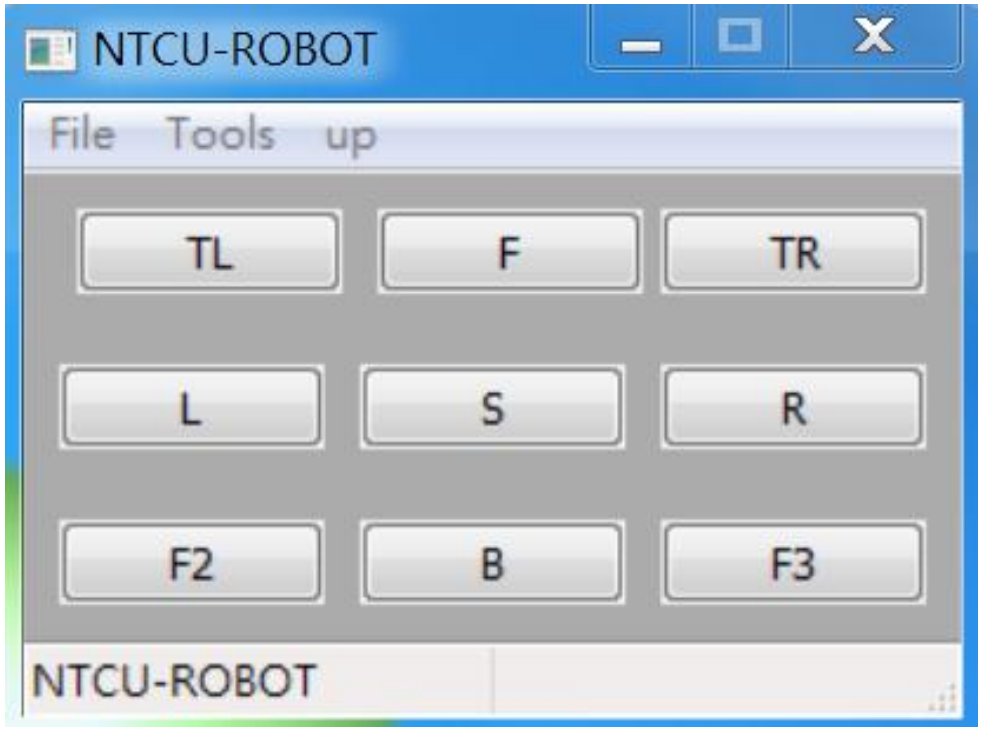
基於類神經網路動作辨識之Kinect機器人控制

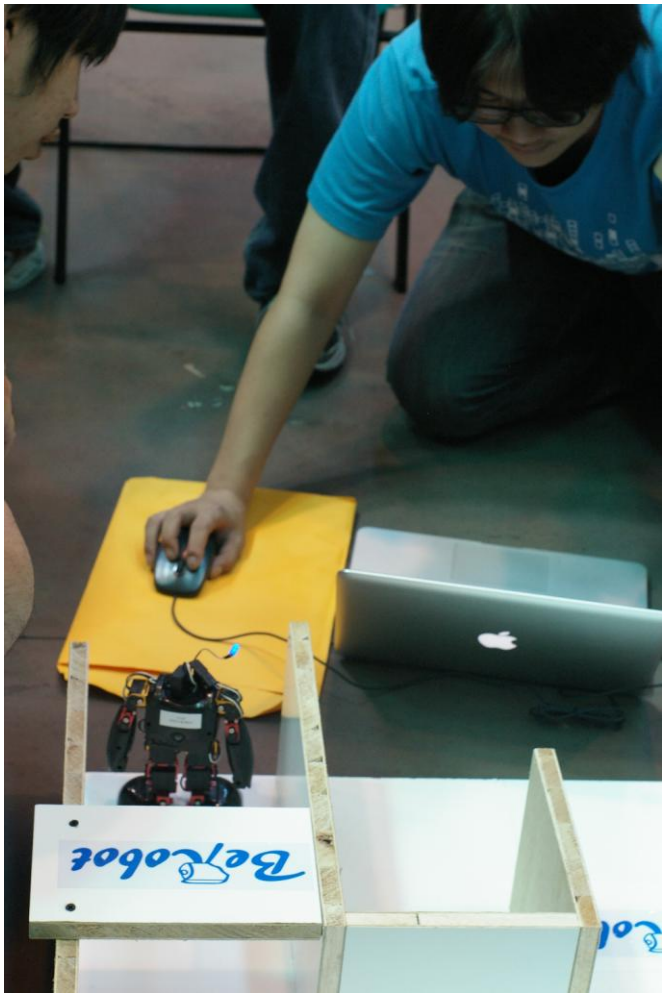
指導教授：　孔崇旭

學生：　洪　亮
郭承諺

動機 - 機器人控制軟體

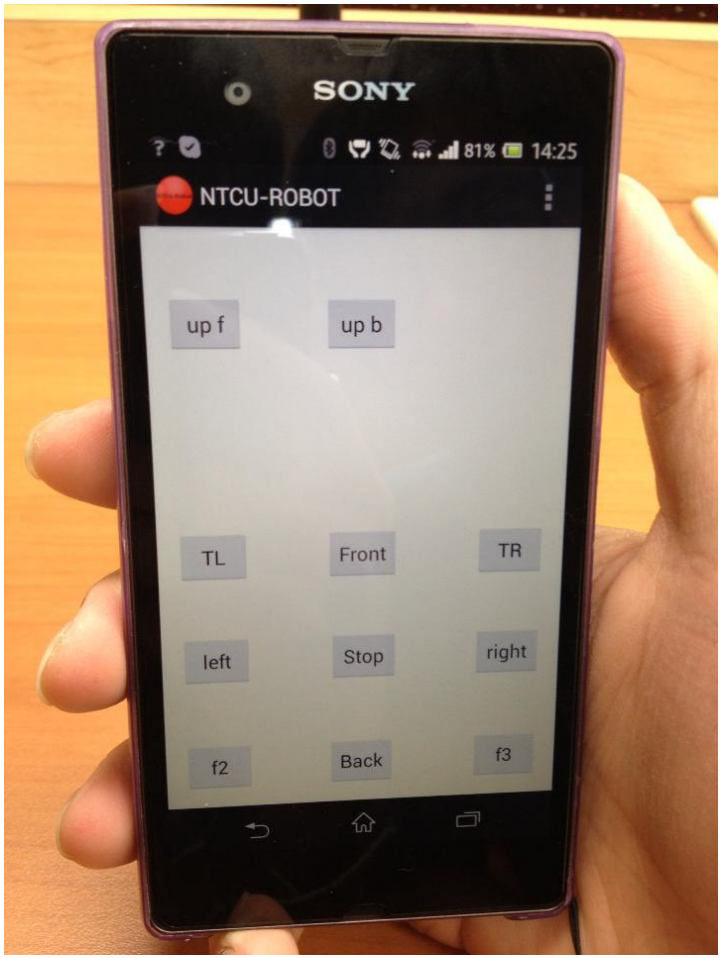
NTCU-Robot第一代控制（PC＋Bluetooth）





動機 - 機器人控制軟體

NTCU-Robot第二代（Smart Phone＋Bluetooth）



動機
（NTCU-Robot第三代？）

專題簡介

專題簡介

= ?

KINECT for XBOX 360.

專題簡介

- Kinect是Microsoft一款具有人體骨架追蹤以及可取得深度影像的設備
  - 在控制上可以更加直覺化
- 但Kinect內建由『機器學習』訓練的動作只有幾種，
  - 不符合我們的需求
- 解決方法
  - 訓練自行定義動作，由程式判斷後發送控制指令給機器人
  - 採用『類神經網路』演算法訓練動作
  - 目的在於可以由程式計算出目前最接近動作，並經由無線傳輸的方式發送指令給予機器人並進行控制

專題目標

- 控制更加直覺 → Natural UI → Kinect
- 提高識別率 → 模糊邏輯 → Neural Network
- 安全控管 → 生物特徵 → 人臉特徵識別
- 增加移動空間 → 新增水平軸向 → Servo

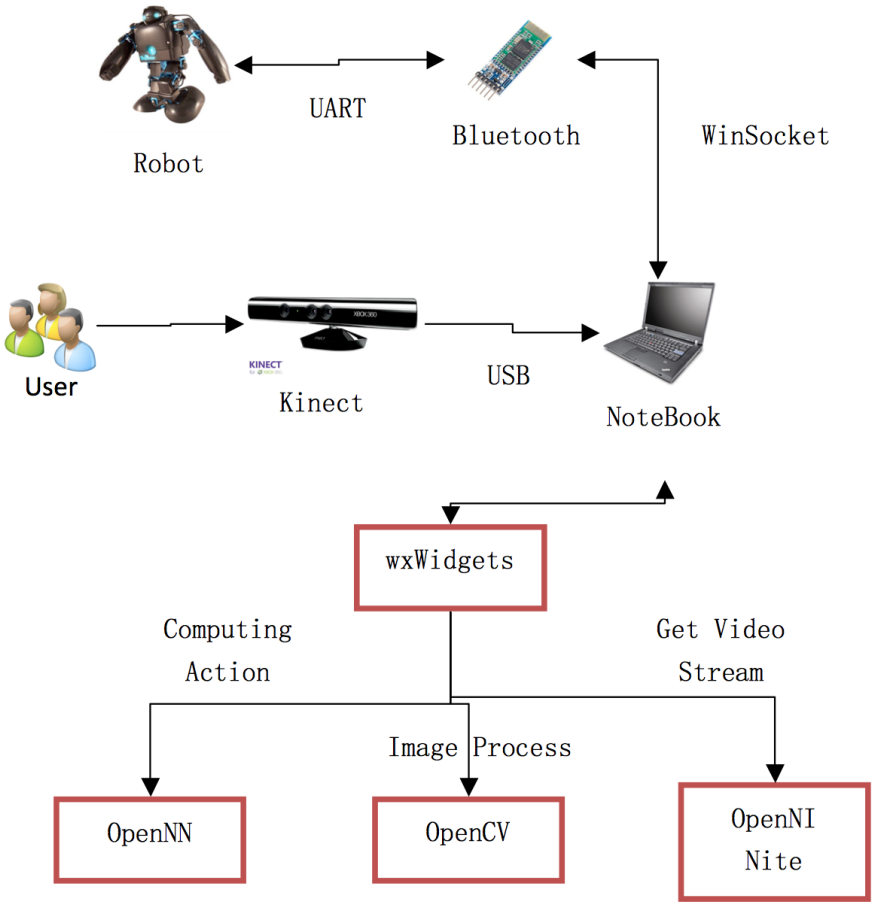
主要技術 & 相關領域

- 影像處理
- 線性代數
- 微積分
- 訊號處理
- 藍芽通訊
- 數位電路設計
- 類神經網路
- 機器學習

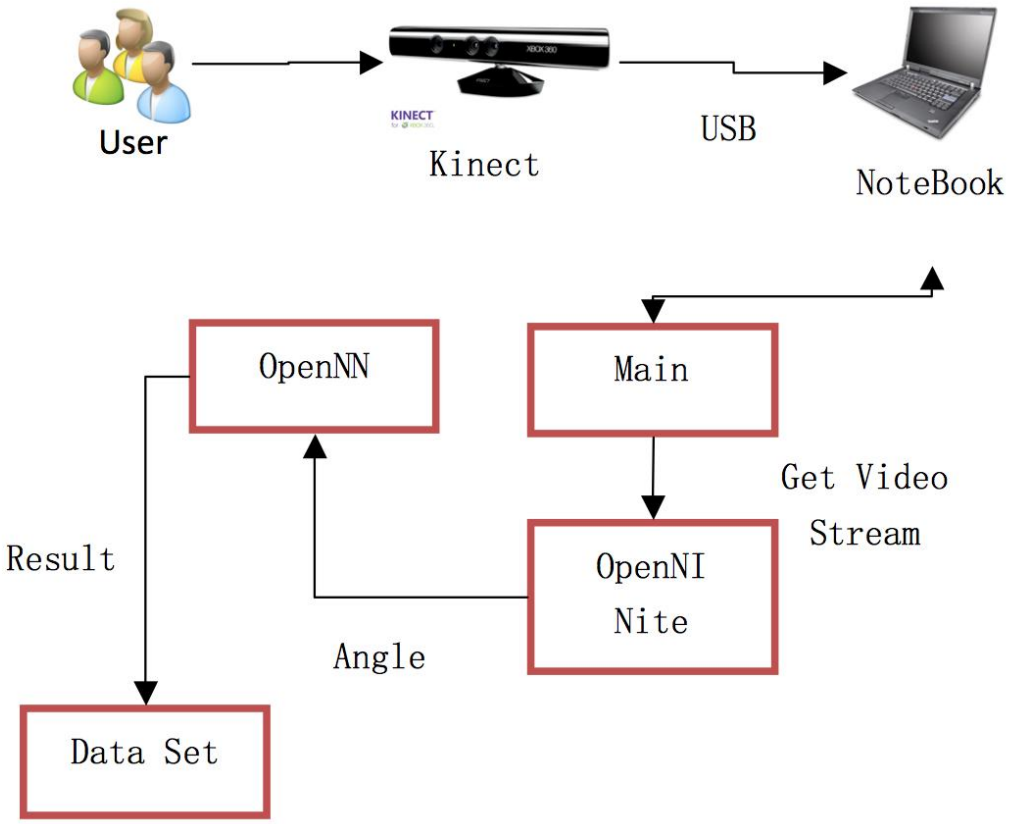
使用Library

- wxWidgets 2.8.12
- OpenCV 2.4.6
- OpenNI2 & NiTe2
- OpenNN 0.9

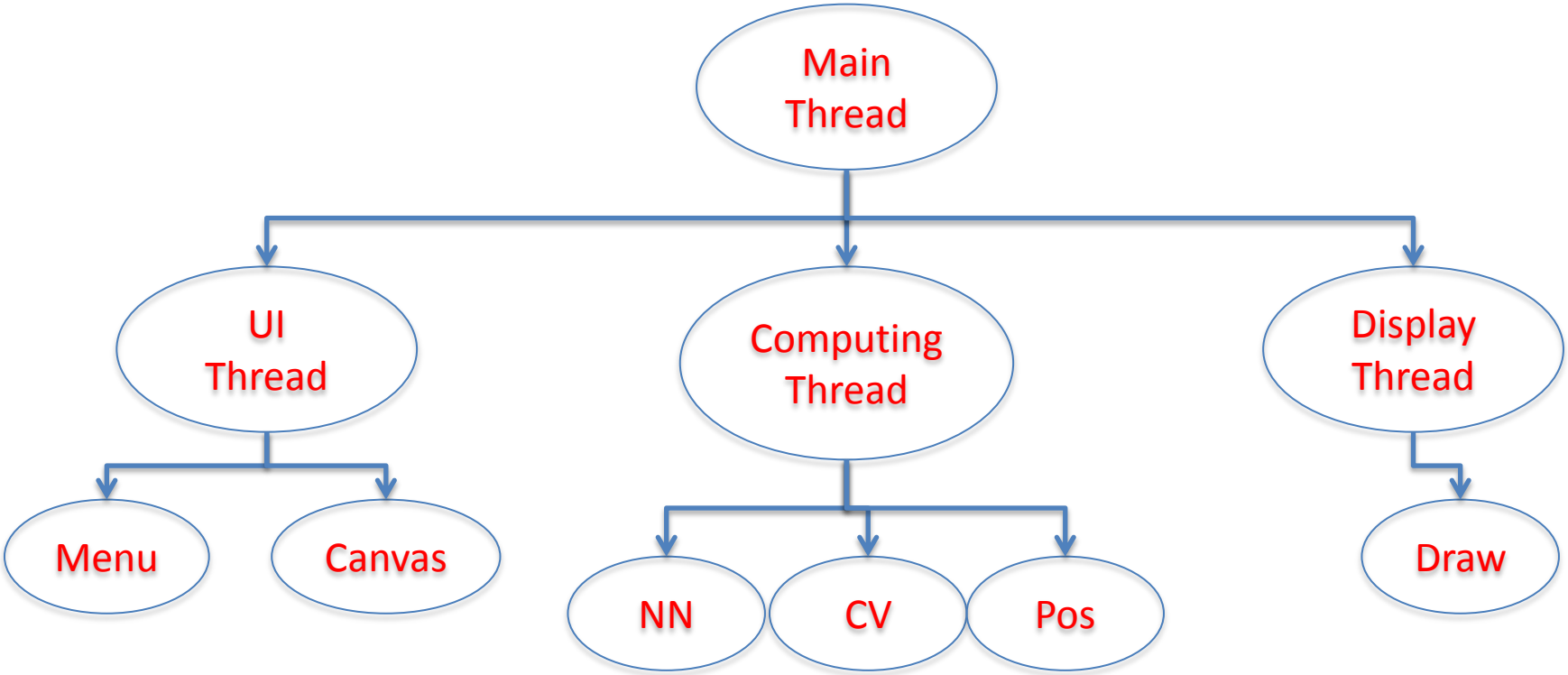
系統架構


Robot
UART
Bluetooth
WinSocket
User
Kinect
USB
NoteBook
wxWidgets
Computing
Action
Get Video
Stream
Image Process
OpenNN
OpenCV
OpenNI
Nite

系統架構


User
Kinect
USB
NoteBook
OpenNN
Main
Get Video
Stream
Result
OpenNI
Nite
Angle
Data Set

程式架構


Main Thread
UI Thread
Computing Thread
Display Thread
Menu
Canvas
NN
CV
Pos
Draw

LIVE DEMO

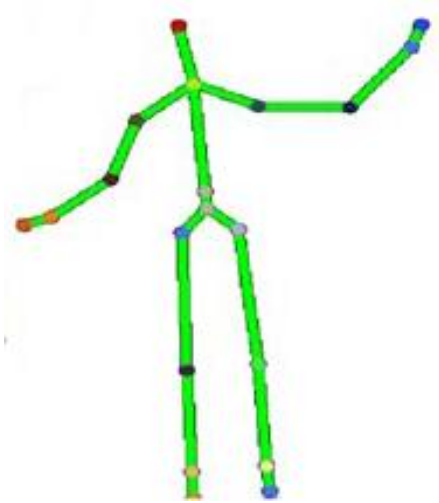
系統設計

骨架取得

骨架取得

- 藉由Kinect可以取得2D影像及Depth Map
- 取出2D影像中的各骨架點的二維平面座標
- 將平面座標與深度值結合成三維空間座標
- 將座標乘上World 、View 、Proj轉換至程式

- 但角度需要自行計算



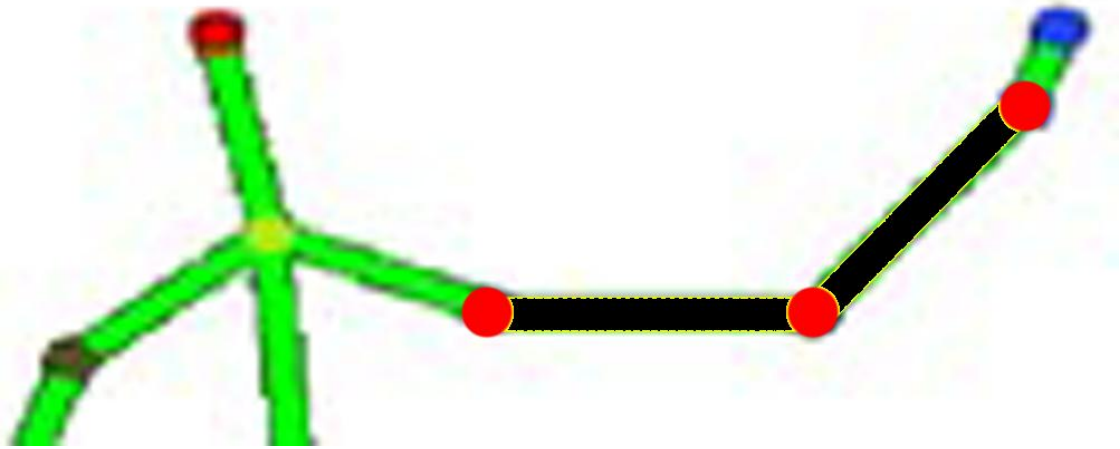
角度取得

角度取得

- 使用空間座標製作出向量後計算夾角
- 以向量夾角公式 $\cos q = \frac{\overrightarrow{AB} \cdot \overrightarrow{AC}}{\left|\overrightarrow{AB}\right| \cdot \left|\overrightarrow{AC}\right|}$ 計算角度
- 求出 cosθ後再使用反三角函數得出角度

角度取得－手肘（一）

- 取 *手肘-手腕* 與 *手肘-肩膀* 兩向量
- 角度範圍為0~180度
  - 不同動作可能有相同結果



角度取得－手肘（二）

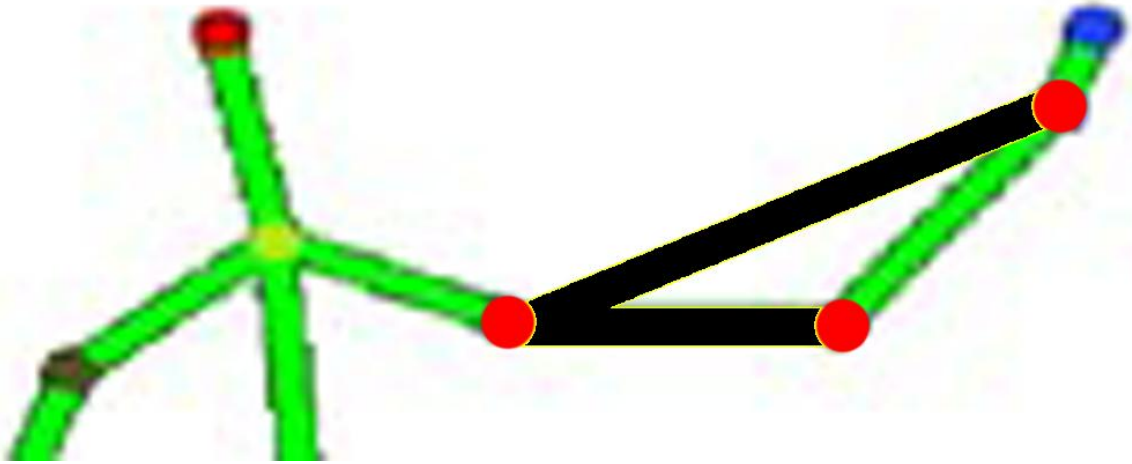
- 將 *手肘* 座標點投影至 *手肘-肩膀* 向量上

- 計算 *投影點-手肘* 此向量的斜率
  - 斜率大於0 → 夾角 = θ
  - 斜率小於0 → 夾角 = 360 – θ

- 使角度範圍擴增至0~360

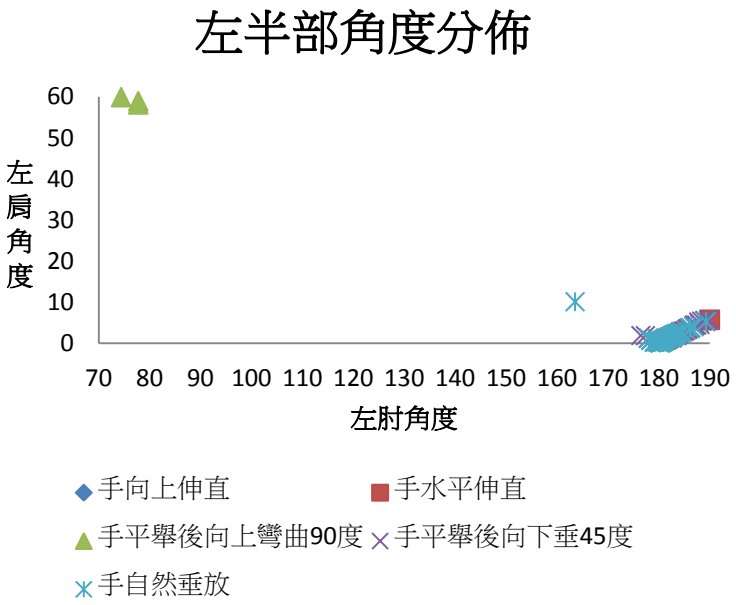
角度取得－肩膀（一）

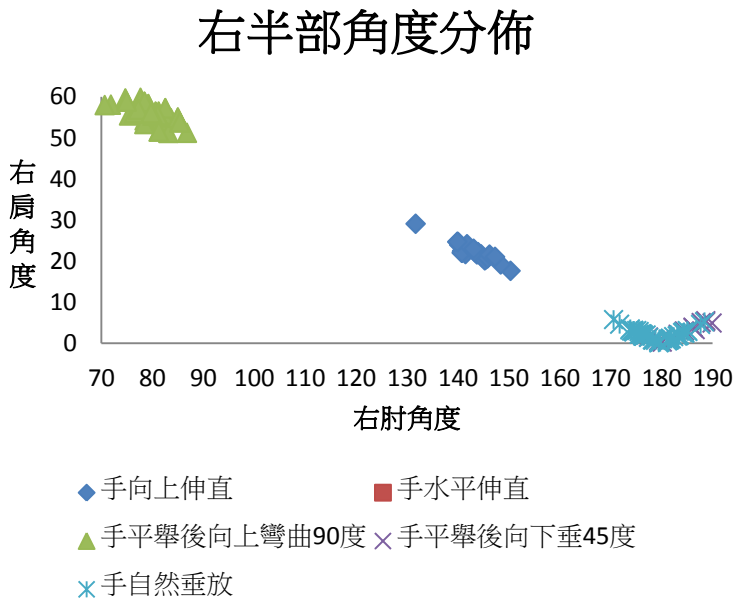
- 取 *肩膀-手肘* 與 *肩膀-手腕* 兩向量



角度取得－肩膀（方法一）

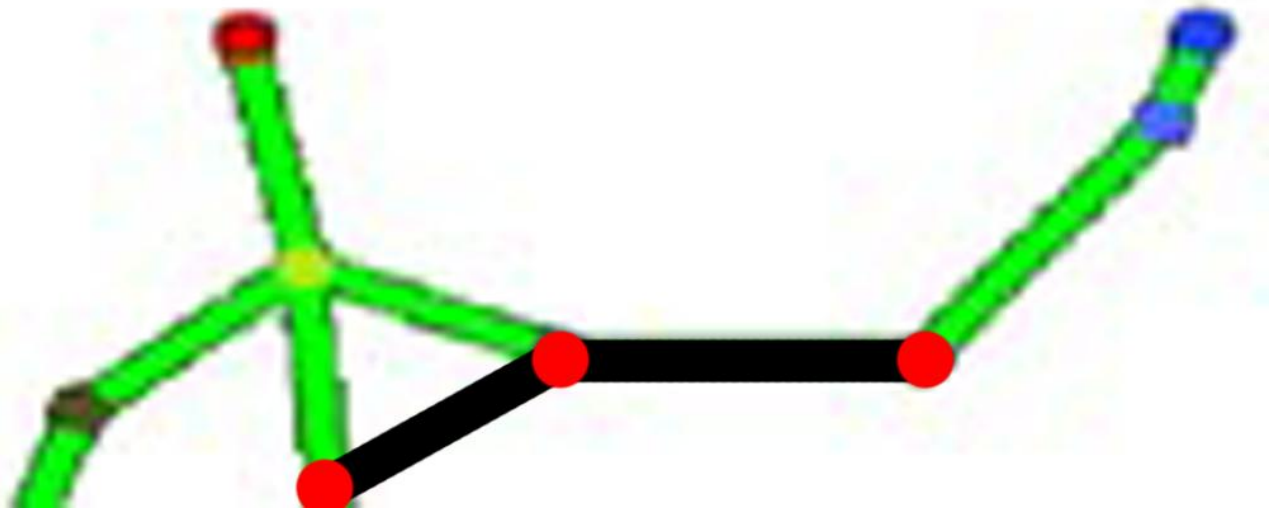
- 無意義的角度值
  - 導致訓練出的類神經網路辨識度差





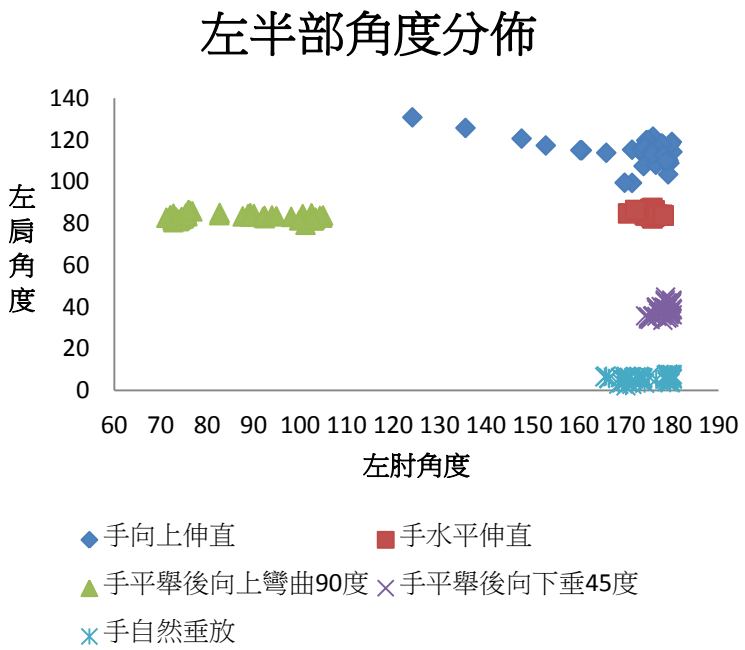
角度取得－肩膀（方法二）

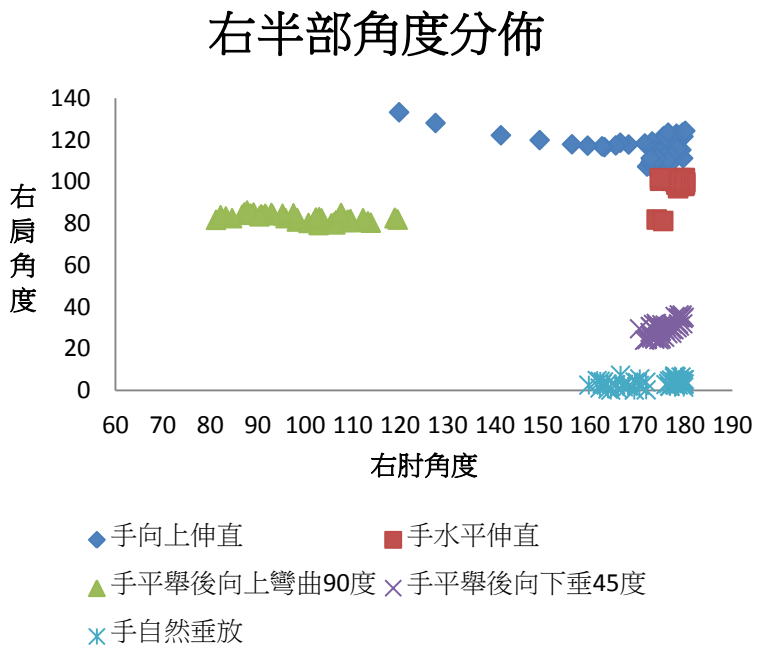
- 取 *肩膀-手肘* 與 *肩膀-軀幹* 兩向量
  - 將結果減去45度使向下垂時為0度



角度取得－肩膀（方法二）

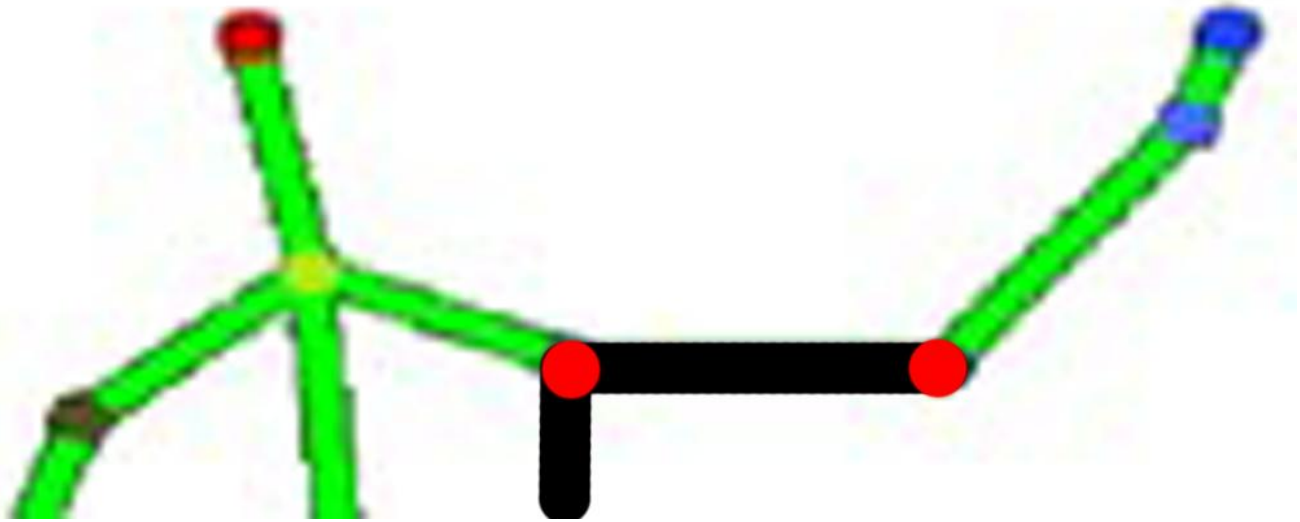
- 角度範圍為 -45~135度
  - 部分動作角度可能重疊
- 使辨識度大幅提升





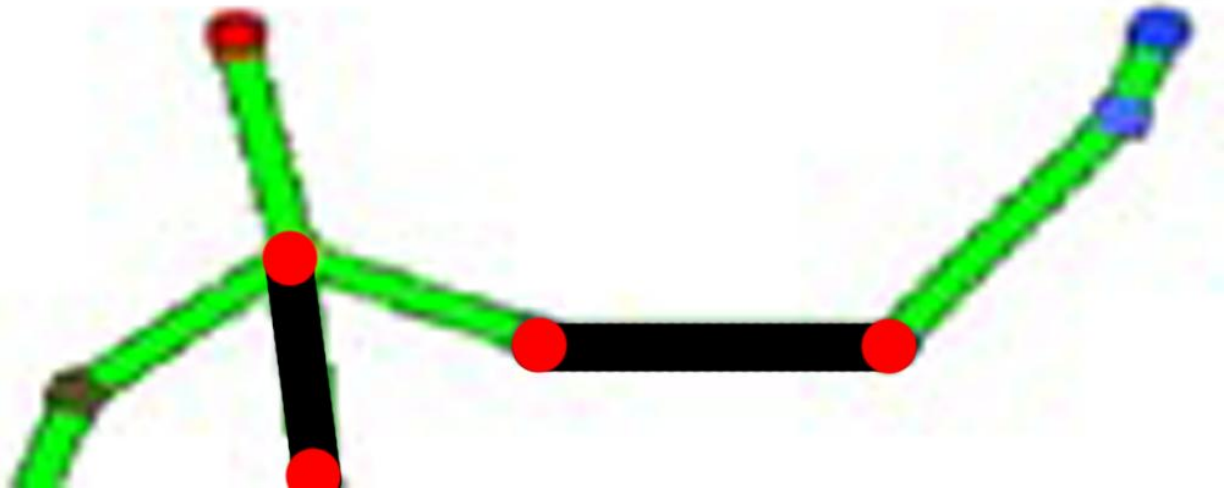
角度取得－肩膀（方法三）

- 取 *肩膀-手肘* 與 *向下垂直* 兩向量
  - 使角度範圍擴充為0~180度



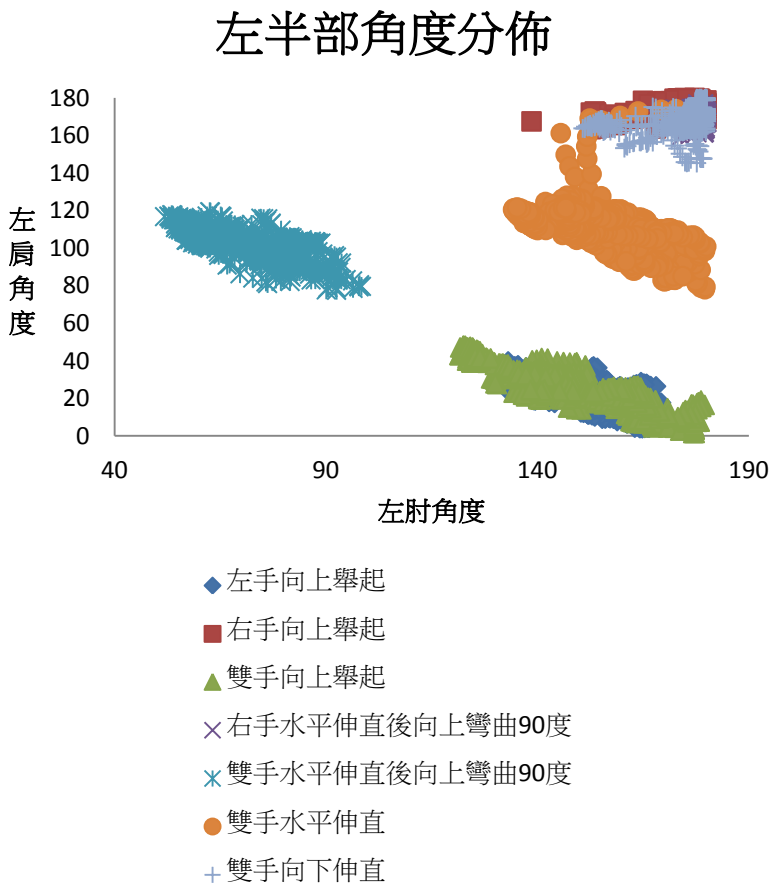
角度取得－肩膀（方法四）

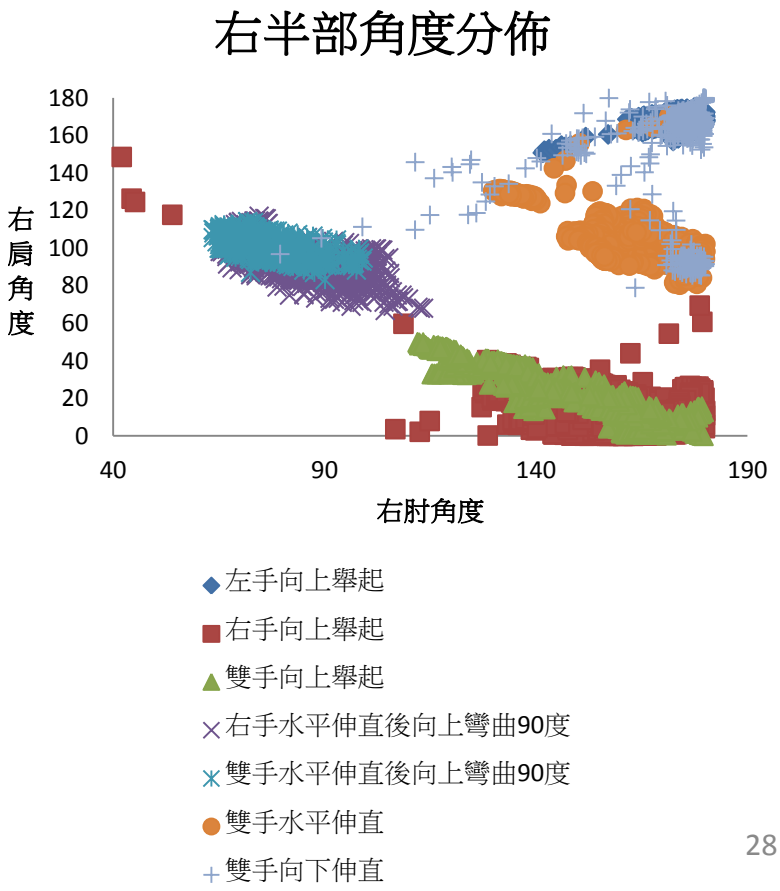
- 取 *肩膀-手肘* 與 *脖子-軀幹* 兩向量



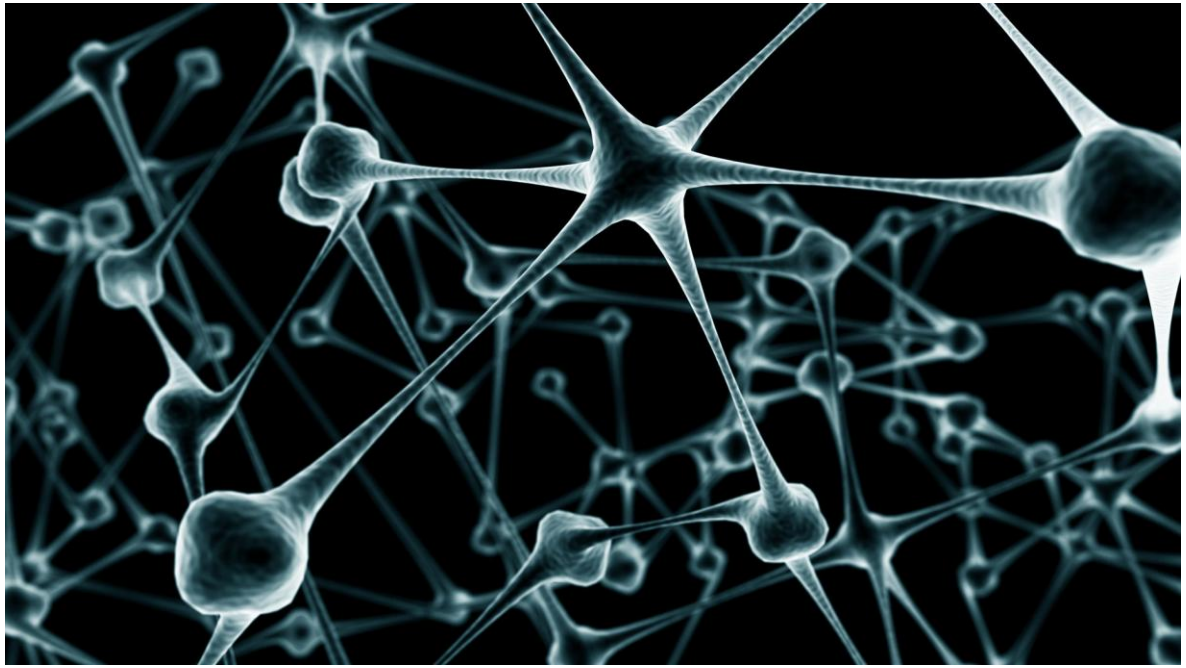
角度取得－肩膀（方法四）

- 身體傾斜時，不影響角度計算結果



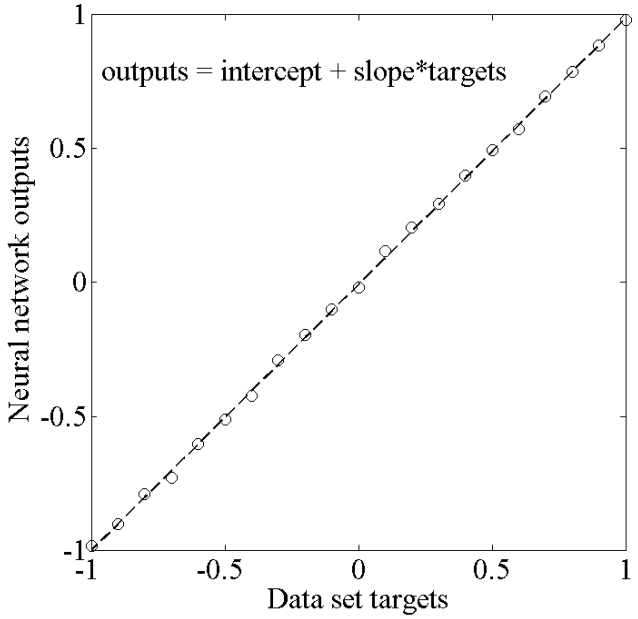


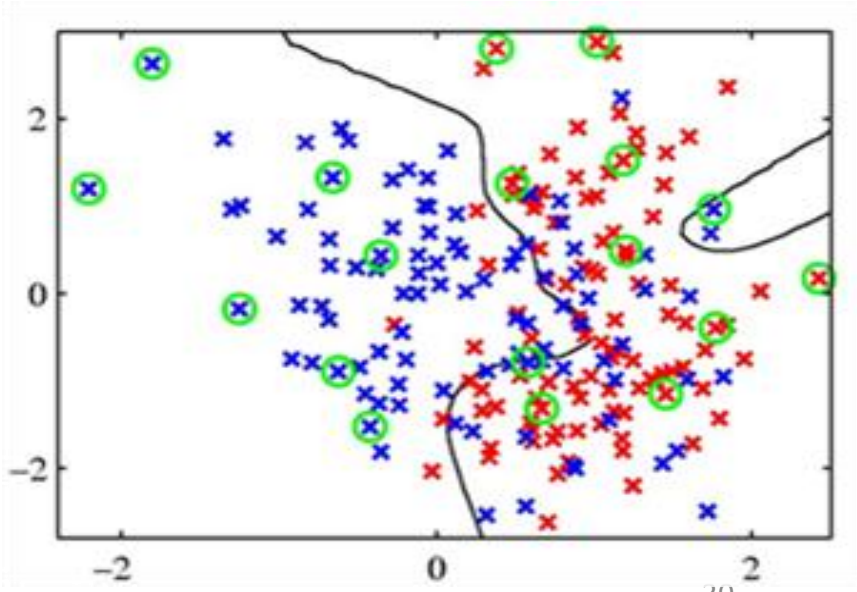
類神經網路



類神經網路

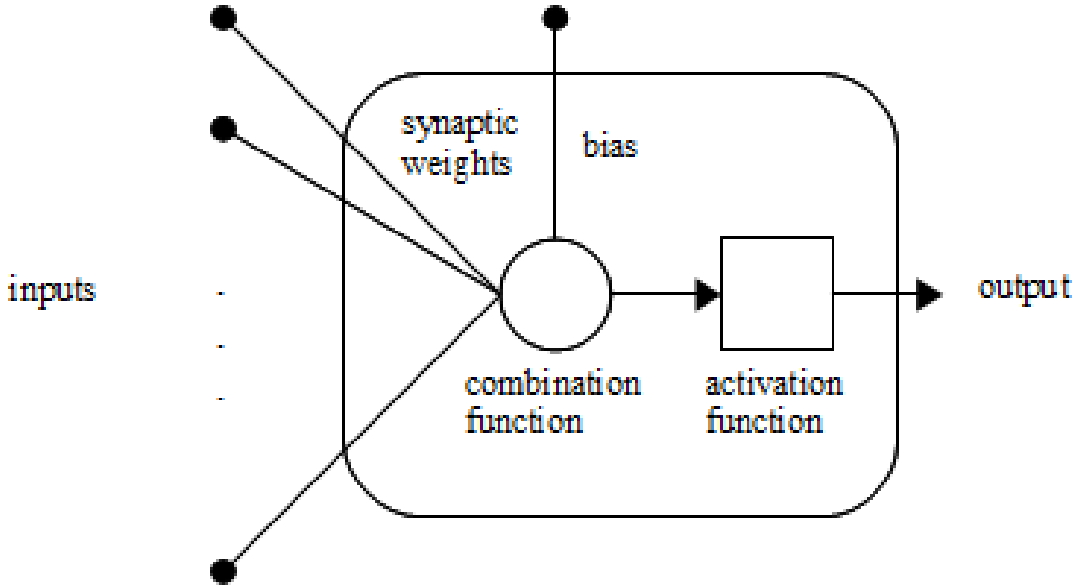
- 一種模擬人類神經系統的運算模型
- Feed-Forward Architecture
- 可用於建立非線性統計模型
- 常用於尋找輸入與輸出間的關係





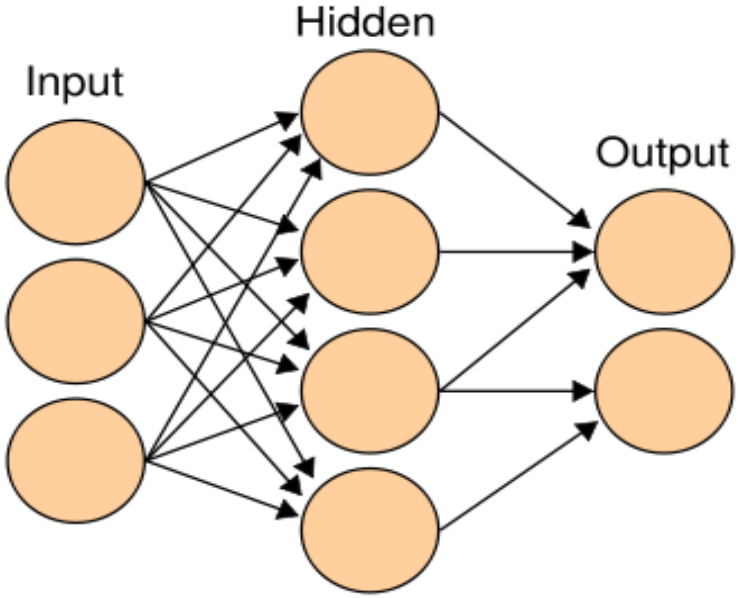
類神經網路構造

- 每個神經元包含
  - 輸入(Input)
  - 輸出(Output)
  - 權重(Weight)
  - 偏移值(Bias)



類神經網路構造

- 一到數個神經元組成一層(Layer)
- 一個類神經網路包含
  - 輸入層(Input layer)
  - 輸出層(Output layer)
  - 隱藏層(Hidden layer)



類神經網路訓練

訓練方式

- 目標需求為分類器
- 以事先製作的資料集(Dataset)訓練
- 使用倒傳遞演算法(back propagation)

倒傳遞演算法

- 為一種監督學習的演算法
- 包含兩大步驟

1. 傳遞
   (1) 正向傳遞，取得輸出結果
   (2) 反向傳遞，求隱藏層與輸出層的誤差

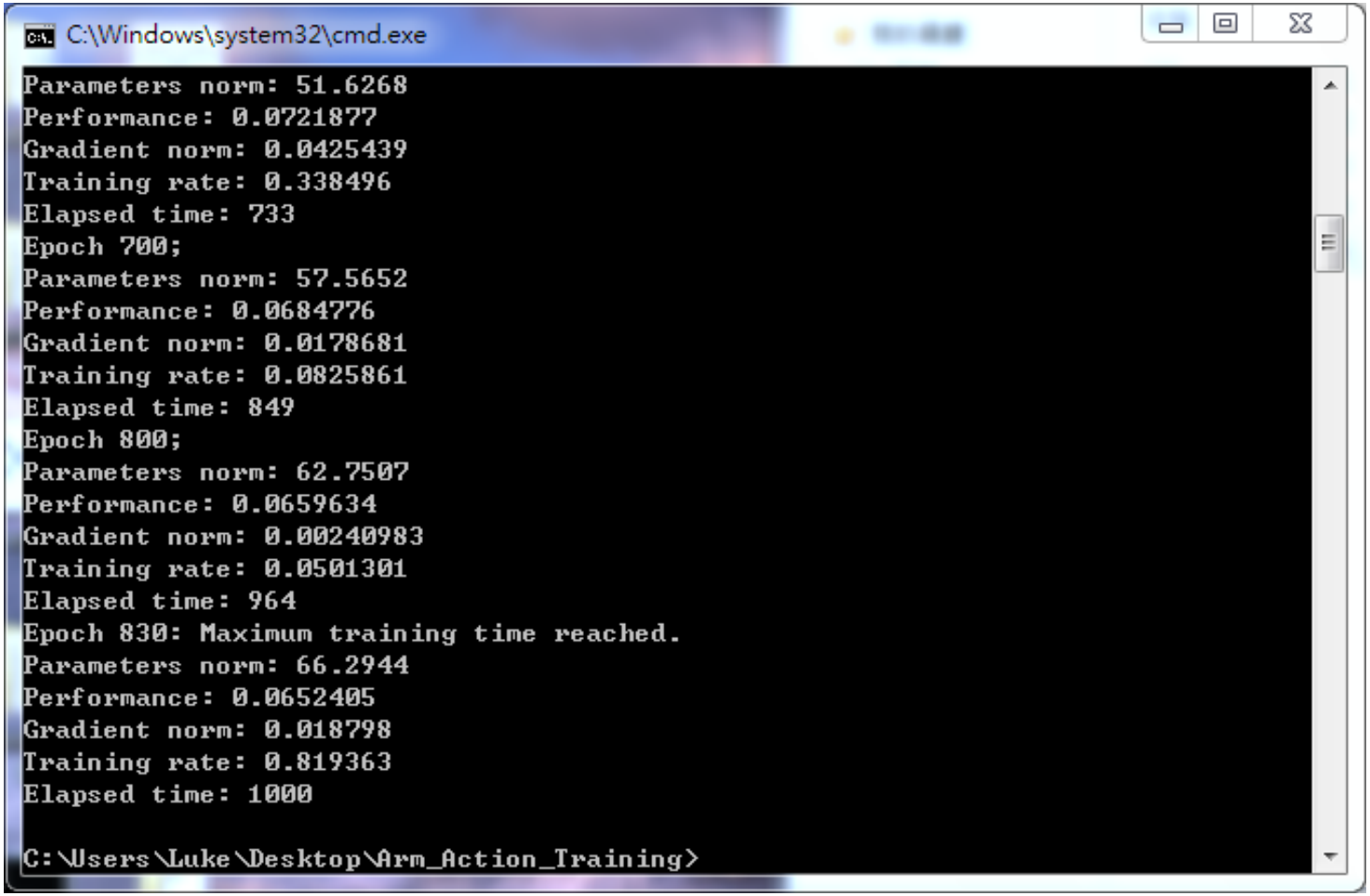
2. 更新權重
   (1) 將輸入與誤差相乘，得到權重的斜率
   (2) 將權重減去該斜率乘上訓練率*的結果

資料集

- 每一行為一項數據
- 每項數據包含輸入值與目標輸出
- 訓練程式會進行過濾
  - 去除大於兩個標準差的資料
- 我們使用約兩萬筆資料進行訓練

```
272.907 144.033 298.227 116.460 1 0 0 0 0 0 0 0 0
112.361 80.7093 220.419 168.197 0 1 0 0 0 0 0 0 0
224.672 158.212 82.2939 74.1352 0 0 1 0 0 0 0 0 0
105.125 78.2888 102.138 80.1228 0 0 0 1 0 0 0 0 0
301.034 124.344 323.647 118.035 0 0 0 0 1 0 0 0 0
116.913 98.0988 198.178 112.131 0 0 0 0 0 1 0 0 0
145.077 59.0568 155.464 84.2795 0 0 0 0 0 0 1 0 0
74.8657 78.5338 303.313 114.403 0 0 0 0 0 0 0 1 0
307.948 115.011 59.3113 97.4398 0 0 0 0 0 0 0 0 1
```

訓練程式


C:\Windows\system32\cmd.exe
Parameters norm: 51.6268
Performance: 0.0721877
Gradient norm: 0.0425439
Training rate: 0.338496
Elapsed time: 733
Epoch 700;
Parameters norm: 57.5652
Performance: 0.0684776
Gradient norm: 0.0178681
Training rate: 0.0825861
Elapsed time: 849
Epoch 800;
Parameters norm: 62.7507
Performance: 0.0659634
Gradient norm: 0.00240983
Training rate: 0.0501301
Elapsed time: 964
Epoch 830: Maximum training time reached.
Parameters norm: 66.2944
Performance: 0.0652405
Gradient norm: 0.018798
Training rate: 0.819363
Elapsed time: 1000
C:\Users\Luke\Desktop\Arm_Action_Training>

訓練結果

訓練資料

神經元個數 / 動作	15 個	20 個	30 個
雙手向下伸直	93.05%	93.25%	93.2%
左手向上彎曲	61.9%	49.5%	39.7%
右手向上彎曲	60%	81.55%	40.5%
雙手向上彎曲	99.25%	99.35%	99.47%
雙手向下彎曲	94%	95.95%	95.75%
雙手水平伸直	95.6%	98.4%	97.1%
雙手向上伸直	96.45%	95.55%	95.45%
左手向上彎曲 右手向下彎曲	99.56%	99.73%	99.84%
左手向下彎曲 右手向上彎曲	99.05%	99.85%	99.92%

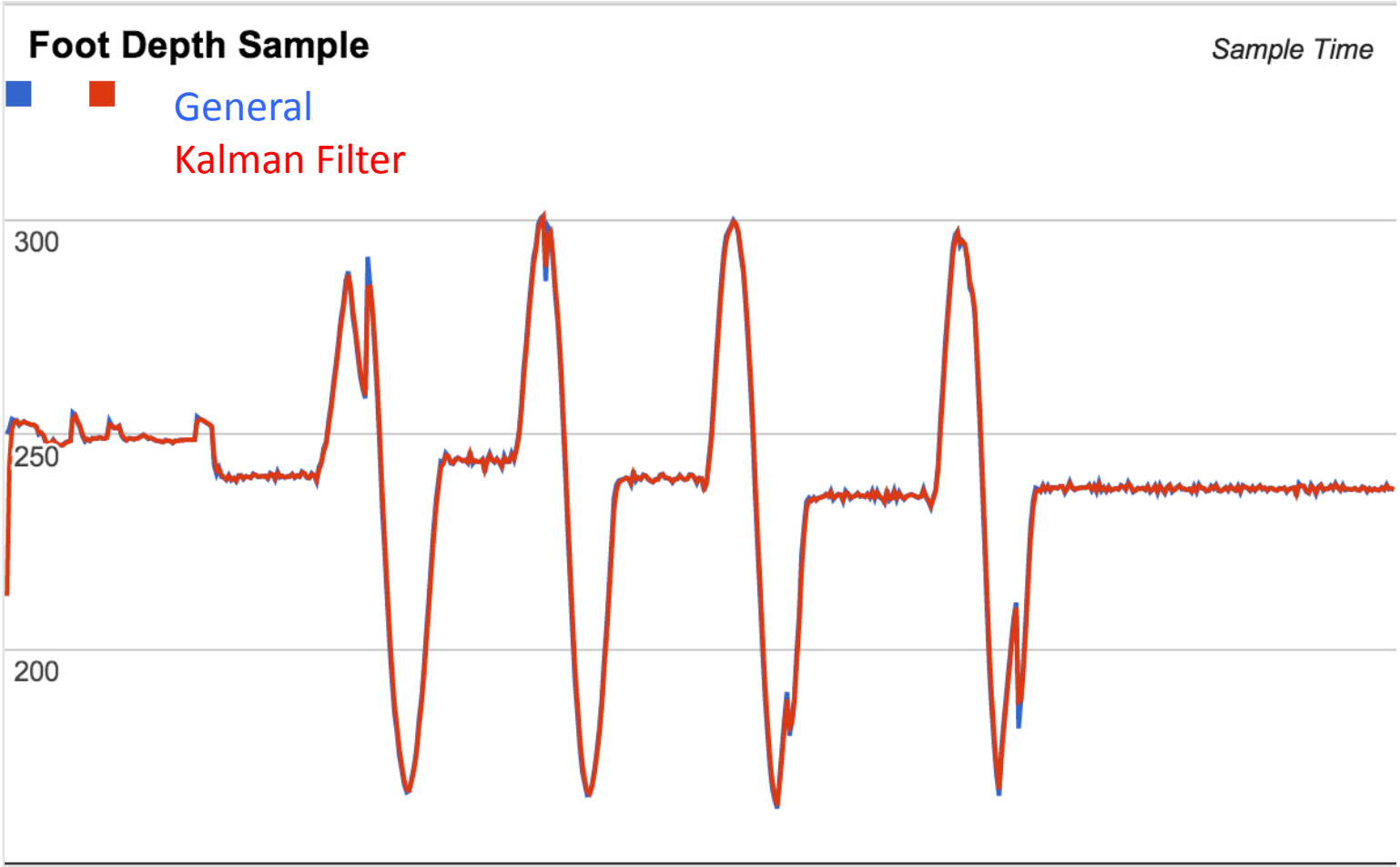
測試資料

神經元個數 / 動作	15 個	20 個	30 個
雙手向下伸直	93.2%	87.6%	88.8%
左手向上彎曲	44.4%	27.2%	17%
右手向上彎曲	44.2%	78.4%	71.2%
雙手向上彎曲	100%	100%	100%
雙手向下彎曲	87.6%	92.6%	93.4%
雙手水平伸直	99.8%	97%	99.8%
雙手向上伸直	74.6%	74%	74%
左手向上彎曲 右手向下彎曲	100%	100%	100%
左手向下彎曲 右手向上彎曲	99%	99.6%	99.6%

判斷踢球

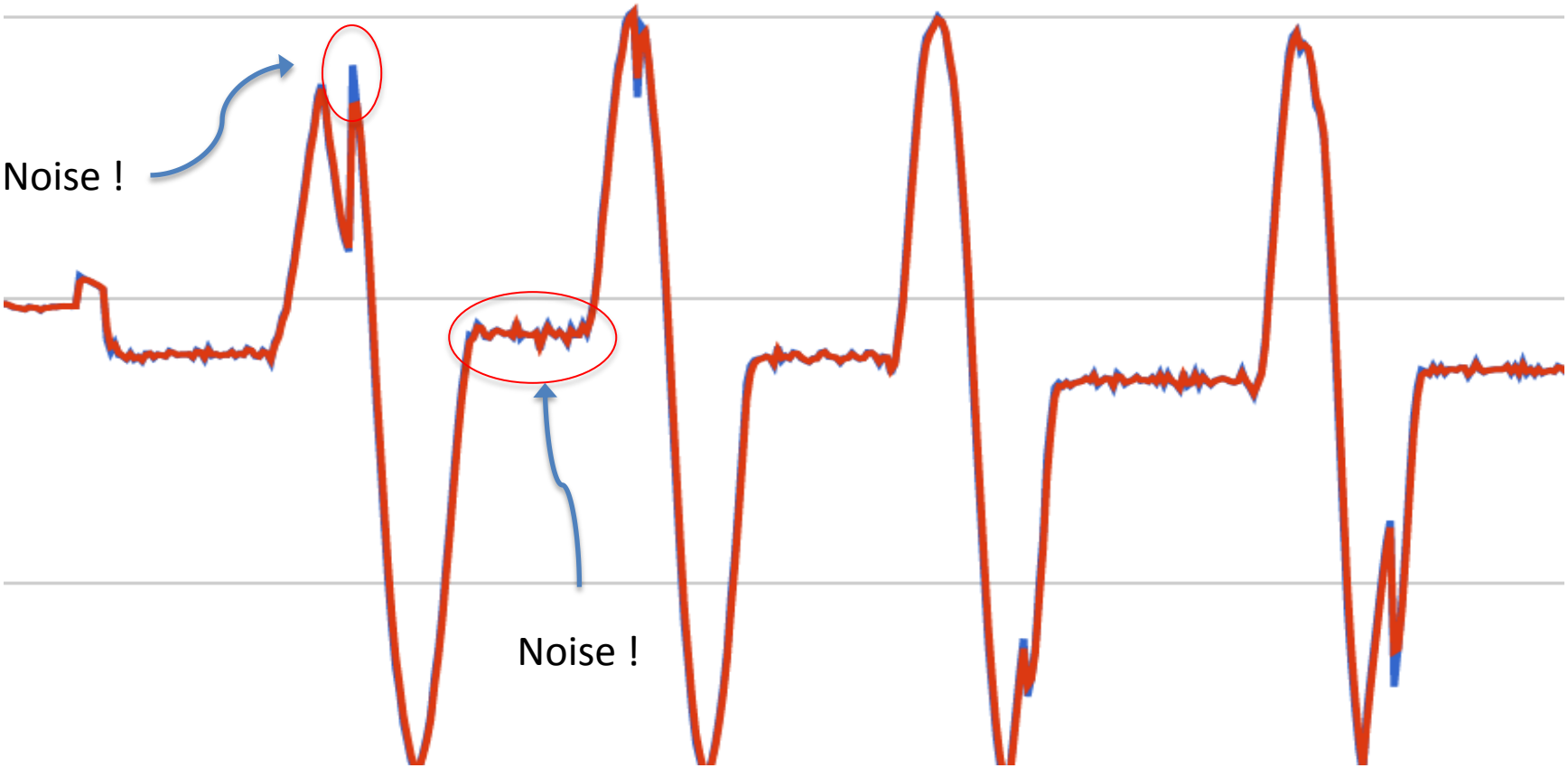
判斷踢球

- 藍色為原始波形
- 紅色為使用卡爾曼濾波



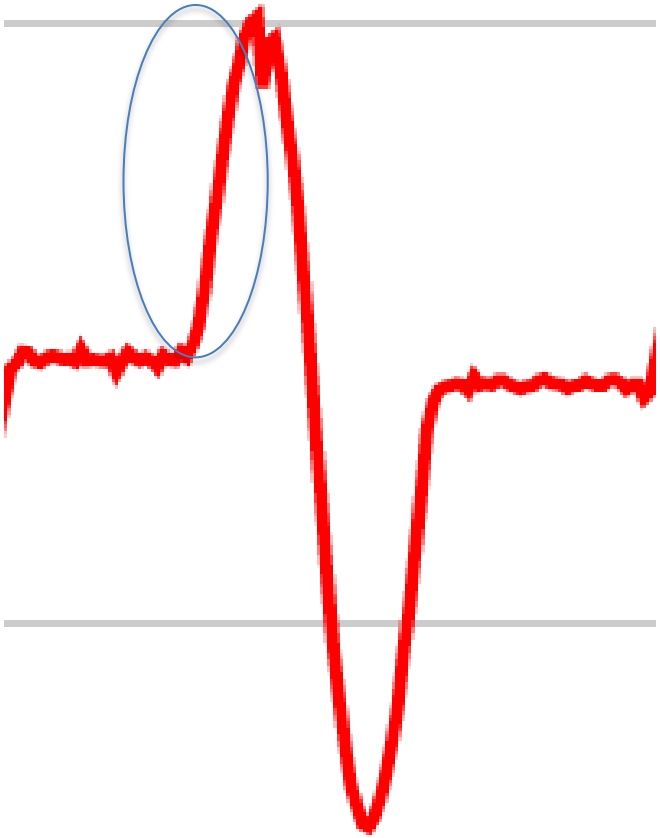
判斷踢球

- 卡爾曼濾波可消除藍色Noise



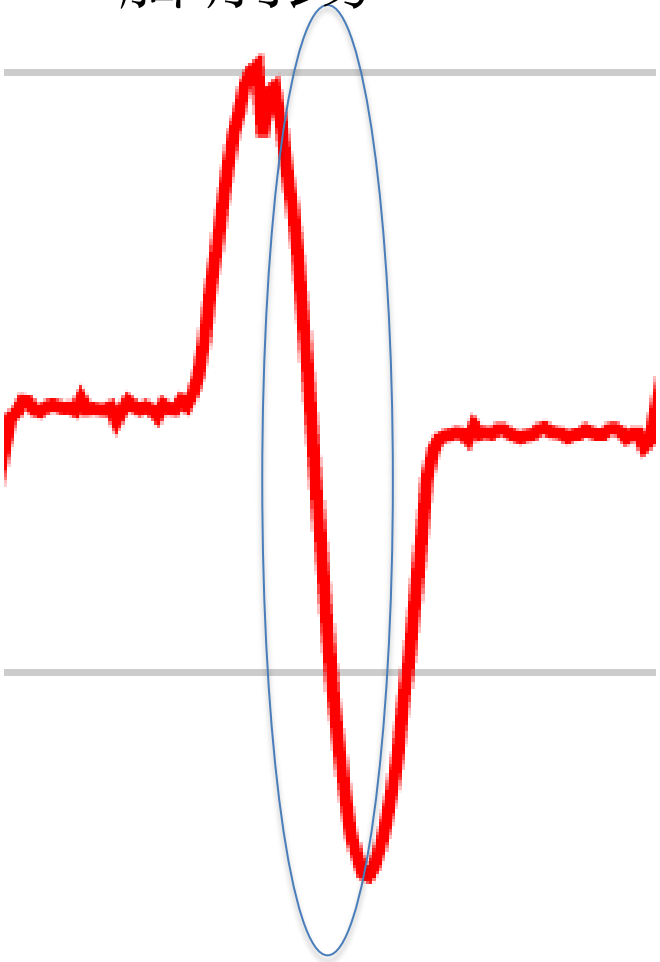
判斷踢球

- 腳後抬 +L



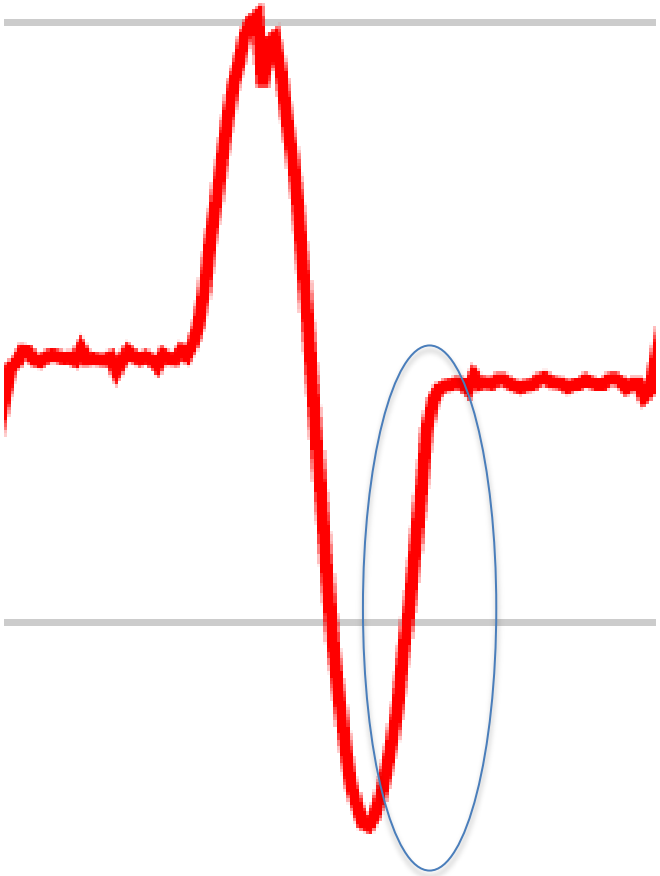
判斷踢球

- 腳前踢 -2L



判斷踢球

- 腳回到原點 +L



判斷踢球

- L + (-2L) + L

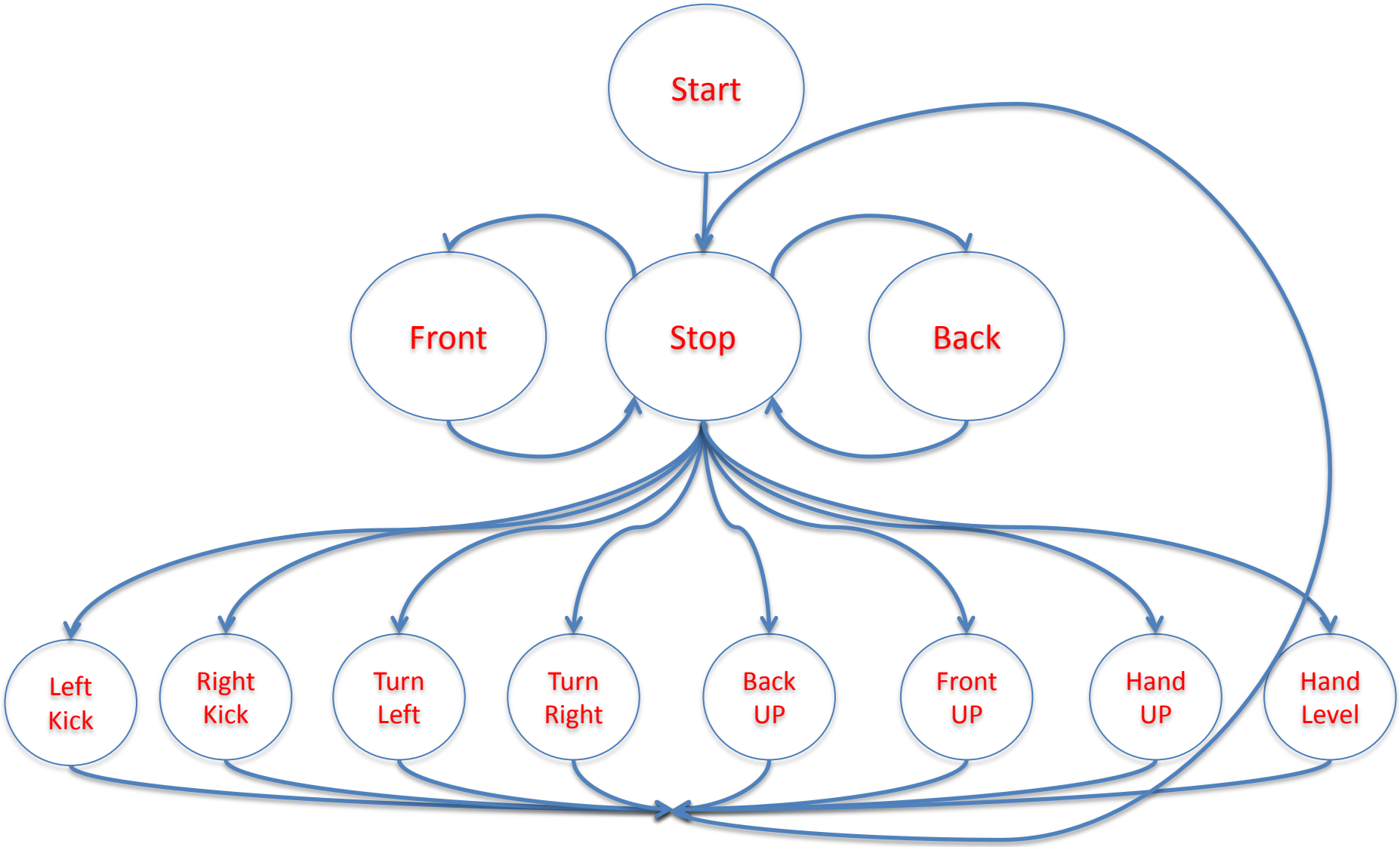
```
Different = n – (n - 1);

If ((Different > min && Different < max) &&
(Integral_plus > threshold_min && Integral_minus < threshold_max)){
   State = Kick;
   Integral_plus = Integral_minus = 0;
}

else if ((Different > min && Different < max) ){
   Integral_plus += Different;
}

else if ((Different > -max && Different < -min) ){
   Integral_minus += Different;
}
```

有限狀態機

狀態列表

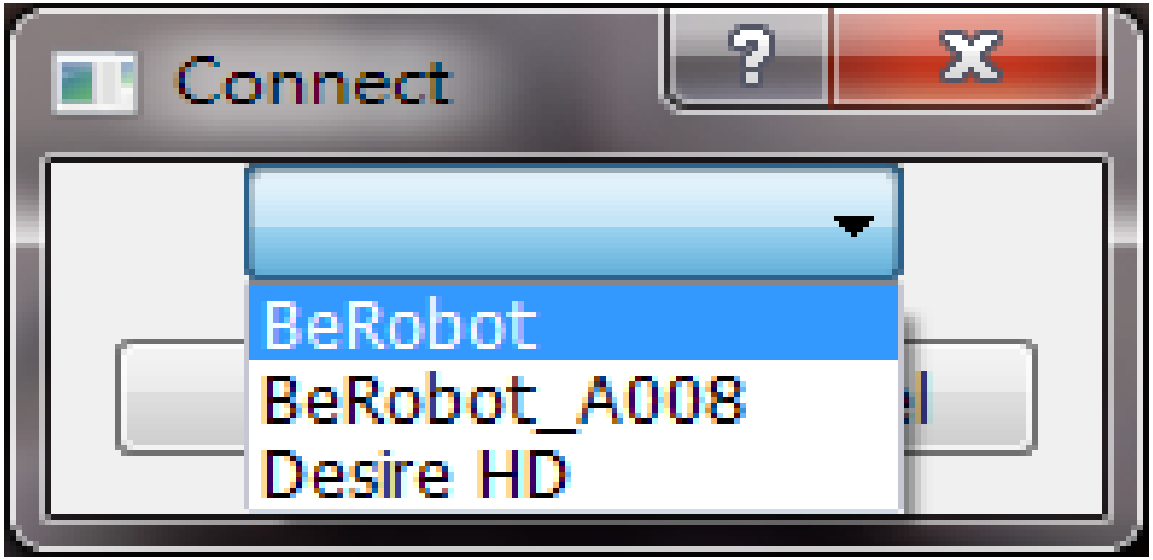
No.	ACTION NAME	手勢及姿態
	立正(準備動作)	
1	Stop	立正
2	Turn left	90 度舉左手
3	Tuen right	90 度舉右手
4	Front	原點深度負 35 公分
5	Back	原點深度正 35 公分
6	Back UP	雙手水平+ 90 度雙手向下舉
7	Front up	雙手水平+ 90 度雙手向上舉
8	舉手	
	Hand UP	雙手打直上舉
	Hand Level	雙手水平
9	深度原點初始化	左手向上 90+ 右手向下 90
10	保留	右手向上 90+ 左手向下 90
11	左踢	達到積分波型
12	右踢	達到積分波型

有限狀態機


Start
Front
Stop
Back
Left Kick
Right Kick
Turn Left
Turn Right
Back UP
Front UP
Hand UP
Hand Level

連結機器人

連接機器人



- 搜尋藍芽裝置

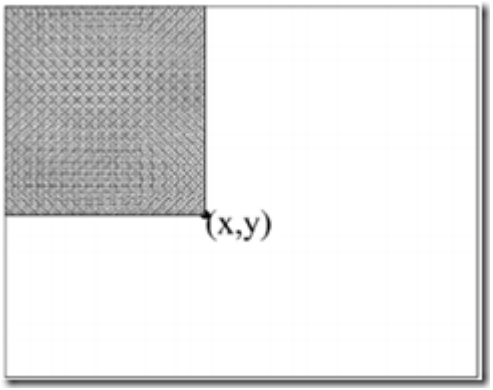
連接機器人

- BluetoothFindFirstDevice
- BluetoothFindNextDevice
- Socket
- Connect
- Send
- Closesocket
- WSACleanup

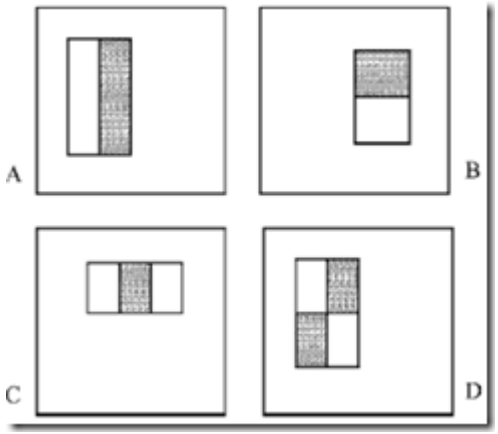
臉部身分辨識

臉部身份辨識

Integral Image



Rectangle Feature



臉部身份辨識

- Machine Learning
- Adaptive Boosting

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
  2. Select the best weak classifier with respect to the weighted error

     $$\epsilon_t = \min_{f,p,\theta} \sum_i w_i \left| h(x_i, f, p, \theta) - y_i \right|.$$

     See Section 3.1 for a discussion of an efficient implementation.
  3. Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t, and θ_t are the minimizers of ϵ_t.
  4. Update the weights:

     $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

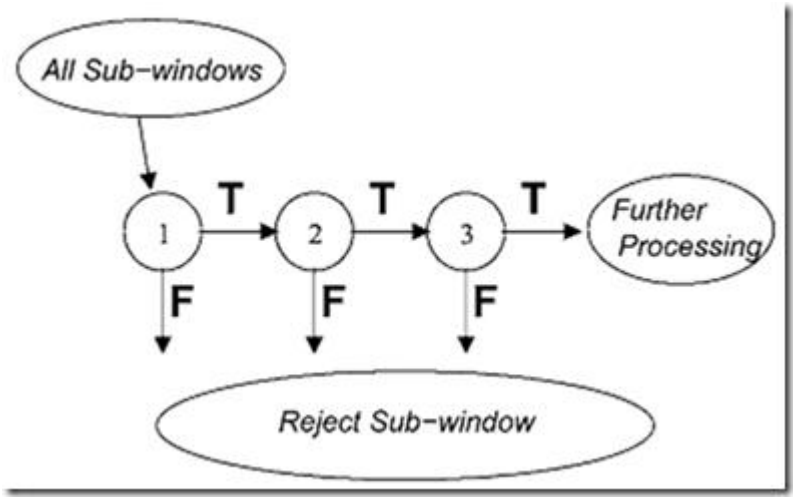
     where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

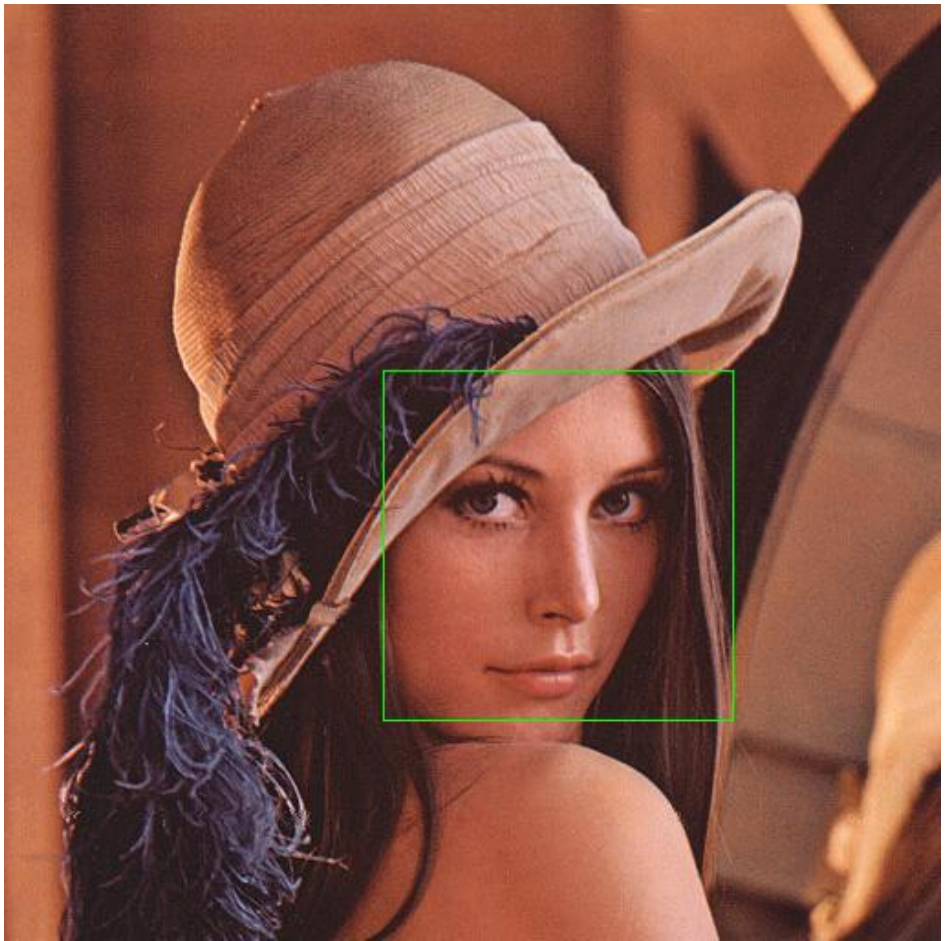
臉部身份辨識

- Cascade Classifier
- False Positive Rate



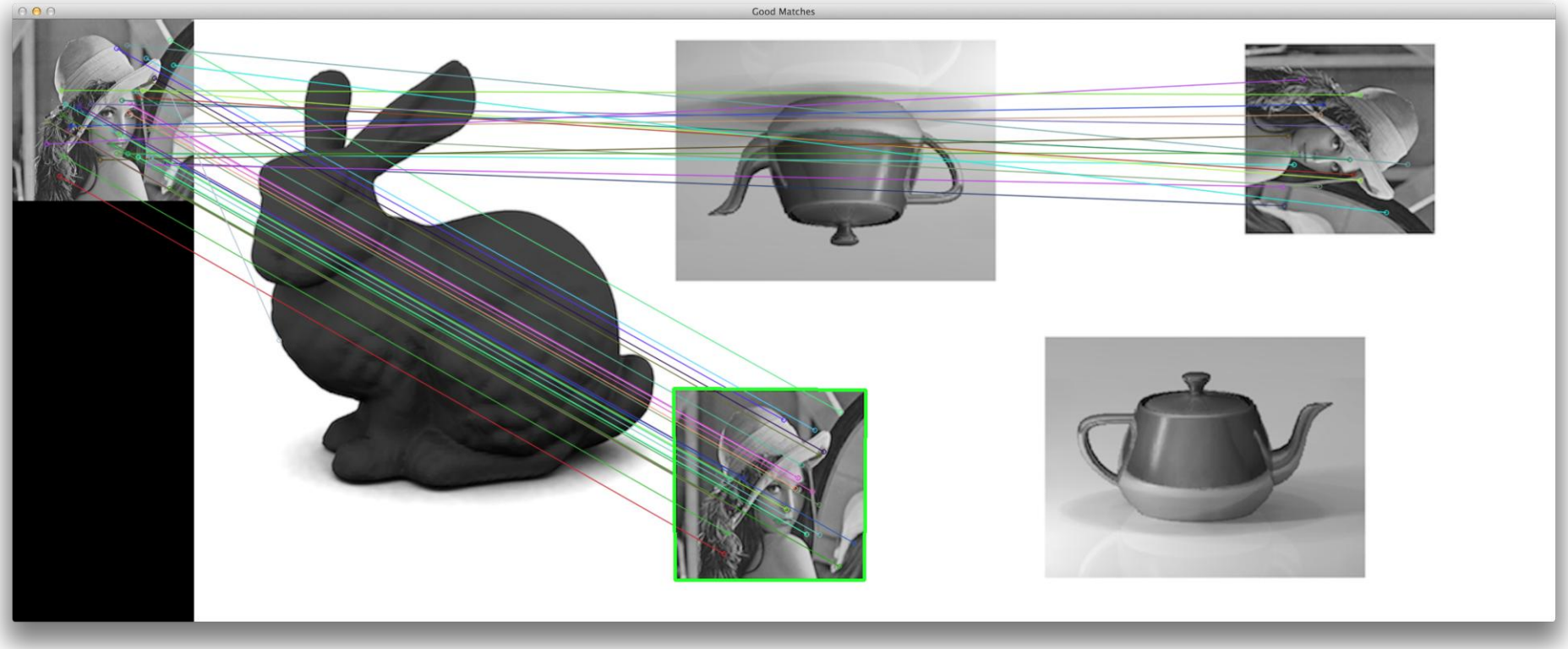
臉部身份辨識

- ROI（Region of Interest）



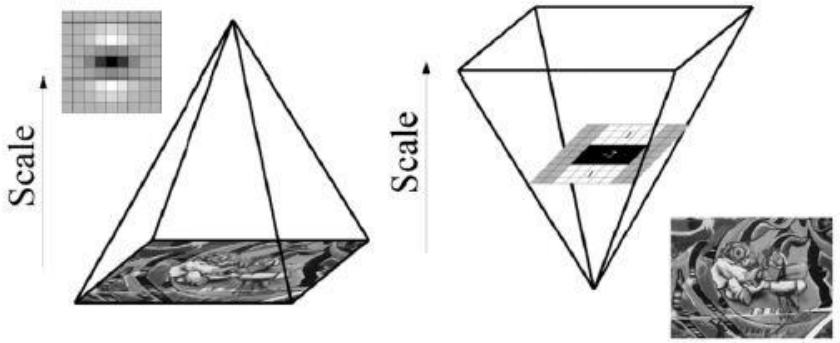
臉部身份辨識

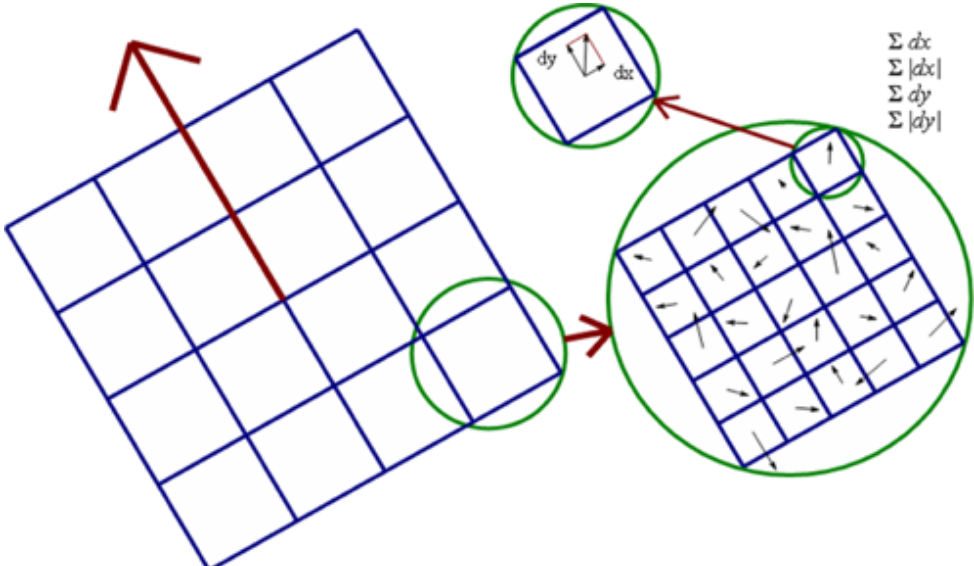
- SURF（Speeded Up Robust Features）



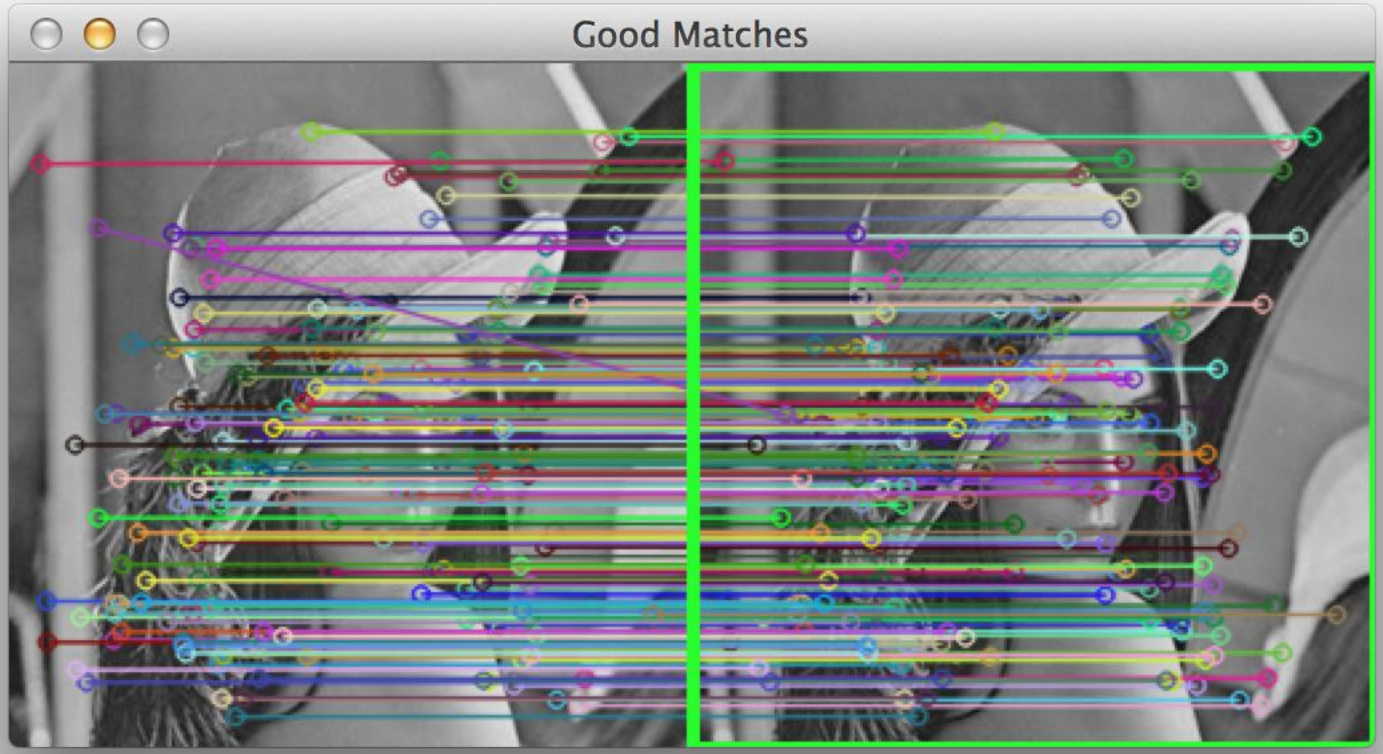
臉部身份辨識

- Integral Image
- Box Filter（Convolution Integral Image）
- Hessian matrix
- Haar Wavelet transform



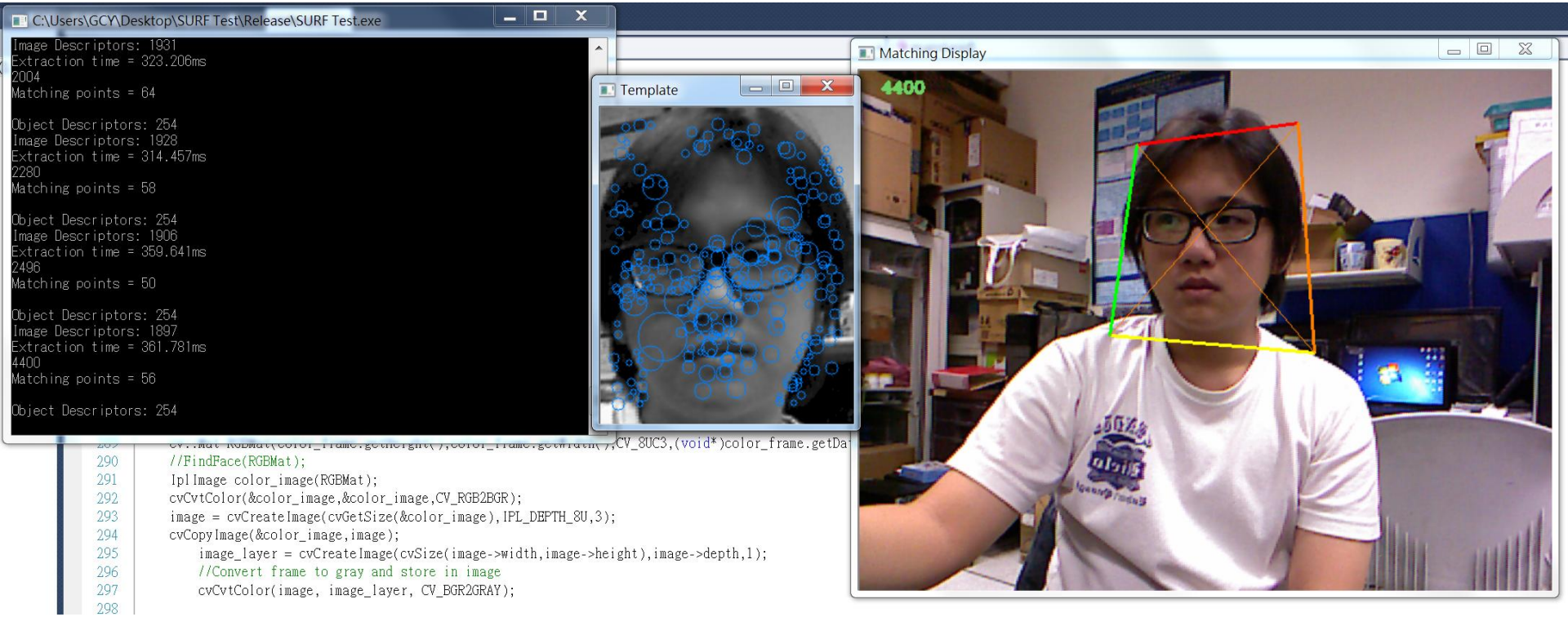


臉部身份辨識



臉部身份辨識

- 實驗結果

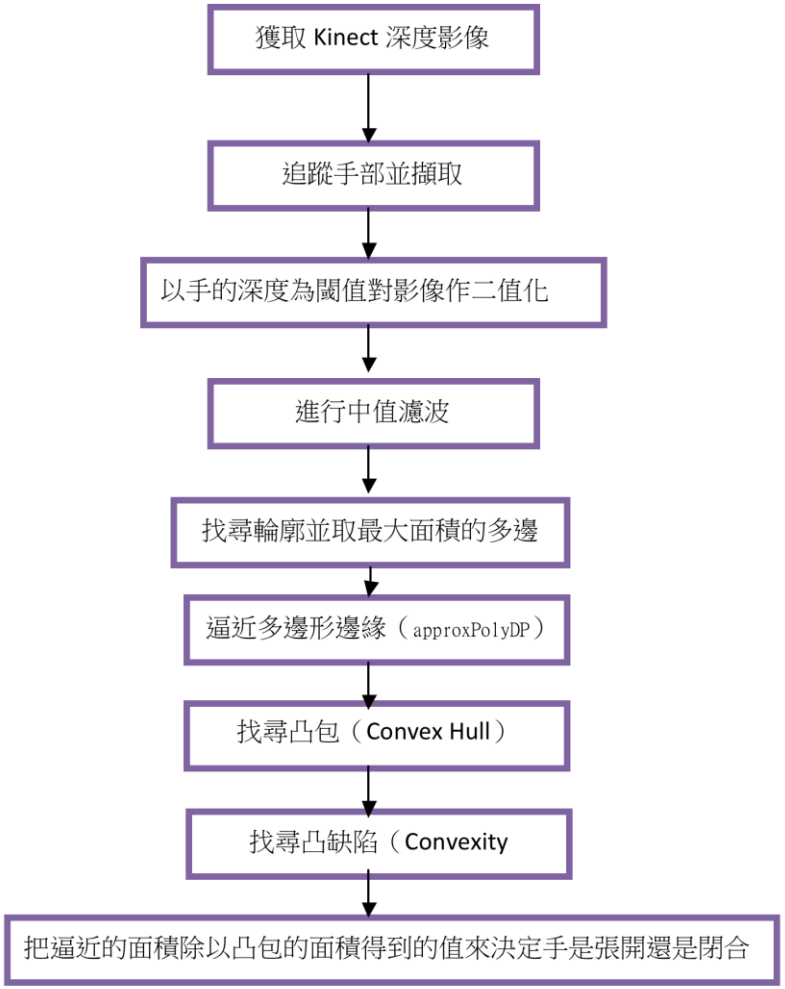

C:\Users\GCY\Desktop\SURF Test\Release\SURF Test.exe
Image Descriptors: 1931
Extraction time = 323.206ms
2004
Matching points = 64
Object Descriptors: 254
Image Descriptors: 1928
Extraction time = 314.457ms
2280
Matching points = 58
Object Descriptors: 254
Image Descriptors: 1906
Extraction time = 359.641ms
2496
Matching points = 50
Object Descriptors: 254
Image Descriptors: 1897
Extraction time = 361.781ms
4400
Matching points = 56
Object Descriptors: 254
Template
Matching Display
4400
CV_8UC3,(void*)color_frame.getDa
//FindFace(RGBMat);
IplImage color_image(RGBMat);
cvCvtColor(&color_image,&color_image,CV_RGB2BGR);
image = cvCreateImage(cvGetSize(&color_image),IPL_DEPTH_8U,3);
cvCopyImage(&color_image,image);
image_layer = cvCreateImage(cvSize(image->width,image->height),image->depth,1);
//Convert frame to gray and store in image
cvCvtColor(image, image_layer, CV_BGR2GRAY);

DEMO

手部識別

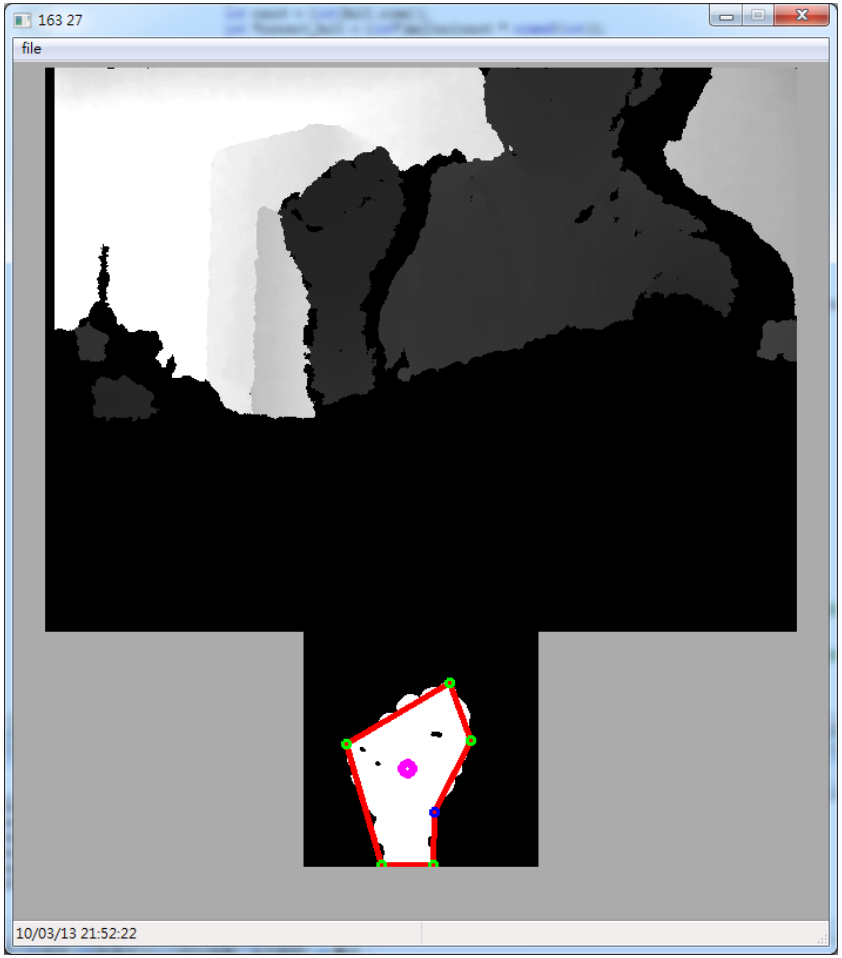
手部識別

- 演算法流程


獲取 Kinect 深度影像
追蹤手部並擷取
以手的深度為閾值對影像作二值化
進行中值濾波
找尋輪廓並取最大面積的多邊
逼近多邊形邊緣（approxPolyDP）
找尋凸包（Convex Hull）
找尋凸缺陷（Convexity
把逼近的面積除以凸包的面積得到的值來決定手是張開還是閉合

手部識別

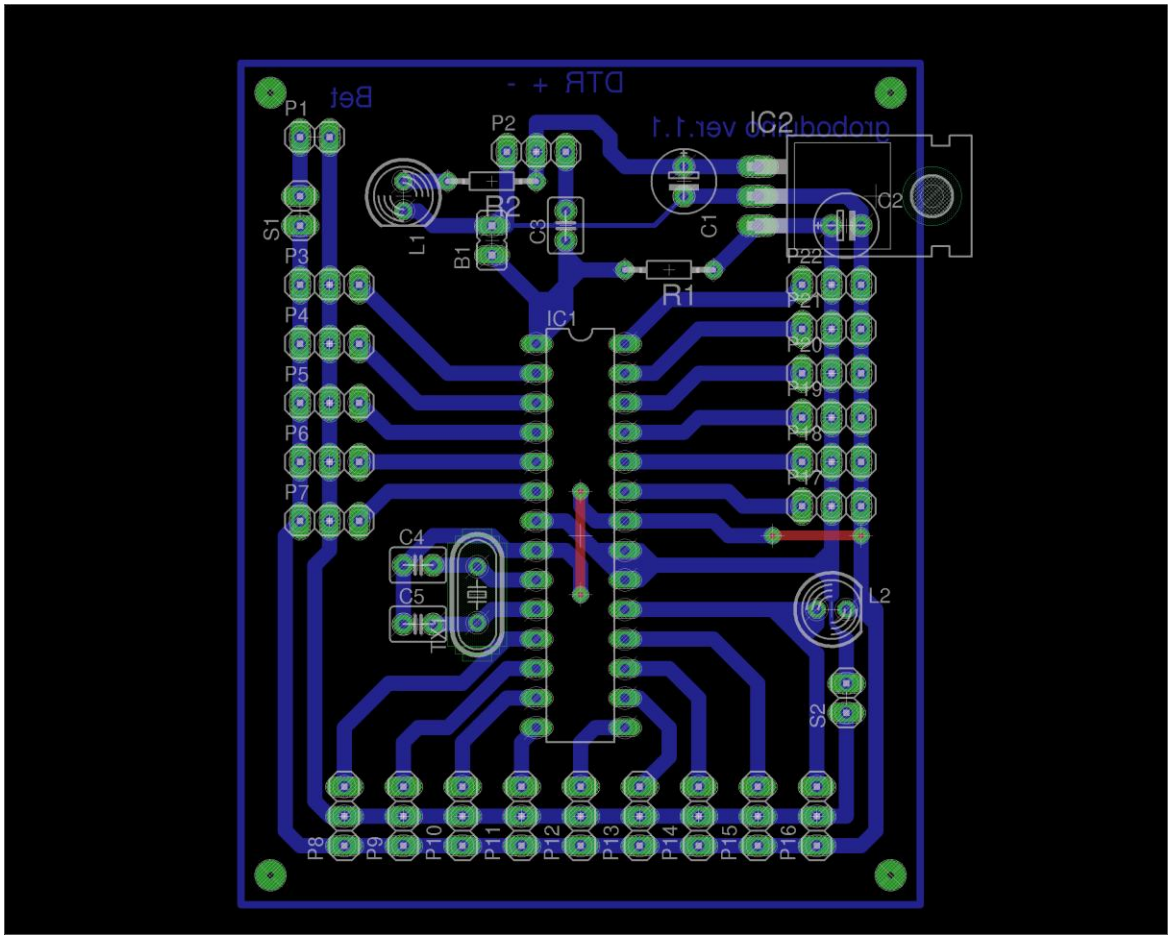
手部識別



水平軸設計

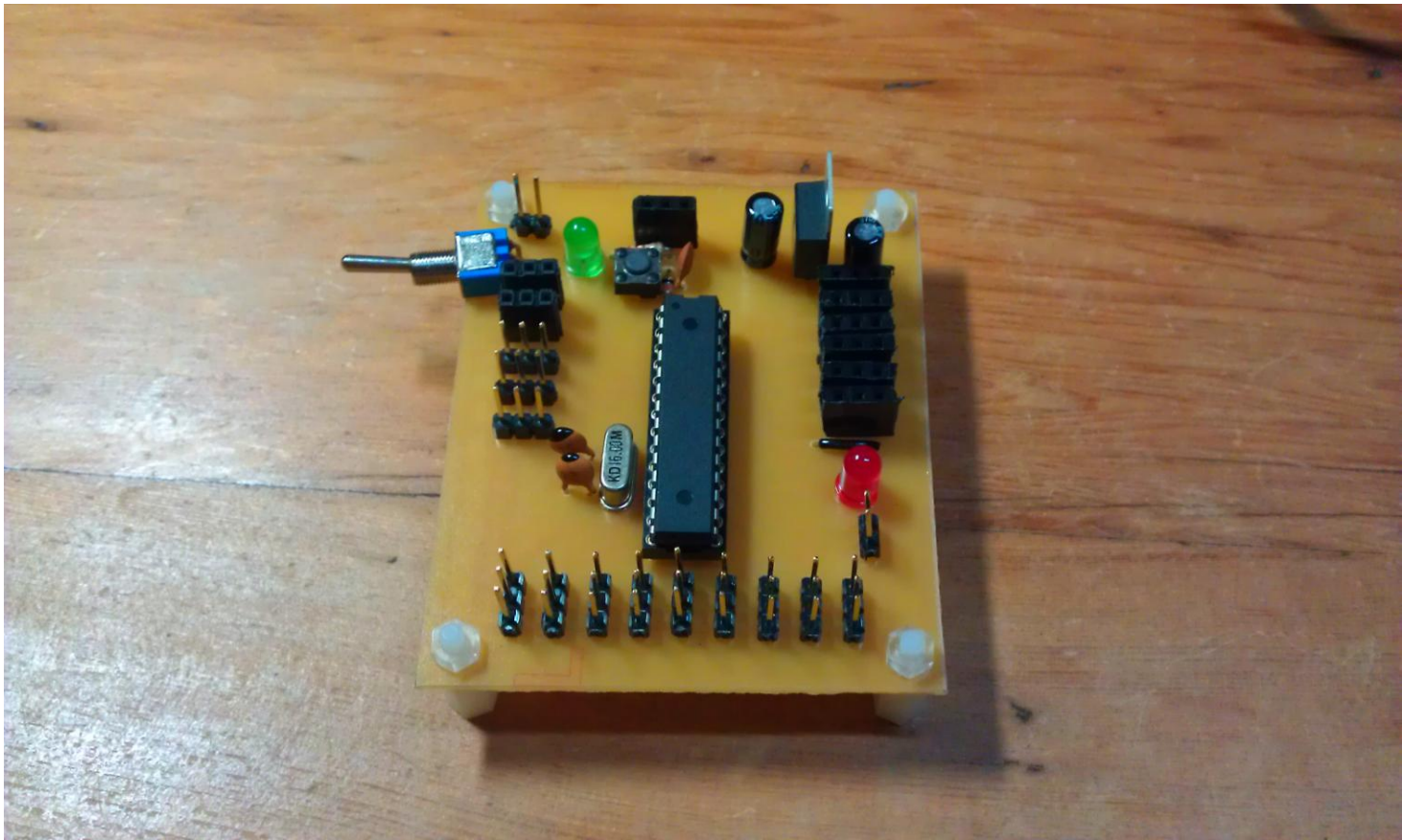
水平軸向設計

- 自行設計Layout馬達控制板



水平軸向設計

- 製作出馬達控制板



水平軸向設計

水平軸向設計

- CreateFile
- BuildCommDCB
- SetCommState
- SetCommTimeouts
- CreateEvent
- PurgeComm
- WriteFile
- CloseHandle

DEMO

結論

本專題貢獻如下：

- 使機器人控制更加直覺
- 計算標準差用於過濾資料集
- 發展判斷踢腿的深度積分波形演算法
- 人臉身分辨識
  - 利用人臉辨識取得ROI，精簡SURF計算量
- 利用類神經網路提昇動作辨識率
  - 設計與改良角度計算公式
- 手部輪廓辨識與基本手勢
- 自行設計製作電路與韌體，增加Kinect水平軸向以延伸辨識空間（原先Kinect只有垂直軸向）
- 建構軟硬體整合的NUI系統

未來目標

- 操作更直覺
- 增加反向運動學計算讓機器人更靈活
- 提高人臉身份識別度
- 可將動作影片放映至實體機器人
- 增加更多手部辨識的應用

工作分配

組員	工作內容
郭承諺	•主程式界面 •Kinect控制 •影像處理 •藍芽連接 •馬達控制板連接 •機器人動作編輯 •整合所有程式至主程式
洪亮	• 類神經網路訓練動作 • 動作識別 • 優化角度公式 • 機器人動作編輯

Q&A