Computer Vision HW1 Report

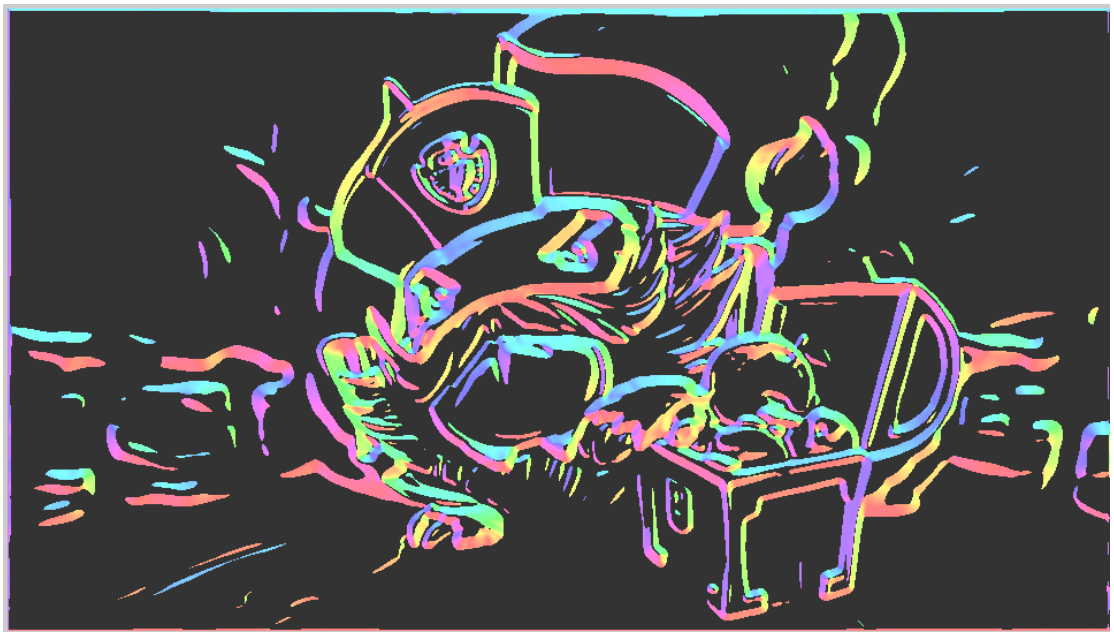
102062209 邱政凱

這次的作業要求我們實作 Edge Detection 和 Corner Detection。
在 Edge Detection 的部分，第一部分叫我們用不同的 Kernel Size 和 Sigma (高斯模糊的標準差)來對原圖進行模糊處裡。

可以發現，Kernel Size 較大和 Sigma 較大都會導致圖片更加模糊。題目指定 Sigma 要使用 2 跟 5，而我的 Kernel size 設為 5*5 和 15*15，在 15*15 的 Kernel size 中可以很明顯地觀察到 Sigma= 5 比 Sigma=2 的結果還要來的模糊非常多，然而 5*5 的 Kernel size 中兩個 Sigma 值模糊後的結果卻沒什麼差別，這應該是導因於 Kernel Size 太小會讓高 Sigma 值的 Gaussian Distribution 沒有足夠的空間去精確地呈現它廣泛分布的情形。

(上：kenel = 5x5 , sigma=2 ，下 kernel = 15x15, sigma = 5)

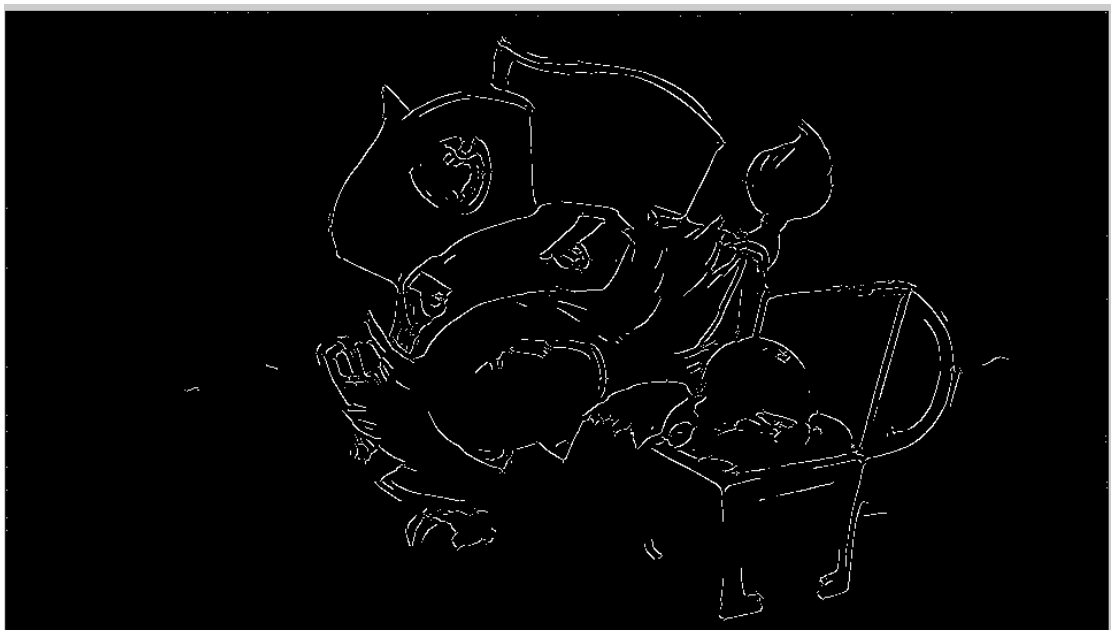
1- B 要我們用 Sobel Operator 去計算圖片的 x 梯度和 y 梯度，並計算每個像素的梯度強度和梯度方向。我使用了兩個模糊後的圖片(Kernel size 5x5, sigma = 2 和 Kernel size 15*15, sigma = 2)來進行這題的實驗。可以發現 size5x5 模糊後的圖片的梯度強度是比較強的，而 size15x15 模糊後的圖片的梯度強度是比較弱的，這應該是導因於上提得出來的結論：15x15 的圖片是比較模糊的，而比較模糊的圖片像素顏色的改變應當是比較緩慢的，因此圖片的梯度會比較低。梯度方向的表示我則是使用了助教提供的 rgb color map，不同的角度直接對應到不同的顏色(從 0’ -> 紅色 到 360’ ->紫色)



(上：梯度強度，下:梯度方向)

1_C 則是要我們用 Non-Maximum suppression 來把偵測到的邊緣做 Thinning，然

後用不同的 Threshold value 做 Thresholding 來優化結果。
Non_maximum_suppression 我的做法是去比較每個像素的梯度方向上鄰近的兩個點(用內插來計算落在兩像素中間的點)，若兩個點都比該像素的梯度強度小，則把該點保留，否則去掉該像素。Threshold value 我用了 0.1 和 0.2 來做，0.1 的結果已經可以把畫面上的雜點都清除掉，不過 0.2 的結果可以更進一步讓畫面上的線幾乎只剩下角色身上的線。可以驗證這張圖上高強度的梯度變化是發生在中間的角色身上，而背景的煙霧因為整體是比較模糊的遠景而造成梯度的強度比較低。



(上，Nom_maximum_suppression with threshold = 0.1,下，

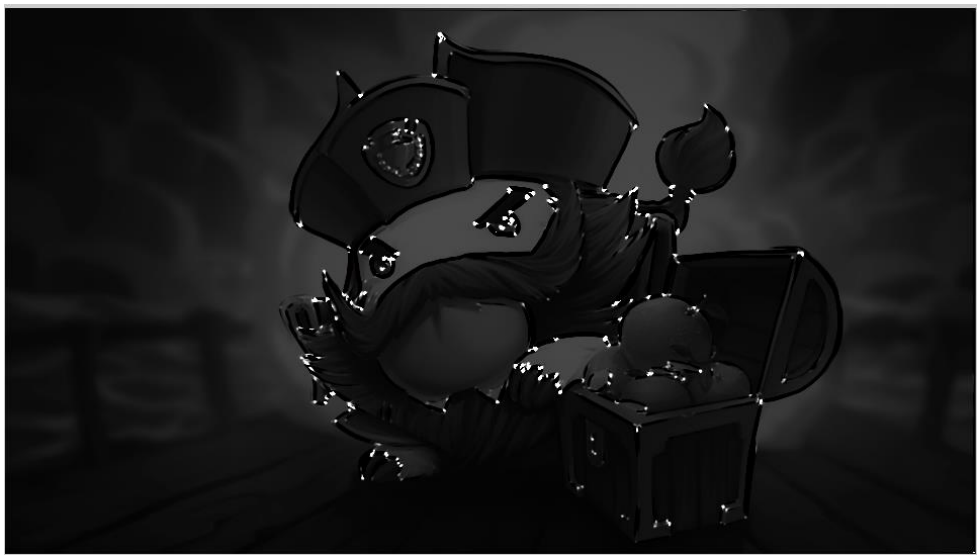
Non_maximum_suppression=0.2)

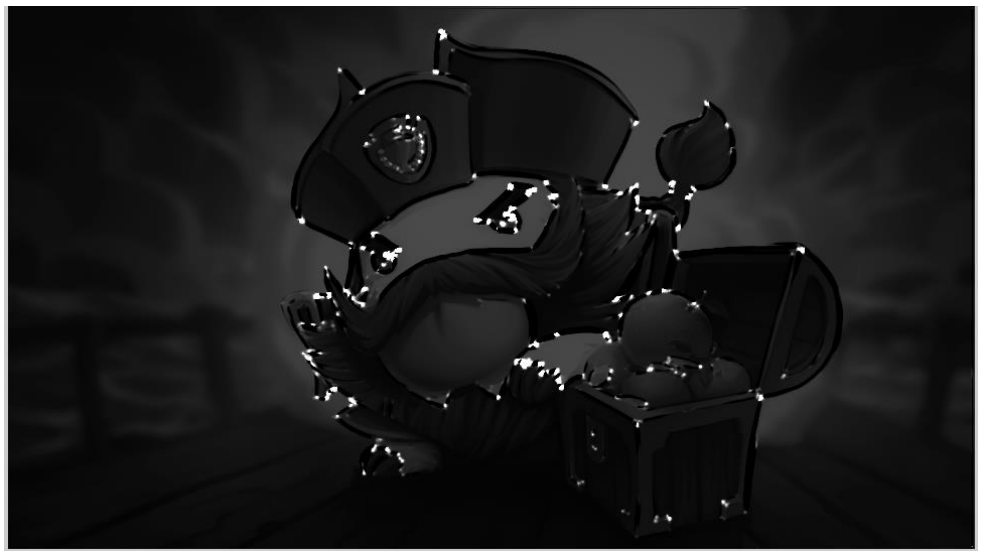
第二題則是要我們用 Harris Detector 去做 Corner Detection。

2_A 的部分，要我們用不同的 window size 去計算每個像素的 Structure Tensor Matrix(H)，並且把算出來的 smaller Eigenvalue 秀出來。我的計算方式是以每個像素為中心，把 window 中的每個像素的梯度用不同的權重加總(使用高斯分布的 kernel，3x3 及 5x5，並且都對應 1 和 3 兩個 Sigma 值)，放入 Ix(x 梯度)、Iy(y 梯度)、Iz(z 梯度)中，並且解 [Ixx Ixy; Ixy Iyy]的特徵向量和特徵值。可以發現，3x3 的 window 偵測出來的 corner 點區塊會比較小塊一點，而 5x5 的 window size 偵測出來的 corner 點區塊(亮點)則比較大塊一點。Sigma 值也會有一點些微的影響，越大的 Sigma 值的 corner 點的亮點也會比較大塊一些。這其實還滿直覺的，偵測比較的範圍(window size)越大，則越容易偵測到包含邊緣和角落的區塊，因此會有高響應的地區會變多。

(上，smaller Eigenvalue with window 3x3，下，with window 5x5)

2_B 的部分要我們印出 Harris Detector 的 Response Function 的結果。可以發現 Response Function 和 smaller eigenvalue 其實結果有一些差別。Response 的結果比較清楚分明一點，因邊緣用 Response Function 測出來的結果會小於 0，因此疊在圖片上面會呈現全黑，因此用 Response Function 可以明確地看到 Corner 跟 edge 的差別。而且其實也可以看到在同樣的取樣視窗大小和 sigma 下 smaller eigenvalue 的高響應地區(白點的區域)較 Response function 還要來的多。





(上，response function with window3x3，下，with window 5x5)

2_C 的部分叫我們實作 Non maximum suppression，在這邊我的做法就是把每個像素跟周遭的八個像素比較，如果該像素為最大值則保留，否則丟棄。可以看到 Smaller Eigenvalue 保留下來的 Local maximum 比 Response Function 做出來的

還要多，這點也跟上面那題的結果呼應(smaller Eigenvalue 的高響應地區比較多)。而同樣的偵測方式，不同的 window size 則很神奇地呈現不一樣的結果：雖然在未 NMS 前 window 大的白色區塊比較大，但是 window size 較大的偵測到的 local maximum 卻比較少(雖然只少了一點點點)。

(上，smaller eigenvalue NMS ；下，Response Function NMS)

2_D/E 要我們去比較旋轉跟縮放後的原圖，再次經過 Harris Detector 是不是可以求出一樣的結果。為了方便比較，我又做了一次 Non maximum suppression。結果可以看出來，rotate 後偵測出來的特徵點位置其實變化不大，然而 scale 過後偵測出來的特徵點確有比較多的位置偏移和數量差別。這點跟理論上的”Harris 是 rotate invariant，not scale invariant”相吻合。(因為 ROTATE 之後不方便疊圖，所以沒有疊圖)

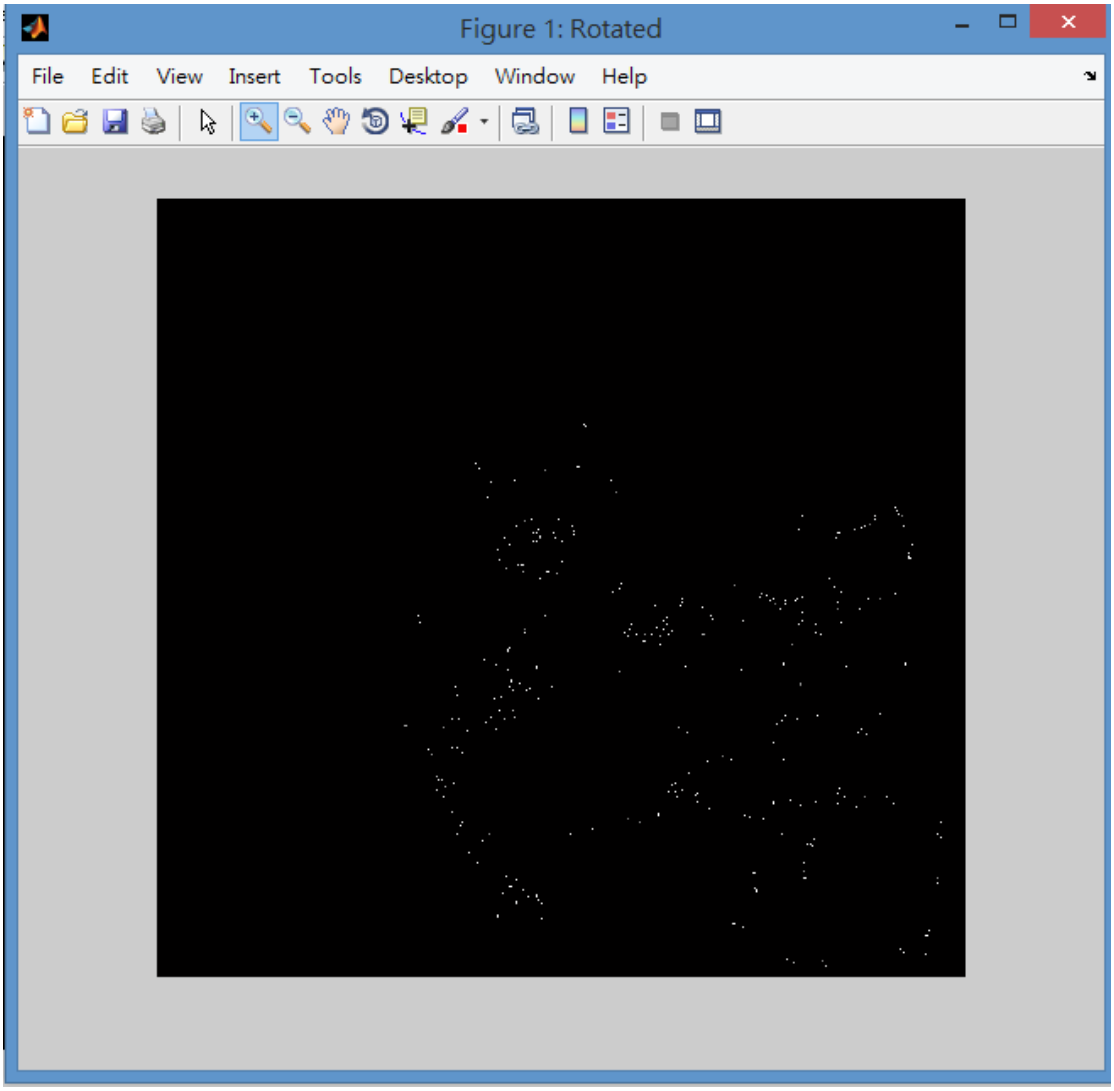
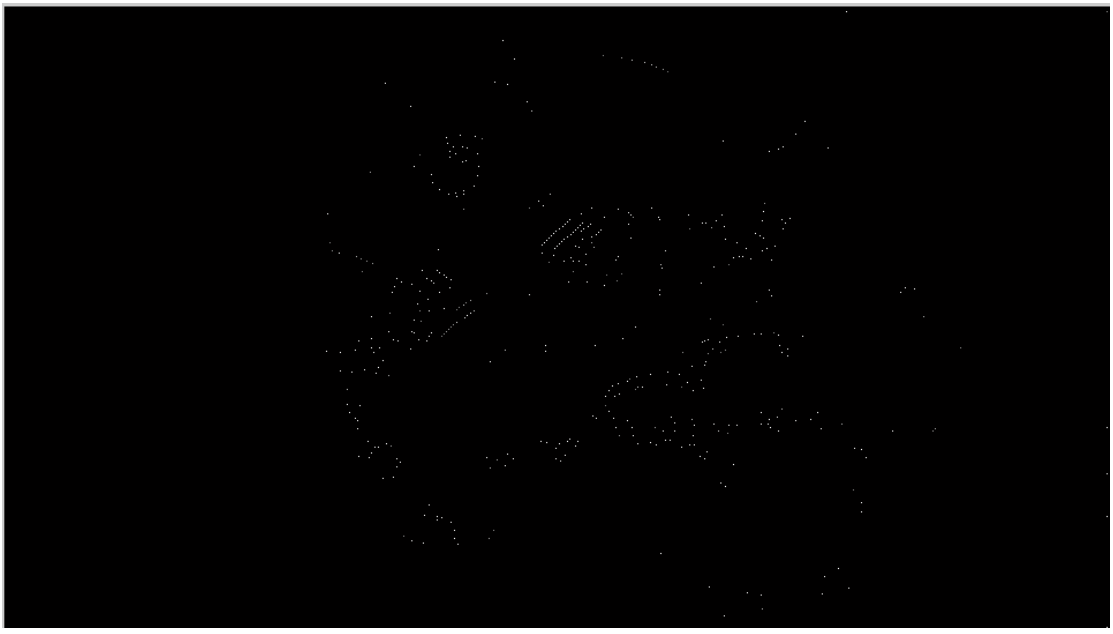
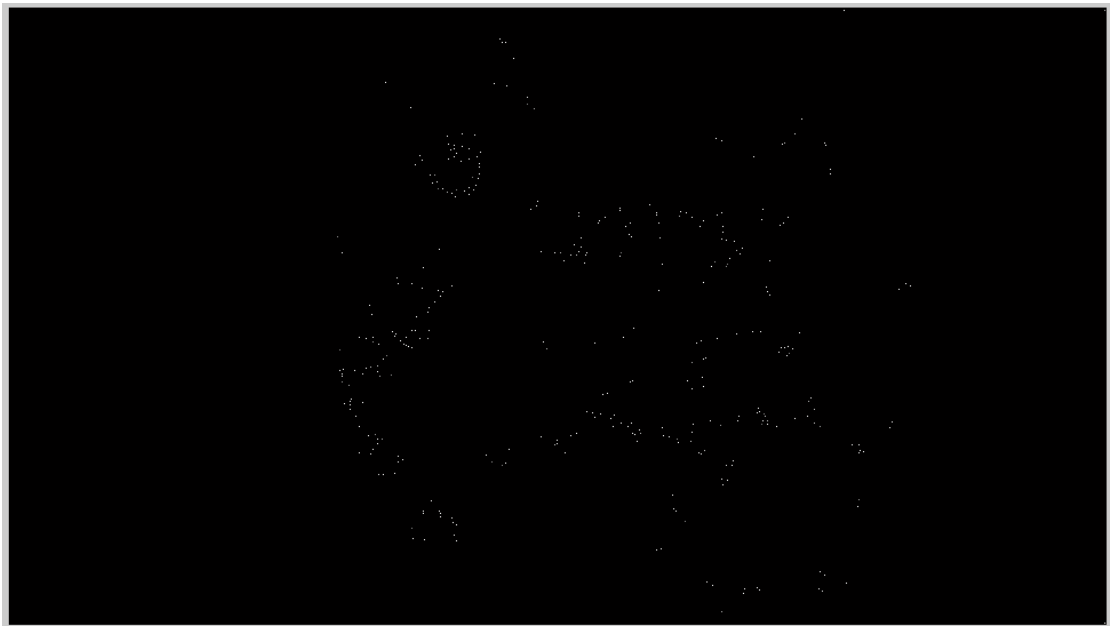


Figure 1: Rotated
File Edit View Insert Tools Desktop Window Help





(上，rotated，中，scaled，下，original；可以發現縮放過後跟 original 的特徵點差別頗大，不過 original 跟 rotated 的特徵點則比較沒有那麼大的差別(集中比較角色的帽子及頭部附近))。圖片若看不清楚還是請助教跑過一次我的 code XD