班級：資工三
學號：110590034
姓名：楊榮鈞

```python
'''
Modules import
'''
import numpy as np
import cv2

def rgb_to_gray(image):
    '''
    Convert color image to grayscale image
    '''
    gray_image = np.dot(image[..., :3], [0.3, 0.59, 0.11])
    return gray_image.astype(np.float64)

def get_n_kernel(image, row, col, kernel_size):
    '''
    Get n kernel
    '''
    height = image.shape[0]
    width = image.shape[1]
    kernel = np.zeros((kernel_size, kernel_size)).astype(np.float64)
    for i in range(-kernel_size//2, kernel_size//2+1):
        for j in range(-kernel_size//2, kernel_size//2+1):
            # if 0 <= row+i < height and 0 <= col+j < width:
            #     kernel[i+kernel_size//2][j+kernel_size//2] = image[row+i][col+j]
            new_row = min(max(row + i, 0), height - 1)
            new_col = min(max(col + j, 0), width - 1)
            kernel[i + kernel_size//2][j + kernel_size//2] = image[new_row][new_col]
    return kernel
```

Function rgb_to_gray:
使用 numpy 的 dot 去實現。
rgb 轉灰階是利用 hw1 的公式(0.3×R)+(0.59×G)+(0.11×B)去轉成灰階。
最後用 astype(np.uint8)是因為現在的的範圍是 0~255。
回傳 gray_image。

Function get_n_kernel:
先用 image.shape 找出 height 和 width，再根據 row 和 col(當前 pixel 的位置)抓取 kernel_size x kernel_size 大小的 kernel。
其中我不是使用 zero padding，而是超出 size 的範圍的話，則是取離此 pixel 範圍內離 pixel 最近的位置，如下圖所示。

	8	10	21	17	35
	2	43	15	72	21
	30	94	55	43	74
	36	28	69	88	56
	45	75	42	47	20

```python
def gaussian_kernel(kernel_size, sigma):
    '''
    Gaussian kernel
    '''
    kernel = np.zeros((kernel_size, kernel_size)).astype(np.float64)
    for x in range(-kernel_size//2, kernel_size//2+1):
        for y in range(-kernel_size//2, kernel_size//2+1):
            kernel[x+kernel_size//2][y+kernel_size//2] = 1 / (2 * np.pi * sigma**2) * np.exp(-(x**2 + y**2) / (2 * sigma**2))
    return kernel
```

Function gaussian_kernel:

計算 gaussian 的 value，以 Gaussian 2D filter 的公式進行計算，如作業簡報的公式算法 G(x, y) = (1 / 2 * pi * sigma ** 2) * exp ** (-(x ** 2 + y ** 2) / (2 * sigma ** 2))。

其中會以 kernel_size 的大小去做整個 kernel 的處理如圖所示(圖片是 3x3 的 kernel 大小)。

Gaussian 2D Filter

1. $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

   If $\sigma = 1$,

(-1,-1)	(0,-1)	(1,-1)
(-1,0)	(0,0)	(1,0)
(-1,1)	(0,1)	(1,1)

→

0.0585	0.0965	0.0585
0.0965	0.1591	0.0965
0.0585	0.0965	0.0585

```python
def gaussian_filter(image, kernel_size, sigma):
    '''
    Gaussian filter
    '''
    height = image.shape[0]
    width = image.shape[1]
    image = image.astype(np.float64)
    gaussian_filtered_image = np.zeros((height, width)).astype(np.float64)
    gaussian = gaussian_kernel(kernel_size, sigma)
    gaussian = gaussian / np.sum(gaussian)
    for row in range(height):
        for col in range(width):
            kernel = get_n_kernel(image, row, col, kernel_size)
            gaussian_filtered_image[row][col] = np.sum(kernel * gaussian)
    return gaussian_filtered_image
```

Function gaussian_filter:

利用 shape 找出 image 的 height 和 width，並利用 astype 設定 image 的 type 為 np.float64，然後利用 np.zeros 創建 gaussian_filtered_image 的 array。

利用 gaussian_kernel 找出 kernel 的 gaussian value，然後將 gaussian 除 gaussian 的總和並存在 gaussian 中。

利用 for 迴圈計算每個 pixel 的 value，其中會先用 get_n_kernel 抓取當前 pixel 的 kernel，並利用 np.sum(kernel * gaussian)得出 kernel * gaussian 的總和，再存入當前 pixel 的 value 中。

Gaussian 2D Filter

1. $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

   If $\sigma = 1$,

(-1,-1)	(0,-1)	(1,-1)
(-1,0)	(0,0)	(1,0)
(-1,1)	(0,1)	(1,1)

⟶

0.0585	0.0965	0.0585
0.0965	0.1591	0.0965
0.0585	0.0965	0.0585

2. Normalization
3. Convolution ：I*G

```python
def sobel_filter(image):
    '''
    Sobel filter, get gradient magnitude and slope
    '''
    height = image.shape[0]
    width = image.shape[1]
    image = image.astype(np.float64)
    G = np.zeros((height, width)).astype(np.float64)
    theta = np.zeros((height, width)).astype(np.float64)
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    # kernel = get_n_kernel(image, 0, 0, 3)
    # print(np.hypot(np.sum(kernel * sobel_x), np.sum(kernel * sobel_y)))
    # print(np.sqrt(np.sum(kernel * sobel_x)**2 + np.sum(kernel * sobel_y)**2))
    for row in range(height):
        for col in range(width):
            kernel = get_n_kernel(image, row, col, 3)
            Gx = np.sum(kernel * sobel_x)
            Gy = np.sum(kernel * sobel_y)
            G[row][col] = np.hypot(Gx, Gy)
            theta[row][col] = np.arctan2(Gy, Gx)
    G = G / G.max() * 255
    return G, theta
```

Function sobel_filter:

利用 shape 找出 height 和 width，並使用 astype 把 image 的 type 設成 np.float64，然後利用 np.zeros 創建 G 和 theta 分別存 magnitude 和 slope，接著建立 sobel_x 和 sobel_y 的 array。

註解的部分是觀察 np.hypot(np.sum(kernel * sobel_x), np.sum(kernel * sobel_y))是否跟 np.sqrt(np.sum(kernel * sobel_x)**2 + np.sum(kernel * sobel_y)**2)相等。

利用 for 迴圈計算每個 pixel 的 magnitude 和 slope，首先會利用 get_n_kernel 找到 kernel，然後找到 pixel 的 Gx 和 Gy，計算 magnitude($(Gx^2+Gy^2)^{1/2}$)和 slope(arctan(Gy/Gx))，並將結果分別存入 G 和 theta。

最後計算完後，將 G 進行 normalize 使其的值在 0-255 間，這樣可以輸出圖片方便觀察。

2. Finding Intensity Gradient of the Image

- Use operator to get image gradient in x and y directions.
- Then, the magnitude $\boldsymbol{G}$ and the slope $\boldsymbol{\theta}$ of the gradient are calculated

Sobel

-1	0	1
-2	0	2
-1	0	1

Gx

-1	-2	-1
0	0	0
1	2	1

Gy

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = arctan\left(\frac{G_y}{G_x}\right)$$

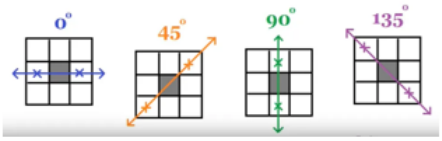
```python
def non_maximum_suppression(image, theta):
    '''
    Non-maximum suppression
    '''
    height = image.shape[0]
    width = image.shape[1]
    suppressed_image = np.zeros((height, width)).astype(np.float64)
    angle = theta * 180. / np.pi
    angle[angle < 0] += 180

    for row in range(1, height-1):
        for col in range(1, width-1):
            pixel_q = 255
            pixel_r = 255
            # angle 0
            if (0 <= angle[row][col] < 22.5) or (157.5 <= angle[row][col] <= 180):
                pixel_q = image[row][col-1]
                pixel_r = image[row][col+1]
            # angle 45
            elif (22.5 <= angle[row][col] < 67.5):
                pixel_q = image[row-1][col+1]
                pixel_r = image[row+1][col-1]
            # angle 90
            elif (67.5 <= angle[row][col] < 112.5):
                pixel_q = image[row-1][col]
                pixel_r = image[row+1][col]
            # angle 135
            elif (112.5 <= angle[row][col] < 157.5):
                pixel_q = image[row-1][col-1]
                pixel_r = image[row+1][col+1]

            if image[row][col] >= pixel_q and image[row][col] >= pixel_r:
                suppressed_image[row][col] = image[row][col]
            else:
                suppressed_image[row][col] = 0

    return suppressed_image
```

Function non_maximum_suppression:

利用 shape 找出 height 和 width，利用 np.zeros 創建 suppressed_image 的 array，然後將 theta 轉成角度，並存入 angle，接著在將 angle 中<0 的值都加上 180 度。

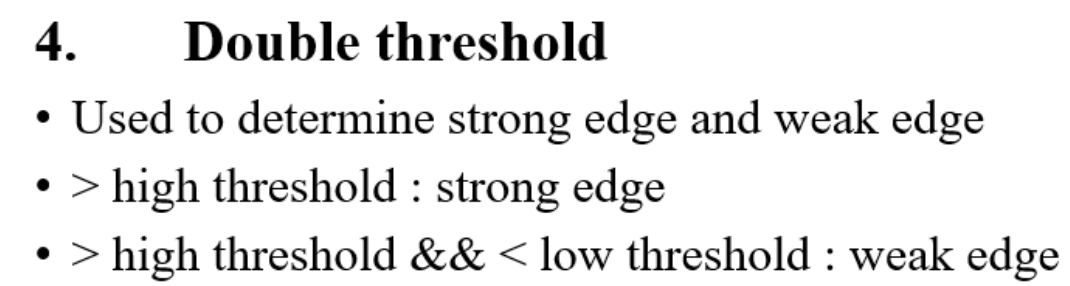
使用 for 迴圈根據 pixel 的 angle 去判斷 pixel 在 direction 的 neighbor 中是否為最大的點，如果是的話會保留 pixel 的值，不是的話則設成 0，如同下圖所示。

- Consider in 4 directions and compare with neighbor pixels



```python
def double_threshold(image, low_threshold_ratio, high_threshold_ratio):
    '''
    Double threshold with dynamic thresholding
    '''
    high_threshold = image.max() * high_threshold_ratio
    low_threshold = high_threshold * low_threshold_ratio
    height = image.shape[0]
    width = image.shape[1]
    double_threshold_image = np.zeros((height, width)).astype(np.float64)

    weak = 127.5
    strong = 255

    double_threshold_image[image >= high_threshold] = strong
    double_threshold_image[image < low_threshold] = 0
    double_threshold_image[(image >= low_threshold) & (image < high_threshold)] = weak

    return double_threshold_image, weak, strong
```

Function double_threshold:
這次的 threshold 是利用 ratio 的方式(動態)去尋找 high_threshold 和 low_threshold，其中 high_threshold 會是 image 的最大值乘上 high_threshold_ratio，low_threshold 是 high_threshold 乘上 low_threshold_ratio。
使用 shape 找出 image 的 height 和 width，並利用 np.zeros 創建 double_threshold_image 的 array。
將 weak 的值設成 127.5，strong 的值設成 255
然後將 image 中 pixel >= high_threshold 的 value 都設成 strong，image 中 pixel < low_threshold 的 value 都設成 0，image 中 low_threshold <= pixel < high_threshold 的 value 都設成 weak，如作業簡報上所示。



```python
def edge_tracking(image, weak, strong):
    '''
    Edge tracking by Hysteresis
    '''
    height = image.shape[0]
    width = image.shape[1]
    for row in range(1, height-1):
        for col in range(1, width-1):
            if image[row][col] == weak:
                kernel = get_n_kernel(image, row, col, 3)
                if strong in kernel:
                    image[row][col] = strong
                else:
                    image[row][col] = 0
    return image
```

Function edge_tracking:

利用 shape 找出 image 的 height 和 width，然後使用 for 迴圈判斷所有 weak edge 是否為 strong edge，如果是的話會將 weak edge 轉成 strong edge，如果不是的話，則將此 weak edge 刪除(設成 0)。做法是如果其 pixel 的 kernel 內含有 strong 的 value，則將此 pixel 設成 strong，如果沒有 strong 的 value，則設成 0。

```python
def image(number, gaussian_kernel_size, sigma, double_threshold_low, double_threshold_high):
    '''
    For img{number}.png
    '''
    if number == 1:
        origin_image = cv2.imread(f'images/img{number}.jpeg')
    else:
        origin_image = cv2.imread(f'images/img{number}.jpg')

    gray_image = rgb_to_gray(origin_image)
    cv2.imwrite(f'results/img{number}_0.jpg', gray_image)

    gaussian_filtered_image = gaussian_filter(gray_image, gaussian_kernel_size, sigma)
    cv2.imwrite(f'results/img{number}_1.jpg', gaussian_filtered_image)

    magnitude, slope = sobel_filter(gaussian_filtered_image)
    cv2.imwrite(f'results/img{number}_2.jpg', magnitude)
    print(magnitude.max(), magnitude.min())

    suppressed_image = non_maximum_suppression(magnitude, slope)
    cv2.imwrite(f'results/img{number}_3.jpg', suppressed_image)

    double_threshold_image, weak, strong = double_threshold(suppressed_image, double_threshold_low, double_threshold_high)
    cv2.imwrite(f'results/img{number}_4.jpg', double_threshold_image)

    edge_image = edge_tracking(double_threshold_image, weak, strong)
    cv2.imwrite(f'results/img{number}_sobel.jpg', edge_image)
```

Function image:

使用 cv2 的 imread 讀取圖片到 origin_image。

使用 rgb_to_gray(origin_image)得出 gray_image，然後使用 cv2.imwrite 存檔。

使用 gaussian_filter(gray_image, gaussian_kernel_size, sigma)得出 gaussian_filtered_image，然後使用 cv2.imwrite 存檔。

使用 sobel_filter(gaussian_filtered_image)得出 magnitude 和 slope，然後使用

cv2.imwrite 將 magnitude(sobel_filtered_image)存檔，並 print 出 magnitude 的最大值和最小值用來觀察。
使用 non_maximum_suppression(magnitude, slope)得出 suppressed_image，然後使用 cv2.imwrite 存檔。
使用 double_threshold(suppressed_image, double_threshold_low, double_threshold_high)得出 double_threshold_image、weak 和 strong，然後使用 cv2.imwrite 將 double_threshold_image 存檔。
使用 edge_tracking(double_threshold_image, weak, strong)得出 edge_image，然後使用 cv2.imwrite 存檔。

```
if __name__ == '__main__':
    image(number=1, gaussian_kernel_size=5, sigma=1.7, double_threshold_low=0.7, double_threshold_high=0.45)
    image(number=2, gaussian_kernel_size=7, sigma=1.7, double_threshold_low=0.7, double_threshold_high=0.25)
    image(number=3, gaussian_kernel_size=3, sigma=1.3, double_threshold_low=0.2, double_threshold_high=0.19)
```

執行 image，產生所有助教給的圖片的 edge_image。

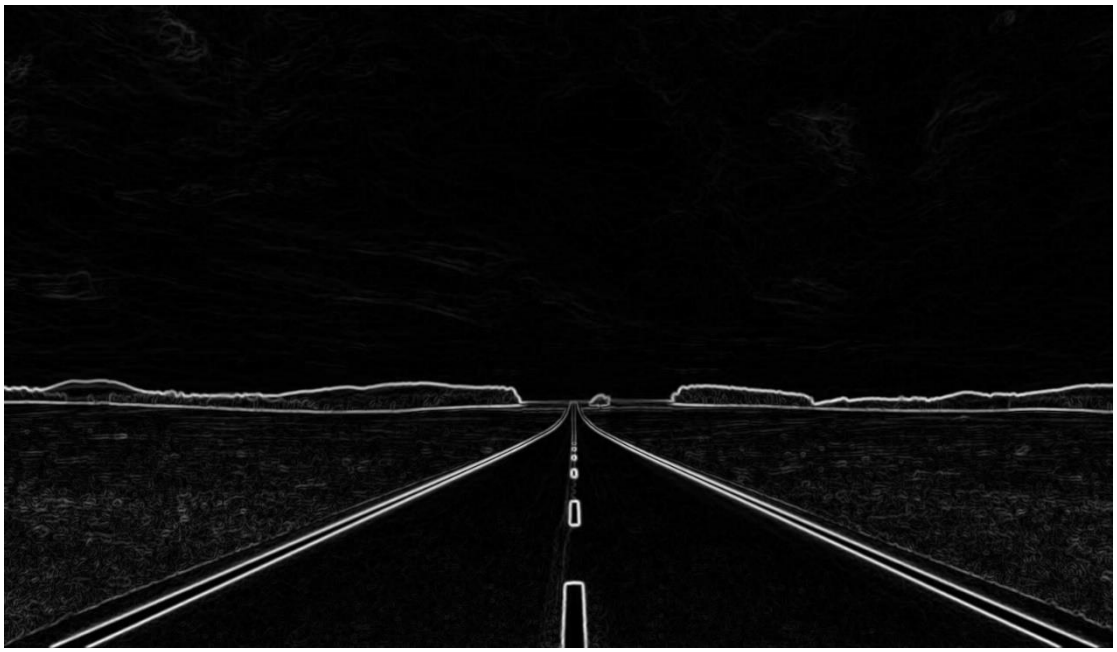
Result images

img1.jpg

(gray image)

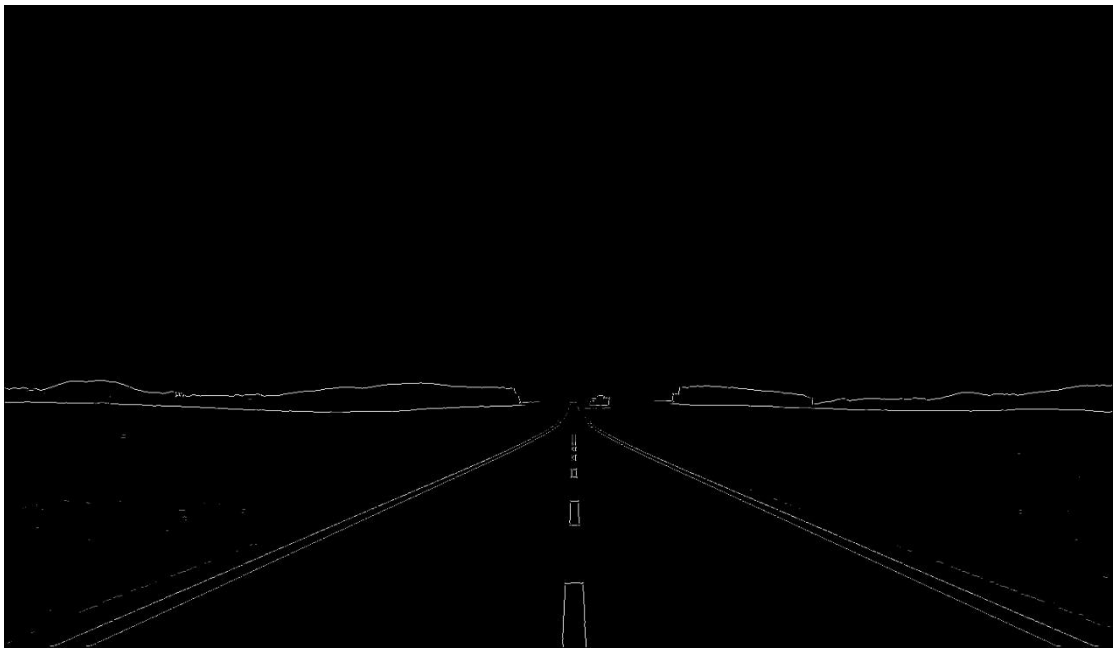
(gaussian_filter)

(sobel filter/magnitude)

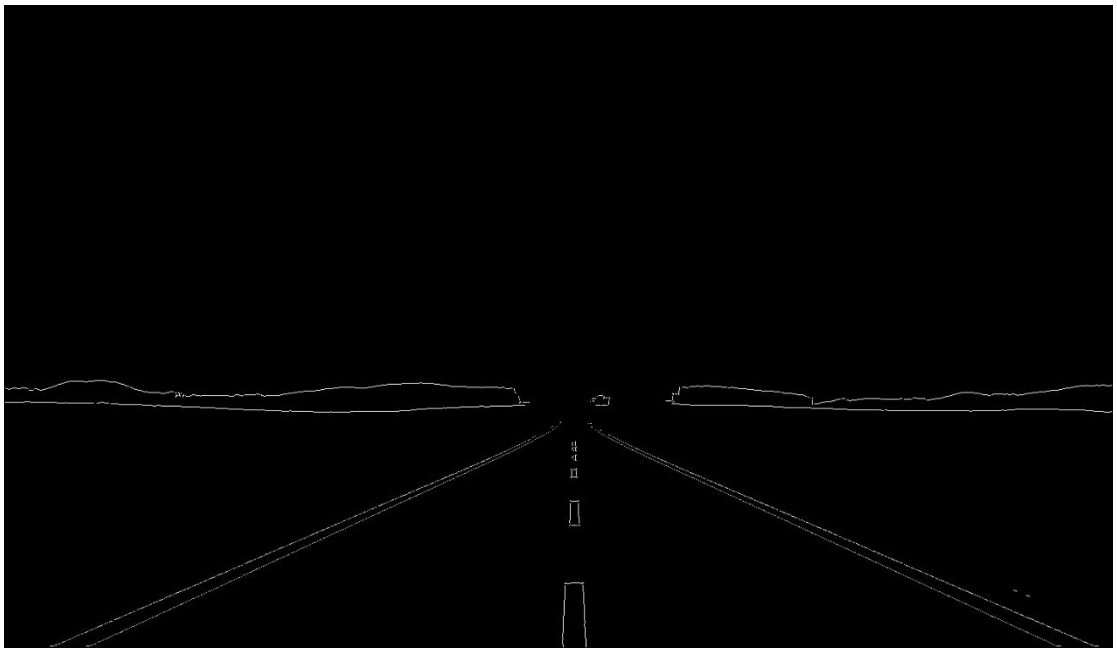


(non-maximum suppression)

(double threshold)



(edge tracking by Hysteresis)
img1_sobel.jpg



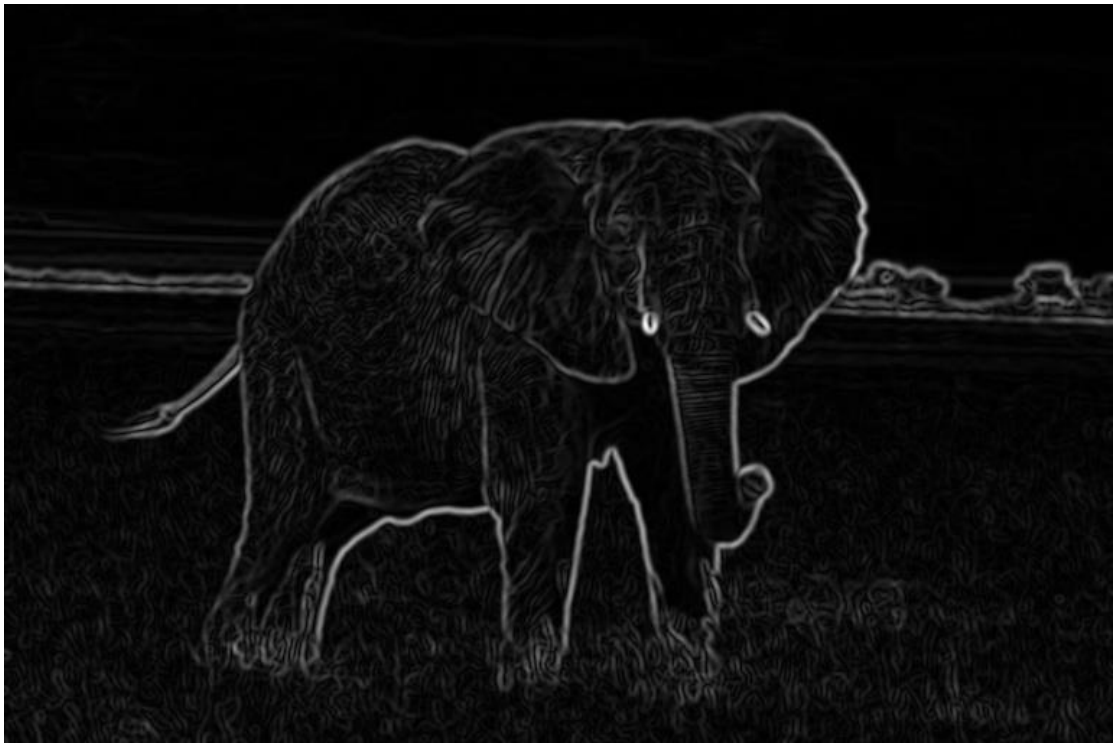
我預期的結果與最後的輸出相同，只有右下角的部分有一點雜訊。
其中在 sobel filter 的圖片中可以明顯地看到 edge，可以看出最後的輸出的雛型，在 non-maximum suppression 中有明顯的將 sobel filter 的圖片細化 edge，在 double threshold 中有消除了一些雜訊，到 edge tracking by Hysteresis 中有成功將連到 strong edge 的 weak edge 接起來，沒有連到 strong edge 的 weak edge 當雜訊消除。

img2.jpg

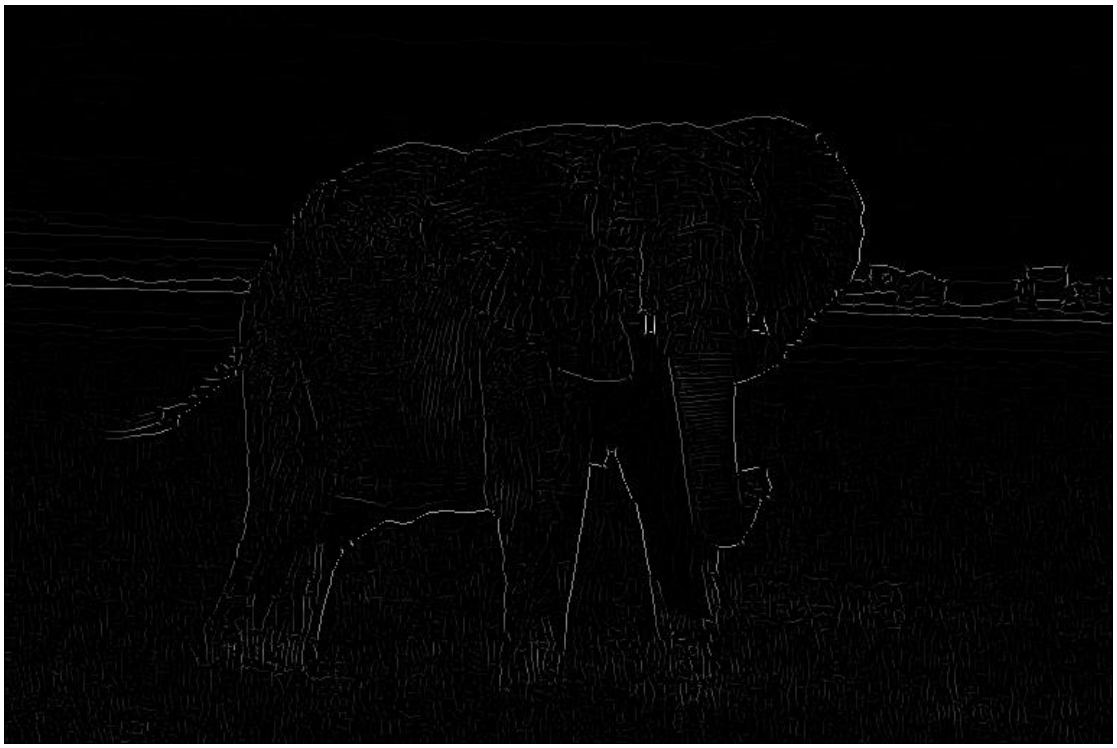
(gray image)

(gaussian_filter)

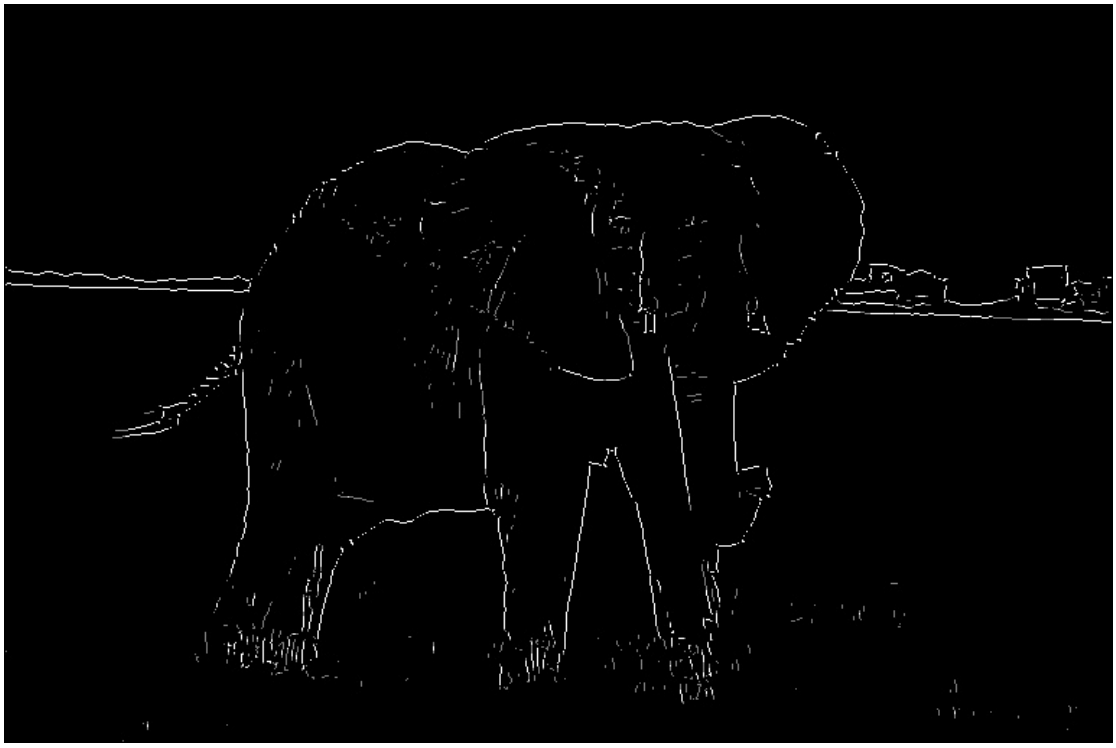
(sobel filter/magnitude)



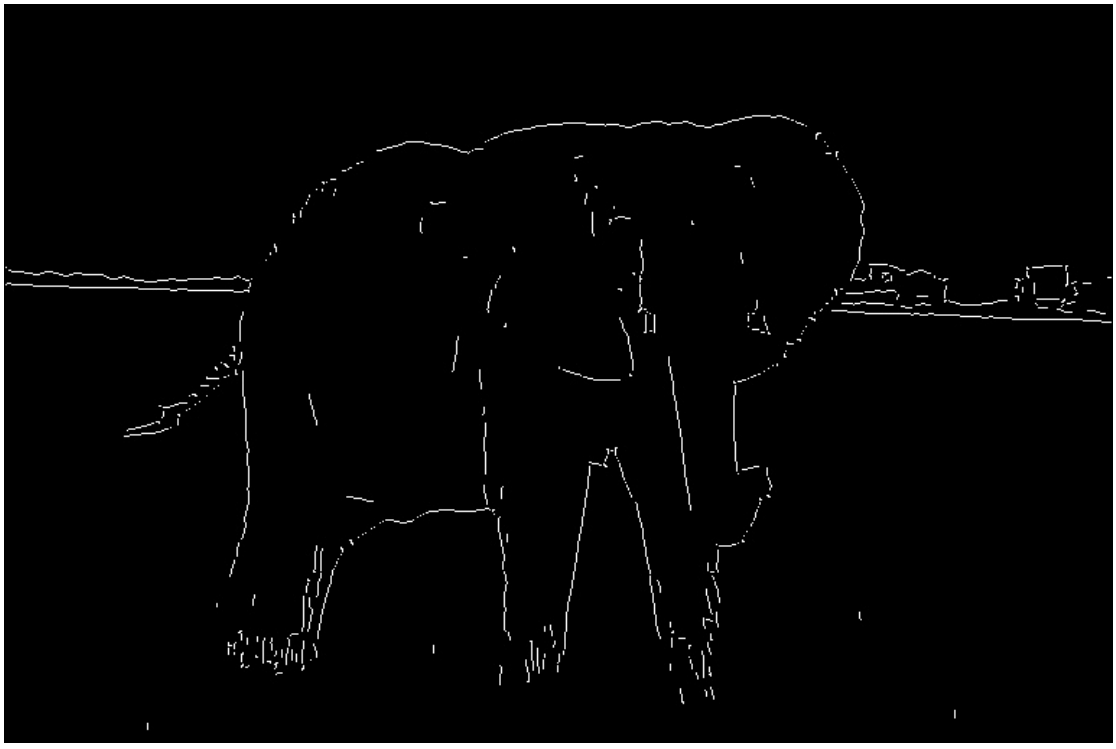
(non-maximum suppression)



(double threshold)



(edge tracking by Hysteresis)
img2_sobel.jpg



我預期的結果與最後的輸出大致上相同，仍然有一些草的雜訊沒有成功消除，大象的左半部分也有些 edge 沒有接起來。
其中在 sobel filter 的圖片中可以明顯地看到 edge，可以看出最後的輸出的雛型，在 non-maximum suppression 中有明顯的將 sobel filter 的圖片細化 edge，在

double threshold 中有消除了一些雜訊，到 edge tracking by Hysteresis 中有成功將大部分連到 strong edge 的 weak edge 接起來，沒有連到 strong edge 的 weak edge 當雜訊消除。

img3.jpg
(gray image)

(gaussian_filter)

(sobel filter/magnitude)

(non-maximum suppression)

(double threshold)

(edge tracking by Hysteresis)
img3_sobel.jpg

我預期的結果與最後的輸出大致上相同，左上的帽子沒有很好的連接起來，我覺得可能是因為在 sobel filter 的結果中帽子的左上角的 gradient magnitude 比較小一點。
其中在 sobel filter 的圖片中可以明顯地看到 edge，可以看出最後的輸出的雛型，在 non-maximum suppression 中有明顯的將 sobel filter 的圖片細化 edge，在 double threshold 中有消除了一些雜訊，到 edge tracking by Hysteresis 中有成功將連到 strong edge 的 weak edge 接起來，沒有連到 strong edge 的 weak edge 當雜訊消除。

最後的結果總結：
這次 hw6 的作業比較要注意的地方是找出 kernel 時，不要使用 zero padding，因為這樣圖片的邊框要是在經過 RGB image 轉成 gray image 和 gaussian filter 後 pixel 的 value 不是黑色的話，就會讓 zero padding 的結果將邊框設成 edge，這樣會導致圖片出來的結果會出現白色的邊框。我的解決方式是在找 kernel 時，讓超出 image 範圍的 pixel 的 value 設為 image 對應邊框的 pixel。

然後大象的圖和 Lenna 的圖我覺得如果 gaussian filter 時，消除的 noise 和 smooth 的程度在多一點的話，我覺得最後出來的 image 的 edge 效果會更好。其中我覺得這兩張圖的 gaussian filter 再加上 median filter 後，最後的 image 的 edge 會比較好的呈現，只是相對應的會少一些圖片的 feature。