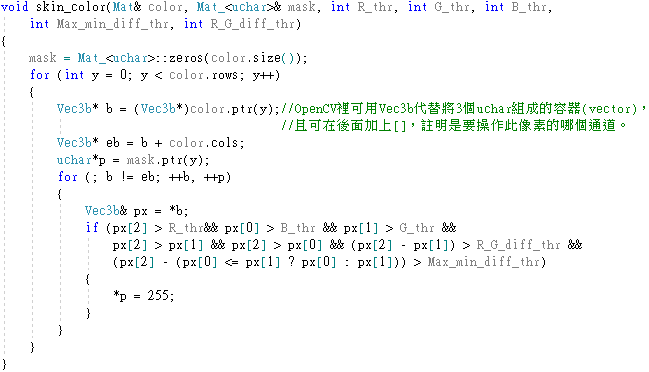
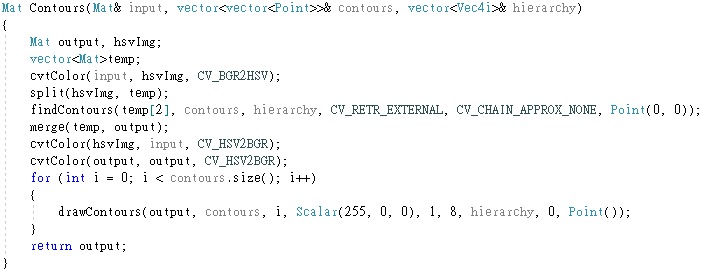
|  |
| --- |
| 機器視覺作業報告 |
| 擴增實境作業 |
|  |
|  |
| 姓名:翁偉恆  學號:00453048  日期:2019/6/9 |
|  |

|  |
| --- |
|  |

# 方法

skin\_color()函式主要用來個別判別輸入影像的每個像素是否為接近皮膚的顏色，其中參數R\_thr為紅色門檻，紅色通道的矩陣元素的值小於紅色門檻就會被過濾為黑色，以此類推。G\_thr為綠色門檻、B\_thr為藍色門檻。R\_G\_diff\_thr為 (紅色-綠色) 的門檻，也就是設定該矩陣元素的紅色通道值要比綠色通道值大多少，以此類推。Max\_min\_diff\_thr為 (紅色-綠色和藍色較大者) 的門檻。



Contours()為自創函式，主要用來找出輸入影像的輪廓。

findContours()為opencv內建函式，第一個參數為欲找出輪廓的圖片，個人認為輪廓主要和亮度有關，所以輸入的圖片一律先從BGR通道轉為HSV通道，取亮度通道，代入findContours()第一個參數。

第二個參數為輸出所有的輪廓點，型態為vector<vector<Point>>

第三個參數是以階層的方式記錄所有輪廓，型態為vector<Vec4i>。

第四個參數為取得輪廓的模式：

CV\_RETR\_EXTERNAL：只取最外層的輪廓。

CV\_RETR\_LIST：取得所有輪廓，不建立階層(hierarchy)。

CV\_RETR\_CCOMP：取得所有輪廓，儲存成兩層的階層，首階層為物件外圍，第二階層為內部空心部分的輪廓，如果更內部有其餘物件，包含於首階層。

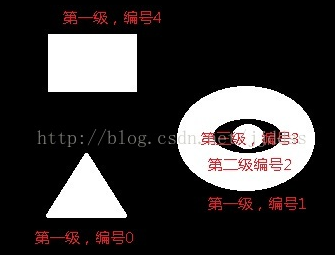
CV\_RETR\_TREE：取得所有輪廓，以全階層的方式儲存。

第四個參數為儲存輪廓點的方法：

CV\_CHAIN\_APPROX\_NONE：儲存所有輪廓點。

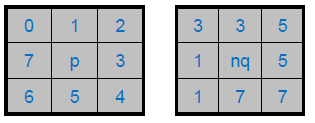
CV\_CHAIN\_APPROX\_SIMPLE：對水平、垂直、對角線留下頭尾點，所以假如輪廓為一矩形，只儲存對角的四個頂點。

階層(hierarchy)的意思，看以下圖便可明瞭



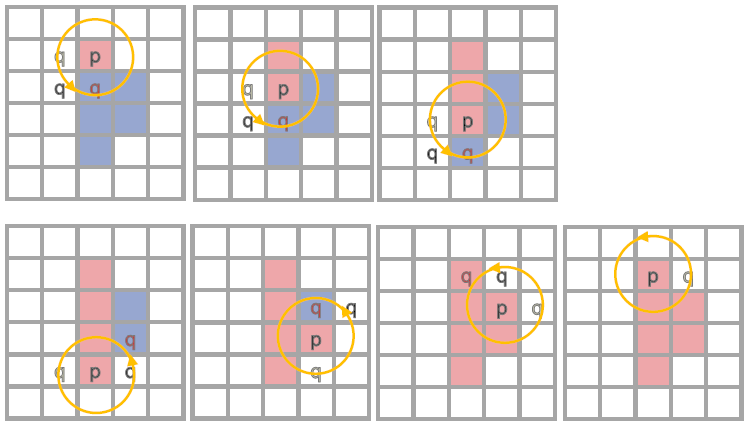
第一階層為最外面的輪廓，第二階層為第二外面的輪廓，以此類推。

findContours()原理：



假設p表示當下欲找出邊界的像素，q為從p矩陣相鄰像素，逆時針搜索的第一個欲判斷是否為非背景點的像素，nq為下一個q像對於下一個p的方向

以下圖為範例：



如圖可見，初始q為從p矩陣的7開始，逆時針搜索的第一個背景像素，如圖，初始q=7，初始p標記為p0

當p從q的方向開始逆時針掃描：

7的方向為背景點

6的方向為背景點

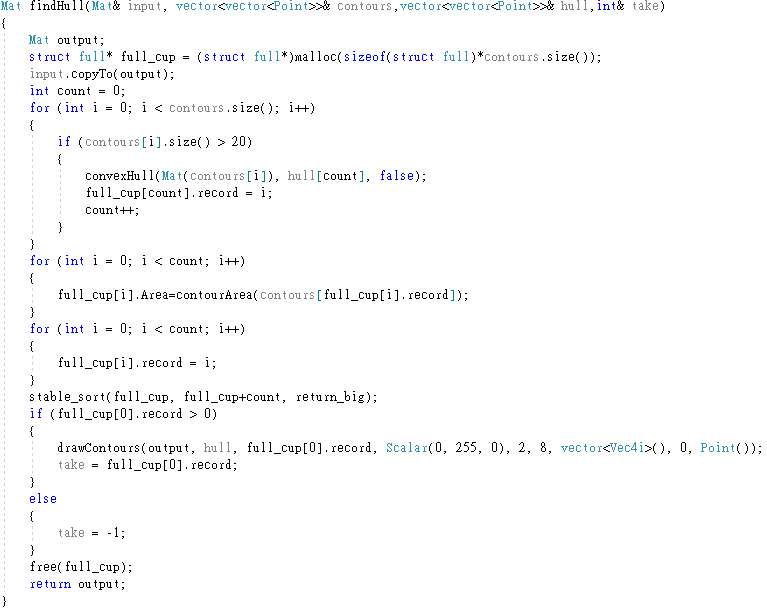
5的方向為背景點

找到第一個非背景點，也就是p像素的下方像素

p往下方移動一像素

對照nq矩陣，往下移動的下一個q為7

以次類推直到p回到p0為止



findHull ()為自創函式，是利用輸入影像的輪廓，找出包含該輪廓的多邊形。

convexHull()為opencv內建函式，第一個參數為欲求出多邊形的所有點集合。第二個參數輸出的多邊形。第三個參數為點集合畫出多邊形的方向，true為順時針，false為逆時針。

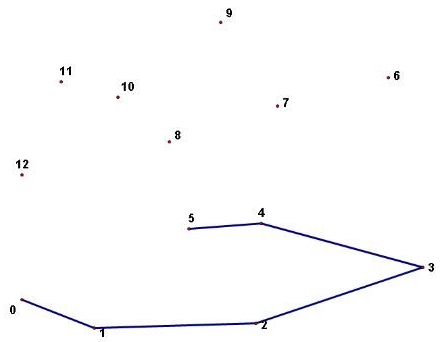
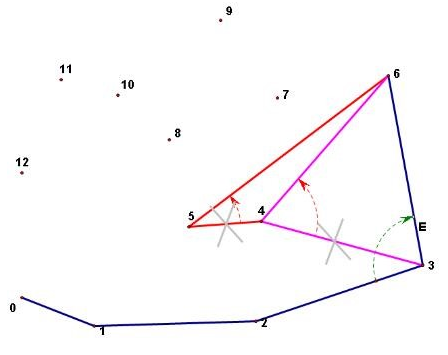
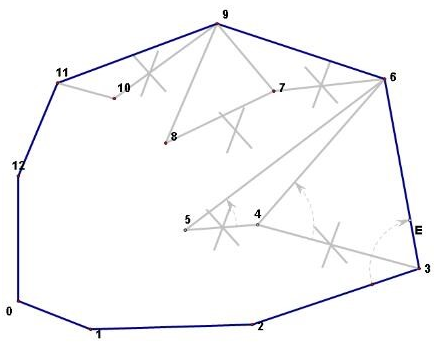
contourArea()為opencv內建函式，主要用來求出輪廓的面積，參數為欲求出面積的輪廓。

drawContours()為opencv內建函式，用來畫出包住輪廓的多邊形。

convexHull()原理：

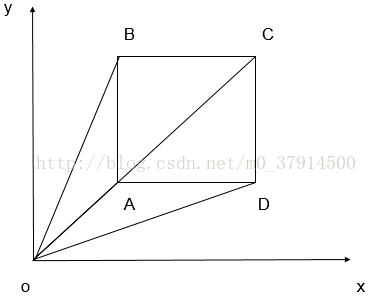
convexHull()主要是利用Graham掃描法，第一步是對點進行排序，將雜亂的點進行梳理，並以逆時針方向編號。

接著按照編號連線，使該點與編號比該點大的點連線後，角度為順時針轉，如下圖

最後形成封閉凸多邊形。

contourArea()原理：



1. 輪廓點為：A（1,1），B（1,2），C（2,2），D（2,1）
2. 兩向量所為出的面積=1/2\*(|兩項量的行列式值|)
3. 根據向量OA和OB的行列式值，可以求出三角形OAB面積
4. 再選取B和C兩點，可計算三角形OBC的面積
5. 三角形OAB和三角形OBC的面積差就是三角形ABC的面積
6. 以此類推

舉例說明：

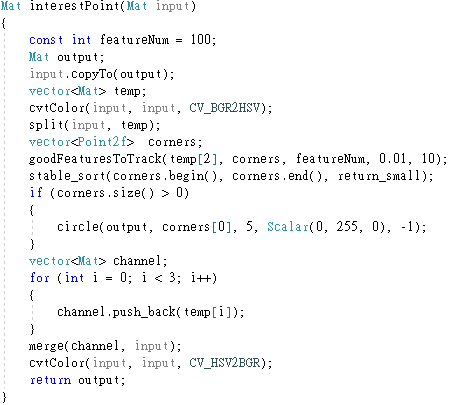
(A,B)：(1/2) \* |(1\*2) - (1\*1)|= 1/2

(B,C)：(1/2) \* |(1\*2)­­ - (2\*2)|= 1

(C,D)：(1/2) \* |(2\*1) - (2\*2)|= 1

(D,A)：(1/2) \* |(2\*1) - (1\*1)|= 1/2

面積=(1 - 1/2) + (1 - 1/2) = 1



interestPoint()為自創函式，主要用來尋找影像角點。

goodFeaturesToTrack()為opencv內建函式，第一個參數為輸入圖像，與尋找輪廓一樣，個人認為角點主要和亮度有關，所以輸入的圖片一律先從BGR通道轉為HSV通道，取亮度通道，代入goodFeaturesToTrack ()第一個參數。

第二個參數為輸出的角點集合，型態為vector<Point2f>。

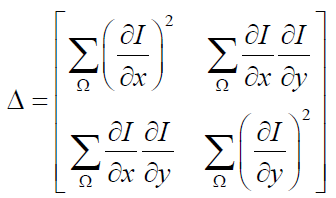
第三個參數為欲尋找角點的數量。

第四個參數為角點品質因數門檻，若找出的角點品質因數小於門檻，則會被刪除。

第五個參數為角點之間的最短距離，單位為像素，如果找出的角點，在其周圍最短距離的範圍內存在其他更強角點，則會將原始角點刪除。

goodFeaturesToTrack()原理：

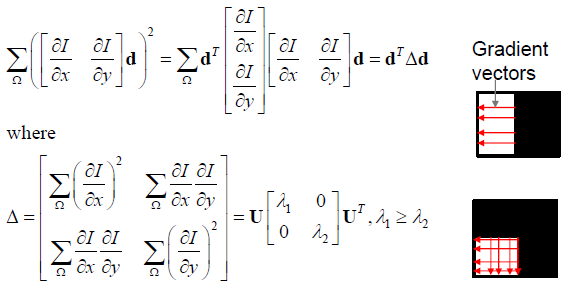
首先利用Harris演算法，也就是對輸入圖像的x方向和y方向進行下列運算，



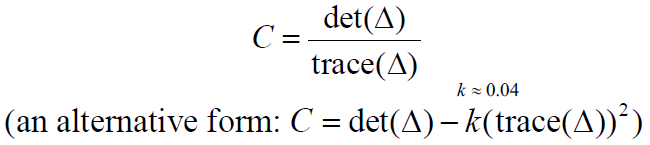
算出的矩陣經過對角化之後，會算出2個特徵值，

如果其中只有1個特徵值不為0，則表示此圖像只有1個方向像素值有變化，判別為邊；

如果2個特徵值皆不為0，則表示此圖像有2個方向像素值有變化，判別為角。

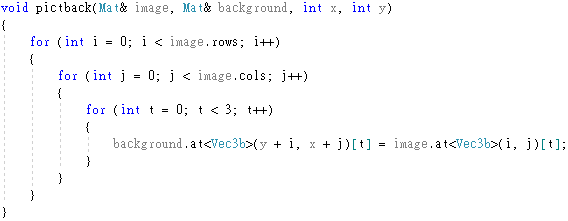
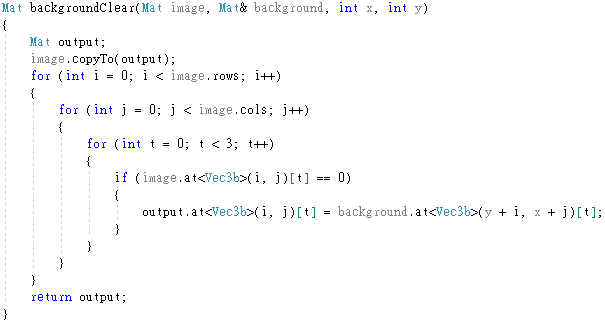


接著經過下列運算，



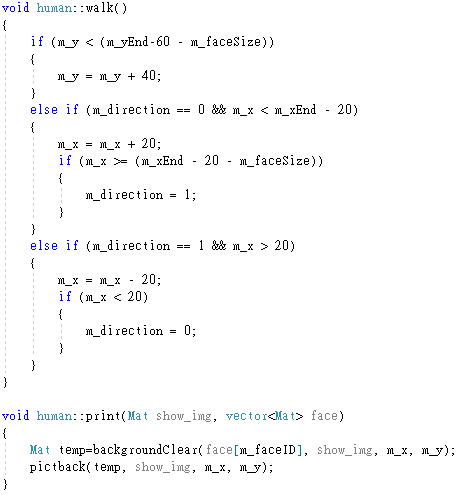
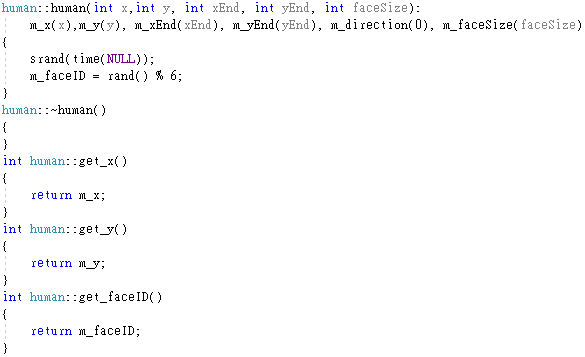
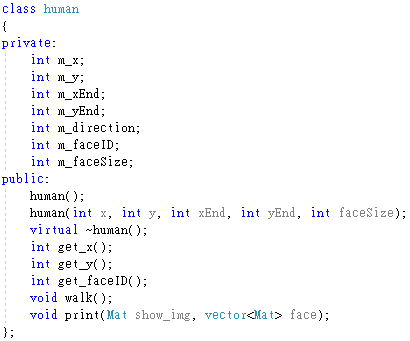
其中，det()為特徵值的積，trace()特徵值的和，得到C值，C值越大代表2個變化方向越垂直。

而goodFeaturesToTrack()則直接用較小的特徵值與角點品質因數門檻值做比較，大於門檻值的角點保留，小於門檻值的角點刪掉。



backgroundClear()為自創函式，參數image為輸入圖片，參數background為輸入背景，參數x和參數y分別為輸入圖片的左上角，向對於輸入背景左上角的(x,y)值，這函式主要是將輸入圖片的背景，改成輸入背景相對應的位置。

pictback()為自創函式，所有參數與backgroundClear()相同，主要是將已經改過背景的輸入圖片，還原到輸入背景相對應的位置上。



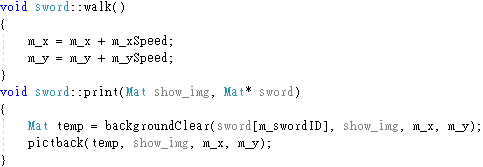
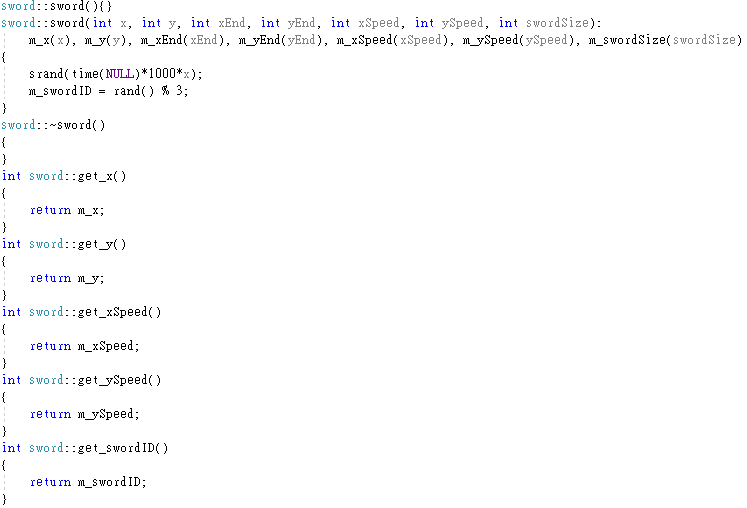
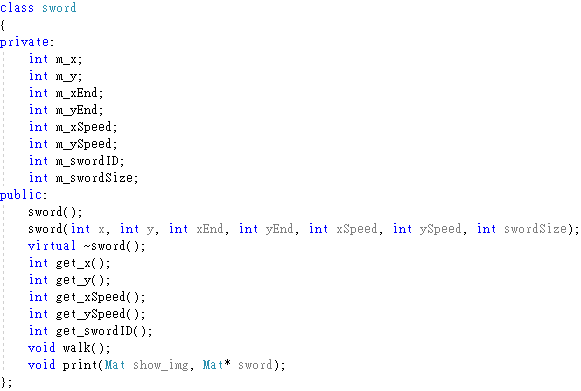
以上為human類別的程式碼

get\_x()和get\_y()為取得human物件相對於背景的位置。

get\_faceID()為取得該human物件的圖片流水號。

walk()描述human物件的移動軌跡。

print()為畫出human物件的圖像。



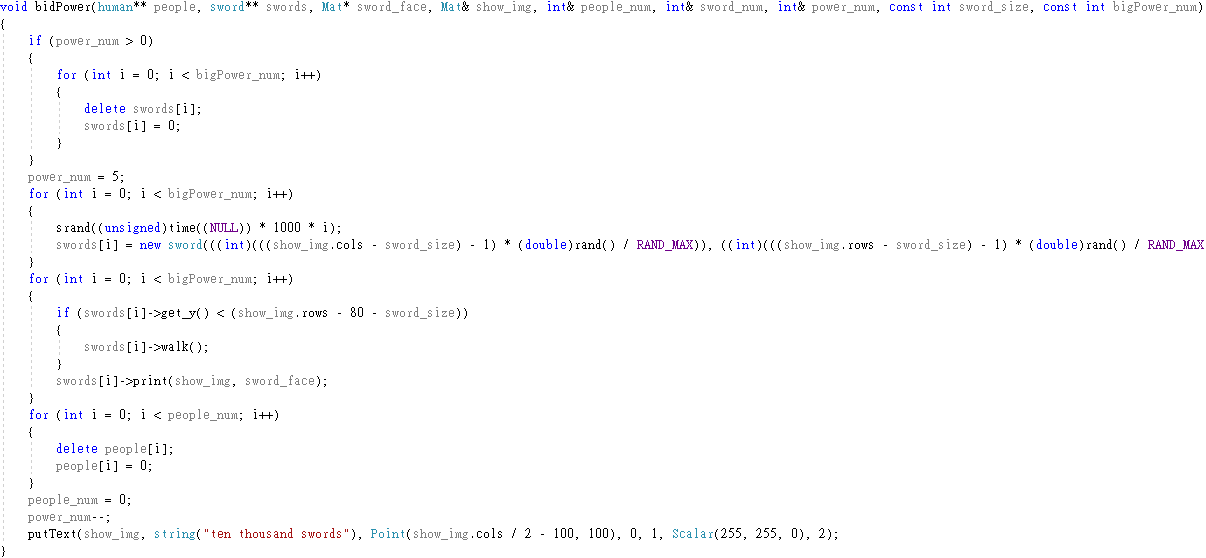
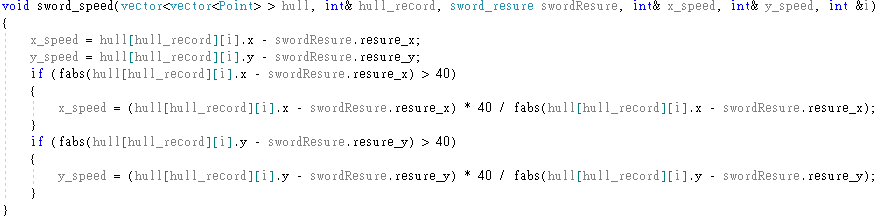
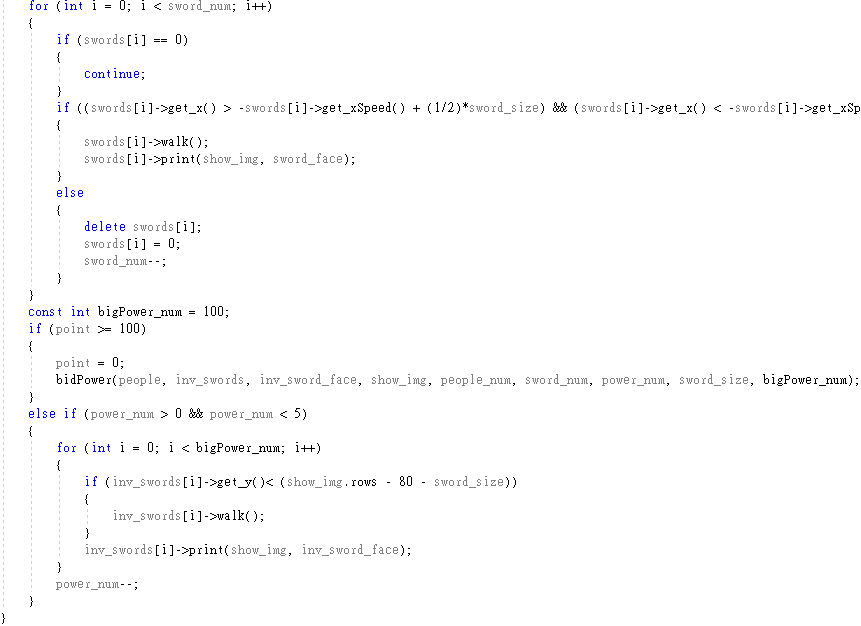
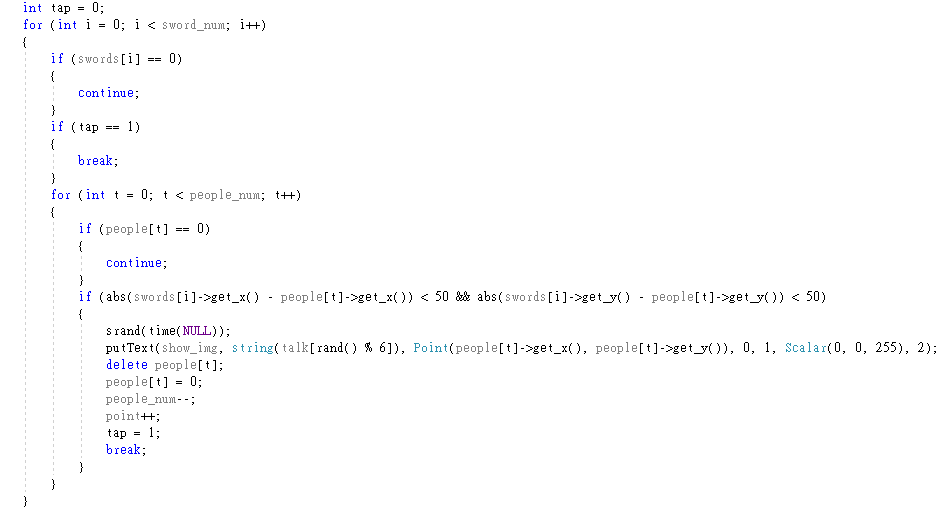
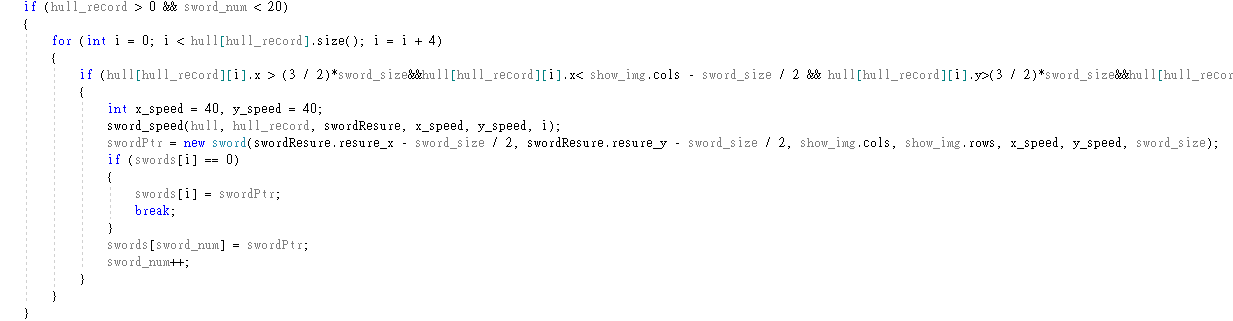
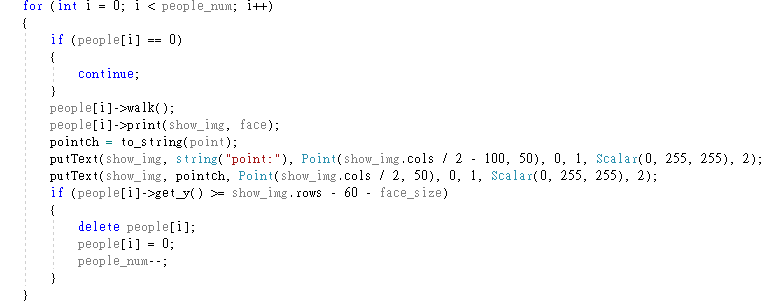
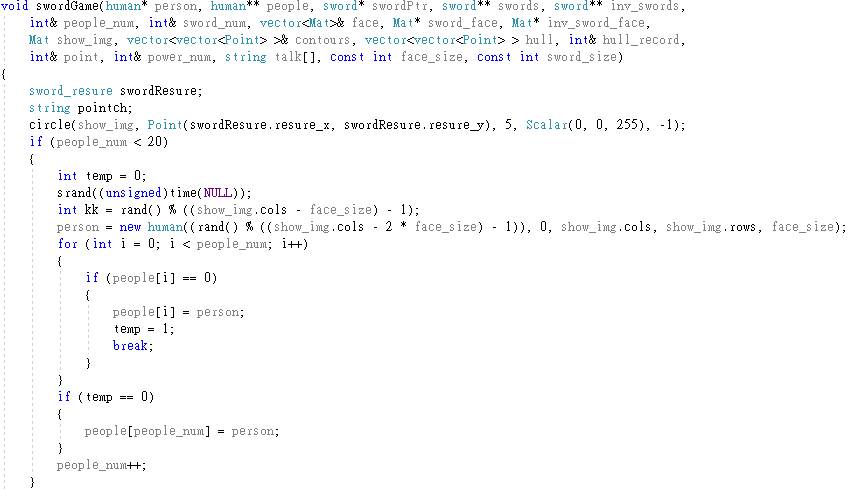
以上為sword類別的程式碼，與human類別大同小異

get\_x()和get\_y()為取得sword物件相對於背景的位置。

get\_swordID()為取得該sword物件的圖片流水號。

walk()描述sword物件的移動軌跡。

print()為畫出sword物件的圖像。

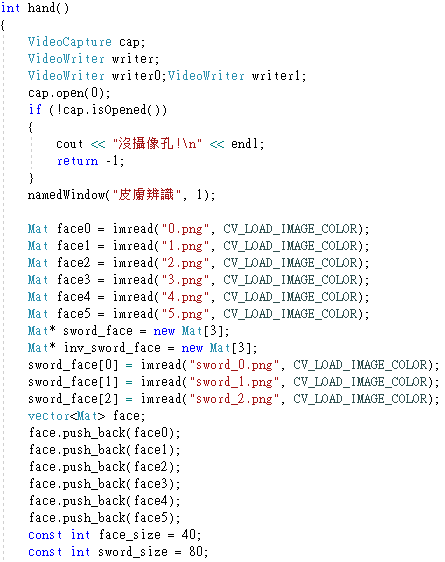
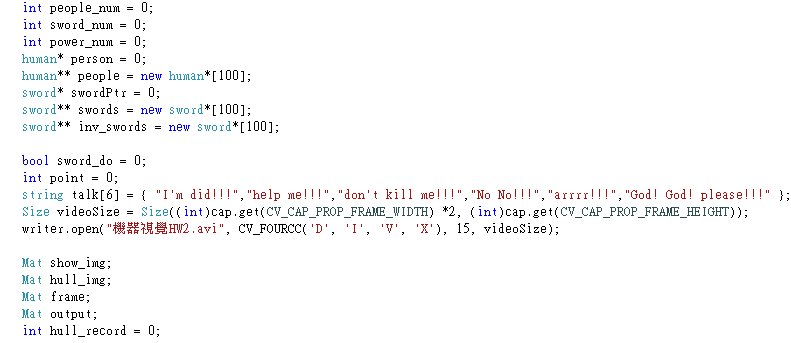
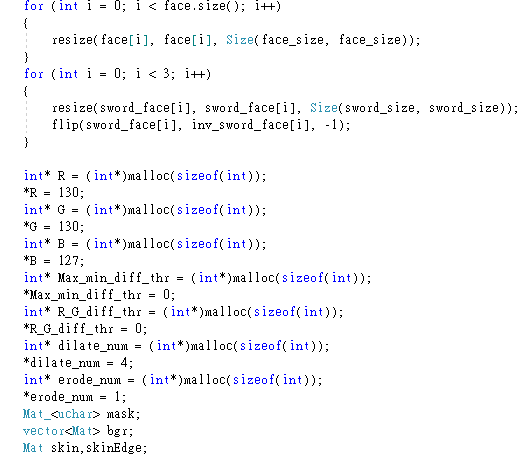
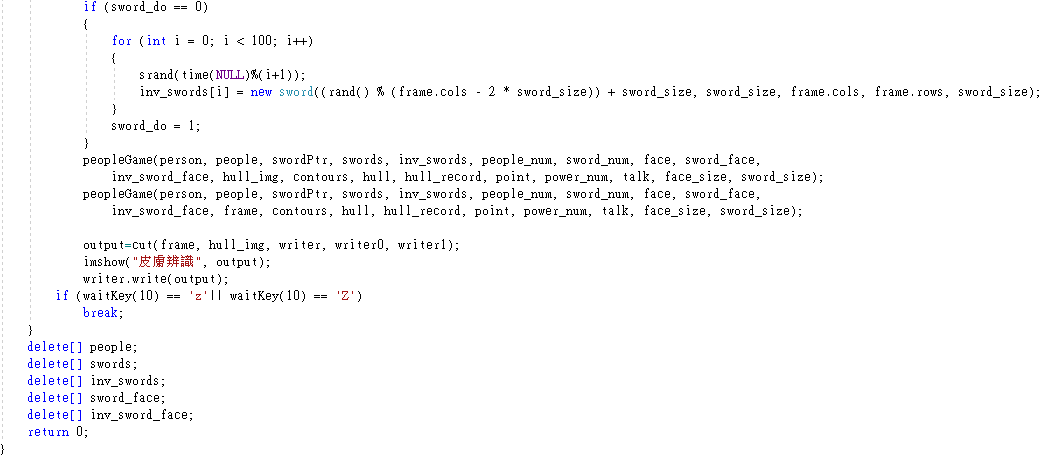


peopleGame()為遊戲主體架構，首先當敵人少於一定數量時(以上程式碼為20)，每秒會多生出(new)1個敵人，並畫出敵人，然後畫面有一個射 ”劍” 中心，與玩家的手指凸點偵測距離與方向，若玩家的手指凸點與射劍中心的像素的x距離大於上限距離時(以上程式碼為40)，該點射出的飛劍會以每秒走上線距離的方式移動；若小於上限距離，則每秒以真實距離移動，y方向亦然。此處玩家的手為用手偵測出輪廓，再以比較輪廓包住的面積，面積最大者的輪廓所預估出的多邊形。

當手與敵人的像素差距小於一定距離時(以上程式碼為5)，會被判定為殺死敵人，此時敵人會被delete，並獲得1分，當分數累積到一定額度時(以上程式碼為100)，便可放出大招。

sword\_speed()為判斷手指與射劍中心的x距離或y距離是否超過上限距離，判斷後，設定飛劍的速度。

bidPower()為大招函式，主要是描述超大量的劍從天而降。

dilate()為opencv內建函式，用來對一圖像的非0像素進行擴張

第一個參數為輸入圖像。

第二個參數為輸出圖像。

第三個參數為mask矩陣，預設的3×3的矩形，mask的size越大膨脹效果越明顯。

第四個參數為原點位置，預設為mask矩陣的中央，也就是Point(-1, -1)。

第五個參數為膨脹迭代次數，3×3的mask迭代2次會等效於3×3的mask迭代1次，但可以把時間複雜度壓縮為(2\*3\*3)/(5\*5)倍，以此類推。

第六個參數為擴增邊界的方法，常用的有：

BORDER\_CONSTANT：將擴增出的像素值設為常數。

BORDER\_REPLICATE：將擴增出的像素值設為未擴增時的邊界值。例如：aaaaaa|abcdefgh|hhhhhhh

BORDER\_REFLECT：將擴增出的像素值設為未擴增時的鏡像投射。例如：fedcba|abcdefgh|hgfedcb

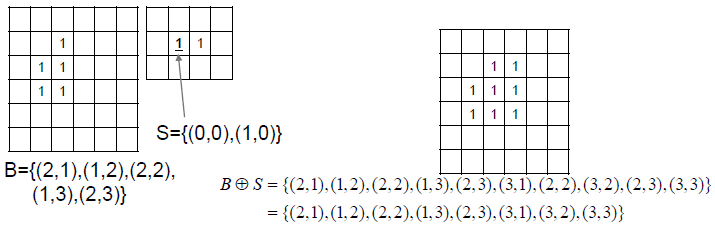
BORDER\_WRAP：將擴增出的像素值設為未擴增時的位移。例如：cdefgh|abcdefgh|abcdefg

預設為BORDER\_CONSTANT

假如第六個參數選擇BORDER\_CONSTANT，那第七個參數便是欲將擴增像素設為的常數值。

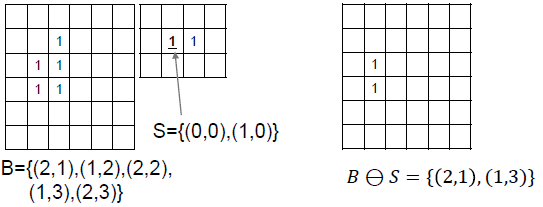
erode()為opencv內建函式，用法與dilate()一模一樣，只是效果從膨脹轉為侵蝕。

dilate()原理：



如圖，矩陣元素分為0與1，0代表像素值為0，1代表像素值為非0，用mask與輸入圖片進行位移運算，然後進行聯集，聯集後，矩陣元素為1的位置保留原像素值，矩陣元素為0的位置像素值設為0

erode()原理：



如圖，矩陣元素分為0與1，0代表像素值為0，1代表像素值為非0，用mask與輸入圖片用mask與輸入圖片進行位移運算，若位移前的矩陣元素值為1，且該位置位移後的矩陣元素值仍為1，則該位置保留原像素值，其餘情況下，該位置的位置像素值設為0

hand()為偵測+遊戲主體結構的結合函式，建議演算步驟如下：

1. 讀入影像。
2. 利用自創函式skin\_color()找出影像接近皮膚的像素，並過濾掉其他像素。
3. BGR通道轉HSV通道，並取出亮度通道。
4. 利用自創函式Contours()找出皮膚亮度通道的輪廓。
5. 利用自創函式findHull()算出哪一個輪廓所為的面積最大，並用一多邊形近似該輪廓(輪廓的點太多，近似的多邊形點較少，且比較集中，加速用)。
6. 將預測出的多邊形路徑的點集合，敵人圖片，劍的圖片…等資訊代入代入遊戲之中。
7. 以上步驟一直循環到玩家按keyboard結束遊戲為止。

# 結果

<https://www.youtube.com/watch?v=60ZEQzDr8hw&list=PLBTefedmJ955eKkALh2rPBbSodZzOiqbt&index=9>此上連結為本遊戲的demo結果，可以看出手指相對於射劍中心的位置能控制飛劍的方向與速度，當手指靠近飛劍的距離小於40像素的位置時，飛劍會以每秒跑手指與射劍中心的像素距離為飛劍的速度；當手指靠近飛劍的距離大於40像素的位置時，飛劍會以每秒跑40像素距離為飛劍的速度。

也不然看出結果還是有很多缺陷，最大的缺陷就是玩此遊戲時，背景一定藥店有黑色物體方能保持遊戲品質，假如有些燈光或物品很接近皮膚色時，也有可能會被辨識成手指，尤其是臉入境的話，往往因為臉也是皮膚色，且所佔有螢幕面積極大，因此常常會被辨識成手。

還有一個問題就是每1人，就會有1種膚色；每100人，就會有100種膚色。若把偵測膚色的門檻設為定值或範圍，我個人認為不是一個好選擇，因此我最後的解決方案是，除了遊戲畫面之外，我額外加了一個遊戲選單，專門及時調配膚色門檻，使不同的玩家能擁有不同的膚色判別。

因此我此遊戲設定以下規則：

1. 臉不能入鏡。
2. 建議在大型黑色物品上進行遊戲，例如：電腦手提包。
3. 玩家可在遊戲的任何時段調整遊戲選單的膚色門檻，以利於不同人的手部辨別

# 結論

說明心得，例如完成甚麼，學到那些技巧。

這本次作業中，我學到了很多讓我覺得很棒的演算法，例如：Harris演算法、Graham掃描法…等。

並清楚的了解findContours()、convexHull()、contourArea()、goodFeaturesToTrack()、dilate()、erode()這些函式背後的原理，特別是contourArea()，竟然是用高中的向量面積公式，便可迎刃而解，讓我越來越對數學及演算法產生敬畏之心。

我覺得這次的作業發揮的空間極大，也具有一定的難度，特別是偵測手的演算法，我思考了好久，我一開始是希望手和頭一起入鏡時，能單純用演算法就能分辨出手和頭的區別，但我在網路上找了好久，都沒有一個單純用演算法的範例是頭有入鏡的，最後還因此跑來問經驗豐富的老師，老師說這種情況大概只有分類器或深度學習能解決，讓我一則以喜，一則以憂。喜的是我終於不用再在網路上爬文了，憂的是我錄影像時，頭大概不能入鏡了(我的電腦跑不動分類器或深度學習)，這讓我覺得這次的作品不夠完美。

# 參考文獻

(如果有參考文獻, 請列出)

<http://monkeycoding.com/?p=615>

<http://monkeycoding.com/?p=577>

<https://blog.csdn.net/CorCplusplusorjava/article/details/20536251>

<https://blog.csdn.net/dcrmg/article/details/51987348>

<https://blog.csdn.net/lbaihao/article/details/78921368>

<https://blog.csdn.net/xdfyoga1/article/details/44175637>

<https://blog.csdn.net/helei001/article/details/23602449>

<https://blog.csdn.net/jjddss/article/details/73527990>

<https://blog.csdn.net/luoshixian099/article/details/48244255>

<https://blog.csdn.net/keith_bb/article/details/70194073>

<https://blog.csdn.net/m0_37914500/article/details/78615284>

上課講義