DIP HW2　R06944023 吳尚真

PROBLEM 1: EDGE DETECTION

a) 做 1st order edge detection，輸出E_1

1) Your motivation and approach:

先算出 row 跟 column 的 gradients。根據講義，可分為 3 種不同的算法，分別為: 考慮 3 個點、考慮 4 個點、考慮 9 個點(Prewitt, Sobel)。得到 gradient 後，計算相對應的 histogram 並繪出。根據 histogram 決定 Threshold，小於 T 的地方設為 0，其他設為 1。

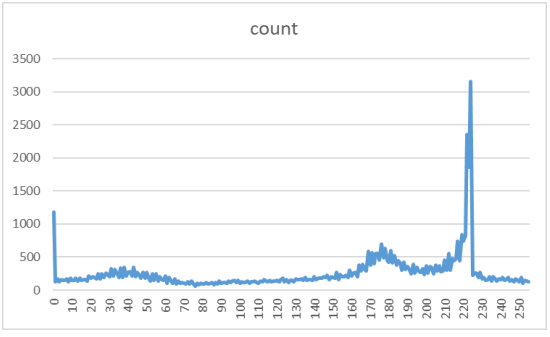
2) Original images

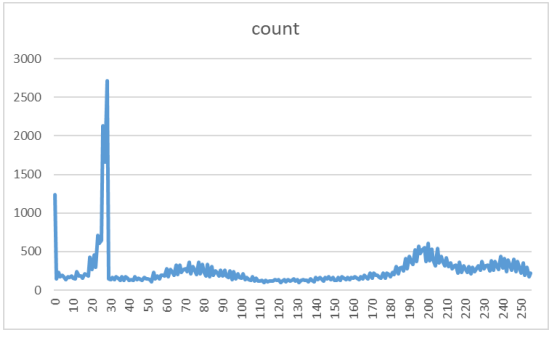


3) Output images

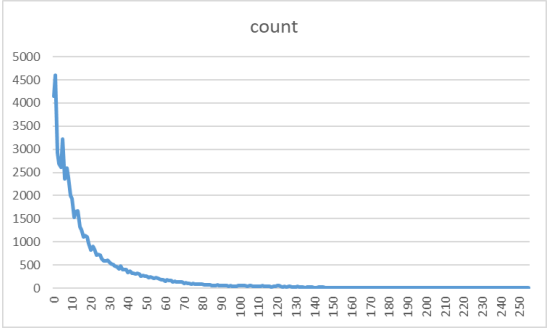
當用 4 種不同方法算 sample 1 的 gradient map 後，畫出的 histogram 如下所示:

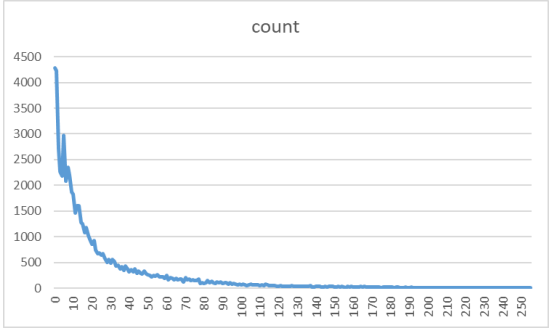
使用 Prewitt(左)　　使用 Sobel(右)





考慮 3 個點(左)　　考慮 4 個點(右)





由上面四張 histogram 的圖可以發現，考慮 3 個點 與 考慮 4 個點 gradient map 的 histogram 很相近，所以我只用其中一個輸出(T 取 40)。因為算出的 gradient

代表是灰階值的差異，所以 T 應該要試著讓較低的值跟較高的值分開，以確保取到的是真的 edge。Prewitt 的 T 取 220，Sobel 取 180。下面分別列出不同方法、不同 T 的結果。

Prewitt T=210　　Prewitt T=220　　Prewitt T=230

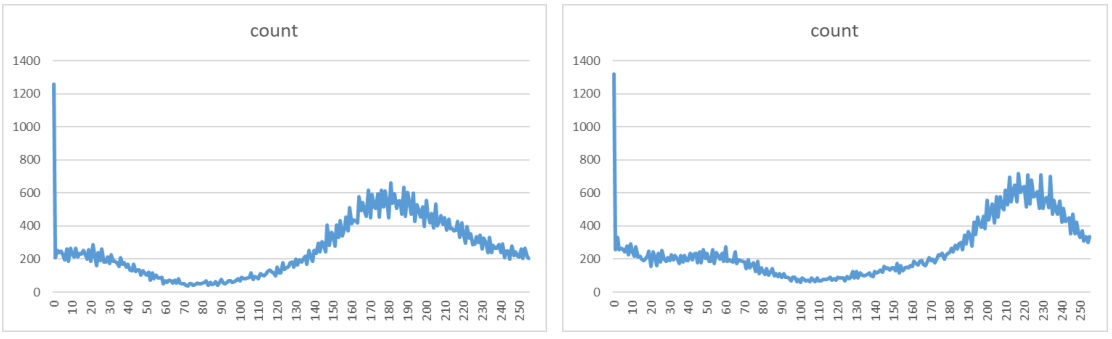
Prewitt T=210 時，屋頂跟天空會被誤判為 edge，T=220 後，天空還是被誤判，當 t=230 時，雖然屋頂跟天空都沒被誤判，但是窗戶的細節不佳。

Sobel T=180　　Sobel T=220　　考慮 3 個點 T=40

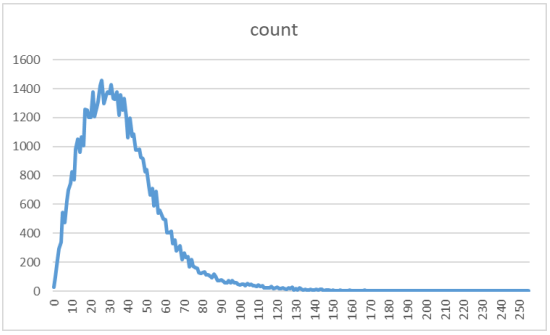
在 Sobel T=180 和 T=220 時，屋頂都會被誤判。相較之下，考慮 3 個點 T=40 時，不但天空、屋頂都沒被誤判，窗戶跟樹的細節也都顯現的很好!

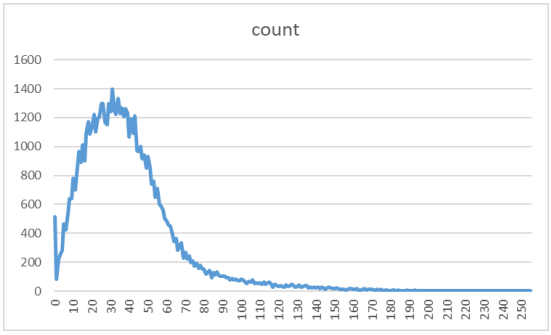
當用 4 種不同方法算 sample 2 的 gradient map 後，畫出的 histogram 如下所示:

使用 Prewitt(左)　　使用 Sobel(右)



考慮 3 個點(左)　　考慮 4 個點(右)

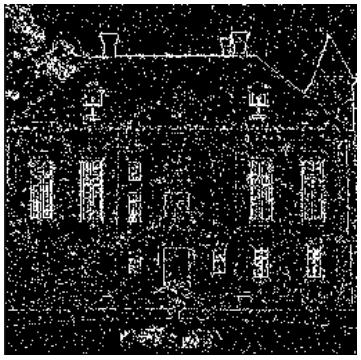




比較 sample1 gradient map 的 histogram 跟 sample2 的，可以發現因為 noise 讓原來比較尖銳(高、窄)的 histogram 變得比較圓弧(低、寬)。下面分別列出不同方法、不同 T 的結果。可以發現當 T 取的越高，原圖 uniform noise 輸出的雜點就越少，但 T 也不能取太大，否則真正的 edge 也會消失。

Prewitt T=100

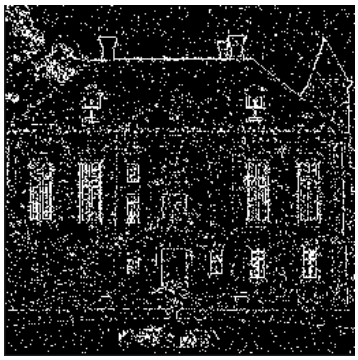
Prewitt T=180



Prewitt T=220

Sobel T=190

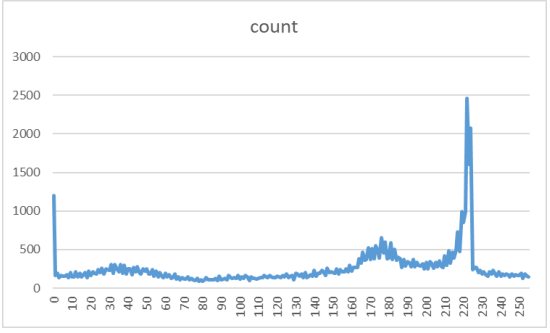
Sobel T=220



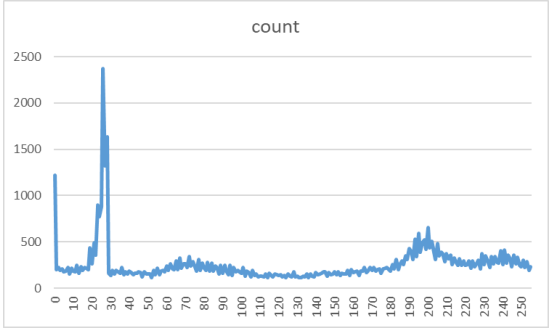
考慮 3 個點 T=80

當用 4 種不同方法算 sample 3 的 gradient map 後，畫出的 histogram 如下所示:

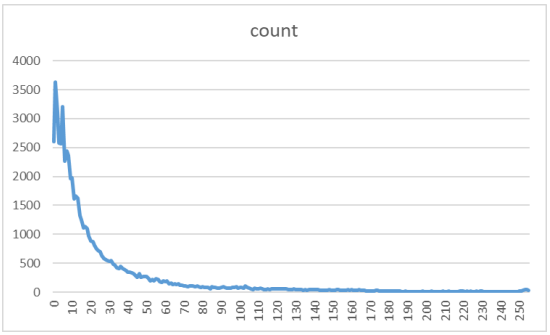
使用 Prewitt(左)

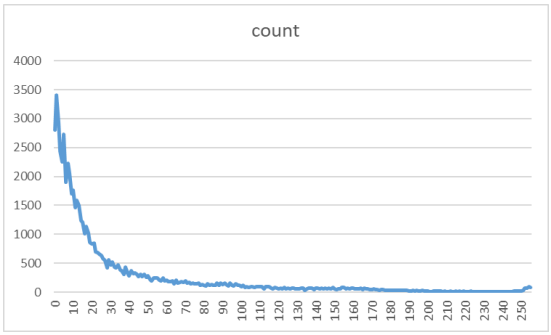


使用 Sobel(右)



考慮 3 個點(左)　　考慮 4 個點(右)





可以發現 sample 3 gradient map 的 histogram 與 sample1 的很像，因為比起 sample2 的 uniform noise，sample3 加上 salt and pepper noise 後，整體的 pixel value 還是類似 sample1。所以直接使用跟在 sample1 一樣的方法和 T。

sample1E1.raw　　sample2E1.raw　　Sample3E1.raw

4) Discussion of results:

上面三張輸出圖都是使用'考慮 3 個點 (0-90 度)' 這個方法(唯有 sample2 使用的 T 不一樣)。加了 uniform noise 的 sample2 在 gradient map 的 histogram 就與 sample1 不一樣，所以可以預期他的 output (sample2E1.raw) 也還是會有一些 noise；加了 salt and pepper noise 的 sample3 雖然在 gradient map 的 histogram 與 sample1 很像，但是他的 output (sample3E1.raw)依舊沒辦法濾掉這些 noise。不過若只比較 sample2E1.raw 與 sample3E1.raw 可以發現，前者的 noise 看起來比較輕微，但 edge 的線條較不明顯 (因為 T 調比較大的關係)，後者的 noise 看起來比較多，但 edge 的線條基本上與 sample1E1.raw 的結果差不多。

b) 做 2nd order edge detection，輸出E_2

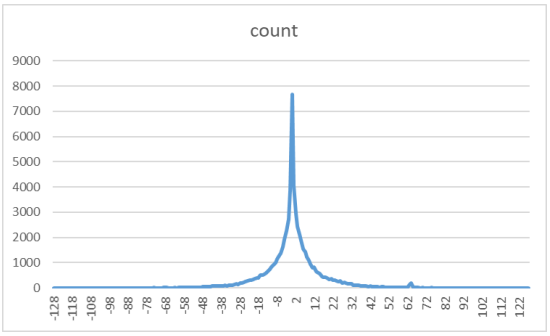
1) Your motivation and approach:

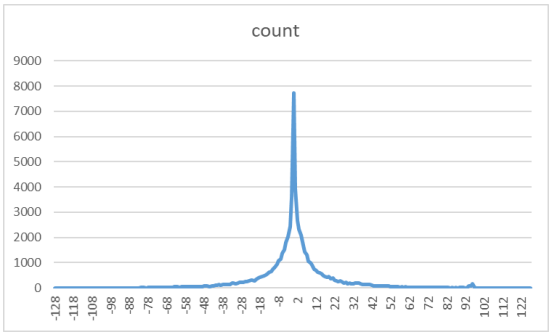
分別使用 4-neighbor 與 8-neighbor 的 Laplacian impulse response 當作 mask 來計算 gradient map，接著畫出 gradient map 的 histogram，要注意的是，做二階後的結果是有正、有負、有 0，所以在 count histogram 的時候，可以+128，以避免對不到 list 的 index。 有了 histogram 後，就可以決定 T。原先 gradient map 的值取絕對值後<=T 的點就是有可能為 edge 的點，再比較這些點的左右鄰居有沒有出現 zero-crossing (gradient map 的 pixel value 一正一負)。

3) Output images

用 4-neighbor 與 8-neighbor 算 sample1 的 gradient map 的 histogram 如下所示:

使用 4-neighbor (左)　　使用 8-neighbor (右)





使用 4-neighbor，採用不同的 T 的結果如下:

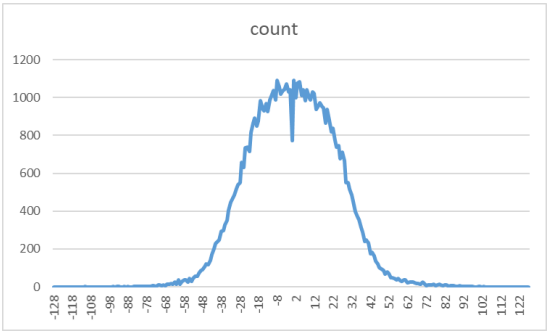
T=10　　T=15　　T=20

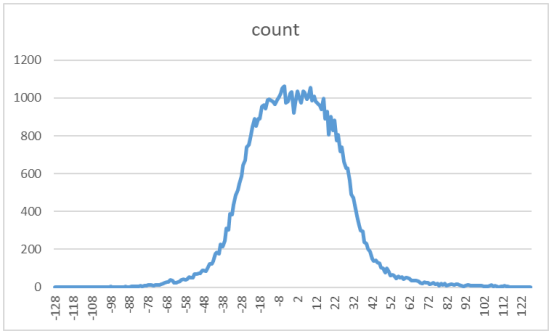
使用 8-neighbor，採用不同的 T 的結果如下:

T=10　　T=15　　T=20

當 T 增加時，因為被設為 G'=0 的比較多，所以在比較左右鄰居有無 zero-crossing 時的機率會下降，造成許多 edge 無法順利輸出，所以 T 不能設太大。相較於 1st order edge detection，2nd order 會輸出過多細節，造成整體的 edge 看起來不是那麼清楚。 另外，4-neighbor 與 8-neighbor 相比，雖然差異沒有很大，但仔細觀察可以發現，8-neighbor 產生的 edge 較多。最後選擇 8-neighbor T=15。

用 4-neighbor 與 8-neighbor 算 sample2 的 gradient map 的 histogram 如下所示:





比較 sample1 gradient map 的 histogram 跟 sample2 的，可以發現因為 noise 讓原來比較尖銳(高、窄)的 histogram 變得比較圓弧(低、寬)。下面分別列出不同方法、不同 T 的結果。

使用 4-neighbor，採用不同的 T 的結果如下:

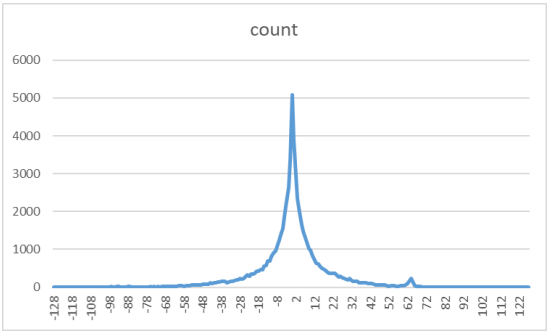
T=15 T=20 T=25

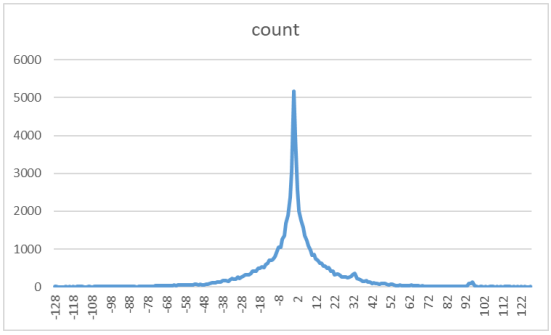
使用 8-neighbor，採用不同的 T 的結果如下:

T=15 T=20 T=25

跟 sample1 的結果相比，T=15 時結果還可以，但在有加 uniform noise 的 sample2 裡，結果非常糟糕，必須得將 T 調大，雜音才不會太恐怖，但這麼做，又必須犧牲掉細節。最後我選擇 8-neighbor T=20。

用 4-neighbor 與 8-neighbor 算 sample3 的 gradient map 的 histogram 如下所示:

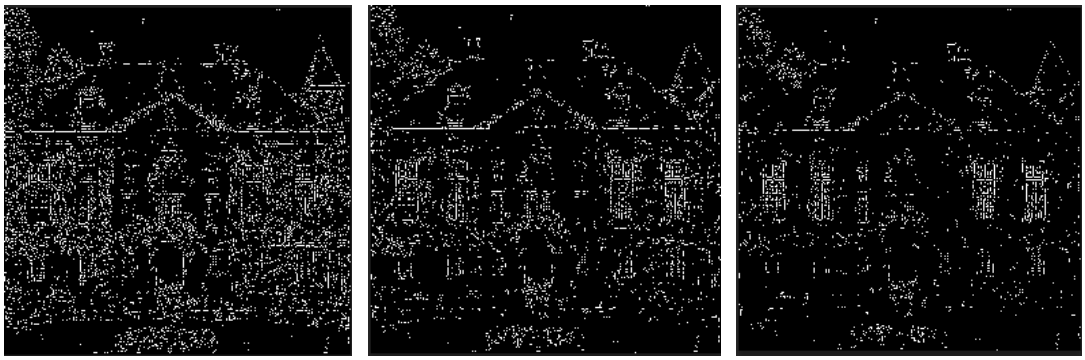




可以發現 sample 3 gradient map 的 histogram 的 shape 與 sample1 的很像(相較於 sample2)，只是 peak 的 value 從 8000 降到 5000。

使用 4-neighbor，採用不同的 T 的結果如下:

T=10 T=15 T=20

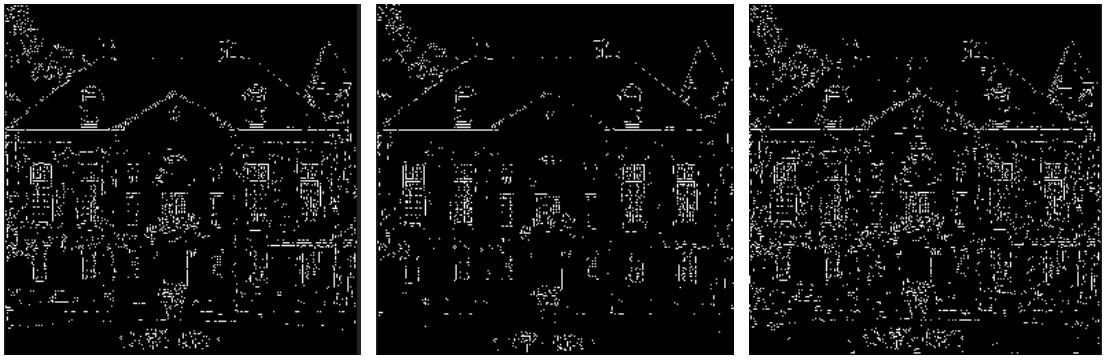


使用 8-neighbor，採用不同的 T 的結果如下:

T=10 T=15 T=20

4) Discussion of results:

Sample1E2.raw Sample2E2.raw Sample3E2.raw



比較 1st order 跟 2nd order，1st order 選擇不同的 Threshold 會讓輸出結果差異很大。若使用 Prewitt 和 Sobel，會遇到要決定 T 的難題，而且，試了好幾種 T 結果都不太好 (除了使用'考慮 3 個點, 0-90 度'之外)。而 2nd order 因為 gradient map 的 histogram 大致的 shape 都差不多，所以 T 比較好決定，也比較容易觀察到 T 的變化與輸出的變化。不過，2nd order 在處理 noise 方面比較差，1st order 雖然輸出圖還是有 noise，但是看得出原先的輪廓，且可以知道哪些比較是邊，哪些是雜訊，但是 2nd order 幾乎混在一起了，如果想要讓雜訊若一點，只能調高 Threshold，但這樣子 edge 跟 noise 都會一起被去掉。

c) 做 canny edge detection，輸出E_3

1) Your motivation and approach:

根據上課內容，具體步驟如下:

- 使用講義上提供的 5x5 Gaussian filter 來撫平 noise。
- 使用與 1st order edge detection 相同的方式(考慮 3 個點, 0-90 度) 來計算 gradient 和 theta 角度
- 根據 theta 角度來決定每一個 pixel 要比較的兩個附近鄰居。如果此點比他的兩個鄰居都大，才能留下來
- 自訂兩個 Threshold，T_H、T_L，當 pixel value > T_H，則此 pixel 為 edge，若 pixel value < T_L，則此 pixel 不為 edge，若 pixel value 介於T_H與T_L之間，則將此 pixel 設為 candidate。
- 針對上一步求到的所有 candidate，檢查其鄰居是否為 edge 或是有連到其他 edge 的 candidate，若是，則將此 candidate 設為 edge。

3) Output images

在 sample1 測試不同T_H、T_L 的結果圖:

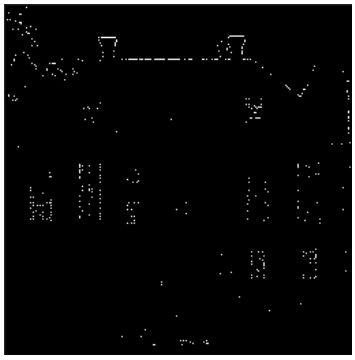
$T_L = 10$、$T_H = 30$　　$T_L = 10$、$T_H = 40$　　$T_L = 10$、$T_H = 50$

可以觀察到，當T_H取的越小時，越多的細節可以顯現出來，像是花圃的拱形在$T_H = 30$時顯現的最完整，但是房子牆上的斑駁也比較明顯；當T_H慢慢增加，細節會越來越少，但整體來看，效果還是不錯的!

在 sample2、sample3 測試不同T_H、T_L 的結果圖:

$T_L = 50$、$T_H = 70$　　$T_L = 40$、$T_H = 80$　　$T_L = 40$、$T_H = 120$





$T_L = 80$,$T_H = 120$(sample2) sample3: $T_L = 20$ $T_H = 50$ $T_L = 30$ $T_H = 60$



4) Discussion of results:

由上面的結果可以觀察到，一樣是在沒 noise 的 sample1 效果最好，sample2 與 sample3 雖然輸出還是有 noise，但比 1st order 及 2nd order 的效果好(noise 輸出情況沒有那麼嚴重)。另外，可以發現 canny 的T_H的設定對輸出結果有比較大的影響。當固定T_L改變T_H的時候，輸出會有變化，但是反過來，固定T_H改變T_L的時候，輸出幾乎沒變。原因應該是因為，在 canny 的最後一步會去檢查 candidate 最終有沒有連到 edge，所以當 edge 的判斷條件一樣時(固定T_H)，canny 最終還是能找到。

方法	pros	cons
1st order	抗 noise 的效果優於 2nd order	較不容易決定 Threshold
2nd order	比較容易決定 Threshold	抗 noise 的效果最差
canny	抗 noise 的效果優於前兩者	Implementation 較繁雜

PROBLEM 2: GEOMETRICAL MODIFICATION

a) 做 edge crispening 並輸出成 C

1) Your motivation and approach:

根據上課內容，edge crispening 可以有兩種作法:

- 做 high pass filter，根據講義，有兩種 filter 可以嘗試:

$$H_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- 用 unsharp masking (用一個 all pass filter 搭配 low pass filter)

$$\mathrm{G(j,k)} = \frac{c}{2c-1}F(j,k) - \frac{1-c}{2c-1}F_L(j,k), \qquad where \ \ \frac{3}{5} \le c \le \frac{5}{6}$$

這裡的 low pass filter 可以拿 HW1 實做過的來用

$$\mathrm{H} = \frac{1}{(b+2)^2}\begin{bmatrix} 1 & b & 1 \\ b & b^2 & b \\ 1 & b & 1 \end{bmatrix}$$

3) Output images

Original image　　High pass filter(H1)　　High pass filter(H2)



Unsharp mask b=2, c=0.65　　mask=7, c=0.65　　mask=7, c=0.8



4) Discussion of results:

相較於直接使用固定權重的 high pass filter，unsharp masking 可以調整 low pass filter 的權重與 c 的大小(根據規定，c 應取[0.6, 0.83])。雖然當圖的大小為 256x256 時，肉眼看不出差異，但是當放大輸出結果圖後，可以微小的觀察到，c 取小一點，出來的 edge 會比較清晰，b 取小一點，也稍微好一點。

b) 設計一個 warping method 將 C 轉成 D

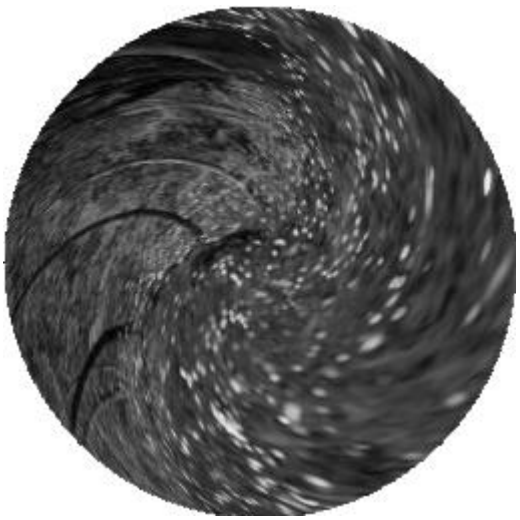
1) Your motivation and approach:

步驟如下:

- 找出在一個 256X256 的圖形上，哪些 x, y 座標點屬於圓內部。因為圓的 boundary 條件可以想像成: 令某一點的 x 座標和 y 座標到圓點的距離為Δx, Δy，如果$\Delta x^2 + \Delta y^2 \leq radius^2$，這個點就在圓內部。
- 對於每一個在圓內部的點 (x', y')，先轉成 polar coordinates，再算出d, θ，$d = \sqrt{x'^2 + y'^2}$, $\theta = \tan^{-1}\frac{y'}{x'}$ ，再求其對應到原圖的座標 (x, y)的 bilinear interpolation。當$x \geq y$ 時，$x = d, y = \frac{d}{\cos\theta} \times \sin\theta$。 當$x < y$ 時，則$y = d$, $x = \frac{d}{\sin(90° - \theta)} \times \cos\theta$，最後再將 polar coordinates 轉回 image coordinates。
- 處理完圓後，再來做 swirl。根據網路上查到的 swirl.java 的做法[1]，一樣先轉成 polar coordinates，再求 $d = \sqrt{x'^2 + y'^2}$，再算新的$angle = \pi/256 \times d$，新對應到的 x 座標為 $\cos(angle) \times x - \sin(angle) \times y$；新對應到的 y 座標為 $\sin(angle) \times x + \cos(angle) \times y$。

2) Original images (左)

3) Output images (右)



4) Discussion of results:

雖然 swirl 轉的方向跟題目給的不太相同，但只要調整步驟中提到的 x, y 公式的正負號，就能改變想要的方向。

Reference:

[1] https://introcs.cs.princeton.edu/java/31datatype/Swirl.java.html