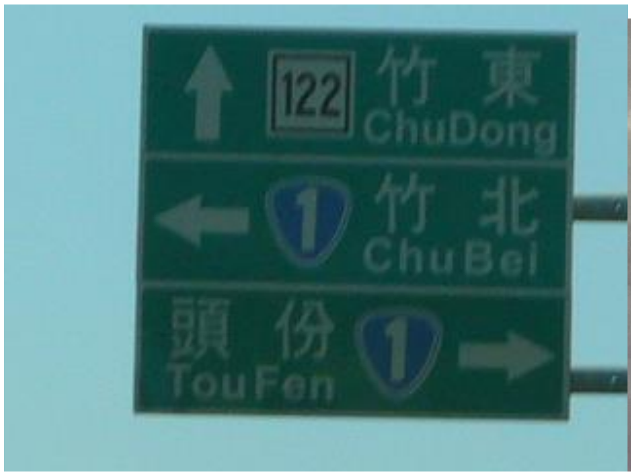
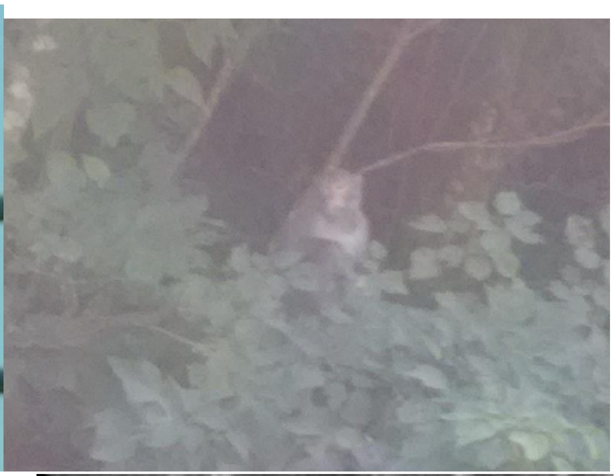
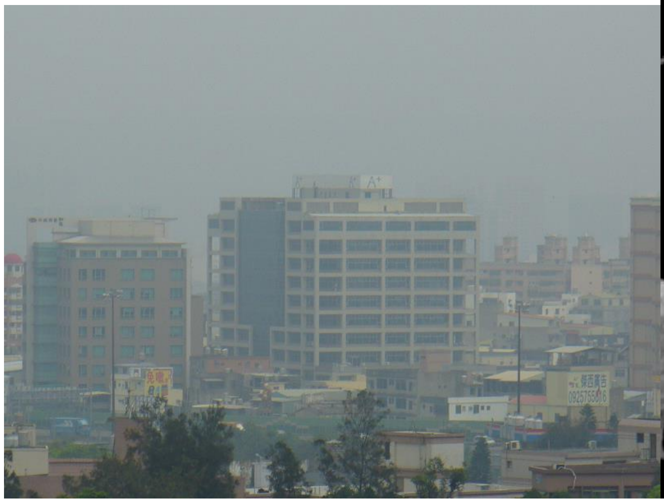
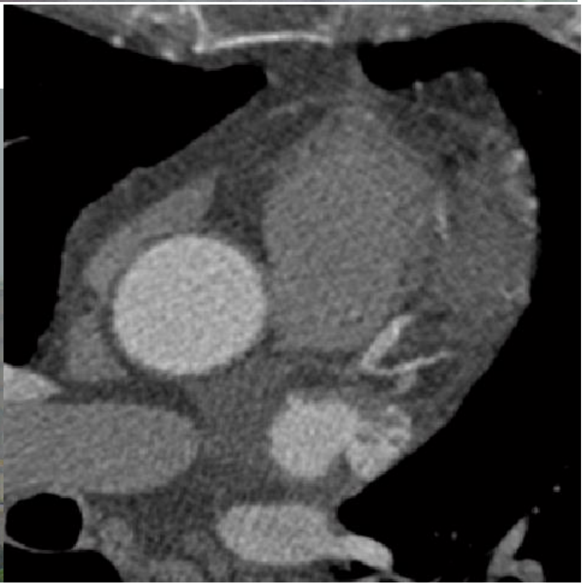
影像處理 HW2

資科工碩 309551013 周昱辰

這次作業的目標為使用各種邊緣偵測的濾波器來對上次作業所提供的其中四張照片做使用，並利用各種 threshold 來觀察每條邊的重要程度，再使用上次作業所學的對比調整、模糊濾波器等對邊緣偵測的影響，本次使用的照片如下：



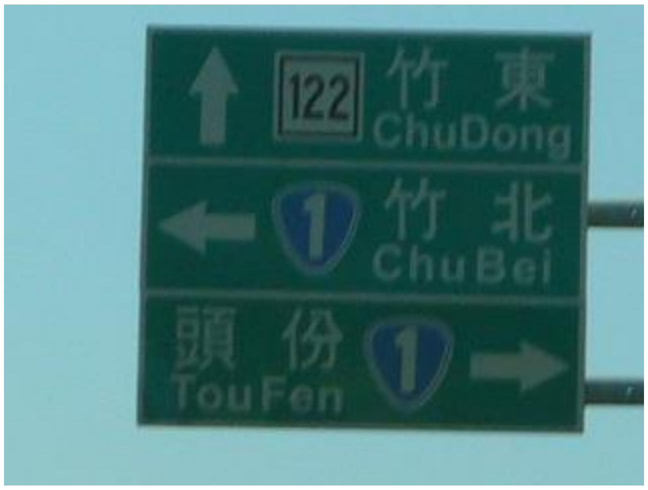
這次作業使用到的技術如下：

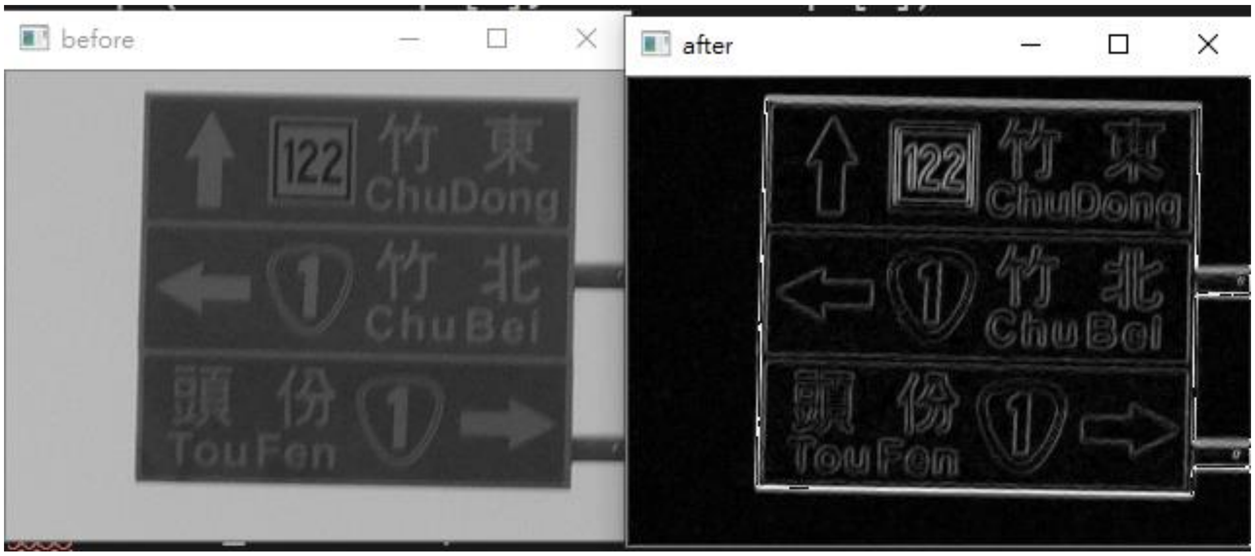
1. gaussian filter:
   使用 3*3 的 box size 並採用 zero padding 對圖像做模糊化處理。
2. contrast stretching:
   以 128 做分界，對其值做調整，已達到對比強化的效果。
3. prewitt filter:

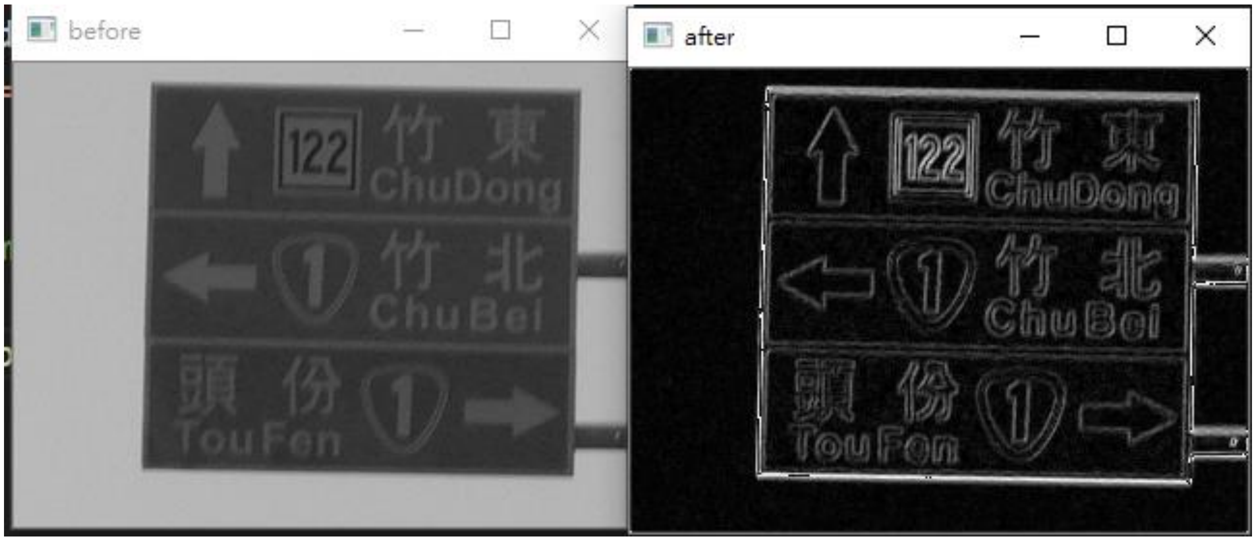
使用 3*3 的 box size 並採用 zero padding 對圖像做邊緣偵測，並算出其 1-norm 和 2-norm 的 gradient magnitude。

4. sobel filter:
   使用 3*3 的 box size 並採用 zero padding 對圖像做邊緣偵測，並算出其 1-norm 和 2-norm 的 gradient magnitude 以及其 gradient direction 為 canny edge detector 做準備。
5. Laplacian of Gaussian (LoG):
   先對圖片做 gaussian filter 後，再對其做 laplace filter，最後再做正規化得到結果。
6. Canny edge detector:
   第一步:Gaussian filter 模糊圖片 。
   第二步:算出其 gradient magnitude(1-norm) 還有 gradient direction。
   第三步:去除 direction 上比較小的部分來降低線條寬度。
   第四步:挑出 strong edge 還有 weak edge。
   第五步:若 weak edge 和 strong edge 相連則選，否則遺棄。

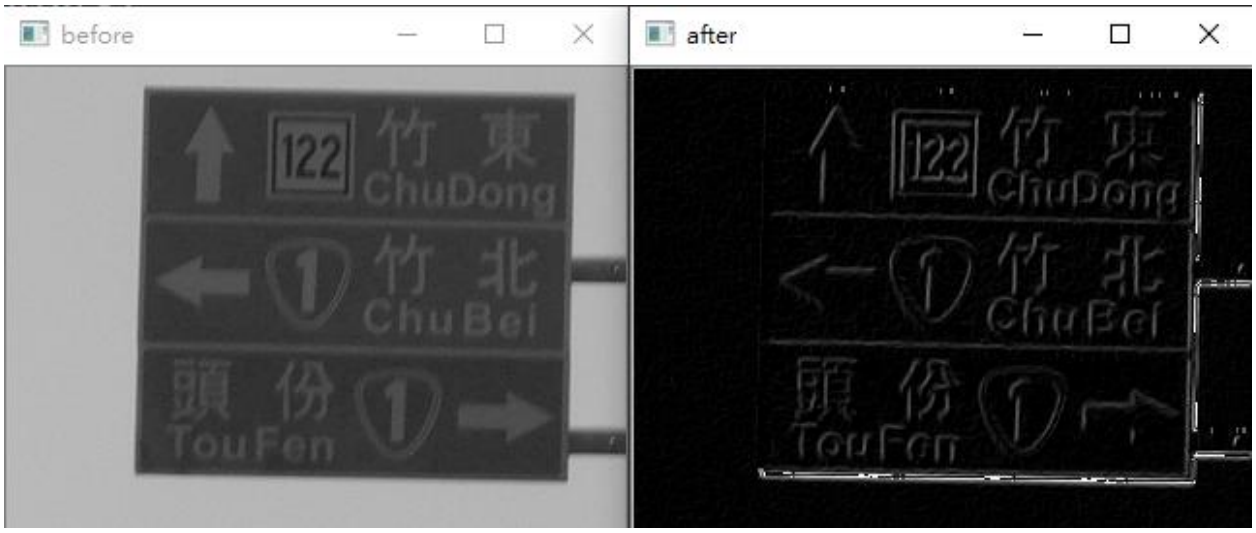
第一張圖:



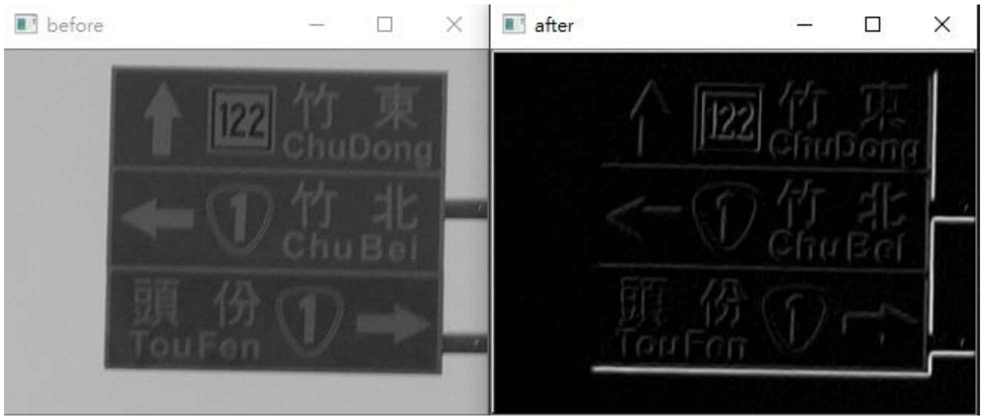




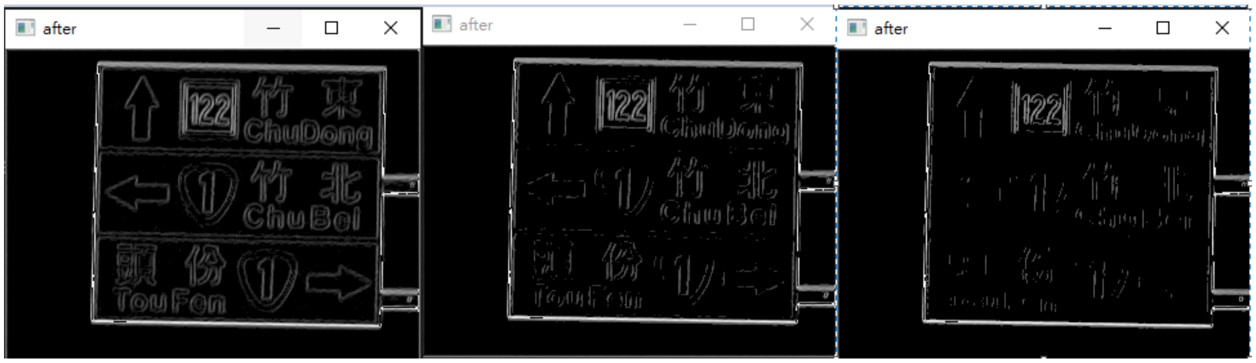
兩張圖都是使用 prewitt filter，第一張是 1-norm 而第二張則是 2-norm，看起來差異不大。



這張則是增加 contrast 後再做 prewitt filter，可以看到有些邊反而不見了。

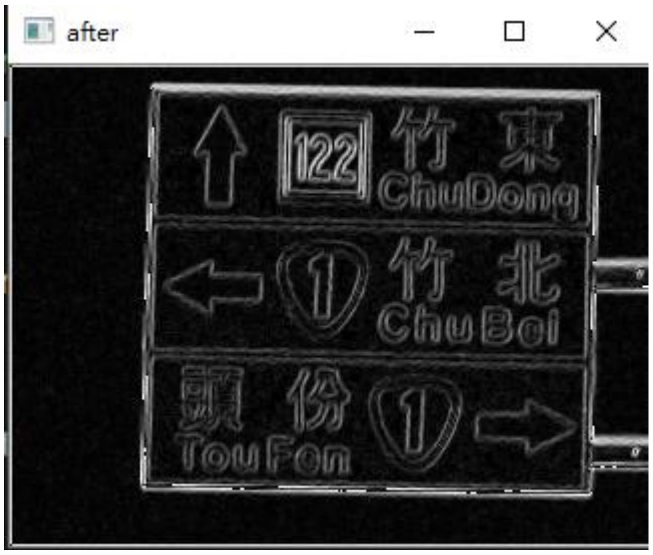


這張則是先做 gaussian 模糊化之後再做 prewitt，可以發現雜訊少了很多，線條較為筆直。

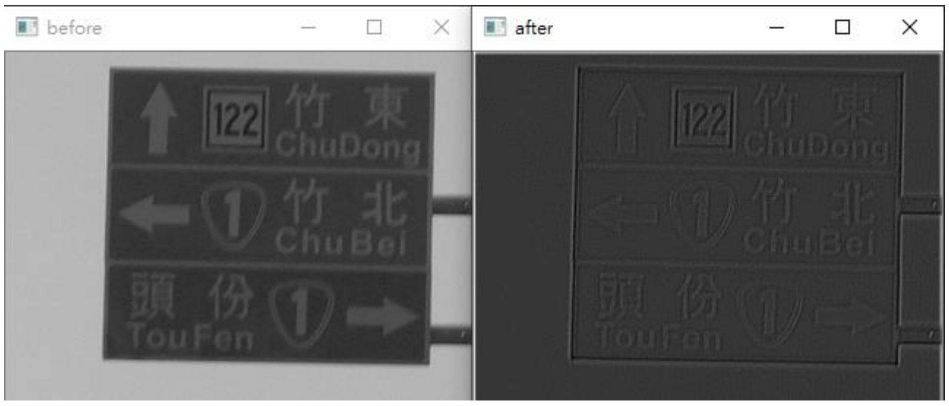


這三張由左到右分別是 threshold=30,50,70 的 prewitt filter。

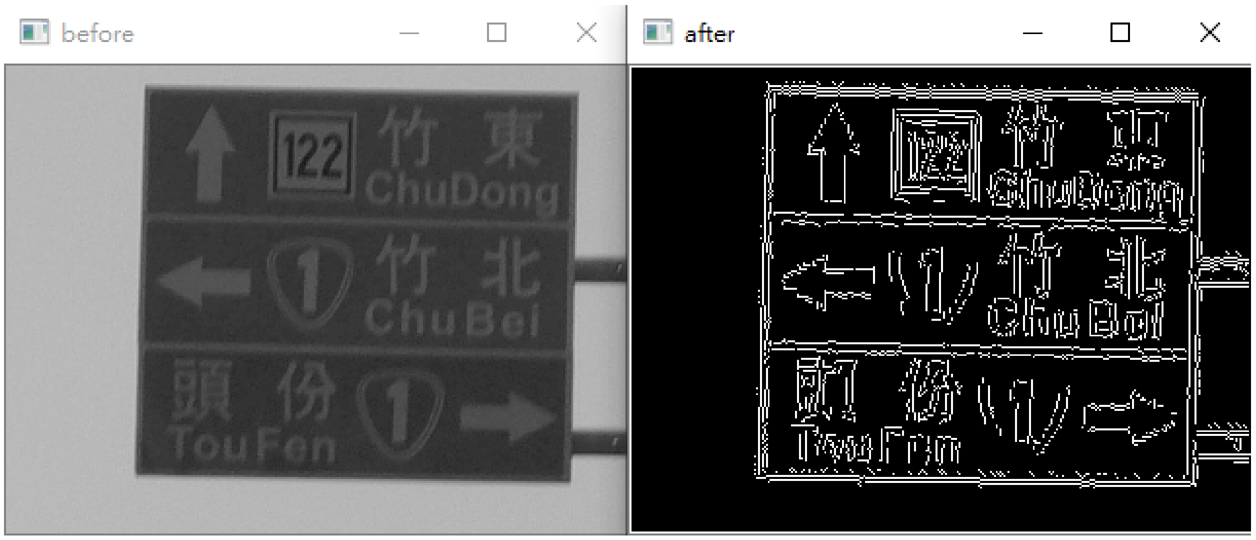
感覺因為這圖雜訊本來就不高，threshold 反而是讓原本的邊變得模糊。



這張則是 sobel filter 的結果，單純以選擇的圖例來說，我認為跟 prewitt 相差不大。



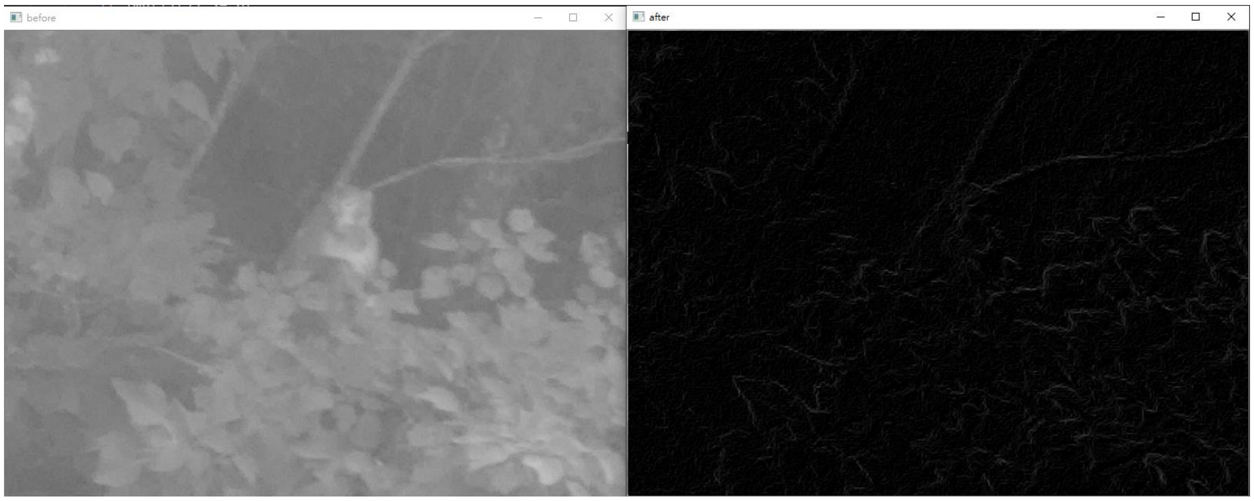
這張圖則是 LoG，框框勉強可以看的到，但是箭頭倒是快消失了。



最後則是 canny edge ，我試了一段時間感覺 threshold 設為 40,20 的時候算是比較清晰的了。

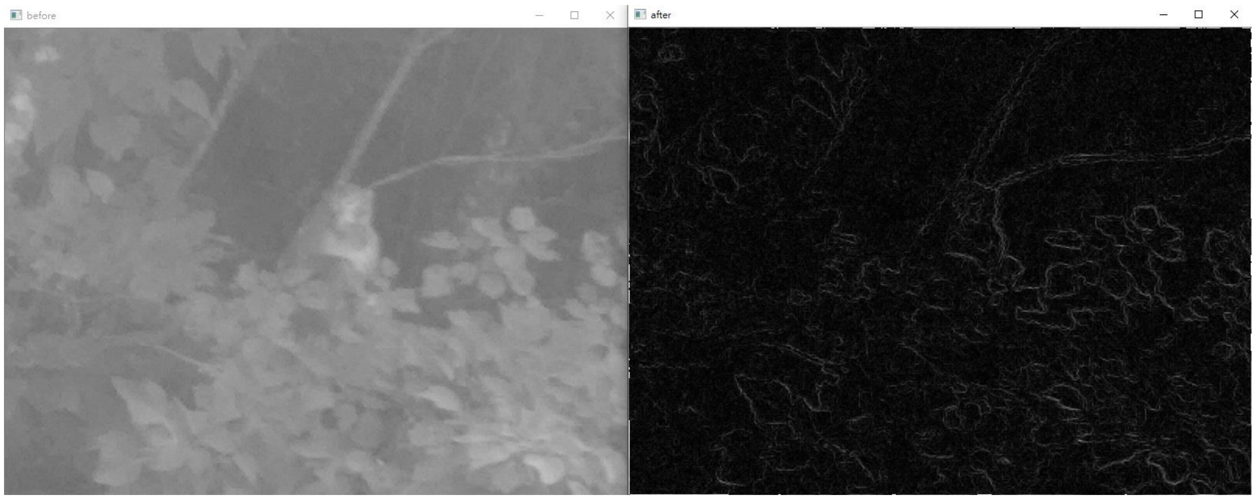
第二張圖:

首先是 prewitt

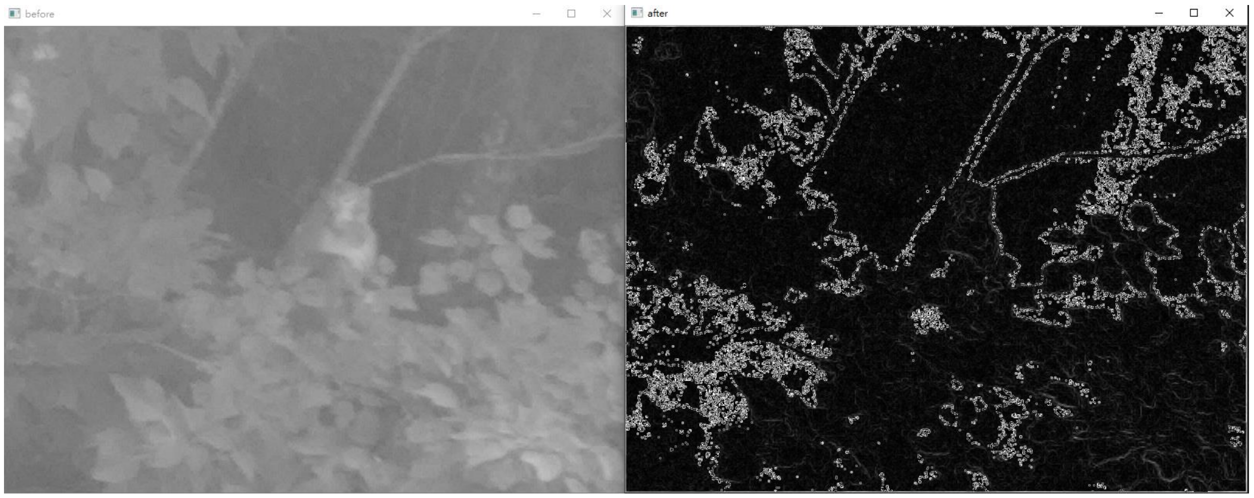


恩… 由於圖片本身就非常不清楚，效果滿差的。

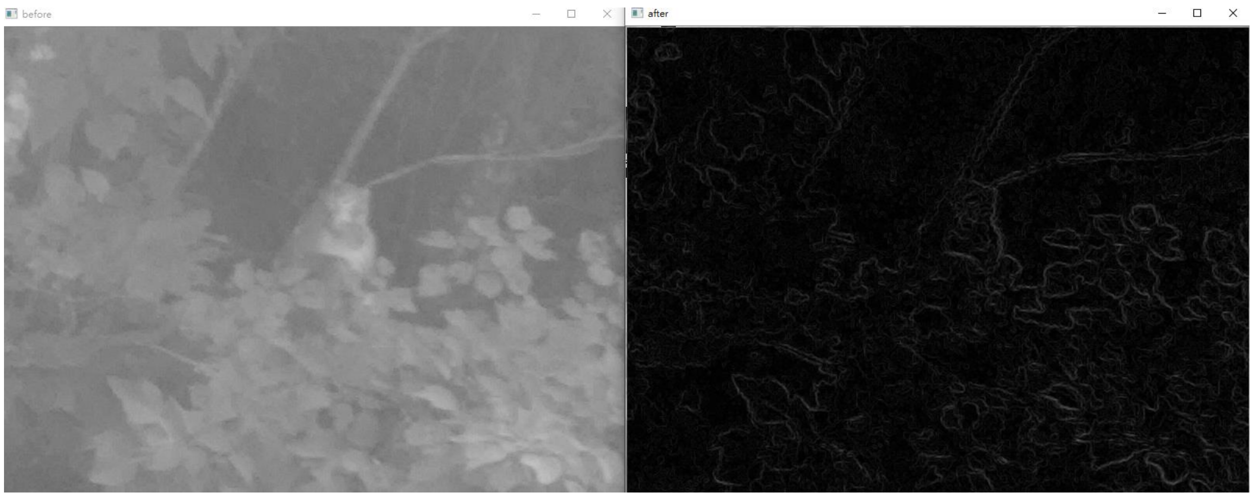
若是先用 gaussian 再做 sobel 就會很慘，原圖夠模糊了還用 gaussian…



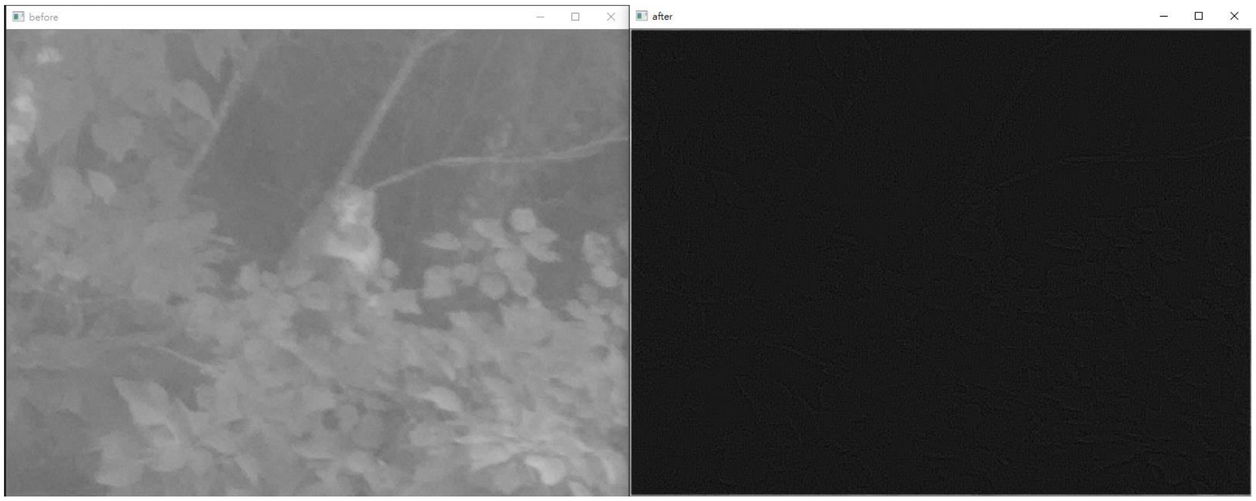
Sobel 在這邊就明顯的比 prewitt 好上很多。



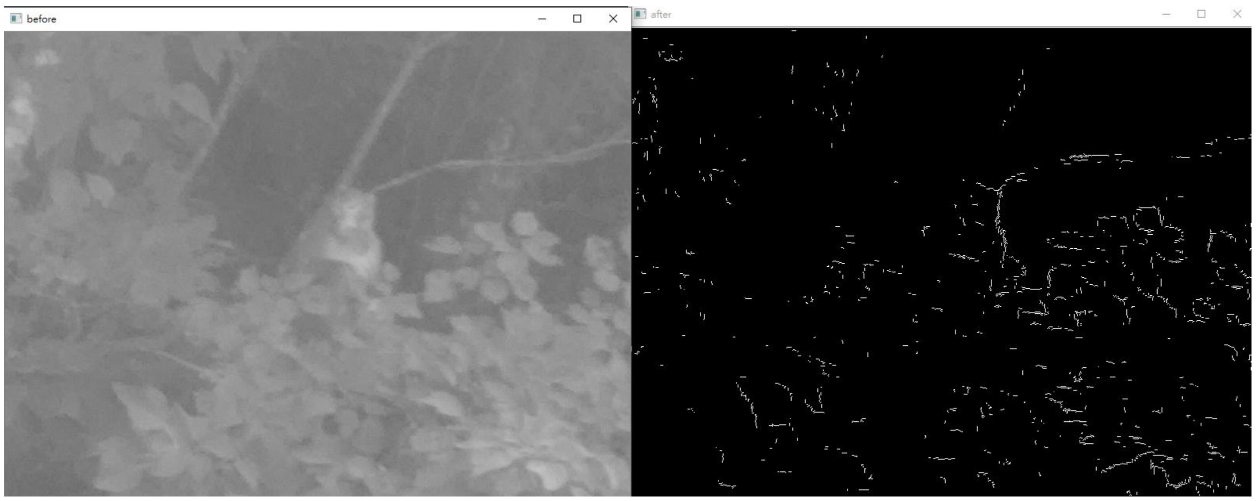
若先增加 contrast 再做 sobel 就可以看到很明顯的輪廓了。



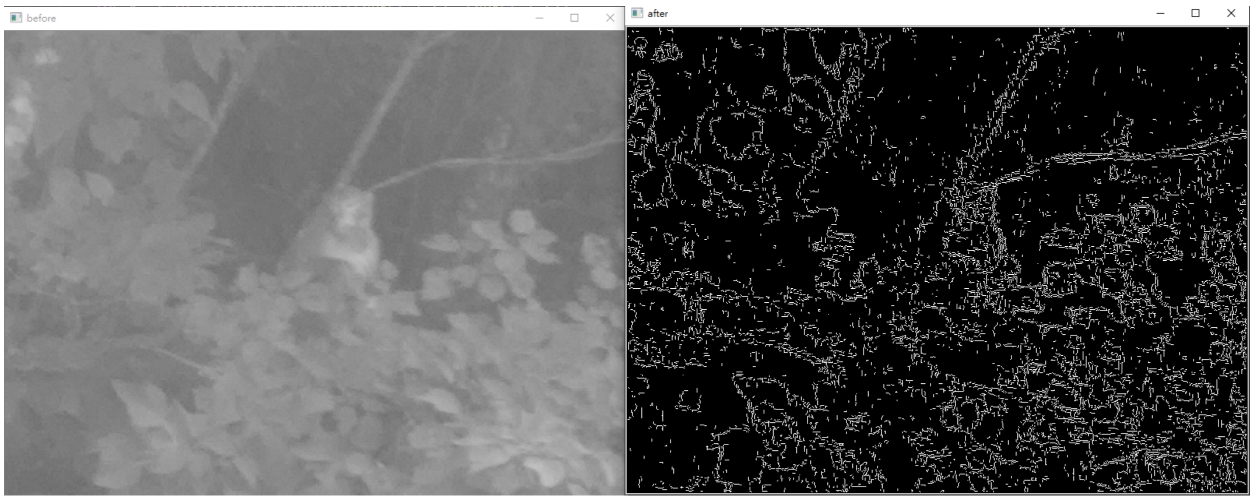
若是先用 gaussian 再做 sobel 就會很慘，原圖夠模糊了還用 gaussian…



Log 基本上也看不到啥東西。

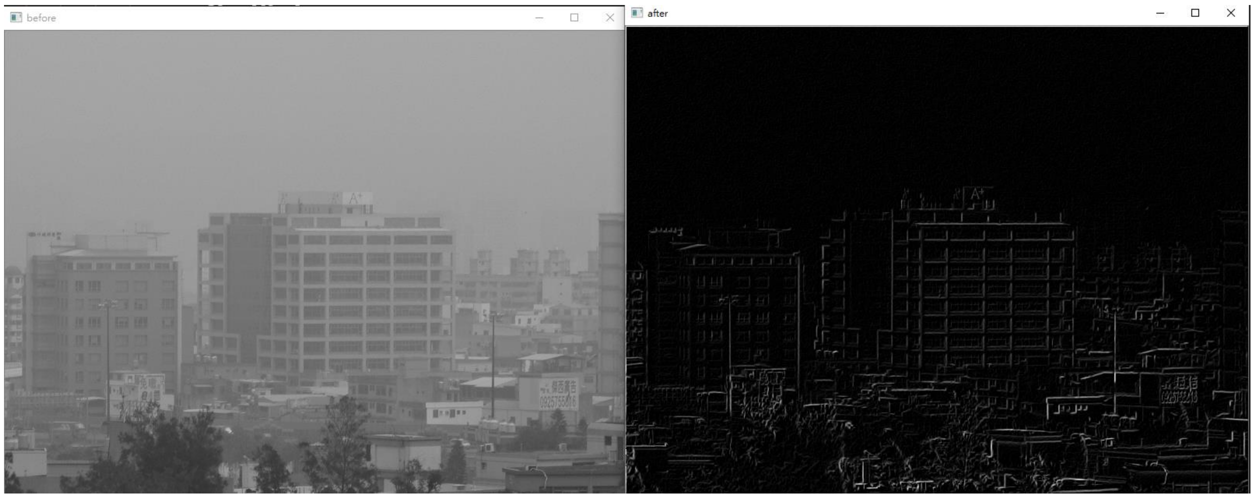


Canny 設 40,20 的話很多邊都會消失。

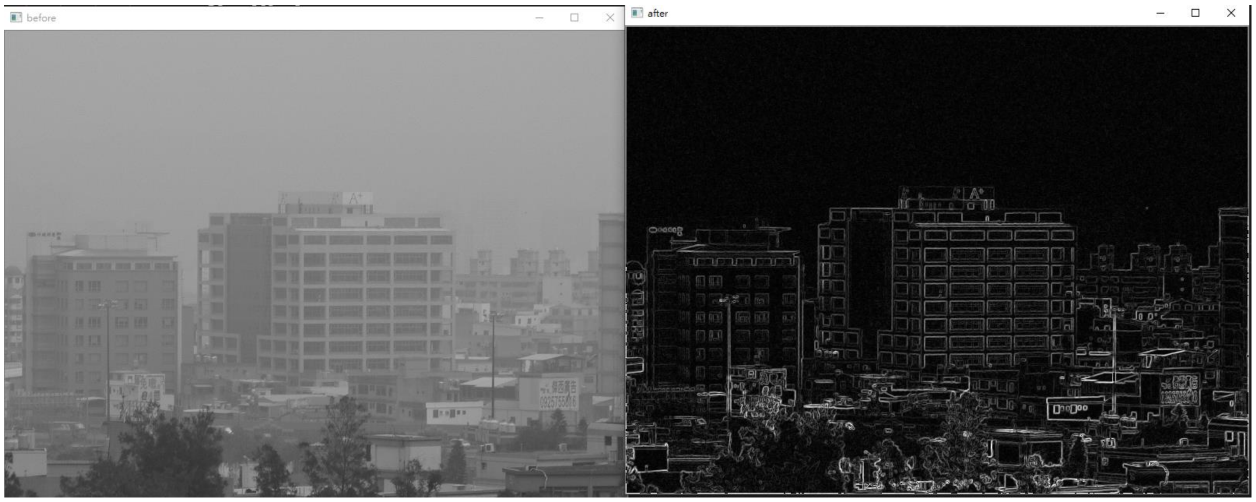


但 Canny 設 20,10 的話會把太多東西收進來，雜訊很多。

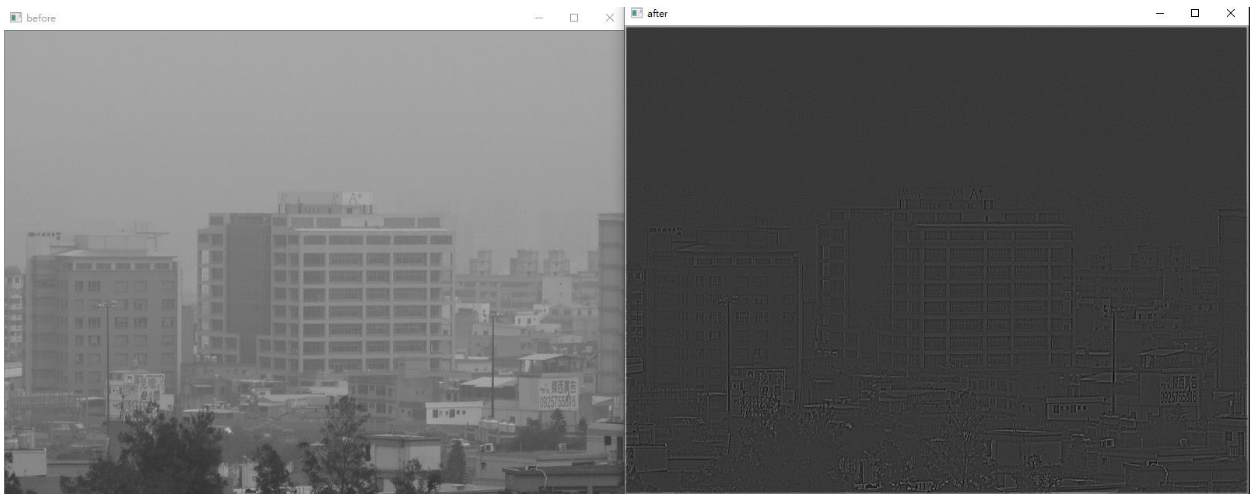
第三張圖:



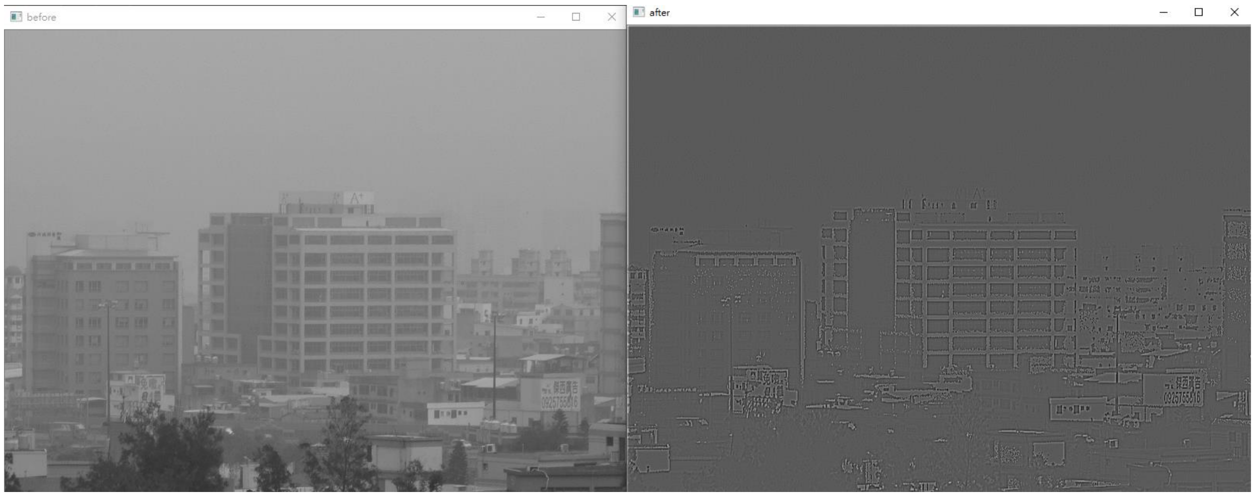
先用 prewitt 測試看看， 整體輪廓算有出現了。



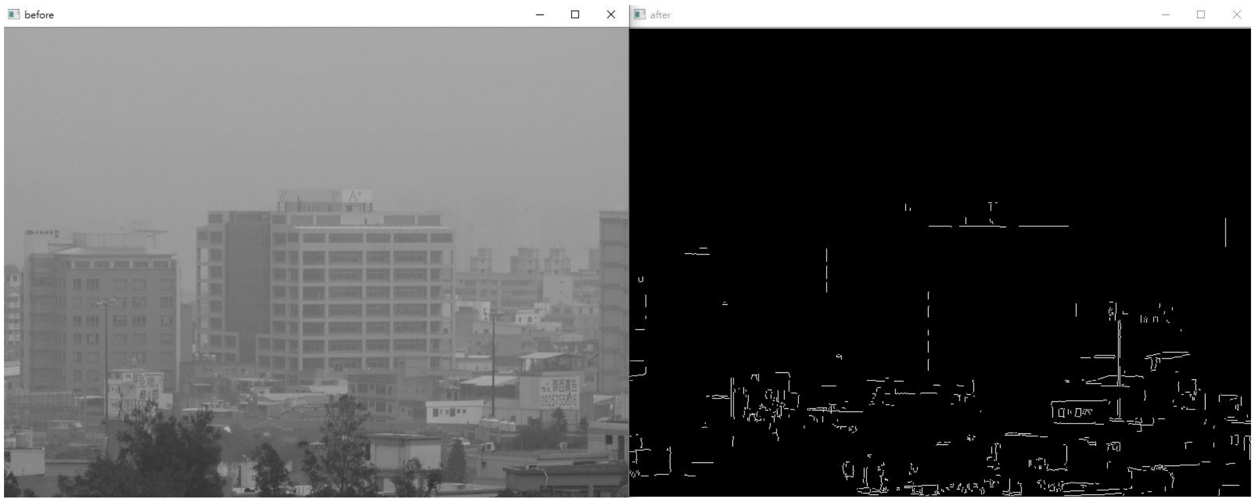
Sobel 很明顯地又比 prewitt 好了。



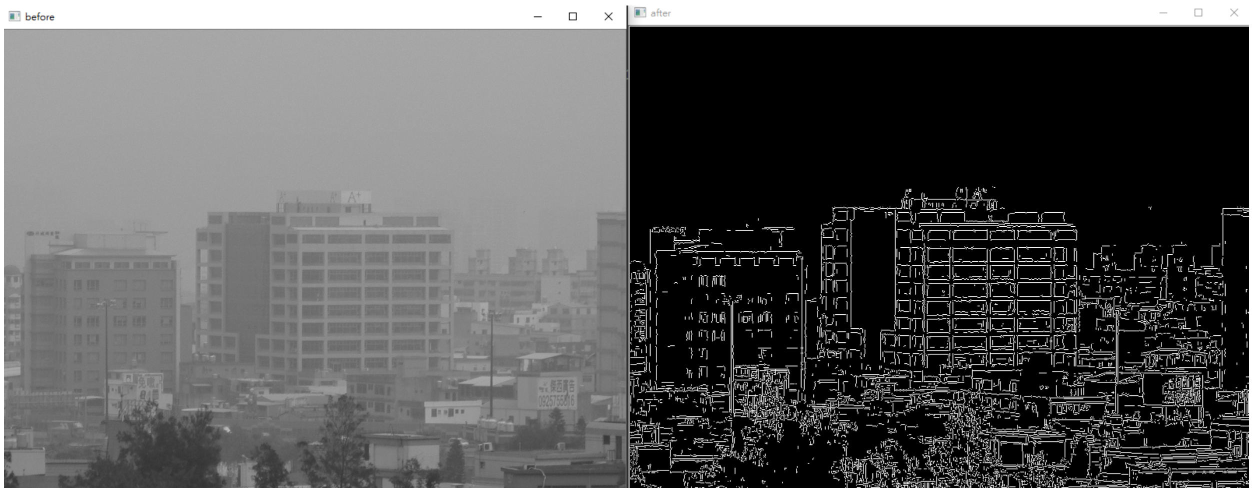
LoG 本身則沒有很清楚 甚至比 prewitt 還爛一點。



先做 contrast 強化過後 LoG 就有明顯的改善了。

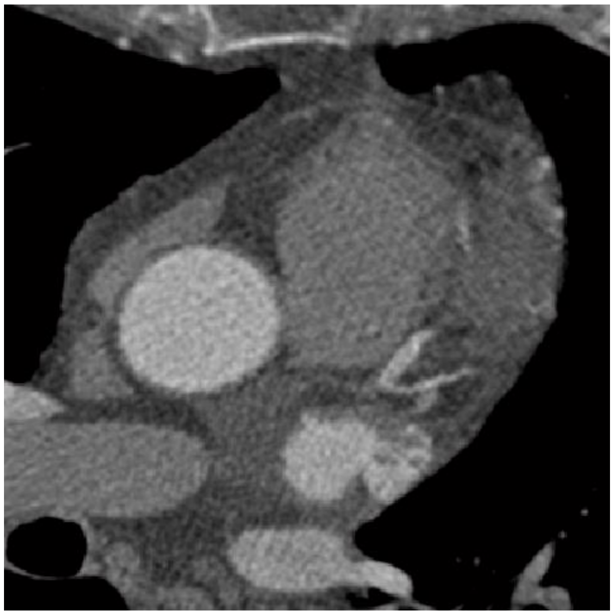


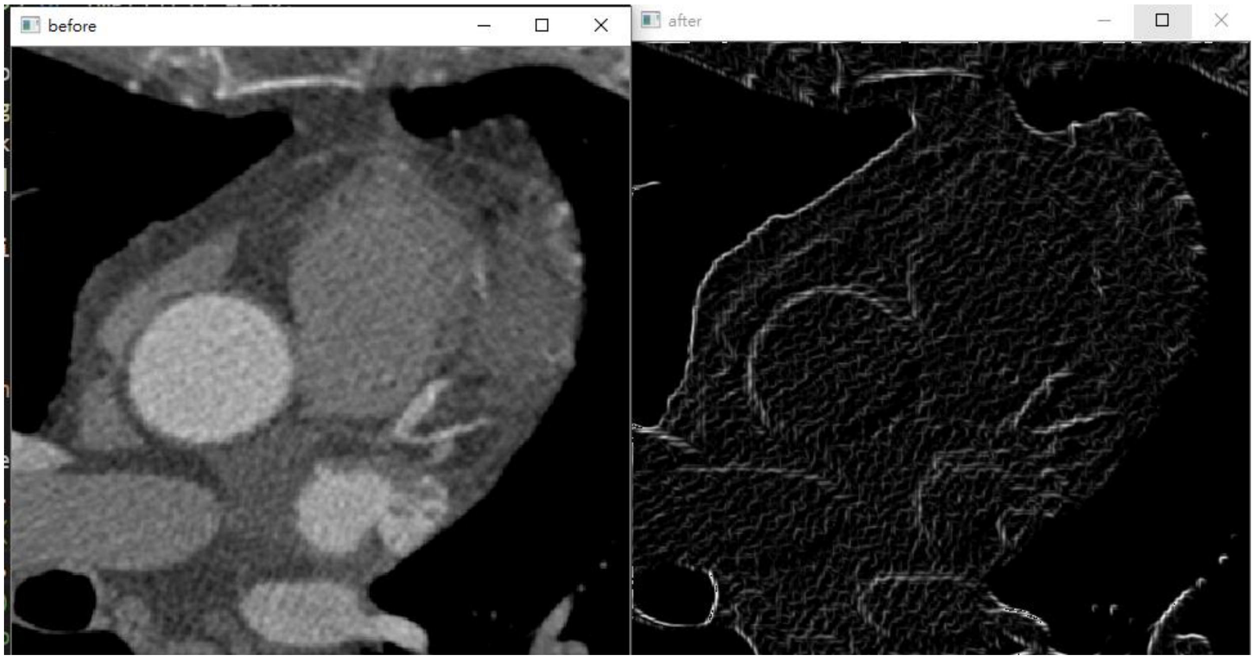
Canny 用 90,50 很明顯沒有用。



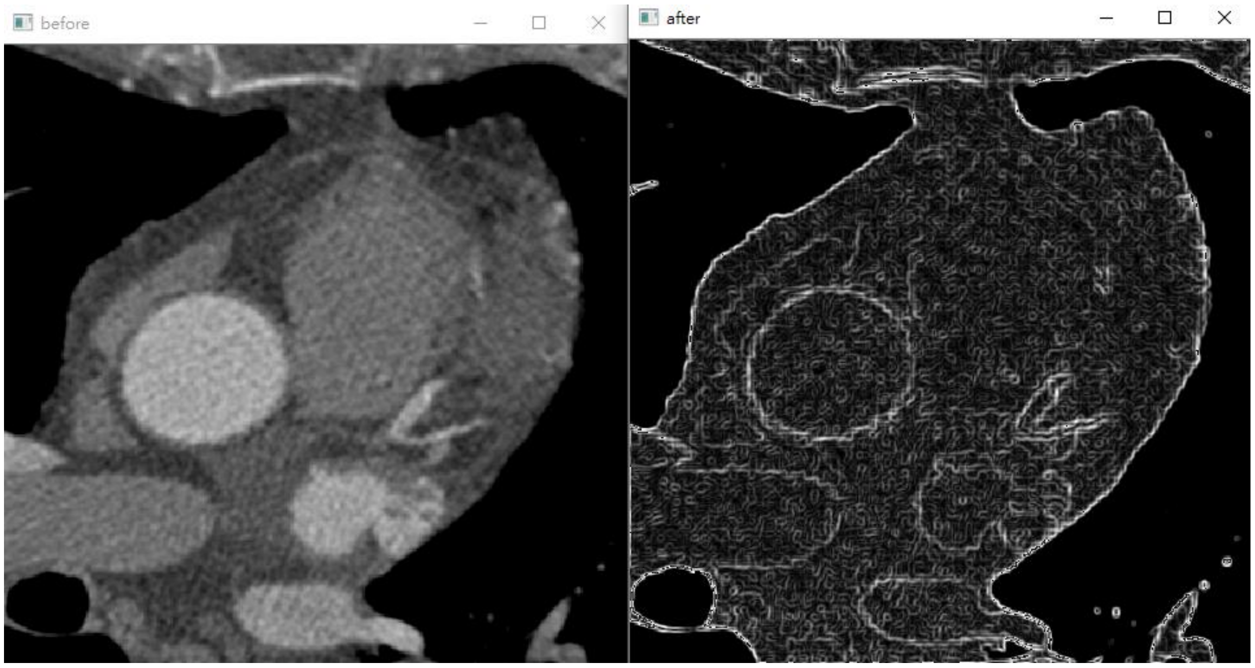
30,20 就挺好的。

第四張圖

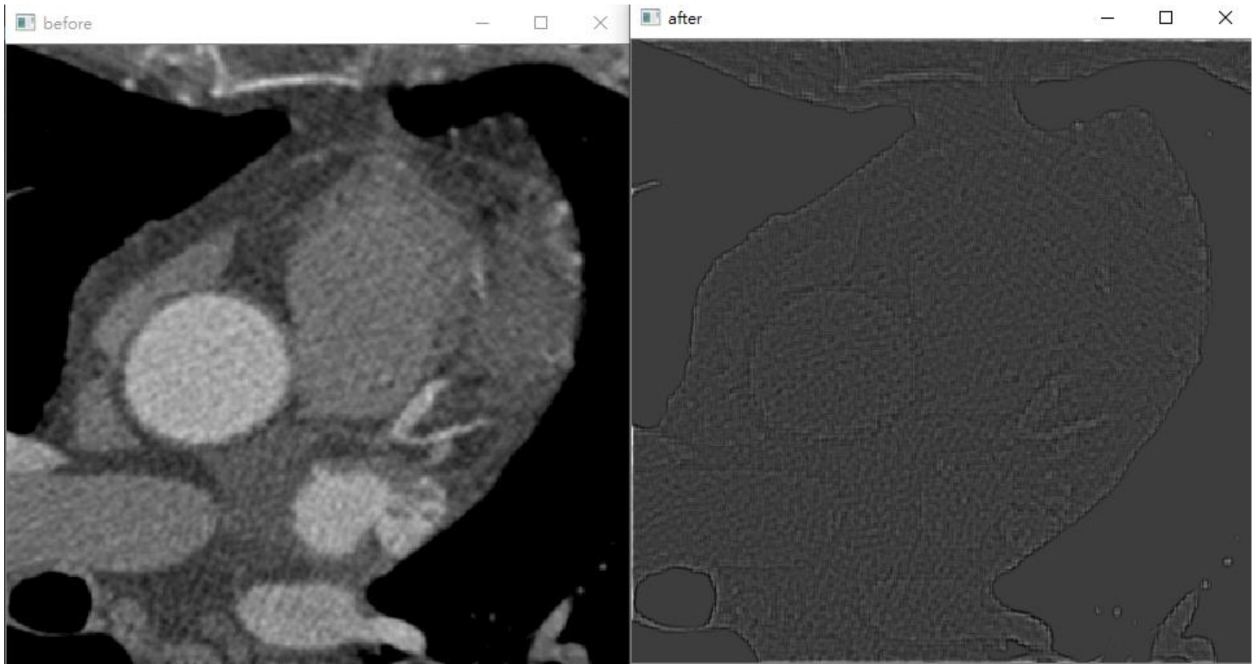




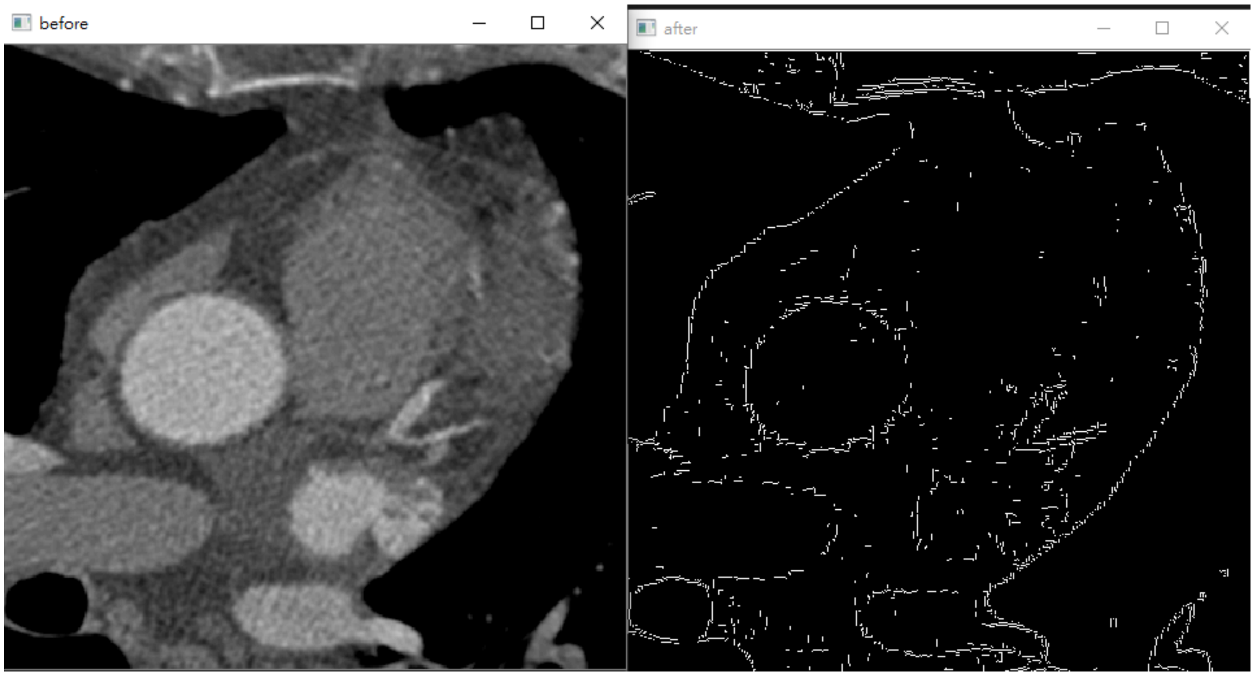
一樣先用 prewitt ，雜訊真的很多，線條也不明顯。



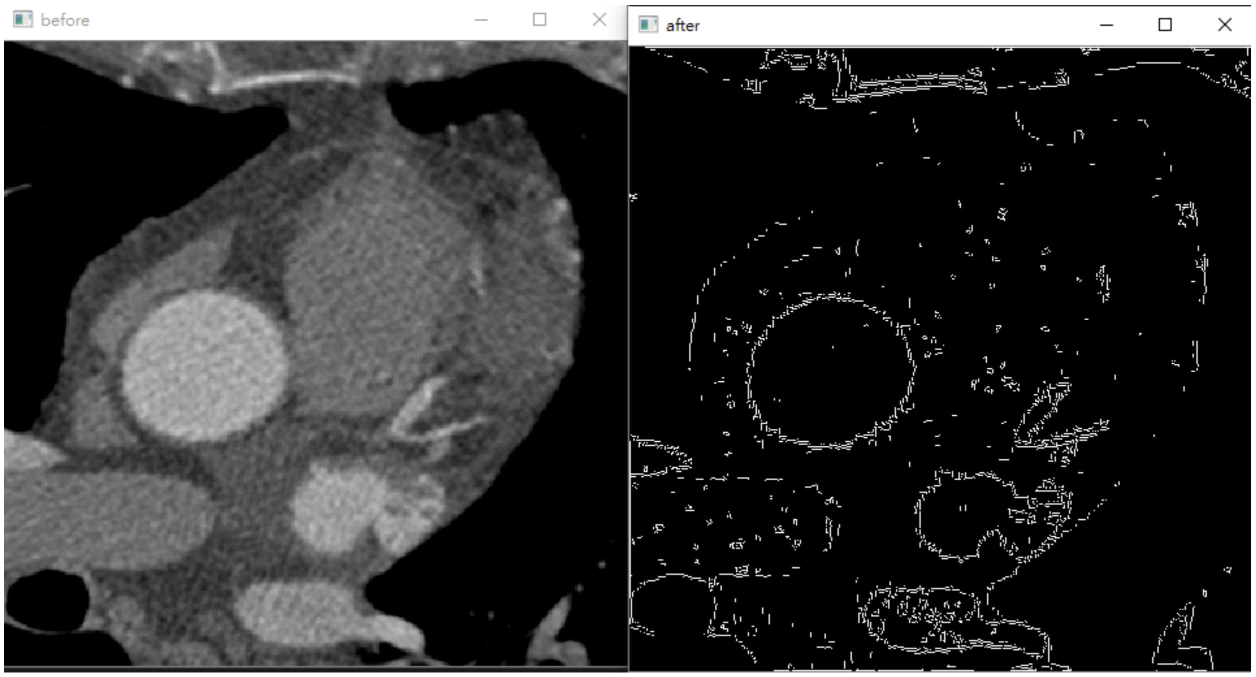
Sobel 的話一樣有雜訊但線條明顯很多。



LoG 則是指有明顯的邊才比較有感覺。



直接 canny 用 90,70 的情況，很多邊還是沒有跑出來。



做完強化對比後的情況，少的邊數又更多了。

Discussions:

實驗證明的是，如果原圖品質不好有雜訊或者是過於不清楚的話，直接做邊緣偵測效果會非常差，還有 sobel 好像整體來說沒有哪次是輸 prewitt 的。LoG 則是我不確定到底要怎麼用會比較好。留下來的問題是 canny 的 threshold 到底要怎麼選會比較好。

#pic.py

調整下面這行即可讀取不同圖片，用的方法都一樣。

```
img = cv2.imread('p2im4.png', cv2.IMREAD_GRAYSCALE)
```

```
import numpy as np
import cv2
import itertools
import math

np.set_printoptions(threshold=np.inf)
def gamma_trans(channel, gamma):
    new_channel = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    table = np.zeros(256, dtype=int)
    tmp = channel.flatten()
    for i in range(256):
        table[i] = int(pow(i/255.0, gamma) * 255.0)

    for i in range(channel.shape[0]*channel.shape[1]):
        new_channel[i] += table[tmp[i]]

    new_channel = new_channel.reshape(channel.shape[0], channel.shape[1])
    return new_channel
def increase_contrast(channel, x):
    new_channel = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    new_channel = new_channel.reshape(channel.shape[0], channel.shape[1])
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1]
)):
        if(channel[i][j] < 128):
            new_channel[i][j] = channel[i][j] - x if channel[i][j] - x > 0 else 0
        else:
            new_channel[i][j] = channel[i][j] + x if channel[i][j] + x < 255 else
 255
    return new_channel

def gausian_filter(channel):
    #use the 3*3 box size
    gf = [0.045, 0.122, 0.045, 0.0122, 0.332, 0.122, 0.045, 0.122, 0.045]
    new_channel = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    new_channel = new_channel.reshape(channel.shape[0], channel.shape[1])
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1]
)):
        cnt = 0
```

```python
        for k, l in itertools.product(range(-1,2), range(-1,2)):
            if i+k in range(channel.shape[0]) and j+l in range(channel.shape[1]):
                new_channel[i][j] += (channel[i+k][j+l]*gf[cnt])
            cnt += 1
    return new_channel

def prewitt_filter(channel):
    #use the 3*3 box size
    pf_horizon = [-1, -1, -1, 0, 0, 0, 1, 1, 1]
    pf_vertical = [-1, 0, 1, -1, 0, 1, -1, 0, 1]
    horizon = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    horizon = horizon.reshape(channel.shape[0], channel.shape[1])
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1]
)):
        cnt = 0
        for k, l in itertools.product(range(-1,2), range(-1,2)):
            if i+k in range(channel.shape[0]) and j+l in range(channel.shape[1]):
                horizon[i][j] += (channel[i+k][j+l]*pf_horizon[cnt])
            cnt += 1

    vertical = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    vertical = vertical.reshape(channel.shape[0], channel.shape[1])
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1]
)):
        cnt = 0
        for k, l in itertools.product(range(-1,2), range(-1,2)):
            if i+k in range(channel.shape[0]) and j+l in range(channel.shape[1]):
                vertical[i][j] += (channel[i+k][j+l]*pf_vertical[cnt])
            cnt += 1
    new_channel = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    new_channel = new_channel.reshape(channel.shape[0], channel.shape[1])
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1]
)):
        #1-norm
        new_channel[i][j] = math.sqrt(vertical[i][j]**2 + horizon[i][j] **2)
        #2-norm
        #new_channel[i][j] = abs(vertical[i][j]) + abs(horizon[i][j])
    return new_channel
def sobel_filter(channel):
    #use the 3*3 box size
    pf_horizon = [1, 2, 1, 0, 0, 0, -1, -2, -1]
    pf_vertical = [1, 0, -1, 2, 0, -2, 1, 0, -1]
    horizon = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    horizon = horizon.reshape(channel.shape[0], channel.shape[1])
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1]
)):
        cnt = 0
        for k, l in itertools.product(range(-1,2), range(-1,2)):
            if i+k in range(channel.shape[0]) and j+l in range(channel.shape[1]):
                horizon[i][j] += (channel[i+k][j+l]*pf_horizon[cnt])
            cnt += 1
    vertical = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    vertical = vertical.reshape(channel.shape[0], channel.shape[1])
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1]
)):
        cnt = 0
        for k, l in itertools.product(range(-1,2), range(-1,2)):
            if i+k in range(channel.shape[0]) and j+l in range(channel.shape[1]):
                vertical[i][j] += (channel[i+k][j+l]*pf_vertical[cnt])
            cnt += 1
    new_channel = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    new_channel = new_channel.reshape(channel.shape[0], channel.shape[1])

    global angle
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1]
)):
        #calculate angle here
        #and we sort them to 4 class
        angle[i][j] = math.atan2(horizon[i][j],vertical[i][j])*180/math.pi
        angle[i][j] = angle[i][j] % 180
        if angle[i][j] > 0 and angle[i][j] < 22.5:
            angle[i][j] = 0
        elif angle[i][j] >= 22.5 and angle[i][j] < 67.5 :
            angle[i][j] = 45
        elif angle[i][j] >=67.5 and angle[i][j] < 112.5 :
            angle[i][j] = 90
        elif angle[i][j] >= 112.5 and angle[i][j] < 157.5:
            angle[i][j] = 135
        else:
            angle[i][j] = 0
        #1-norm
        new_channel[i][j] = math.sqrt(vertical[i][j]**2 + horizon[i][j] **2)
        #2-norm
        #new_channel[i][j] = abs(vertical[i][j]) + abs(horizon[i][j])

    return new_channel
def laplace(channel):
    #use the 3*3 box size
    lp = [-1, -1, -1, -1, 8, -1, -1, -1, -1]
```

```python
    new_channel = np.zeros(channel.shape[0]*channel.shape[1], dtype=int)
    new_channel = new_channel.reshape(channel.shape[0], channel.shape[1])
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1])):
        cnt = 0
        for k, l in itertools.product(range(-1,2), range(-1,2)):
            if i+k in range(channel.shape[0]) and j+l in range(channel.shape[1]):
                new_channel[i][j] += (channel[i+k][j+l]*lp[cnt])
            cnt += 1
    return new_channel

def LoG_filter(channel):
    new_channel = gausian_filter(channel)
    new_channel = laplace(new_channel)
    #then normalize
    max = -1
    min = 256
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1])):
        if new_channel[i][j] > max:
            max = new_channel[i][j]
        if new_channel[i][j] < min:
            min = new_channel[i][j]
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1])):
        new_channel[i][j] = 255*(new_channel[i][j]-min)/(max-min)
    for i, j in itertools.product(range(channel.shape[0]), range(channel.shape[1])):
        new_channel[i][j] = new_channel[i][j] if new_channel[i][j] > threshold else 0
    return new_channel

def Canny_edge_detector(img,bt,lt):
    #first step:blur
    img = gausian_filter(img)
    #second step:calculate  magnitude and angle
    img = sobel_filter(img)#use 1 norm
    #third step:thin the edge
    for i, j in itertools.product(range(img.shape[0]), range(img.shape[1])):
        if angle[i][j] == 0:
            while j+1 in range(img.shape[1]) and img[i][j+1] >= img[i][j]:
                img[i][j] = 0
                j = j+1
            while j-1 in range(img.shape[1]) and img[i][j-1] >= img[i][j]:
                img[i][j] = 0
                j = j-1
        if angle[i][j] == 45:
            while i-
1 in range(img.shape[0]) and j+1 in range(img.shape[1]) and img[i-
1][j+1] >= img[i][j]:
                img[i][j] = 0
                i = i-1
                j = j+1
            while i+1 in range(img.shape[0]) and j-
1 in range(img.shape[1]) and img[i+1][j-1] >= img[i][j]:
                img[i][j] = 0
                i = i+1
                j = j-1
        if angle[i][j] == 90:
            while i+1 in range(img.shape[0]) and img[i+1][j] >= img[i][j]:
                img[i][j] = 0
                i = i+1
            while i-1 in range(img.shape[0]) and img[i-1][j] >= img[i][j]:
                img[i][j] = 0
                i = i-1
        if angle[i][j] == 135:
            while i+1 in range(img.shape[0]) and j+1 in range(img.shape[1]) and i
mg[i+1][j+1] >= img[i][j]:
                img[i][j] = 0
                i = i+1
                j = j+1
            while i-1 in range(img.shape[0]) and j-
1 in range(img.shape[1]) and img[i-1][j-1] >= img[i][j]:
                img[i][j] = 0
                i = i-1
                j = j-1
    #choose strong edge
    for i, j in itertools.product(range(img.shape[0]), range(img.shape[1])):
        if img[i][j] >= bt:
            img[i][j] = 255
        if img[i][j] < lt:
            img[i][j] = 0
    #check whether weak edge is connected to strong edge
    for i, j in itertools.product(range(img.shape[0]), range(img.shape[1])):
        if img[i][j] == 255 or img[i][j] == 0:
            continue
        for k, l in itertools.product(range(-1,2), range(-1,2)):
            if i+k in range(img.shape[0]) and j+l in range(img.shape[1]):
                if img[i+k][j+l] == 255:
                    img[i][j] = 255
```

```python
                    break
        img[i][j] = 0 if img[i][j] <255 else 255
    return img

def threshold(img,num):
    for i, j in itertools.product(range(img.shape[0]), range(img.shape[1])):
        if img[i][j] < num:
            img[i][j] = 0
    return img

#read image here
img = cv2.imread('p2im4.png', cv2.IMREAD_GRAYSCALE)
#define angle here
angle = np.zeros(img.shape[0]*img.shape[1], dtype=float)
angle = angle.reshape(img.shape[0], img.shape[1])
img2 = increase_contrast(img,30)
#img2 = sobel_filter(img)
#img2 = LoG_filter(img)
img2 = Canny_edge_detector(img2,90,70)
img2 = np.array(img2,dtype=np.uint8)

#show image
cv2.imshow('before', img)
cv2.imshow('after', img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```