國立交通大學

電機與控制工程學系

碩士論文

TFT-LCD 視覺檢測之伺服定位系統研製

Implementation and Design of Servo Orientation System of TFT-LCD Vision Inspection

研究生：林俊杰

指導教授：林錫寬 博士

中華民國九十四年六月

誌謝

首先，我想感謝我的指導老師林錫寬教授，在研究所這兩年的生涯中，老師給了我很多的意見與指導。此外，老師在學術研究上所抱持的嚴謹態度、豐富的學識一直是我深感值得效法的典範。

其次，非常感謝廖德誠教授、陳傳生教授和蔡清元教授，在百忙之中來幫我進行論文口試，也感謝各位老師對本論文的建議與指正，以及對我個人的勉勵。感謝博士班王世杰學長、方志行學長與李宗原學長以及邱俊傑、謝孟勳、張豪揚和張維娜幾位學長在我研究過程中對我的指導與建議，並感謝我的同窗好友存堯、威勳、啓昌、宇中，與學弟品齊、星宇、典偉、匯欽，陪伴我在實驗室做研究的日子中，給我的鼓勵和支持，使得我在研究所這兩年獲益良多。

最後，我更要感謝我的家人，他們在這段時間內不曾間斷的鼓勵和關懷，使得我能順利完成論文的撰寫。在此僅以本份論文的結果獻給我的家人與其他關心、幫助我師長及朋友，非常的感謝你們。

TFT-LCD 視覺檢測之伺服定位系統研製

Implementation and Design of Servo Orientation System of TFT-LCD Vision Inspection

研究生　：　林俊杰　　Student　：　Jiun-Jie Lin

指導教授　：　林錫寬 博士　　Advisor　：　Dr. Shir-Kuan Lin

國立交通大學

電機與控制工程學系

碩士論文

A Thesis

Submitted to Department of

Electrical and Control Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

TFT-LCD 視覺檢測之伺服定位系統研製

研究生：林俊杰　　指導教授 : 林錫寬 博士
國立交通大學電機與控制工程學系

摘要

本論文主要目的是以 Visual C++ 作爲軟體開發工具，利用市面上所販售的運動控制卡，設計一套以廠商的 TFTD 機台 (三軸微步進馬達平台) 爲受控端的運動控制系統。

使用者在 PC 端透過視窗化的控制介面，將下達的運動指令傳送至 PCI-8134 運動控制卡，運動控制卡再將運動指令轉成脈波形式輸送至馬達驅動器，而驅動器在將脈波訊號轉成電壓或電流形式控制三軸的步進馬達。控制介面分成四部份 : 第一部分的功能是控制速度和位移量；第二部份的功能是監控平台目前的狀態；第三部份的功能是將面板切割成九宮格，根據使用者所點選的區域做移動控制；第四部份的功能則是在該區域以弓字形的方式作移動控制。

此運動控制系統。提供異於操作的視覺化控制介面，能及時精準的控制運動平台，並隨時監控平台的所在位置與回授狀態，對平台運動極限的發生，做出適當的保護措施，避免使用者不當的操作，而導致機台毀損甚至發生意外，更能針對業界的需求，規劃運動控制的路徑。

Implementation and Design of Servo Orientation System of TFT-LCD Vision Inspection

Student : Jiun-Jie Lin Advisor : Dr. Shir-Kuan Lin
Department of Electrical and Control Engineering
National Chiao Tung University

ABSTRACT

This main purpose of thesis is using Visual C ++ as the software developing tool , utilizing the motion control board sold on the market, to design a set of axis controlled motion control system for a manufacturer's TFTD machine(three axes micro-stepper motor platform)

Users can send motion instruction through the windowing control interface at a PC terminal to the PCI-8134 motion control card which translates the motion instruction into pulse signal and then sends it to motor driver.Furthermore, the motor driver transfers the pulse into voltage or current type to control a three axes micro-stepper motor. The control interface can be divided into four parts: the first part is used to control the speed,and displacement ; the second is used to monitor the status of platform ; the third is to provide a command the panel with nine sections,according to user-selected area to control the movement ; the fourth is to move and control by way of bow font in this area.

This motion control system provides simple visual control interface, and it can accurately control the motion platform immediately. Moreover, it can not only monitor the location and feedback status of the platform at any time to take appropriate protection procedures when it comes up against the over-limited of motion platform, but also avoid improper operation by users which can causes damage to machines or even worse result in accidents.As well, this system can plan the route where motion control to the demand of the industry.

目錄

圖例目錄

表格目錄

第一章

緒論

1.1 研究背景

TFT-LCD 亦即 Thin Film Transistor Liquid Crystal Display(薄膜電晶體液晶顯示器)。目前業界所採用的 LCD 量測程序，大致上是將已完成的待測 LCD 燈管先預熱，然後再搬移至暗房並固定於量測機台上，再由操作者設定相對位置、適合檢測參數予以量測；在此檢測程序中，因功能參數多且校正程序複雜，操作員需要長時間的摸索，才能讓檢測機台順暢作業。至於在自動化部份，僅可達到由軟體控制光學量測儀器 BM7 部份，無法符合連續自動化的要求。因爲近期內 LCD 產品將進入 TV 時代，尺寸亦將由現行的 15-19 吋增加至 20-50 吋，若因循舊制，以人工搬移、定位量測的方式，不僅效率低，且精確度亦堪虞。因此檢討國內 TFT-LCD 檢測程序，可發現有下列顯而易見之缺失 : (1) 現有設備欠缺自動輸送搬運機台。(2) 定位機構與光學量測系統尚未完全整合。(3) 至於國外現有檢測機台，價位偏高，故難以全面採用，若客戶嚴格要求，只能象徵性採購使用，以應付客戶。基於上述之現場實際使用的缺失，包含自動輸送設備與整合定位機構與光學量測系統之 LCD 自動檢測機台仍是目前急待改善課題。

1.2 研究動機及目的

TFT-LCD 顯示器工業近期在台灣蓬勃發展，帶動相關光學零組件上、中、下游迅速發展。光學零組件新產品設計、開發、驗證皆需仰賴光學檢測系統來掌控產品設計品質。因爲檢測系統穩定性、精度相當重要，所以機械光學檢測系統廣泛應用於 TFT-LCD 各相關零組件製程上。在機械光學視覺檢測系統中，量產品質檢測項目包含，如 : 點滅輝點 (有時候會亮、有時候又不會亮的” 亮點”)、異物 (灰塵 .. 等)、數量、色差程度，干涉條紋等等係由人工判定，其錯誤可能性即在所難免；爲了減少人工判定之錯誤，本研究採用晶研廠商所設計出 TFT-LCD 具精密度與穩定性之光學檢測架構與馬達控制系統的 TFTD 機台，以提供檢測面版是否因灰塵造成畫質不良、顯示不佳或是量測其電阻值來評估電流量。本機台利用運動控制卡來控制伺服馬達精準的定位到面板的每一格方格，和利用影像擷取卡來擷取影像，將擷取之影像作處理，進而達到自動化面板 Debug 功能。此設計也適用於大尺寸面板量測自動化機構系統，而如何實現這樣的系統就是本論文所要探討的。

根據上述討論的各項因素，本研究爲使用廠商的 TFTD 檢測系統發展出一套：1. 具備有運動控制、影像檢測、2. 可在任何作業平台上執行、 3. 開發成本低的人機介面，並搭配設備中常用的運動平台 (X-Y-Z Table)、CCD 檢測儀器來完成自動化的 LCD 面板檢測或是量測微機電蝕刻，結合運動控制與影像檢測的『視覺導引與定位系統』的整合性應用實例。

本論文的以 Visual C++ 程式語言爲軟體開發架構，雖然 Visual C++ 在學習上較其他程式困難得多，但其功能強大的函式庫與 Microsoft Window 作業系統的相容性高，且就工業界絕大部分都選擇以 Visual C++ 作爲系統軟體開發而言，其穩定性高，維修容易的缺點，使得 Visual C++ 成爲準工業標準之一。

1.3 文獻回顧

2000 年，蔡佳仁 [1]，此論文實現一套具有圖控、運動控制、影像檢測等三個功能模組之整合型監控系統。並且藉由各模組之間的整合，而完成遠端監控之工廠現場資料收集系統、遠端監控之工廠馬達定位系統、遠端監控之工廠影像檢測系統、遠端監控之工廠自動檢測系統等四個工廠自動化裡頭常見的例子。

2002 年，郭俊宏 [3]，由於目前相關的運動控制系統皆以 LabVIEW 發展視窗圖控介面，優點在於易學易懂，操作容易，缺點在於元件多而雜時，無法對於單一目的系統進行直接有效的控制，因此，此論文嘗試以 Visual C++ 程式語言爲軟體開發架構取代之，三軸微步進馬達平台爲控制對象，利用現有的運動控制卡，發展出一套精準快速的運控控制系統，以分工的方式，將所需精密複雜的運動演算法交由運動控制卡計算，相關的控制機制、運動軌跡則交由個人電腦掌控。

2003 年，曾彥馨 [4]，設計 TFT-LCD 瑕疵檢驗機台，CCD 攝影機架設於直桿上，可以改變 CCD 鏡面中心點到 TFT 液晶板之間的距離，其距離以專業協會之經驗提供，在此研究中的 15 吋 TFT-LCD 其與 CCD 鏡面中心距離爲 50 公分 (正負 1-2mm 之公差範圍內)，其架構如圖 1.1所示。圖 1.1中左側可旋轉之臂上裝載 LED 光源，以提供足夠光線照明 TFT-LCD，同時可以在平面上旋轉以檢測不同位置。使用的開發軟體是 Borland 之 Builder C++5.0 版。

2003 年，羅文振 [5]，採用機器視覺檢測技術設計出 TFT-LCD 具精度與穩定性之自動化光學檢測雛型機的架構與馬達控制系統，以提供檢測 TFT-LCD 設計驗證階段光學特性之亮度、均一性、色度、色飽和度 (NTSC)、階調反應速度、閃爍、對比、視角等有關光學之數據，再以所欲檢驗之需求，來控制馬達之轉動，而此論文則是以 LabVIEW 爲開發軟體。

1.4 論文架構

本論文架構可分爲五大章節：第一章爲緒論，說明研究動機、文獻回

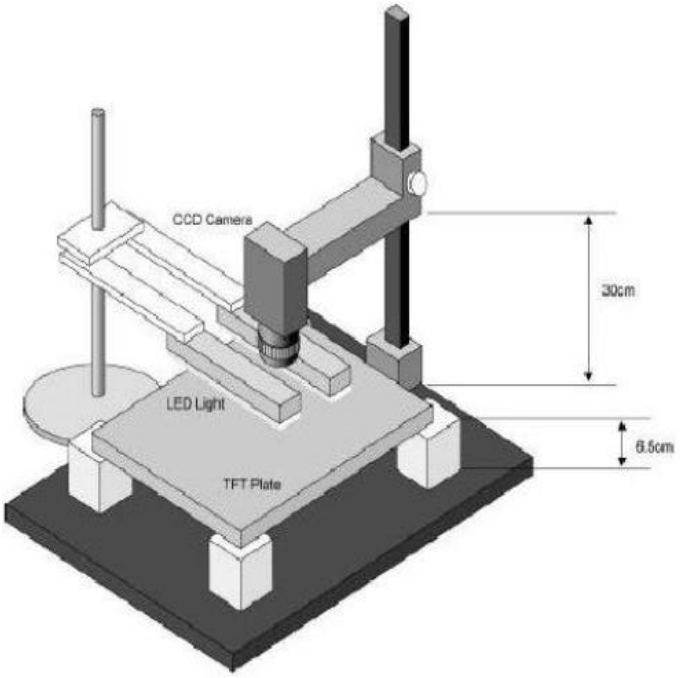


圖 1.1: TFT 瑕疵檢測機台 [4]

顧、研究目的及論文架構。第二章爲系統架構，說明系統軟體和硬體兩方面的主要架構。第三章爲系統設計，對本論文所實作的系統程式設計作詳細說明。第四章爲實作測試，針對本系統之各種功能作結果測試。第五章爲結論與未來發展，對實作結果進行討論，並提出未來改進方向及建議。

第二章

基本知識介紹

2.1 概述

對本論文之系統硬體架構，大致如圖 2.1所示，包括個人電腦、運動控制卡、轉接盒、驅動器及真空腔量測 (TFTD) 機台，並在 Visual C++ 的整合性開發環境下，發展運動控制系統。

此章節分成軟體架構、硬體架構兩大部分。

軟體方面：介紹 Visual C++ [9] 的 IDE(Integrated Development Environment，整合性開發環境)，以及所使用的 Applicaition Framework：MFC(Microsoft Foundation Class)，和常用的類別元件。

硬體方面：介紹真空腔量測平台架構和運動控制卡的基本架構。

2.2 軟體架構

2.2.1 整合性開發環境

IDE 是指一個應用程式開發的環境，一個完全自給自足的環境，也就是將設計所使用的工具整合在一個環境之下，可以建立編輯、連結程式，也可以對程式進行測試，對於 C++ 的開發環境而言是相當良好的學習或作業平台，圖 2.2爲 Visual C++ 的 IDE 介面。組成的單元包含：

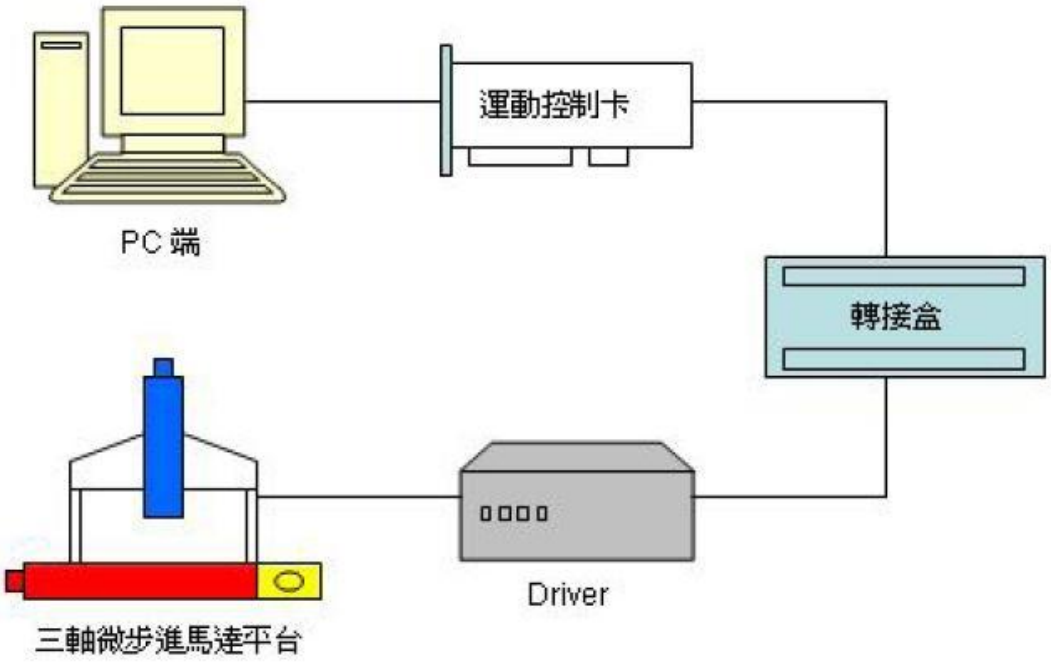


圖 2.1: 系統主要硬體架構 [3]

● 程式編輯器

用來建立和編輯 C++ 的原始程式碼，且自動辨識 C++ 的關鍵文字，和根據不同的類別以不同的顏色來顯示 (可以自行設定改變顏色)，主要是讓程式碼更容易辨識和除錯。

● 編譯器

將程式碼轉換爲機械語言，轉換的過程若發現程式碼有錯誤，則會回報錯誤的狀況 (顯示在【Output】視窗中)，編譯偵錯的範圍包括無效或無法辨識

的程式碼、結構的錯誤等等。編譯後的程式碼稱之 object code，其檔案的名稱爲 object file，所以編譯後的檔案副檔名爲 obj。

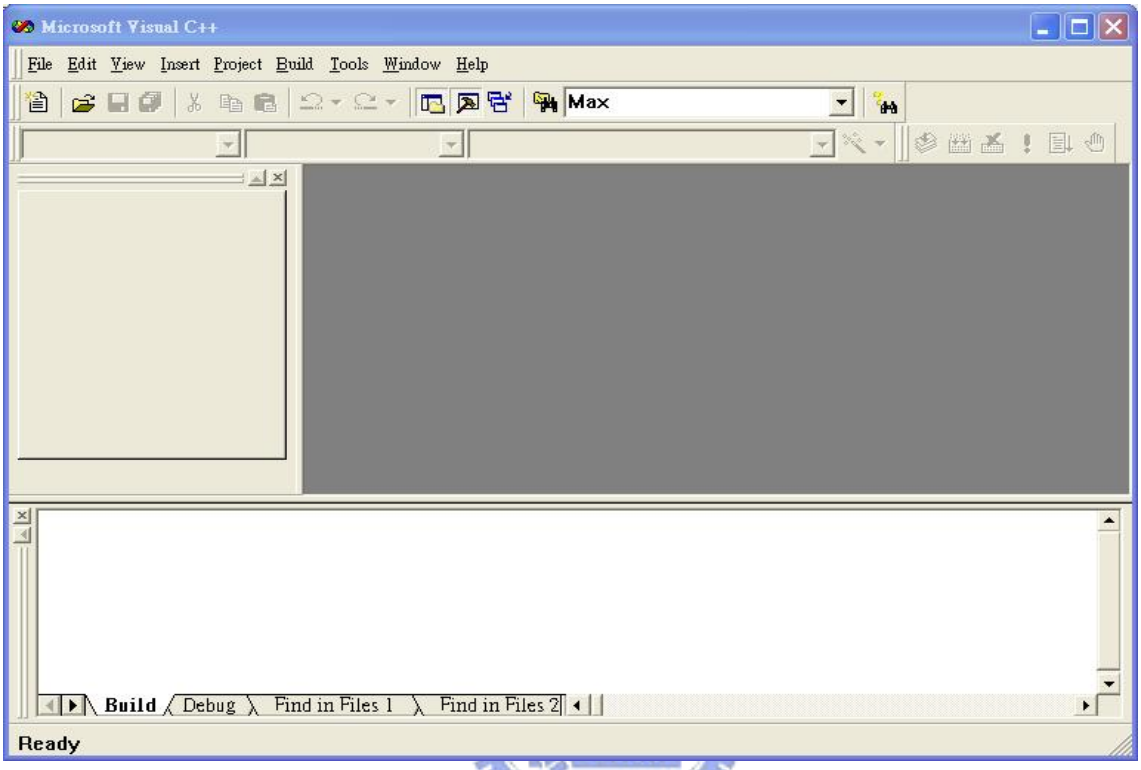


圖 2.2: Visual C++ 的整合性開發環境

● 連結器

將許多模組結合爲一個完整的可執行檔，這些模組是由原始檔案經編譯器編譯而產生的，然後再由函式庫中加入某些程式所需的程式碼模組。當某些程式不存在，或關聯到不存在的函式庫元件時，連結器就會偵查到並且產生訊息。

● 函式庫

提供某些特定程序來支援程式語言未提供的功能，共分爲兩種函式庫。第一種是所謂的標準函式庫，可跨平台。第二種是 MFC 和 Template (MFC&T)，分爲三個部分：MFC(產生視窗元件之用)、ATL(Active Template Library) 和

OLEDB(Object Linking and Embedding DataBase Template Library)。因爲 ATL 及 OLEDB 部分與本論文無關，故省略不作介紹。

以上所列爲 IDE 之基本單元部份，尚還有其他輔助工具：

● **應用程式精靈** *(AppWizard)*

AppWizard，如圖 2.3，會爲 Window 程式產生基礎的架構，而這架構本身是完整可執行的程式。本論文將利用 AppWizard 產生一個對話盒的基礎架構。

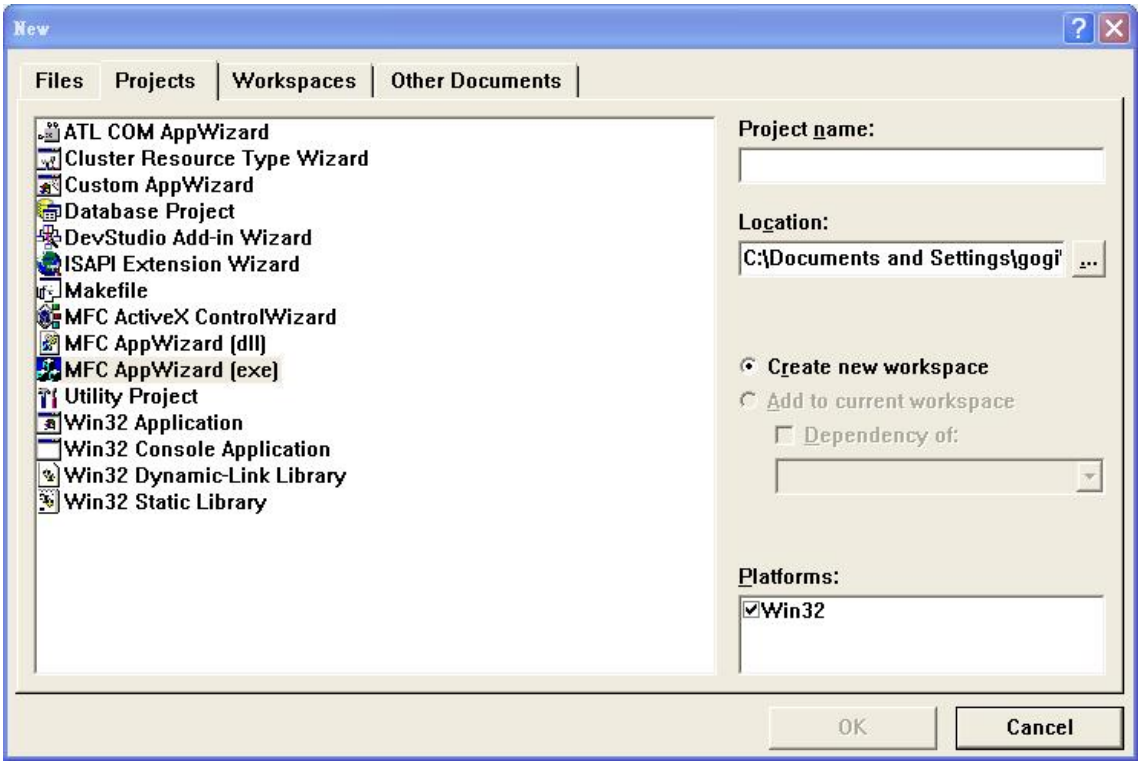


圖 2.3: AppWizard 應用程式精靈

● **類別精靈** *(ClassWizard)*

ClassWizard，如圖 2.4，會在 AppWizard 所產生的架構中加入基礎類別，亦允許以 MFC 爲基礎來新增類別。

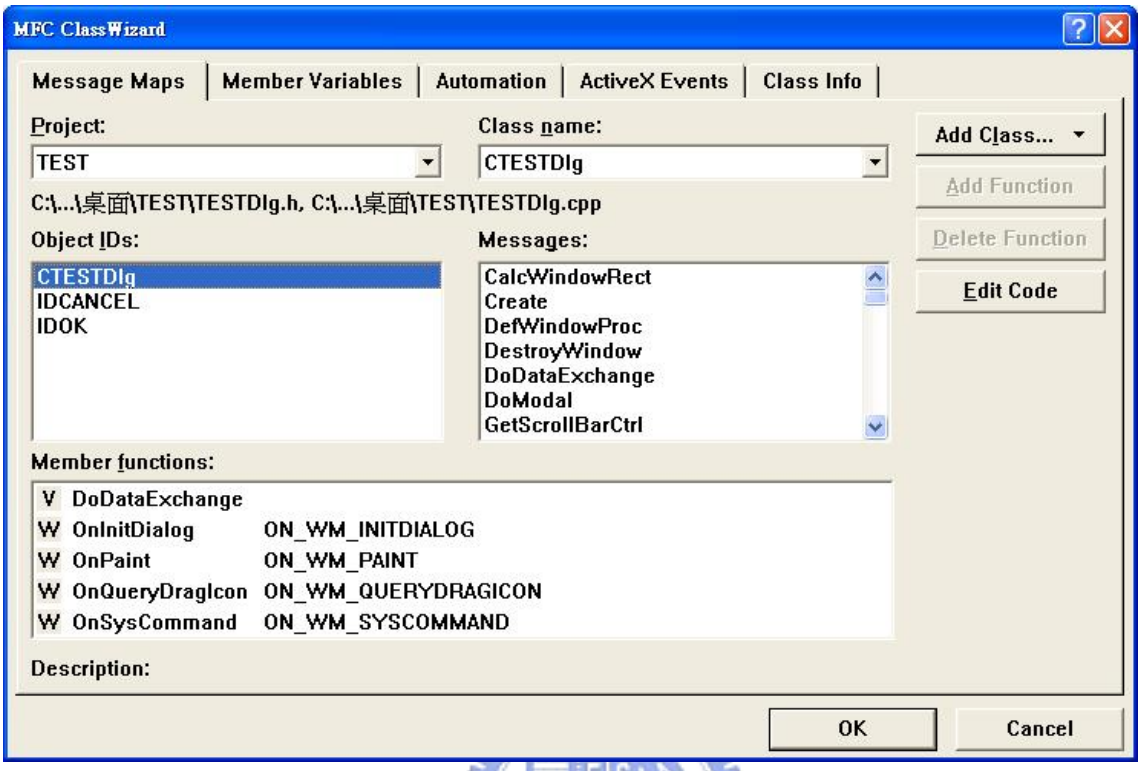


圖 2.4: ClassWizard 類別精靈

2.2.2 MFC

在介紹 MFC 之前，首先必須了解到什麼是 Application Framework。基本上，Application Framework，是一個完整的程式模型，具備標準應用軟體所需的一切基本功能，像是檔案存取、列印預視、資料交換 ...，以及這些功能的使用介面 (工具列、狀態列、選單、對話盒)，由這些合作無間的物件，彼此藉訊息的流動而溝通，並且互相呼叫對方的函式以求完成任務。Application Framework 是一個軟體開發工具的骨幹，當骨幹一致化，便會使優越的軟體開發工具更容易開發出來，也因爲程式碼大架構掌握在 Application Framework 設計者手上，於是他們有能力製作出整合開發環境。這也是爲什麼 Microsoft、Borland、Symantec、Watcom、IBM 等公司的整合開發環境進步的如此令人咋舌的原因。

雖然 Application Framework 並不是新觀念，它們卻在最近數年才成爲 PC 平台上軟體開發的主流工具。物件導向語言是具體實現 Application Framework 的理想工具，而 C++ 編譯器在 PC 平台上的出現與普及，終於允許主流 PC

程式員能夠享受 Application Framework 帶來的利益。

目前共有三套 Application Framework，分別是：Microsoft 的 MFC，Borland 的 OWL(Object Window Library)，以及 IBM VisualAge C++ 的 Open Class Library。至於其他 C++ 編譯器廠商，如 Watcom、Symantec、Metaware，只提供硬整合開發環境，其 Application Framework 都是採用微軟公司的 MFC。

早期，開發 Windows 應用程式必須使用微軟的 SDK(Software Development Kit)，直接呼叫 Windows API 函式，向 Windows 作業系統提出各種要求，例如配置記憶體、開啓視窗、輸出圖形 ...。數以千計的 Windows APIs 彼此雖有群組關係，卻沒有相近或組織化的函式名稱。星羅棋布，霧列星馳；又似雪球一般越滾越多，越滾越大。撰寫 Windows 應用程式需要大量的耐力與毅力，以及大量的小心謹慎 !MFC 把這些浩繁的 APIs，利用物件導向的原理，邏輯地組織起來，使它們具備抽象化、封裝化、繼承性、多型性、模組化的性質。

2.2.3 MFC 類別元件

視窗大致分兩種：對話盒 (Dialog)，見圖 2.5，和文件 (Document)，而文件又分爲 SDI(Single Document Interface) 和 MDI(Multiple Document Interface)，見圖 2.6和 2.7。本論文選用對話盒作爲視窗介面，乃是取其明確簡潔、易操控的優點，因此對文件部分不加著墨。

圖 2.5: 對話盒

Visual C++ 提供一個資源編輯器的環境，讓對話可任意地改變視窗型

圖 2.6: SDI

態的大小和內容。圖 2.8爲利用 Application Wizard 產生一個對話盒架構。此時，新的對話盒只有兩個按鈕：OK 和 Cancel。在對話盒旁有個子控制工具列 (圖 2.9)，裡面包含許多對話盒所需要的子控制類別，包括靜態物件 (Static Text)、編輯子控制 (Edit Box)、群組合 (Group Box)、旋鈕 (Radio Button)、按鈕 (Button)、拖曳棒 (Slider Control) 等。

Window 程式採取所謂的事件驅動 (Event-driven)，就是所謂的使用者按下滑鼠或按下某個鍵，或是記時時間到了，此時就會產生事件 (Event)。Windows 將每個事件記錄成一個訊息，並將其排列成好放在訊息佇列中，讓程式知道現在要處理的是哪個訊息，並透過訊息映射，將訊息與處理的函式做連結。

圖 2.7: MDI

以往設計 Window 程式時，程式設計師會把必須要處理的訊息宣告在標

頭檔裡 (filenameDlg.h)：

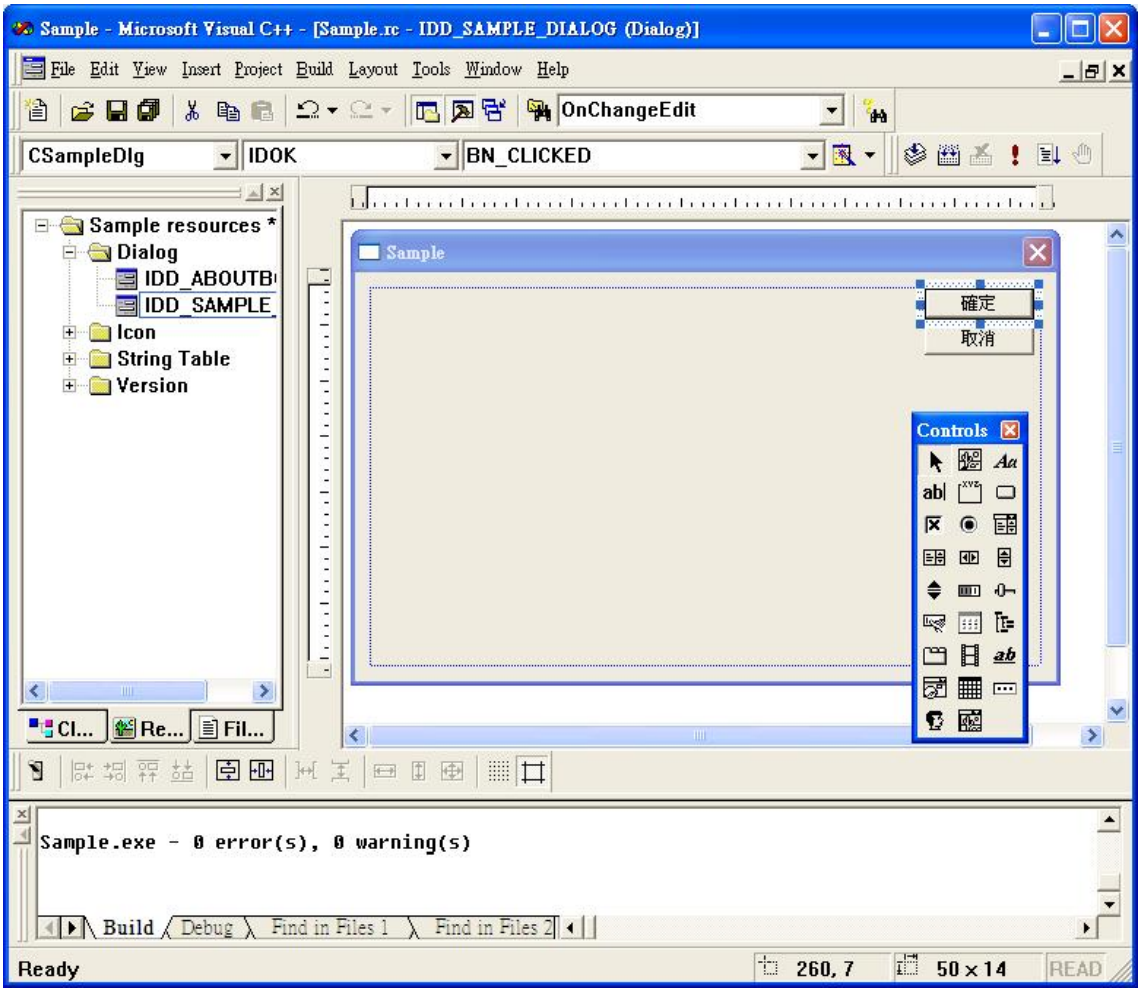


圖 2.8: 空白對話盒編輯環境

```
    afx_msg void OnButton();
afx_msg void OnChangeEdit();
afx_msg void OnReleasedcaptureSlider(NMHDR* pNMHDR, LRESULT* pResult);
DECLARE_MESSAGE_MAP()
```

afx_msg 後面宣告的函式代表訊息對應的處理事件。以 OnChangeEdit() 來說，所對應的便是輸入資料進去時，要處理事件的函式。接著要在主程式 (filenameDlg.cpp) 碼裡加入：

```
BEGIN_MESSAGE_MAP(CSampleDlg,CDialog)
ON_BN_CLICKED(IDC_BUTTON,OnButton)
ON_EN_CHANGE(IDC_EDIT,OnChangeEdit)
ON_NOTIFY(NM_RELEASEDCAPTURE,IDC_SLIDER,OnReleasedcaptureSlider)
END_MESSAGE_MAP()
```

BEGIN_MESSAGE_MAP() 和 END_MESSAGE_MAP() 之間的程式，針對所發生的事件將相關聯的元件及處理函式做連結。舉列來說，當代號 IDC_BUTTON 的按鍵被按下時，Window 會發出 BN_CLICKED 的訊息，程式便會根據元件 (IDC_BUTTON) 所觸發的事件 (BN_CLICKED) 找到相對應的處理函式 (OnButton())。

圖 2.9: 子控制工具列

現在，透過 MFC 所提供的 ClassWizard，程式設計者能很輕易方便的寫

出想要的訊息對應程式，只要點選元件即發生的事件，ClassWizard 便會自動在程式裡加入所需的程式，設計者只需撰寫處理函式內容即可。

以下將介紹本論文所使用的子控制、觸發事件和對應的處理函式 (參閱表 2.1)：

●按鈕 **(Button)**

當按紐被按下時，Windows 送出 ON_BN_CLICKED 的訊息，並執行 OnButton()。在本運動控制系統的用途是當步階移動、校正回歸和關閉視窗。

●特殊按鈕 **(Microsoft Forms 2.0 CommandButton)**

此按鈕的處理事件的函式，放在 filenameDlg.cpp 內的 BEGIN_EVENTSINK_MAP() 和 END_EVENTSINK_MAP() 之間。當按紐被按下時，Windows 送出 ON_EVENT 的訊息，並執行 OnMouseDownCommandbutton()。當按紐被放開時，Windows 送出 ON_EVENT 的訊息，並執行 OnMouseUpCommandbutton() 在本運動控制系統的用途是當方向鍵。

●編輯 **(Edit)**

若 Edit 元件內容被更改時，Windows 會送出 ON_EN_CHANGE 訊息，並根據映射表所聯結的函式處理，執行 OnChange()。在本系統中的功能是顯示並輸入資料，以控制速度、加速度、減速度和移動間距。

●特殊編輯 **(Microsoft Forms 2.0 TextBox)**

此按鈕的處理事件的函式，放在 filenameDlg.cpp 內的 BEGIN_EVENTSINK_MAP() 和 END_EVENTSINK_MAP() 之間。在本系統中的功能不是以傳送訊息爲主，而是以改變該 TextBox 物件的特性來做 I/O Static 顯示，以顏色代表該 Static 爲 High 或 Low。

表 2.1: Visual C++ 函式列表

按鈕 (Button)	OnButton( )
特殊按鈕 (Microsoft Forms 2.0 CommandButton)	OnMouseDownCommandbutton( )
	OnMouseUpCommandbutton( )
編輯 (Edit)	OnChange( )
特殊編輯 (Microsoft Forms 2.0 TextBox)	SetBkColor( )
拖移棒 (Slider)	OnReleasedcapture( )
對話盒(Dialog)	OnClose( )
	OnOK( )
	ONInitDialog( )
	OnPaint( )
	OnQueryDragIcon( )
	OnSysCommand( )

●拖移棒 **(Slider)**

藉由按住滑鼠左鍵或鍵盤左右方向鍵移動拖曳棒的位置，當放開滑鼠或鍵盤的按鍵時，Windows 會送出 NM_RELEASEDCAPTURE 訊息，根據映射表，執行 OnReleasedcapture()。在系統中的功能是藉由拖曳棒移動的位置，控制速度、加速度、減速度和移動間距。

Edit 和 Slider 元件控制的項目是相同的，而且是同步調的，因此必須在其對應的函式中，加入適當的程式，這部份將會再往後的第三章提到，故在此先跳過不提。

●對話盒 **(Dialog)**

雖然對話盒對應的處理函式那麼多，不過卻都是 MFC 在架構新的對話何時，就已經加進去的，程式設計者不必再透過 ClassWizard 新增這些函式，只需針對系統的要求，修改部分函式的程式碼即可。依本系統的要求，需修改 OnClose()、OnOK() 和 OnInitDialog()。

– OnClose() 此函式對應的元件事對話盒視窗右上角畫 x 的按鈕，用途在於關閉對話盒。在本系統中的功能也是關閉作用，不過必須稍加修改，因爲系統關閉的時候，必須讓運動平台回歸至指定的原點，確保當下次系統開啓，能在安全的範圍內運動。

– OnOK() 此函式對應的子控制元件爲 OK 按鈕，是在建構新的對話盒時就已經和另一個 Cancel 按鈕同時存在，只不過在系統中，Cancel 按鈕不曾使用，所以隱藏起來。OnOK() 處理的事件和 OnClose() 一樣，進行關閉系統時所需的步驟，讓運動平台回歸至指定的原點，確保當下次系統開啓，能在安全的範圍內運動 (即所謂的避免碰撞 Limit 開關)。

– OnInitDialog() 此函式負責起始對話盒的狀態及視窗的規劃，需修改的是系統開啓時，相關初始的設定。當控制視窗開啓時，必須檢查上次關閉前，原點的位置狀態；速度、加速度、減速度、和移動間距的極大值、極小值的設定；各個 Edit 的顯示內容和 Slider 控制棒的位置等。

其餘的子控制類別，例如：Static 和 Group Box，由於功能在於顯示狀態及標示說明，無事件觸發及相對應的處理函式，所以在此先省略，往後章節會在詳細介紹。

2.3 硬體架構

硬體架構包含 : 個人電腦、運動控制卡、轉接盒、電源供應器 (power supply)、馬達驅動器 (driver) 和三軸微步進馬達平台。此節將介紹主要架構 : 運動控制卡和三軸微步進馬達平台。

2.3.1 運動控制卡

在介紹此系統所使用的運動控制卡之前，先大致對運動控制卡的主要架構做個簡單的敘述說明。就控制馬達訊號而言，當使用者經由視窗下達一個動作指令給控制卡時，控制卡會將其指令轉換成 DDA(Digital Differential Analyzer) [10] 的脈波訊號，經由轉接盒 (圖 2.10) 傳至馬達驅動器，馬達驅動

器則將控制訊號轉換成電流或電壓，供應給步進馬達。轉接盒的用途便是將運動控制卡介面的接腳和驅動器訊號線作連接。

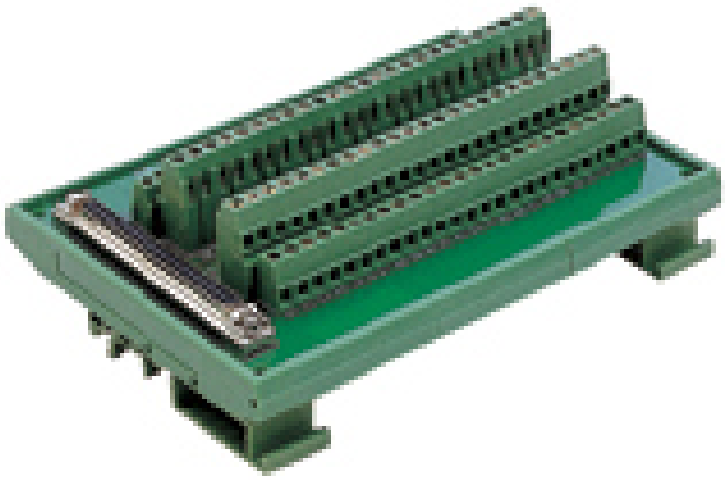


圖 2.10: 控制卡轉接盒 [11]

DDA 提供的脈波輸出是最普遍的步進馬達控制信號。使用者自行設定 DDA 週期時間 (DDA cycle time)，然後將馬達速度 (pulses/sec) 換算成在 DDA 週期內所要的脈波數，運動控制卡則在每一個 DDA 週期內產生均勻的脈波數，輸出給驅動器，因而馬達在每一個 DDA 週期內被要求作等速運動。DDA 要求差異 (differential) 輸出，意即任一信號以二條線輸出，差異信號爲原信號之反相，圖 2.11 展示兩種差異輸出法：脈波／方向 (pulse/dir) 法和順時／逆時 (CW/CCW) 法。

驅動器在運動過程中也會將訊息回售給控制卡 (位置或速度)，回授的訊息通常分兩種 : 四象限增量編碼器回授 (quadrature increment encoder feedback) 和類比回授 (analog feedback)。四象限增量編碼器是最普遍使用的訊息回授，這種編碼器隨馬達旋轉產生 90° 相差的 A 相和 B 相方波（見圖 2.12），在一個方波週期內 A 相和 B 相有四種組合，故能提供四倍數的記數，同時從組合的出現順序判斷出正轉或反轉。當編碼器轉一圈時，亦會送出一個 Index 的脈波給控制卡，作爲運動控制卡回歸零點時的一個依據。

由於編碼器的信號爲控制用的迴授信號，信號故障或斷線都會造成馬達

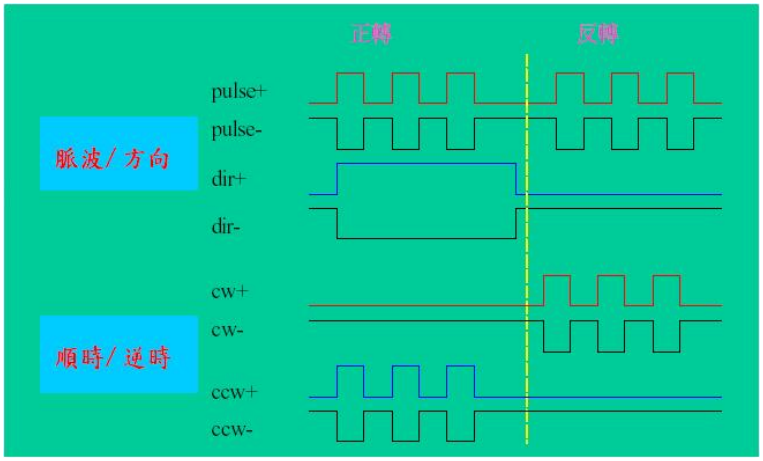


圖 2.11: DDA 的輸出訊號 [10]

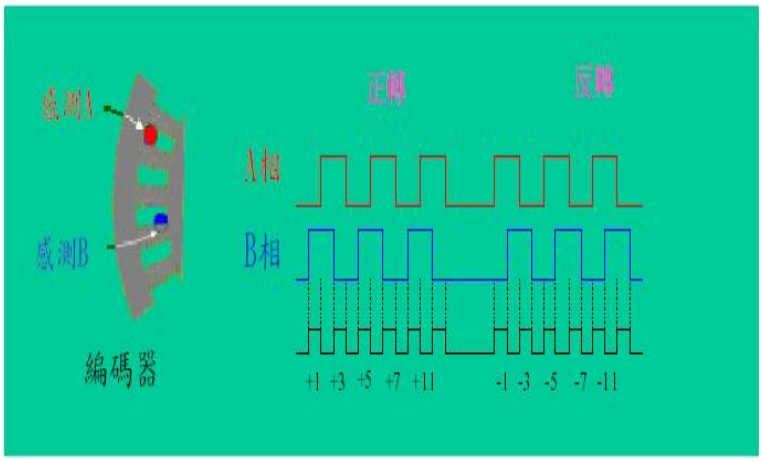


圖 2.12: quadrature increment encoder 的 A 相和 B 相訊號 [10]

亂動而使機械系統損毀或其他危險事故，因而斷線檢測功能極爲重要。類似 DDA 的差異輸出法能用來解決此問題，定義 A+ 和 B+ 爲原信號，A － 和 B － 爲反相信號。圖 2.13 中 A+/A － 與 B+/B － 各經一個 XOR 邏輯器，再經過一個 NAND 邏輯器。唯有兩信號不同時（一爲 0，另一爲 1）XOR 輸出才爲 1；而僅有當兩 XOR 輸出皆爲 1 時，NAND 輸出方爲 0；所以一旦有信號不正確，NAND 輸出就爲 1，並立即觸發信號錯誤處理程式。

根據運動控制的所需功能，可以將市面上的運動控制卡的構造綜合以圖 2.14 來闡述，但是以一軸爲例。對外介面中與個人電腦相接爲 FIFO(First-In-First-Out) 暫存器 , 其通常有 256-byte 的空間，若一個指令佔 2 bytes，則可

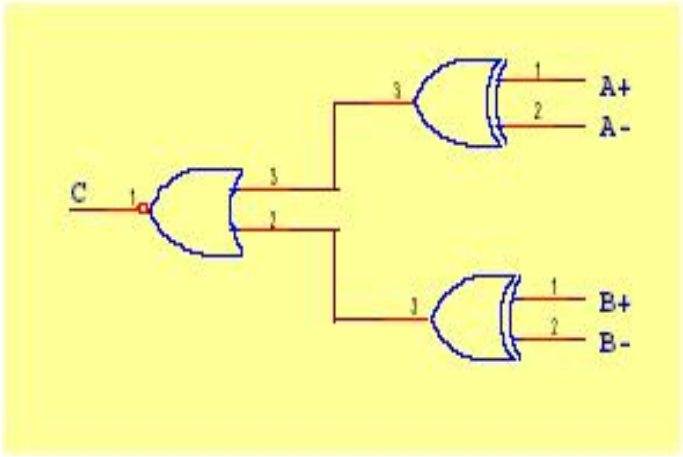


圖 2.13: A、B 相訊號斷線檢測法 [10]

存 128 個指令。其他介面可分二部分：列於圖之右側爲基本介面，其餘爲額外配備。DAC 的標準規格爲 16-bit 解析度和± 10V 的輸出，雖然對大多數的應用系統，DAC 是多餘的。每軸需 6 條編碼器輸入線，4 條 DDA 輸出線，工作電壓皆爲 5V，僅在極特殊情形才會使用 12V。數位輸入線有三條，利用光耦合隔離 (opto-isolation)，內部工作電壓也爲 5V，但外部接點的電壓爲 24V。通常，編碼器的信號線也會用光耦合隔離，以防控制卡因誤接電氣線而毀損。編碼器和 DDA 的取樣頻率決定高速控制能力，規格從 1MHz 至 20MHz。常見的運動控制卡的編碼器頻率爲 12MHz，而 DDA 爲 3MHz；較高級者則分別爲 20MHz 和 8MHz。

額外配備中，3 個 DI 和 2 個 DO 是用來讀絕對位置編碼器的記數器。ADC 爲類比轉數位輸入可量測感測器的電壓輸出，規格與 DAC 相同，也是大多數情況下用不到。栓鎖觸發輸入 (latch trigger input) 爲高速位置捕捉功能，當此觸發輸入發生時，會在 0.1ms 的時間內將編碼器的位置值鎖住並記下來。另外，編碼器比較輸出 (encoder compare output) 提供高速的位置比較功能，設定起始脈波和目標的脈波增量後，當增量脈波數達到時會在 0.1ms 內輸出信號，所以可以作精確位置觸發輸出。被控制的系統都會有許多週邊設備，例如氣壓控制元件、近接開關等，所以需要一些額外的數位輸出輸入點，可規劃的 I/O 就是提供這方面的需要，需注意的是這些爲可擴充 I/O 點，必須連接擴充 I/O 板（上面有光耦合隔離元件）來使用。現在市售運動控制板大多內含一顆 DSP（數位訊號處理晶片）用來處理高速運算，有 DSP

的運動控制板可載入程式自行作即時控制，而個人電腦僅作視窗式的人機介面，否則需再用單板工業型個人電腦作即時控制程式。

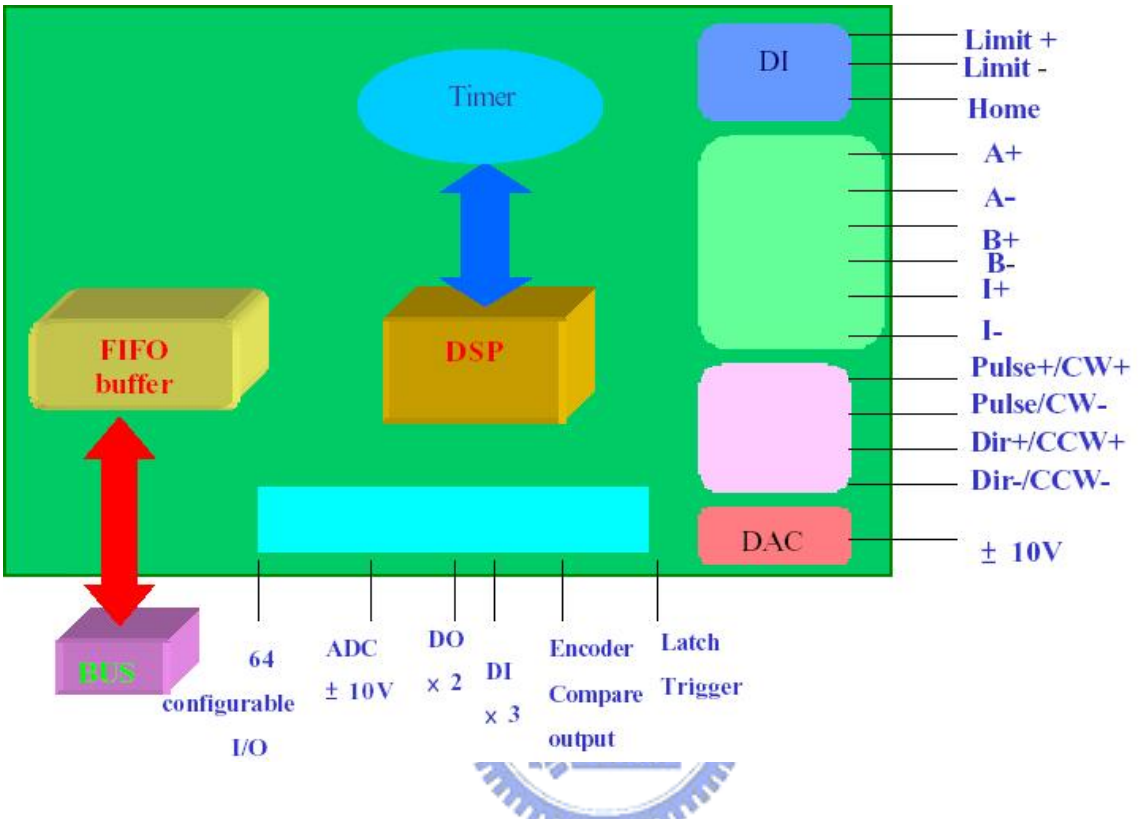


圖 2.14: 運動控制卡基本架構 [10]

本系統所採用的用動控制卡爲 ADLINK 公司所出，型號爲 PCI-8134 的 PCI BUS 運動控制卡 (圖 2.15)。主要運動模式如下 :

1. 連續運動
2. 絕對運動
3. 相對運動
4. 手動脈衝方式
5. 同步運動
6. 運動中改變速度
7. 線性插補

PCI-8134 運動控制卡雖然具備種種之運動模式，但並非所有的模式都會同時用到，必須由受控端而決定，以此論文之運動控制系統來說，此系統是

圖 2.15: PCI-8134 PCI BUS 運動控制卡 [11]

以三軸微步進馬達平台爲受控對象，設定 XY 軸平面控制、Z 軸垂直控制，因此運動模式是以直線補間爲主。PCI-8134 運動控制卡皆具備前述運動控制卡之基本架構功能，爲獨三軸微步進馬達平台並沒有位置回授功能，此原因在於相較伺服馬達而言，步進馬達之控制精準，位置誤差甚小，所以對大部分之步進馬達而言，並沒有所謂的位置回授編碼器。但是，對於 PCI-8134 而言其還有外加編碼器介面的功能，主要用於位置反饋。編碼計數器提供位置資訊來校正由於錯誤的機械傳動所造成的位置誤差。而差動型編碼器反饋能夠消除雜訊的影響。28 位元計數器能夠覆蓋大多數應用的位置範圍。

2.3.2 PCI-8134 VC++ 函式庫

以下兩個爲提供 PCI-8134 運動控制卡一個專屬的 C/C++ 控制函式庫，如以下所列 :

1.Pci-8134.H: 包含了所有用到的控制函式。

2. 8134.LIB: 包含了所有函式的 Linking。

並針對 Borland 和 Microsoft 兩種不同的 C++ 程式開發環境，提供了 8134.LIB，所以在 Visual C++ 開發視窗程式時，必須先加入 PCI-8134.H。表 2.2爲本論文所引用之 PCI 函式列表及說明，所有函式接宣告在 PCI-8134.H 中。

一般程式的撰寫者，爲了讓程式看起來更簡潔，因此會將所有會用到的定義用簡短的英文單字組合，所以會再程式一開頭就先宣告，以方便以後需要使用。因此表 2.3爲 PCI-8134.H 內所定義函式的宣告和該宣告的描述

表 2.2: 運動控制卡函式列表

W_8134_InitialA	Open communications w/r/interrupt
W_8134_Close	Close communications
W_8134_Set_Config	Set Config
Command	Send a command

以下便以表 2.2所列之函式，參考 PCI-8134 所附之工具箱參考手冊，作更加詳細的說明 :

- W_8134_InitialA

U16 PASCAL W_8134_InitialA(I16 *TotalCard);

This function is used to initialize PCI-8134 card. Every PCI-8134 card has to be initialize by this function before calling other functions。

表 2.3: PCI-8134 程式宣告對照表

[11]

```
typedef unsigned char    U8;
typedef short            I16;
typedef unsigned short   U16;
typedef long             I32;
typedef unsigned long    U32;
typedef float            F32;
typedef double           F64;
typedef char             Boolean;
```

Type Name	Description	Range
U8	8-bit ASCII character	0 to 255
I16	16-bit signed integer	-32768 to 32767
U16	16-bit unsigned integer	0 to 65535
I32	32-bit signed long integer	-2147483648 to 2147483647
U32	32-bit unsigned long integer	0 to 4294967295
F32	32-bit single-precision floatint- point	-3.402823E38 to 3.402823E38
F64	64-bit double-precision floating-point	-1.797683134862315E308 to 1.797683134862315E308E309
Boolean	Boolean logic value	TRUE,FALSE

U16 定義爲 16-bits unsigned integer(0 to 65535)

TotalCard : 給該 PC 上的 PCI-8134 的運動控制卡一個編號 (數字)，作爲登入的代碼。

- W_8134_Close

U16 PASCAL W_8134_Close(int card_number);

This function is used to close PCI-8134 card and release the PCI-8134 related re-

sources，which should be called at the end of an application。

card_number : number of existing PCI-8134 cards。

● Command

Note 所有設定控制卡的參數和運動的函式 ... 等等。此於下一節說明。

● W_8134_Set_Config

U16 PASCAL W_8134_Set_Config(char *fileName); //fileName 是一個設定的資料庫

This function is uesd to configure PCI-8134 card.All the I/O configurations and some operating modes appears on "Axis Configuration Window" of Motion Creator will be set to PCI-8134. Click "Save Configuration" button on "Axis Configuration Window" if you want to use this function in the application program. Click "Save Configuration" button will save all the configurations to a file call "8134.cfg". This file will appear in the "WINDOWS/SYSTEM/" directory.

以下的範例爲整合上述的說明 :

```
#include "pci_8134.h"
I16 TotalCards=0;

    char SysDir[255];
CString FStr;

    int main(void)
{
// 啓動運動控制卡
```

```
W_8134_InitialA(&TotalCards);
// 將設定安裝在 Window 下的 Configuration，產生的 8134.cfg 檔案
GetSystemDirectory(SysDir,1024);
FStr.Format(" %s",SysDir);
// 將 8134.cfg 寫入
FStr+="//8134.cfg";
Err=W_8134_Set_Config(FStr.GetBuffer(0));
// 若找不到該檔則會出現錯誤的訊息
if( Err )
AfxMessageBox("Can't  File in 8134.cfg");
// 在 X 軸上以加速度 0.1 從速度 0 到速度 100，移動 1000 個單位
start_r_move(0,1000,0,100,0.1);
// 關閉編號 0 的運動控制卡
W_8134_Close(0);
return 0;
}
```

2.3.3 PCI-8134 指令集

ADLINK 的控制卡所採取的指令爲縮寫和口語化的指令，也就是每個指令的英文組合直接翻後，皆可以解讀其動作意思，具有易學、易用等優點。本節將逐一說明系統所用到的指令集，從控制到 I/O 的讀寫用法，各項指令說明如下 :

- U16 v_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tacc)

－ 功能 :

開始運動。

－ 描述 :

此函式被用來加速某一軸到達你設定的速度。該軸連續移動直到速度改變或是該軸被下命令停止移動。然而移動的方向則是輸入速度變數的符號。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

str_vel : 爲初速。

max_vel : 爲最大速度值和移動方向。

Tacc : 爲加速的時間。

– 例子 :

v_move (0, 0, 1000, 0.1) : X 軸正向移動從速度爲 0 加到 1000 需要 0.1 秒。

v_move (0, 0,-1000, 0.1) : X 軸反向移動從速度爲 0 減到 1000 需要 0.1 秒。

- U16 v_change(I16 axis, F64 max_vel, F64 Tacc)

– 功能 :

改變移動的速度。

– 描述 :

此函式爲改變速度到達使用者設定的新速度和從初速到達新速度所需的時間。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

max_vel : 爲最大速度值和移動方向。

Tacc : 爲加速的時間。

– 例子 :

v_change (0, 2000, 0.1) : X 軸從初速加速到 2000 需要 0.1 秒。

v_change (0, -2000, 0.1) : X 軸從初速減速到 2000 需要 0.1 秒。

- U16 v_stop(I16 axis, F64 Tdec)

– 功能 :

停止運動。

– 描述 :

此函式被用來以多少時間來停止 X 或 Y 或 Z 軸的移動。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

Tdec : 爲減速至 0 所需的時間。

– 例子 :

v_stop (0, 0.1) : X 軸從目前的速度到靜止需要 0.1 秒。

● U16 start_r_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tacc)

– 功能 :

相對的運動方式。

– 描述 :

此函式是以輸入 distance 的方式來達到位置控制。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

distance : 是指給定 pulse 的數量，而 v_move 則是一直送出 pulse 直到停止輸入，且 distance 給的正或負責表示馬達移動方向。

str_vel : 爲初速。

max_vel : 爲最大速度值。

Tacc : 爲加速至最高速度所需的時間。

– 例子 :

r_move (0, 2000, 0, 1000, 0.1) : X 軸正向移動以 0.1 秒從靜止到 1000 的速度來送出 2000 個 pulse。

r_move (0, -2000, 0, 1000, 0.1) : X 軸反向移動以 0.1 秒從靜止到 1000 的速度來送出 2000 個 pulse。

● U16 r_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tacc)

– 功能 :

相對的運動方式。

– 描述 :

此函式是以輸入 distance 的方式來達到位置控制，但須等待這次命令完成才能再下命令。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

distance : 是指給定 pulse 的數量，而 v_move 則是一直送出 pulse 直到停止輸入，且 distance 給的正或負責表示馬達移動方向。

str_vel : 爲初速。

max_vel : 爲最大速度值。

Tacc : 爲加速至最高速度所需的時間。

– 例子 :

r_move (0, 2000, 0, 1000, 0.1) : X 軸正向移動以 0.1 秒從靜止到 1000 的速度來送出 2000 個 pulse。

r_move (0, -2000, 0, 1000, 0.1) : X 軸反向移動以 0.1 秒從靜止到 1000 的速度來送出 2000 個 pulse。

- U16 wait_for_done(I16 axis)

– 功能 :

等待該軸移動完畢。

– 描述 :

此函式目的是確定移動的距離能確實達到，當該軸的目標達到才能繼續下個命令。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

– 例子 :

wait_for_done (0) 等待 X 軸移動停止。

- U16 home_move(I16 axis, F64 svel, F64 mvel, F64 accsl)

– 功能 :

原點歸位。

– 描述 :

此函式爲將目前的位置移動至滿足 set_home_config() 函式的設定才停止。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

svel : 爲初速。

mvel : 爲最大速度值且單位的正負爲其移動方向。

accel : 爲加速至最高速度所需的時間。

– 例子 :

home_move (0, 0, 1500 , 0.2) 等待 X 軸以 0.2 秒的時間從 0 到 1500 的速度正向進行移動，直到滿足 home 點的設定。

home_move (0, 0, -1500 , 0.2) 等待 X 軸以 0.2 秒的時間從 0 到 1500 的速度反向進行移動，直到滿足 home 點的設定。

● U16 set_home_config(I16 axis, I16 home_mode, I16 org_logic, I16 org_latch, I16 EZ_logic)

– 功能 :

設定原點歸位的條件。

– 描述 :

此函式將 ORG 和 EZ 的發生條件作爲滿足原點移動停止的條件。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

home_mode : 停止回到 ORG 的模式。

home_mode = 0，ORG active only。

home_mode = 1，ORG active and then EZ active to stop，high speed all the way。

home_mode = 2，ORG active and then EZ active to stop，high speed till ORG active then low speed till EZ active。

org_logic : ORG 訊號發生的 logic 設定。

org_logic = 0，active low。

org_logic = 1，active high。

org_latch : 鎖住 ORG 信號的狀態控制。

org_latch = 0，don't latch input。

org_latch = 1，latch input。

EZ_logic : EZ 訊號發生的 logic 設定。

EZ_logic = 0，active low。

EZ_logic = 1，active high。

– 例子 :

set_home_config (0, 2, 1, 1, 0) 。

● U16 motion_done(I16 axis)

– 功能 :

觀察該軸移動狀態。

– 描述 :

此函式目的是用來觀測使用者設定的軸目前處於的情況，當回傳值爲。

0 : 該軸正在運作。

1 : 運作結速。

2 : 該軸停在正極限。

3 : 該軸停在負極限。

4 : 該軸停在原點。

5 : 因爲 ALARM 訊號發生，該軸靜止。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

– 例子 :

motion_done (0) 觀察 x 軸運動狀態。

● U16 W_8134_Set_SVON(I16 axis, I16 on_off)

– 功能 :

設定該軸馬達的觸發電位。

– 描述 :

此函式爲設定 SVON 一般輸出 pin 腳的狀態爲 High/Low，通常這需要根據原廠馬達的 on/off的設定而加以修改腳位爲 0 或 1。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

on_off : 設定 SVON pin 腳的數位輸出。

on_off = 0，SVON is Low。

on_off = 1，SVON is High。

– 例子 :

W_8134_Set_SVON (0, 1) x 軸爲 1 時，使得 x 軸 reset。

W_8134_Set_SVON (0, 0) x 軸爲 0 時，使得 x 軸 reset。

● U16 get_io_status(I16 axis, U16 *io_status)

– 功能 :

顯示所有 PCI-8134 的 I/O 狀態。

– 描述 :

此函式爲取得每一軸的 I/O 狀態，該指令回傳的資訊包括 home switch、forward limit switch、reverse limit switch、error condition、motion condition 和 motor 狀態。回傳的值以 16 進位顯示，然而每一個狀態的 bit 定義如表 2.4。由於本系統必須即時偵測到機台有無觸碰到 forward 和 reverse 感測器或其他 motor 狀態改變，所以將該函式回傳的 16 進位值要跟檢查碼作 and 的動作，好讓系統判斷有無達到 forward 或 reverse 極限和監視 motor 狀態，附上表以作 forward 或 reverse 極限的說明 2.5。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

*io_status : ”1” 表示 ON 和 ”0” 表示 OFF。ON/OFF 狀態值對應於起初的邏輯的設定。

– 例子 :

get_io_status (0, &io_status) = 0x4 當 x 軸回傳值爲 0x4 時，表示啓動監視 +SD 的狀態。

get_io_status (0, &io_status) = 0x10 當 x 軸回傳值爲 0x10 時，表示啓動監視 ORG 的狀態。

get_io_status (0, &io_status) = 0x80 當 x 軸回傳值爲 0x80 時，表示啓動監視 SVON 的狀態。

get_io_status (0, &io_status) = 0x400 當 x 軸回傳值爲 0x400 時，表示啓動監視 ERC 的狀態。

表 2.4: I/O Status 各 bit 代表的意義 [11]

Bit	Name	Description
0	+EL	Positive Limit Switch
1	-EL	Negative Limit Switch
2	+SD	Positive Slow Down Point
3	-SD	Negative Slow Down Ponit
4	ORG	Origin Switch
5	EZ	Index signal
6	ALM	Alarm signal
7	SVON	SVON of PCL5023 pin output
8	RDY	RDY pin input
9	INT	Interrupt status
10	ERC	ERC pin output
11	INP	In-Position signal input

表 2.5: get_io_status 回傳值與檢查碼作 and 之結果說明表

	get_ io_ status	檢查碼	And結果	說明
二進位	000000000001	000000000001	000000000001	到達forward極限
十六進位	0x1	0x1	0x1	
二進位	000000000010	000000000010	000000000010	到達reverse極限
十六進位	0x2	0x2	0x2	
二進位	000000000100	000000000001 000000000010	000000000000	無法到達forward和reverse極限
十六進位	0x4	0x1	0x0	
註：bit0 和 bit1不可能同時爲0				

- U16 get_position(I16 axis, F64 *pos[axis])

– 功能 :

取得實際上的位置。

– 描述 :

此函式爲讀出現在實際上的位置，換言之，其職是從下位置命令得來的，且

是否啓動這項功能則與初始設定 set_cnt_src 有關。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

*pos[axis] : 實際的位置值。

– 例子 :

get_position (0, &pos[0]) 將 x 軸的實際上位置的值放入 pos[0] 內。

- U16 get_command(I16 axis, F64 *pos[axis])

– 功能 :

取得目前命令的位置。

– 描述 :

此函式將讀出使用者下 pulse 命令的值。

– 引數 :

axis : 指的是 X=0 或 Y=1 或 Z=2 軸。

*pos[axis] : 命令的位置值。

– 例子 :

get_position (0, &pos[0]) 將使用者下命令 x 軸移動多少位置的值放入 pos[0] 內。

2.3.4 三軸微步進馬達平台

本運動控制系統主要是控制平面的運動和影像的聚焦運動，所以對於三軸微步進馬達機台而言，需要用到 X 及 Y 和 Z 三軸。如圖 2.16 所示，一組馬達模組包含微步進馬達、機械螺旋組、光隔離感應器及兩條訊號線，一條爲來自驅動器的控制線，一條則是把訊息傳回運動控制卡的訊號回授線。

三個馬達模組的訊號回授線與一條驅動器的控制輸入線，匯集成一條具有 37 pin D-Type 接頭訊號整合線，經由轉接盒，和運動控制卡做聯繫。驅動器藉由具有 37 pin D-Type 接頭之訊號整合線的連接，可接受來自運動控制卡的命令，再將命令轉換成電壓或電流的形式，經由馬達控制線，來控制步進馬達的動作；或將步進馬達的回授量，經由訊號回授和訊號整合線，傳至運

動控制卡做處理，如圖 2.17。

圖 2.18爲三軸微步進馬達平台之馬達訊號回授線的腳位配線顏色標示。表 2.6敘述各腳位所代表的意義。並非所有的腳位都會用到，例如第 3 號和第 4 號腳位都代表 HOME 訊號，卻有 N.O.(normal open) 和 N.C.(normal close) 之分。所謂 N.O. 是指電路在正常情況下，爲開路高阻抗；N.C. 是指電路在正常情況下，爲閉路低阻抗。在此採用第 1 號、第 2 號、第 3 號、第 7 號、第 9 號腳位當作 +5 VCD、接地端、CW limit 及 CCW limit 訊號。每條馬達訊號回授線聯合驅動器控制輸入線匯集成一條 37 pin D-Type 接頭，表 2.7爲 37 pin D-Type 接頭，每一隻接腳的定義說明，並以圖 2.19說明，標示每一組馬達回收訊號及驅動器控制輸入訊號之代表腳位所在。圖 2.20爲 PCI-8134 運動控制卡的 100 pin D-Type 的腳位編號順序和腳位編號的功能說明如附表 2.8，至於 37 pin D-Type 接頭和 100 pin 轉接合之間的訊號連接情形，可參閱表 2.9，在「相對應轉接盒接腳」的欄位中，OUT 和 DIR 代表輸入驅動器的控制訊號；PEL 及 MEL 分別爲 forward limit switch、reverse limit switch，是接收來自馬達回授的平台極限訊號變化；ORG 爲接收平台回歸 home 點的訊號變化；+5V 及 GND 則爲電源的接腳。

表 2.6: 馬達訊號回授線腳位說明 [3]

PIN NO.	DESCRIPTION	WIRE COLOR
1	+5 VDC	RED
2	GROUND	BLACK
3	HOME(N.O.)	GREEN
4	HOME(N.C.)	BROWN
5	KEYING PLUG	
6	CW LIMIT(N.O.)	WHITE
7	CW LIMIT(N.C.)	BLUE
8	CCW LIMIT(N.O.)	YELLOW
9	CCW LIMIT(N.C.)	ORANG

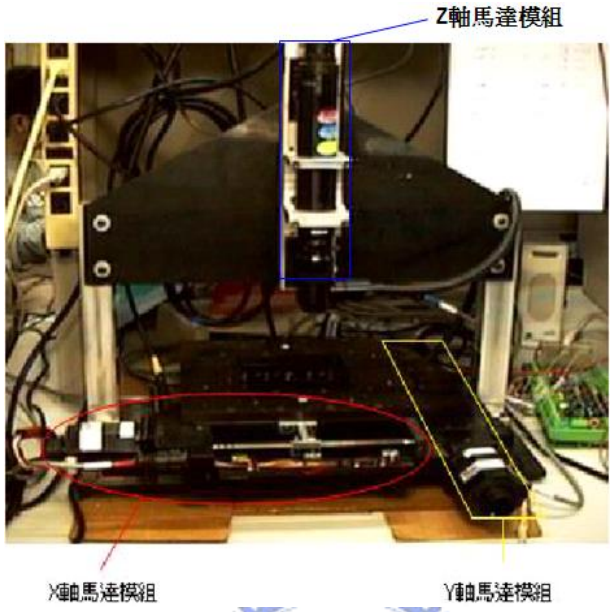


圖 2.16: 三軸微步進馬達平台

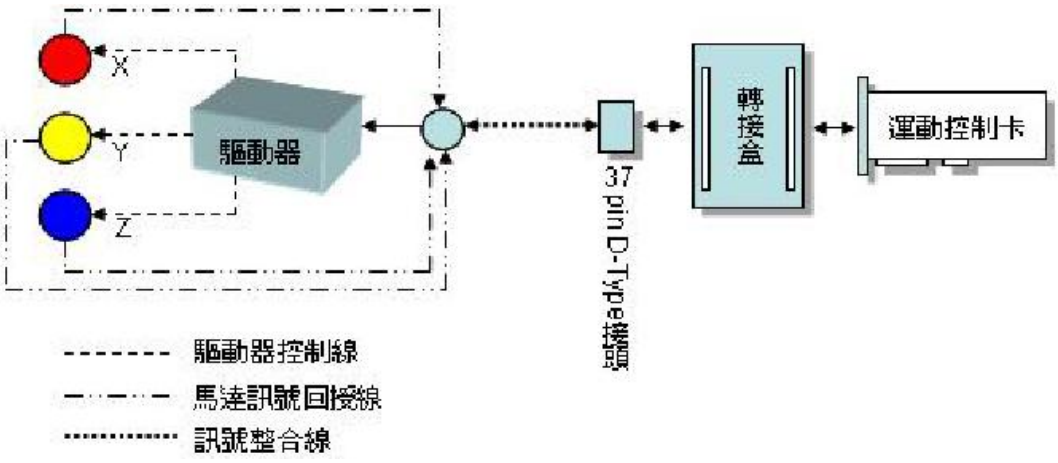


圖 2.17: 訊號線連結 [3]

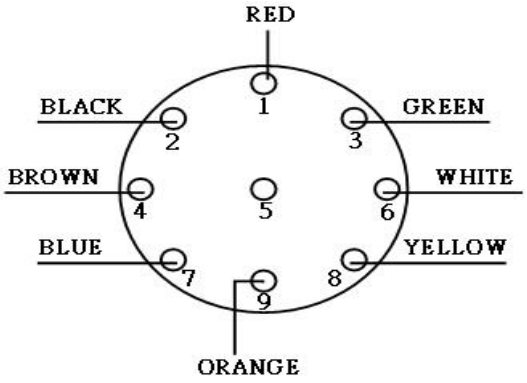


圖 2.18: 馬達訊號回授線腳位配線顏色標明 [3]

表 2.7: 37 pin D-Type 接頭之接腳定義 [3]

PIN NO.	定義	PIN NO.	定義
1	In1	20	In2
2	In3	21	In4
3	In5	22	In6
4	In7	23	In8
5	In9	24	In10
6	In11	25	In12
7	In13	26	In14
8	In15	27	In16
9	In17	28	In18
10	Out1	29	Out2
11	Out3	30	Out4
12	Out5	31	Out6
13	Out7	32	Out8
14	Out9	33	Out10
15	Out11	34	Out12
16	No use	35	No use
17	+5V	36	+5V
18	GND	37	GND
19	GND		

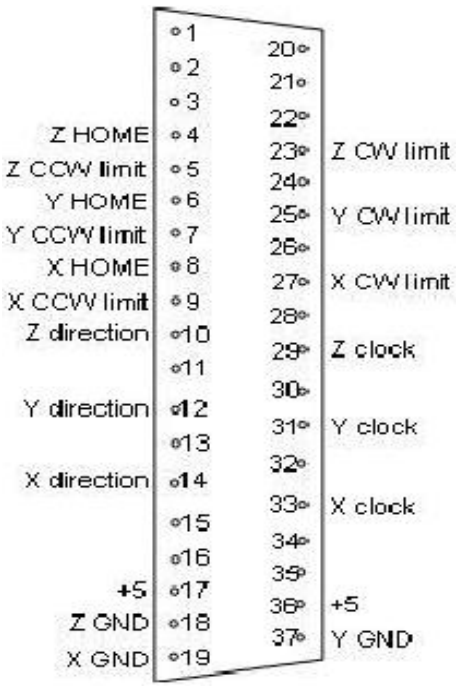


圖 2.19: D-Type 母接頭之腳位相對訊號說明 [3]

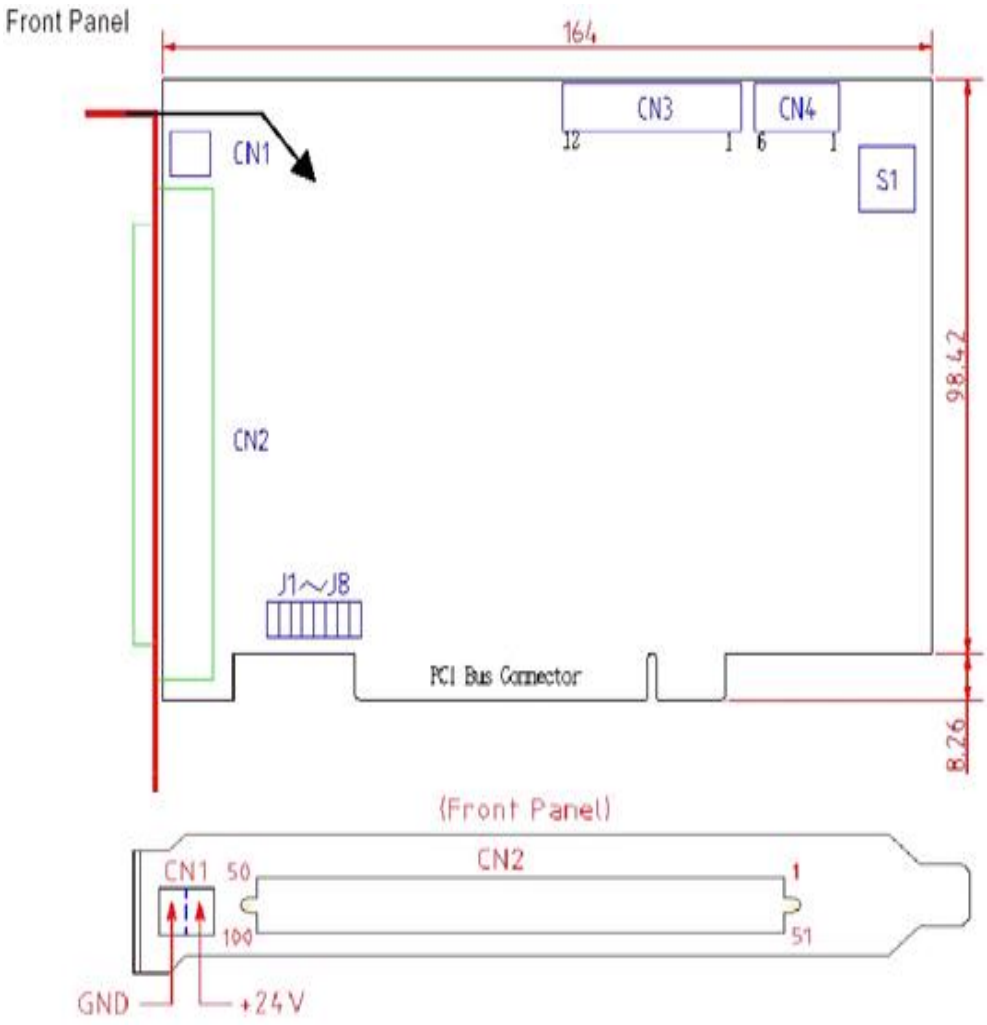


圖 2.20: PCI-8134 運動控制卡 D-Type 腳位編號 [11]

表 2.8: PCI-8134 運動控制卡 D-Type 腳位功能表 [11]

No.	Name	I/O	Function(axis①/②)	No.	Name	I/O	Function(axis③/④)
1	EX+5V	O	+5V power supply output	51	EX+5V	O	+5V power supply output
2	EXGND		Ext. power ground	52	EXGND		Ext. power ground
3	OUT1+	O	Pulse signal (+),①	53	OUT3+	O	Pulse signal (+), ③
4	OUT1-	O	Pulse signal (-),①	54	OUT3-	O	Pulse signal (-),③
5	DIR1+	O	Dir. signal (+),①	55	DIR3+	O	Dir. signal (+), ③
6	DIR1-	O	Dir. signal (-),①	56	DIR3-	O	Dir. signal (-), ③
7	SVON1	O	Multi-purpose signal, ①	57	SVON3	O	Multi-purpose signal, ③
8	ERC1	O	Dev. ctr, dr. signal, ①	58	ERC3	O	Dev. ctr, dr. signal, ③
9	ALM1	I	Alarm signal, ①	59	ALM3	I	Alarm signal, ③
10	INP1	I	In-position signal, ①	60	INP3	I	In-position signal, ③
11	RDY1	I	Multi-purpose signal, ①	61	RDY3	I	Multi-purpose signal, ③
12	EXGND		Ext. power ground	62	EXGND		Ext. power ground
13	EA1+	I	Encoder A-phase (+), ①	63	EA3+	I	Encoder A-phase (+), ③
14	EA1-	I	Encoder A-phase (-), ①	64	EA3-	I	Encoder A-phase (-),③
15	EB1+	I	Encoder B-phase (+), ①	65	EB3+	I	Encoder B-phase (+),③
16	EB1-	I	Encoder B-phase (-), ①	66	EB3-	I	Encoder B-phase (-),③
17	EZ1+	I	Encoder Z-phase (+), ①	67	EZ3+	I	Encoder Z-phase (+),③
18	EZ1-	I	Encoder Z-phase (-), ①	68	EZ3-	I	Encoder Z-phase (-),③
19	EX+5V	O	+5V power supply output	69	EX+5V	O	+5V power supply output
20	EXGND		Ext. power ground	70	EXGND		Ext. power ground
21	OUT2+	O	Pulse signal (+), ②	71	OUT4+	O	Pulse signal (+),④
22	OUT2-	O	Pulse signal (-), ②	72	OUT4-	O	Pulse signal (-),④
23	DIR2+	O	Dir. signal (+), ②	73	DIR4+	O	Dir. signal (+),④
24	DIR2-	O	Dir. signal (-), ②	74	DIR4-	O	Dir. signal (-),④
25	SVON2	O	Multi-purpose signal, ②	75	SVON4	O	Multi-purpose signal, ④
26	ERC2	O	Dev. ctr, dr. signal, ②	76	ERC4	O	Dev. ctr, dr. signal, ④
27	ALM2	I	Alarm signal, ②	77	ALM4	I	Alarm signal, ④
28	INP2	I	In-position signal, ②	78	INP4	I	In-position signal, ④
29	RDY2	I	Multi-purpose signal, ②	79	RDY4	I	Multi-purpose signal, ④
30	EXGND		Ext. power ground	80	EXGND		Ext. power ground
31	EA2+	I	Encoder A-phase (+), ②	81	EA4+	I	Encoder A-phase (+), ④
32	EA2-	I	Encoder A-phase (-), ②	82	EA4-	I	Encoder A-phase (-), ④
33	EB2+	I	Encoder B-phase (+), ②	83	EB4+	I	Encoder B-phase (+), ④
34	EB2-	I	Encoder B-phase (-), ②	84	EB4-	I	Encoder B-phase (-), ④
35	EZ2+	I	Encoder Z-phase (+), ②	85	EZ4+	I	Encoder Z-phase (+), ④
36	EZ2-	I	Encoder Z-phase (-), ②	86	EZ4-	I	Encoder Z-phase (-), ④
37	PEL1	I	End limit signal (+), ①	87	PEL3	I	End limit signal (+), ③
38	MEL1	I	End limit signal (-), ①	88	MEL3	I	End limit signal (-), ③
39	PSD1	I	Ramp-down signal (+), ①	89	PSD3	I	Ramp-down signal (+), ③
40	MSD1	I	Ramp-down signal (-), ①	90	MSD3	I	Ramp-down signal (-), ③
41	ORG1	I	Origin signal, ①	91	ORG3	I	Origin signal, ③
42	EXGND		Ext. power ground	92	EXGND		Ext. power ground
43	PEL2	I	End limit signal (+), ②	93	PEL4	I	End limit signal (+), ④
44	MEL2	I	End limit signal (-), ②	94	MEL4	I	End limit signal (-), ④
45	PSD2	I	Ramp-down signal (+), ②	95	PSD4	I	Ramp-down signal (+), ④
46	MSD2	I	Ramp-down signal (-), ②	96	MSD4	I	Ramp-down signal (-), ④
47	ORG2	I	Origin signal, ②	97	ORG4	I	Origin signal, ④
48	EXGND		Ext. power ground	98	EXGND		Ext. power ground
49	EXGND		Ext. power ground	99	EX+24V	I	Ext. power supply, +24V
50	EXGND		Ext. power ground	100	EX+24V	I	Ext. power supply, +24V

表 2.9: 37 D-Type 接頭和轉接合之間的訊號相對連接

37 D-Type 接頭接腳	相對應轉接盒接腳(CN2)	
Z HOME	No.91	ORG3
Z CCW limit	No.87	PEL3
Y HOME	No.47	ORG2
Y CCW limit	No.43	PEL2
X HOME	No.41	ORG1
X CCW limit	No.37	PEL1
Z direction	No.55	OUT3+
Y direction	No.21	OUT2+
X direction	No.3	OUT1+
+5	No.1	EX+5V
Z GND	No.52	EXGND
X GND	No.2	EXGND
Z CW limit	No.88	MEL3
Y CW limit	No.44	MEL2
X CW limit	No.38	MEL1
Z clock	No.55	DIR3+
Y clock	No.23	DIR2+
X clock	No.5	DIR1+
Y GND	No.20	EXGND

第三章

系統設計

3.1 概述

此章節主要目的是再 PC 端建構一個控制視窗，利用滑鼠或鍵盤下達控制命令，達成運動控制的目的。

●提供功能需求

爲了自動化檢測，因此需做路徑規劃，先將面板約略分割成 N 個區塊，並在每個區塊內作棋盤矩陣的方式作更仔細的檢測。至於將面板分割成幾個區塊則沒有任何規定，在此則以井字型九宮格爲例；對於該區域內的行逕路線，因爲，面板內的每個 Pixel 都是等距的，所以我們則使用規律的弓字型爲其檢測路進。

●實際設計

我們將控制視窗分四個區塊：第一個區塊主要作用爲設定初始參數，決定系統的加速度、減速度、速度、移動間隔、home 點移動和控制平台運動及監控位置，藉由滑鼠或鍵盤發出按下的命令，控制平台的運動，並回傳即時的位置到控制視窗；第二個區塊爲監控從運動控制平台回傳目前伺服馬達

的狀態，給予使用者做爲判別或處理的訊息；第三個區塊主要將面板區分爲井字型共九個區域，且依選擇先後作爲移動的先後；第四個區塊爲棋盤式運動，在選擇區域中以弓字型的運動模式來檢測面板。

3.2 對話盒 CMy3AXIS_CONTROLDlg

在 Visual C++ 的整合性開發環境中，透過 AppWizard 應用程式精靈，建構一個對話盒 CMy3axis_controlDlg 的骨幹架構，並在對話盒編輯環境中，從子控制工具列中拉取所需的元件至對話何適當的位置。

控制視窗的第一個區塊，參考圖 3.1，所包含的子控制元件有靜態物件 Static、編輯 Edit、拖曳棒 Slider、按紐 Button。 Static 作用爲標示加速度、減速度、速度、間距的控制項目；Slider 藉由拖曳位置的改變，控制加速度、減速度、速度及間距的值；Edit 則是藉由內容的輸入或輸出，控制或顯示加速度、減速度、速度、間距及目前位置的值；Button 則主要用來下達移動命令和切換 Slider 拖曳的範圍。

第二區塊，請參考圖 3.2，包含的子控制元件有群組盒 Group Box、靜態物件 static、編輯 Edit。Group Box 將 Static 圈成一個群組，無實質用途；Static 的作用爲標示哪一軸和該軸有哪些狀態；Edit 的作用以顏色來顯示狀態的情況。

第三區塊，請參考圖 3.3，包含的子控制元件有群組盒 Group Box、靜態物件 static、按紐 Button、編輯 Edit、列式盒 List Box。Group Box 將 Static 圈成一個群組，無實質用途；Static 的作用爲標示面板的區域和面板的長和寬；Edit 的作用爲讀取輸入的長和寬的值； Button 的作用在選擇使用者所需移動的區域、一步一步的移動所選擇的區域和無人操作的全自動功能；List Box 爲顯示使用者所選取的區域的順序。

第四區塊，請參考圖 3.4，包含的子控制元件有群組盒 Group Box、靜態物件 static、按紐 Button、編輯 Edit。Group Box 將 Static 圈成一個群組，無實

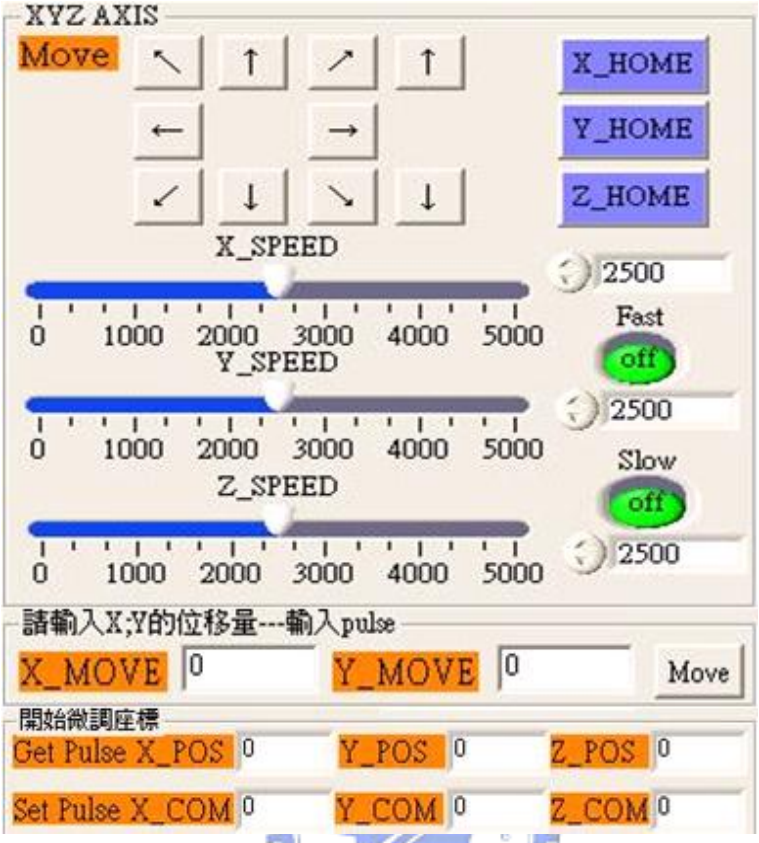


圖 3.1: 控制視窗第一區塊

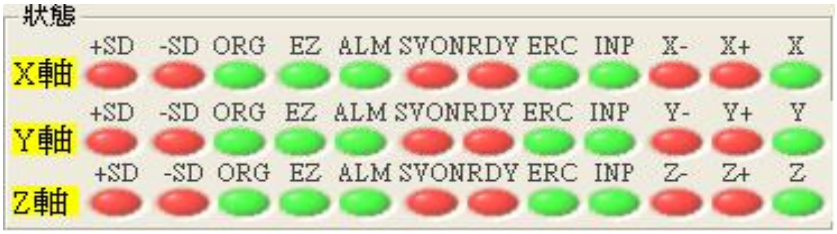


圖 3.2: 控制視窗第二區塊

質用途；Static 的作用爲標示矩陣大小和其間距、相對運動座標、旋轉角度、定位觀測時間的控制項目； Edit 的作用爲輸入弓字形矩陣大小和間距、目前與下一步矩陣位置和相對座標、輸入面板旋轉角度和顯示其選轉後的座標、輸入每一定點觀測的時間；Button 的作用爲以弓字形的軌跡一步一步移動、全自動的作一步一步移動。

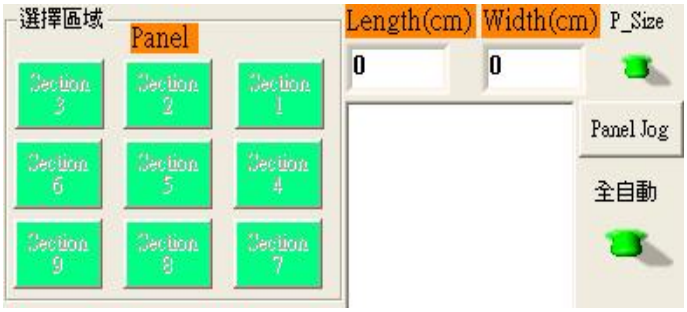


圖 3.3: 控制視窗第三區塊

圖 3.4: 控制視窗第四區塊

運動控制卡相關的 Pci_8134.h 等標頭檔及 8134.lib 檔必須一開始就加入成開發專案中，程式所需的一切參數皆宣告在 3axis_controlDlg.h 檔案的 public 區域中。參數的起始值設定、子控制元件指標的起始化、偵測並建立運動控制卡的溝通等，則交由 3axis_controlDlg.cpp 檔案的 CMy3axis_controlDlg::OnInitDialog() 函式執行。

3.3　參數設定與起始化

在 3axis_controlDlg.h 檔案中的 public 區域中，有個宣告爲以下所列 :

```
// 宣告 ThreadControl
////////////////////////////////////////////////////////////////////////////
```

```
CWinThread *m_pThread[4];//4 個 Threads
bool Continue[4];
static UINT ThreadFun1(LPVOID lParam);
static UINT ThreadFun2(LPVOID lParam);
static UINT ThreadFun3(LPVOID lParam);
static UINT ThreadFun4(LPVOID lParam);
void OnThreadExit();
void AllThreadTurnOff();

struct THREAD_INFO
{
bool* Continue;
int CtrlID1, CtrlID2;
HWND hWnd;
}Thread_Info[5];
////////////////////////////////////////////////////////////////////////

// 宣告一些副程式
////////////////////////////////////////////////////////////////////////
void MotionCardInitial(); // 啓動運動控制卡的起始化函式
void MoveStopAll(); // 停止所有運動的函式
void StartInitial(); // 程式所需的初值設定的函式
void DisableXyzAxis(); // 將 X 和 Y 軸的相關元件 Disable
void EnableXyzAxis(); // 將 X 和 Y 軸的相關元件 Enable
////////////////////////////////////////////////////////////////////////
```

爲了往後寫程式的方便，因此我們宣告了屬於 Edit、Button、List Box 和 Slider 的元件指標於 DoDataExchange() 函式內，用來指向控制視窗上所有的子控制元件，利用 DDX_Control() 函式取得該元件控制權 :

DDX_Control(pDX, IDC_CWSLIDE_X_SLIDER, m_Xslider);
DDX_Control(pDX, IDC_COMMANDBUTTON_BACK, m_Back);
DDX_Control(pDX, IDC_COMMANDBUTTON_DOWN, m_Down);
DDX_Control(pDX, IDC_COMMANDBUTTON_FRONT, m_Front);
DDX_Control(pDX, IDC_COMMANDBUTTON_LEFT, m_Left);
DDX_Control(pDX, IDC_COMMANDBUTTON_LEFTBACK, m_Leftback);
DDX_Control(pDX, IDC_COMMANDBUTTON_LEFTFRONT, m_Leftfront);
DDX_Control(pDX, IDC_COMMANDBUTTON_RIGHT, m_Right);
DDX_Control(pDX, IDC_COMMANDBUTTON_RIGHTBACK, m_Rightback);
DDX_Control(pDX, IDC_COMMANDBUTTON_RIGHTFRONT, m_Rightfront);
DDX_Control(pDX, IDC_COMMANDBUTTON_UP, m_Up);
DDX_Control(pDX, IDC_COMMANDBUTTON_XHOME, m_Xhome);
DDX_Control(pDX, IDC_COMMANDBUTTON_YHOME, m_Yhome);
DDX_Control(pDX, IDC_COMMANDBUTTON_ZHOME, m_Zhome);
DDX_Control(pDX, IDC_CWSLIDE_Y_SLIDER, m_Yslider);
DDX_Control(pDX, IDC_CWSLIDE_Z_SLIDER, m_Zslider);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION1, m_section1);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION2, m_section2);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION3, m_section3);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION4, m_section4);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION5, m_section5);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION6, m_section6);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION7, m_section7);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION8, m_section8);
DDX_Control(pDX, IDC_TOGGLEBUTTON_SECTION9, m_section9);
DDX_Control(pDX, IDC_COMMANDBUTTON_XY_MOVE, m_Xymove);
DDX_Control(pDX, IDC_CWBOOLEAN_FAST, m_Fast);
DDX_Control(pDX, IDC_CWBOOLEAN_SLOW, m_Slow);
DDX_Control(pDX, IDC_CWNUMEDIT_X_SPEEDTEXT, m_Xspeedtext);
DDX_Control(pDX, IDC_CWNUMEDIT_Y_SPEEDTEXT, m_Yspeedtext);

DDX_Control(pDX, IDC_CWNUMEDIT_Z_SPEEDTEXT, m_Zspeedtext);

DDX_Control(pDX, IDC_CWNUMEDIT_X_COMMAND, m_Xcommandtext);

DDX_Control(pDX, IDC_CWNUMEDIT_X_POSITION, m_Xpositiontext);

DDX_Control(pDX, IDC_CWNUMEDIT_Y_COMMAND, m_Ycommandtext);

DDX_Control(pDX, IDC_CWNUMEDIT_Y_POSITION, m_Ypositiontext);

DDX_Control(pDX, IDC_CWNUMEDIT_Z_COMMAND, m_Zcommandtext);

DDX_Control(pDX, IDC_CWNUMEDIT_Z_POSITION, m_Zpositiontext);

DDX_Control(pDX, IDC_CWNUMEDIT_Z_FOCUSTEXT, m_Zfocustext);

DDX_Control(pDX, IDC_COMMANDBUTTON_XY_POS_RESET, m_Xypos_reset);

DDX_Control(pDX, IDC_COMMANDBUTTON_Z_FOCUS, m_Zfocus);

DDX_Control(pDX, IDC_CWBOOLEAN_X_ALM, m_Xalm);

DDX_Control(pDX, IDC_CWBOOLEAN_X_ERC, m_Xerc);

DDX_Control(pDX, IDC_CWBOOLEAN_X_EZ, m_Xez);

DDX_Control(pDX, IDC_CWBOOLEAN_X_INP, m_Xinp);

DDX_Control(pDX, IDC_CWBOOLEAN_X_LEFT, m_Xleft);

DDX_Control(pDX, IDC_CWBOOLEAN_X_ORG, m_Xorg);

DDX_Control(pDX, IDC_CWBOOLEAN_X_ORG1, m_Xorg1);

DDX_Control(pDX, IDC_CWBOOLEAN_X_RDY, m_Xrdy);

DDX_Control(pDX, IDC_CWBOOLEAN_X_RIGHT, m_Xright);

DDX_Control(pDX, IDC_CWBOOLEAN_X_SD1, m_Xsd1);

DDX_Control(pDX, IDC_CWBOOLEAN_X_SD2, m_Xsd2);

DDX_Control(pDX, IDC_CWBOOLEAN_X_SVON, m_Xsvon);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_ALM, m_Yalm);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_ERC, m_Yerc);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_EZ, m_Yez);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_INP, m_Yinp);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_LEFT, m_Yleft);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_ORG, m_Yorg);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_ORG1, m_Yorg1);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_RDY, m_Yrdy);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_RIGHT, m_Yright);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_SD2, m_Ysd2);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_SVON, m_Ysvon);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_ALM, m_Zalm);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_ERC, m_Zerc);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_EZ, m_Zez);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_INP, m_Zinp);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_LEFT, m_Zleft);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_ORG, m_Zorg);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_ORG1, m_Zorg1);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_RDY, m_Zrdy);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_RIGHT, m_Zright);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_SD1, m_Zsd1);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_SD2, m_Zsd2);

DDX_Control(pDX, IDC_CWBOOLEAN_Z_SVON, m_Zsvon);

DDX_Control(pDX, IDC_CWBOOLEAN_Y_SD1, m_Ysd1);

DDX_Control(pDX, IDC_COMMANDBUTTON_JOG, m_Jog);

DDX_Control(pDX, IDC_CWNUMEDIT_SETUP_ROW, m_Setuprow);

DDX_Control(pDX, IDC_CWNUMEDIT_SETUP_COLUM_DISTANCE, m_SetColDis);

DDX_Control(pDX, IDC_CWNUMEDIT_SETUP_ROW_DISTANCE, m_SetRowDis);

DDX_Control(pDX, IDC_CWNUMEDIT_SETUP_COLUM, m_Setupcolum);

DDX_Control(pDX, IDC_CWNUMEDIT_RATATION_NOW_V, m_Ratation_Nowv);

DDX_Control(pDX, IDC_CWNUMEDIT_RATATION_NOW_U, m_Ratation_Nowu);

DDX_Control(pDX, IDC_CWNUMEDIT_RATATION_NEXT_V, m_Ratation_Nextv);

DDX_Control(pDX, IDC_CWNUMEDIT_RATATION_NEXT_U, m_Ratation_Nextu);

DDX_Control(pDX, IDC_CWNUMEDIT_RATATION_ANGLE, m_Ratation_Ang);

DDX_Control(pDX, IDC_CWNUMEDIT_NOW_Y_AXIS, m_Now_Yaxis);

DDX_Control(pDX, IDC_CWNUMEDIT_NOW_X_AXIS, m_Now_Xaxis);

DDX_Control(pDX, IDC_CWNUMEDIT_NOW_ROW, m_Nowrow);

DDX_Control(pDX, IDC_CWNUMEDIT_NOW_COLUM, m_Nowcolum);

DDX_Control(pDX, IDC_CWNUMEDIT_NEXT_Y_AXIS, m_Next_Yaxis);

DDX_Control(pDX, IDC_CWNUMEDIT_NEXT_X_AXIS, m_Next_Xaxis);

DDX_Control(pDX, IDC_CWNUMEDIT_NEXT_ROW, m_Nextrow);

DDX_Control(pDX, IDC_CWNUMEDIT_NEXT_COLUM, m_Nextcol);

DDX_Control(pDX, IDC_CWBOOLEAN_CHECKER_INPUT_ONOFF, m_Checker_Input);

DDX_Control(pDX, IDC_CWNUMEDIT_VIEW_TIMER, m_View_Timer);

DDX_Control(pDX, IDC_CWNUMEDIT_X_SAMPLE_COORD, m_Xsample_Coordtext);

DDX_Control(pDX, IDC_CWNUMEDIT_Y_SAMPLE_COORD, m_Ysample_Coordtext);

DDX_Control(pDX, IDC_CWNUMEDIT_PANEL_LENGTH, m_Panel_Length);

DDX_Control(pDX, IDC_CWNUMEDIT_PANEL_WIDTH, m_Panel_Width);

DDX_Control(pDX, IDC_COMMANDBUTTON_PANEL_JOG, m_Panel_Jog);

DDX_Control(pDX, IDC_CWBOOLEAN_CKECKER_AUTO, m_Checker_Auto);

DDX_Control(pDX, IDC_TEXTBOX_MOVE_DONE, m_movedone);

當系統執行起始化時，OnInitDialog() 函式一開頭，便是要偵測運動控制卡的存在並取得該卡的控制權，MotionCardInitial() 函式爲啓動控制卡的功能；StartInitial() 函式則設定程式起始和各子控制所需要初值設定 :

```
void CMy3axis_controlDlg::MotionCardInitial()
{
// 偵測 PC 上有多少張控制卡，且當找到了卡，則給予該卡一個位置，並透過 W_8134_InitialA() 函式來啓動控制卡
//////////////////////////////////////////////////////////////////////////
I16 Err,i;
for(int j=0;j < MaxCards;j++)
{
Err=W_8134_InitialA(j);
if( Err==0)
TotalCards++;
}
W_8134_InitialA(&TotalCards);
```

```
if( TotalCards == 0 )
{
AfxMessageBox("No Cards Found!");
exit(0);
}
////////////////////////////////////////////////////////////////////////////

// 透過 W_8134_Set_Config() 函式，讀取 8134.cfg 內容來設定控制卡的狀態
////////////////////////////////////////////////////////////////////////////
FStr+="8134.cfg";
Err=W_8134_Set_Config(FStr.GetBuffer(0));
if( Err )
AfxMessageBox("Can't File in 8134.cfg");
////////////////////////////////////////////////////////////////////////////

// 利用 W_8134_INT_Enable() 函式，致能每一軸中斷功能；利用 W_8134_Set_INT_Control() 函
式，設定控制中斷權力
////////////////////////////////////////////////////////////////////////////
for(i=0;i < TotalCards;i++)
{
W_8134_INT_Enable(i,&hEvent[4*i]);
W_8134_Set_INT_Control(i,TRUE);
}
////////////////////////////////////////////////////////////////////////////

// 處理每一軸的訊息排程
////////////////////////////////////////////////////////////////////////////
UINT IntThreadProc( LPVOID lpParam )
{
int n;
n=(int)lpParam;
```

```
while( ISR_ON )
{
WaitForSingleObject(hEvent[n],INFINITE);
ResetEvent(hEvent[n]);
int_counter[n]++;
get_int_status(n,&int_status[n]); }
return TRUE;
}

void DoEvents(void)
{
MSG message;
while( :: PeekMessage(&message,NULL,0,0,PM_REMOVE) )
{
::TranslateMessage(&message);
::DispatchMessage(&message);
}
}

ISR_ON=1;
for(i=0;i < TotalAxis;i++)
{
pThread[i]=AfxBeginThread(IntThreadProc,(LPVOID)i,THREAD_PRIORITY_NORMAL,0,CREATE_S
ASSERT( pThread[i] );
pThread[i]- > m_bAutoDelete=FALSE;
pThread[i]- > ResumeThread();
int_counter[i]=0;
}
////////////////////////////////////////////////////////////////////////////
```

Slider 拖曳的每一個點位置代表不同的加速度、減速度、速度，因此在初始化過程中，利用 Slider 的成員函式 GetAxis() 設定最大及最小值，並決定三個軸的 Slider 的起始值爲藉於最大與最小值的中間，接著利用 Slider 成員函式 SetValue()，移動拖曳棒至初值位置，利用 Edit 的成員函式 SetValue() 顯示拖曳棒初值的值 :

```
void CMy3axis_controlDlg::StartInitial()
{
maxspeed=5000; // 爲最大值
minspeed=0;// 爲最小值
m_Xslider.GetAxis().Maximum = maxspeed;
m_Xslider.GetAxis().Minimum = minspeed;
m_Yslider.GetAxis().Maximum = maxspeed;
m_Yslider.GetAxis().Minimum = minspeed;
m_Zslider.GetAxis().Maximum = maxspeed;
m_Zslider.GetAxis().Minimum = minspeed;
xspeedbuffer = (minspeed+maxspeed)/2;
yspeedbuffer = (minspeed+maxspeed)/2;
zspeedbuffer = (minspeed+maxspeed)/2;
m_Xspeedtext.SetValue(xspeedbuffer);
m_Yspeedtext.SetValue(yspeedbuffer);
m_Zspeedtext.SetValue(zspeedbuffer);
xsliderbuffer = m_Xspeedtext.GetValue();
ysliderbuffer = m_Yspeedtext.GetValue();
zsliderbuffer = m_Zspeedtext.GetValue();
m_Xslider.Pointers.Item(1).SetValue(xsliderbuffer);
m_Yslider.Pointers.Item(1).SetValue(ysliderbuffer);
m_Zslider.Pointers.Item(1).SetValue(zsliderbuffer);
}
```

3.4 控制視窗區塊 I

此區塊的主要功能在於藉由 Slider 的拖曳、Edit 的輸入內容及 Button 的觸發，改變加速度、減速度、速度、間距及移動方向五個參數值，以下將各個部份分別介紹。對於 Slider 和 Edit 兩個元件的動作必須一致同步的。當 Slider 拖曳改變位置時，觸發 OnPointerValueChangedCwslide()，取得新的拖曳位置的數值，透過 v_change() 函式輸入運動控制卡，同時將新數值輸出至 Edit 內容中 :

```
// 取得新的 Slider 位置的數值
xspeedbuffer = m_Xslider.Pointers.Item(1).GetValue();
// 輸出至 Edit 顯示
m_Xspeedtext.SetValue(xspeedbuffer);
// 將新的數值輸出至運動控制卡，改變速度 v_change(AxisNoX,xspeedbuffer,0.1);
```

當 Edit 內容改變時，觸發 OnValueChangedCwnumedit()，新的內容將轉成 int 資料型態，傳給 Slider，並轉成相對應的指令列給運動控制卡，不過必須注意的是，輸入的內容可能超過最大值或小於最小值，所以需加入適當的判斷機制，以防止系統發生錯誤反應 :

```
// 將新的輸入值傳給 int 資料型態
xsliderbuffer = m_Xspeedtext.GetValue();
// 若超過最大值
if(xsliderbuffer > maxspeed)
{
// 警告視窗
MessageBox("Out of range!","ERROR!!",MB_ICONSTOP|MB_OK);
// 設新數值爲最大值
m_Xslider.Pointers.Item(1).SetValue(maxspeed);
```

```
xspeedbuffer = maxspeed;
m_Xspeedtext.SetValue(xspeedbuffer);
}
// 若小於最小值
else if(xsliderbuffer < minspeed)
{
// 警告視窗
MessageBox("Out of range!","ERROR!!",MB_ICONSTOP|MB_OK);
// 設新數值爲最小值
m_Xslider.Pointers.Item(1).SetValue(minspeed);
xspeedbuffer = minspeed;
m_Xspeedtext.SetValue(xspeedbuffer);
}
// 若無超出極限，則以輸入內容爲新數值
else
{
m_Xslider.Pointers.Item(1).SetValue(xsliderbuffer);
}
// 將新的數值傳至運動控制卡
xsliderbuffer = m_Xspeedtext.GetValue();
v_change(AxisNoX,xsliderbuffer,0.1);
```

當 Move Button 主要目的在於操控運動平台，依照滑鼠或鍵盤發出的訊息，進行定位的移動。在定位按鈕按下對應的訊息處理函式 OnClickCommandbutton() 中，將輸入定位點的值轉換成字串陣列，並透過 start_r_move() 函式輸入運動控制卡 :

```
// 將取得 X 軸輸入的定位點的值傳給 CString 資料型態
m_Xmovetext- > GetWindowText(xymovebuffer);
// 將字串轉成數值，輸出運動控制卡作 X 軸定位移動
start_r_move(AxisNoX,atoi(xymovebuffer),0,xsliderbuffer,0.1);
```

```
// 將取得 Y 軸輸入的定位點的值傳給 CString 資料型態
m_Ymovetext- > GetWindowText(xymovebuffer);
// 將字串轉成數值，輸出運動控制卡作 Y 軸定位移動
start_r_move(AxisNoY,atoi(xymovebuffer),0,ysliderbuffer,0.1);
```

當程式開啓時，則啓動 OnTimer() 函式，以每 1ms 的時間透過 get_position() 函式來回傳目前座標和 get_command() 函式來取得命令給控制卡的移動位置值，輸出於視窗的 Edit 顯示 :

```
// 取得 X、Y 和 Z 軸的目前座標
get_position(AxisNoX,&pos[AxisNoX]);
get_position(AxisNoY,&pos[AxisNoY]);
get_position(AxisNoZ,&pos[AxisNoZ]);
// 輸出至控制視窗 Edit 顯示
m_Xpositiontext.SetValue(pos[AxisNoX]);
m_Ypositiontext.SetValue(pos[AxisNoY]);
m_Zpositiontext.SetValue(pos[AxisNoZ]);
// 讀取送給馬達的位置 (pulse)
get_command(AxisNoX,&pos[AxisNoX]);
get_command(AxisNoY,&pos[AxisNoY]);
get_command(AxisNoZ,&pos[AxisNoZ]);
// 輸出至控制視窗 Edit 顯示
m_Xcommandtext.SetValue(pos[AxisNoX]);
m_Ycommandtext.SetValue(pos[AxisNoY]);
m_Zcommandtext.SetValue(pos[AxisNoZ]);
```

當方向性 Button 主要目的在於操控運動平台，依照滑鼠或鍵盤發出的訊息，控制平台方向、瞬間停止行進運動及回歸原點 (home 點)。十個方向控制分爲 : 左、又、前、後、左前、左後、右前、右後及上、下，其處理函式架構大致相同，在此以「左」處理函式爲例子說明。在「左」按鈕按下對應的訊息處理函式 OnMouseDownCommandbutton() 中，程式會透過 v_move() 函

式發出開始運動命令給運動控制卡 :

// 取得目前速度的數值傳給 int 資料型態

xsliderbuffer = m_Xspeedtext.GetValue();

// 發出移動相關指令給運動控制卡

v_move(AxisNoX,0,-zsliderbuffer,0.1);

在放開按鈕對應的訊息處理函式 OnMouseUpCommandbutton() 中，透過 v_stop() 函式輸入運動控制卡 :

// 發出停止命令給運動控制卡

v_stop(AxisNoX,0);

有時平台反覆進行相同動作時，需回歸至原點，在繼續執行相同的動作，所以當「HOME」鈕按下時，以 X 軸爲例，相對應的訊息處理函式 OnClickCommandbuttonhome() 便會進行回歸運動，參考圖 3.5爲運動控制平台的 home sensor 位置示意圖 :

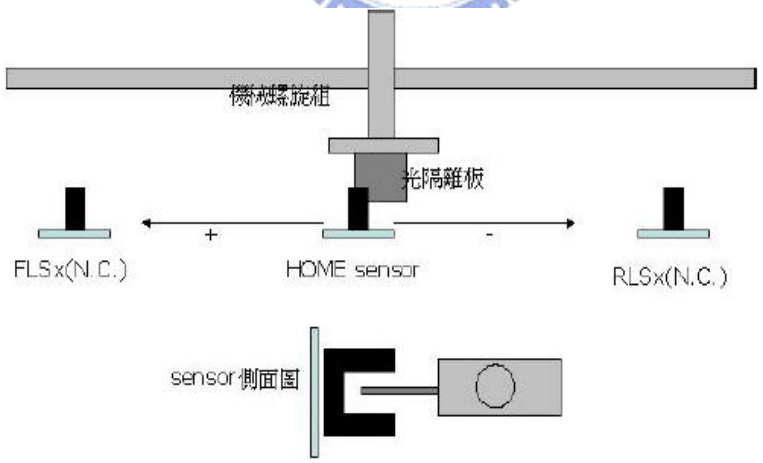


圖 3.5: 感應器位置示意圖 [3]

// 設定 home 點回歸運動的條件

set_home_config(AxisNoX,2,1,1,0);

// 取得目前的位置作爲判斷 home 點移動方向

xhomebuffer = m_Xpositiontext.GetValue();

```
// 若位置大於 0
if (xhomebuffer > 0)
{
// 反向移動，直到滿足 home 點的設定
home_move(AxisNoX,0,-xsliderbuffer,0.2);
}
else
{
// 反向移動，直到滿足 home 點的設定
home_move(AxisNoX,0,xsliderbuffer,0.2);
}
```

3.5 控制視窗區塊 II

此區塊的主要功能在於藉由 Edit 的變色顯示目前運動控制卡所回授的狀態，在程式一啓動時則觸發 OnTimer() 函式，以每 1ms 的時間透過 get_io_status() 函式取得三軸所有回傳值和我們給定的檢查碼作 and 的結果以顏色輸出至 Edit，表 3.1爲三軸所有的十六位元的狀態回傳碼、檢查碼、and 結果和狀態說明對照表；程式則以 X 軸爲例子。

```
 // 取得回傳值存爲指標型態
get_io_status(AxisNoX,&io_status[AxisNoX]);
//+X 狀態
if( (io_status[AxisNoX] & 0x2) == 0x2)
// 顯示綠色
m_Xright.SetOffColor(RGB(255,0,0));
else
// 顯示紅色
m_Xright.SetOffColor(RGB(0,255,0));
```

表 3.1: 各狀態回傳碼、說明表

名稱	get_io_status	檢查碼	And 結果	說明
+SD	0x4	0x4	0x4	(右) 爲了防止高速停止，因而需要降速
-SD	0x8	0x8	0x8	(左) 爲了防止高速停止，因而需要降速
ORG	0x10	0x10	0x10	用於 home 點是否達到
EZ	0x20	0x20	0x20	用於 zero position index(編碼器標誌信號輸入)
ALM	0x40	0x40	0x40	伺服驅動器的狀態
SVON	0x80	0x80	0x80	用於偵測伺服馬達是否爲激磁
RDY	0x100	0x100	0x100	用於偵測馬達驅動器是否準備好了
ERC	0x400	0x400	0x400	用於 clear the deviation counter of servomotor driver
INP	0x800	0x800	0x800	Indicate the deviation error is zero
X-	0x1	0x1	0x1	左極限是否到達
X+	0x2	0x2	0x2	右極限是否到達
INT	0x200	0x200	0x200	外部中斷是否發生

```
//-X 狀態
if( (io_status[AxisNoX] & 0x1) == 0x1)
m_Xleft.SetOffColor(RGB(255,0,0));
else
m_Xleft.SetOffColor(RGB(0,255,0));
//home 點狀態
if( (io_status[AxisNoX] & 0x10) == 0x10)
m_Xorg.SetOffColor(RGB(255,0,0));
else
m_Xorg.SetOffColor(RGB(0,255,0));
//-SD 狀態
if( (io_status[AxisNoX] & 0x4) == 0x4)
m_Xsd1.SetOffColor(RGB(255,0,0));
else
```

```
m_Xsd1.SetOffColor(RGB(0,255,0));
//-SD 狀態
if( (io_status[AxisNoX] & 0x8) == 0x8)
m_Xsd2.SetOffColor(RGB(255,0,0));
else
m_Xsd2.SetOffColor(RGB(0,255,0));
//EZ 狀態
if( (io_status[AxisNoX] & 0x20) == 0x20)
m_Xez.SetOffColor(RGB(255,0,0));
else
m_Xez.SetOffColor(RGB(0,255,0));
//ALM 狀態
if( (io_status[AxisNoX] & 0x40) == 0x40)
m_Xalm.SetOffColor(RGB(255,0,0));
else
m_Xalm.SetOffColor(RGB(0,255,0));
//SVON 狀態
if( (io_status[AxisNoX] & 0x80) == 0x80)
m_Xsvon.SetOffColor(RGB(255,0,0));
else
m_Xsvon.SetOffColor(RGB(0,255,0));
//RDY 狀態
if( (io_status[AxisNoX] & 0x100) == 0x100)
m_Xrdy.SetOffColor(RGB(255,0,0));
else
m_Xrdy.SetOffColor(RGB(0,255,0));
//INT 狀態
if( (io_status[AxisNoX] & 0x200) == 0x200)
m_Xint.SetOffColor(RGB(255,0,0));
else
m_Xint.SetOffColor(RGB(0,255,0));
```

```
//ERC 狀態
if( (io_status[AxisNoX] & 0x400) == 0x400)
m_Xerc.SetOffColor(RGB(255,0,0));
else
m_Xerc.SetOffColor(RGB(0,255,0));
//INP 狀態
if( (io_status[AxisNoX] & 0x800) == 0x800)
m_Xinp.SetOffColor(RGB(255,0,0));
else
m_Xinp.SetOffColor(RGB(0,255,0))
```

3.6 控制視窗區塊 III

此區塊的主要功能在於依照滑鼠或鍵盤發出的訊息，控制平台運動的方式。因此我們將面板區分成井字型九宮格，當面版的長與寬經由鍵盤輸入後，若使用者「按下」所選擇的區域時，則觸發 OnChangeTogglebutton() 函式，將選擇的區域的位置算出，分別呈現在 List Box，且由上而下分別給各區域一個 Index 來代表其編號，而此編號的用意在於先後處理的順序，此順序由 Index 0 到 Index 9，而 Index 0 爲最高優先權以此類推至 Index 9；當使用者「釋放」所選擇的區域時，觸發 OnChangeTogglebutton() 函式，去尋找 List Box 中釋放區域的編號，並把其名稱從 List Box 刪除，且將目前在 List Box 中的名單再重新給予編號。當已完成選擇後，點選 Panel Jog Button 時，觸發 OnClick-Commandbutton() 函式，將從 List Box 內從 Index 爲 0 開始依序把內容取出成字串，再將該字串的位置值轉成 int 資料型態，結合移動的指令傳給運動控制卡：

```
// 宣告 List Box
CListBox* m_section = static_cast < CListBox* > (GetDlgItem(IDC_LIST_PANEL_LIST));
// 算出目前在 List Box 中有幾筆資料存爲 int 資料型態
section_max = m_section- > GetCount();
```

```
// 判斷目前讀取的 Index 是否爲最後一筆
if (section_index == section_max)
{
// 將目前 Index 累加 1
section_index++;
}
若目前的 Index 在可讀取的範圍內
if((section_max != 0) && (section_index < section_max) )
{
// 取出 Index 的內容
m_section- > GetText(section_index,list_context);
// 顯示目前 Index 的位置
m_section- > SetCurSel(section_index);
// 讀取該 Index 內容的區域名稱後的有效移動的數值
////////////////////////////////////////////////////////////////////////
for(i=0;i < 9;i++)
{
memcpy(section_name[i]+8,list_context+8,20);
if(!memcmp(section_name[i],list_context,28))
{
index = i+1;
}
}
////////////////////////////////////////////////////////////////////////
// 計算移動量值
////////////////////////////////////////////////////////////////////////
if(section_index == 0)
{
panel_x_move = section_length[index];
panel_y_move = section_width[index];
j=index;
```

```
}
else
{
panel_x_move = section_length[index]-section_length[j];
panel_y_move = section_width[index]-section_width[j];
j=index;
}
// 計算旋轉後的移動位置
ratation_panel_u_move=(panel_x_move)*cos(-ratation_angle)+(panel_y_move)*sin(-ratation_angle);
ratation_panel_v_move=-(panel_x_move)*sin(-ratation_angle)+(panel_y_move)*cos(-ratation_angle);
////////////////////////////////////////////////////////////////////////////
// 四捨五入
////////////////////////////////////////////////////////////////////////////
if (ratation_panel_u_move > = 0)
panel_u_move = ratation_panel_u_move+0.501;
else
panel_u_move = ratation_panel_u_move-0.501;
if (ratation_panel_v_move > = 0)
panel_v_move = ratation_panel_v_move+0.501;
else
panel_v_move = ratation_panel_v_move-0.501;
////////////////////////////////////////////////////////////////////////////
// 將最後處理完成的移動量，透過 start_r_move() 函式傳給運動控制卡
start_r_move(AxisNoX,panel_u_move,0,xsliderbuffer,0.1);
start_r_move(AxisNoY,panel_v_move,0,ysliderbuffer,0.1);
```

3.7 控制視窗區塊 IV

此區塊與視窗 III 有關係，其主要功能用於定位於該區塊內，以棋盤格式的弓字型的運動方式在平台上移動與檢測定點的影像。藉由 Edit 的輸入棋盤矩陣的大小、間距和觀測時間以秒爲單位，將輸入的值經過 Button click 所觸發的 OnClickCommandbutton() 函式執行後，得知下一步和目前的矩陣與座標、旋轉後的座標，必須注意的是此程式有防止矩陣輸入爲 0x0 或是負值。

```
// 計算下一步矩陣的程式
////////////////////////////////////////////////////////////////////////////
// 設定下一步矩陣爲 1x1
if(next_matrix_row > c_rowbuffer)
{
next_matrix_row=1;
next_matrix_col=1;
}
// 弓字型程式
////////////////////////////////////
// 若不爲 2 所整除，則爲 colum +1
if((next_matrix_row)%2 != 0)
{
m_Nextrow.SetValue(next_matrix_row);
m_Nextcol.SetValue(next_matrix_col);
next_matrix_col++;
}
// 若爲 2 所整除，則爲 colum -1
if((next_matrix_row)%2 == 0)
{
m_Nextrow.SetValue(next_matrix_row);
next_matrix_col–;
m_Nextcol.SetValue(next_matrix_col);
}
```

```
// 若 Colum 大於設定值或爲 1
if(next_matrix_col > c_columbuffer || next_matrix_col==1)
{
//Row 的值 +1
next_matrix_row++;
}
////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
// 計算下一步座標的程式
////////////////////////////////////////////////////////////////////////////
// 設定下一步座標爲 [0,0]
if(next_axis_row > (c_rowbuffer-1))
{
next_axis_row=0;
next_axis_col=0;
}
// 弓字型程式
////////////////////////////////////
// 若爲 2 所整除，則爲 x 座標 +1
if((next_axis_row+1)%2 != 0)
{
// 將計算結果乘上一個 pulse 與公分的換算單位，輸出於 Edit
m_Next_Xaxis.SetValue(next_axis_row*c_row_d_buffer*cm_To_pulse);
m_Next_Yaxis.SetValue(next_axis_col*c_col_d_buffer*cm_To_pulse);
next_axis_col++;
}
// 若爲 2 所整除，則爲 x 座標 -1
if((next_axis_row+1)%2 == 0)
{
// 將計算結果乘上一個 pulse 與公分的換算單位，輸出於 Edit
m_Next_Xaxis.SetValue(next_axis_row*c_row_d_buffer*cm_To_pulse);
```

```
next_axis_col–;
m_Next_Yaxis.SetValue(next_axis_col*c_col_d_buffer*cm_To_pulse);
}
if(next_axis_col > (c_columbuffer-1) | | next_axis_col==0)
{
//Y 座標 +1
next_axis_row++;
}
////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
if (delay_one_sec == true)
{
// 計算棋盤旋轉後 X 軸的移動量
checker_u_move = carry_spin_next_u - carry_spin_now_u;
// 計算棋盤旋轉後 Y 軸的移動量
checker_v_move = carry_spin_next_v - carry_spin_now_v;
}
////////////////////////////////////////////////////////////////////////////

if(delay_one_sec == true)
{
// 計算目前矩陣的程式
////////////////////////////////////////////////////////////////////////////
{ 同上 ....}
////////////////////////////////////////////////////////////////////////////
// 計算目前座標的程式
////////////////////////////////////////////////////////////////////////////
{ 同上 ....}
////////////////////////////////////////////////////////////////////////////
```

```
}

// 取得旋轉後的座標
////////////////////////////////////////////////////////////////////////////
// 取得輸入的旋轉角度
ratation_angle = m_Ratation_Ang.GetValue();
// 取得目前座標
now_v = m_Now_Xaxis.GetValue();
now_u = m_Now_Yaxis.GetValue();
// 取得下一步座標
next_v = m_Next_Xaxis.GetValue();
next_u = m_Next_Yaxis.GetValue();
////////////////////////////////////////////////////////////////////////////

// 算出目前的旋轉座標
////////////////////////////////////////////////////////////////////////////
ratation_now_u=(now_u)*cos(-ratation_angle)+(now_v)*sin(-ratation_angle);
m_Ratation_Nowu.SetValue(ratation_now_u);
ratation_now_v=-(now_u)*sin(-ratation_angle)+(now_v)*cos(-ratation_angle);
m_Ratation_Nowv.SetValue(ratation_now_v);
////////////////////////////////////////////////////////////////////////////
// 四捨五入
////////////////////////////////////////////////////////////////////////////
if (ratation_now_u > =0)
carry_spin_now_u = ratation_now_u+0.501;
else
carry_spin_now_u = ratation_now_u-0.501;
if (ratation_now_v > =0)
carry_spin_now_v = ratation_now_v+0.501;
else
carry_spin_now_v = ratation_now_v-0.501;
```

```
////////////////////////////////////////////////////////////////////////////

// 算出下一步的旋轉座標
////////////////////////////////////////////////////////////////////////////
ratation_next_u=(next_u)*cos(-ratation_angle)+(next_v)*sin(-ratation_angle);
m_Ratation_Nextu.SetValue(ratation_next_u);
ratation_next_v=-(next_u)*sin(-ratation_angle)+(next_v)*cos(-ratation_angle);
m_Ratation_Nextv.SetValue(ratation_next_v);
////////////////////////////////////////////////////////////////////////////
// 四捨五入
////////////////////////////////////////////////////////////////////////////
if (ratation_next_u > =0)
carry_spin_next_u = ratation_next_u+0.501;
else
carry_spin_next_u = ratation_next_u-0.501;
if (ratation_next_v > =0)
carry_spin_next_v = ratation_next_v+0.501;
else
carry_spin_next_v = ratation_next_v-0.501;
////////////////////////////////////////////////////////////////////////////

// 判斷目前的座標和矩陣的程式啓動與否
////////////////////////////////////////////////////////////////////////////
delay_one_sec = true;
////////////////////////////////////////////////////////////////////////////
// 將最後處理完成的 X 軸移動量，透過 start_r_move() 函式傳給運動控制卡
start_r_move(AxisNoX,checker_u_move,0,xsliderbuffer,0.1);
// 將最後處理完成的 Y 軸移動量，透過 start_r_move() 函式傳給運動控制卡
start_r_move(AxisNoY,checker_v_move,0,ysliderbuffer,0.1);
```

3.8 Multithread 機制

對於一個視窗而言，若無法 Mulitithread 機制 (執行緒機制)，則當按紐按下，處理對應事件時，視窗是固定無法變動的，因爲 CPU 的所有資源都用來處理按鈕按下的事件，除非事件處理完畢，否則無再作其他事情。假如運動控制系統，在平台行進過程中，無法即時的改變加速度、減速度及速度，或停止運動的話，那「控制」變得毫無意義。

MFC 將所有相關 multithread 的函式參數，包成 CWinThread 物件，透過 CWinThread 物件及 AfxBeginThread() 函式的宣告，連結執行緒函式，建立成一個執行緒，CPU 則會根據執行緒的優先權設定，分配排定處理時間。以下爲執行緒函式及 AfxBeginThread 函式的宣告型態 :

static UINT ThreadFunc(LPVOID lparam);

CWinThread* AfxBeginThread(AFX_THREADPROC pfnThreadProc,LPVOID lParam,int nPriority,UNIT nStackSize,DWORD dwCreateFlags, LPSECURITY_ATTRIBUTES lpSecurityAttrs);

AfxBeginThread 函式的第一個參數 pfnThreadProc 爲執行緒函式。第二個參數 lParam 爲傳給執行緒函式的參數。參數三 nPriority 表示優先權的微調值，預設爲 THREAD_PRIORITY_HIGHEST。參數四 nStackSize 表示堆疊大小，預設值表示堆疊最大容量爲 1MB。參數五 dwCreateFlags 如果預設值 0，表示執行緒產生後立刻開始執行；如果其值爲 CREATE_SUSPENDED，就表示執行緒產生後暫停執行。

系統的程式設計開頭，便在 3axis_controlDlg.h 標頭檔裡宣告了 CWinThread 物件和執行緒函式 :

```
CWinThread *m_pThread[4];
bool Continue[4];
static UINT ThreadFun1(LPVOID lParam);
static UINT ThreadFun2(LPVOID lParam);
static UINT ThreadFun3(LPVOID lParam);
static UINT ThreadFun4(LPVOID lParam);
//Thread 結束時呼叫
void OnThreadExit();
// 關閉所有 Thread 開關
void AllThreadTurnOff();
```

系統的程式設計開頭，便在 3axis_controlDlg.cpp 標頭檔裡宣告了 CWinThread 物件和執行緒函式 :

```
#define WMU_THREAD_EXIT (WM_USER+1)
bool Thread1Enable = false;
bool Thread2Enable = false;
bool Thread3Enable = false;
struct THREAD_INFO
{
int SleepTime;
bool* Continue;
int CtrlID1, CtrlID2, CtrlID3, CtrlID4, CtrlID5, CtrlID6;
HWND hWnd;
}Thread_Info[5];

void CMy3axis_controlDlg::OnThreadExit()
{.....}
```

改寫之前設計的方向鍵與停止鍵函式，將其原本的處理事件內容，移植到其所屬的執行緒函式，而新的處理事件內容將改爲建立執行緒並立刻執

行，無可置疑的，停止執行緒的優先權將高於方向鍵的優先權 :

```
// 改寫方向鍵觸發事件內容 : 建立執行緒
void CMy3axis_controlDlg::OnClickCommandbuttonPanelJog()
{
Continue[2] = true;
Thread2Enable = true;
Thread_Info[2].hWnd = m_hWnd;
Thread_Info[2].Continue = &Continue[2];
// 將視窗內的子控制元件 IDC_EDIT_X_MOVETEXT 轉爲有 Thread 功能繼承
的 CtrlID1
Thread_Info[2].CtrlID1 = IDC_EDIT_X_MOVETEXT;
// 將視窗內的子控制元件 IDC_EDIT_Y_MOVETEXT 轉爲有 Thread 功能繼承
的 CtrlID2
Thread_Info[2].CtrlID2 = IDC_EDIT_Y_MOVETEXT;
m_pThread[2] = AfxBeginThread(ThreadFun2, (LPVOID)&Thread_Info[2]);
// 設定執行緒的優先權
m_pThread[2]- > SetThreadPriority(THREAD_PRIORITY_HIGHEST);
}

// 執行緒函式
UINT CMy3axis_controlDlg::ThreadFun2(LPVOID lParam)
{
THREAD_INFO* Thread_Info = (THREAD_INFO*) lParam;
CMy3axis_controlDlg *hWnd = (CMy3axis_controlDlg*)CWnd::FromHandle((HWND)
Thread_Info- > hWnd);
// 原寫法爲 CEdit *m_Xmovetext=(CEdit *)GetDlgItem(IDC_EDIT_X_MOVETEXT);
// 將在視窗內的子控制元件繼承 Thread 的功能，因此寫法爲 :
CEdit *m_Xmovetext = (CEdit*) hWnd- > GetDlgItem(Thread_Info- > CtrlID1);
CEdit *m_Ymovetext = (CEdit*) hWnd- > GetDlgItem(Thread_Info- > CtrlID2);
```

```
// 程式爲原本處理按鈕觸發事件的內容
.....
// 結束 CWinThread 功能的程序
*Thread_Info- > Continue = false;
::PostMessage(hWnd- > m_hWnd, WMU_THREAD_EXIT, 0, 0);
return 0;
}
```

第四章

實驗測試

4.1 概述

本章節主要目的爲系統操作測試結果，共分四個測試項目 :

1. 移動控制測試

2. 狀態回授測試

3. 面板區域移動測試

4. 棋盤式移動測試

因步進馬達平台的軟體限制，僅 X 軸容易觀察平台移動的情形，故以下測試皆以 X 軸爲控制對象進行觀察。圖 4.1爲實際系統完整硬體架構圖，而圖 4.2爲運動控制系統之控制端介面。

4.2 系統測試

圖 4.1: 實際系統完整硬體架構

4.2.1 測試一

測試 X 軸的左移或右移運動控制，參考圖 4.3、圖 4.4和 4.5說明，測試步驟如下 :

步驟 1. 調整 X 軸的速度值，且按「左」方向鍵，測試平台有無預期運動。

步驟 2. 按慢速移動鍵，且按「左」方向鍵，測試平台有無預期運動。

步驟 3. 按快速移動鍵，且按「左」方向鍵，測試平台有無預期運動。

步驟 4. 調整行進間距至最大值，延長平台移動的時間，以便觀察。

步驟 5. 測試運動控制卡傳回的位置值，有無正確地顯示在控制視窗上。

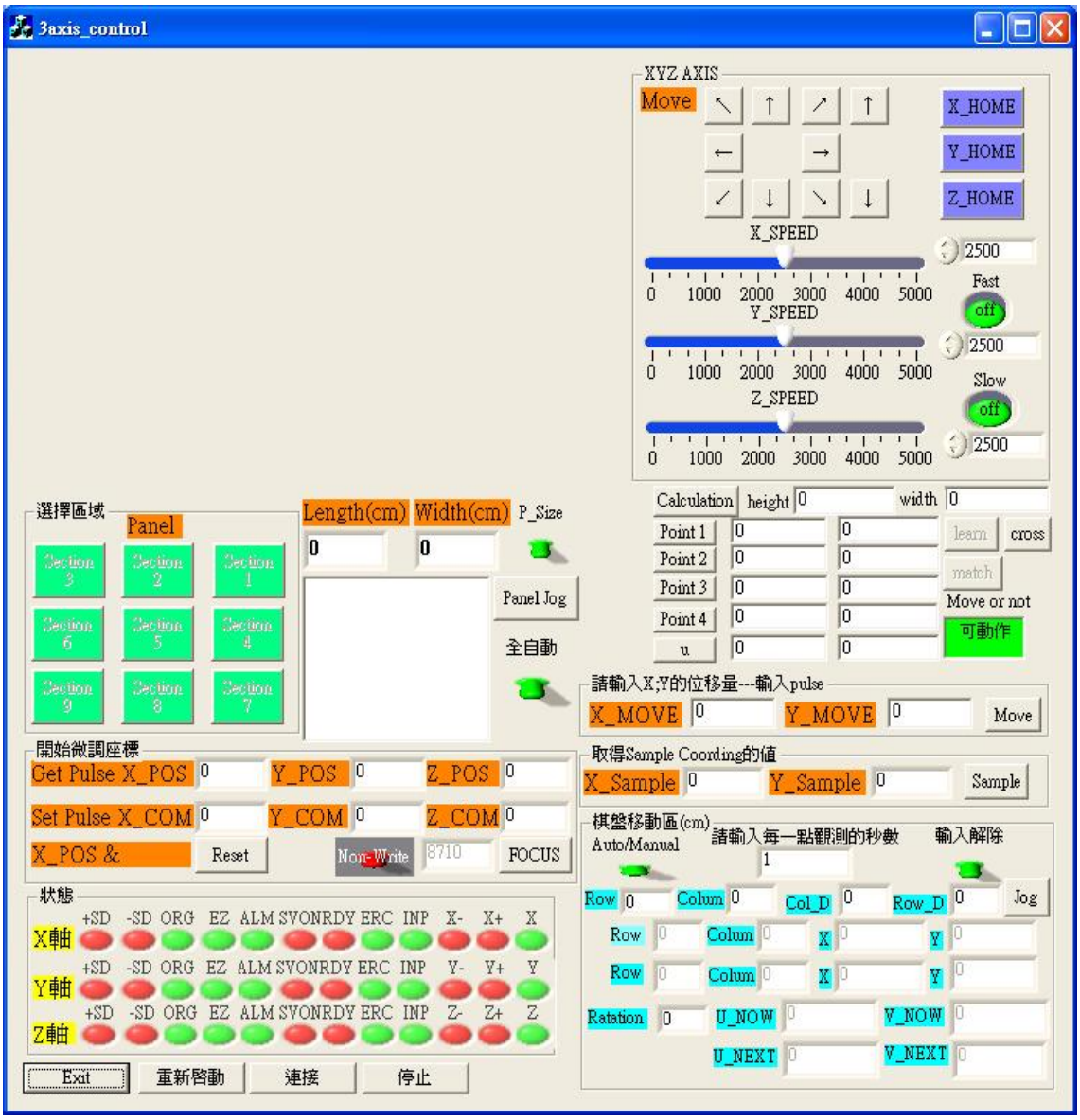


圖 4.2: 運動控制系統之控制端介面

4.2.2　測試二

此節測試並觀察平台 X 軸的左右極限運動與運動控制卡回傳的狀態，參考圖 4.6、圖 4.7和 4.8說明，測試步驟如下 :

步驟 1. 讓平台不斷往左移動，測試當到達極限時，有無停止運動發生，並呼叫警告視窗，告知使用者。

步驟 2. 讓平台不斷往右移動，重複以上步驟，測試右極限運動。

4.2.3 測試三

此節測試 Panel Section 的選擇程式和運動控制卡是否與選擇的方式進行移動，參考圖 4.9、圖 4.10、圖 4.11和 4.12說明，測試步驟如下 :

步驟 1. 輸入 Panel 長寬大小後，測試程式的 Panel Section 隨機選取，有無預期的執行。

步驟 2. 測試將已選取的 Panel Section 隨機刪除，有無預期的成效。

步驟 3. 再次隨機選取 Panel Section，測試平台有無預期的順序移動

步驟 4. 接著步驟三，重新輸入新的面板尺寸，測試平台有無預期的順序移動。

4.2.4 測試四

此節測試並觀察弓字形棋盤式運動，參考圖 4.13、圖 4.14和 4.15說明，測試步驟如下 :

步驟 1. 使用者輸入矩陣大小與間距與觀測時間。

步驟 2. 無旋轉角度與自動化的一步一步移動，測試平台回傳位置值有無預期的運動量。

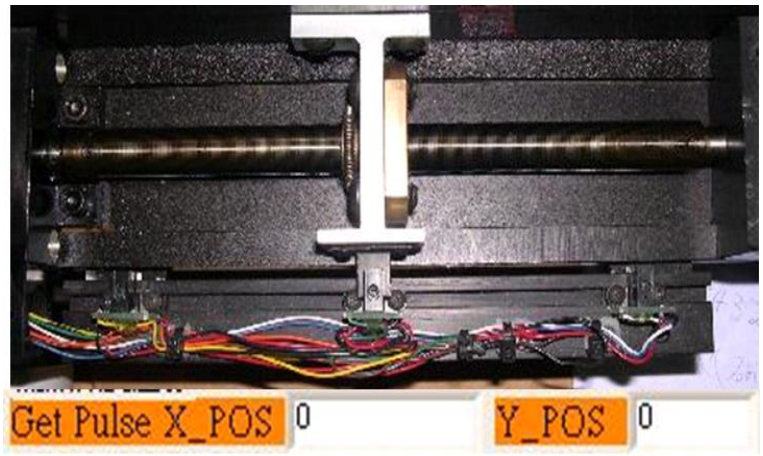
步驟 3. 有旋轉角度與自動化的一步一步移動，測試平台回傳位置值有無

預期的運動量。

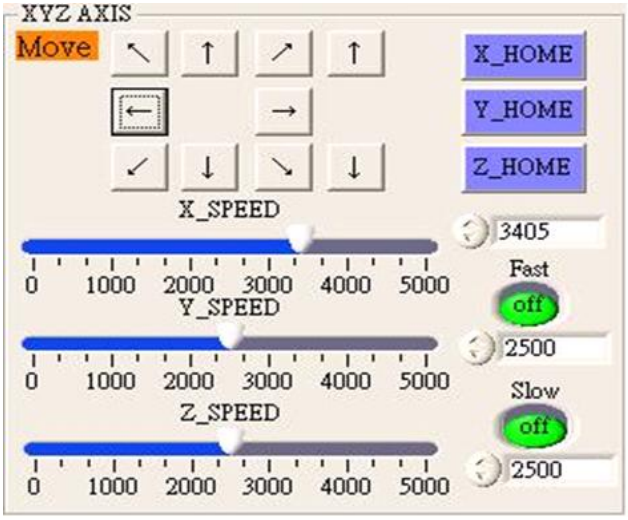
4.3 測試結論

經由測試一、測試二、測試三及測試四，和圖 4.3、圖 4.4、圖 4.5、圖 4.6、圖 4.7、圖 4.8、圖 4.9、圖 4.10、圖 4.11、圖 4.12、圖 4.13、圖 4.14、和圖 4.15的測試成果顯示，此運動控制系統爲了確實能即時並精確地達到預期的控制目的。至於其精密度爲 10um 位置誤差; 脈波速度範圍爲 0 到 2.4MHz

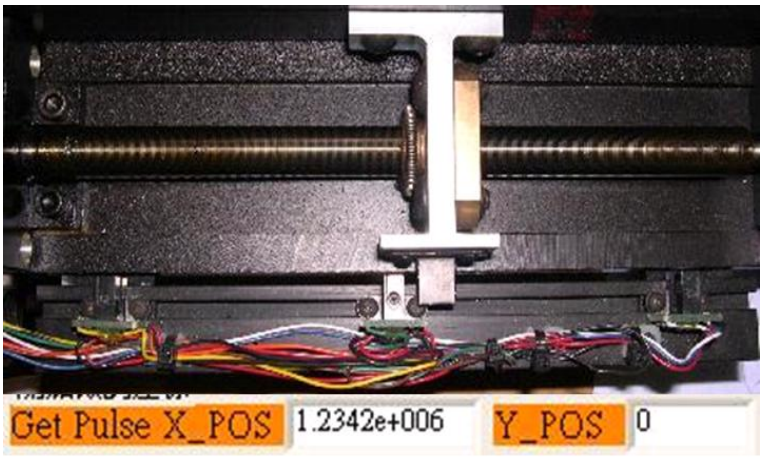
爲了更具體地顯示此運動控制系統的測試成果，我們將固定一隻筆在三軸微步進馬達平台的 Z 軸上，畫出多邊形、「永」字和面板選擇與棋盤的全自動化一步一步移動，如圖 4.16、圖 4.17和圖 4.18418 所示。



A. 平台起始位置及其座標

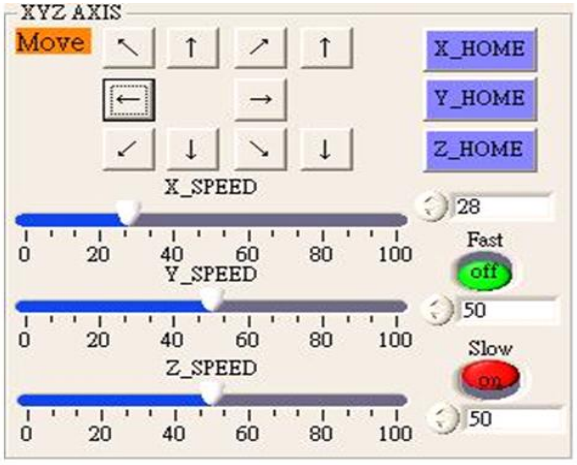


B. 按下「左」方向鍵

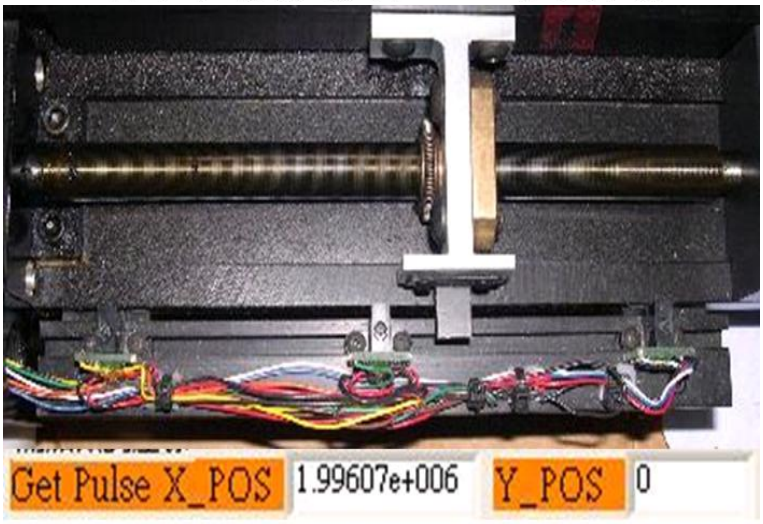


C. 平台開始移動及其目前座標

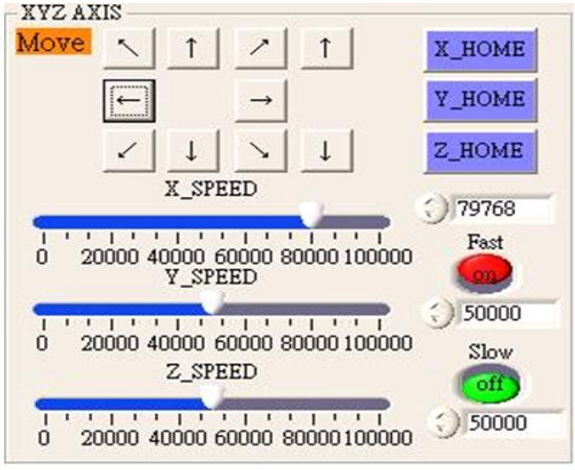
圖 4.3: 測試一



D. 降低速度及按下「左」方向鍵

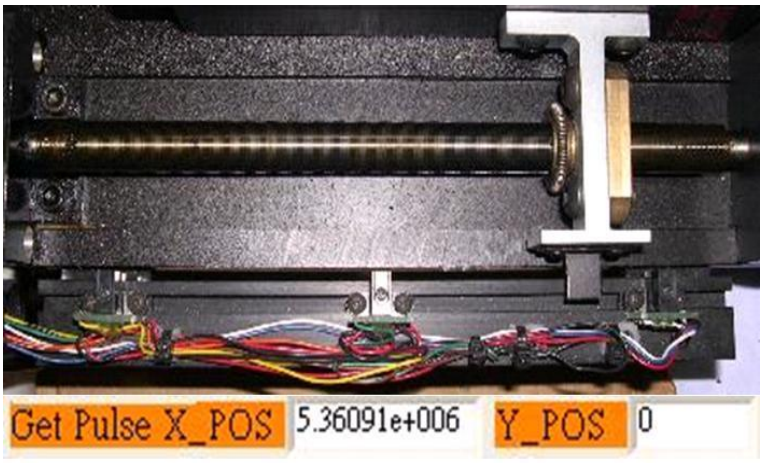


E. 平台速度變慢及其目前座標

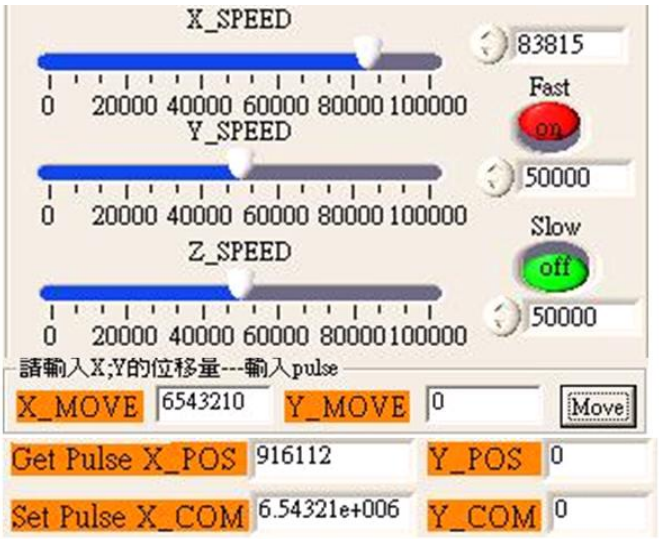


F. 加快速度及按下「左」方向鍵

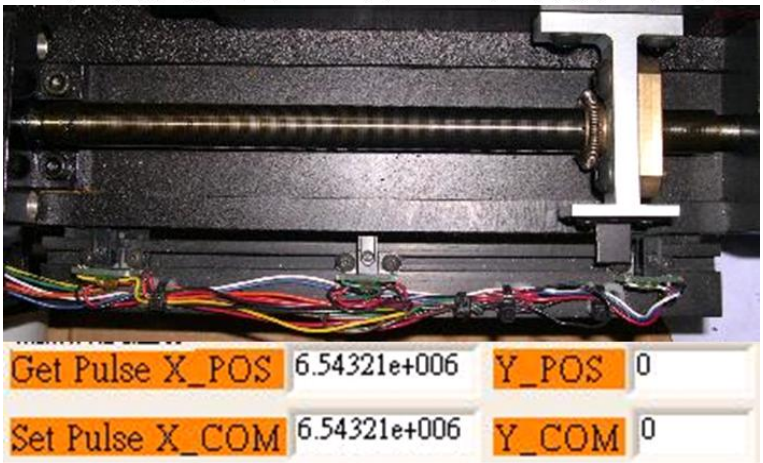
圖 4.4: 測試一



G. 平台速度變快及其目前座標

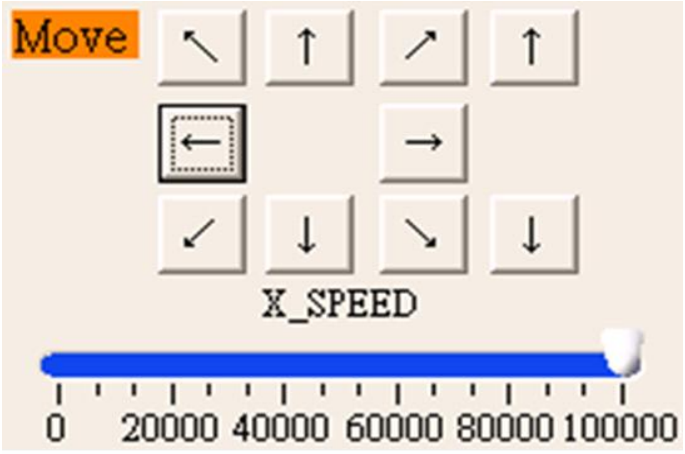


H. 控制參數設定與目的地和目前的座標

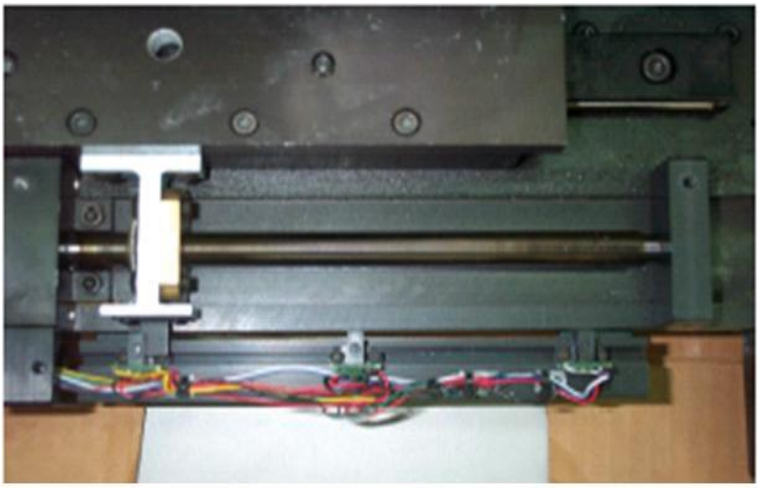


I. 視窗不斷更新位置且達到目標

圖 4.5: 測試一



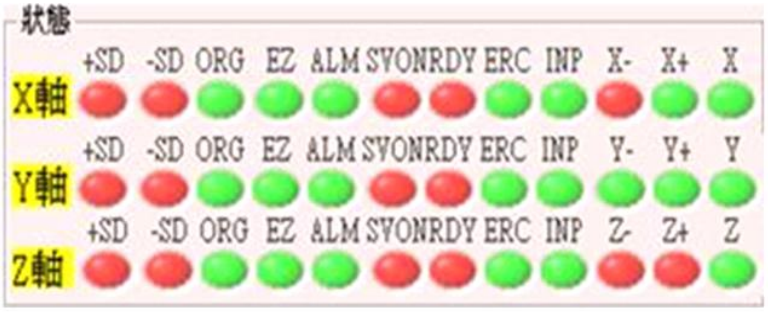
A. 按下「左」方向鍵



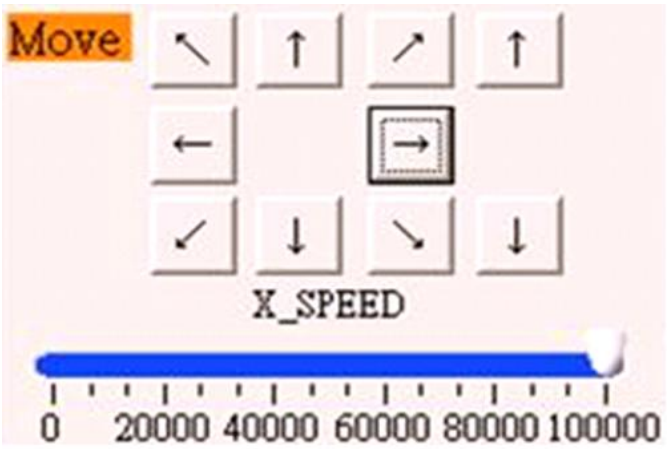
B. 平台走到左極限

C. 警告使用者

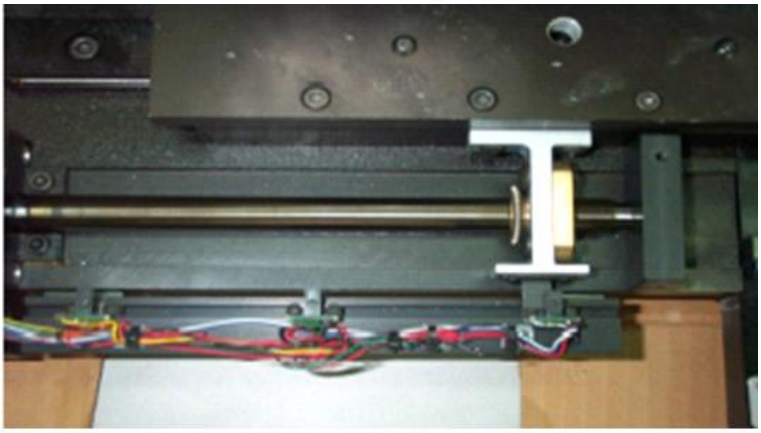
圖 4.6: 測試二



D. 運動控制卡回傳狀態；左極限變色



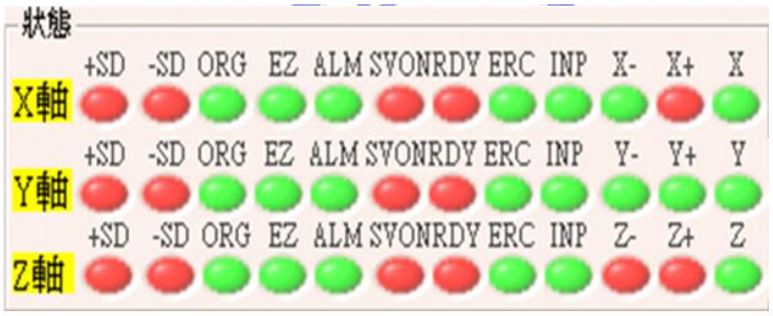
E. 按下「右」方向鍵



F. 平台走到右極限

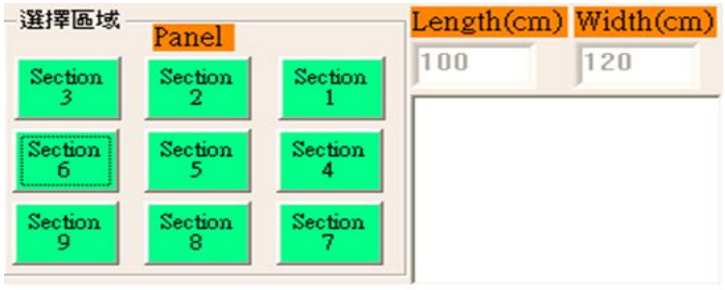
圖 4.7: 測試二

G. 警告使用者

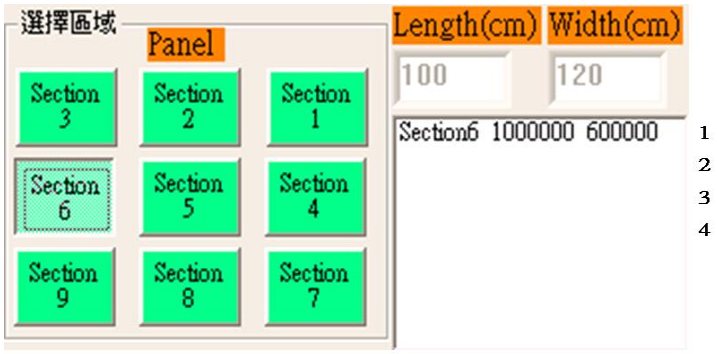


H. 運動控制卡回傳狀態；右極限變色

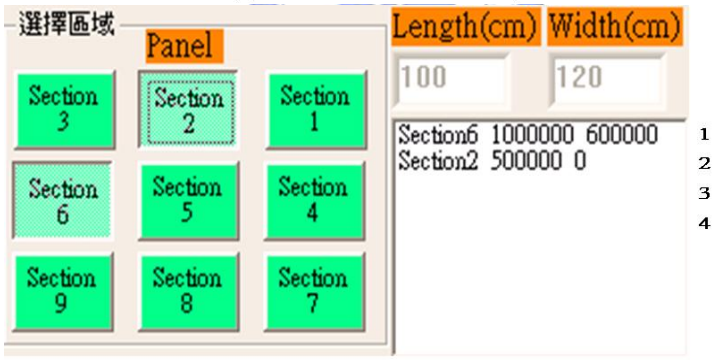
圖 4.8: 測試二



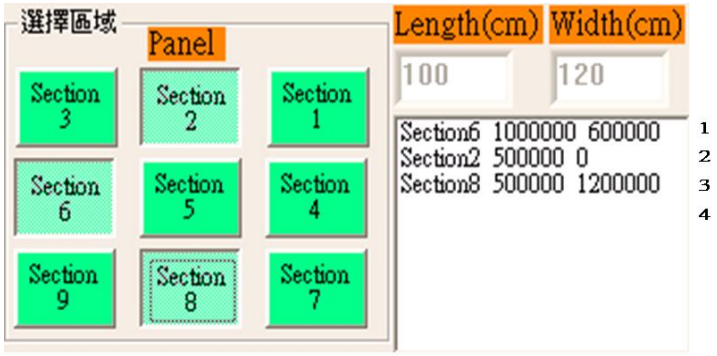
A. 輸入面板規格大小



B. 選擇Section 6且Index爲1

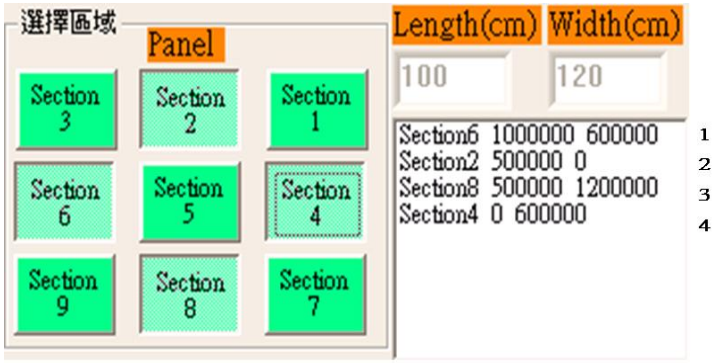


C. 選擇Section 2且Index爲2

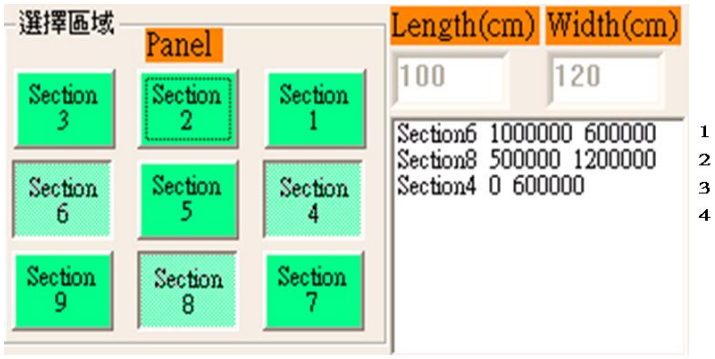


D. 選擇Section 8且Index爲3

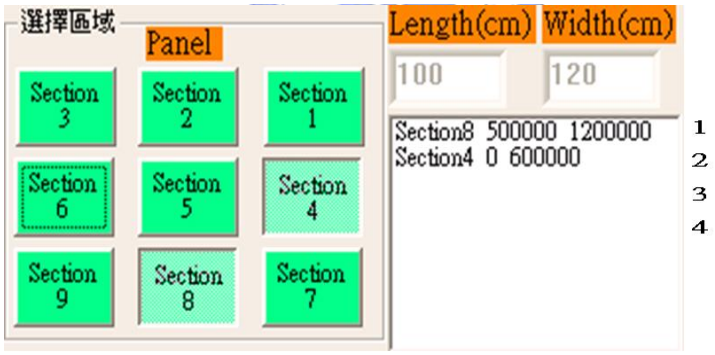
圖 4.9: 測試三



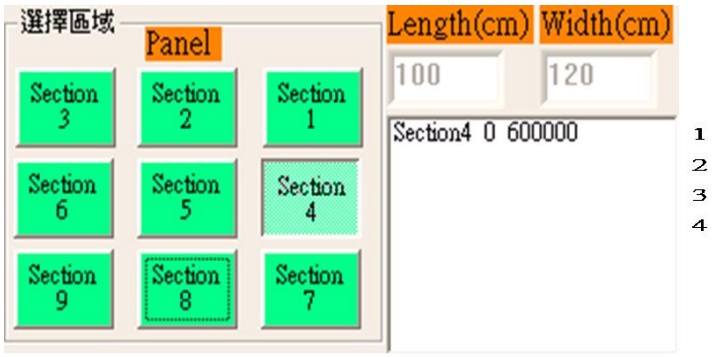
E. 選擇Section 4且Index爲4



F. 釋放Section 2且重給新的Index

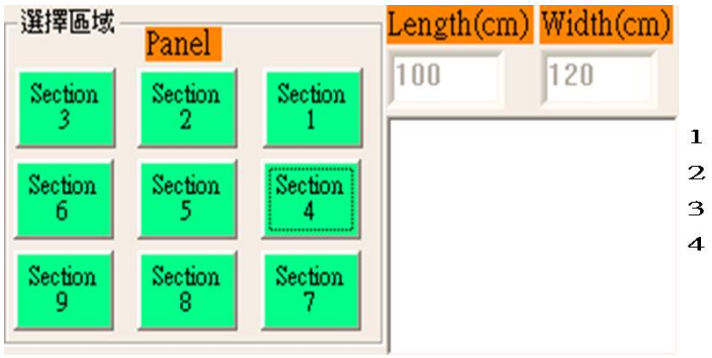


G.釋放Section 6且重新給定Index

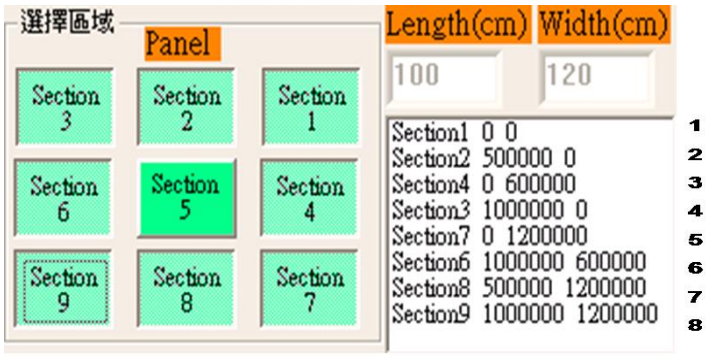


H.釋放Section 8且重新給定Index

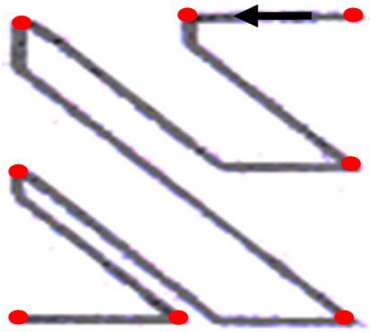
圖 4.10: 測試三



I. 恢復爲原來未輸入

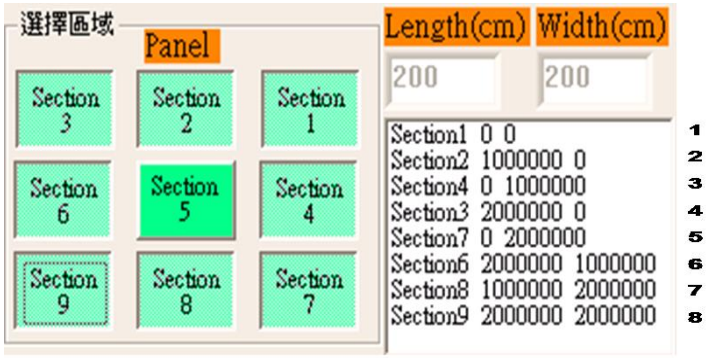


J. 重新隨機選取Section

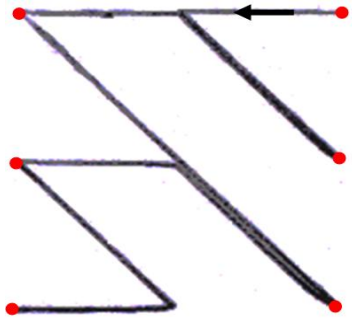


K. 實際運動路線圖

圖 4.11: 測試三



L. 重新輸入面板的尺寸再次模擬

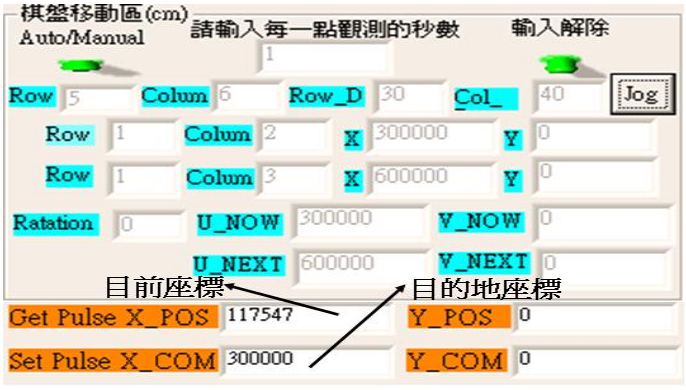


M. 實際運動路線圖

圖 4.12: 測試三

A. 輸入弓字形移動的參數； $\theta = 0$

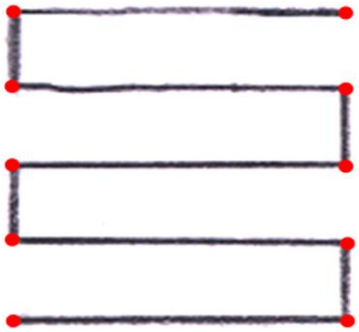
B. 按Jog按鈕；手動操作； $\theta = 0$



C. 運動控制卡回授位置值； $\theta = 0$

圖 4.13: 測試四

D. 啓動自動化模擬； $\theta = 0$

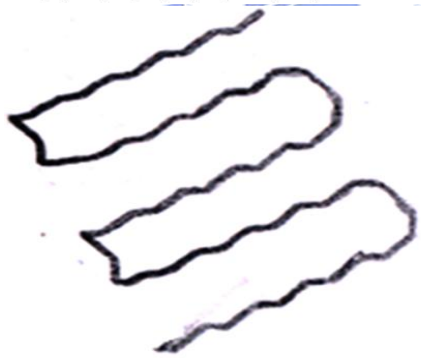


E. 模擬結果； $\theta = 0$

F. 輸入弓字形移動的參數； $\theta = 0.5$

圖 4.14: 測試四

G. 啓動自動化模擬； $\theta = 0.5$



H. 模擬結果； $\theta = 0.5$

圖 4.15: 測試四

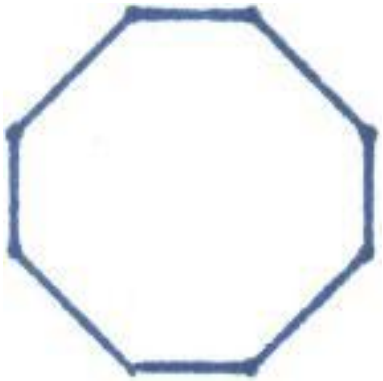


圖 4.16: 多邊形

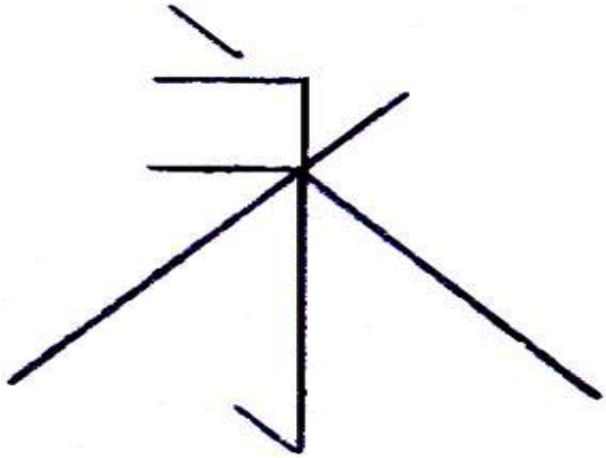


圖 4.17:「永」字

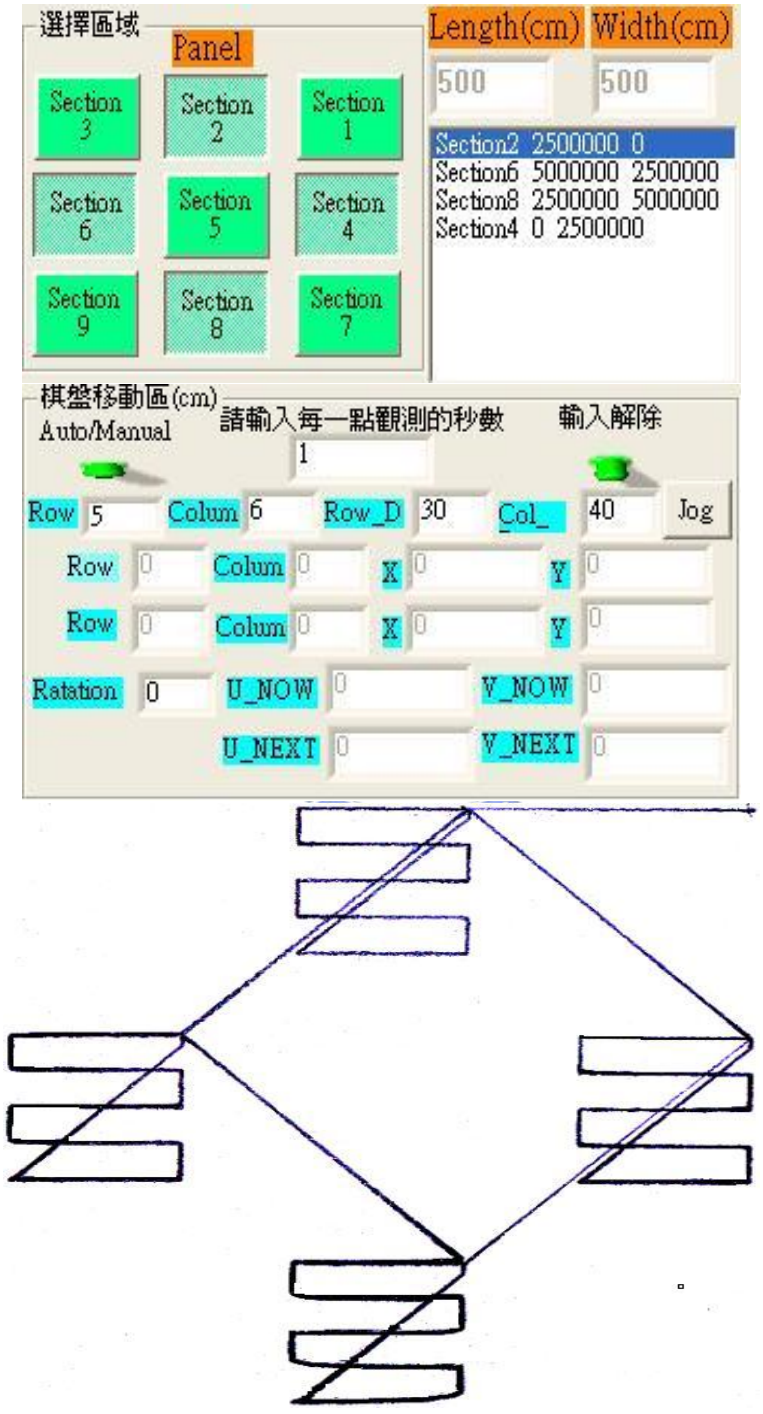


圖 4.18: 井字與弓字形全自動化模擬

第五章

結論與未來發展

5.1 結論

本論文的目的在於設計一套有別於其他相關研究的運動控制系統，在 Visaul C++ 的軟體設計環境中，建構人性化的控制視窗，提供視覺化的操控介面，讓使用者一目了然的並易於控制平台的運動。

一個運動控制系統的設計，除了能即時的控制運動外，還要能對運動平台的限制作出適當的保護措施，避免使用者不當的操作，而導致機台損毀甚至發生意外。本論文所設計的運動控制系統不但能及時精準的控制運動和隨時監控運動平台的所在位置外，對平台運動極限的發生，做出停止並回歸原點的機制，更能配合影像檢測的需求，規劃影像檢測的移動路徑與檢測時間。第四章實作測試的結果證實，本論文所研製之運動控制系統確實能正確無誤地達成運動控制的目的。

5.2 未來發展

本論文所研製之運動控制系統，尚未完備，能有許多可改善進步的空間。大致可分成四個發展空間 :

1. 除了八個基礎的直線方向控制外，控制系統上可加入圓形、橢圓形及圓弧等運動控制，使用者能透過人性化的介面視窗，決定圓心位置、半徑、焦點、及行走路徑等，增加控制路徑的多樣性。

2. 位置監控機制，能對於運動控制的位置，經由回授後傳達給使用者，將其以繪圖方式將目前位置，以軌跡繪畫出來，搭配運動控制卡的狀態回授，更能讓使用者更具體從圖中判斷出目前位置與邊界的相對距離。

3. 除了 Pattern Match 的功能外，影像處理功能上可加入邊緣檢測處理，使用者可透過程式來自動調整 Z 軸聚焦、短路或斷路和更精準的面板 Debug.... 等功能。

4. 網路通訊的建構，再加上影像處理及邊緣檢測的機制，就能成爲一個遠端運動控制系統，使用者可利用 Web 瀏覽器，透過網際網路，對遠端的運動平台進行監控，並即時回傳畫面至主控端，使操控者無論在何時何地，都能掌握運動平台的資訊。

參考文獻

[1] 蔡佳仁 , 整合型監控系統之開發 , 國立中山大學機械工程學系研究所碩士論文 , 2000 年 .

[2] 張建宏 , 具網路監控功能之多軸運動控制系統研製 , 國立台灣科技大學電機工程系碩士論文 , 2000 年 .

[3] 郭俊宏 ,*PC-based* 運動控制系統之研製 , 國立交通大學電機與控制工程學系碩士論文 , 2003 年 .

[4] 曾彥馨 , 應用機器視覺 *TFT* 面板之表面瑕疵檢測與分類 , 元智大學工業工程與管理學系碩士論文 , 2003 年 .

[5] 羅文振 ,*TFT-LCD* 檢測機台與 *XY* 定位馬達控制系統之設計 , 屏東科技大學機械工程學系碩士論文 , 2003 年 .

[6] 林俊杰 ,*Visual C++ 6* 視窗程式設計經典 , 碁峰資訊股份有限公司 , 1999 年 .

[7] 侯俊傑 , 深入淺出 *MFC* 第 *2* 版 , 松崗電腦圖書資料股份有限公司 , 1997 年 .

[8] 位元文化 ,*Visual C++* 入門進階 , 文魁資訊股份有限公司 , 1999 年 .

[9] 蔡明志 ,*Visual C++* 教學手冊 , 碁峰資訊股份有限公司 , 2002 年 .

[10] 林錫寬 , 運動控制與運動控制卡之架構 , e 科技雜誌 , 2001 年 , no.3, pp.33-36.

[11] 凌華科技股份有限公司網站 http://www.adlink.com.tw。