

Problem 1 - Edge Detection

ORIGINAL



(a) 1st Order Edge Detection

<step 1> Orthogonal gradient generation

<step 2> Thresholding to get the edge

<step 1> 以 Prewitt Mask 9-points 近似 First order differentiation，得到所有 pixel 的 gradient。(下式 K=2)

$$G_R(j,k) = \frac{1}{K+2} [(A_2 + KA_3 + A_4) - (A_0 + KA_7 + A_6)]$$

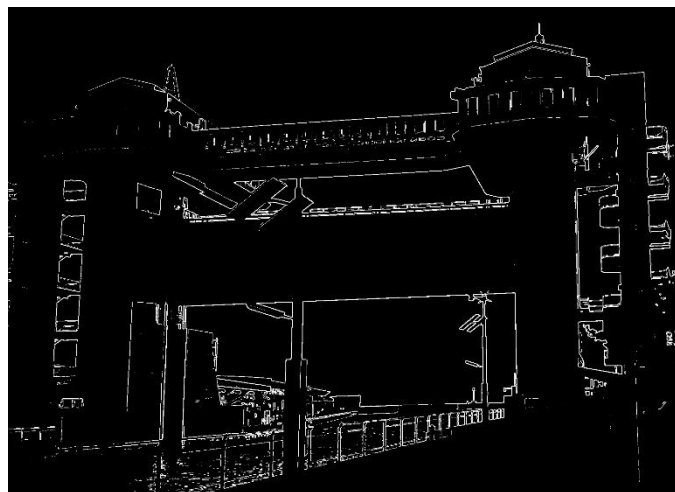
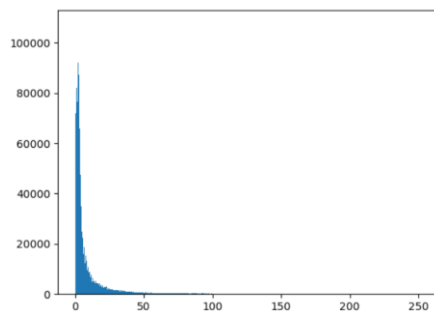
$$G_C(j,k) = \frac{1}{K+2} [(A_0 + KA_1 + A_2) - (A_6 + KA_5 + A_4)]$$

$$\Rightarrow G(j,k) = \sqrt{G_R^2(j,k) + G_C^2(j,k)} \quad \theta(j,k) = \tan^{-1} \left(\frac{G_C(j,k)}{G_R(j,k)} \right)$$

A_0	A_1	A_2
A_7	$F(j,k)$	A_3
A_6	A_5	A_4

<step 2> 設定 threshold 的方法為觀察 histogram，並且 trial and error。最後決定以 80 作為 threshold 得到最佳 edge detection 結果。

RESULT:



(b) 2nd Order Edge Detection

<step 1> Laplacian generation

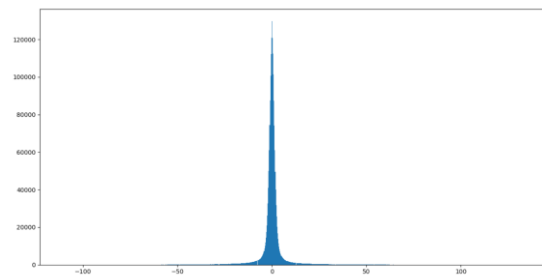
<step 2> Generate the histogram to thresholding

<step 3> Decide the zero-crossing point by edge map

<step 1> 以 discrete approximation 近似 laplacian generation，得到所有 pixel 的 laplacian。

$$-\frac{\partial^2}{\partial x^2} \cong [-1 \quad 2 \quad -1] \Rightarrow 2f(x) - (f(x+h) + f(x-h))$$

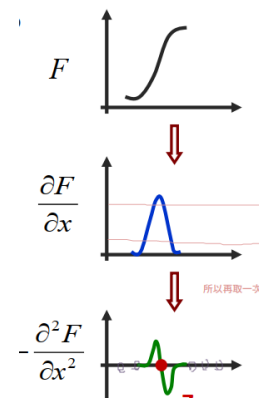
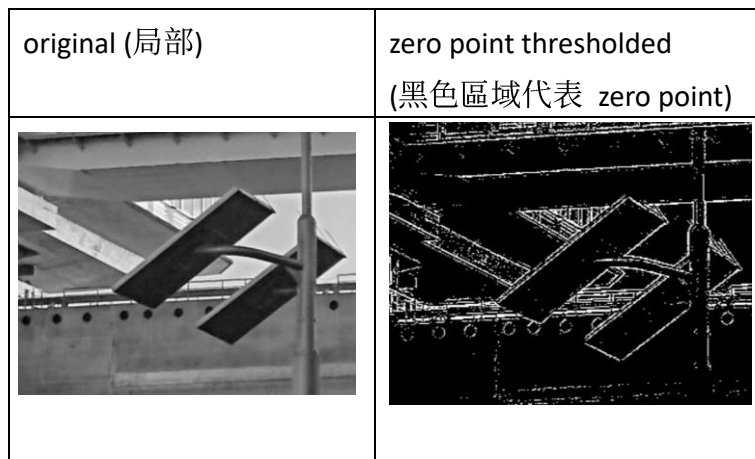
<step 2> 為了要找到合適的 threshold，根據 histogram 選取範圍後，將 2nd-differentiation 結果用不同 threshold output 出來，將 threshold 篩選過接近為零者標為黑點，其它標為白點，如下表。



threshold = 1	threshold = 3

選擇 $\text{threshold} = 5$ ，原因是在影像中 edge 的附近有合理的白點數量，那些黑色區域的 zero point 才能在下一步 $\text{zero-crossing point}$ 被合理偵測出來。

<step 3> Edge map 是為了要篩選出 $\text{zero-crossing point}$ 。在右下圖，二次微分的作圖結果中，我們需要的是紅色的點作為 edge ，他才能代表性的表示 F 中強度變化的中點，而非那些在原圖無變化的紫色點。所以在<step2>中所篩選二次微分為零的 zero 點，其周遭的二次微分應該也不為零，稱為 $\text{zero-crossing point}$ ，故設計一些方法來偵測該點。(以下分析以照片中太陽能板附近的畫面為準，比較好觀察。)


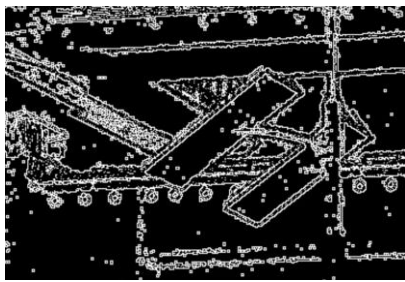



方法一：設計 edge map ，對於該 zero point ，若相對於該點臨邊有對邊恰好為 $-1, +1$ (i.e. 值相乘 $= -1$ 。右圖為對角線的例子)，則判定該點為 $\text{zero-crossing point}$ ，因為由二次微分結果可以看出 $\text{zero-crossing point}$ 左右必包含一正值和一負值。實作後發現用這個方法會有太多點被捨棄，導致如下表 edge 破碎化的現象。

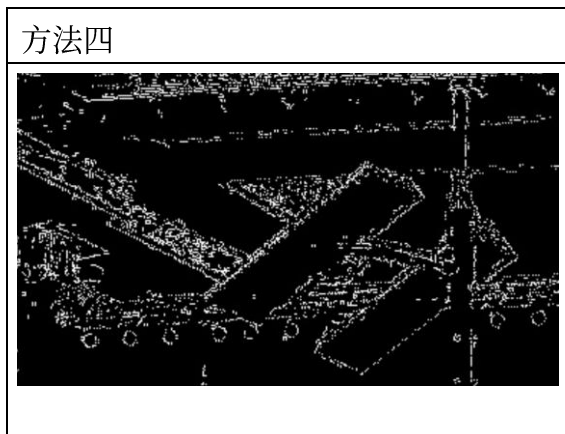
-1		
	0	
		1

方法二： edge map 中，對於該 zero point ，只要有臨邊(8 connected neighbor)為 -1 或 $+1$ 者，判斷其為 $\text{zero-crossing point}$ 。實作後這個方法會導致 thresholded 結果中一些雜點被放大。如表中的圖，可以看到原本雜點的位子被環繞圈出。

方法三： edge map 中，定義有和 zero point 相連之非零點為一種 bond ，(i.e. 4 connected 位子 $\text{bond} = 2$ ；8 connected 位子 $\text{bond} = 1$ 。) 保留 $\text{bond} > 3$ 者判定其為 $\text{zero crossing point}$ ，結果如下，有正確框出原始照片中深度改變的邊界。雖然方法三結果看似不錯，但其實仔細看會發現一些 locally 變化被放大。像是橋上黑色圓圈應該只會有一個代表性圓形的 edge ，卻被圈成很多圈。

decide zero-crossing point (with different edge map)		
方法一	方法二	方法三
		

方法四：考量到正確的 zero-crossing point 附近應該有 $+1, -1$ 的點，所以最後決定改良方法一，改為鄰近點(8 neighbor)若包含 -1 及 $+1$ 即判斷該點為 zero-crossing point，結果如下。



RESULT:



(c) Canny Edge Detection

<step 1> Noise Reduction

<step 2> Compute gradient magnitude and orientation

<step 3> Non-maximal suppression

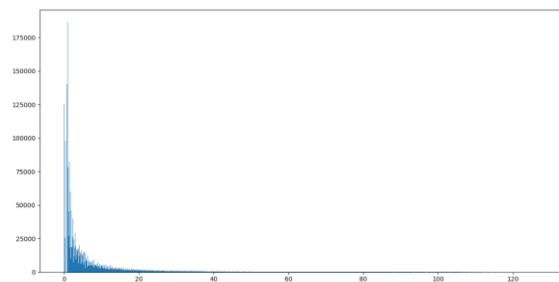
<step 4> Hysteretic thresholding

<step 5> Connected component labeling method

<step 1> Noise Reduction 的作法是使用 5x5 Gaussian filter 做 Convolution :

$$F_{NR} = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * F$$

<step 2> Gradient 的計算同樣使用(a)小題的作法，以 Prewitt Mask 近似而得。



<step 1> Gaussian filter 後結果



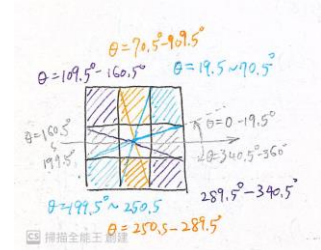
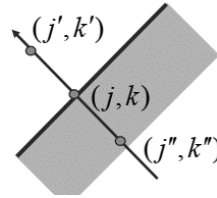
<step 2> 取 Gradient

<step 3> Non-maximal suppression

為了找出邊界最大點並且只留下最大點，進行 maximal detection，因此需要有該 pixel 的 Gradient Orientation 資訊，即上題所計算出的 $\theta(j,k)$ 公式在第一小題中，再計算九宮格對邊幾何關係，如右下圖，找到角度對應的 neighbor pixels position

$G(j',k')$ & $G(j'',k'')$ 。再由下式得到 $G_N(j,k)$ 。

$$G_N(j,k) = \begin{cases} G(j,k) & \text{if } G(j,k) > G(j',k') \\ & \text{and } G(j,k) > G(j'',k'') \\ 0 & \text{otherwise} \end{cases}$$



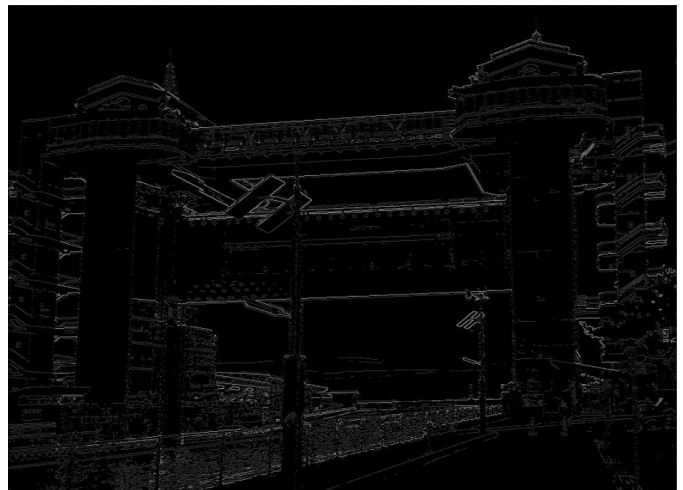
<step 4> Hysteretic thresholding

由 step2 所得到的 histogram 以及一些 trial and error，決定 $T_H = 50, T_L = 10$ ，使用 Hysteretic thresholding 方法找出合理數量的 edge pixel 及 candidate pixel。

$$\begin{aligned} G_N(x,y) &\geq T_H && \text{Edge Pixel} \\ T_H &> G_N(x,y) &\geq T_L && \text{Candidate Pixel} \\ G_N(x,y) &< T_L && \text{Non-edge Pixel} \end{aligned}$$



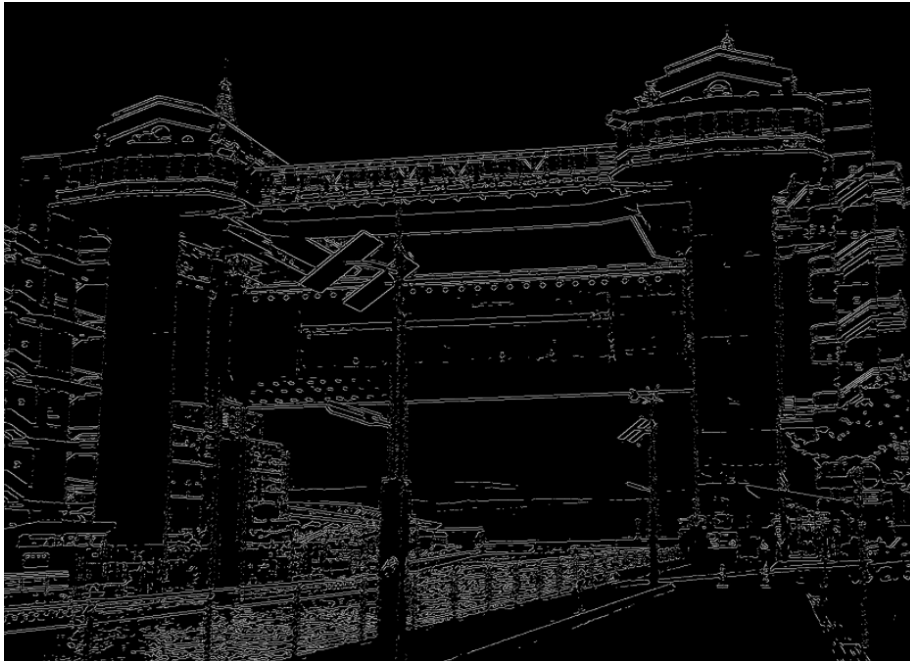
<step 3> non-maximal suppression 後的結果



<step 4> threshold 後的結果

<step 5>篩選 candidate pixel 中，有和其它 candidate pixel 相連的(8 neighbor connected)設為 edge pixel，其它則設為 non-edge pixel。

RESULT



(d) Edge Detection after Edge Crispening

<step 1> Apply high pass filter for edge crispening

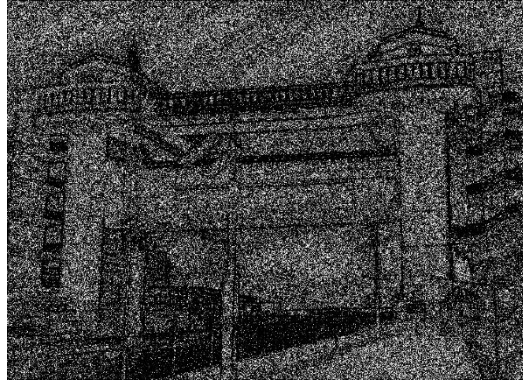
<step 2> Edge detection

<step 1> 這題使用的 high pass filter 如下，作用是強化和鄰近 pixel 差異性較大的 pixel，以達到 edge crispening 的效果。

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

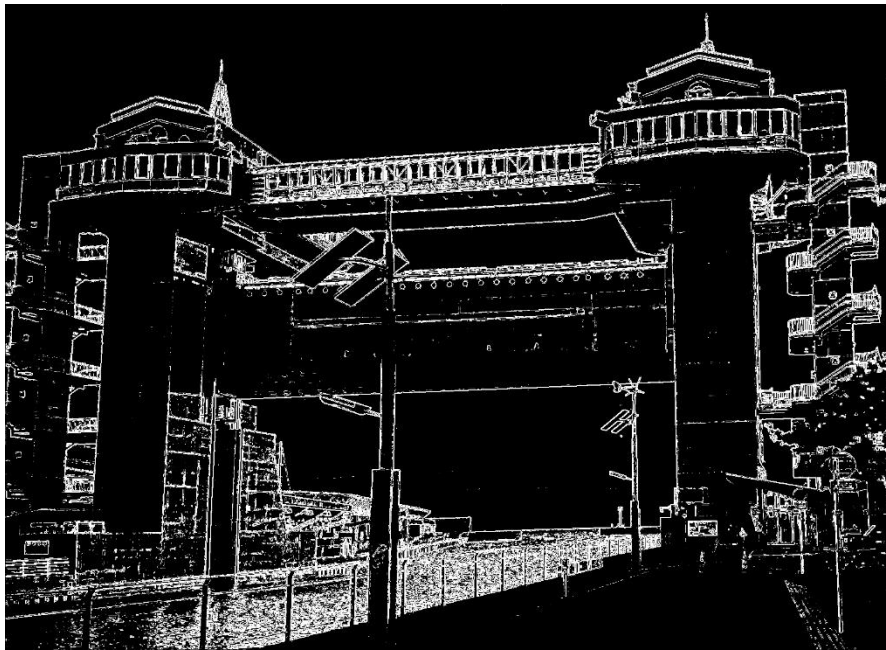
RESULT

<step 2> 因為考量到 high pass filter 後的結果同時也會強化 noise，所以不適合再使用 2nd order edge detection，因為在二次微分時也會將 noise 更加放大。所以使用 1st order edge detection。



2nd order method 結果不佳

RESULT



(e) 比較

以上我們實作了四種 edge detection 方式，分別為 1st order、2nd order、Canny method、edge detection after crispening。

1st order 的操作最為單純，僅將原圖的 gradient 作為篩選依據，所以輸出的 edge 接近肉眼觀察的結果。一些細部的 edge 需要調低 threshold 才看得出來，但同時也會濾進 noise，而經過 crispening 處理後的照片再以 1st order edge detection 效果較佳。缺點是，對於變化比較大的邊緣，會有較粗的 mark of edge，因為它不像 2nd order 會找代表性的 zero crossing point、或像 canny 有做 non maximal suppression，就只是單純的呈現 gradient 大小。

2nd order 的操作較麻煩，需要很多 trial and error。他的優點是幾乎所有深淺變化都可以被監測出（連橋墩旁邊樓梯間的天空，拍攝產生太陽繞射的狀況都被框選出來，我把照片放大才發現 XD）。缺點是 noise 會被放大，還有太多篩選機制某種程度降低了容錯性，尤其是用 edge map 找 zero-crossing point 的過程，因為具有邊緣代表性的 pixel 數量本來就不多，再以此嚴格條件篩選後，最後輸出有很多邊緣都變得十分破碎化。所以我認為 2nd order detection 更適合應用在解析度較高的照片上。

我認為 Canny method 融合了 1st order、2nd order 及 crispening 的優點，輸出的結果既沒有 noise，還有十分細緻的邊緣線。我認為 Canny method 的流程很符合人類視覺對 edge 的辨識：深淺突然發生變化的邊界、即使該邊界遇到某些強度相近的背景，人類會預設該邊界由連續的線條框著，所以找得到他。Canny method 把這些大大小小的細節都包含住了。缺點是流程很多，電腦跑起來比較慢。

(f) Case study

ORIGINAL:

<step 1> Image enhancement

<step 2> Edge Detection

觀察這張景色昏暗的照片，其 edge 不易辨識出，輸出該原始圖檔的 histogram，可以看到多數 pixel 集中在高強度的位子，因此對他作 image enhancement。

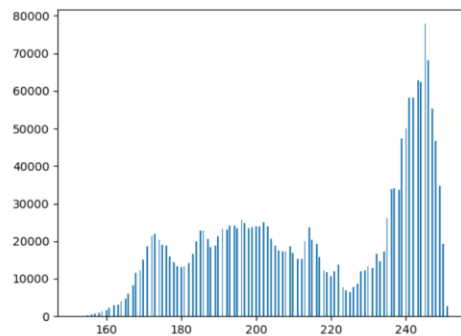
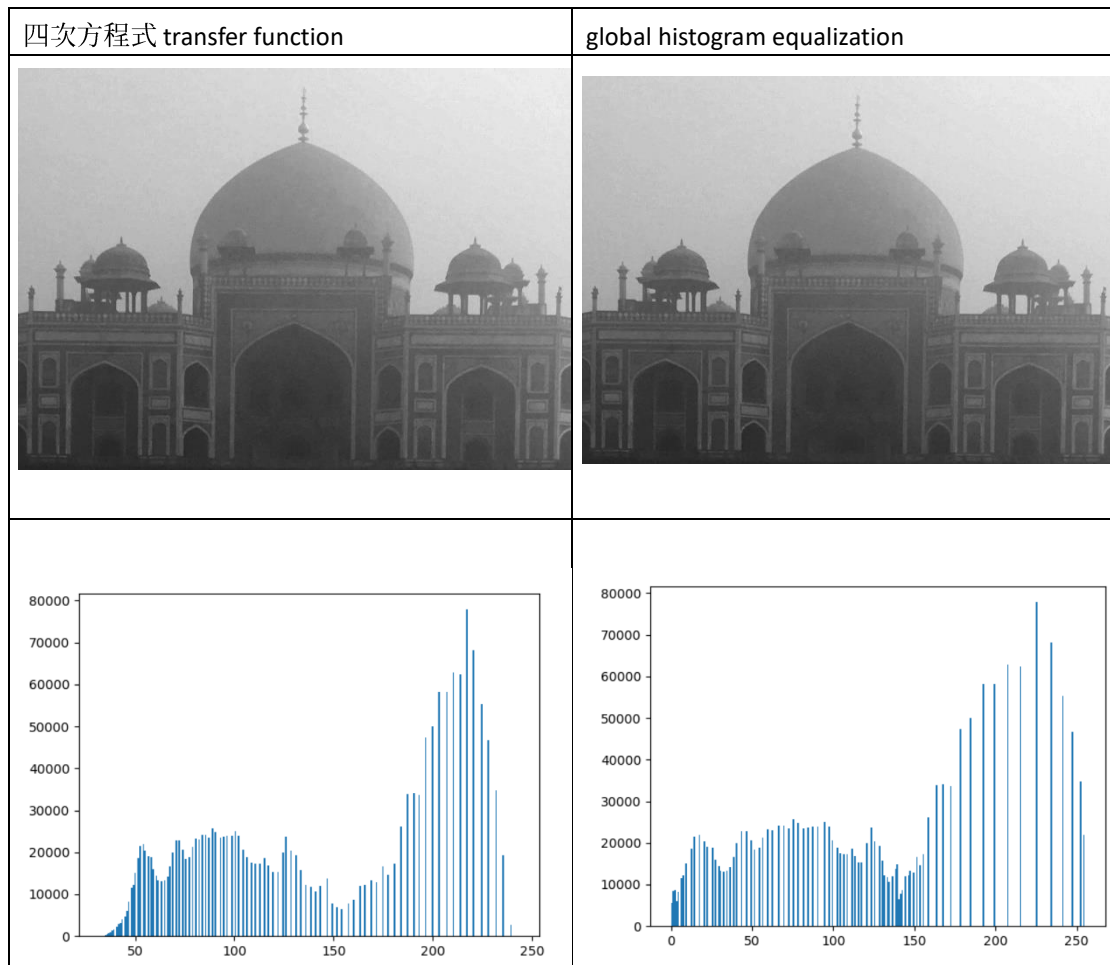
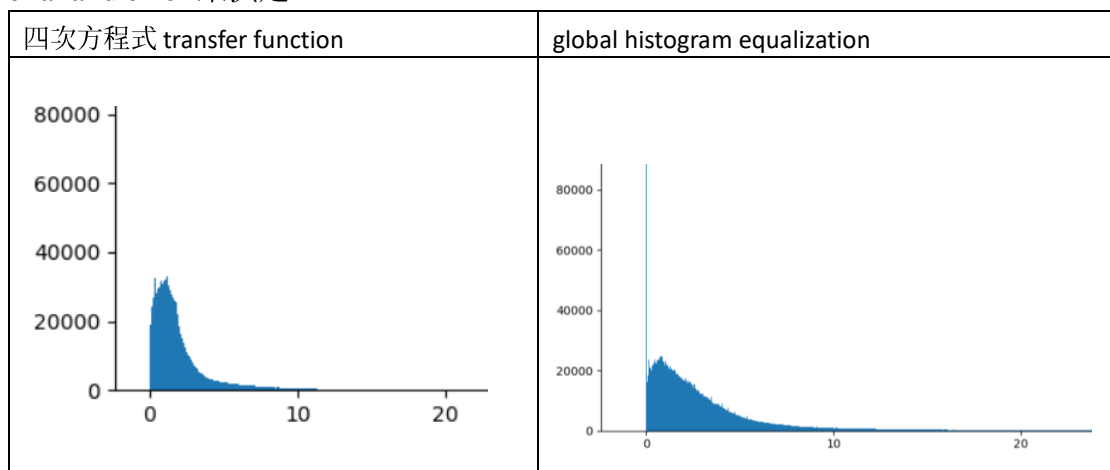


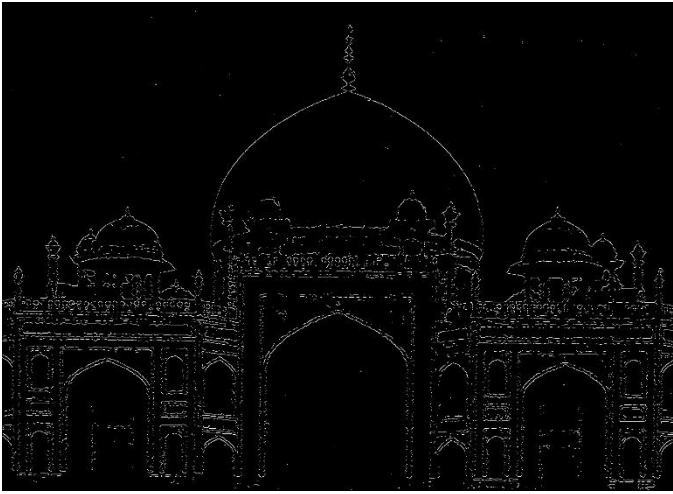
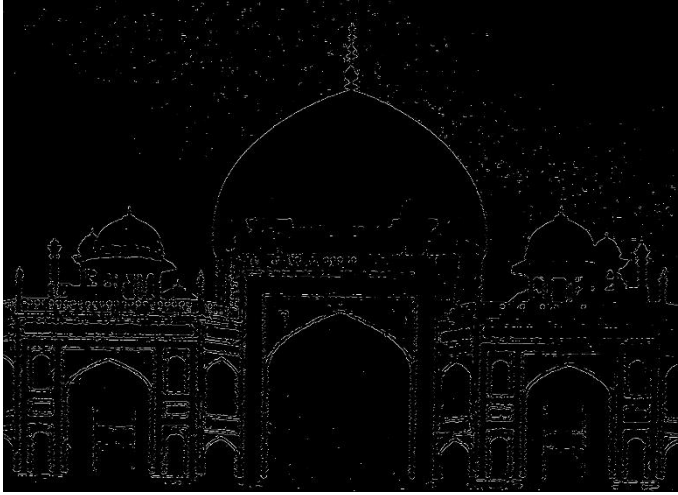
Image enhancement 的方法我做了兩種，一種是以四次方程式的 transfer function 調整其 histogram，另一種是作 global histogram equalization，結果如下。兩種方法對原圖都有顯著的強化，接下來做下一部的 edge detection。



由上一小題的探討，我認為 Canny 的效果最好，所以採用 Canny 來做這題的 edge detection 分析。thresholding 的 Tl 和 Th，同樣參考 gradient 後 histogram 和一些 trial and error 來決定。



RESULT:

四次方程式 transfer function	global histogram equalization
	

由結果看出四次方程式 enhancement 再做 edge detection 的效果明顯比 histogram equalization 後的還要好。觀察強化後的照片的 histogram，我認為可能原因是 histogram equalization 為了讓圖片 pixel 強度 cdf 線性增加，在亮度強的地方，histogram 呈現更離散的分布，這樣會導致 gradient 在亮部過強，所以在 Canny edge detection 的結果中，發現較明亮的天空些無法消去的雜點，雖然調低 threshold 可以消除，但同時清真寺那裡的 edge 也變得不清晰。

Problem 2 - Geometrical Modification

(a) Rotation, Scaling and Translation

<step 1> Transform every pixel (row, col) to Cartesian coordinate (x, y)

<step 2> Linear transformation of translation, scaling and rotation on every pixels coordinate by backward treatment

<step3> Map every pixel of output (row', col') to pixel on original point

ORIGINAL:



這題的實作方式，主要是將照片上的 pixel 轉為卡式座標，並進行 matrix 的 linear transformation 運算。為了加快速度，我將針對照片想做的 Geometrical Modification 寫在同一個 function 裡，也就是所有 translation, scaling, rotation 都在一次運算中完成。

我採用 backward treatment。針對要新建立的輸出圖片，先將代表 pixel 的 (row', col') 點轉為卡式座標 (x', y')，進行線性代數 translation, scaling, rotation 運算完畢後，得到每個輸出點 map 到原本 image 中的座標 (x, y)，再用代表照片 pixel 的 (row, col) 表示。

此時，(row, col) 數值大小可能超出原本照片的邊界，也就無法取得的相片資訊，比方說 scaling 1/2，對於一樣尺寸的輸出照片，可以想見輸出結果應該是一張比原本小的照片，其它部分只能以全黑處理，所以將該點設為 0。此外，(row, col) 數值可能是一個小數點，這種狀況就必須做內插法，以該點和鄰近 pixel 距離關係，求得內插後該點合理的強度對應。

描述	對中心逆時針轉 90 度	往左平移 500，往上平移 100	對原點縮小 0.5 倍
指令	<code>output = Deformation(img = img, rotation = True, cx = 0.5, cy = 0.5, angle = 90/180*math.pi)</code>	<code>output = Deformation(img = img, translation = True, tx = -500, ty = 100)</code>	<code>output = Deformation(img = img, scaling = True, sx = 0.5, sy = 0.5)</code>
輸出			
描述	針對本題題目要求理想輸出結果，往左平移 500，往下平移 400，對照片中心位子逆時針旋轉 90 度，將照片 xy 尺度皆對原點放大 1.5 倍		
指令	<code>output = Deformation(img = img, translation = True, tx = -500, ty = -400, rotation = True, cx = 0.5, cy = 0.5, angle = 90/180*math.pi, scaling = True, sx = 1.5, sy = 1.5)</code>		
輸出			

(b) Case: Swirling image**ORIGINAL:****TARGET:**

觀察本題目標 **target image**，會發現越接近中央的部分扭曲越大，同時也有吸進去的感覺。所以我把處理步驟分為兩步：吸入和旋轉，再將兩者合併。

<step 1> Suction to center

<step 2> Swirling around center

<step 3> Suction + Swirling

<step 1> 要達到吸入的效果，可以做往中心不等價縮小(scaling)，因此我先計算每個 pixel 和中心的距離，越接近中心，縮小程度越小。以下為三種不同公式的縮小效果。

x: 該 pixel 和中心的距離除以照片角落到中心的距離。

y: scaling rate °

x': scaling 後的位子。作為 backward treatment 於函式中所套用的公式。

	linear function	quadratic function(2 degree)
公 式	$y = x$ $y = \sqrt{x'}$	$y = x^2$ $y = x'^{\frac{2}{3}}$
作 圖		
結 果		

發現上述皆不太像這題 **target image** 的效果，因為似乎只有統神被吸入，外圍像是白色框框就比較沒有縮進去的效果，所以決定用 **y** 方向的 **cubic function** 來達到接近中心縮小程度大，外圍縮小程度小的效果，以及另一種有條件的縮小。經過 **trial and error**，發現在距離中心約 $x = 1/3$ 半徑範圍內，以 **2 degree function** 縮小，效果蠻不錯的。

	cubic function on y (3 degree)	conditional scaling (2 degree)
公式	$y = (0.25(x - 0.5))^{1/3} + 0.5$ $x' = xy$	$y = 9x^2 \quad (x < 1/3)$ $y = (3x')^{2/3} \quad (x < 1/3)$ $y = x = x' \quad (x \geq 1/3)$
作圖		
結果	(這題需要解析解，工作站跑約三小時還沒跑出來 所以放棄)	

<step 2> 觀察 target image，可以發現有往中心旋入的效果，因此也做往中心不等價 rotation。同 step 1，經過一些 trial and error 找到最好的公式、最好的旋轉範圍和旋轉程度。得到的最佳解是 exponential，再根據 target image，細調各項參數後得到最好的公式如下：

x ：該 pixel 和中心的距離除以照片角落到中心的距離。

y ：rotation angle。

x' ：rotation 後的位子 = x ，因為僅旋轉。

	exponential function
公式	$y = 2.9e^{-5.7x} + 0.2$ $\text{range: angle } y = \frac{\pi}{16} \sim \pi$
作圖	
結果	<p>(純旋轉)</p>

<Step 3> 合併以上兩步驟，最理想的結果：

RESULT:

