逢甲大學

資訊工程學系

專題研究報告

人臉辨識之點名系統

指導教授：陳德生　老師

學生：吳東翰 D0001166

林煒軒 D0071764

黃懷鋐 D0071703

鄭孟宸 D0071688

中 華 民 國 一 零 三 年 十 一 月

目錄

圖目錄

表目錄

摘要

本專題主要是發展一套以人臉辨識技術為基礎之點名系統。內容主要是在.Net 相關程式語言下進行開發，並以 Open CV (Open Source Computer Vision Library)以及其外覆函式庫 Emgu CV 來進行人臉的辨識。在硬體實現方面與 Kinect 內建的攝影機進行結合。程式內容主要是使用了 Kinect for Windows SDK 來對整台機器進行操控。在實現步驟上，首先利用彩色攝影機拍攝出測試人員影像，並且顯示在視窗中之後，再將其轉換成深度影像，並將深度影像送入人臉辨識的演算法中計算出人臉位置，之後再加上特徵值來辨識每一張人臉深度影像的不同，當辨識成功時系統會進行語音合成並唸出被辨識者的英文姓名，最後再將結果匯入到點名系統之中，讓使用者可以看到目前的出缺席狀況。由實驗結果顯示本系統最佳訓練張數為 10 張，其辨識率可達 85%以上

格式化: 字型: (英文)Times New Roman

前言

傳統課堂上的點名方式耗時又容易有冒名頂替的現象發生，為了改善此一缺點，本專題研究試圖利用影像處理及人臉辨識的技術來取代傳統的點名方式。在 2010 年，微軟研究院(Microsoft Research)在家用主機 Xbox 360 上推出 Kinect for Xbox360，它具備了直接辨識人體骨架，關節位置及手勢等功能，在此同時微軟也提供了官方開發的 SDK 及 Windows 上的驅動程式讓有興趣針對 Kinect 特性來開發程式的人員使用。加上自己對於 Kinect 這台機器有著很大的好奇心，因為在大學求學過程中，並沒有去接觸過這方面的知識，所以就決定要以 Kinect 來進行專題的製作。在實現上我們首先嘗試在 Kinect 感應器來完成遊戲的製作當作練習，一開始為了熟悉這台機器就開始去尋找網路上許多的程式範例，例如：彩色攝影機、深度攝影機、骨架追蹤、手指辨識等許多不同種的功能，而且 Kinect 程式碼大多是以 C#來進行撰寫，所以為了熟悉這些資料我們也開始去學習 C#語言方面的知識以便可以完成此次的專題。而人臉辨識具備人員的不可取代性，因此本專題就利用此一套軟硬體去開發一款此一點名系統，為了達成此目標我們也開始研究 Open CV 來輔助我們完成這項工作，選擇 Open CV 原因是因為他可以免費使用，且雖然 Kinect For Windows SDK 提供了高階影像的骨架資訊與臉部追蹤功能，但這些並非是計算機視覺與影像處理的全部，藉由發展多年的 Open CV，可以補足 Kinect For Windows SDK 無法提供的功能。

第一章　開發環境

1.1 系統需求

- 作業系統：Windows 7(x86/x64)
- 硬體：
  - CPU：雙核 2.66GHz 以上
  - RAM：2GB 以上
  - 顯示卡：支援 DirectX9.0c 以上
  - Kinect 感應器
- 軟體：
  - Visual Studio 2010 或 Visual C# Express
  - .Net Framework 4.0
  - Kinect SDK for Windows
  - Microsoft Speech Platform Runtime v10.2 (x86 版)
  - Microsoft Speech Platform SDK v10.2 (x86 版)
  - Kinect for Windows Runtime Language Pack v0.9
  - Open Source Computer Vision Library(OpenCV)
  - EmguCV
  - Coding4Fun Kinect Toolkit

1.2 Kinect for Windows SDK v1.8

目前最新的 Kinect SDK 版本是 v2.0，但是因為它~~他~~必須在 Windows 8 的環境下執行，本次專題所選擇的作業系統是 Windows 7，所以選擇了 SDK v1.8 來進行開發。Kinect 的 SDK 為 Microsoft 提供用於非商業用途的使用，例如：研究、展示、開發測試、教學用途。Kinect SDK 主要功能有深度感應器(Depth Sensor)、彩色攝影機感應器(Color Camera Sensor)、骨架追蹤(Skelton Tracking)以及語音辨識(Speech Recognition)

Kinect 感應器安裝：

1. 確保 Kinect 傳感器沒有插入任何電腦上的 USB 端口。
2. 如果目前電腦中有先前版本的 Kinect SDK 請先所有關閉並移除，並跳到步驟 5。
3. 刪除其他 Kinect 感應器的驅動程式。
4. 如果您有安裝 Microsoft 服務器語音平台 10.2，卸載微軟語音服務器平台運行時和 SDK 組件既包括 x86 和 x64 位版本，加上微軟服務器的語音識別語言

Kinect 的語言包。

5. 關閉 Visual Studio。您必須在安裝 SDK 之前關閉 Visual Studio，然後安裝後重新啟動它並重新設定 SDK 所需要的環境變量
6. 從下載位置，雙擊 KinectSDK-V1.8-的 Setup.exe。這種單一的安裝程序適用於 32 位和 64 位 Windows。
7. 一旦 SDK 完成安裝成功，確保 Kinect 傳感器被連接至外部電源，再插上 Kinect 感應器插入電腦的 USB 端口。該驅動程序會自動加載。
8. 成功完成後可以在裝置管理員看到下面的圖案，如[圖 1.1]， Microsoft Kinect Audio Array Control 和 Microsoft Kinect Camera 的話，代表可以使用 Kinect 的攝影機和麥克風
9. Kinect 感應器現在應該正常工作。

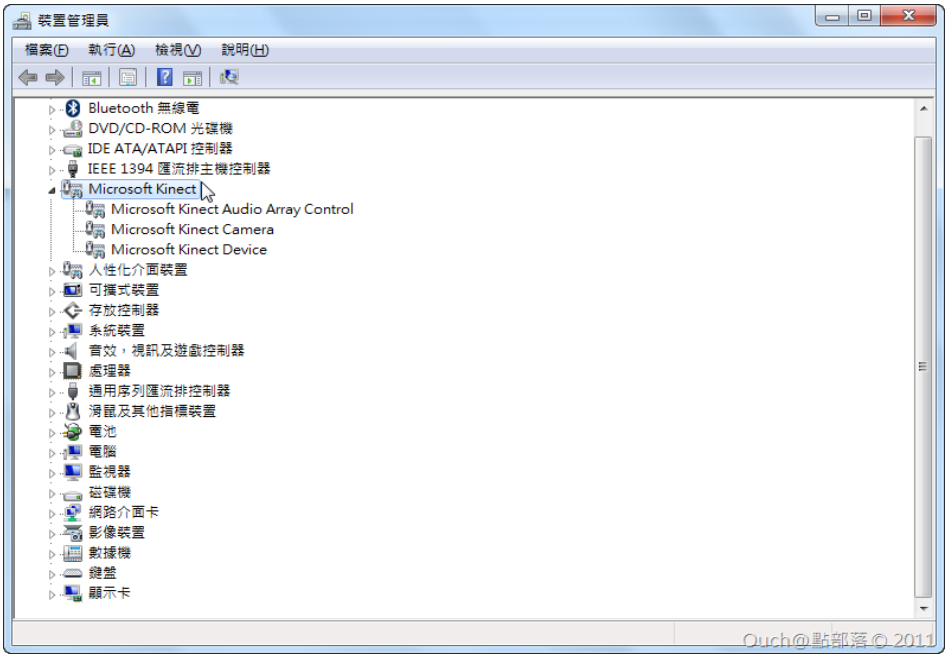

圖 1.1　裝置管理員

1.3 相關平台及技術

1.3.1 Micorsoft Visual Studio 2010

Microsoft Visual Studio(簡稱 VS)是美國微軟公司的開發工具套件系列產品[圖 1.2]。VS 是一個基本完整的開發工具集，它包括了整個軟體生命周期中所需要的大部分工具，如 UML 工具、代碼管控工具、整合式開發環境(IDE)等等。

所寫的目的碼適用於微軟支援的所有平台，包括 Microsoft Windows、Windows Phone、Windows CE、.Net Framework、.Net Compact Framework 等，而 Visual Studio .Net 是用於快速生成企業級 ASP.NET Web 應用程式和高效能桌面應用程式的工具。Visual Studio 包含基於元件的開發工具(如 Visual C#、Visual J#、Visual Basic 和 Visual C++)，以及許多用於簡化小組的解決方案的設計、開發和部屬的其他技術。

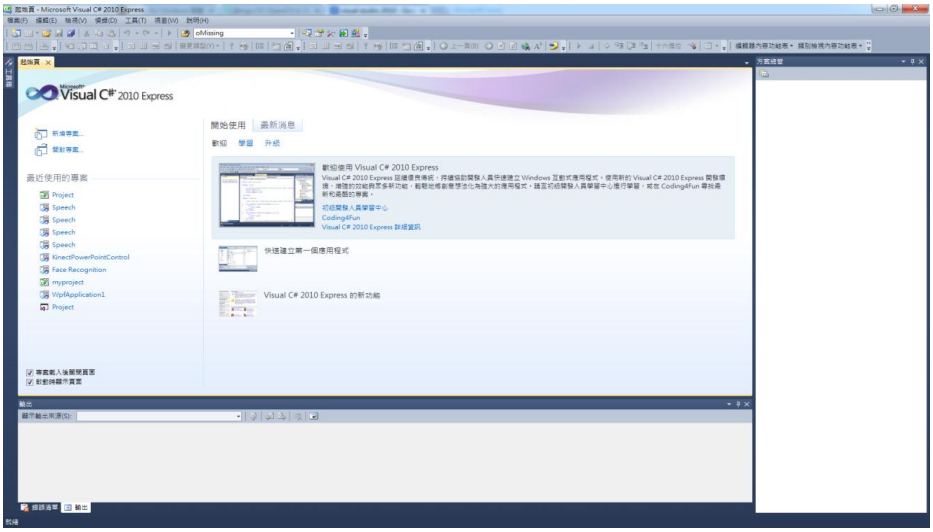

圖 1.2 Visual C#2010 介面圖

1.3.2 Open CV 及 Emgu CV

Open CV 全名 Open Source Computer Vision Library，由 Intel 發起，目前由 Window Garage(從事機器人用開放源碼業務,目前也銷售機器人設備)繼續維護。由於採用 BSD License 授權，所以在商業和研究領域皆可免費使用。Open CV 是一個跨平台的計算機視覺庫，通常用於即時影像的處理、計算機視覺(Computer Vision)以及模式識別。

雖然 Kinect for Windows SDK 提供了高階的骨架資訊以及臉部追蹤功能，但是這些並非計算機視覺與影像處理的全部，藉由發展多年的 Open CV，可以補足許多其無法完成的功能。舉凡人臉辨識、物體追蹤、動作辨識、人機互動、圖形分割、運動追蹤甚至是機器人的視覺領域都是 Open CV 的專精領域。

Open CV 主要是以 C 語言撰寫而成，Open CV 2.0 以後同時提供了 C 與 C++ 為介面的函式庫以供原生程式使用，針對目前興起的 C#、Python、Java 也都可以找到對應的函式庫來進行支援。，以本專題來說是以 C#進行開發，其外覆函

式庫就是 Emgu CV，可以看到 Emgu CV 叫用 Open CV 功能如[圖 1.3]。

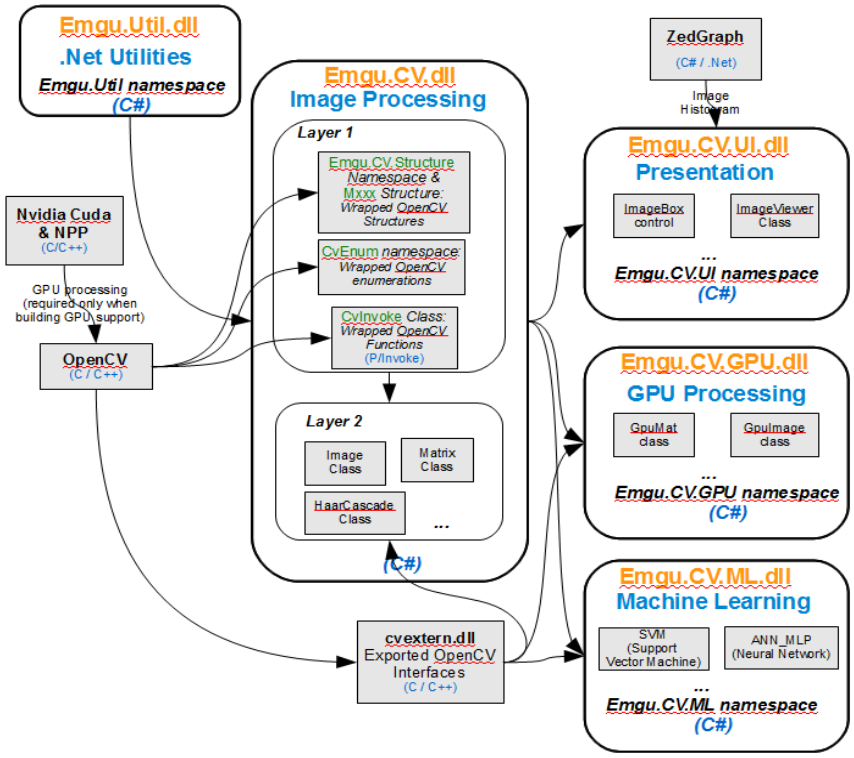

圖 1.3　Emgu CV 與 Open CV 關係圖

1.3.3 Open NI

Open NI 全名 Open Natural Interaction，由發展 Kinect 硬體技術的 Prime Sense 公司所發起，主要的目的是要為各種使用 Prime Sense 公司技術的硬體提供一致性的介面，其基本架構如[圖 1.4]，頂層的 Application 為程式部分，可以提供使用者自行撰寫，中間曾為 Open NI 部分，擔任軟硬體的銜接，以及可以做手勢辨識的處理，內部再包含中介軟體(Middle Ware)，以及下層的硬體驅動包含了語音、影像以及 Kinect，在微軟官方公布 Kinect SDK 之前，Open NI 為主流，與微軟官方 SDK 的最大差異在於 Open NI 可以橫跨 Mac OS、Linux 與 Windows 三個平台，使用 C/C++作為函式庫標準介面，而 Kinect for Windows 僅適用於 Windows 平台，但是如果就開發的便利性、輔助工具豐富性，Kinect for Windows SDK 略勝一籌。

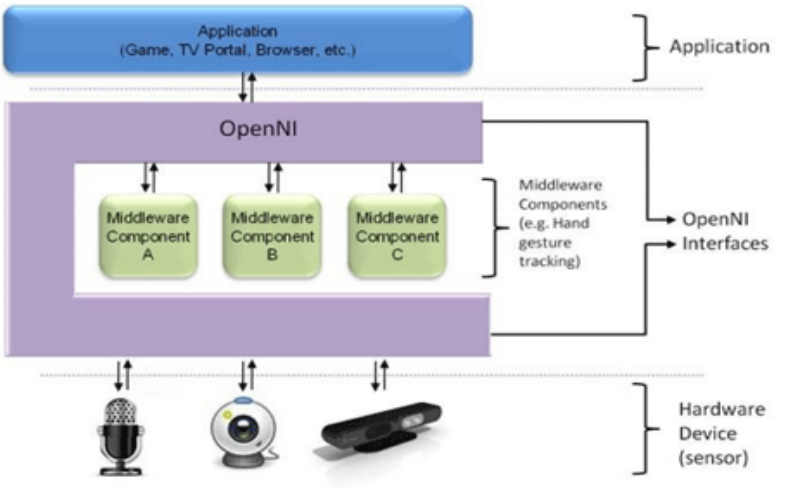

圖 1.4　Open NI 基本架構

第二章 Kinect 簡介

2.1 硬體介紹

2.1.1 Kinect 架構

圖 2.1　　Kinect 架構圖

如[圖 2.1]所示，左右兩邊的鏡頭取得 3D 影像深度，為紅外線及 CMOS 攝影機、中間為辨識出色彩的鏡頭、聲音則是透過 4 個陣列式的麥克風來取得。Kinect 本身也支援追焦的功能，底座馬達會隨著焦點人物而轉動 Kinect 的方向(上下各 27 度)，其臉部追焦功能會讓人以為 Kinect 可以水平轉動，但是實際上是不行的，此功能是利用數位變焦方式所達成的，在 PC 上透過臉部追蹤函式庫是可以做到一樣的效果得。

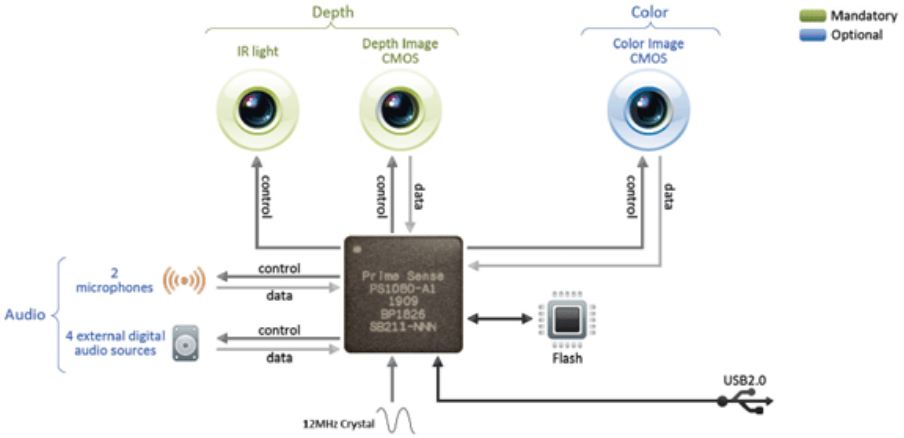

圖 2. 2 Kinect 架構摘要

由圖[2.2]可以得知 Kinect 是有三個鏡頭來辨識物體的，在最上方兩個鏡頭分別是紅外線攝影機以及 CMOS 攝影機，用在處理影像深度的。右上方的鏡頭為辨識出彩色影像的鏡頭。由紅外線射出光線照到物體後再反射回來的結果，交給晶片來進行運算，就可以輕易得知物體距離 Kinect 距離的遠近了。

2.1.2 原理(3D 深度影像/彩色影像/骨架追蹤/語音系統)

Kinect 的感應器可取得三種資訊:

彩色影像 (透過中間那顆 RGB 鏡頭) 、3D 深度影像 (透過左右兩顆鏡頭紅外線發射器和紅外線 CMOS 攝影機) 、聲音 (透過陣列式麥克風)。

3D 深度影像-Light Coding:

深度影像採用叫 Light coding 的技術，此技術是利用連續光（近紅外線）對測量空間進行編碼，也就是 kinect 感應器能感測到的範圍進行投射光線，經感應器讀取編碼的光線，交由 kinect 內建的晶片運算並解碼後，產生成一張具有深度的圖像，如[圖 2.3]。

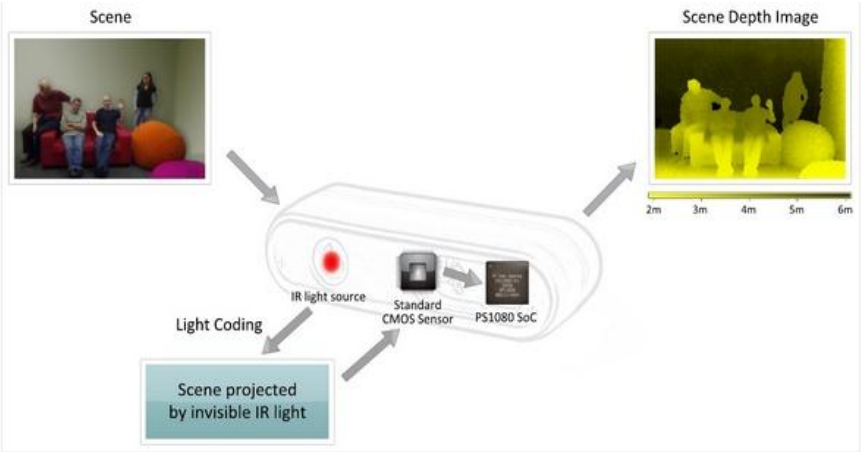

圖 2-3　深度影像處理流程

此技術關鍵是 Laser Speckle 雷射光散斑，當雷射光照射到粗糙物體、或是穿透毛玻璃(註 1)後，會形成隨機的反射斑點，稱之為散斑。散斑具有高度隨機性，也會隨著距離而變換圖案，空間中任何兩處的散斑都會是不同的圖案，等於是將整個空間加上了標記，所以任何物體進入該空間、以及移動時，都可確切紀錄物體的位置。Light Coding 發出雷射光對測量空間進行編碼，就是指產生散斑。

註 1:毛玻璃:一種表面不平整的玻璃，光線通過毛玻璃被反射後向四面八方射出去，產生漫反射，折射到視網膜上是不完整的像，於是就看不见玻璃背後的東西。

Kinect 就是以紅外線發出人眼看不見的雷射光，透過鏡頭前的 diffuser（光柵、擴散片）將雷射光均勻分佈投射在可感測到的空間中，再透過紅外線攝影機記錄下空間中的每個散斑，擷取原始資料後，再透過晶片計算成具有 3D 深度的圖像。

彩色影像:

kinect 中間的鏡頭是一般常見的 RGB 彩色攝影，用來錄製彩色圖像，鏡頭使用 CMOS 感光元件(註 2)，利用光學鏡頭將物體反射的光聚焦在 kinect 並將物體所反射的光轉換為數位訊號，壓縮後儲存於 kinect 上並可顯示圖像。

註 2:CMOS:是互補性金氧半導體，CMOS 主要是利用矽和鍺這兩種元素所做成的半導體，透過 CMOS 上帶負電和帶正電的電晶體來實現基本的功能，兩個互補效應所產生的電流即可被處理晶片記錄和解讀成影像。提供兩種品質的影像，一般品質與高品質，我們使用了一般品質來當做此實驗範例，如[圖 2.4]。

圖 2.4 彩色影像

- 一般品質

  感應器取得 1280x1024 影像，轉換成 RGB 模式並壓縮後，才傳給 NUI 執行時期，NUI 執行時期將收到的資料解壓縮，再送給應用程式，因為使用壓縮技術，所以提升了畫格傳輸的速度到 30 FPS，但是卻降低了影像品質。
- 高品質

感應器不壓縮影像，直接送給 NUI 執行時期，畫格速度只能到 15 FPS，而且 NUI 需要更大的緩衝區。同時也提供兩種影像格式，RGB 和 YUV，我們使用 RGB 當做此實驗範例。

- RGB
  - 32 位元、線性 X8R8G8B8 彩色點陣圖 (sRGB color space)
  - 應用程式在開啓串流時，需要在最後一個參數中指定 ImageType.Color 或 8 ImageType.ColorYuv 影像類型

- YUV
  - 16 位元、Gamma 校正的線性 UYVY 彩色點陣圖
  - 等同 RGB Gamma 校正
  - 使用較少的記憶體
  - 應用程式在開啓串流時，需要指定 ImageType.ColorYuvRaw
  - 只支援到 640x480 15FPS 影像

骨架追蹤:

透過CMOS紅外線感應器,將偵測到的3D深度圖像,轉換到骨架追蹤系統，採用分割策略(註 3)來將人體從背景環境中區分出來，感應器透過黑白光譜的方式來感知環境，純黑代表無窮遠，純白代表無窮近，如[圖 2.5]，接著我們把每個被追蹤的玩家在深度圖像中創建了所謂的分割遮罩(註4),分割化的玩家圖像，每一個像素都被傳送進一個辨別人體部位的機器學習系統(註5)中。

註3:分割策略:即雜亂的影像中提取出有用訊號。

註4:分割遮罩:一種將背景物體（比如椅子和寵物等）剔除後的景深圖像。

註5:機器學習系統:一人工智能被稱為 Exemplar，數以 TB 的數據被輸入到系統中來教會 Kinect 以畫素級技術來辨認手、腳以及它看到的其他身體部位。

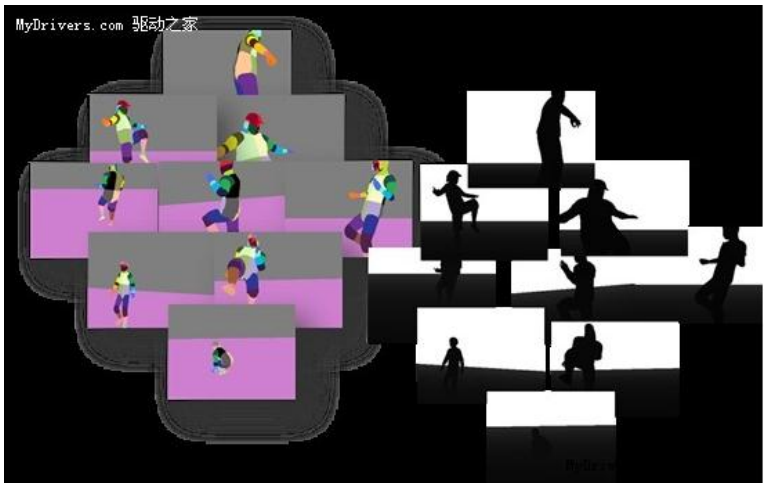

圖 2.5 灰階圖

隨後機器學習系統將給出某個特定像素屬於哪個身體部位的可能性。接著最後一步是使用 Exemplar 階段輸出的結果，根據追蹤到的 20 個關節點來生成一幅骨架系統，Kinect 會評估輸出的每一個可能的像素來確定關節點，在生成骨架系統時還做了一些附加輸出濾鏡來平滑輸出以及處理閉塞關節等特殊事件，如[圖 2.6]。

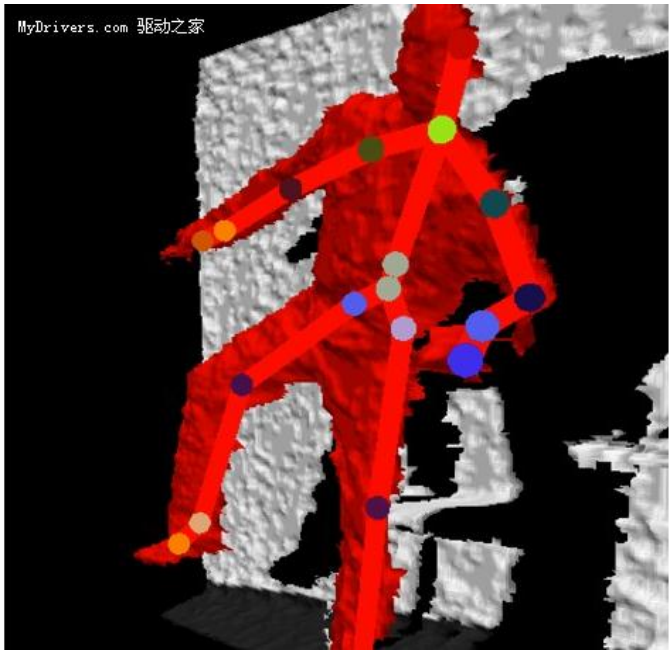

圖 2.6　骨架圖

骨架系統最多可同時偵測到 6 個人，包含同時辨識 2 個人的動作，感應範圍約 1.2-3.6 公尺；kinect 本身內建 200 多組常見人體姿勢，可用於配合「組成」使用者可能呈現的姿勢。每個人共可記錄 20 組細節，包含軀幹、四肢以及手指等都是追蹤的範圍，達成全身體感操作也用上機器學習技術(machine learning)，建立出龐大的圖像資料庫，形成智慧辨識能力，盡可能理解使用者的肢體動作所代表的涵義。

語音系統:

Kinect 內建四個麥克風的陣列、24 位元類比數位轉換(A/D)、雜音消除，口朝下的麥克風左邊一個，右邊三個，內含數位信號處理器可過濾背景雜音並強化聲音清晰度，可消除 20 分貝的環境雜音。聲音取樣頻率為 16 kHzZ，解析度 16 BITS，能辨識的水平聲音約正負 50 度，透過比對各個麥克風的聲音並傳入晶片達成語音辨別，辨別的文法需使用者輸入才可達成辨識。系統還包含一個叫 Beam Forming 的程式來配合鏡頭識別出使用者位置，能準確的把語言識別位置鎖定在使用者身上。我們使用微軟 microsoft Anna 當做語音合成的合成器，它可以說出英文。

2.1 規格

感應項目	有效範圍
顏色與深度	0.8~4 公尺(近距離模式 0.4~3 公尺)
視野角度	水平 57 度、垂直 43 度
底座馬達旋轉	上下各 27 度
每秒畫格	12、15、30 FPS(可由 API 改變)
深度解析度	VGA(640*480) QVGA(320*240)(80*60) 皆為 4:3
顏色解析度	XSGA(1280*960) VGA(640*480) 皆為 4:3
聲音格式	16KHz、32 位元 單聲道 PCM
聲音輸入	麥克風陣列 24 位元類比數位轉換 聲學回音消除(AEC) 自動增益功能(AGC) 抑制噪音

表 2.1　　硬體規格表

2.2 Kinect for Windows 架構

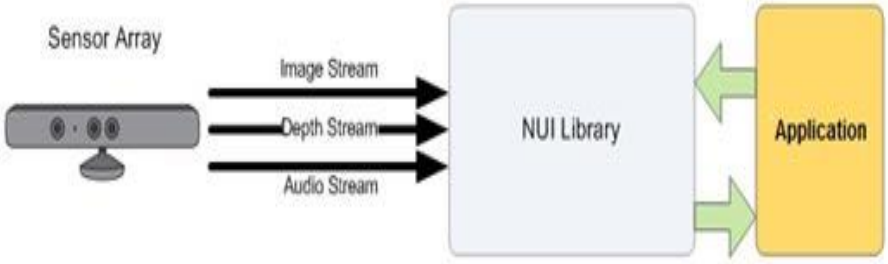

圖 2.7　Kinect For Windows 簡略架構

由[圖 2.7]可以得知，Kinect 應用程式皆透過 NUI Library 操控 Kinect 所有相關設定參數，再藉由 NUI Library 擷取 Kinect 感應器中的資訊包刮：彩色影像資訊(Stream，串流)、深度影像資訊(Stream，串流)、聲音資訊(Stream，串流) C++ 應用程式直接連結 NUI Library(動態連結函式庫)，.Net 應用程式則使用一個名為 Microsoft.Kinect.dll 的組件來對 NUI Library 進行了簡單的重組，處理了低階資料與.Net 資料型態的差異，讓.Net 能用純物件導向的方式與 Kinect 來進行互動。

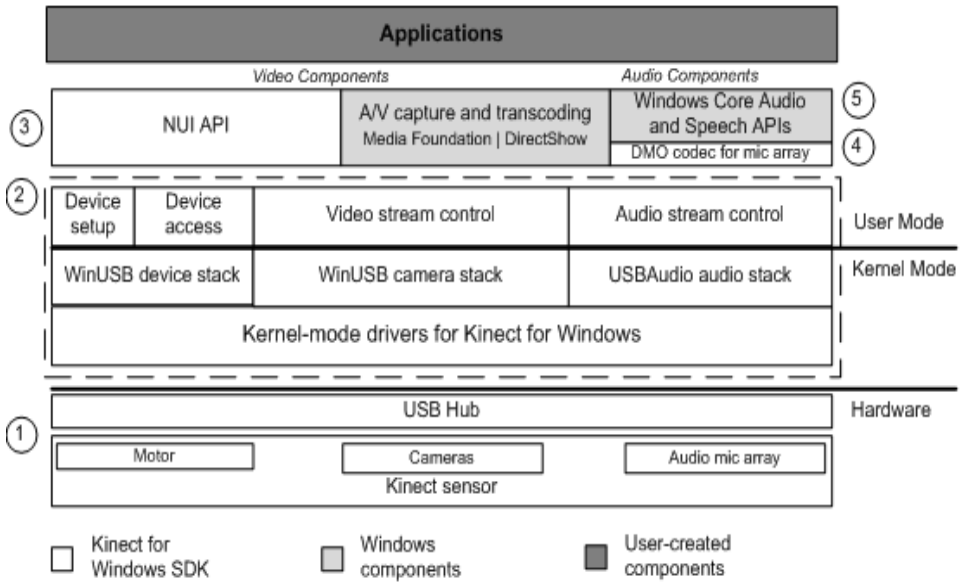

圖 2.8　Kinect For Windows 整體架構

由[圖 2.8]為 Kinect for Windows 整體架構，以下將分層說明

(1)Hardware：透過 USB 與 PC 連結，包含馬達(Motor)、照相機(Camera)、陣列式麥克風(Audio mic array)

(2)驅動程式：

User Mode：包含 4 種應用程式介面

Kernel Mode：Kinect SDK 中所安裝的四種驅動程式

(3)高階的 NUI Library

(4)DMO(DirectX Media Object)：為 DirectX 中的組成分子，被 Kinect 拿來接收聲音訊號與分析聲音來源之用，Windows 7 內建 DirectX，因此 DMO 本身就存在於我們的機器之中，並非 Kinect SDK 所安裝

(5)Windows 7 API：Windows 內建的 API，用來處理聲音，語音等這些與多媒體相關的應用，本身就內建於 Windows 7 之中，並非 Kinect SDK 所安裝

第三章 系統實作與開發

3.1 Kinect 實作

3.1.1 偵測 Kinect

檢查是否存在 Kinect 機器

```
private void DeActivateSensor()//偵測感應器是否存在
{
    if (kinectSensor != null)//如果不存在會使KinectSensor停止運作並釋放記憶體
    {
        kinectSensor.Stop();
        kinectSensor.ColorFrameReady -= new EventHandler<ColorImageFrameReadyEventArgs>(sensor_ColorFrameReady);
        kinectSensor.Dispose();
    }
}
```

圖 3.1 偵測 Kinect

作用：防止使用者按下開始按鈕時，卻沒有辨識機器導致程式的錯誤[圖 3.1]

將存在的感應器加入系統選項中

```
private void PopulateAvailableSensors()//將有的感應器加入
{
    cmbSensors.Items.Clear();

    foreach (KinectSensor sensor in KinectSensor.KinectSensors)//對於每一台Sensor進行偵測，如果存在就將其加入
    {
        cmbSensors.Items.Add(sensor.UniqueKinectId);
        cmbSensors.SelectedItem = sensor.UniqueKinectId;
    }
}
```

圖 3.2 加入感應到的裝置

作用：將所有可使用的 Kinect 感應器放入系統之中，以便使用者選取[圖 3.2]

3.1.2 啟動彩色攝影機

啟動 Kienct 彩色串流接收資料

```
private void SetupSensorVideoInput()//啟動Kinect：彩色串流
{
    if (kinectSensor != null)//檢查kinectSensor是否有值
    {
        imageFormat = (ColorImageFormat)cmbDisplayMode.SelectedItem;
        kinectSensor.ColorStream.Enable(imageFormat);//啟動彩色串流

        //當Kinect準備好時就會自動呼叫sensor_ColorFrameReady
        kinectSensor.ColorFrameReady += new EventHandler<ColorImageFrameReadyEventArgs>(sensor_ColorFrameReady);
        kinectSensor.Start();
    }
}
```

圖 3.3 初始化彩色影像

作用：當使用者按下開始按鈕時，系統會去開始檢視kinect_Sensor是否有值，並且開始啟動彩色影像的串流，且Kinect會自動呼叫sensor_Color_Frame_Ready來對影像進行顯示部分的處理[圖3.3]

接收資料並顯示於 WPF 的 Image 組件下

```
void sensor_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
{
    using (ColorImageFrame image = e.OpenColorImageFrame())//取得彩色串流裡面的東西
    {
        if (image == null)//無效影像跳開
            return;
        if (colorBytes == null ||
            colorBytes.Length != image.PixelDataLength)
        {
            colorBytes = new byte[image.PixelDataLength];//產生一個和影像大小相同的byte陣列
        }

        image.CopyPixelDataTo(colorBytes);

        BitmapSource source = BitmapSource.Create(image.Width,
            image.Height,
            96,
            96,
            PixelFormats.Bgr32,
            null,
            colorBytes,
            image.Width * image.BytesPerPixel);
        picVideoDisplay.Source = source;
    }
```

圖 3.4 顯示彩色影像

作用：首先我們先檢查所傳回的image影像是否有效，因為傳回的影像未必都可以使用，此做法主要是為了避免影像撕裂(Tearing)的現象，當確認為有效的影像之後，系統會接收到一個型別為ColorImage_Frame的影像，但是在WPF環境下想要顯示影像必須使用BitmapSource的格式，所以我們必須使用Bitmap_Source中的create方法來建立影像，Create方法的前兩個參數為影像的寬度與高度，第三個參數表示32 bits RGB格式，最後一個參數為Stride，定義為畫面每一行所佔的byte數[圖3.4]

3.1.3 啟動聲音串流

找尋系統所有語音辨識器

```
private static RecognizerInfo GetKinectRecognizer()
{
    Func<RecognizerInfo, bool> matchingFunc = r =>
    {
        string value;
        r.AdditionalInfo.TryGetValue("Kinect", out value);
        return "True".Equals(value, StringComparison.InvariantCultureIgnoreCase) &&
            "en-US".Equals(r.Culture.Name, StringComparison.InvariantCultureIgnoreCase);
    };
    return SpeechRecognitionEngine.InstalledRecognizers().Where(matchingFunc).FirstOrDefault();
}
```

圖 3.5 找尋系統所有語音辨識器

作用：主體為一個 foreach 的迴圈，利用 Spech_Recognition_Engine_-Installed Recognizers()找出系統內所影與音辨識器，接著過濾出符合我們需要的辨識器，Windows 7 內建為 Microsoft Anna[圖 3.5]

語音合成

```
static void VoiceResponse(string input)//語音合成
{
    SpeechSynthesizer synthesizer = new SpeechSynthesizer();//語音合成器

    synthesizer.Rate = 0;
    synthesizer.Volume = 100;
    synthesizer.Speak(input);
}
```

圖 3.6 語音合成

作用:建立SpeechSynthesizer物件後,會自動回傳預設的語音合成器供我們使用，最後 Rate 屬性表示發音速度，Volume 表示音量，最後的 Speak 函示就可以念出想要講的句子[圖 3.6]

3.2 臉部辨識實作

3.2.1 OpenCV 及 EmguCV 使用

安裝方法及使用步驟

1. 將 Opencv 及 Emgucv 安裝好

2.在電腦中加入環境變數,［圖 3.7］

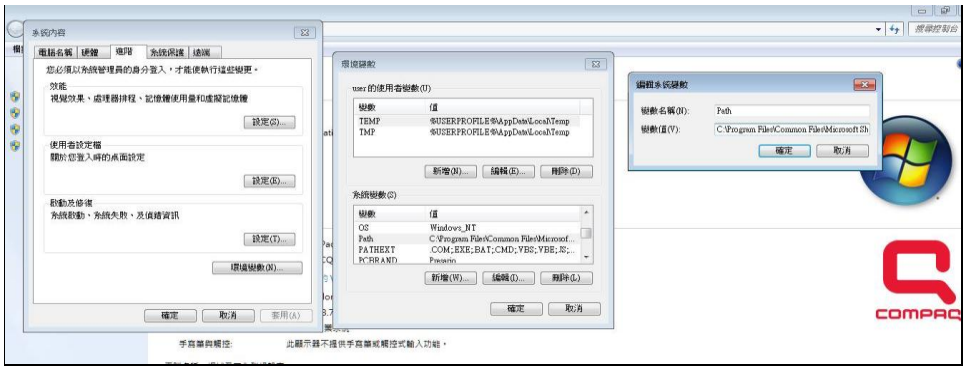

圖 3.7 環境變數設定

3.加入參考路徑

4.加入 EmguCV 組件,[圖 3.8]

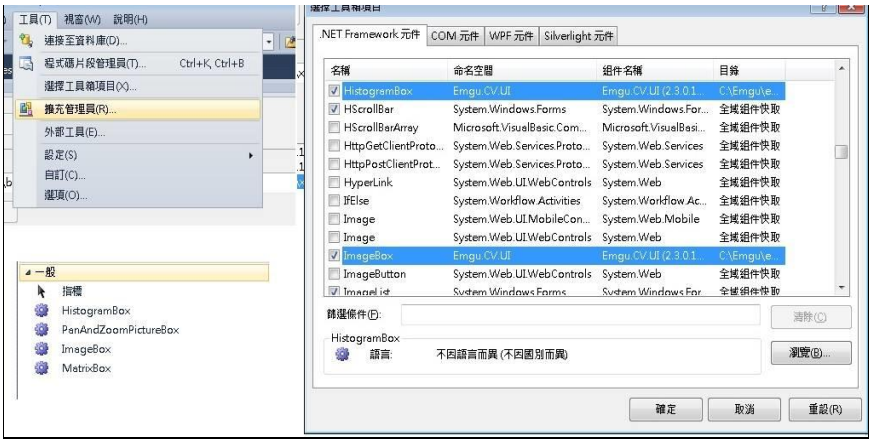

圖 3.8 加入組件

5.加入haarcascade_frontalface_alt2.xml(抓取臉部)

6.方案平台變更為 x86 或 x64(視電腦而定),[圖 3.9]

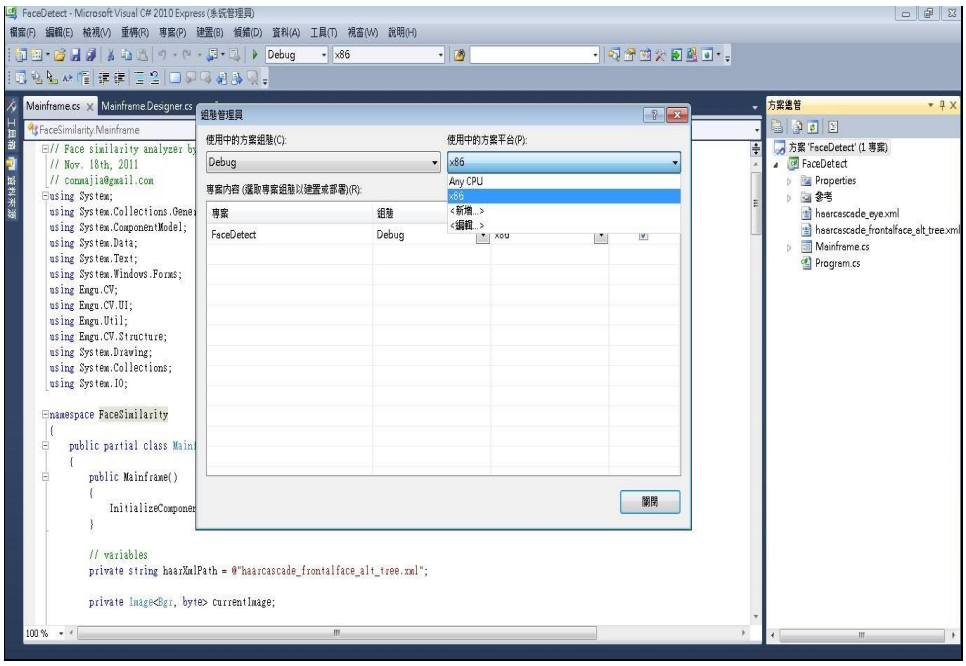

圖 3.9 變更平台

3.2.2 抓取臉部位置(使用 Haar)

・Opencv 使用內建的 Haar training 來訓練分類器(Classifiers)，而 Haar training 所使用的特徵就是 haar 特徵，經由以下幾個步驟訓練完成:

(1) 收集訓練樣本:由幾百個相同物件的正面影像來做訓練，之後將它們轉換成特定的大小。另外也訓練負面影像，負面影像使用任意影像即可，盡量跟辨識物體有關聯的東西。例如:要辨識的物體是車子，則正面影像應該就只含有車子的圖片，而負面影像應是包含機車、馬路、行人、交通號誌等等的圖片。

(2) 對所有正面影像做正規化

(3) 生成正面影像描述文件

(4) 創建正面影像的 vec 文件:由於 haar Training 訓練的時候需要輸入的正面影像是 vec 文件，所以需要使用 createsamples 來將正面影像轉換為 vec 文件。

(5) 創建負面影像描述文件

(6) 進行樣本的訓練:使用 haar Training 來完成訓練

(7) 生成 xml 文件:在進行 haar training 的時候，會在 data 目錄下生成一些目錄和 txt 文件檔，我們要將這些 txt 文件檔轉換為 xml 文件，也就是所謂的分類器。之後程式中就可以載入 xml 文件，如[圖 3.10]，也就是可以用人臉訓練集來進行臉部抓取和偵測的功能

(CascadeClassifier 是用來表示分類器的資料結構)

```
//Classifier
CascadeClassifier Face = new CascadeClassifier("C:/opencv/sources/data/haarcascades/haarcascade_frontalface_alt2.xml");
```

圖 3.10 載入 Xml 文件

之後宣告一個矩形陣列來儲存抓一個人臉的範圍大小，用迴圈來處理陣列，抓取人臉去除多餘的雜訊[圖 3.11]。

```
System.Drawing.Rectangle[] facesDetected = Face.DetectMultiScale(grayFrame, 1.2, 10, new System.Drawing.Size(50, 50), System.Drawing.Size.Empty);
```

圖 3.11 宣告抓取人臉陣列

3.2.3 運用特徵值來進行臉部辨識

影像轉換成灰階影像

```
Image<Gray, byte> grayFrame = currentImage.Convert<Gray, byte>();

System.Drawing.Rectangle[] facesDetected = Face.DetectMultiScale(grayFrame, 1.2, 10, new System.Drawing.Size(50, 50), System.Drawing.Size.Empty);

for (int i = 0; i < facesDetected.Length; i++)// (Rectangle face_found in facesDetected)
{
    //This will focus in on the face from the haar results its not perfect but it will remove a majorly
    //of the background noise

    facesDetected[i].X += (int)(facesDetected[i].Height * 0.15);
    facesDetected[i].Y += (int)(facesDetected[i].Width * 0.22);
    facesDetected[i].Height -= (int)(facesDetected[i].Height * 0.3);
    facesDetected[i].Width -= (int)(facesDetected[i].Width * 0.35);

    result = currentImage.Copy(facesDetected[i]).Convert<Gray, byte>().Resize(100, 100, Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
    result._EqualizeHist();
```

圖 3.12 轉換灰階圖與去除雜訊

將影像轉成灰階來偵測和辨識臉部位置，日後影像處理需要以灰階 image 來進行臉部偵測預測結果[圖 3.12]。

・臉部辨識演算法(Haar)
Opencv 裡有提供三個已經訓練好的分類器，分別是 hogcascades(Hog)、lbpcascades(lbp)、haarcascades(Haar)。

[1]Hoghogcascades:裡有提供順練好的 pedestrians 分類器，主要辨識行人和整個人體，人體必需要在完美的區域內辨識才會成功，太近或太遠會無法辨識，對於像是人臉辨識沒有提供例外的訓練集。

[2]Lbpcascades:有提供正臉和側臉的人臉辨識，lbp 特徵是用整數下去計算，所以速度上比 Haar 特徵演算法來的快許多，可是在辨識度上卻不是那麼精準(比 Haar 差 10%-20%)，常常抓取的範圍很奇怪。

[3]Haarcascades:有許多已經訓練好的訓練集，光是人臉的部份就有好幾個訓練集，另外還有針對眼睛、全身辨識等等，haar 對人臉的辨識度相對較其他特徵演算法高，所以此專題選擇使用 Haar 演算法來完成。
Haar 特徵也就是 Haar-like 特徵，是電腦影像處理常用的一種特徵描述算子，目前常用的特徵可分為三類:邊緣特徵(Edge features)、線性特徵(Line features)、中心特徵(Center-surround features)
如[圖 3.13]所示:

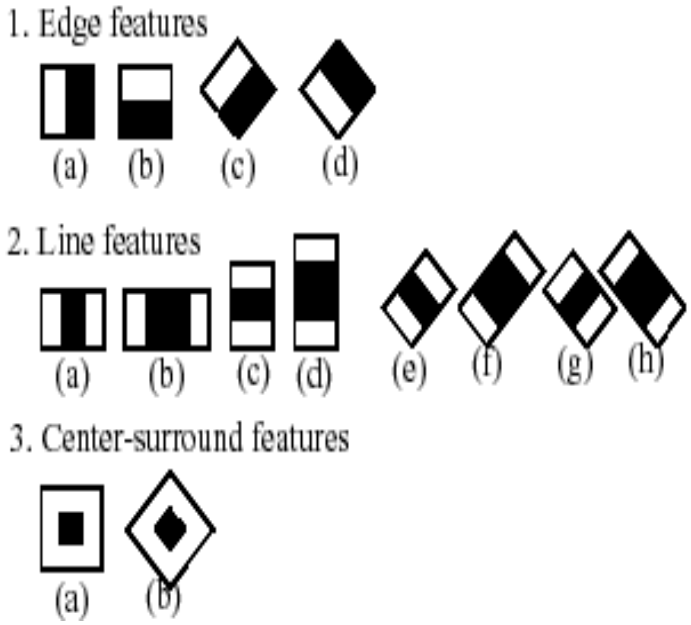

圖 3.13 特徵種類

每一種特徵值都是由黑色區域的像素值和與白色區域像數值和的差值計算而成，這個計算出來的差值就是 Haar 特徵的特徵值。

Haar 特徵優點:

1. 形狀簡單，座標容易描述

2. 對於邊緣和顏色變化大的地方比較敏感

3. 辨識度較高

3.2.4 儲存 Training 後的資料

```
private bool save_training_data(ImageSource face_data)...
```

圖3.14 儲存Training資料

作用:將偵測到並且正規化後的人臉送入save_training_data()[圖3.14]函式內儲存Training後的資料，儲存步驟大致如下:

```
while (file_create)//判斷亂數產生得檔案是否已經存在
{

    if (!File.Exists(AppDomain.CurrentDomain.BaseDirectory + "/TrainedFaces/" + facename))
    {
        file_create = false;
    }
    else
    {
        facename = "face_" + NAME_PERSON.Text + "_" + rand.Next().ToString() + ".jpg";
    }
}
```

圖3.15 圖片檔是否存在

•首先先判斷該圖檔的檔名是否已經存在資料夾內，直到檔名確定無重複才可跳出迴圈繼續執行[圖3.15]

```
if (Directory.Exists(AppDomain.CurrentDomain.BaseDirectory + "/TrainedFaces/"))//判斷TrainedFaces資料夾是否存在
{
    FaceImage.Save(AppDomain.CurrentDomain.BaseDirectory + "/TrainedFaces/" + facename, Coding4Fun.Kinect.Wpf.ImageFormat.Jpeg);
}
else
{
    Directory.CreateDirectory(AppDomain.CurrentDomain.BaseDirectory + "/TrainedFaces/");
    FaceImage.Save(AppDomain.CurrentDomain.BaseDirectory + "/TrainedFaces/" + facename, Coding4Fun.Kinect.Wpf.ImageFormat.Jpeg);
}
```

圖3.16 檢查儲存檔案的資料夾

・接下來要儲存圖片檔，先判斷該資料夾是否存在，如果不存在就create新資料夾，如果資料夾存在就直接儲存圖片[圖3.16]

```
FileStream FS_Face = File.OpenWrite(AppDomain.CurrentDomain.BaseDirectory + "/TrainedFaces/TrainedLabels.xml");
using (XmlWriter writer = XmlWriter.Create(FS_Face))
{
    writer.WriteStartDocument();
    writer.WriteStartElement("Faces_For_Training");

    writer.WriteStartElement("FACE");
    writer.WriteElementString("NAME", NAME_PERSON.Text);
    writer.WriteElementString("FILE", facename);
    writer.WriteEndElement();

    writer.WriteEndElement();
    writer.WriteEndDocument();
}
FS_Face.Close();
```

圖3.17 建立Xml

・接下來將圖片資訊存入xml檔(Face、Name、File)，這樣就將Training後的資料儲存起來了[圖3.17]。

3.3 點名系統實作

3.3.1 送出辨識結果

```
if (Eigen_Recog.IsTrained)
{
    string name = Eigen_Recog.Recognise(result);
    int match_value = (int)Eigen_Recog.Get_Eigen_Distance;

    //Draw the label for each face detected and recognized
    currentImage.Draw(name + " ", ref font, new System.Drawing.Point(facesDetected[i].X - 2, facesDetected[i].Y - 2),
                                                                    new Bgr(System.Drawing.Color.LightGreen));
    VoiceResponse(name);

    ADD_Face_Found(result, name, match_value);
}
```

圖 3.18 宣告物件型態

作法:一開始宣告Eigen_Recog為Classifier_Train的物件型態，
當KINECT偵測到人臉後，會執行上圖圈選部分進行人臉辨識[圖3.18]

```
public string Recognise(Image<Gray, byte> Input_image, int Eigen_Thresh = -1)
{
    if (_IsTrained)
    {
        FaceRecognizer.PredictionResult ER = recognizer.Predict(Input_image);

        if (ER.Label == -1)
        {
            Eigen_label = "Unknown";
            Eigen_Distance = 0;
            return Eigen_label;
        }
        else
        {
            Eigen_label = Names_List[ER.Label];
            Eigen_Distance = (float)ER.Distance;
            if (Eigen_Thresh > -1) Eigen_threshold = Eigen_Thresh;

            //Only use the post threshold rule if we are using an Eigen Recognizer
            //since Fisher and LBHP threshold set during the constructor will work correctly
            switch (Recognizer_Type)
            {
                case ("EMGU.CV.EigenFaceRecognizer"):
                    if (Eigen_Distance > Eigen_threshold) return Eigen_label;
                    else return "Unknown";
                case ("EMGU.CV.LBPHFaceRecognizer"):
                case ("EMGU.CV.FisherFaceRecognizer"):
                default:
                    return Eigen_label; //the threshold set in training controls unknowns
```

圖 3.19 人臉比對

接著使用到 Class Classifier_Train 中的 Method Recogniseg 如[圖 3.19]，對人臉做比對並回傳 Label。

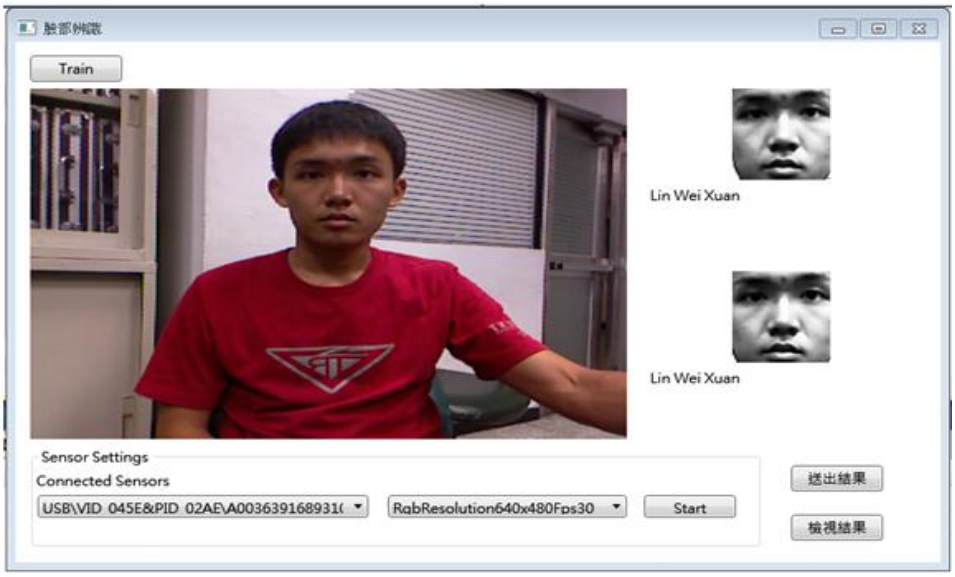

圖 3.20 辨識結果

最後，結果如[圖 3.20]將辨識到的人臉顯示在右方，並告知其人名，其中，被偵測者在未離開 kinect 的偵測範圍期間，系統會不斷抓取人臉。

3.3.2 讀取辨識結果

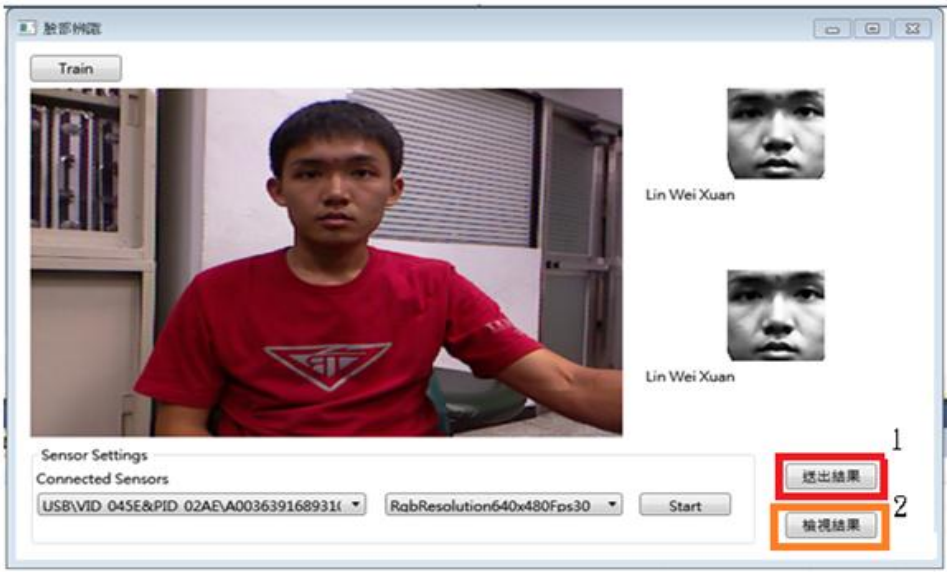

圖 3.21 操作流程圖

當右方顯示出偵測到的人臉及名字時表示已辨識成功，根據步驟[圖 3.21]按下送出結果，此時會執行[圖 3.22]程式碼，將辨識到的人臉的人名寫入 Name.txt

```
private void Print_Click(object sender, RoutedEventArgs e)
{
    System.IO.StreamWriter sw = new System.IO.StreamWriter("C:\\Users\\user\\Desktop\\myproject-wu\\myproject\\bin\\x64\\Name.txt"); //開啟一個檔案作寫入

    for (int i = 0; i < PersonName.Length; i++)
    {
        for (int j = i + 1; j < PersonName.Length; j++)
        {
            if (PersonName[i] != null && PersonName[j] != null)
            {
                if (PersonName[i].Equals(PersonName[j]))
                {
                    PersonName[i] = null;
                }
                else
                {
                }
            }
            else
            {
            }
        }
    }
    for (int i = 0; i < PersonName.Length; i++)
    {
        if (PersonName[i] != null && PersonName[i] != "Unknown")
        {
            sw.WriteLine(PersonName[i]); //寫入並換行, 如果不換行可以改用Write
        }
```

圖 3.22 寫入人名至 Name.txt

接著點選檢視結果，會執行以下程式碼[圖 3.23]

```
private void Result_Click(object sender, RoutedEventArgs e)
{
    ResultWPF MainToResult = new ResultWPF();
    MainToResult.Show();
}
```

```
public ResultWPF()
{
    InitializeComponent();

    StreamReader sr = new StreamReader(@"C:\Users\user\Desktop\myproject-wu\myproject\bin\x64\Name.txt");

    while (!sr.EndOfStream)
    {
        if (count == 1)
        {
            label1.Content = sr.ReadLine();
            checkBox1.IsChecked = true;
            image1.Source = new BitmapImage(new Uri("C:\\Users\\user\\Desktop\\myproject-wu\\myproject\\Image\\Green.jpg",
                                                                                    UriKind.Absolute));
            count++;
        }
```

圖 3.23 輸出結果

程式會從剛剛的 Name.txt 讀取使用者的名字，並對系統做勾選及更換圖片(由小紅人變小綠人)，最後結果會如[圖 3.24]

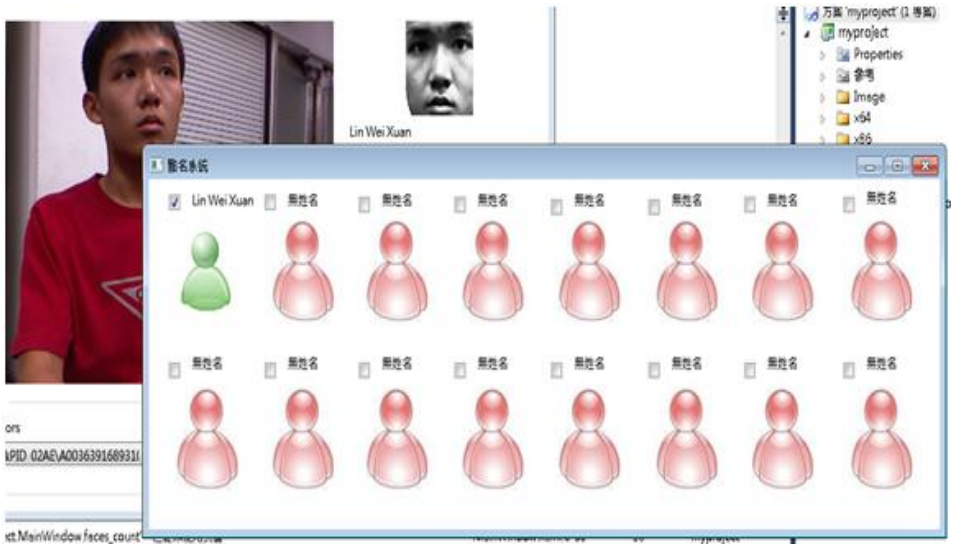

圖 3.24 顯示結果

3.4 分工

組員	工作內容
吳東翰	
林煒軒	
黃懷鋐	
鄭孟宸	

表 3.2 工作分配表

第四章總結

4.1 遭遇問題和解決辦法

1. C#學習:

一開始接觸專題製作時，有點摸不著頭緒，不清楚該以什麼做為題目，但幸好經過組員的幾經討論和老師建議下，我們選擇了人臉辨識做為我們的專題方案，在專題開工的前段，C#的運作是我遇到的第一個瓶頸，不熟悉語言及語法的情況加上繁複的資料收集，讓專題的進度在一開始前進的有些緩慢，但在經過幾天的摸索，漸漸的有抓到其中的訣竅之後，對於 C#的使用變得更加熟練

2.平台的不同導致效能大減:

起初 Winform 效能不佳，導致 kinect 接收影像經過辨識演算法後，整個畫面無法動彈，就算起動雙 buffer 問題依舊，之後改成 WPF 平台後有所改善。winform 使用早期開發出的 Graphics Device Interface(GDI)，wpf 則使用後來的 DirectX，winform 屬於較早期開發出的平台，有些像效能上的問題會有點難改善，則又開發出新的 WPF 平台，WPF 效能上比 winform 好，WPF 有多線程處理速度上會比 winform 的處理速度快

3.OpenCV 以及 EmguCV 設定上的問題:

剛開始接觸 opencv 和 emgucv，有很多地方都不了解，只知道有支援一些函式庫和訓練集等等，從網路上抓練習程式碼時也是難關重重，一堆參考錯誤或是缺東缺西的，之後查資料才知道 emgucv 要加入參考和設定 path，當這些都處理完畢之後，程式碼依舊無法動彈，之後才發現 C#還有選擇 x86 和 x64 等一些方案平台，換了一個適合電腦的平台才把問題解決。

4.Kinect 攝影機結束後未對資源進行釋放導致的問題:

從 mainwindow 按 start 之後再進入訓練視窗訓練，返回 maindow 要再次開啟攝影機辨識時會沒有動作，之後在要進入 train 視窗的函式裡把 kinect 停止和釋放支援，問題就解決了

5.不同平台之間導製程式碼不同的問題:

我們專題在截稿前的 3 個月前完成了初步的模型，但是卻因為平台導致效能不佳的問題，所以我們將整個程式碼移植到另一個平台上面，但是 A 平台所支援的程式碼 B 平台的語法一定是不相同的，為了找尋一些特定功能的函式，只能從官方得 MSDN 的函式庫下慢慢尋找或是上網請求他人的意見以及幫助，最後才順利將程式碼完整的轉到另一平台上

6.電腦無法處理 kinect 回傳的影像:

在 kinect 回傳影像格式分為 RGB 及 YUV 兩種，我們所選用的 RGB 格式在 640x480 可以到 30FPS、在 1280x960 可以到 12FPS，當我們選擇以 640x480 執行

時,影像無法顯示,甚至連程式也完全當機,後來意識到可能是電腦硬體的不足，來不及處理 30FPS 的回傳，之後，改以較好的電腦硬體設備操作，程式就順利執行了，但是，為了能讓程式在原本的電腦上執行，我們又想到可以從回傳的影像張數下手，將原本的每秒 30 幅改以每秒 5 幅來執行，最後，雖然程式也順利地執行，但是顯示出來的影像有延遲的問題。

訓練張數 / 測試次數	1 張	5 張	10 張	20 張
1	66.7%	33.3%	66.7%	66.7%
5	33.3%	66.7%	100%	73.3%
10	33.3%	76.7%	76.7%	63.3%
20	36.7%	75%	78.3%	81.7%
辨識率	34.4%	72.8%	85%	72.8%

4.2 未來展望

人臉辨識系統研究和應用,最主要的兩個目標就是達到低辨識時間和高辨識率，輸入影像皆已確定為人臉,所以免除了很多前置處理的部分,加上我們專題皆使用灰階人臉影像,所以不用考慮光源強弱和顏色不同等因素所造成的影響,隨著訓練資料庫影像的增加,辨識率就會跟著上升。由於專題現階段是雛形,有許多的限制,因為演算法需要大量的計算,以及電腦硬體配備效能不足,因此辨識速度和精準度,還需要大幅的改善,希望在未來可以改善這些問題。

經過一年的研究，我們的專題:人臉辨識之點名系統，終於告了一段落，我們期望在未來可以透過此一研究,進一步做成門禁系統,但是在即時辨識上無法快速辨識,以及用紙張或照片的人臉就可以通過辨識,這些都是目前所遇到的困難，所以在人臉辨識上還有很長的一段路要走，但是卻也是吸引大家的原因。

第五章參考文獻

5.1 參考書籍

Kinect 體感設計入門，王森著
3D 視覺專題製作：Kinect、Processing、Arduino 及 MakerBot ，蔣大偉著
Kinect 人機互動體感探索終極體驗：同場加映 微軟菁英大挑戰得獎專題剖析 ， 吳國斌、李斌、閻驥洲著

5.2 參考網站

曹祖聖系列文章
http://msdn.microsoft.com/zh-tw/evalcenter/hh367958.aspx

Sugizo 系列文章
http://www.techbang.com/posts/2936-get-to-know-how-it-works-kinect

Ron Forbes, Arjun Dayal 著、阿信 gg 譯
http://jgospel.net/news/tech/%E5%BE%AE%E8%BB%9F%E5%AE%98%E6%96%B9%E6%8F%AD%E7%A7%98kinect%E5%B7%A5%E4%BD%9C%E5%8E%9F%E7%90%86._gc28095.aspx

ChenLee_1 系列文章(Haar 特徵)
http://blog.csdn.net/carson2005/article/details/8094699

Internet 系列文章
http://fanli7.net/a/bianchengyuyan/C__/20121112/251841.html

維基百科
http://zh.wikipedia.org/wiki/%E5%93%88%E5%B0%94%E7%89%B9%E5%BE%81

秉程系列文章
http://charlesbc.blogspot.tw/2013/09/anycpu-x86-x64-arm.html