國立台中教育大學
資訊工程學系大學專題

基於類神經網路動作識別之 Kinect 應用於機器人控制

指導教授：孔崇旭

組員：郭承諺、洪亮

1 簡介

1.1 研究動機與目的

早期控制機器人都是使用一個按鈕對應一個或一組動作，即使從個人電腦到智慧型終端控制軟體的本質還是一樣，因此本專題探討的是如何用更自然的方式來控制機器人以及解決可能的問題。

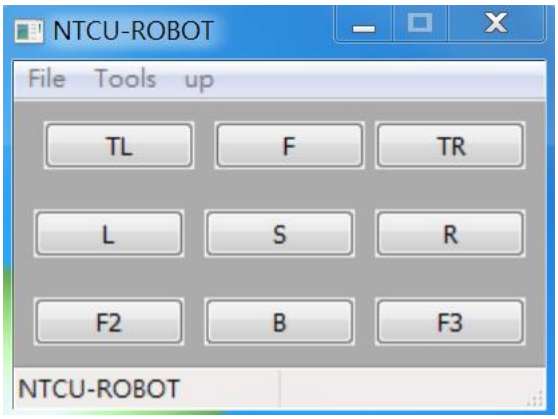


圖 1-1 PC 端機器人控制程式

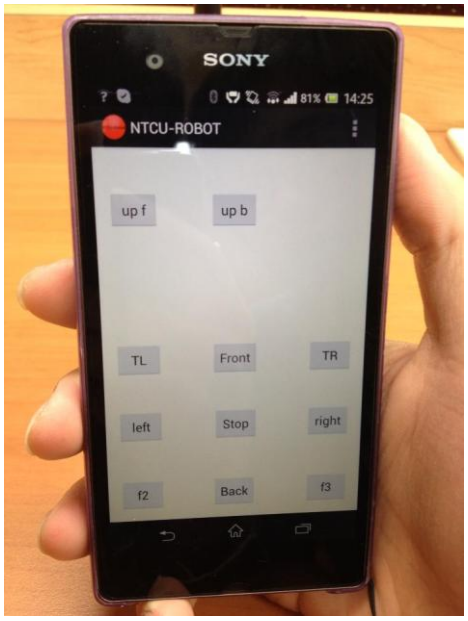


圖 1-2 智慧型手機機器人控制程式

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，XBOX360 使用此設備後可以讓玩家用肢體控制遊戲中虛擬的角色，我們認為將這項能力用在控制機器人上會是不錯的應用，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，為此我們得想辦法訓練自定義動作，接著由程式判斷後發送控制指令給機器人，為此我們採用『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。

1.2 系統功能特色

- 使用 NUI 控制機器人
- 可經由藍芽無線傳輸控制機器人
- 可快速追蹤定位人體骨架
- 動作識別率高
- 機器人本體體積小
- 程式可錄製骨架供參考分析
- 幫 Kinect 增加橫向軸增加使用範圍
- 使用之第三方函式庫皆為 Open Source

1.3 使用技術與平台

A. Kinect

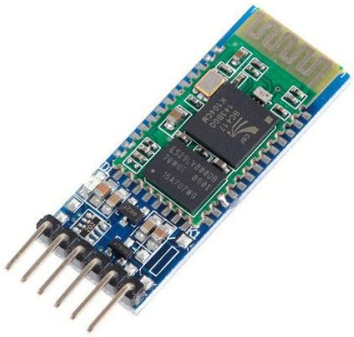
KINECT
for XBOX 360.

Kinect 在本系統用來取得使用者影像與骨架，亦及可對手部進行追蹤，Kinect 除了可以取得色彩影像外，也可以利用紅外線得到深度影像。

B. BeRobot

我們的展示機器人平台採用 BeRobot 兩足機器人，因為身長十五公分的大小以及軸數只有十四軸非常適合用來實驗，以及各零件相較於三十公分的機器人上容易取得也更便宜。

C. Bluetooth



我們選用藍芽作為將指令送給機器人的傳輸方式，因為我們的系統必須是在可以看得到機器人實體的情況下操作，因此只需要在幾十公尺內的無線傳輸技術即可。

D. WinSocket

在 Windows 作業系統底下，要對藍芽進行操作必須先列舉出範圍內所有藍芽設備，再 WinSocket 的方式對藍芽進行連接，接著就如同網路傳輸般傳送資料。

E. wxWidgets

wxWidgwets 是一款 GUI 的 Framework，底層封裝了 Win32、GTK、Mac OS API，只需要將程式碼移至各種作業系統上做些許修改再重新編譯即可跨平台執行。

F. OpenNI & NiTe

OpenNI 在本系統用取得 Kinect 的色彩影像與深度影像，而 NiTe 則是與 OpenNI 搭配使用，用來取得 Kinect 追蹤到的使用者與骨架和手部追蹤，兩者皆是可跨平台使用的。

G. OpenCV

OpenCV 用來對從 Kinect 取得的影像進行處理，包含畫出骨架、錄影、人臉身分識別等等，且提供很多影像處理的基本函式，讓我們專注在演算法的設計，且 OpenCV 也是跨平台的。

H. OpenNN

對於類神經網路 Library 的選擇我們使用 OpenNN，我們可以將要訓練元素的集合與結果用 OpenNN 訓練，並且產出訓練後的資料集合，再由我們的控制程式引入這個集合併套用 OpenNN 的公式計算出最相近的結果。

I. Arduino

Arduino 是一個用於 ATmega 系列晶片的 Bootloader，可用來運行我們的程式，本專題用來將要修正角度傳送並控制伺服馬達。

J. Eagle PCB

由於本專題需要能夠運行 Arduino Bootloader 的硬體，因此我們自行設計一個能夠讓 ATmega 系列晶片運行的電路，並且 Layout 成為 PCB 電路板，而 Layout PCB 用的就是 Eagle PCB。

1.4 系統架構與設計

主程式包含：取得 Kinect 資料、影像處理、類神經網路計算、藍芽通訊、計算修正角度，而資料集合訓練程式可讀取 Kinect 人體骨架角度資料並由 OpenNN 進行訓練，最終由主程式讀取訓練後的資料集合再由輸入並計算最佳動作，由藍芽傳送指令給予機器人，並且可在使用者操作時，即時修正偏離畫面角度並控制伺服馬達使 Kinect 轉向。

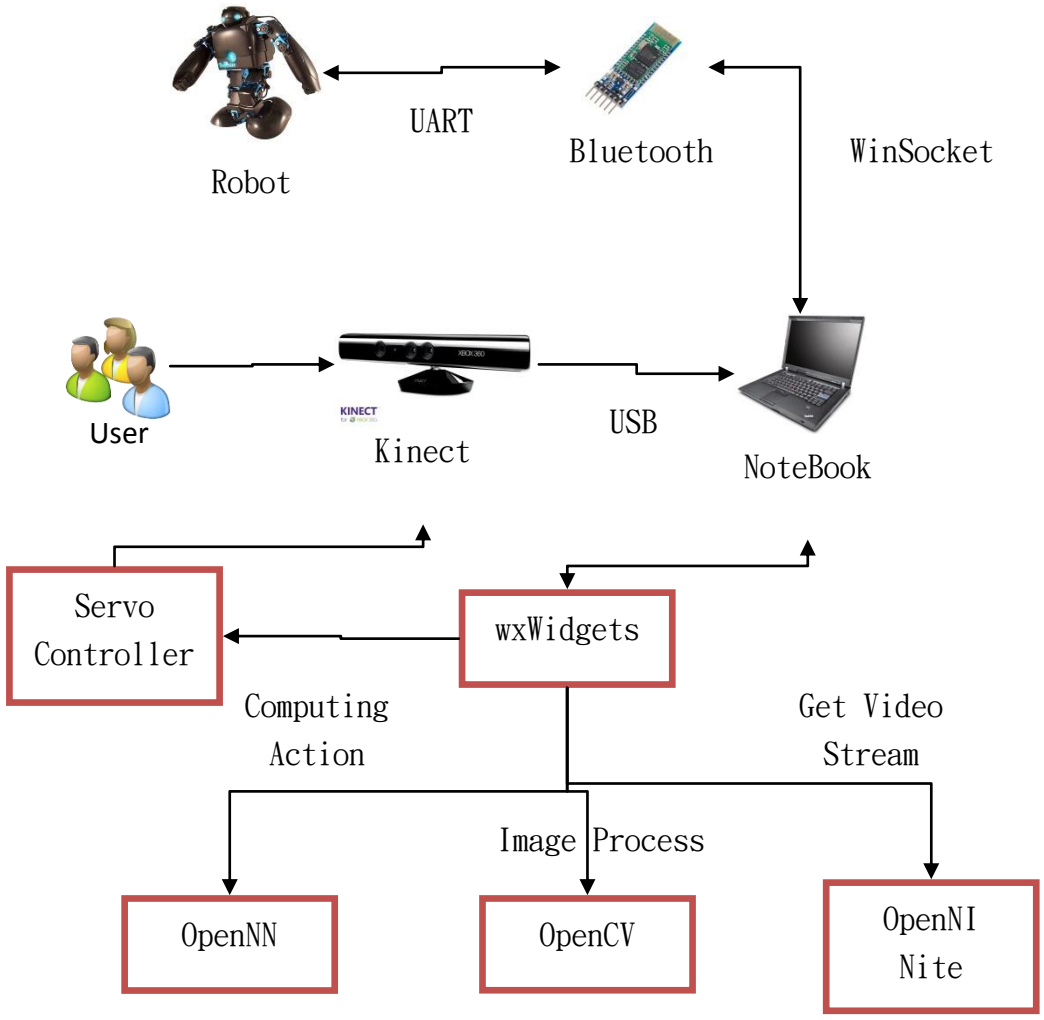


圖 1-3 主程式系統架構圖表

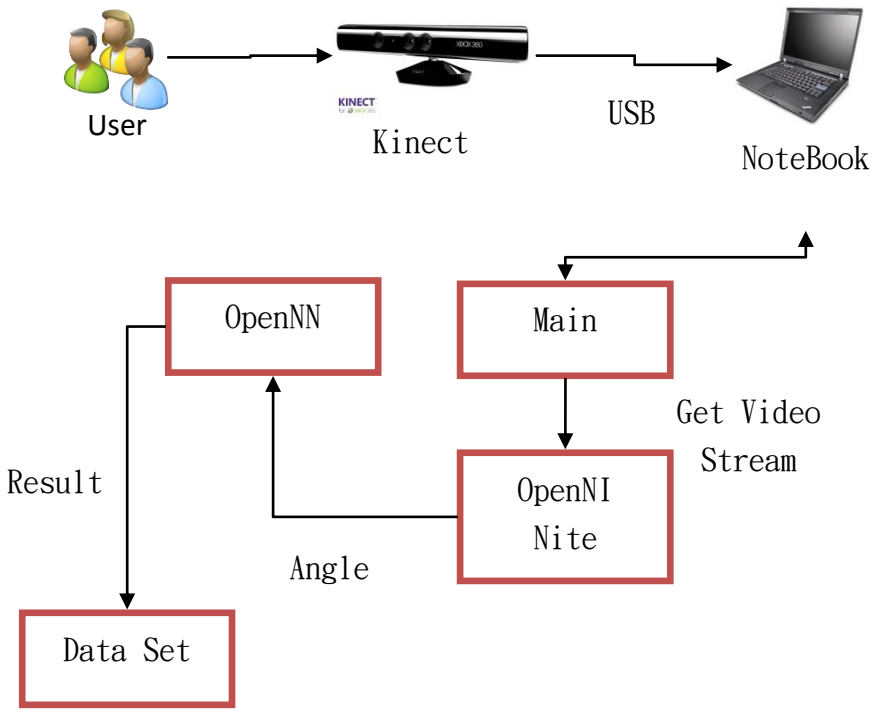


圖 1-4 資料集合訓練程式架構圖表

2 相關研究

2.1 類神經網路

2.1.1 類神經網路簡介

類神經網路是一種模仿大腦與神經元做成的運算模型，以多個神經元連接後進行計算。類神經網路可以透過預期的輸入輸出進行訓練，為非線性的統計模型，因此可用來尋找輸入與輸出間的關係、探索資料的模式以及處理包含雜訊的資料。

2.1.2 類神經網路

類神經網路中每個神經元包含輸入(input)、突觸權值(synaptic weight)、偏值(bias)、激勵函數(activation function)以及目標輸出(output)。每個輸入都有相對應的權值，激勵函數通常為非線性函數，而最後輸出計算結果。

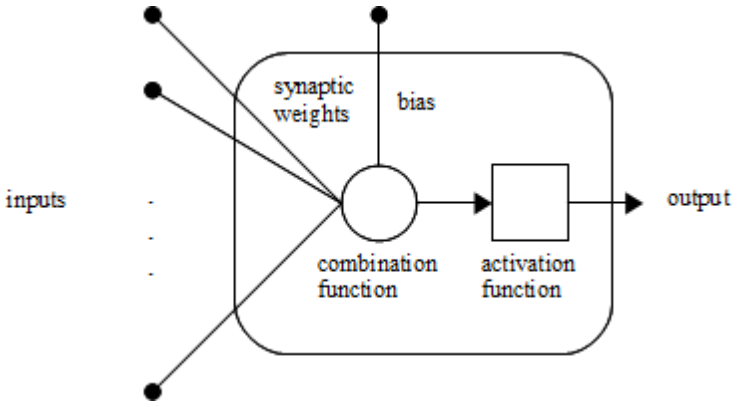


圖 2-1 類神經網路的神經元構造

類神經網路的基本架構包含輸入層(Input layer)、輸出層(Output layer)以及隱藏層(Hidden Layer)。輸入層負責接收傳遞進來的問題，而最後結果會由輸出層輸出。隱藏層中可為多層，但一般使用一層即可。隱藏層的神經元數量並不固定，數量越多則非線性的特徵會更明顯。

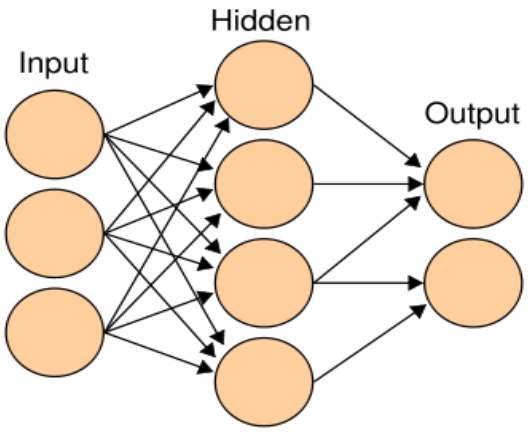


圖 2-2 類神經網路的基本架構

2.1.3 訓練機制

目前類神經網路常見的訓練方法為「倒傳遞演算法」。需要給予一組輸入與預期的輸出，先正向計算出結果後，在逆向取得計算結果與目標結果的差值。再對每個突觸權重都進行以下步驟：

1. 將輸入乘以差值，就取得了權重的梯度。
2. 再將權重梯度乘以一個比率後，由突觸權重減去該數值。

權重梯度乘以的該比率就稱為「訓練率」。訓練率越大則類神經網路的訓練速度越快；訓練率越小則訓練結果更精準。由於權重梯度的大小表示著類神經網路的誤差，因此必須減少梯度以達到訓練類神經網路之目標。

2.1.4 類神經網路優缺點

類神經網路的優點有：

1. 可自行從訓練的資料集中找出規律。
2. 可用於分析非線性的統計模型。
3. 訓練完畢後可迅速得到結果，不需再經過繁瑣的運算。

類神經網路的缺點有：

1. 監督學習式需要較大量且正確的資料集。
2. 訓練過程需耗費大量的效能與時間
3. 可能會有過擬合(Over-Fitting)或欠擬合(Under-Fitting)的問題

本專題使用的資料集筆數大約在兩萬筆左右，且輸入與目標輸出的個數也較少，因為實際的訓練時間大約在十五分鐘以內。

3 動作姿態辨識

3.1 身體姿態判斷

3.1.1 身體姿態判斷簡介

我們以雙手實驗如何辨識身體姿態，其中包含四個數據：

I. 左肩膀角度
II. 右肩膀角度
III. 左手肘角度
IV. 右手肘角度

並且做出九種不同的姿勢：

I. 雙手向下伸直
II. 左手向上彎曲 90 度
III. 右手向上彎曲 90 度
IV. 雙手向上彎曲 90 度
V. 雙手向下彎曲 90 度
VI. 雙手水平伸直
VII. 雙手向上伸直
VIII. 左手向上彎曲 90 度＋右手向下彎曲 90 度
IX. 右手向上彎曲 90 度＋左手向下彎曲 90 度

並將這些數據整合成一個資料集，作為訓練類神經網路的使用。

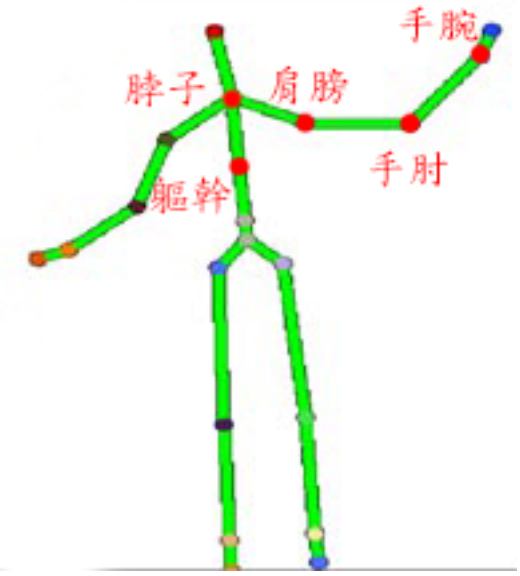


圖 3-1 Kinect 抓取的關節點與主要使用的關節點

3.1.2 取得關節角度方法

我們寫了一支程式，會讀取 Kinect 抓到的關節座標，並且以向量夾角的三角函數公式取得個別的角度數據，再用固定的格式製成訓練用的資料集。

$$\cos\theta = \frac{\overrightarrow{AB} \cdot \overrightarrow{AC}}{|AB| \times |AC|}$$

向量夾角的三角函數公式

將兩個向量帶入公式後，即可獲得該夾角的 cos 值，再以反三角函數的反餘弦 arc cos 取得其弧度，最後再轉換回角度即為所求。

3.1.3 計算手肘角度

一開始手肘角度的計算方式使用 手肘-手腕 與 手肘-肩膀 兩個向量，取得的角度範圍為 0~180 度，但是當手肘向上彎曲 90 度與向下彎曲 90 度時，得出的結果皆為 90 度。

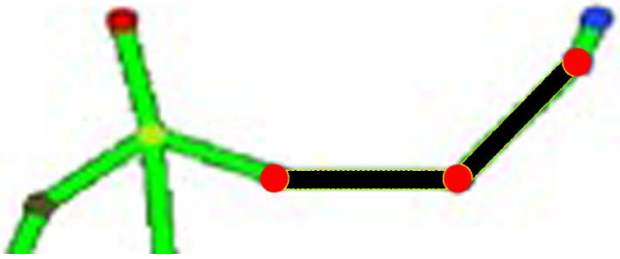


圖 3-2 以向量方式計算手肘角度

因此在第四版時我們嘗試了第二種方法，增加手腕座標點投影至 手肘-肩膀 向量上的投影點，透過投影點到手腕的斜率，我們可以簡單判斷手肘部分是朝上或是朝下，也因此將辨識範圍擴充至 0~360 度。

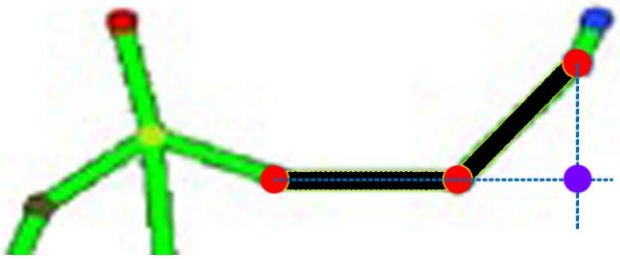


圖 3-3 增加投影點使角度辨識範圍擴充

3.1.4 計算肩膀角度

而肩膀角度部分也是以同樣方式，最一開始時使用 肩膀-手肘 與 肩膀-手腕 兩向量，但是以此方式計算出來的數值實際上與手肘數值相關，因此整體數據的資料分佈並不明確，並且類神經網路訓練後辨識度非常低。

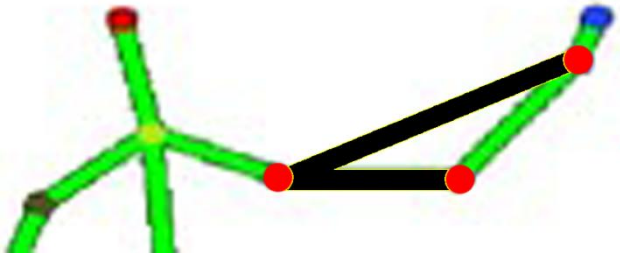


圖 3-4 第一版計算肩膀角度

為了解決這問題，在第二版時我們改進了計算肩膀角度的方式，改以 肩膀-軀幹 與 肩膀-手肘 這兩向量計算，再把計算出的數值減去 45 度，就可以使手臂垂下時的角度大約在 0 度左右，而改進後也大幅的提升了類神經網路的辨識能力。但是同時產生了新的問題，當手臂水平伸直後往上舉起 45 度時，可以得到最大角度 135 度，而此時不管手臂舉起或放下皆是使角度值減少。

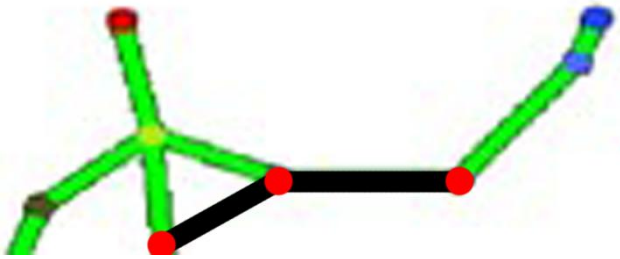


圖 3-5 第二版改以肩膀與軀幹向量計算肩膀角度

我們在第三版改進的計算方式使用了 肩膀-手肘 與 向下垂直向量 兩向量，這個方法可以使手臂向下伸直時，角度值約在 0 度上下；而手臂向上伸直時，角度約在 180 度上下，因此得到了較正確且較直觀的數值。

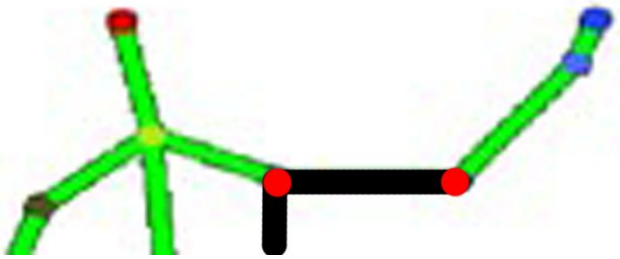


圖 3-6 第三版改以垂直向量計算肩膀角度

上述的第三版計算方法有個問題是，當使用者的身體傾斜時，所得到的角度值會跟著有所偏差，因此我們最後使用的計算方式是將垂直向量改為 脖子-軀幹 此向量，如此一來以身體作為主軸，就算使用者身體傾斜也不影響肩膀角度計算結果。

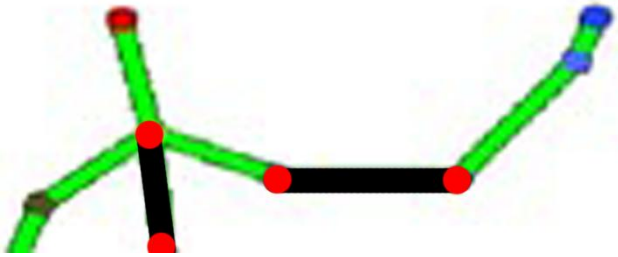
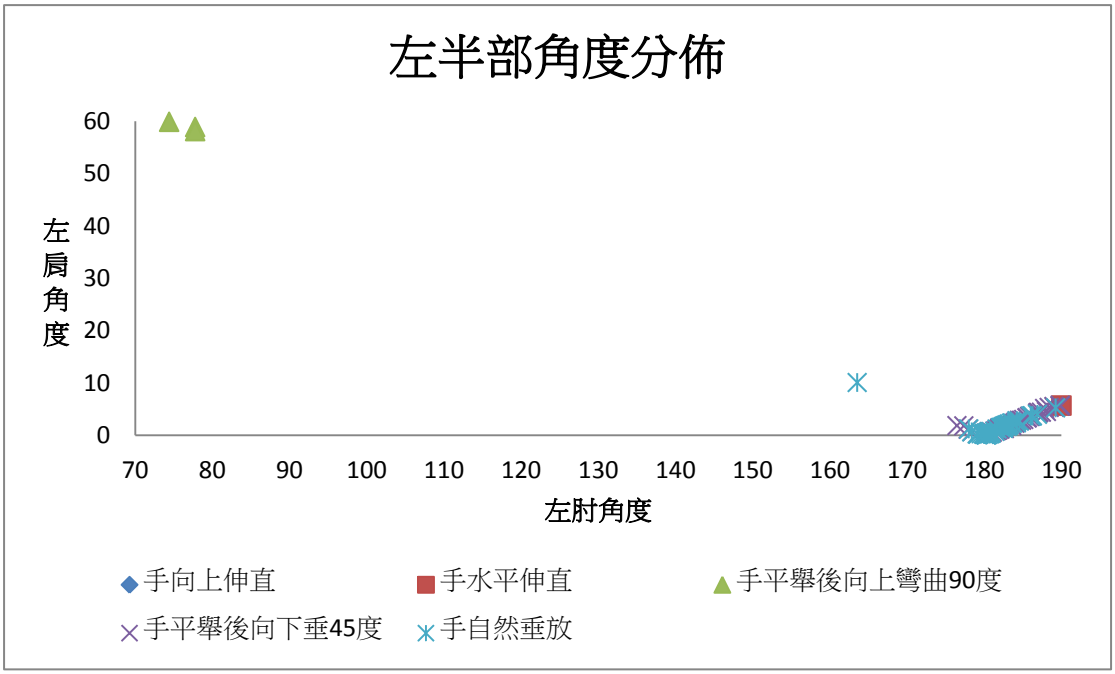
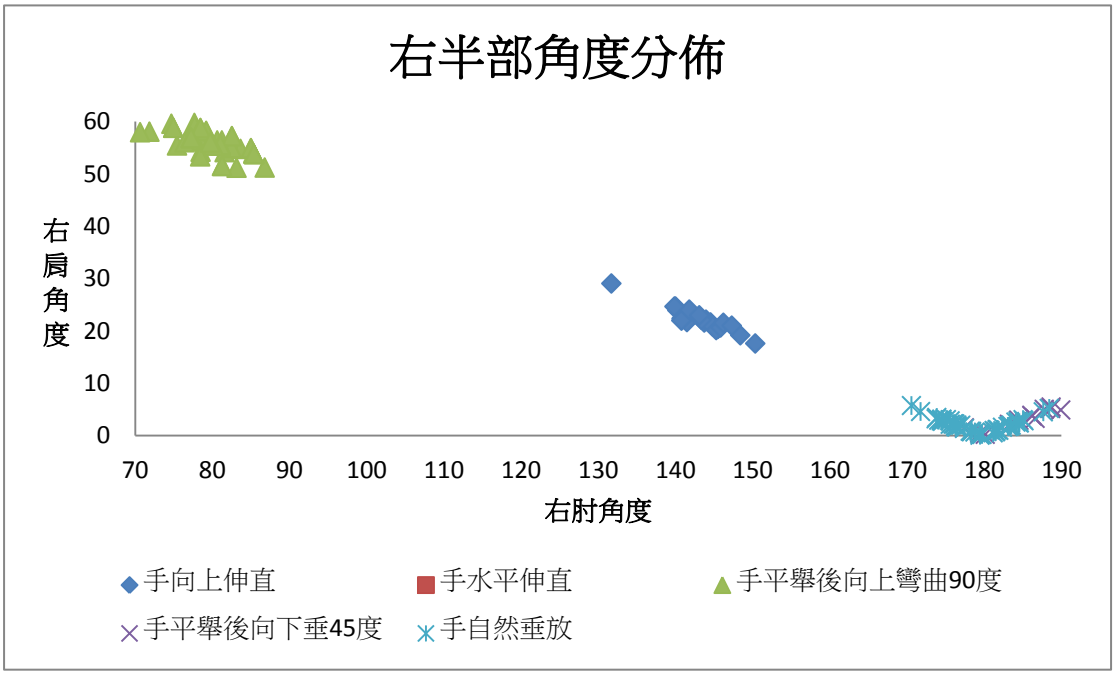


圖 3-7 第三版最後改以身體主軸計算肩膀角度

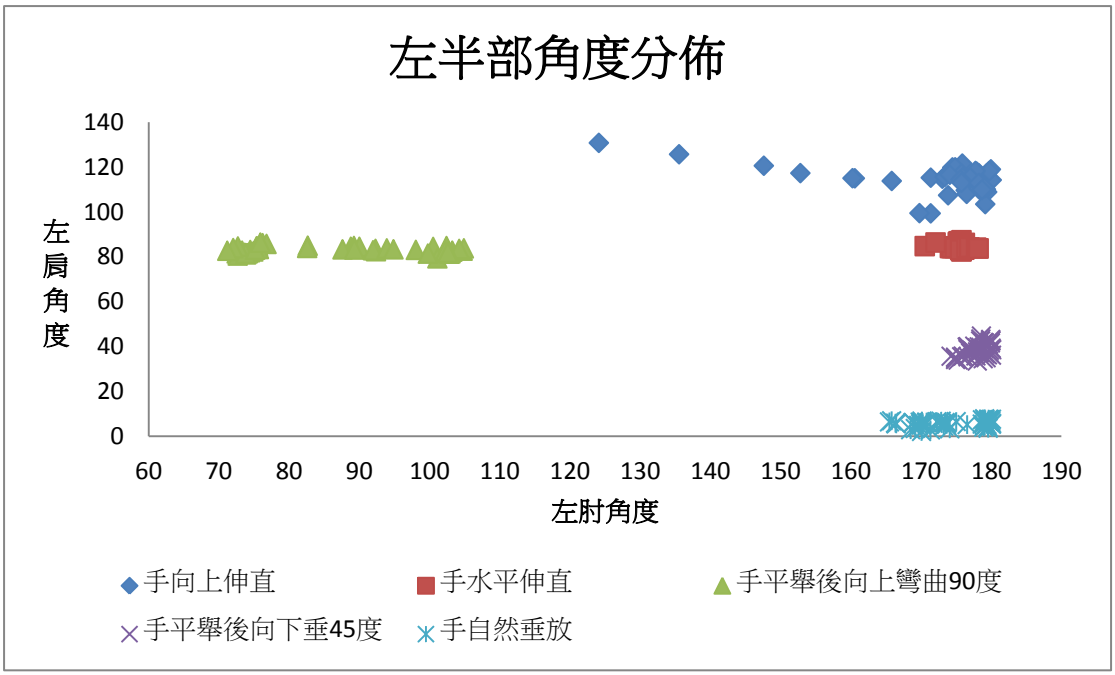
3.1.5 資料分佈



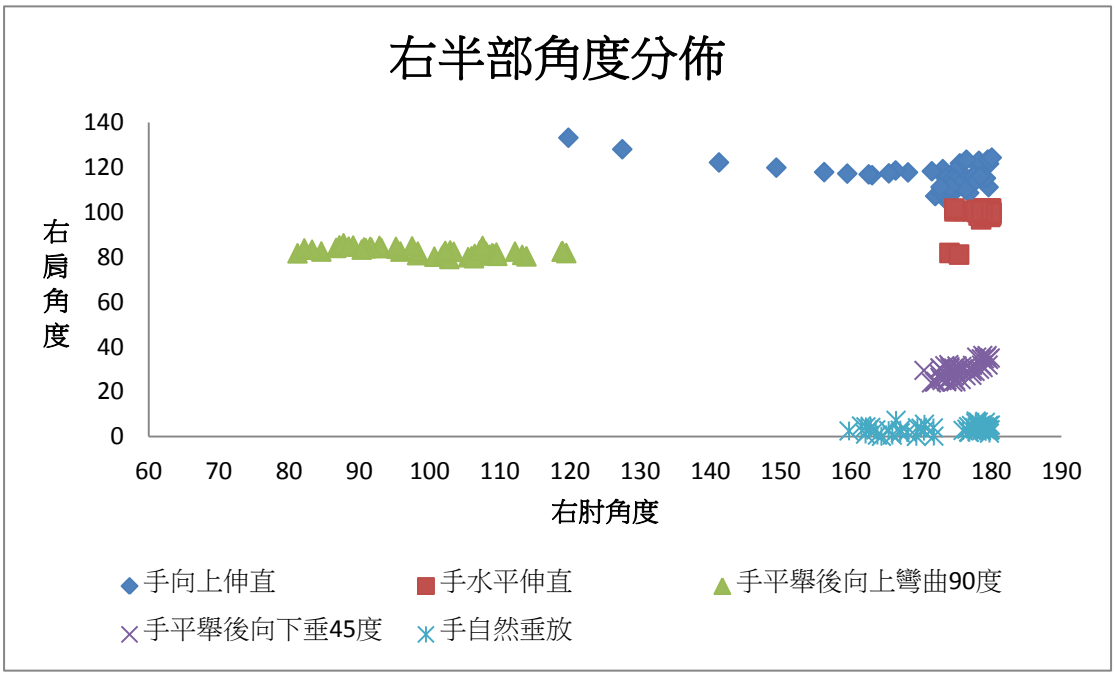
第一版計算方式的左半部角度分佈圖



第一版計算方式的右半部角度分佈圖

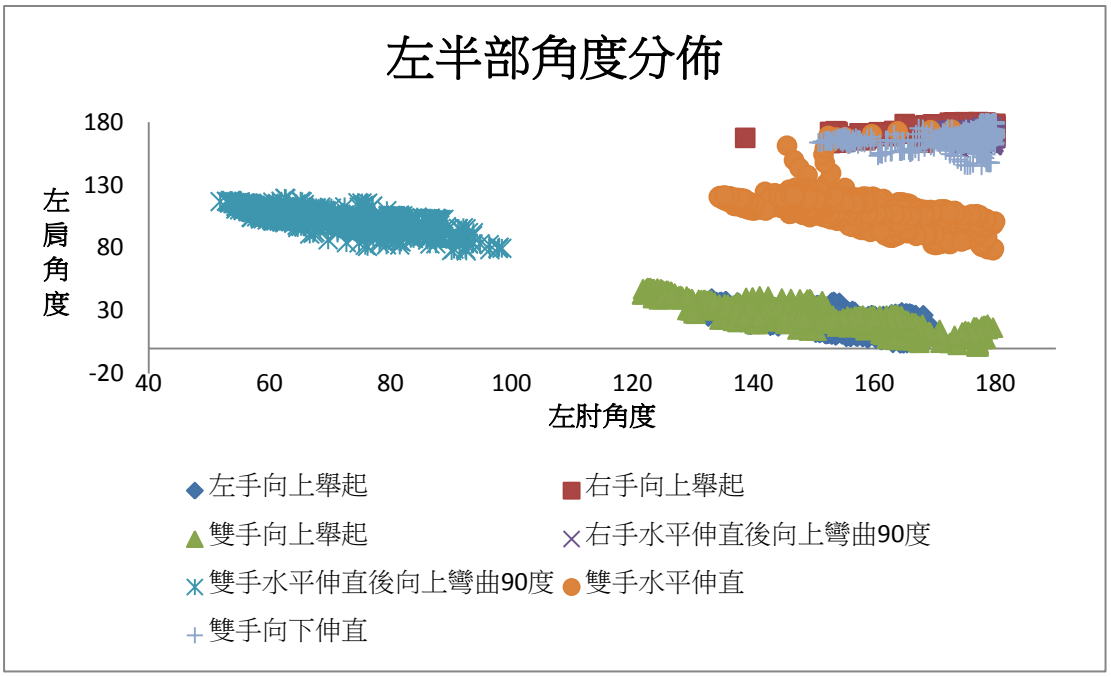


第二版計算方式的左半部資料分佈圖

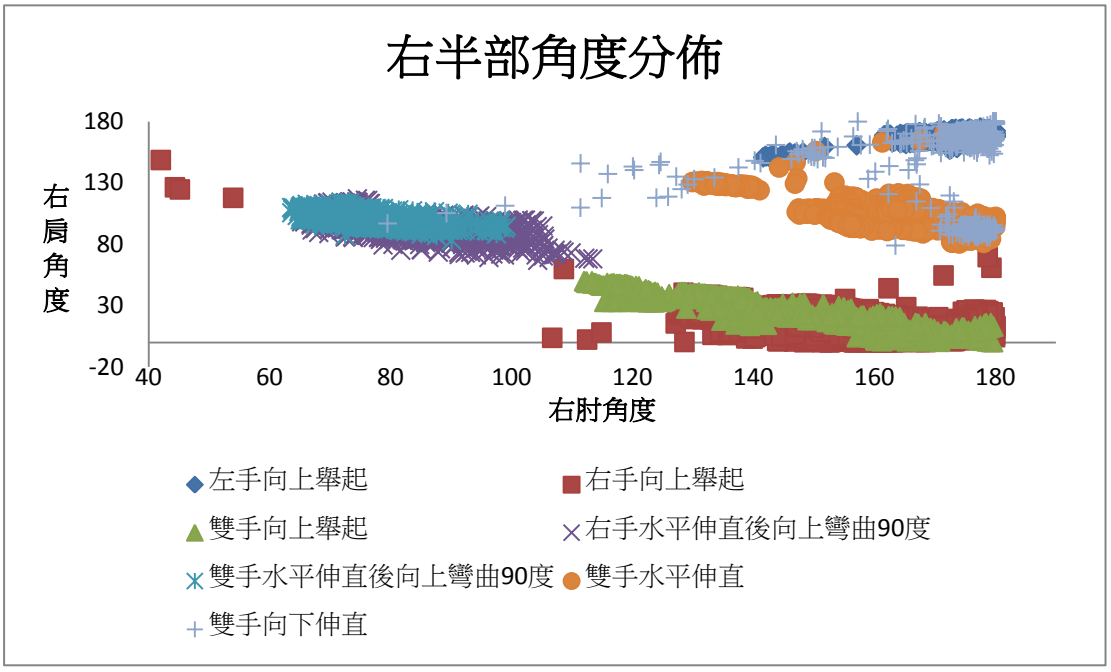


第二版計算方式的右半部角度分佈圖

由圖我們可以發現，其中有數個動作皆為手臂伸直的狀況，因此當肩膀角度的分辨度不佳時，數據的分區同時也非常不明確。而在第二版改進後很明顯的改善了數據分佈的問題，因此改進後的類神經網路辨識度也大幅度的提升了。



第三版計算方式的左半部資料分佈圖



第三版計算方式的右半部資料分佈圖

3.1.6 辨識成功率

我們在專題最後有做了一次測試，分別以訓練用資料以及一份特別製作的測試用資料進行測試，並使用程式自動將測試資料輸入類神經網路後，比對判斷結果是否正確。

訓練資料

動作 \ 神經元個數	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

動作 \ 神經元個數	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%

圖 3-8 訓練資料與測試資料之判斷正確率

由上圖我們可以發現，第二及第三個動作可能訓練時的資料集就較為不準確，因此測試出來的正確率皆比較低。

並且由圖同樣可以發現，當神經元的個數越高，並不代表正確率也會跟著提升。成功率較高的動作在神經元個數增加時，大致上辨識成功率也會隨之增加；但是觀察成功率較低的動作中，神經元個數增加可能導致成功率上升也可能導致成功率下降，在測試資料的第二個動作尤其下降的嚴重，此現象就是類神經網路因為訓練用資料集不夠正確導致過擬合或欠擬合的問題。

3.2 踢腿判斷

3.2.1 取得深度變化波形

由於踢的動作無法使用類神經網路辨識，因此僅能由我們自己觀察踢球時足部的深度變化，這一連串的動作即是對於足部運動進行積分，正負號代表的是方向性，我們可以藉由採樣結束後判斷積分值是否達到閾質(threshold)，首先人在踢腿時會先將腳後抬(圖 3-9)，接著會再從後抬的高點往前踢(圖 3-10)，當到達前方的高點後就是完成踢的動作，這時候腳會回到原本的位置(圖 3-11)，因此我們可以得到「L － 2L ＋ L＝0」，另外 Kinect 深度變化是越遠越大，因此遠離 Kinect 的積分會是正號反之則是負號。

圖 3-9 腳後抬 L

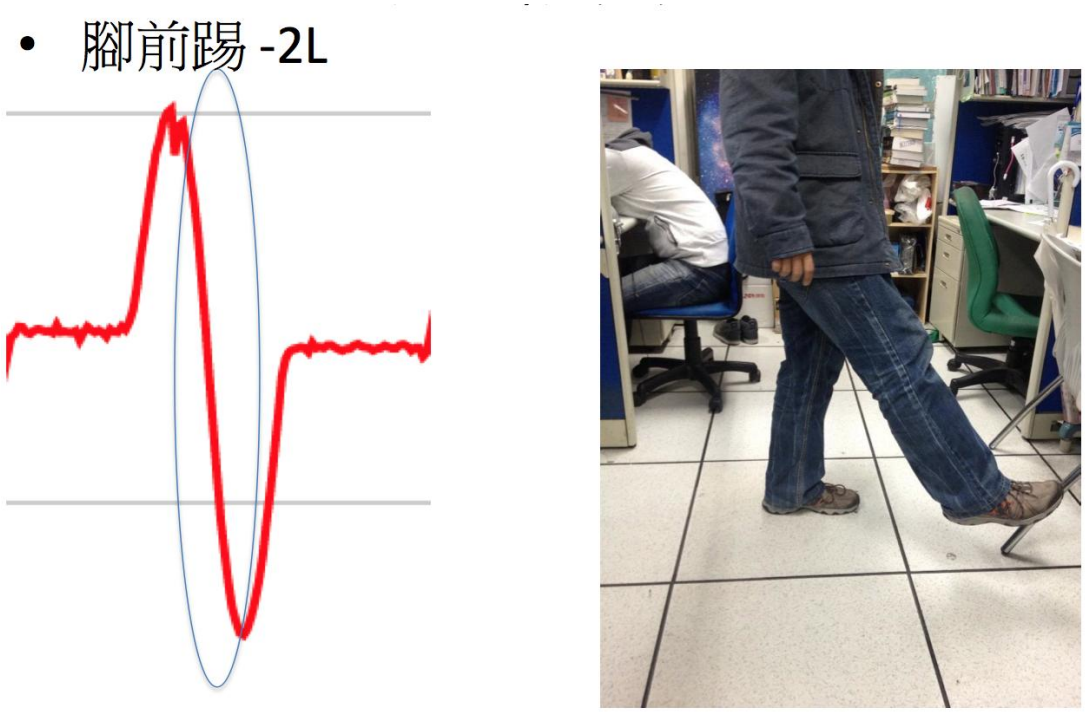


圖 3-10 前踢-2L

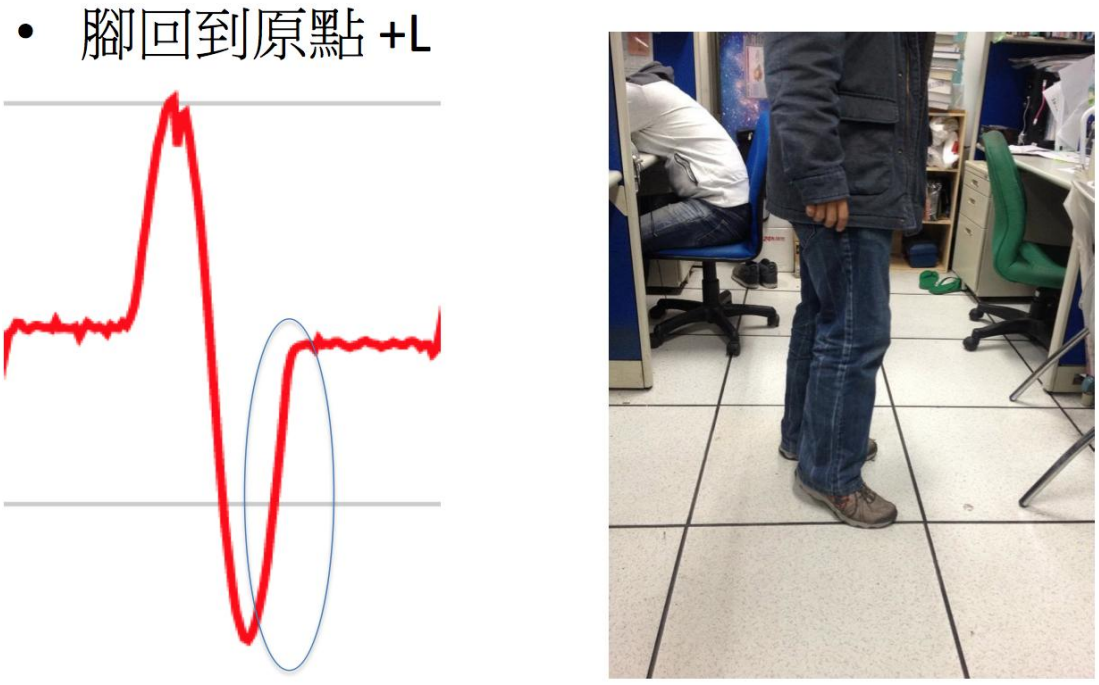


圖 3-11 回到原點 L

3.2.2 卡爾曼濾波

一開始我們取得的深度變化波型會有太多的 Noise，考慮到可能會影響到程式判斷，因此我們得使用濾波器來消除這些 Noise，而選擇的濾波器我們選擇卡爾曼濾波器(kalman filter)，卡爾曼濾波可以根據上一次的結果計算出現在的結果並且預測下一次的結果，因此每次的預測都可以使得到的值進行修正來達到濾波的效果，此外卡爾曼濾波的另一個好處是由於只需要保存上次結果與下次預測結果，因此記憶體的占用較小，腳的深度變化是線性的，因此卡爾曼濾波的選擇也是線性版本，使用隱馬可夫模型(Hidden Markov Model)來統計並估計結果，最終可以看到藍色部分是原始數據而紅色線則是使用卡爾曼濾波後的結果，明顯看到紅線濾除了藍色部分的一些 Noise。

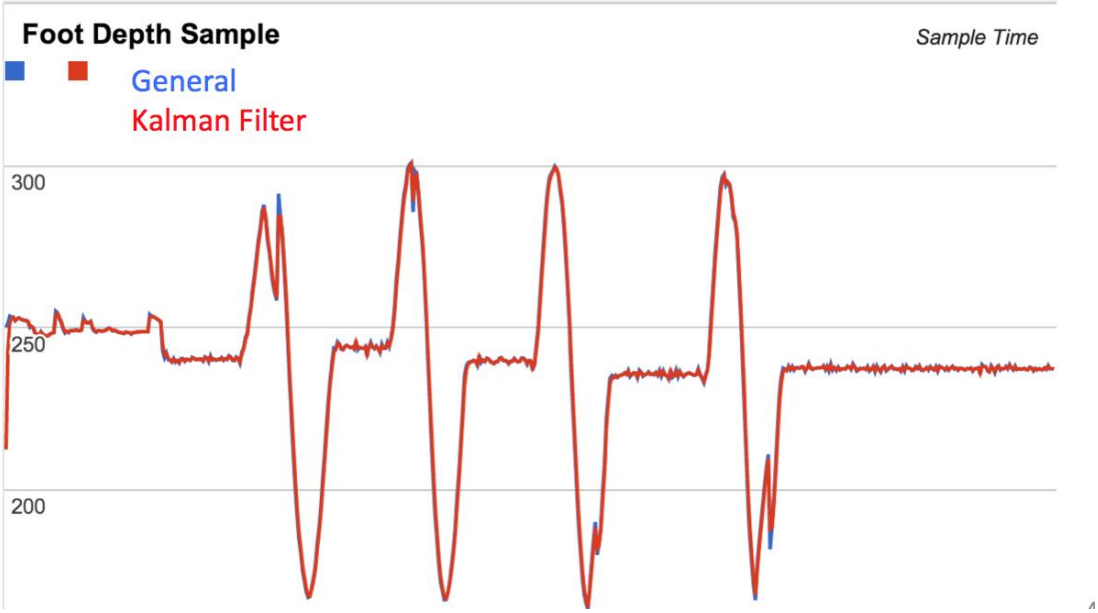


圖 3-12 結果

3.2.3 判斷

程式部分(圖 3-13)只要將這一次與上一次的腳部位置相減，並且將這個值累加做積分，就可以判斷是否爲踢的連續動作波型，然後藉由藍芽發送踢的指令給予機器人，並且要停頓幾秒等動作執行完以免受到其他動作干擾穩定性。

```
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;
}
```

圖 3-13 判斷程式碼

4 影像處理

4.1 人體影像處理(Kinect)

選擇 Kinect 主要的原因除了可以取得人體骨架資料外，也可以取得人體各部位的各個點的三維座標來轉換成電腦中的三維座標，Kinect 可以由紅外線測出目標在空間的深度，借此讓影像除了平面的兩個軸向也有第三的維度，使我們有更多的發揮空間。

4.1.1 取得影像

Kinect 可以利用 Kinect SDK 或者是 OpenNI 這兩個 Library 來取得影像，本專題主程式使用的是 OpenNI，OpenNI 可以取得 Video Stream，有彩色(圖 4-1)和黑白(圖 4-2)的差別，而 User Tracker 以及 Hand Tracker 可以追蹤人體(圖 4-2)與手部(圖 4-3)，借由 OpenNI 這個 Interface 取得到我們要的資料後就可以做出更多的應用。

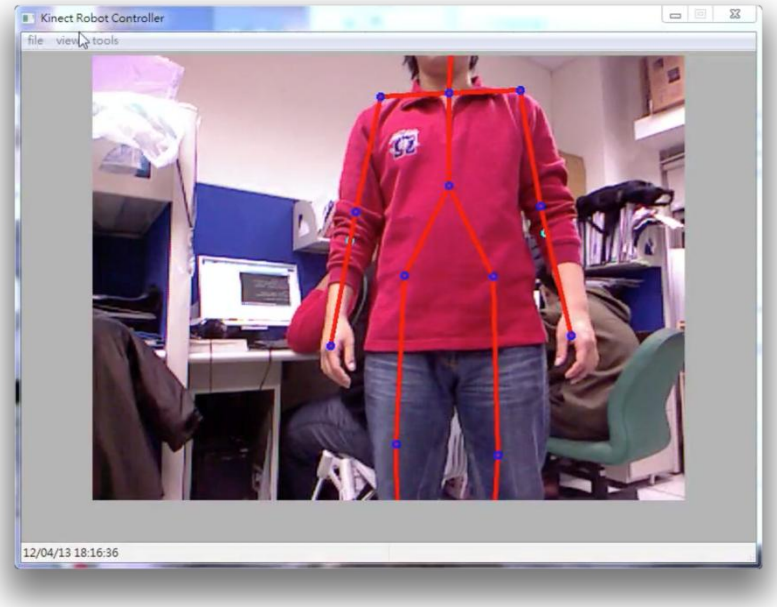


圖 4-1 彩色影像

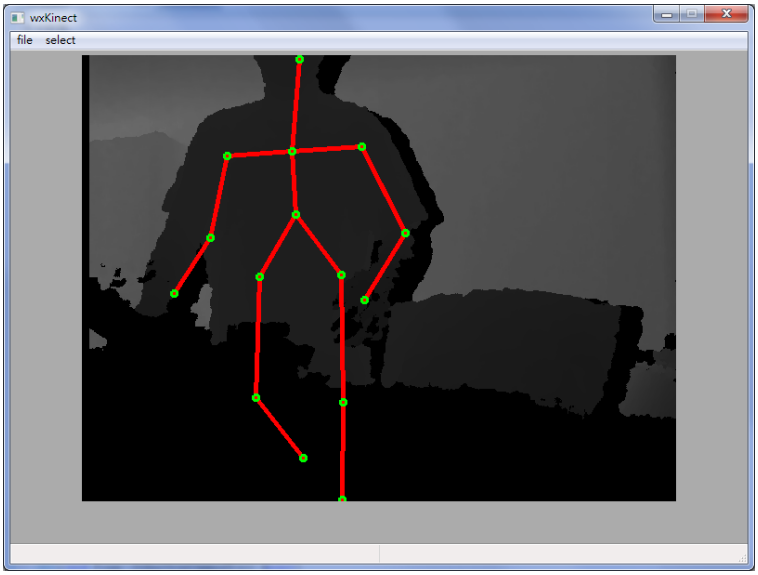


圖 4-2 深度影像與骨架

圖 4-3 手部追蹤與輪廓辨識

4.1.2 影像格式轉換

由於從 Kinect 取得的影像要由 OpenCV 來作一些影像處理的工作，以及也要將畫面畫到我們用 wxWidgets 所寫成的主程式 Panel 上，因此得對從 Kinect 取得的畫面進行影像格式的轉換，得先將 Kinect 得到的影像轉換為 OpenCV 的八位元三通道的 RGB 圖像，接著要將 OpenCV 格式的影像記憶體拷貝一份供影像處理用，而原本就轉換為 wxImage 的格式，差異在於 OpenCV 的格式為 RGB 通道按照順序排列，而 wxImage 是以 BGR 為順序排列，而進行影像處理的部分等到處理完得到資料後再轉換到 wxImage 就可以標示出人體的骨架或者臉部範圍等等(圖 4-4)。

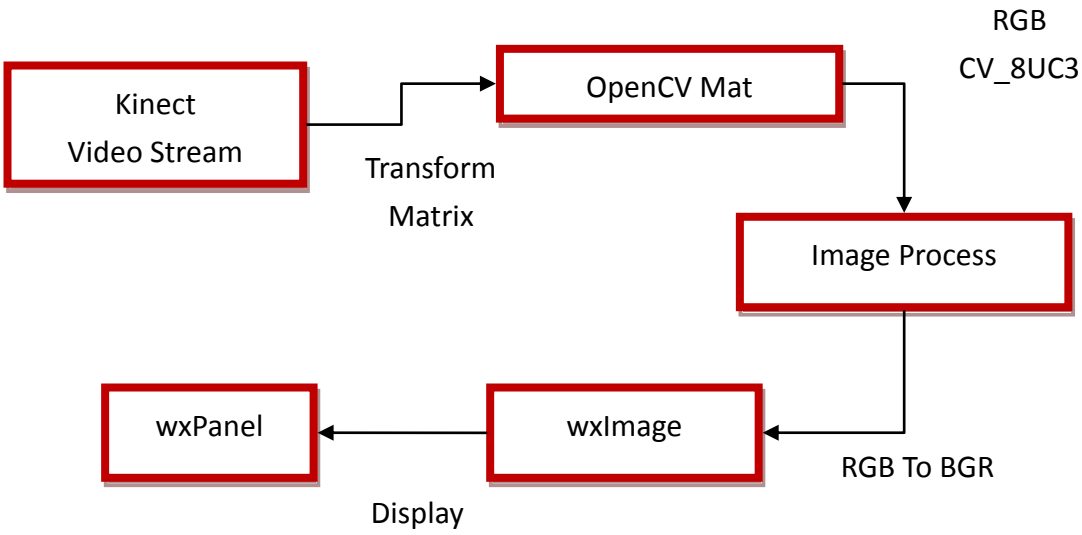


圖 4-4 處理流程

4.1.3 繪出骨架

繪出骨架則是利用 User Tracker 得到使用者各個關節的節點之後，就可以連結各個節點形成一個人形並且即時跟著互動(圖 4-5)。

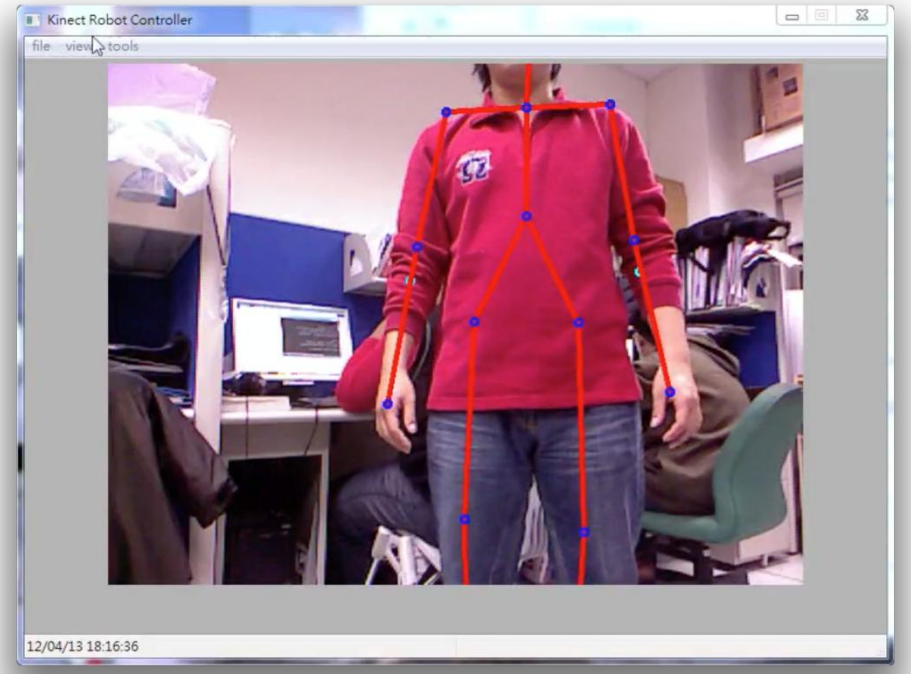


圖 4-5 使用者骨架繪製

4.2 人臉身分識別

4.2.1 人臉辨識

為了對使用者進行臉部特徵的辨識，第一步要先找出畫面中使用者的臉，其概念類似濾波器的原理並搭配機器學習來提高辨識率，本專題參考「Robust Real-Time Face Detection」這篇被廣泛引用與實作的論文的演算法來進行實作。

整個演算法的流程主要有三個階段：Integral Image & Rectangle Feature、Ada boost、Cascade classifier，Integral Image & Rectangle Feature 階段主要是利用 Integral Image 對影像進行積分即對某一區域進行加總(圖 4-6)得到一個值就是 Feature，可以用來建立 Rectangle Feature(圖 4-7)，Rectangle Feature 就像是遮罩一樣在要進行濾波的圖像進行移動並且相減來找到想要的影像區塊，而 Ada boost 演算法會先由是辨識目標的圖片與不是辨識目標的圖片組成訓練集(training set)訓練出弱分類器，得到一堆弱分類器後可以再將它們集合起來變成一個強分類器，而 Ada boost 結合 Rectangle Feature 可以從中取出一堆 Feature，對目標影像進行迭代，每次迭代都是改變 Rectangle Feature 取出的 Feature 的寬度來找尋人臉，同時因為訓練集的關西每次根據 hypotheses function(圖 4-8)取 error 值最小的部分即是人臉，另外在這個過程中每個 Rectangle Feature 所取出的 Feature 都代表一個權衡值，當傳入分類器的圖片在所有 Feature 投票的結果中大於一半的分數才會被認為是人臉，而每個 Feature 的權衡值是不等同的，最終我們就可以得到大量的 Cascade Classifier 並合成一個 Pipeline(圖 4-9)，越前面者辨識強度越弱，藉由每一級的過濾最終就可以得到畫面中人臉的範圍。

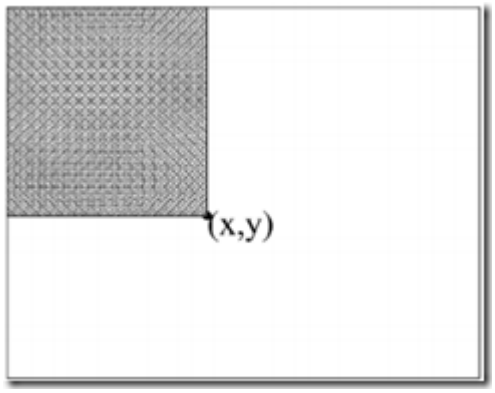


圖 4-6 Integral Image

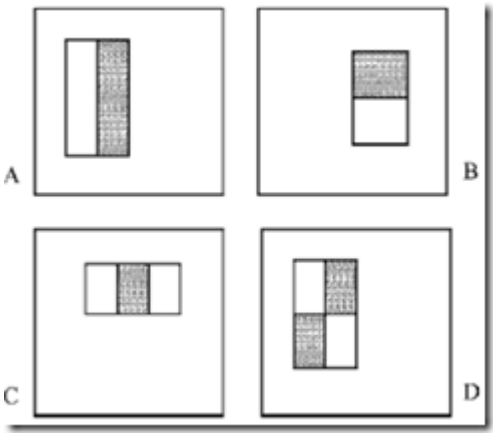


圖 4-7 Rectangle Feature

- 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
     $$\epsilon_t = \min_{f,p,\theta} \sum_i w_i \, |h(x_i, f, p, \theta) - y_i| \, .$$
     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 $\alpha_t = \log \frac{1}{\beta_t}$

圖 4-8 hypotheses function

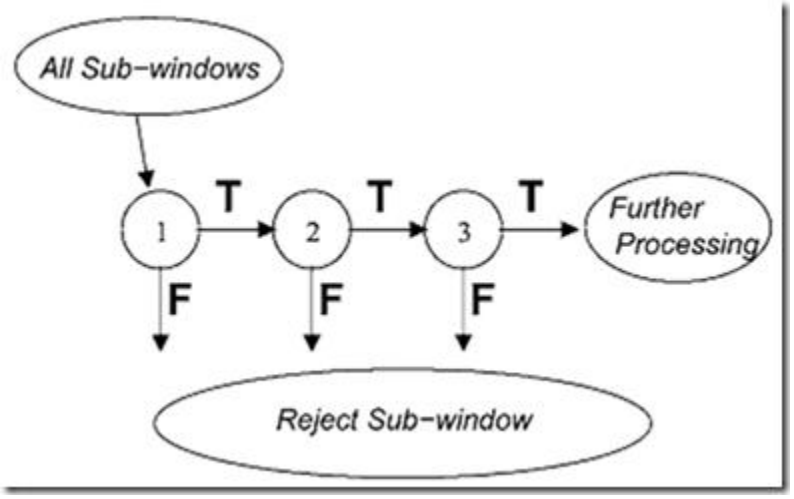


圖 4-9 Cascade Classifier

4.2.2 身分識別

當找出人臉後最後一個階段就是讓畫面跟登錄的人臉進行比對來確認身分了，因此我們需要一個可以比對圖片特徵並確認的演算法，本專題使用SURF(Speeded Up Robust Features)演算法來避開 SIFT(Scale-invariant feature transform)演算法專利的問題。

當我們得到人臉的範圍即感興趣區域(Region Of Interesting)後，再進行 SURF 的計算可以減少計算量，同時也可以去除不必要的部分提高辨識率，這就是為什麼我們要先進行人臉辨識才做身份識別。

SURF 與 SIFT 在演算法上的不同在於，SIFT 將圖片變換成各種不同的大小(尺度)與高斯函數進行運算得到高斯金字塔(Gaussian Pyramid)，而 SURF 利用 Integral Image 與 box filter 相乘可以得到近似於高斯金字塔的結果，且只變換 box filter 的大小而不是圖片(圖 4-10)，因此尺度空間與特徵點的建立速度比 SIFT 快，接著 SIFT 先進行非極大抑制(Non-Maximum Suppression)去除對比度低的點，再用海森矩陣(Hessian matrix)去除邊緣的點，而 SURF 利用海森矩陣決定點後才用非極大抑制來去除點，速度上也比 SIFT 快，接著 SIFT 利用方形區域的統計梯度建立直方圖幫特徵點建立方向，而一個特徵點可以有多個向，SURF 則是利用哈爾小波轉換(Wavelet transform)取圓形扇形面積來決定特徵點的方向(圖 4-11)，同時哈爾小波轉換也可以利用來確定區域面積，來對兩張圖進行特徵點的比對並找出相似的區域(圖 4-12)，最終我們可以利用預先登錄的人臉資料庫來進行使用者身份的核查，實驗(圖 4-13)可以看到我們利用資料庫的圖片對 Kinect 得到的影像來進行即時的人臉身份確認(圖 4-13)(圖 4-14)。

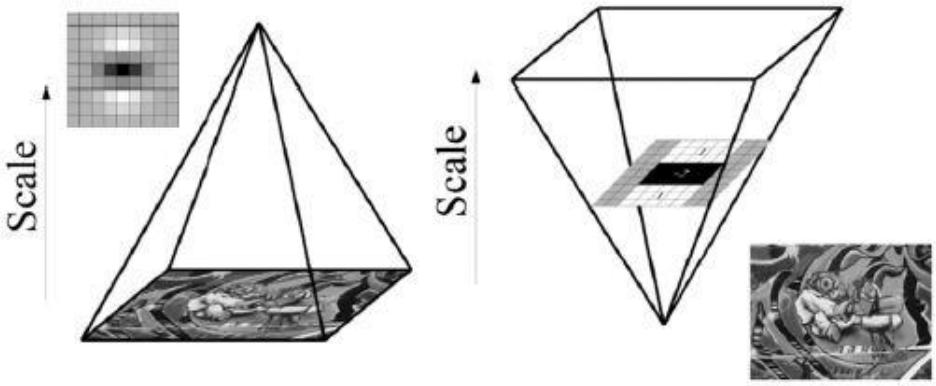


圖 4-10 尺度建立

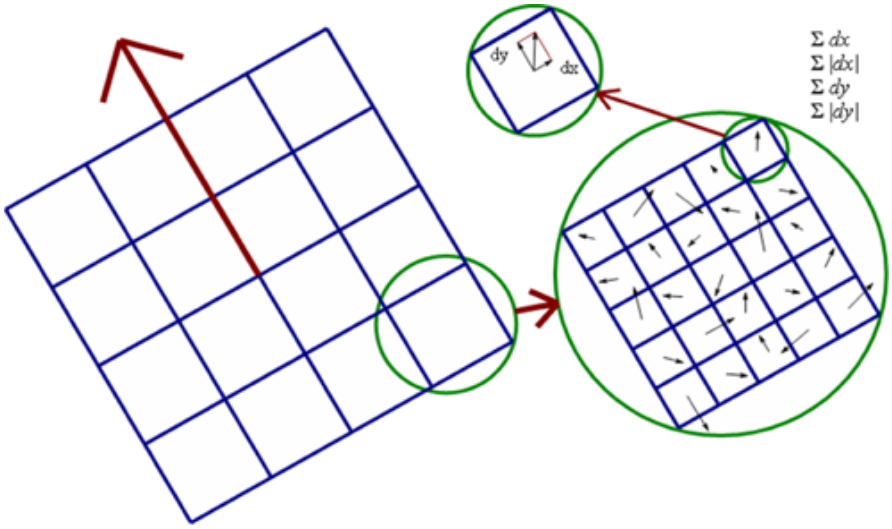


圖 4-11 特徵點方向

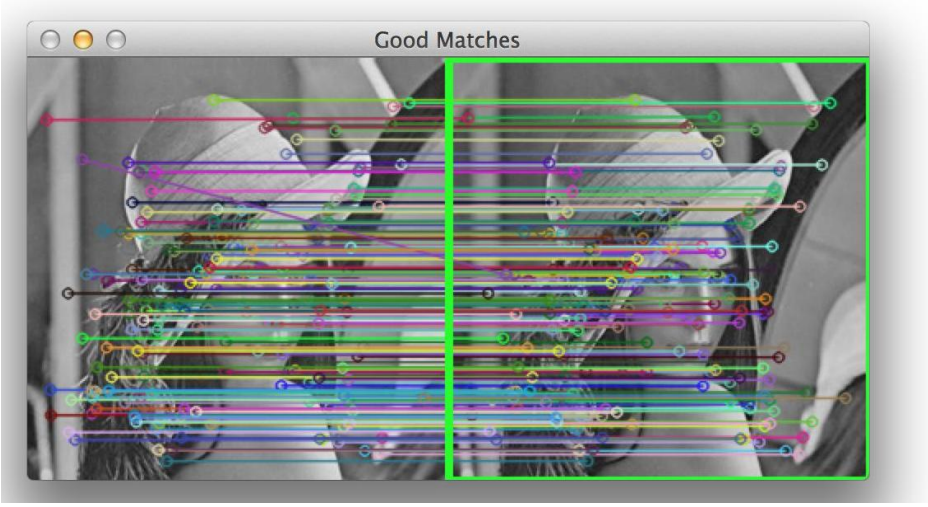


圖 4-12 相似區域

圖 4-13 身份識別-使用者 L

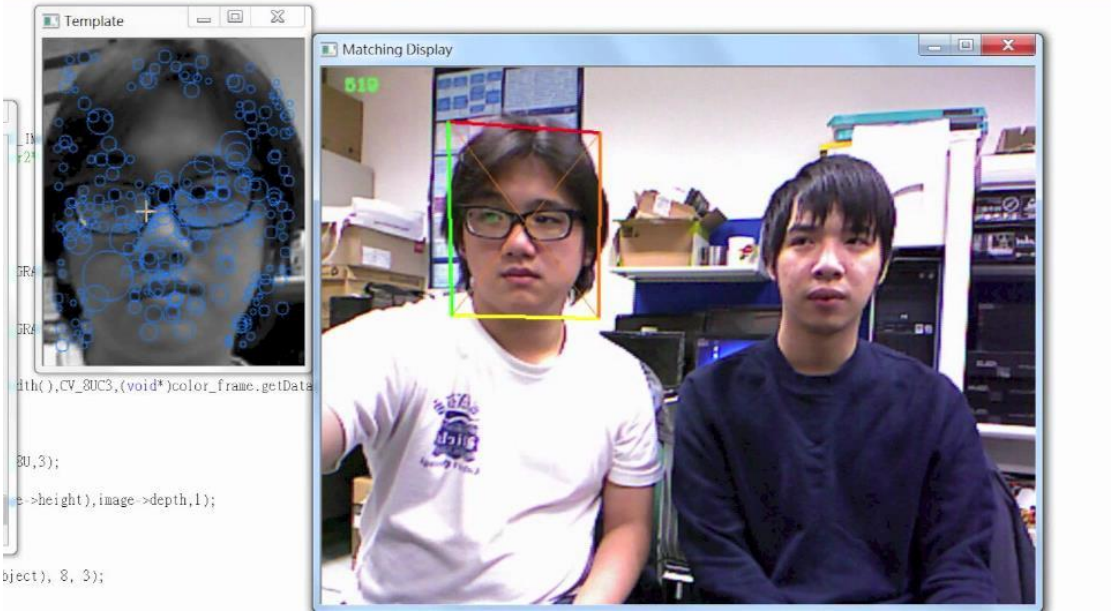


圖 4-14 身份識別-使用者 G

4.3 延伸骨架辨識空間

在畫面中心設立一個範圍(圖 4-15)，當身體骨架中心離開這個範圍時，就依照方向發送角度修正訊息給控制板，來轉動 Kinect 橫向的軸向。

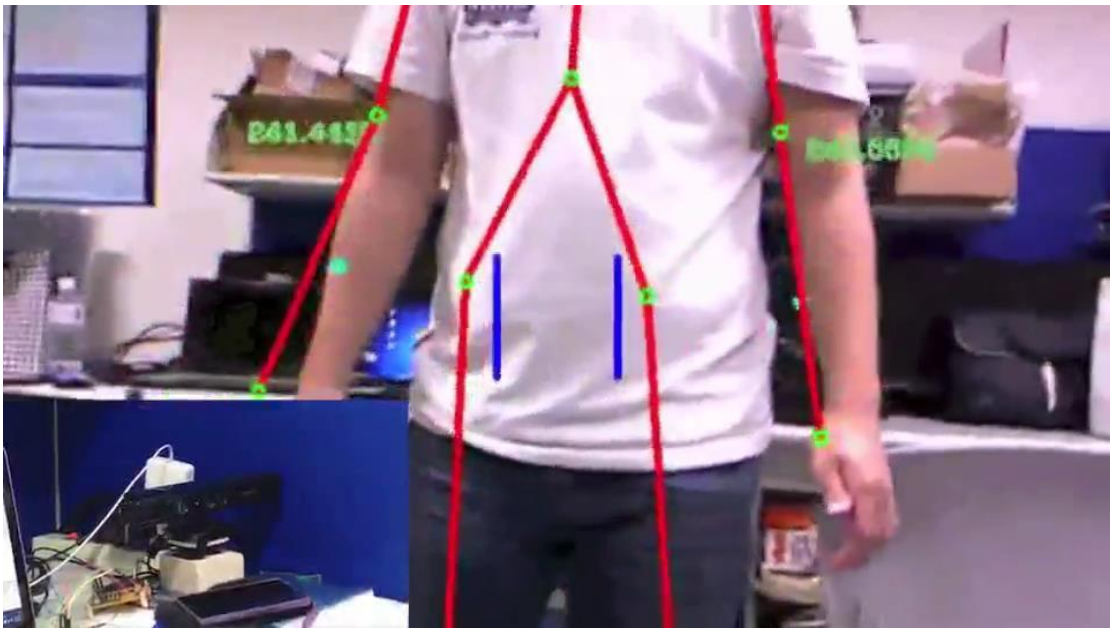


圖 4-15 畫面中兩條藍線就是中心範圍

5 控制端與機器人通訊（藍芽）

5.1 技術原理

本專題利用藍芽與機器人進行通訊，首先要先搜尋鄰近的藍芽裝置，取得目標藍芽裝置的位址以及名稱，這個功能可以由 BluetoothFindFirstDevice、BluetoothFindNextDevice 這兩個函數達成，這兩個函數搭配使用前者在第一次迭代週遭藍芽設備時使用，後者則是第一次之後的每次去使用直到沒有設備可以列舉，而正規的藍芽通訊是依照各種作業系統的 Socket API 來建立連接，Windows 下可經由 Socket 函數建立一個 Socket，接著再由 Connect 函數來進行連接，連接成功之後就可以用 Send 函數將資料送給對方，結束後再用 Closesocket 函數關閉 Socket 以及 WSACleanup 函數釋放 Socket 建立時所佔用的資源。

而機器人上的藍芽模組則是利用 UART 與伺服馬達控制板進行控制，事實上經過配對過後的藍芽設備在作業系統是以一個 UART 傳輸設備的形式存在，所以也可以利用 UART 相關的 Native API 去對藍芽進行操作。

5.2 實作

本專題主程式利用 BluetoothFindFirstDevice、BluetoothFindNextDevice(圖 5-1)找出所有可用設備並且包裝成一個對話盒(Dialog)提供使用者選擇要連結的設備(圖 5-2)。

```
53    m_bt_dev = BluetoothFindFirstDevice(&m_search_params, &m_device_info);
54
55    do{
56        //wxMessageBox(wxString(m_device_info.szName));
57        wxString name(m_device_info.szName);
58        paths.Add(name);
59        wxString addr;
60        addr.Printf(wxT("%02x%02x%02x%02x%02x%02x"),m_device_info.Address.rgBytes[5],m_device_info.Address.rgBytes[4],
61            m_device_info.Address.rgBytes[3], m_device_info.Address.rgBytes[2],
62            m_device_info.Address.rgBytes[1], m_device_info.Address.rgBytes[0]);
63        //wxMessageBox(addr);
64        bt_devices.insert(std::make_pair(name,addr));
65        if(!BluetoothFindNextDevice(m_bt_dev, &m_device_info)){
66            break;
67        }
68    }while(BluetoothFindNextDevice(m_bt_dev, &m_device_info));
```

圖 5-1 搜尋週遭藍芽設備程式碼

圖 5-2 搜尋週遭藍芽設備對話盒

接著將選擇的藍芽設備位址用來設定建立 Socket 時傳入的 SOCKADDR_BTH 結構，並且設定 AF_BTH 模式表是藍芽通訊(圖 5-3)。

```
606 void Frame::OnConnectBluetooth(wxCommandEvent &event)
607 {
608     BluetoothConnectArgsDialog dlg(this,wxID_ANY,wxT("Connect Bluetooth"),wxDefaultPosition,wxDefaultSize,wxDEFAULT_DIALOG_STYLE);
609
610     if(dlg.ShowModal() == wxID_OK){
611
612         if(socket != INVALID_SOCKET){
613             shutdown(socket,SD_SEND);
614             WSACleanup ();
615             closesocket(socket);
616         }
617
618         BT_ADDR aSddr;
619         std::istringstream iss(std::string(dlg.GetDeviceAddr(dlg.GetDevicePath()).mb_str()));
620         iss >> std::hex >> aSddr;
621
622         WSAStartup(MAKEWORD(2, 2), &wsadata);
623
624         socket = ::socket(AF_BTH,SOCK_STREAM,BTHPROTO_RFCOMM);
625         memset (&sockaddr_bth,0,sizeof(sockaddr_bth));
626         sockaddr_bth.addressFamily = AF_BTH;
627         sockaddr_bth.btAddr = aSddr;
628         sockaddr_bth.port = 1;
629
630         if(connect(socket,(SOCKADDR*)&sockaddr_bth,sizeof(sockaddr_bth)) != SOCKET_ERROR){
631             Send("BR\xff\x41\xff");
632         }
633         else{
634             wxMessageBox(wxT("can't connect"));
635             closesocket(socket);
636             WSACleanup();
637         }
638     }
639 }
640
```

圖 5-3 搜尋週遭藍芽設備對話盒

完成連線後就可以用 Send 函數將資料傳送到目的了(圖 5-4)。

```
695 void Frame::Send(const char *command)
696 {
697     if(socket != INVALID_SOCKET){
698         for(int i = 0;i < (int)strlen(command);++i){
699             send(socket,(const char *)&command[i],1, 0 );
700             Sleep(50);
701         }
702     }
703     //Sleep(200);
704 }
705
```

圖 5-4 搜尋週遭藍芽設備對話盒

6 伺服馬達控制板

6.1 製作原理

由於要幫 Kinect 新增橫向轉動修正畫面的機制，因此最好的方法是使用伺服馬達(圖 6-1)，控制伺服馬達必須使用 PWM（Pulse Width Modulation）來控制(圖 6-2)，藉由 PWM 的寬度來控制伺服馬達的角度，控制板使用的 MCU 是 ATmega328(圖 6-3)且在上面預先寫入 Arduino Bootloader，而控制板分為伺服馬達電流源與 MCU 電流源，以防止兩者共用同一電流導致馬達有時消耗太多電流或者有反向電動勢導致 MCU 損毀，控制板可由 UART（Universal Asynchronous Receiver/Transmitter）接收本專題主程式所傳送的修正角度資料，並且依照角度資料產生相對應的 PWM 寬度，經由 GPIO（General Purpose I/O）送至伺服馬達達到修正 Kinect 橫向方向的目的。

圖 6-1 伺服馬達

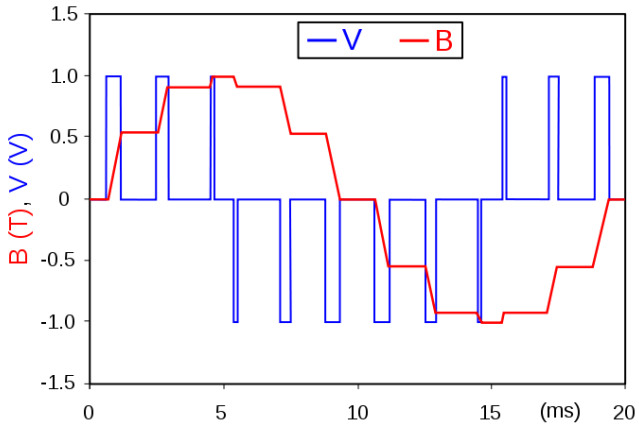


圖 6-2 PWM 波形圖

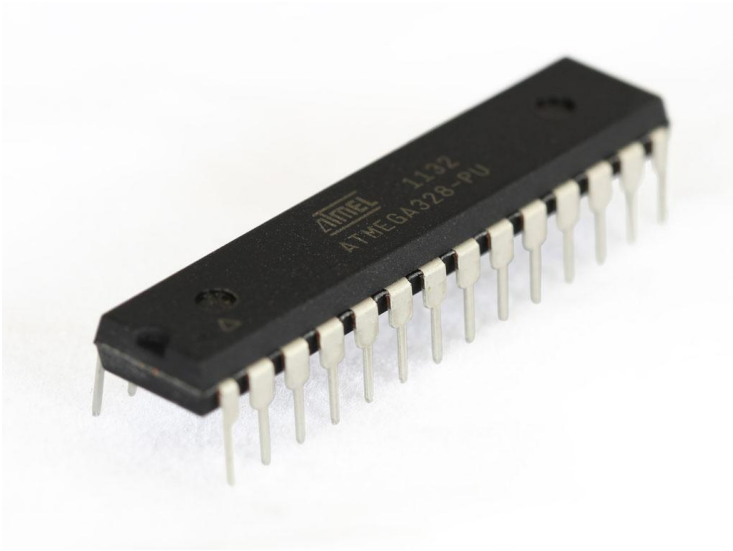


圖 6-3 ATmega328P-PU

6.2 PCB Layout

先看 ATmega328P-PU 的腳未定義(圖 6-4)，可以知道除了 VCC 電壓源要接上 5V 的電壓以及 GND 要接地外，AREF（Analog Reference）也要接上 5V 的電壓，要使 ATmega328P-PU 晶片運作還需要一個 16MHz 的石英震盪器(圖 6-5)，這個震盪器還要兩個陶瓷電容(圖 6-6)才可以起震，最後再將 Reset 這支腳串連一個 10k 歐姆電阻接上 VCC 使晶片進入運作模式，GPIO 直接直接拉出並且每個 GPIO 旁都有 VCC 與 GND 方便伺服馬達直接接上使用，最終的 Layout(圖 6-7)以及感光電路板成品(圖 6-8)如下。

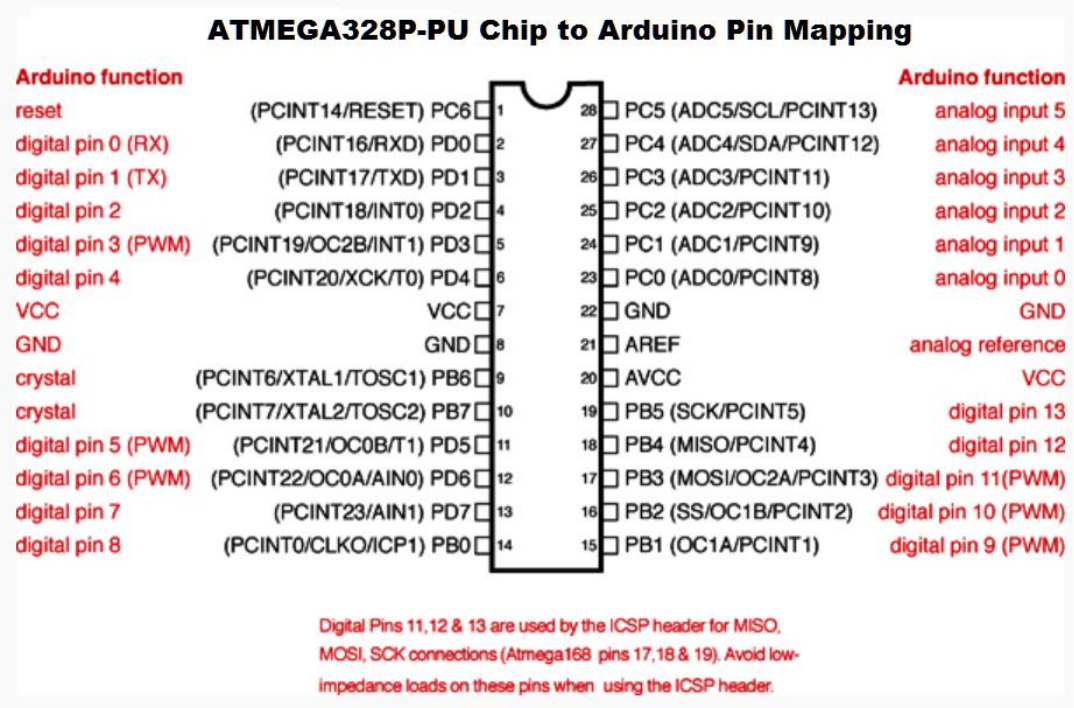


圖 6-4 ATmega328P-PU 腳位定義

圖 6-5 16MHz 石英震盪器

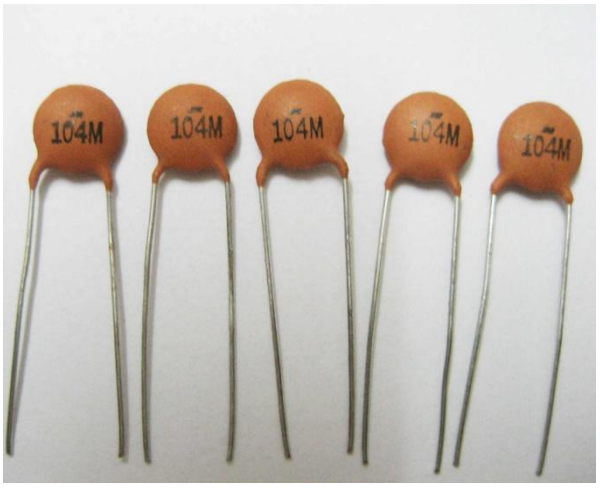


圖 6-6 陶瓷電容

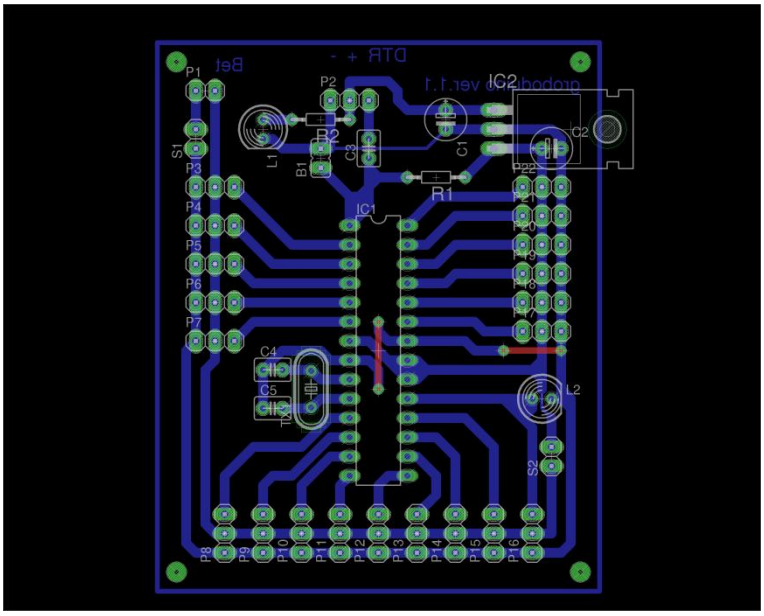
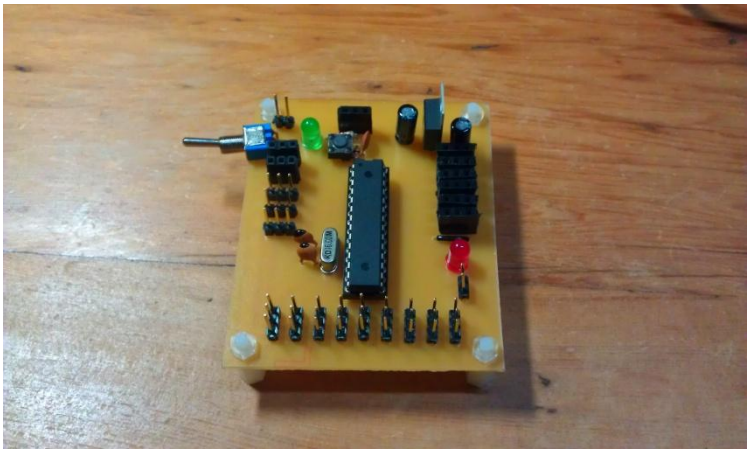


圖 6-7 成品 PCB Layout



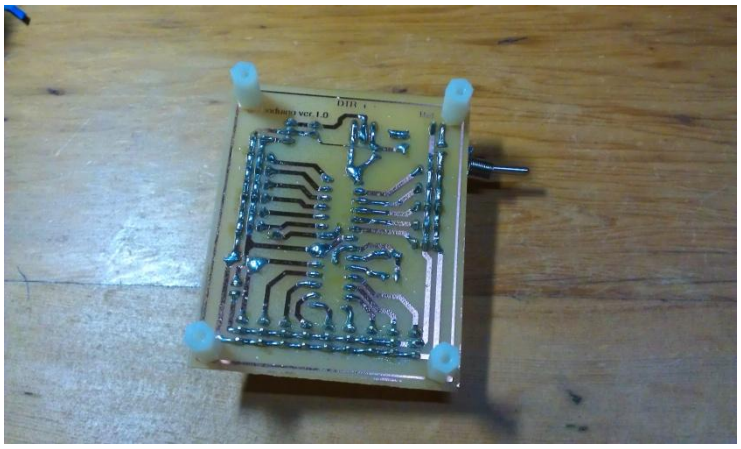


圖 6-8 感光電路板成品

6.3 接收與控制程式

控制程式只需要等待從 UART 得到的角度資料，接著對伺服馬達做出修正就可以控制 Kinect 的面向，最終程式碼如下(圖 6-9)。

而主程式透過 Windows Native API 即可與控制板進行連接，使用 CreateFile 可與傳入的 Serial Port 建立連接，同時由 BuildCommDCB 設定 Serial Port 的鮑率或校驗等等的參數，接著就由 WriteFile 送出資料給控制板。

```
#include <Servo.h>

const unsigned int MAX = 2;

Servo servos[MAX];

void setup() {
  Serial.begin(9600);

  for(int i = 0;i < MAX;++i){
    servos[i].attach(i + 3);
    servos[i].write(90);
  }
}

void loop() {

  for(int i = 0;Serial.available() > 0;++i){
    delay(1);     //速度小於1ms接收資料會錯誤
    int angle = Serial.read();
    servos[i].write(angle);
  }
}
```

圖 6-9 程式碼

7 動作控制指令 （有限狀態機）

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

圖 7-1 狀態映射表

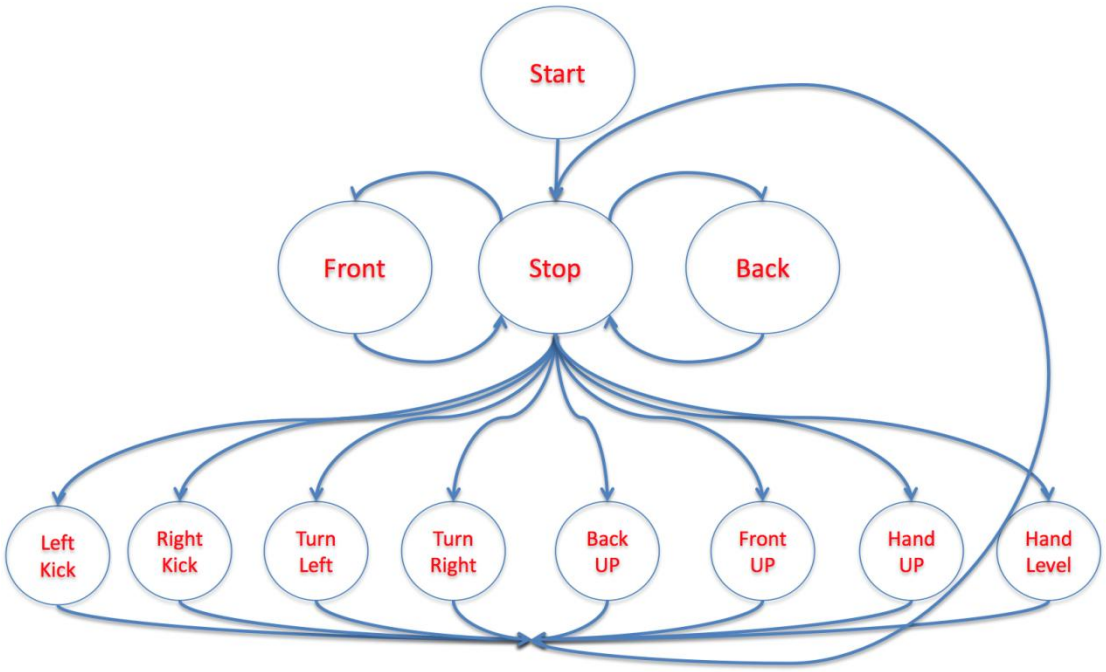

Start
Front
Stop
Back
Left Kick
Right Kick
Turn Left
Turn Right
Back UP
Front UP
Hand UP
Hand Level

圖 7-2 有限狀態機

8 機器人動作調整

本專題使用機器人原廠提供之動作編輯軟體。將每個所需要的動作分解成數個框架(frame)，逐個調整後再燒入機器人內建的記憶體中，透過指令使其做出已編輯的動作。

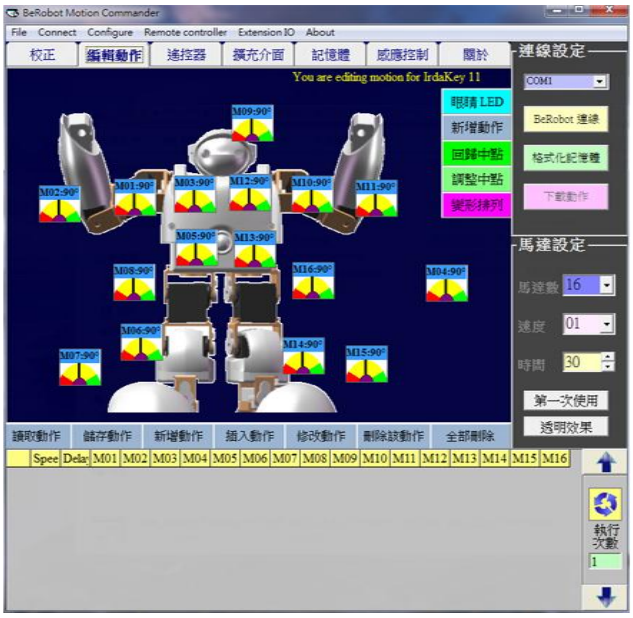


圖 8-1 機器人動做編輯軟體之畫面截圖(一)

圖 8-2 機器人動做編輯軟體之畫面截圖(二)

9 結論

9.1 問題討論

在本專題中一開始訓練出來的資料集合準確率很低，且會有 Jitter 的現象，導致能辨識度不到三成，但是我們修改骨架角度公式後，再以此得到的數據去訓練資料集合後，我們得到的結果就變為更加準確，使其準確率可達到九成以上。而本系統的展示機器人平台有一個我們無法修改的問題，就是由藍芽傳送資料給機器人的時候，機器人的控制板似乎是因為藍芽訊號干擾而會有特定幾個伺服馬達控制口送出不穩定訊號，干擾機器人穩定，本問題因為機器人控制板不是我們自行設計，且我們也沒有它的開發資料，因此只能藉由更換機器人平台來解決此問題。

9.2 工作分工

本系統的設計分為類神經網路、系統整合、機器人動作校調三部分，洪亮負責類神經網路的部分，主要內容為負責撰寫訓練資料集合程式，與負責撰寫辨識計算公式使用程式，以及研究與校調類神經網路參數，而郭承諺負責解決系統的整合與影像處理，內容包含主程式 UI 的撰寫以及撰寫 NoteBook 的藍芽連接機器人程式，以及設計所有影像處理相關的演算法，和伺服馬達控制板的製作與程式撰寫，最終再將所有程式整合起來成為程式主體，而機器人動作校調則由兩人一起合作。

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

9.3 心得

洪亮：

我覺得做專題帶給我的好處是讓我能接觸到以前沒碰過的東西。我一直很想鑽研關於人工智慧的東西，但是自己研究如果沒有明確的目的，就比較容易一直原地踏步，但是專題有了一個方向以後，就可以依照這個方向去研究。整體來說專題讓我學習到了很多東西，不只是自己學習東西，與組員互動之間也是讓我獲益良多，還有稍微了解到如何應對客戶(老師)的要求等。我覺得如果用心做專題的話，可以讓人有很大的改變。

郭承諺：

感謝我的 team 成員，負責主要演算法的洪亮、讓我們的報告與文件更嚴謹的指導老師孔崇旭、還有幫我們製作海報的學妹吳孟真，我認為 team 裡的每個人都是專業人士，沒有必要過度干涉對方，只要理解各自做的事是什麼即可，不可以有人自認官大學問大就可以對其他成員擅自出餿主意搗毀整個團隊，在這次專題中，洪亮對於整個系統的貢獻功不可沒，一開始題目方向確認時，針對姿態 pattern 識別的部分我給出可以往類神經網路與機器學習兩個方向去 research，洪亮參考不少資料後選擇了前者，並且研究了關鍵的演算法與變數，一開始將我程式採樣到的角度資料進行訓練後得到的結果進行識別成功率不到兩成，後來洪亮針對我的空間向量取得角度的公式進行改進後成功率可達九成五以上！且他還加入投影向量可以判別象限使得上或下得以判別，不用另外寫演算法判斷，這點是當初我沒想到的，最後在本次專題我認為技術只是實現目標的手段是次要的，最重要的還是目標的正確性，在 team work 不是要展現各自的能力達到什麼頂峰，而是如何能讓大家在正確的位置做正確的事發揮 100%的能力才是 team work。

9.4 未來展望

對於 UI 的發展由最早的命令列模式 CLI，再到圖形化的 GUI 後，由於各項技術的發展使得科幻小說中的以人體最直覺的控制方式 NUI 得以實現，在未來此類應用絕對會越來越多，尤其搭配機器人的控制可以，令原本只能由滑鼠或遙控器控制的機器人更加靈活，並且由於使用類神經網路學習人類的學習模式去辨識人體姿態，使此類系統的錯誤率降得更低，因此未來此類系統的發展必將盛行。

10參考資料

OpenNI：http://www.openni.org/

Kinect SDK：http://www.microsoft.com/en-us/kinectforwindows/

OpenCV：http://opencv.org/

RGB color model：
http://zh.wikipedia.org/wiki/%E4%B8%89%E5%8E%9F%E8%89%B2%E5%85%89%E6%A8%A1%E5%BC%8F

Robust Real-Time Face Detection：
http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf

SURF：http://en.wikipedia.org/wiki/SURF

SIFT：http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

Gaussian Pyramid：
http://zh.wikipedia.org/wiki/%E9%AB%98%E6%96%AF%E9%87%91%E5%AD%97%E5%A1%94

Hessian matrix：
http://zh.wikipedia.org/wiki/%E6%B5%B7%E6%A3%AE%E7%9F%A9%E9%98%B5

Wavelet transform：
http://zh.wikipedia.org/wiki/%E5%93%88%E7%88%BE%E5%B0%8F%E6%B3%A2%E8%BD%89%E6%8F%9B

Convolution：http://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF

影像濾波：
http://140.130.15.147/%E5%8D%8A%E5%B0%8E%E9%AB%94%E5%8F%8A%E5%85%89%E9%9B%BB/%E8%AA%B2%E7%A8%8B%E6%95%99%E6%9D%90/%E8%87%AA%E5%8B%95%E5%8C%96%E5%85%89%E5%AD%B8%E6%AA%A2%E6%B8%AC/ch05_%E5%BD%B1%E5%83%8F%E6%BF%BE%E6%B3%A2.pdf

維基百科-類神經網路：http://en.wikipedia.org/wiki/Artificial_neural_network

維基百科-倒傳遞演算法：http://en.wikipedia.org/wiki/Backpropagation

OpenNN ：http://www.intelnics.com/opennn

Kinect 体感机器人(三)-空间向量法计算关节角度：
http://blog.csdn.net/mdl13412/article/details/8025680

Kalman Filter：
http://zh.wikipedia.org/wiki/%E5%8D%A1%E5%B0%94%E6%9B%BC%E6%BB%A4%E6%B3%A2

Hidden Markov Model：
http://zh.wikipedia.org/wiki/%E9%9A%90%E9%A9%AC%E5%B0%94%E5%8F%AF%E5%A4%AB%E6%A8%A1%E5%9E%8B

OpenNI：http://www.openni.org/

Kinect SDK：http://www.microsoft.com/en-us/kinectforwindows/
OpenCV：http://opencv.org/
RGB color model：
http://zh.wikipedia.org/wiki/%E4%B8%89%E5%8E%9F%E8%89%B2%E5%85%89%E6%A8%A1%E5%BC%8F
Robust Real-Time Face Detection：
http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf
SURF：http://en.wikipedia.org/wiki/SURF
SIFT：http://en.wikipedia.org/wiki/Scale-invariant_feature_transform
Gaussian Pyramid：
http://zh.wikipedia.org/wiki/%E9%AB%98%E6%96%AF%E9%87%91%E5%AD%97%E5%A1%94
Hessian matrix：
http://zh.wikipedia.org/wiki/%E6%B5%B7%E6%A3%AE%E7%9F%A9%E9%98%B5
Wavelet transform：
http://zh.wikipedia.org/wiki/%E5%93%88%E7%88%BE%E5%B0%8F%E6%B3%A2%E8%BD%89%E6%8F%9B
Convolution：http://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF
影像濾波：
http://140.130.15.147/%E5%8D%8A%E5%B0%8E%E9%AB%94%E5%8F%8A%E5%85%89%E9%9B%BB/%E8%AA%B2%E7%A8%8B%E6%95%99%E6%9D%90/%E8%87%AA%E5%8B%95%E5%8C%96%E5%85%89%E5%AD%B8%E6%AA%A2%E6%B8%AC/ch05_%E5%BD%B1%E5%83%8F%E6%BF%BE%E6%B3%A2.pdf
藍芽規範：
http://zh.wikipedia.org/zh-tw/%E8%97%8D%E7%89%99%E8%A6%8F%E7%AF%84
MSDN Library：http://msdn.microsoft.com/zh-tw/library/ms123401.aspx
Arduino：http://arduino.cc/
Eagle PCB：http://www.cadsoftusa.com/
PWM WiKi：
http://zh.wikipedia.org/wiki/%E8%84%88%E8%A1%9D%E5%AF%AC%E5%BA%A6%E8%AA%BF%E8%AE%8A
UART WiKi：http://zh.wikipedia.org/wiki/UART
GPIO WiKi：http://zh.wikipedia.org/wiki/GPIO