高等電腦視覺 作業二

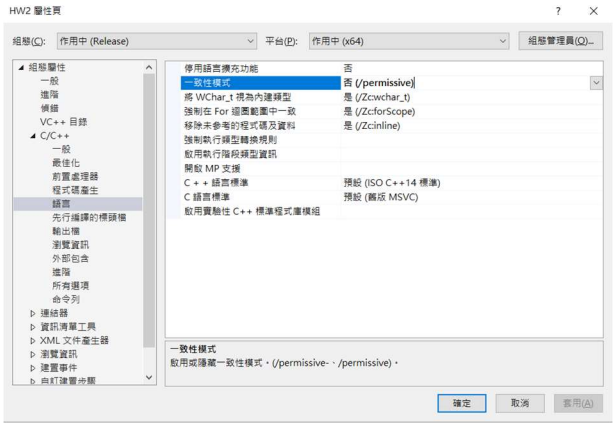
姓名: 陳祐毅　學號:109618028

使用 C 語言

編譯環境 : Visual Studio 2019 (16.11.5)

如果遇到 const char to char 相關的 error，請在

屬性 > C/C++ > 語言 > 一致性模式　改成否



會自動建立 ./ Image_no_cv2/ 資料夾，存放所有照片

引用函式庫

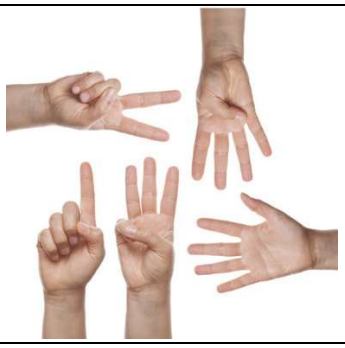
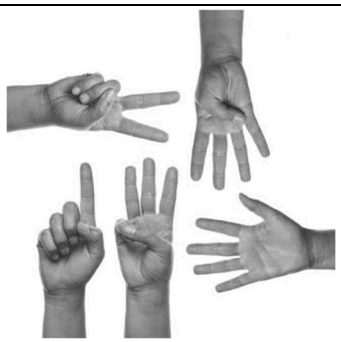
```
#include<stdio.h>
#include<stdlib.h>
#include <io.h>   //創 img資料夾用
#include <direct.h>  //創 img資料夾用
#include<math.h>
#include<time.h>  //計算時間
#define PI 3.1415926
```

以下有些型態、函數看起來會跟 opencv 一樣，是因為我想藉此熟悉 opencv 語法，不要做兩個部分的作業要記不同名稱，也可藉此熟悉 opencv 語法。
(以下皆使用純 C 寫)

步驟一　彩色轉灰階

使用公式 Gray = 0.299 * Red + 0.587 * Green + 0.114 * Blue 將三個通道都改成了一樣的數值，後面的所有函數都會保留 RGB 三通道，但數值一樣。

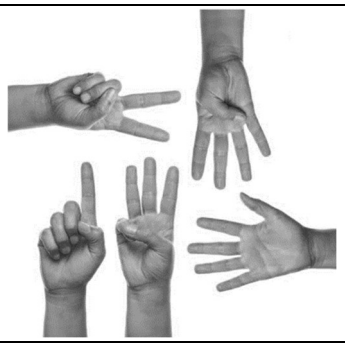
```
// 灰階
Mat color2Gray(Mat* src) {
    Mat out = copyImg(*src);
    for (int i = 0; i < src->rows; i++) {
        for (int j = 0; j < src->cols; j++) {
            out.image[0][i][j] = src->image[0][i][j] * 0.114+ src->image[0][i][j]*0.587+ src->image[0][i][j]*0.299;
            out.image[1][i][j] = out.image[0][i][j];
            out.image[2][i][j] = out.image[0][i][j];
        }
    }
    return out;
}
```

原圖	灰階
	

步驟二 二值化

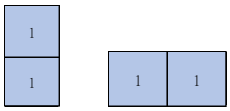
這裡我只採用自己手調閥值的方式實現，由於照片背景是白色，所以設置一個參數 inverse='t' 時，可以讓黑白互換。

```
// 二值化
Mat binarizing(Mat* src, uch thres_value, char inverse) {
    Mat out = copyImg(*src);
    for (int i = 0; i < src->rows; i++) {
        for (int j = 0; j < src->cols; j++) {
            if (src->image[0][i][j] > thres_value)
                out.image[0][i][j] = 255;
            else
                out.image[0][i][j] = 0;

            if (inverse == 't') {
                out.image[0][i][j] = out.image[0][i][j]^0xff;
            }
            else {
                out.image[0][i][j] = out.image[0][i][j];
            }

            out.image[1][i][j] = out.image[0][i][j];
            out.image[2][i][j] = out.image[0][i][j];
        }
    }
    return out;
}
```

灰階	二值化
	

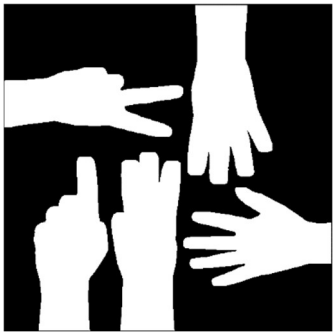
步驟三 形態學

形態學我採用上課教的方式，使用最小的 mask



利用這兩種 mask，經過反覆操作，比起較大的 mask，可以有效降低運算時間。(不過使用這兩種 mask 有一方面是程式比較好寫)
在函式的設定中可以藉有 direction 選擇 mask，然後 iteration 可以決定操作次數。

```
Mat dilate(Mat* src, const char direction, int iteration);
Mat erode(Mat* src, const char direction, int iteration);
```

Dilate(水平方向)	Erode(垂直方向)
	

在這裡我先對二值化的影像作侵蝕再膨脹，主要是消除雜訊，所以處理過的圖片看起來跟二值化沒什麼不同(在 bonus 會再使用迭代比較多次的形態學，就可以看出差異)。

```
//morphology

Mat morphoImg;
int iter = 5;

morphoImg = erode(&binaryImg, 'v', iter);   // 垂直方向
morphoImg = erode(&morphoImg, 'h', iter);   // 水平方向
morphoImg = dilate(&morphoImg, 'v', iter);
morphoImg = dilate(&morphoImg, 'h', iter);
```

二值化	形態學
	

步驟四 連通元件

這裡我採用的方法跟上課講的比較不一樣，記憶中，老師提供了幾種方法，一種是來回掃好幾次，一種是掃第一次記錄哪幾個標籤是一樣的，第二次把他們更正。

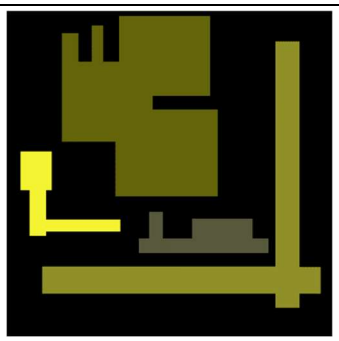
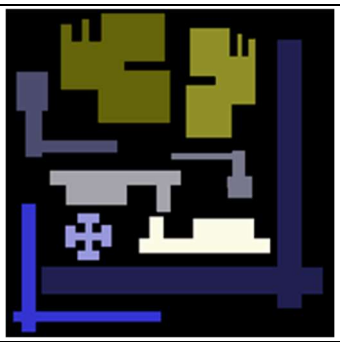
我使用的方法是，只掃一次，但當遇到衝突點時(如下圖)，用**回溯法**立即處理，將比較大的數字改成比較小的數字。比如下圖中，衝突點是 1、2 衝突，那就會以遞迴的方式將所有的 2 改成 1。

				1
		2		1
2	2	2		1
	2			1
	2	2	2	衝突點

回溯法會判斷當前所在像素的**上面**、**左邊**、**右邊**像素是否為需要修改的標籤(會多判斷右邊是因為上圖的狀況)。

在回溯法中並沒有刻意處理可能溢位的問題，因為假使溢位遇到的標籤不符合，並不會處理；符合的話本來就要處理，而且每一點進入即會修改標籤，所以每一像素點並不會重複進入。

```
//回溯法
void collision_func(Mat *ptr,int r, int c, int collision_keep, int collision_del) {
    ptr->image[0][r][c] = collision_keep;
    if (ptr->image[0][r - 1][c] == collision_del) {
        collision_func(ptr, r - 1, c, collision_keep, collision_del);
    }
    if (ptr->image[0][r][c - 1] == collision_del) {
        collision_func(ptr, r, c-1, collision_keep, collision_del);
    }
    if (ptr->image[0][r][c+1] == collision_del) {
        collision_func(ptr, r, c+1, collision_keep, collision_del);
    }
}
```

Test1	Test2
	

沒上色的有些像素質太低，所以放了一張上色過的 ：

形態學	連通元件(上色過)	連通元件(無上色)
		

這裡要小抱怨一下，寫連通元件花了很多時間在除錯，結果發現自己把一個變數 j 寫成了 i，導致結果是錯的…，果然還是要細心一點。

在做連通元件時，我也把性質分析寫在同一個函式內，所以在除錯過成中，可以透過 command window 上的回饋知道，照片結果的確是對的。

步驟五 性質分析

我將 area、centroid、length、orientation 接歸類在性質分析
這部分為了方便，所以設置了一個 Structure 做紀錄:

```
typedef struct {
    int* area;           // 面積
    double** centroids;  // 質心
    int** box;           // box的左上、右下座標
    int* label_list;     // 每隻手連通元件給予的標籤值
    int num;             // 總共判別出幾隻手
    int* longest;        // 最長邊的長度
    int* angle;          // 角度
}Connected;
```

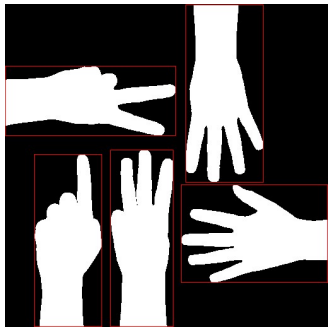
大部分的紀錄過程，是採用一次的迴圈將每個點掃過，然後動態修正 struct 裡面的數值。
area 會判斷這個 pixel 值是在 label_list 的哪個 index 裡，然後 area[index]++;
Box 會逐一比較每個像素值的 row、col 大小
Centroids 也會動態的將每個 row、col 放進去平均。(當然使用的演算法就不會是單純的 所有個數和/總個數)
num 在連通元件時就會記錄 : num = 給新標籤數 – 遇到衝突數
longest 應該就是作業要的 length 吧!? 我取的是 box 最長的那一邊
angle 應該是作業要的角度吧!? 我取的是 artan(height/width)
記錄完會將所有數值顯示在 command window 上 :

Microsoft Visual Studio 偵錯主控台

```
物件1
        area : 20518,   centroid : (138.074520, 337.207281),
        long : 281,     與水平軸的 angle : 66 度
        box座標 : 左上(1, 286), 右下(282, 409)
物件2
        area : 17527,   centroid : (151.998802, 118.995036),
        long : 270,     與水平軸的 angle : 22 度
        box座標 : 左上(98, 0), 右下(208, 270)
物件3
        area : 20146,   centroid : (377.959794, 218.698302),
        long : 280,     與水平軸的 angle : 70 度
        box座標 : 左上(231, 167), 右下(511, 266)
物件4
        area : 17617,   centroid : (393.349776, 98.207924),
        long : 273,     與水平軸的 angle : 68 度
        box座標 : 左上(238, 46), 右下(511, 153)
物件5
        area : 18378,   centroid : (370.363750, 400.497497),
        long : 232,     與水平軸的 angle : 33 度
        box座標 : 左上(286, 279), 右下(441, 511)
共5個物件
```

步驟六 Drawing

由於上一步已經在性質分析中記錄下 Bounding Box 的左上、右下座標，所以這部分就只要利用紀錄好的座標畫出來就好 ：

連通元件(無上色)	Bounding Box
	

步驟七 Bonus 形態學

我們利用侵蝕、膨脹將所有的手指侵蝕掉:

二值化	侵蝕掉手指
	

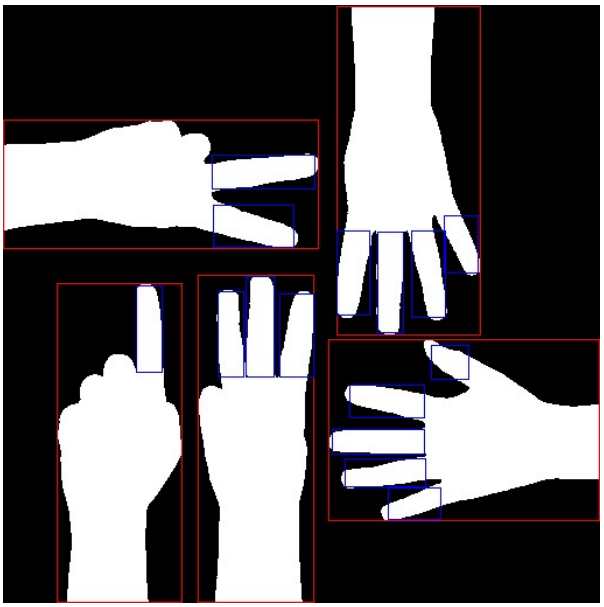
然後使用 二值化影像 減去 侵蝕掉手指的影像，在做一點侵蝕膨脹去掉雜訊。由於 C 語言沒辦法做運算子多載，所以還要另外寫一個函式來做圖片運算。

手指影像


步驟八 Bonus 連通元件

這個步驟跟前面的步驟四、五、六一樣，故只顯示結果:

Box 影像



步驟九 手指歸屬分析

接下來要判別哪隻手有幾隻手指，我寫了一個函示去判斷哪隻手指的質心在哪隻手的 Box 範圍內，則那隻手擁有的手指數加 1。
為了減少麻煩，所以我在性質分析用的結構增加了指向整數的指標來記錄。

```
typedef struct {
    int* area;           // 面積
    double** centroids;  // 質心
    int** box;           // box的左上、右下座標
    int* label_list;     // 每隻手連通元件給予的標籤值
    int num;             // 總共判別出幾隻手
    int* longest;        // 最長邊的長度
    int* angle;          // 角度
    int* finger_num;     // 每隻手的手指數量
}Connected;
```

```
//計算每隻手掌有幾隻手指
void cal_hands_fingers(Connected& hands, Connected& fingers) {
    hands.finger_num = (int*)calloc(hands.num, sizeof(int));


    for (int i = 0; i < hands.num; i++) {
        for (int j = 0; j < fingers.num; j++) {
            if (fingers.centroids[j][0] > hands.box[i][0] && fingers.centroids[j][0] < hands.box[i][2]) {
                if (fingers.centroids[j][1] > hands.box[i][1] && fingers.centroids[j][1] < hands.box[i][3]) {
                    hands.finger_num[i]++;    //手指數量 + 1
                }
            }
        }
    }
}
```

步驟十 手指歸屬分析

我將所有結果改寫在 show()、show_fingers() 函式內，最後都會印在 command windows 內(如下圖) :

```
void show(Connected* reco) {
    for (int i = 0; i < reco->num; i++) {
        printf("\n手掌%d:   手指數 : %d\n\t", i + 1,reco->finger_num[i]);
        printf("area : %d,\t", reco->area[i]);
        printf("centroid : (%f, %f),\n", reco->centroids[i][0], reco->centroids[i][1]);
        printf("\tlong : %d,\t", reco->longest[i]);
        printf("與水平軸的 angle : %d 度\n", reco->angle[i]);
        printf("\tbox座標 : 左上(%d, %d), 右下(%d, %d)", reco->box[i][0], reco->box[i][1], reco->box[i][2], reco->box[i][3]);
    }
    printf("\n共%d隻手掌\n\n", reco->num);
}
```

Microsoft Visual Studio 偵錯主控台

```
手掌1:   手指數 : 4
        area : 20518,   centroid : (138.074520, 337.207281),
        long : 281,     與水平軸的 angle : 66 度
        box座標 : 左上(1, 286), 右下(282, 409)
手掌2:   手指數 : 2
        area : 17527,   centroid : (151.998802, 118.995036),
        long : 270,     與水平軸的 angle : 22 度
        box座標 : 左上(98, 0), 右下(208, 270)
手掌3:   手指數 : 3
        area : 20146,   centroid : (377.959794, 218.698302),
        long : 280,     與水平軸的 angle : 70 度
        box座標 : 左上(231, 167), 右下(511, 266)
手掌4:   手指數 : 1
        area : 17617,   centroid : (393.349776, 98.207924),
        long : 273,     與水平軸的 angle : 68 度
        box座標 : 左上(238, 46), 右下(511, 153)
手掌5:   手指數 : 5
        area : 18378,   centroid : (370.363750, 400.497497),
        long : 232,     與水平軸的 angle : 33 度
        box座標 : 左上(286, 279), 右下(441, 511)
共5隻手掌
```

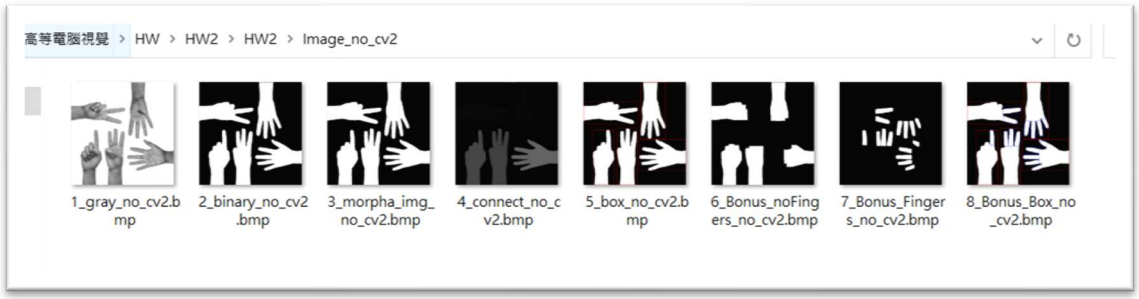
Microsoft Visual Studio 偵錯主控台

```
手指的性質
手指1:
        area : 1968,    centroid : (141.747459, 221.175813),
        long : 88,      與水平軸的 angle : 18 度
        box座標 : 左上(128, 179), 右下(157, 267)
手指2:
        area : 1607,    centroid : (190.065339, 212.538270),
        long : 69,      與水平軸的 angle : 27 度
        box座標 : 左上(171, 180), 右下(207, 249)
手指3:
        area : 884,     centroid : (203.311086, 393.042986),
        long : 49,      與水平軸的 angle : 59 度
        box座標 : 左上(180, 378), 右下(229, 407)
手指4:
        area : 1585,    centroid : (227.121136, 298.946372),
        long : 72,      與水平軸的 angle : 68 度
        box座標 : 左上(193, 286), 右下(265, 314)
手指5:
        area : 1454,    centroid : (228.228336, 365.383081),
        long : 74,      與水平軸的 angle : 68 度
        box座標 : 左上(193, 350), 右下(267, 379)
手指6:
        area : 1815,    centroid : (234.806612, 331.141047),
        long : 86,      與水平軸的 angle : 75 度
        box座標 : 左上(194, 321), 右下(280, 343)
手指7:
        area : 1908,    centroid : (277.582285, 220.716457),
        long : 86,      與水平軸的 angle : 74 度
        box座標 : 左上(232, 208), 右下(318, 232)
手指8:
        area : 1562,    centroid : (278.921255, 124.494238),
        long : 74,      與水平軸的 angle : 73 度
        box座標 : 左上(240, 114), 右下(314, 136)
手指9:
        area : 1454,    centroid : (283.319120, 193.974553),
        long : 73,      與水平軸的 angle : 73 度
        box座標 : 左上(245, 184), 右下(318, 206)
手指10:
        area : 1620,    centroid : (284.738272, 252.909259),
        long : 71,      與水平軸的 angle : 67 度
        box座標 : 左上(247, 237), 右下(318, 266)
手指11:
        area : 695,     centroid : (305.166906, 383.939568),
        long : 32,      與水平軸的 angle : 42 度
        box座標 : 左上(291, 367), 右下(320, 399)
手指12:
        area : 1322,    centroid : (338.097579, 330.572617),
        long : 64,      與水平軸的 angle : 23 度
        box座標 : 左上(325, 297), 右下(353, 361)
手指13:
        area : 1610,    centroid : (373.245963, 323.305590),
        long : 79,      與水平軸的 angle : 14 度
        box座標 : 左上(363, 282), 右下(384, 361)
手指14:
        area : 1263,    centroid : (401.117973, 329.201900),
        long : 69,      與水平軸的 angle : 19 度
        box座標 : 左上(388, 293), 右下(412, 362)
手指15:
        area : 797,     centroid : (426.696361, 353.248432),
        long : 45,      與水平軸的 angle : 30 度
        box座標 : 左上(413, 330), 右下(440, 375)
共15隻手指
```

步驟十一 時間計算

時間計算使用了 time.h 函式庫 (連通元件的時間計算寫在 connetedComponent 函式內部，因為我的性質分析跟四連通寫在一起。)

所有的照片會自動生成在 ./image_no_cv2 資料夾內

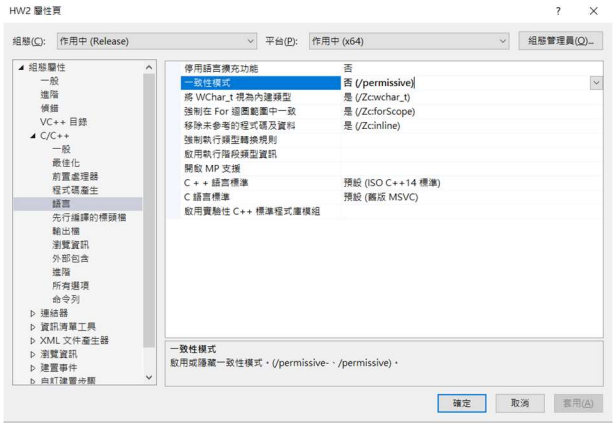


使 Opencv

編譯環境 ：Visual Studio 2019 (16.11.5) 、 opencv (4.4.0)

如果遇到 const char to char 相關的 error，請在

屬性 > C/C++ > 語言 > 一致性模式 改成否



會自動建立 ./ Image_ cv2/ 資料夾，存放所有照片

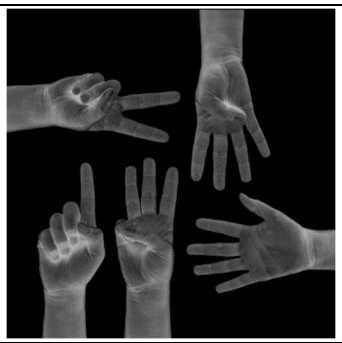
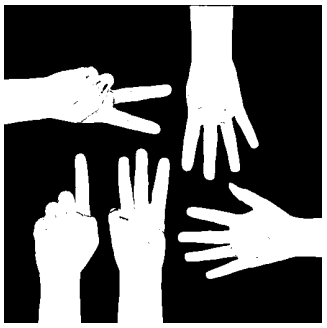
引用函式庫

```
#include<opencv2/opencv.hpp>
#include<opencv2/highgui.hpp>
#include<iostream>
#include<string>
#include <io.h>   //創 img資料夾用
#include <direct.h>  //創 img資料夾用
#include<ctime>
```

步驟一 二值化

使用 opencv 內的 threshold 函式就可以將圖片二直化。我用的是 Otsu 法。

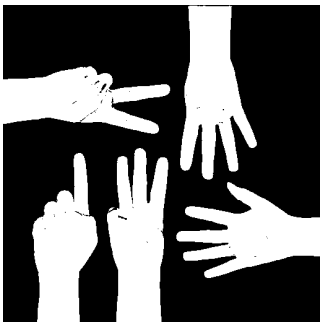
```
// 二值化 Otsu
clock_t start = clock();
Mat otsu_img;
threshold(gray_img, otsu_img, 0, 255, THRESH_OTSU);
clock_t end = clock();
mytime.binarizing = (double)(end - start) / CLK_TCK;
```

原圖	灰階	二值化(otsu)
		

步驟二 型態學

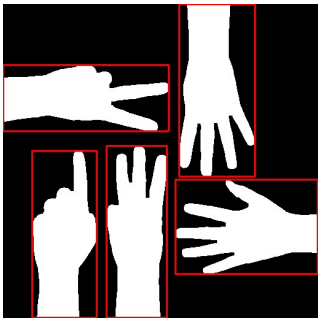
這裡使用 dilate、erode 函式

```
// 侵蝕、膨脹  morphology
start = clock();
dilate(otsu_img, otsu_img, Mat());
dilate(otsu_img, otsu_img, Mat());
erode(otsu_img, otsu_img, Mat());
erode(otsu_img, otsu_img, Mat());
end = clock();
mytime.morphology = (double)(end - start) / CLK_TCK;
imwrite("./Image_cv2/3_morphalogyImg.bmp", otsu_img);
```

二值化	型態學
	

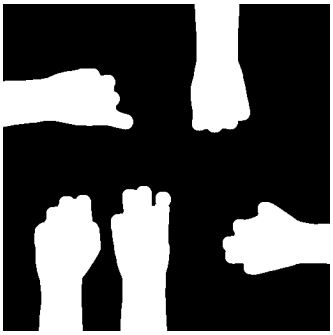
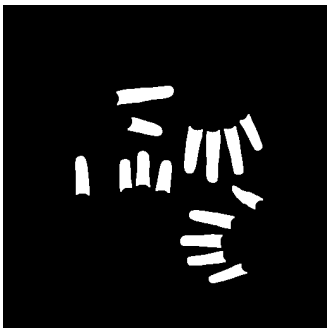
步驟三 Box

這裡使用 connectedComponentsWithStats 函式，不過用完就會連性質分析一起做了。所以直接展示最後成果。(程式碼有點長就不貼了)

型態學	Box
	

步驟四 Bonus

Bonus 步驟跟 C 語言的其實一樣，先侵蝕掉手指再用原圖減去剩下手掌的圖就可以得到手指圖片。

無手指圖	手指圖
	

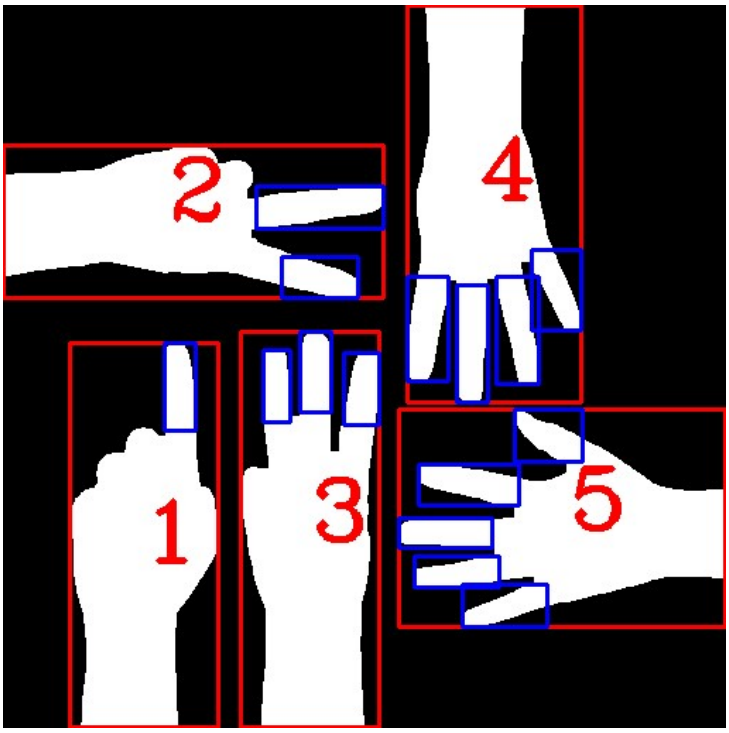
這裡使用的 mask 比較不一樣，是一個(24, 24)的橢圓矩陣，所以手指的下緣都會有圓弧狀。

```
Mat mask = getStructuringElement(MORPH_ELLIPSE, Size(24, 24));
```

```
C:\Users\Yuyi\OneDrive\桌面\cource\高等電腦視覺\HW\HW2\HW2_opencv\x64\Release\HW2_c
手掌 :
centroid :(337.103,136.433) , area : 20147, long :282, orientation: 90度
centroid :(117.773,151.835) , area : 17088, long :270, orientation: 0度
centroid :(218.707,378.711) , area : 19765, long :281, orientation: 90度
centroid :(98.2124,393.834) , area : 17363, long :273, orientation: 90度
centroid :(401.498,370.431) , area : 17899, long :232, orientation: 0度

時間 :
binarizing : 0.003000 秒
morphology : 0.001000 秒
connectedComponent : 0.005000 秒
propertyAnalysis : 0.005000 秒
drawing : 0.006000 秒

手指 :
centroid :(222.539,141.442) , area : 1836, long :91, orientation: 0度
centroid :(392.682,202.112) , area : 928, long :58, orientation: 90度
centroid :(224.348,193.791) , area : 1013, long :55, orientation: 0度
centroid :(298.621,228.614) , area : 1462, long :75, orientation: 90度
centroid :(365.696,229.172) , area : 1357, long :77, orientation: 90度
centroid :(331.112,237.958) , area : 1628, long :84, orientation: 90度
centroid :(221.005,259.949) , area : 1061, long :58, orientation: 90度
centroid :(124.34,271.198) , area : 1183, long :63, orientation: 90度
centroid :(192.485,268.828) , area : 835, long :52, orientation: 90度
centroid :(255.022,271.979) , area : 983, long :52, orientation: 90度
centroid :(384.337,304.272) , area : 824, long :49, orientation: 0度
centroid :(329.848,337.794) , area : 1286, long :72, orientation: 0度
centroid :(314.122,373.26) , area : 1171, long :67, orientation: 0度
centroid :(321.464,402.53) , area : 961, long :61, orientation: 0度
centroid :(354.372,426.438) , area : 896, long :61, orientation: 0度
```

接著使用 connectedComponentsWithStats 函式找手指的 Box，然後畫在有手掌 box 的圖片上，並將手指的數量顯示再手掌的質心上。

找手指數量的方法是使用手掌的 Box 判斷有幾根手指的 centroid 在 Box 內部。

Bonus Img


所有的 opencv 圖片接放在 ./ Image_cv2 裡面

運算時間表格

這裡的 OpenCV 連通元件與性質分析因為只用一個函式，所以時間一樣，實際上是兩個步驟加起來為 0.004 秒。

Time(sec)	binarizing	morphology	connected component	property analysis	drawing
C program	0.006	0.061	0.001	0.002	0.002
OpenCV	0.002	0.001	0.004	0.004	0.005

沒想到我的連通元件竟然可以比 opencv 快，可能是因為我只需要一次 for 迴圈就可以完成吧，不過當初在設計時很怕遞迴的數量太多會占滿記憶體空間導致當掉，有可能遇到大圖會遇到這種問題，但目前這個 case 是蠻好用的。

記得第一次寫作業，我是先從 opencv 著手，不過這次竟然會想從 C 語言開始寫，雖然使用套件很快速簡單，但自己建照一支屬於自己的程式還蠻有成就感的 XD。不過作業太晚開始寫，有些想更正的想法來不及實行，就放在下一次吧。比如有些部分想了很久才發現，配置的記憶體沒有 free，但對於結構內的指標，要全 free 好像還要研究一下。還有發現 C 語言畢竟不是物件導向，有很多東西都要另寫函式，比如 C++有運算子多載，C 語言沒有，所以做矩陣運算就必須另寫函式。
最後，希望每次作業我的程式能力都能有所提升，我好像把作業當日記在寫呢，謝謝。