Edge Detection

電機所　609415159　江昀融

Date due:2020/12/28

Date handed in:2020/12/28

Technical description:

使用 opencv 用於圖片的基本讀寫與顯示
numpy 用於進行陣列操作
math 用於影像處理的公式計算
matplotlib 用於圖表的顯示

```
import cv2 as cv
import numpy as np
import math
from matplotlib import pyplot as plt
```

cv.imread 讀取輸入圖片
並且讀取圖片長寬並顯示
順便將影像轉為灰階

```
#choose read which image
img = cv.imread('image1.jpg', cv.IMREAD_GRAYSCALE)
#img = cv.imread('image2.jpg', cv.IMREAD_GRAYSCALE)
#img = cv.imread('image3.jpg', cv.IMREAD_GRAYSCALE)

#print height and width of input image
height, width = img.shape
print(type(img.shape))
print(img.shape)
print("height = ", height)
print("width = ", width)
print("**************")
```

並且針對兩種不同的影像處理技術進行分別的動作:

1.Laplacian edge detection

生成 laplacian kernel

```
#creat new image for output & laplacian kernel
lap_s = np.zeros( (height-2, width-2), np.float )
l_kernel = np.array([[ 0.0,-1.0, 0.0],
                     [-1.0, 4.0,-1.0],
                     [ 0.0,-1.0, 0.0]])
```

使用 laplacian kernel
進行 convolution 計算

```
#run convolution by laplacian kernel
print("wait for convolution ......")

for i in range(height-2):
    for j in range(width-2):
            lap_s[i, j] = np.sum(l_kernel * img[i:i+3, j:j+3])
            if(lap_s[i, j] < 0):lap_s[i, j] = 0
            if(lap_s[i, j] >= 255):lap_s[i, j] = 255

print("convolution done")
```

顯示圖片

```
cv.imshow('source', img)
cv.imshow('laplacian_sharpening(uint8)', lap_s.astype(np.uint8))
```

2.Sobel edge detection

設定參數
σ = 1
gaussian kernel = 3*3
生成 sobel 的 Gx&Gy

```
#set parameter
sigma = 1.0;
kernel = 3;
gauss = np.zeros( (height-2, width-2), np.float )
Gx = np.zeros( (height-4, width-4), np.float )
Gy = np.zeros( (height-4, width-4), np.float )
sobel = np.zeros( (height-4, width-4), np.float )
G_kernel = np.zeros( (kernel, kernel), np.float )

Gx_kernel = np.array([[-1.0, 0.0, 1.0],
                      [-2.0, 0.0, 2.0],
                      [-1.0, 0.0, 1.0]])

Gy_kernel = np.array([[-1.0,-2.0,-1.0],
                      [ 0.0, 0.0, 0.0],
                      [ 1.0, 2.0, 1.0]])
```

以高斯公式生成 gaussian kernel

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

```
#Create gaussian kernel
for i in range(kernel):
    for j in range(kernel):
        G_kernel[i,j] =math.exp(-1*((i-int(kernel/2))**2 + (j-int(kernel/2))**2)/(2*sigma**2))
```

以 gaussian kernel 進行 convolution 即高斯模糊

```
#blur image by gaussian kernel
print("wait for convolution ......")
for i in range(height-2):
    for j in range(width-2):
            gauss[i, j] = np.sum(G_kernel * img[i:i+3, j:j+3])/np.sum(G_kernel)
```

以 Gx & Gy 進行 convolution 計算
分別得到 x 方向的邊緣
與 y 方向的邊緣
最後再將兩者平方相加取根號
得出最後的 sobel 結果

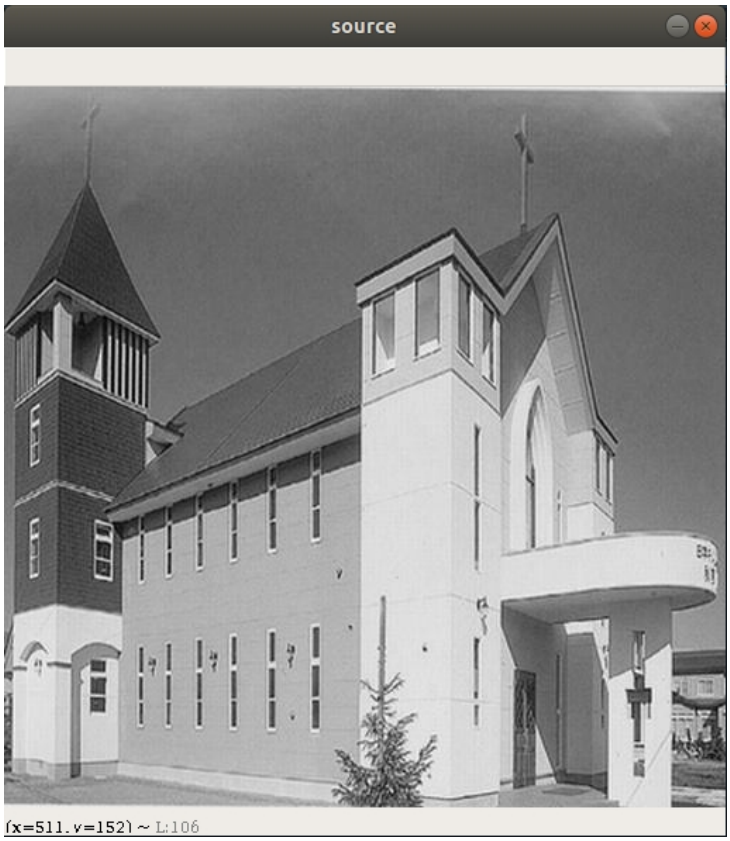
```
#detect edge by sobel kernel
for i in range(height-4):
    for j in range(width-4):
            Gx[i, j] = np.sum(Gx_kernel * gauss[i:i+3, j:j+3])
            if(Gx[i, j] < 0):Gx[i, j] = 0
            elif(Gx[i, j] >= 255):Gx[i, j] = 255
            Gy[i, j] = np.sum(Gy_kernel * gauss[i:i+3, j:j+3])
            if(Gy[i, j] < 0):Gy[i, j] = 0
            elif(Gy[i, j] >= 255):Gy[i, j] = 255
            sobel[i, j] = math.sqrt(Gx[i, j]**2 + Gy[i, j]**2)
            if(sobel[i, j] >= 255):sobel[i, j] = 255
```

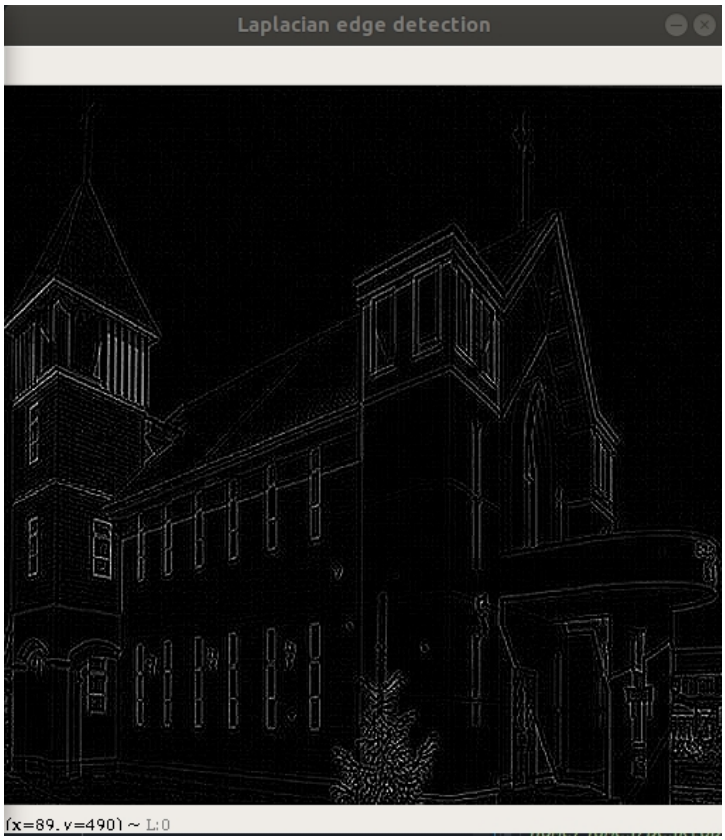
顯示圖片

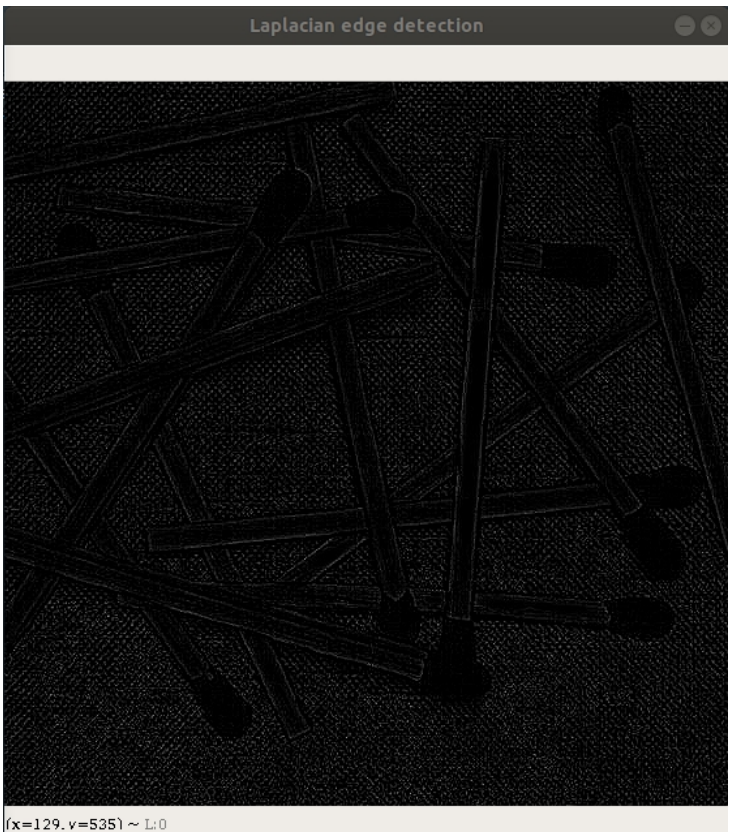
```
#show image
cv.imshow('source', img)
cv.imshow('Gauss', gauss.astype(np.uint8))
cv.imshow('Gx', Gx.astype(np.uint8))
cv.imshow('Gy', Gy.astype(np.uint8))
cv.imshow('Sobel', sobel.astype(np.uint8))
```

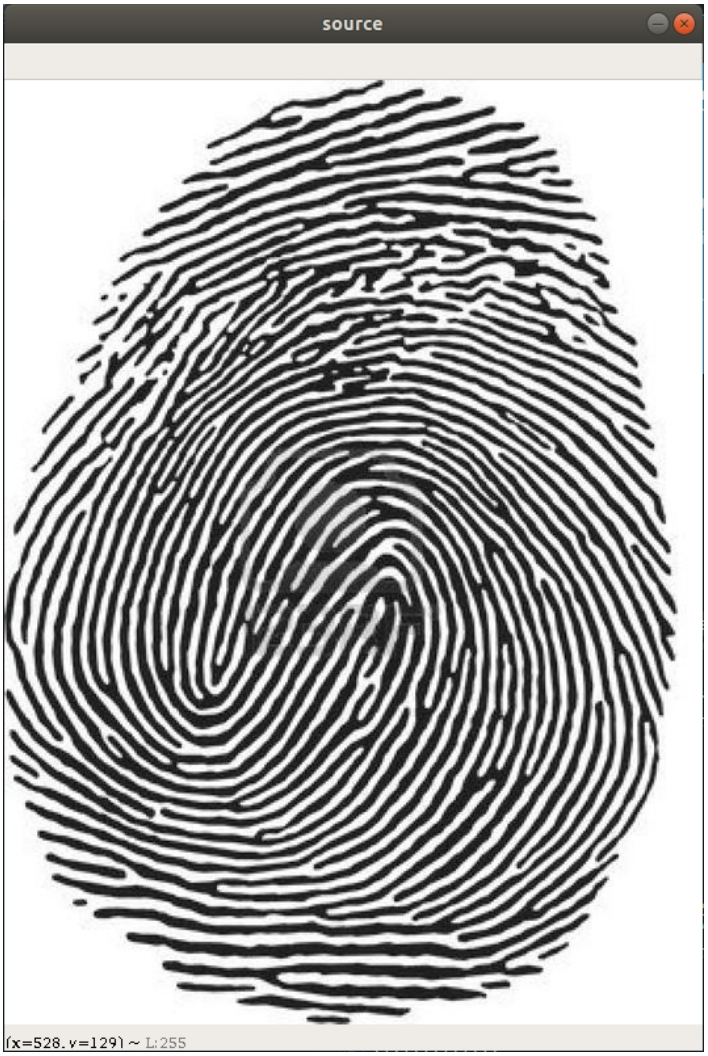
Experimental results:

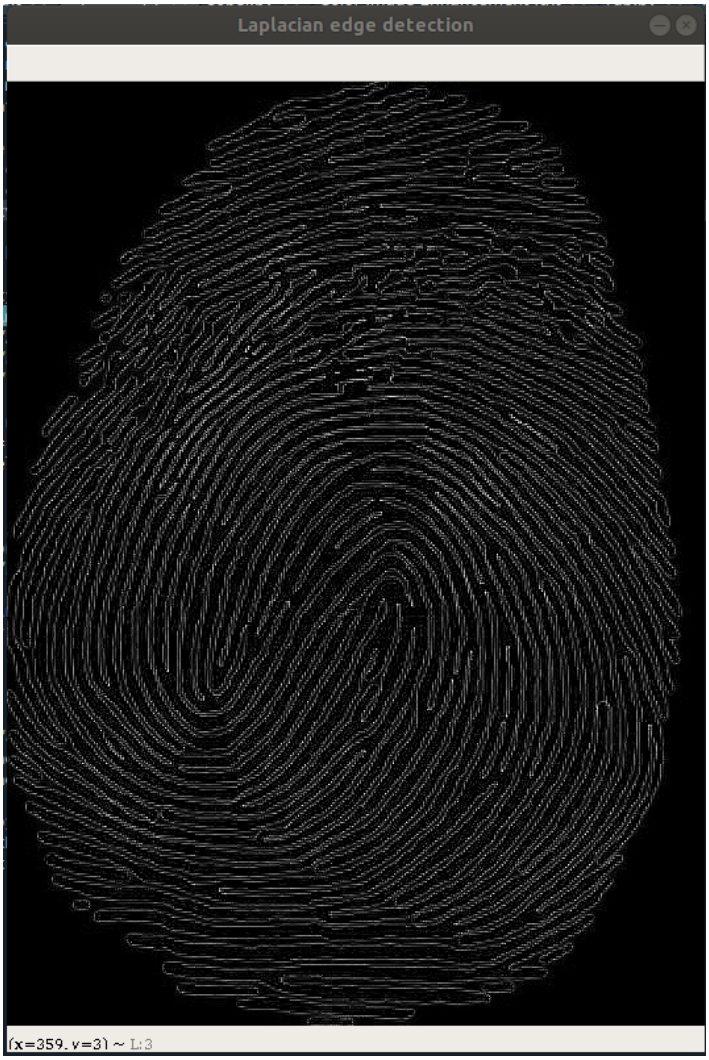
1.Laplacian edge detection



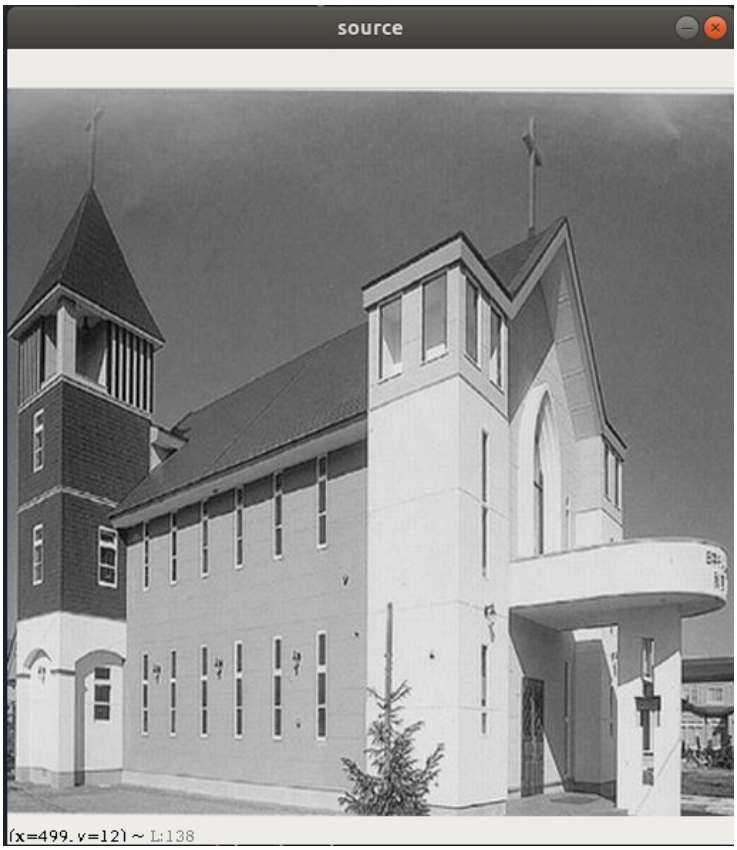


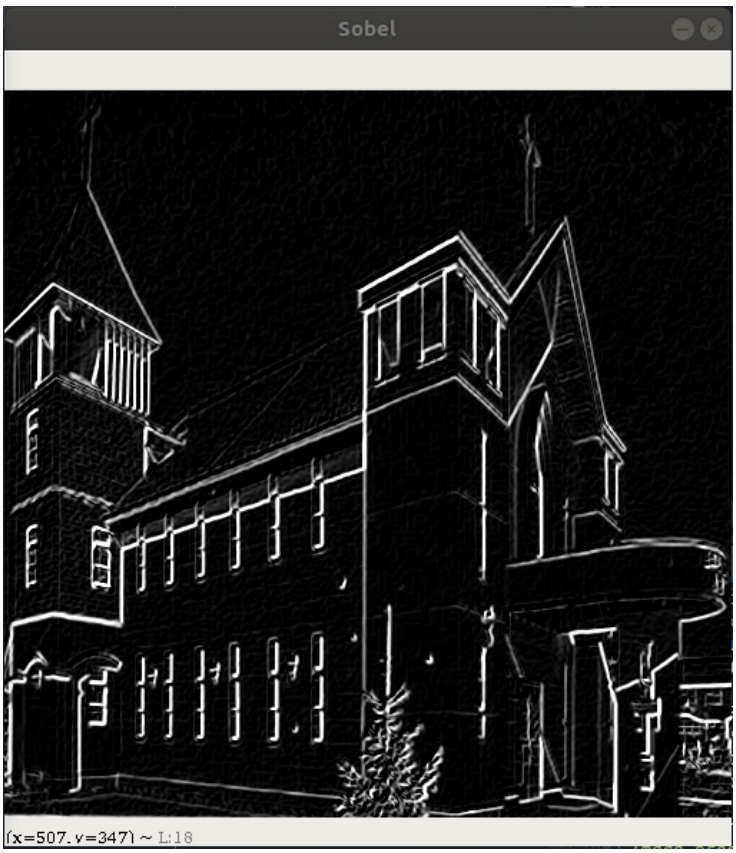


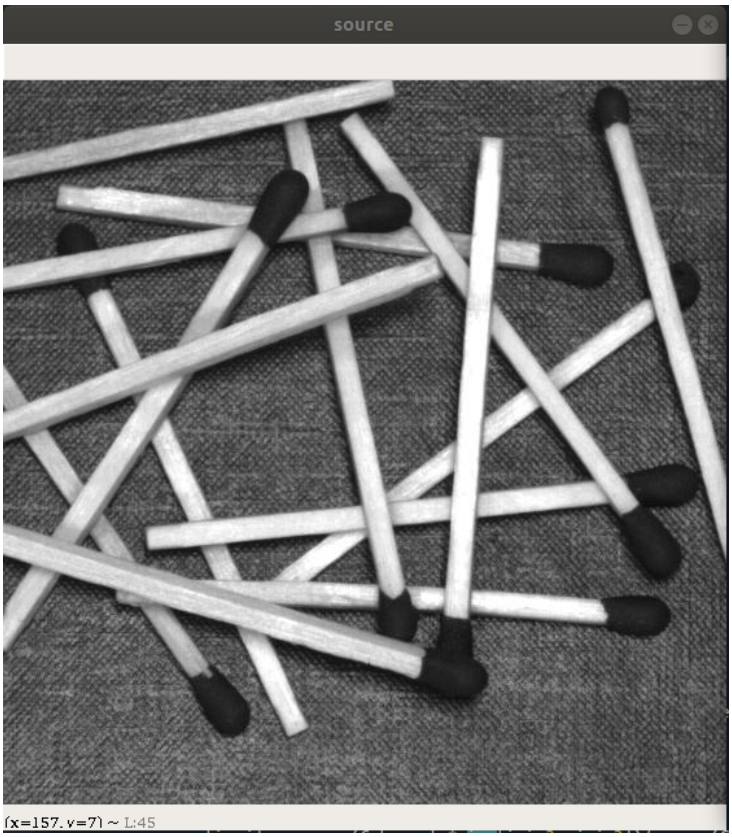


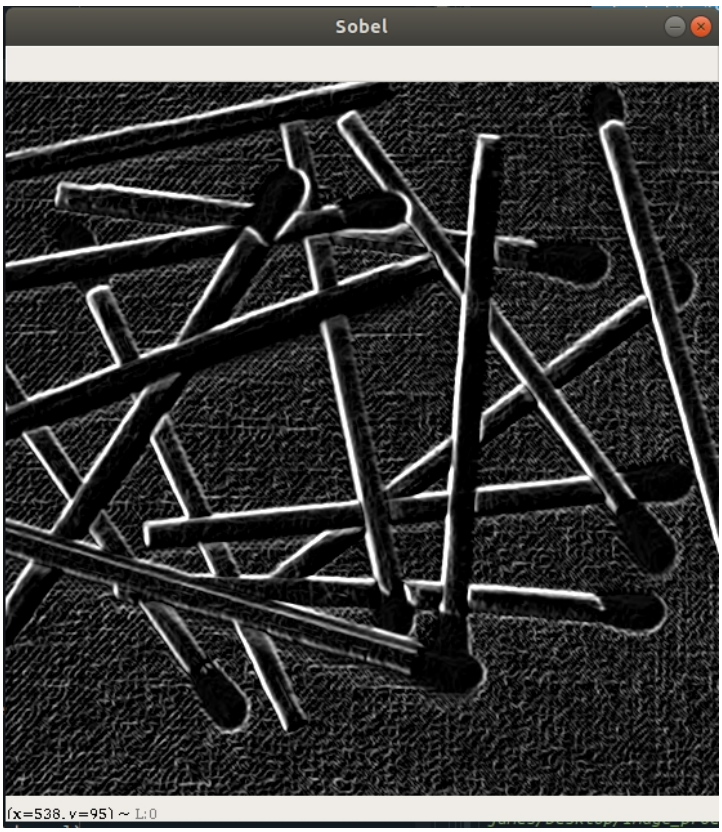


2.Sobel edge detection

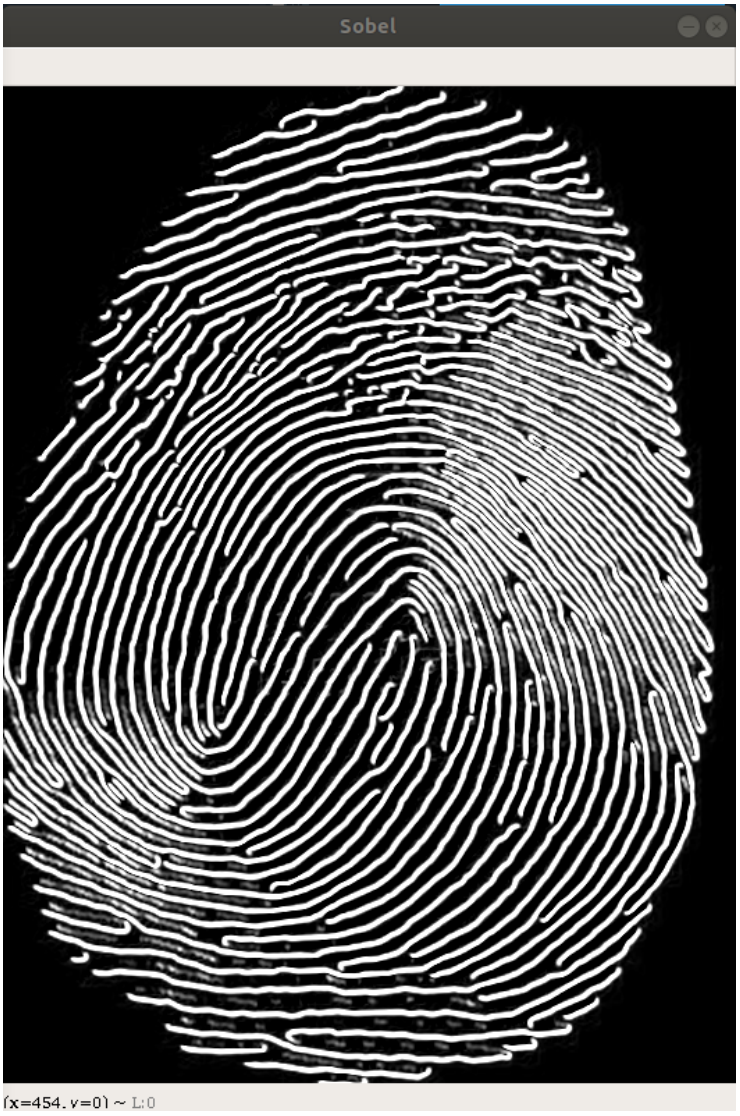





source
(x=157, y=7) ~ L:45


Sobel
(x=538, y=95) ~ L:0


source


Sobel
(x=454, y=0) ~ L:0

Discussions:

這次的邊緣偵測給的輸入圖片雖然是 rgb 彩色圖，但經過測試三通道直接為相等，所以就將它轉為灰階圖。化雖如此，但要做邊緣偵測在灰階的圖片是比較好運作的。這次的邊緣偵測方法選用 Laplacian 與 Sobel 運算兩種方法。

Laplacian 在之前的影像增強裡面有做過，是以空間座標處理近似數學方程式的結果，當初為了要將特徵擷取出來，並在原圖上突顯，所以在 kernel 的內容會有些許的不同。但這次屬於邊緣偵測，所以不需要再將原圖加上去，只要以最原始的 Laplacian kernel 做卷積計算就好了。所以在實作上省去了不少時間。內容屬於簡單的卷積計算，並且此種方式一樣會將雜訊無分別的做放大。

第二種是使用 Sobel 運算子的方法，相較於 Laplacian，Sobel 有更複雜的步驟，所作出來的效果也與 Laplacian 有著些許的不同。在 Sobel 的執行步驟中，必須先用 gaussian kernel 進行影像模糊，雖然 Sobel kernel 也有平滑化的效果，但先以 gaussian 模糊後可以有效的減輕雜訊的影響。經過了模糊步驟之後，以不同方向的 Sobel 濾波器去運算，即為 x 與 y 兩種方向，不同的方向有不同表示的 kernel，通常水平與垂直是最常用的。以兩方向的 kernel 進行運算後將結果平方相加並開根號，所得出來的結果即是 Sobel 邊緣偵測的結果。當然前面的運算有很多種 kernel 可以選擇，而我選擇了水平與垂直的方向，結合可以同時偵測兩種方向的邊緣資訊，所得出的結果也是很不錯了。

經過比較後 Sobel 所得出得邊緣比起 Laplacian 有更清楚的跡象，在性能方面 Sobel 是更勝一籌的，當然上課還有教到很多的邊緣偵測方法，像是 canny 或 hough transform 等方法，但這些演算法比起前面兩種都更加的複雜，但做出來的結果也會更加的顯著，所以針對所要解決的問題複雜度可以選擇相對應演算法，得到做適合的結果。

References and Appendix:

https://en.wikipedia.org/wiki/Sobel_operator
https://blog.csdn.net/tianhai110/article/details/5663756
https://chtseng.wordpress.com/2016/12/05/opencv-edge-detection%E9%82%8A%E7%B7%A3%E5%81%B5%E6%B8%AC/
https://medium.com/@bob800530/python-gaussian-filter-%E6%A6%82%E5%BF%B5%E8%88%87%E5%AF%A6%E4%BD%9C-676aac52ea17