1)
    cv2.imshow("img",img)
    img = img / 255

    #####Gaussian Blur preprocessing#####
    kernel = gaussian_kernel(31,sigma=5)
    gaussian = cv2.filter2D(img, -1, kernel)
    cv2.imshow("Gaussian",gaussian)

    #####Sobel Gradient Calculation#####
    magnitude, theta = sobel_filters(gaussian)
    cv2.imshow("magnitude",magnitude)
    cv2.imshow("angle",theta)

    #####Non-Maximum Suppression#####
    suppression = non_max_suppression(magnitude, theta)
    cv2.imshow("suppression",suppression)

    #####threshold and Edge Tracking#####
    res, img_weak, img_strong = threshold(suppression)
    final = hysteresis(res, 0.5, 1)
    cv2.imshow("res",res)
    cv2.imshow("gNL",img_weak)
    cv2.imshow("gNH",img_strong)
    cv2.imshow("final",final)
    cv2.waitKey(0)
```


```
magnitude = magnitude*255
magnitude_save = Image.fromarray(magnitude.astype(np.uint8))
magnitude_save.save("img/magnitude.png",dpi=(200,200))
theta = (theta+np.pi/2)/np.pi*255
print(max(theta.flatten()))
print(min(theta.flatten()))
theta_save = Image.fromarray(theta.astype(np.uint8))
theta_save.save("img/angle.png",dpi=(200,200))
img_weak = img_weak*255
img_weak_save = Image.fromarray(img_weak.astype(np.uint8))
img_weak_save.save("img/gNL.png",dpi=(200,200))
img_strong = img_strong*255
img_strong_save = Image.fromarray(img_strong.astype(np.uint8))
img_strong_save.save("img/gNH.png",dpi=(200,200))
res = res*255
res_save = Image.fromarray(res.astype(np.uint8))
res_save.save("img/gN.png",dpi=(200,200))
final = final*255
final_save = Image.fromarray(final.astype(np.uint8))
final_save.save("img/final.png",dpi=(200,200))
```
(b) Plot images of the gradient magnitude and gradient angle:

magnitude	Angle
	

(c) Plot nonmaxima suppressed image $g_N(x,y)$ as well as images of $g_{NL}(x,y)$ and $g_{NH}(x,y)$:

$g_N(x,y)$ 	$g_{NL}(x,y)$ 
$g_{NH}(x,y)$ 	

(d) Plot final edge map e(x,y):