

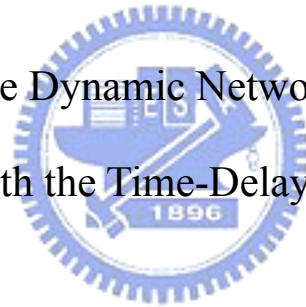
# 國立交通大學

電機與控制工程研究所

碩士論文

動態網路控制系統之時間延遲分析

Analysis of the Dynamic Network Control System  
with the Time-Delay Effect



研究生：鄭景文

指導教授：徐保羅 博士

中華民國九十五年十月

動態網路控制系統之時間延遲分析

Analysis of the Dynamic Network Control System  
with the Time-Delay Effect

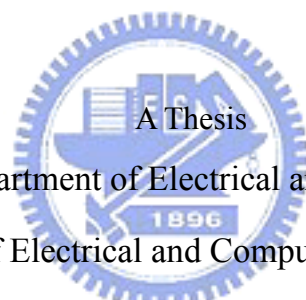
研究生：鄭景文

Student : Ching-Wen Cheng

指導教授：徐保羅 博士

Advisor : Dr. Pau-Lo Hsu

國立交通大學  
電機與控制工程學系  
碩士論文



Submitted to Department of Electrical and Control Engineering

College of Electrical and Computer Engineering

National Chiao-Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

October 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年十月

# 動態網路控制系統之時間延遲分析

學生：鄭景文

指導教授：徐保羅 博士

國立交通大學電機與控制工程所

## 摘要

近年來，網路在工業上的應用越來越多，由於網路化可以帶來許多優點，比如說減省配線，減少安裝和維護的成本，和增加監控和控制的距離等，所以，網路控制系統慢慢也成為一種趨勢。

在建構 CAN 網路方面，我們開發了一套 CAN 網路監控系統，可以在開發 CAN 網路的時候方便網路的監控和除錯。本論文整合了 Ethernet 和 CAN 兩種不協定的網路，建立一個網路控制系統平台，在遠端經由個人電腦，透過可讓 TCP/IP 和 CAN 兩種協定互相轉換的 gateway，成功的控制由微控器和 AC 伺服馬達組成的馬達遠端監控系統。

在網路控制方面，本研究透過實驗來量測客戶端到馬達系統間應用層到應用層的平均延遲時間。藉由實驗的分析，可以發現不同的取樣週期和控制環境皆會對延遲時間造成影響。在實驗中也發現距離因素不會影響延遲時間。隨著取樣時間減少和系統傳輸量增加，網路延遲時間也會慢慢跟著增加。在無線網路下所量測的平均延遲時間，會因環境干擾所造成延遲時間的突發性增加。但在經過處理後，可以取得較準確的延遲時間。本研究成功的設計史密斯估測器 (Smith predictor)，可從遠端建立監控系統，並實際成功的測試了由高雄控制交大實驗室中的伺服控制系統。

# Analysis of the Dynamic Network Control System with the Time-Delay Effect

Student : Ching-Wen Cheng

Advisor : Dr. Pau-Lo Hsu

Department of Electrical and Control Engineering

National Chiao-Tung University

## ABSTRACT

To construct a remote control system, a monitoring system has been developed as a platform in this study to debug the CAN network system. Furthermore, both the industrial network CAN and the commercial networks, the Ethernet and the 802.11, have been integrated as network control systems (NCSs). By applying the TCP/IP as the gateway between different networks, an AC servo motor with different networks (a) the Ethernet and the CAN and (b) the wireless 802.11 and the CAN have been established and tested. The time delay between the application layer of the client and the application layer of the remote control target has been measured, analyzed, and processed through experiments. Results of the Ethernet and the CAN system indicate that different sampling periods and environments greatly affect the delayed time. Moreover, the distance has insignificant effect on the time delay of the NCS. In the wireless network environment, time delay may suddenly increases due to its environment disturbance. In the present NCS with the wireless system, the measured time delay processed by eliminating all sudden change values leads to significantly improved results of NCS. Finally, to verify the proposed NCS with a serious time delay, a remote control system by including the Smith predictor has been

successfully applied to control a servo motor system from Kaohsiung to the laboratory in Nation Chiao-Tung University to verify the feasibility of the proposed systems.



## 誌謝

本論文的完成，對我來說意義非凡，首先要感謝指導教授徐保羅教授對我耐心的指導及不時的把我導引到正確的研究方向，尤其在最後撰寫論文及準備口試這段期間，老師不辭辛勞，還利用晚上和假日的時間來幫忙我，讓我在有限的時間內完成我的論文及口試，順利取得碩士學位，在此，獻上我最誠摯的敬意與感謝。同時也要感謝口試委員胡竹生博士、趙昌博博士和葉賜旭博士在論文上面的建議和指導。

感謝博士班謝鎮洲學長、幸琮政學長、賴建良學長和已經畢業的李俊賢學長平時給我的意見和幫助，以及實驗室的同學尚玲、學弟元銘、興漢、宗翰、瑞原、孝麟、雨坤、林億、宗勝、也強在學業上的互相切磋與指教，並且為枯燥的研究生生活，帶來不少歡樂和笑聲。感謝高中同學徐敏和鄭永祥不厭其煩的幫我做實驗，使我的論文內容更加的完整。

在此，要將本論文獻給我的父親 鄭森元先生、母親 陳明玉女士，感謝你們在我失意的時候可以給我支持與鼓勵，以及在生活上面無後顧之憂，讓我可以專心在研究和學業上面。

感謝所有關心和幫助過我的家人及朋友，因為有你們，這本論文才可以順利的完成，謝謝你們。

# 目錄

中文摘要.....	i
英文摘要.....	ii
誌謝.....	iv
目錄.....	v
表目錄.....	ix
圖目錄.....	x
第一章 緒論.....	1
1-1 研究動機與目的.....	1
1-2 研究背景與概況.....	2
1-3 問題陳述.....	3
1-4 研究方法.....	3
1-5 論文架構.....	4
第二章 CAN、Ethernet 和 IEEE 802.11 網路介紹.....	5
2-1 CAN BUS 介紹.....	5
2-1-1 CAN BUS 網路協定介紹.....	5
2-1-2 CAN 的資料傳輸方式.....	6
2-1-3 CAN 的錯誤處理.....	12
2-2 乙太網路(Ethernet)介紹.....	14

2-3	IEEE 802.11 介紹.....	16
2-3-1	MAC 層的運作機制.....	16
2-3-2	操作模式.....	17
2-4	TCP/IP 網路通訊協定介紹.....	18
第三章	CAN 網路的監控系統設計.....	20
3-1	CAN 網路的建立.....	20
3-1-1	ID 分配.....	20
3-1-2	使用者自訂欄框.....	21
3-2	CAN 網路監控程式介紹.....	22
3-2-1	USB CAN 介紹.....	22
3-2-2	USB CAN 軟體介面的建立.....	23
3-2-3	網路監控程式.....	28
第四章	網路延遲時間與量測.....	32
4-1	網路控制系統.....	32
4-2	網路延遲時間(delay time)探討.....	33
4-3	網路延遲時間測定實驗.....	35
4-3-1	網路延遲時間量測方法.....	35
4-3-2	量測延遲時間人機介面.....	36
4-3-3	網路控制系統的延遲時間.....	37



4-3-4	固定延遲時間的產生及量測 .....	46
4-3-5	遠距離的網路延遲時間 .....	48
第五章	網路遠端控制系統設計 .....	51
5-1	Smith predictor .....	51
5-2	實驗系統架構 .....	53
5-3	TCP/IP 與 CAN 封包轉換 .....	56
5-3-1	Server 架構介紹 .....	56
5-3-2	封包轉換方式及過程 .....	57
5-3-3	程式介面及程式流程圖 .....	58
5-4	系統識別實驗 .....	61
5-4-1	系統識別人機使用介面 .....	63
5-4-2	實驗結果 .....	64
5-5	DSP 端程式 .....	66
5-6	Smith predictor 設計 .....	69
5-6-1	Smith predictor 在數位系統的實現 .....	69
5-6-2	實驗環境 .....	70
5-6-3	Smith predictor 人機介面介紹 .....	71
5-6-4	不同延遲時間對網路控制的影響 .....	72
5-7	遠距離網路控制實驗 .....	80
第六章	結論與未來展望 .....	82



# 表目錄

表 4-1 Ethernet + CAN 的平均延遲時間 .....	38
表 4-2 802.11 + CAN 的平均延遲時間(處理前).....	41
表 4-3 802.11 + CAN 的平均延遲時間(處理後).....	44
表 4-4 遠距離延遲時間 .....	48
表 5-1 網路控制系統系統事件定義表 .....	67



## 圖目錄

圖 2.1 節點優先權仲裁.....	7
圖 2.2 標準 CAN 資料欄框格式 .....	9
圖 2.3 擴展 CAN 資料欄框格式 .....	9
圖 2.4 IEEE 802.11 家族與 OSI 模型的關係.....	16
圖 3.1 三種自訂欄框 .....	21
圖 3.2 USBCANII 外觀 (a)正面 (b)背面.....	23
圖 3.3 網路監控程式主畫面 .....	28
圖 3.4 USB CAN 系統參數設定畫面 .....	29
圖 3.5 監控程式擷取畫面 .....	31
圖 4.1 網路控制系統方塊圖 .....	33
圖 4.2 延遲時間對系統影響圖(delay time = 106.968 ms) .....	34
圖 4.3 延遲時間對系統影響圖(delay time = 255.196 ms) .....	34
圖 4.4 網路系統延遲時間示意圖 .....	35
圖 4.5 延遲時間計算 .....	35
圖 4.6 量測延遲時間人機介面 .....	37
圖 4.7 Ethernet + CAN 之延遲時間量測 .....	40
圖 4.8 802.11 + CAN 之延遲時間量測(處理前) .....	43
圖 4.9 802.11 + CAN 之延遲時間量測(處理後) .....	46

圖 4.10 超過網路覆載的延遲時間.....	46
圖 4.11 回授端加 Buffer 示意圖.....	47
圖 4.12 加 Beffuer 後之延遲時間量測.....	48
圖 4.13 高雄 2 傳輸間隔 2ms 的不正常延遲時間.....	49
圖 4.14 高雄 1 之延遲時間量測.....	50
圖 5.1 網路控制系統的控制方塊圖.....	51
圖 5.2 Smith predictor 方塊圖.....	51
圖 5.3 加上 Smith predictor 的等效方塊圖.....	52
圖 5.4 實驗系統架構圖.....	53
圖 5.5 Ti F2812 ezDSP DSK 板+自製週邊電路+Panasonic AC 伺服馬達.....	54
圖 5.6 本次實驗 Server+USBCAN.....	54
圖 5.7 DSP 和 Server 之間透過 CAN 來做連接.....	55
圖 5.8 Server 和馬達系統.....	55
圖 5.9 Server 架構圖.....	56
圖 5.10 封包轉換圖.....	57
圖 5.11 Server 人機介面.....	58
圖 5.12 Server 連線管理流程圖.....	60
圖 5.13 CAN 轉 TCP/IP 流程圖.....	60
圖 5.14 TCP 轉 CAN 流程圖.....	60

圖 5.15 系統識別流程圖.....	62
圖 5.16 自動產生 C# 系統模型程式片段.....	62
圖 5.17 系統識別人機使用介面.....	63
圖 5.18 系統識別的輸入(a)和輸出(b).....	64
圖 5.19 System Identification Toolbox GUI.....	65
圖 5.20 不同階數估測模型的誤差.....	65
圖 5.21 估測模型和實際系統比較圖.....	65
圖 5.22 系統判別位置誤差.....	65
圖 5.23 Smith predictor 架構圖.....	69
圖 5.24 DSP 對馬達控制的系統響應圖.....	70
圖 5.25 Smith predictor 人機械面.....	72
圖 5.26 Ethernet + CAN 之 Smith Predictor 改善效果 (控制器 A).....	74
圖 5.27 Ethernet + CAN 之 Smith Predictor 改善效果 (控制器 B).....	75
圖 5.28 802.11 + CAN 之 Smith Predictor 改善效果 (未處理)(控制器 A) ...	76
圖 5.29 802.11 + CAN 之 Smith Predictor 改善效果 (未處理)(控制器 B)....	77
圖 5.30 802.11 + CAN，延遲時間處理前後 (控制器 A).....	78
圖 5.31 802.11 + CAN，延遲時間處理前後 (控制器 B).....	79
圖 5.32 延遲時間分佈圖(a)高雄 1 遠端傳送，(b)近端自行產生.....	80
圖 5.33 Ethernet + CAN，Smith predictor 補償的響應圖 (控制器 A).....	81

圖 5.34 Ethernet + CAN，Smith predictor 補償的響應圖 (控制器 B)..... 81



# 第一章 緒論

## 1-1 研究動機與目的

近年來網路化在工業應用上來說是一個趨勢，網路化對工業應用可以帶來節省配線，系統擴充性增加，減少安裝和維護成本，方便監控等優點，為了因應工業應用的網路化，目前市面上出現許多種適合工業使用的網路例如 CAN Bus、Profibus、FIP、SERCOS，Ethernet 等[1]。將網路跟多軸控制系統接合在一起就是網路化運動控制的一種應用。

由於商業網路比工業網路發展還要發達，所以 Ethernet 和無線網路比工業網路還要普及，若能將 Ethernet 或無線網路和工業網路如 CAN 做結合，就可以在更遠的距離操作和監控特定的系統。若是透過無線網路，則可以減少佈線的麻煩和而且可以增加機動性。將異質網路做整合是本論文研究的重點之一，如何將不同網路做連接達到資料交換的目的，本論文將完成整合 Ethernet、無線網路 802.11 和 CAN 網路的遠端監控系統。

當控制系統的控制命令和回授訊號是藉由網路傳輸來完成，稱此系統為網路控制系統(NCS)，雖然網路控制系統帶來許多好處，但是也因為使用網路來做為訊息傳輸的媒介，而帶來網路傳輸產生的延遲時間，資料封包遺失，網路排程，取樣頻率過高使得網路系統飽和等問題[2]。

本論文特別對延遲時間對系統影響來做討論，當延遲時間小於取樣週期時，延遲時間對整個控制系統沒有什麼影響，但是當延遲時間大於取樣週期時，輕則會降低系統的效能，嚴重則會造成整個控制系統的不穩定[3]。分析延遲時間以及如何有效消除延遲時間對網路控制系統的影響，並完成了整合不同網路系統之遠端監控系統。



## 1-2 研究背景與概況

隨著半導體製成技術的進步，處理器的功能越來越多且價格越來越便宜，多軸控制系統中各軸皆可以獨立處理，再加上目前市面上有越來越多種規格工業網路，因此我們現在可以用較低的成本來建立分散式網路控制系統，目前也有相關的研究，有人利用 CAN Bus 建立一套分散式控制系統[4]，並且加入多軸控制理論，來發展高速且高精密分散式的多軸運動控制系統[5]。

一般網路控制系統，雖然擁有網路化的優點，但是往往需要犧牲一點控制效能，有研究指出雖然在數位控制系統下，控制效果會隨著取樣週期變小而變好。但是當我們將網路加入控制系統當中，將會使得取樣週期只能小到一個臨界值，當取樣週期低於此臨界值控制效能會變差，發生此種原因是因為取樣週期越小，相對的網路上面的資料量也越大，使得網路負擔變大，繼而拉大訊息傳遞的延遲時間，造成控制效能不好[6]。

除了因為取樣週期的選擇造成網路控制效能不好以外，網路控制還有其他因素也會造成控制效能不佳，例如網路延遲時間、資料封包遺失，和網路排程等原因[2]，這些問題都有許多研究成果，並且提出解決方法來改善。資料封包的遺失，會造成控制系統的命令遺失，有人提出使用泰勒展開法來估測遺失的命令[7]，使得系統不會因為命令遺失而造成系統控制效果不佳，在網路排程方面，有人提出 adjustable priority scheme 的排程方法，藉由累積 CAN 網路上訊息一段時間的傳輸狀況後，逐漸調整訊息優先權，經由此方法改變網路的排成可以有效的提昇網路傳輸的效率[8]。在網路延遲時間方面，也有相關理論解決延遲時間對系統造成的不良影響[9]。

現今市面上有有各種不同網路，這些網路之間協定不能互通，若有不同的網路需要整合，則必須建立起 gateway 來做封包的轉換已達到資料交換的目的[10]。

## 1-3 問題陳述

以下是本論文將要解決的問題

- 1.在 CAN 網路建立方面，需要事先規劃好 ID 和訊息事件，若不經過事先規劃使用規則，這對在開發系統和除錯會帶來很大困擾。
- 2.由於 DSP 只具有 CAN 網路的介面，所以想要和使用網際網路的 Client 控制端做資料交換，必須建立一套機制來做封包的轉換。
- 3.為了量測網路延遲時間，我們必須先建立好量測平台，在探討延遲時間對系統的影響，需要針對訊息傳送頻率還有延遲時間的變異量來做實驗探討。
- 4.要在一個具有延遲時間的網路系統做有效的控制，必須有適當的控制設計來解決網路延遲時間的影響。



## 1-4 研究方法

在本論文當中，主要試探討網路延遲時間對網路系統造成的影響，並且使用 Smith predictor 來消除網路延遲時間對系統的不良影響[11,12]，在建立網路控制實驗平台時，為了要讓整合 CAN 網路和 TCP/IP 通訊協定，本研究建立一個 gateway 來滿足此需求。在建立 CAN 網路方面，本研究提供一個有系統的建構方式和網路監控系統，來方便我們開發和除錯。

本論文研究方法及步驟如下：

- 1、建立 CAN 網路方面，本研究會先將 CAN 網路做有系統的建制，給予每個節點唯一的識別 ID 和規劃好自訂欄框，方便節點間的溝通，並且建立監控程式，在 CAN 網路上建立可以掌握網路上的訊息且方便開發以及除錯的機制。

- 2、在建立網路控制平台方面，整合 TCP/IP 和 CAN 網路兩種協定部份，本研究建立轉換 TCP/IP 和 CAN 兩種協定的 gateway，來讓 Client 控制端可以對遠端下達命令，遠端也可以透過此 gateway 傳送回授資料。
- 3、量測延遲時間部份，本研究利用 DSP 來做延遲時間的量測，分析 Ethernet 和 IEEE802.11 無線網路在不同環境下所量測到的延遲時間，以作為遠端監控設計。
- 4、補償延遲時間對系統造成的影響方面，本研究將所量測到的平均延遲時間來設計 Smith predictor，藉由 Smith predictor 來消除延遲時間對系統的影響，以建立一整合工業與民間網路之遠端監控系統。

## 1-5 論文架構

本論文共分六章，第一章在說明研究動機目的還有研究方法。第二章介紹 CAN、Ethernet 和 IEEE802.11 三種網路。第三章說明如何建立 CAN 網路和網路監控程式。第四章為討論網路延遲時間對系統的影響，並且分別量測使用 Ethernet 和 IEEE802.11 無線網路搭配 CAN 網路所得到網路延遲時間，並且加以分析。第五章為建立整個網路控制系統的建立，並且使用 Smith predictor 來消除延遲時間對系統的影響。

## 第二章 CAN、Ethernet 和 IEEE802.11 網路介紹

### 2-1 CAN BUS 介紹

#### 2-1-1 CAN BUS 網路協定介紹[13]

CAN 全名為 Controller Area Network 是在 1990 年由德國 Robert Bosch 公司所制定的一種具有高度安全且支援即時分散式控制的通訊協定，最高傳輸速度可達 1M bit per second (bps)。CAN 原先是應用在汽車內，用來連接汽車內防鎖死煞車系統 (ABS) 或是引擎控制單元等的電子訊號，以取代車子內複雜的硬體接線。但由於 CAN 提供可靠快速的連線，適合用在即時系統 (real-time system)，且價格低廉，所以也可以在其他控制場合使用。目前 CAN 已成為國際標準規格 (ISO11898)，CAN 在場域匯流排 (field bus) 的應用已有 Honeywell 的 SDS 及 Allen-Bradley 的 DeviceNet。

為了可以達到透明且彈性的應用，CAN 被細分為不同層次

- CAN 對象層 (the CAN-object layer)
- CAN 傳輸層 (the CAN-transfer layer)
- 物理層 (the physical layer)

對象層和傳輸層包含 ISO/OSI 模型定義的資料鍊結層的所有功能，對象層的功能包括：

- 尋找被發送的訊息
- 確定實際要使用的傳輸層接收那一個訊息
- 替應用層相關硬體提供介面

傳輸層的作用主要是傳輸的通訊協定也就是控制 frame 的結構、訊息的仲裁、錯誤偵測、錯誤標定、故障界定，CAN BUS 上面什麼時候發送訊息什麼時候接

收訊息也是定義在此傳輸層，還有一些位定時（bit timing）的普通功能也屬於傳輸層的一部分，傳輸層的修改是受到限制的。實際硬體如何完成連線接收到的資料要做何種處理等，都不在其規格範圍內。CAN 具有以下特點：

- 資料訊息具有優先權（priority）
- 優先權的仲裁（arbitration）為非破壞性
- 保證延遲時間
- 彈性的架構
- 採用廣播的方式（multicast），並藉由傳輸的訊息做時序同步的動作
- 任一節點皆可主動發出訊息（multimaster）
- 未傳送成功的訊息會自動重新傳送
- 錯誤檢查
- 區分暫時性錯誤和永久性錯誤並且自動關閉有問題的節點

接下來將介紹 CAN 的架構及特性，在 2-1-2 介紹資料的傳輸方式，2-1-3 則是介紹 CAN 的錯誤處理。

## 2-1-2 CAN 的資料傳輸方式

在介紹 CAN 的傳輸格式之前先介紹一些 CAN 的基本概念：

- 傳輸速率

不同的系統有不同的傳輸速度，但在一給定的系統當中傳輸速度是唯一且固定的，在規格書中寫到最高傳輸速度為 1Mbps。

- 仲裁（Arbitration）

只要匯流排空閒的時候，任何節點都可以發送和接收訊息，但是當有兩個節點以上要傳送訊息時，此時在匯流排上面就會產生衝突，透過訊息仲裁區的識別碼一個位元一個位元仲裁，即可解決衝突。在仲裁期間，位元為 0 的時候具有較高的優先權為 dominant 訊號，當位元為 1 的時候優先權較低為 recessive 訊號，因此當有兩個以上的節點發出訊息，每個發送節點都會對自己發送的訊息和匯流排上面的訊息做比較，當匯流排的訊息和自己發送的相同，則繼續發送位元，當發送訊息為一 recessive 訊號，匯流排上面為 dominant 訊號時，則此發送節點失去仲裁，必須退出發送狀態，得等下次訊息發送時再重新發送，圖 2.1 為仲裁時的示意圖，由此圖可以看出當識別碼越小的時候，有越高的優先權。

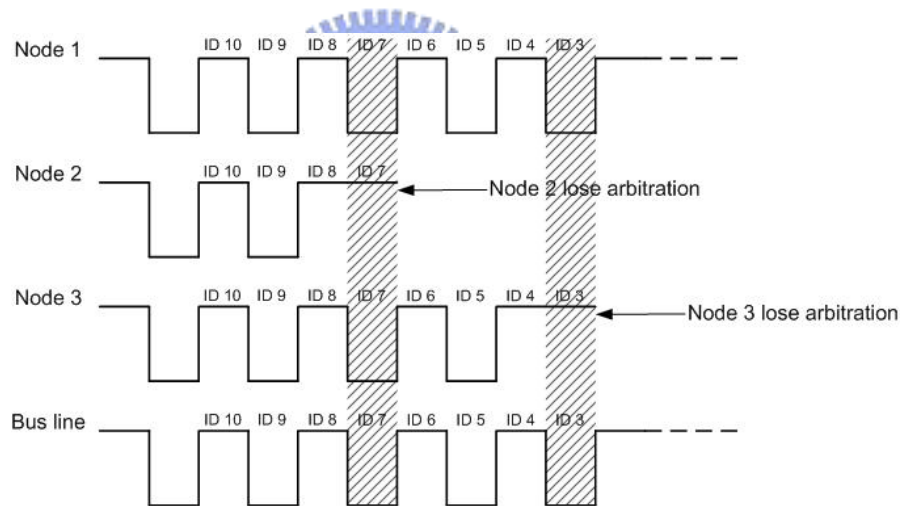


圖 2.1 節點優先權仲裁

- 發送節點 (transmitter) 與接收節點 (receiver)

當有一節點再傳送資料時，此節點稱為傳送節點，其餘的節點稱為接收節點。

- 發送節點對匯流排的偵測

在網路中，發送節點在發送資料的同時，也需偵測匯流排上的值是否與其送出的相同。

接下來要介紹 CAN 的傳輸格式，在原本 CAN 的規範當中，訊息的識別碼只有 11bits，為了擴大應用在 CAN 2.0 的規範中又另外訂了擴展模式 (extended format)，擴展模式包含了 29bits 識別碼，可以讓我們在運用時更有彈性。在 CAN 的網路傳輸中，有四種不同的欄框 (Frame)，分別為資料欄框 (Data frame)、遙控欄框 (Remote frame)、錯誤欄框 (Error frame) 與過載欄框 (Overload frame)，其中兩個資料 (遙控) 欄框或是資料欄框與遙控欄框之間都需要以欄框間隔做分隔，下面將對四個不同欄框做敘述：

- 資料欄框 (Data frame)

當節點有資料要傳輸的時候，需將資料放在資料欄框之中再發送到網路上面，資料欄框為四個欄框中最長的，圖 2.2 是標準 CAN 的資料欄框格式，圖 2.3 是擴展 CAN 的資料欄框格式。在說明欄框前需先介紹 CAN 的填充/編碼 (bit stuffing/coding) 傳輸方式：當發送節點在傳送資料或遙控欄框的時候，若是傳送五個連續且相同位準的位元，則需在第五個位元之後加入一個相反位準的位元。接收節點在接收時會自動將這填充進來的位元給刪除，以得真正的資料。錯誤欄框及過載欄框並不會經過位元填充的動作，而是以固定格式發送。位元填充只適用於資料欄框與遙控欄框的下列區域：

- ◆ 欄框起始 (Start of Frame)
- ◆ 仲裁區 (Arbitration Field)
- ◆ 控制區 (Control Field)
- ◆ 資料區 (Data Field)
- ◆ 循環多餘碼區 (CRC Field)

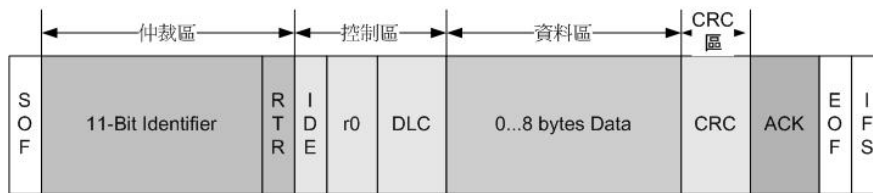


圖 2.2 標準 CAN 資料欄框格式

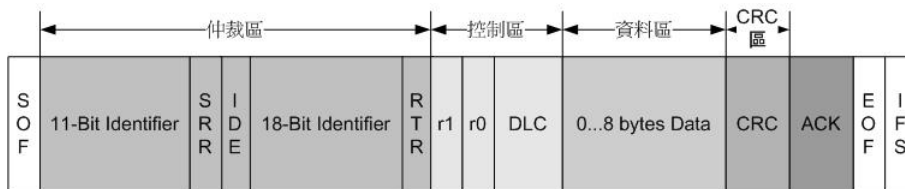


圖 2.3 擴展 CAN 資料欄框格式

以下將介紹資料欄框的各區域：

◆ SOF (Start of Frame)

起始欄框，資料欄框及遙控欄框的起始，由一個 dominant bit 組成，當匯流排閒置的時候才可以開始新的資料或是遙控的欄框。

◆ Identifier

識別碼，識別碼是用來決定訊息優先權，當識別碼越低的時候優先權越高，在標準 CAN 中識別碼為 11 bits，在擴展 CAN 中識別碼為 29 bits。

◆ RTR (Remote Transmission Request)

遙控傳輸請求，RTR 為 dominant bit 的時候此欄框為資料欄框，當 RTR 為 recessive bit 的時候此欄框為遙控欄框。

◆ SRR (Substitute Remote Request)

替代遙控請求，在擴展 CAN 中用來取代標準 CAN 中的 RTR 位置，為一 recessive bit，當標準 CAN 和擴展 CAN



起衝突時由於 SRR 回 recessive bit，根據 CAN 的仲裁方式標準 CAN 的優先權大於擴展 CAN。

◆ IDE (Identifier Extension)

當此 bit 為 dominant 時，此訊息的識別碼為標準 CAN 的識別碼，當此 bit 為 recessive 時，此訊息的識別碼為擴展 CAN 的識別碼。

◆ r0, r1

保留位元，必須為 dominant 位元。

◆ DLC (Data Length Code)

傳送資料的長度，由四個位元所組成。

◆ Data

欲傳送資料的存放地方，最少可傳送 0 個位元組，最多可傳送 8 個位元組。

◆ CRC (Cyclic Redundancy Check)

循環冗餘檢查，用來確認傳送資料是否正確。

◆ ACK (Acknowledgement Filed)

確認區，由兩個位元組成，分別為 ACK Slot 和 ACK 確認邊界，一開始訊息送出去的時候都是 recessive bits，當匯流排上面有任何一個節點成功接受訊息，則將 ACK Slot 的 recessive bit 改成 dominant bit，如此發送端可以藉由偵測 ACK Slot 來判斷訊息是否有正確被接收。

◆ EOF (End of Frame)

終止欄框，由七個 recessive bits 所組成，表示一個資料欄

框的結束。

#### ◆ IFS (Inter-frame Space)

擁有 7 個 bits，其用途主要是用來區隔資料欄框或是遙控欄框，以提供節點處理資料的時間。

#### ● 遙控欄框 (Remote Frame)

用來請求其他節點傳送所需要的資料，遙控欄框和資料欄框相似，但是有兩點不同的地方，第一個不同點是遙控欄框 RTR 為 recessive bit，另外一個不同點是遙控欄框是不帶任何資料。

#### ● 錯誤欄框 (Error Frame)

錯誤欄框由兩個部份所組成，第一個部份是錯誤旗標 (error flag)，第二個部份為錯誤邊界 (error delimiter)。為了正確終止 error frame，「錯誤被動 (error-passive)」的節點要求匯流排最少要有 3 bits 時間空間 (idle) 下來，因此匯流排的負載不應為 100%。

錯誤旗標 (error flag) 有兩種形式，一種是由連續六個 dominant bit 所組成的主動錯誤旗標，另一種則是由六個 recessive bit 所組成的被動錯誤旗標。

「錯誤主動 (error-active)」節點偵測到錯誤訊號時，會發出主動錯誤旗標，此旗標會違反位元填充的規則或是破壞網路欄框的固定格式 (如 ACK 或是 EOF)，其他的節點在偵測到錯誤訊號時也會跟著發出錯誤旗標。因此由各個節點發出的錯誤旗標疊加在一起所形成的 dominant bit 的序列可以確實在匯流排上偵測到。

「錯誤被動 (error-passive)」節點偵測到錯誤訊號，會發出被動錯誤旗標，當「錯誤被動」節點等待六個連續同極性的位元，當這六個位元被偵測到的時候，被動錯誤旗標發送就完成了。

錯誤邊界 (error delimiter) 由 8 個 recessive bits 組成，當完成錯誤旗標的傳輸後，節點會先發出一個 recessive bit 並且開始偵測匯流排直到偵測到第一個 recessive bit 為止，在偵測到第一個 recessive bit 之後節點繼續把剩下的七個 recessive bits 發送出去。當所有節點偵測到 8 個 recessive bits 之後，錯誤欄框及傳送完成。

- 過載欄框 (Overload Frame)

過載欄框由過載旗標和過載邊界所組成，其格式跟錯誤欄相似，過載旗標由六個 dominant bits 所組成，過載邊界由八個 recessive bits 所組成。節點會在三個情況下發出過載欄框：

- ◆ 當接收器在下筆資料欄框或遙控欄框進來之前要有一個延遲時間的時候。
- ◆ 當節點在中斷欄框期間偵測到 dominant bit 的時候
- ◆ 當節點在錯誤邊界及過載邊界的第八個 bit 採樣到 dominant bit 的時候。

在第一種情況，過載欄框只能由接收節點在中斷欄框裡的第一個位元開始發出。第二、三種情況則是在偵測到第一個 dominant bit 之後的位元開始發出。在發出過載旗標後緊接著發送過載邊界，如此即可發送完成。

### 2-1-3 CAN 的錯誤處理

CAN 的錯誤類型有五個種類，分別為位錯誤 (Bit Error)、填充錯誤 (Stuff Error)、循環冗餘碼錯誤 (CRC Error)、格式錯誤 (Form Error)、確認錯誤 (ACK

Error)，下面將針對這五種類型做介紹：

- 位錯誤

節點在發送訊息的同時也會偵測匯流排上面的狀況，當發送的值與偵測的值不一樣時，此時則產生一個位錯誤。但是在仲裁區 (arbitration filed) 時，某一節點所發出的 recessive bit 被另一節點發出的 dominant bit 所覆蓋的時候，或是在確認區 (acknowledgement filed) 時，發送節點送出的 recessive bit 被其他接收節點發出的 dominant bit 所覆蓋的時候，不會被認定為位錯誤。

- 填充錯誤

若在位元填充 (bit stuffing) 時期偵測到連續六個相同位準的位元時，則產生填充錯誤。

- 循環冗餘碼錯誤

CRC 序列中包含發送方節點 CRC 的計算結果，假如接收節點收到的 CRC 序列與計算的結果不同時，則產生循環冗餘碼錯誤

- 格式錯誤

當一個固定形式的區域含有一個或多個非法位元的時候則產生格式錯誤。

- 確認錯誤

當在 ACK SLOT 所監視的位元不為 dominant bit 的時候，則產生確認錯誤。

當偵測出以上五種之一的錯誤的時候，檢測到錯誤的節點就會發出錯誤欄框，「錯誤主動 (error-active)」節點發出主動錯誤旗標，「錯誤被動」節點則發出被動錯誤旗標。

CAN 在故障界定上有三種狀態，分別是

- 錯誤主動 (error-active)


可正常參與匯流排上面的通訊，並且在錯誤的時候發出主動錯誤訊號。

- 錯誤被動 (error-passive)

不可以發送主動錯誤訊號，可參與匯流排的通訊，但是錯誤產生時只能發送被動錯誤訊號，發送後「錯誤被動」在初始下一個傳送之前處於等待狀態。

- 離線 (bus-off)

若節點處於離線狀態，則不允許該節點對匯流排上面有任何影響。



以上三種狀態的區別，主要以該節點產生錯誤多寡來決定，每個節點都有一個傳送錯誤計數器與和一個接收錯誤計數器，當一個節點成功傳送或接收一個欄框的時候則其對應的計數器減一，若發生錯誤則對應計數器加一。當節點中的任何一個計數器值超過 127 的時候，此節點便進入「錯誤被動」的狀態，若超過 255 的時候則此節點進入離線狀態，計數器不超過 127 的時候即為「錯誤主動」的狀態。若在匯流排上監視到連續 11 個 recessive bits 的時候，則連線的節點可以變成「錯誤被動」的狀態，藉由這三種狀態，可以限制錯誤頻繁的節點對網路存取權與整個系統的影響，使整個網路不至於因為少數頻繁出現錯誤的節點而癱瘓。

## 2-2 乙太網路(Ethernet)介紹

乙太網路(Ethernet)源自於 Xerox 公司的區域網路系統，一開始速度只有

2.49Mbps，僅在 Xerox 公司內部使用，隨後由 DEC、Intel 和 Xerox 三家公司共同參與標準之改進與擴展，並且發表了 Ethernet Version 2(EV2)規格，將網路頻寬提升到 10Mbps，之後由 IEEE 根據 EV2 的內容，在 1983 年通過了 802.3 CSMA/CD 規格，從此成為使用最廣泛的區域網路標準之一。

乙太網路使用 CSMA/CD(載波感應多重存取及衝突偵測)方式來存取介質上面的資料，在 CSMA/CD 裡，某一站要送資料前必須檢查是否別有別的佔正在傳送資料，假如沒有其他佔要傳送資料，則這一監聽站就可以將資料傳送到網路上面。在乙太網路上面可能會遇到多站同時要傳遞資料的情況，此情況會造成資料的碰撞。因此所有的站必須持續監聽網路及偵測出任何可能的碰撞。若產生碰撞，所有站放棄所接收到的資料，原先傳送的各站退後等待一段時間再重送。為了降低再次碰撞的可能，每個傳送站各自產生一個亂數來決定要等待多久時間才能傳送。

乙太網路在實際使用上可以為匯流排或是星形方式連接，所有乙太網路的組態在邏輯上是一種匯流排。每個訊框(Frame)可傳送資料到網路線上的各站，但只有被定址到的那個站才能讀到訊框。在實現上目前使用最多的實現方式為 10BASE-T(雙絞線乙太網路)，10BASE-T 是使用星形方式連接各站，速度為 10Mbps，最長傳輸距離為 100 公尺。除了 10BASE-T 之外，還有 10BASE5(粗纜乙太網路)和 10BASE2(細纜乙太網路)，上述兩種均使用匯流排傳輸方式做連接。

目前使用上比較普遍的是快速乙太網路(Fast Ethernet)，快速乙太網路是比乙太網路更新一點的版本，最高速率為 100Mbps 跟乙太網路比較起來，在訊框上沒有做任何改變，唯一改變的是傳輸速度和碰撞區間的大小，傳輸速度提高 10 倍，因此碰撞區間也降低 10 倍。快速乙太網路和 10BASE-T 一樣採用星狀連接方式，其規範定義三種實體介質：100BASE-T4、100BASE-XF 和 100BASE-XF。

目前乙太網路最新的版本為 Gigabit Ethernet 支援傳輸速度高達 1Gbps，但是目前還是以快速乙太網路或乙太網路為主，Gigabit Ethernet 目前還不普及。

## 2-3 IEEE 802.11 介紹[14]

本章主要介紹 IEEE802.11，分別描述 MAC 層的運作機制，和網路使用模式。

### 2-3-1 MAC 層的運作機制

802.11 屬於 IEEE802 的成員之一，如圖 2.4 所示，而 IEEE802 主要規範 OSI 模型最底下兩層分別實體層(PHY Layer)和資料鍊結層(Data Link Layer)。只要是 802 系列的網路，必具備上述兩種元件，其中 MAC 用以決定如何存取媒介與傳送資料的規則，至於傳送與接收的細節，交由 PHY 負責處理。

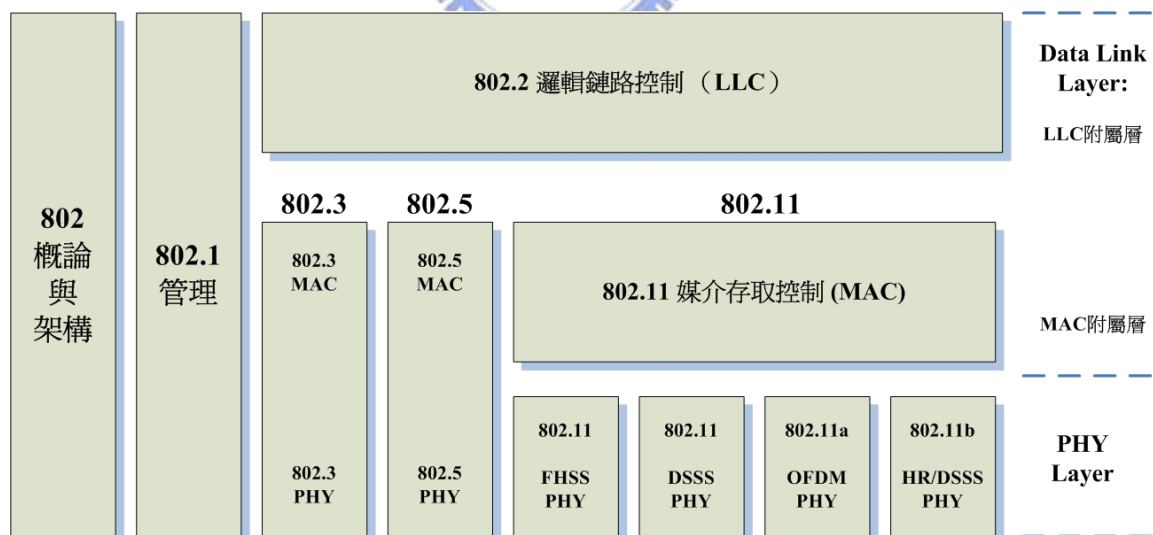


圖 2.4 IEEE 802.11 家族與 OSI 模型的關係

802.11 規格的關鍵在於媒介存取控制層(MAC)，MAC 層位於實體層之上，控制資料的傳輸，負責核心的訊框封裝作業(Core Framing Operation)，以及與相

連接的骨幹網路做必要的互動。至於實體層(PHY)，則可能提供不同的傳輸速度。

相較 2-2 節提到屬於 802.3 的 Ethernet，802.3 是載波感應多重存取/衝突偵測 (carrier sense multiple access network with collision detection；CSMA/CD)，802.11 同樣採取載波感應多重存取機制來控制傳輸媒介的存取，不過由於底層媒介不同，而 802.11 無法達到碰撞偵測的要求，為了避免封包碰撞浪費傳輸資源，因此 802.11 轉而使用載波感應多重存取/碰撞避免(carrier sense multiple access network with collision avoidance；CSMA/CA)，然而和 Ethernet 一樣，802.11 採用不具中樞控制功能的分散式存取機制，因而每部 802.11 工作站存取媒介的方式都一樣，主要差異在於所使用的底層媒介不同而已。

## 2-3-2 操作模式

IEEE 802.11 提供兩種操作模式：(1) Ad-hoc 模式，(2) Infrastructure 模式，以下為兩種模式的介紹。



### (1) Ad-hoc 模式：

使用此模式建立的網路，通常是因應各種特殊需求所建立的特設網路 (ad-hoc network)，由於此網路具有點對點的性質(peer-to-peer)，所以又稱為點對點網路，具有便利性和移動性，不必像 infrastructure network，必須事先架設基礎設備，即可提供工作站之間的傳輸。(註：ad-hoc 為拉丁文，原意為「特別的」、「針對特殊情形的」。)

### (2) Infrastructure 模式：

使用此模式建立的網路，必須使用基地台(access point；AP)負責傳輸範圍內的所有工作站通訊行為，通訊行為包括兩個步驟，首先由起始對話的工作站將訊框傳遞給基地台，然後再由基地台將此訊框傳送到目的地的工作站，如此意味著所有的通訊都必須透過基地台運作。



平時使用的無線網路即是使用此模式來與 Ethernet 做連結，透過基地台可以做異類網路的連接，並且協助工作站節省電力，甚至是更進一步的網路服務。常見的例子即是速食店、學校、機場等提供的無線上網服務，或是最近台北提出的無線城市服務皆是中控型網路(infrastructure network)的典型例子。

在本論文中提到的 Client 端利用無線網路控制馬達的例子時，我們選用的模式為 Ad-hoc 點對點的模式。

## 2-4 TCP/IP 網路通訊協定介紹[10, 15, 16]

TCP/IP 主要是用來連結網路上的電腦主機，作為網路傳輸資料的標準協定，是一套完整的通訊協定，他的名稱來自於其中兩個最重要的協定：傳輸控制協定(transmission control protocol, TCP)及網際網路協定(internet protocol, IP)，除了這兩種協定外，TCP/IP 還包含了其他協定，但最重要的就是 TCP 和 IP 兩種協定。

TCP/IP 是一種階層式的協定，這裡所謂的階層式是指每一個較高的協定是由一個或多個較低分層協定所支援。這裡我們主要介紹網路層的網際網路協定(IP)和傳輸層的傳輸控制層協定(TCP)。

### (a) 網際網路協定(IP)

IP 是 TCP/IP 使用的傳輸機制，是一種非可靠性、非預接式的資料封包協定，只提供盡量傳送的服務。所謂盡量傳送是指 IP 沒有提供錯誤檢查或追蹤，IP 假設他的底層是不可靠的而盡力將資料傳送到目的地，但不保證是否傳到目的地。IP 以資料封包方式傳遞，而各個資料封包分別傳送，資料封包在傳輸過程可以經過不同的路徑，可以不按順序到達，也可以被重複，當資料封包達目的時，也沒有方法將資料封包調整回原來的順序。然

而 IP 的限制不應被視為缺點，IP 提供骨幹傳輸的功能，且讓使用者自由加入所需要的功能，因而獲得較高的效率。

#### (b) 傳輸控制協定(TCP)

TCP 提供完整的傳輸服務給所需的應用程式，在應用層與網路層之間提供應用程式和網路運作的中介服務。TCP 在傳輸層中提供了流量控制和錯誤控制的機制，TCP 使用滑動窗口(sliding window)的協定來做流量控制，使用回應封包、計時(time-out)、重送的機制來做錯誤控制。

TCP 為應用程式提供連線的機制，應用程式能夠以資料流方式傳送訊息給傳輸層，在傳送端的傳輸層負責與接收者建立一個連線後，將資料拆成可傳送的單元並給予序號，再將傳送單元一個一個傳送出去。接收端的傳輸層等待這些資料的到來，然後加以檢查，並將沒有錯誤的資料以資料流的方式傳給接收端的應用程式，等全部資料完成後，TCP 要關掉剛剛使用的連線。TCP 是一種連線導向，具可靠的傳輸協定。TCP 把 IP 的服務加入連接導向及可靠性這兩種特點。

## 第三章 CAN 網路的監控系統設計

CAN 網路只有定義資料欄框的格式以及底層傳輸的規範，並沒有定義資料欄框中資料(DATA)部份要如何分配使用，一切都有使用者自己去定義，因此必須建立一個有系統的使用方式，才可以讓我們開發 CAN 網路程式的時候更加便利。另一方面當要對分散式網路做除錯時，是一件不容易的事情，在此我們建立一個 CAN 監控程式再搭配我們建立 CAN 網路系統的方式以方便我們做 Debug 和程式開發，本章分成二部份 3-1 為 CAN 網路的建立，3-2 為 CAN 網路監控程式的建立。

### 3-1 CAN 網路的建立

目前市面上有幾種以 CAN 為基礎建立來建立較高層的協定 CANOpen、Device Net 等相關協定。這些協定都是方便使用者在應用層開發程式，比方說，假如發送訊息長度大於 8Byte 時候，這些協定都有提供類似處理的方法，但是在本論文當中系統沒有這麼複雜，我們並不需要這麼複雜的協定。本論文建立 CAN 網路系統只需事先將 ID 和資料欄框中的資料區規劃好即可。

#### 3-1-1 ID 分配

CAN 網路上面可以有很多個節點，每個節點設定可以設定多組想要接收的 ID。但是不將這些節點可接收的 ID 規劃好，等系統複雜之後，很難去查出某筆訊息的試給誰接收或傳送，當系統有問題要除錯是一個麻煩。

因此本論文裡面的 CAN 網路系統，規定每個節點只可以接收一種識別碼 (Identifier, ID) 的資料，其他 ID 的訊息一概不接收，此訊息的 ID 就當作是該節點的識別 ID，每個節點的識別 ID 是唯一的，其想法有點像是網際網路的 IP，當要傳送訊息給對方時，該訊息裡面包含對方 IP，這樣只有是該 IP 的接受方才會接收資料。若要追蹤此訊息是傳給那個節點，只要看這筆訊息上面的 ID 即可

得知。舉個例子，當有 A、B、C 三個節點，其識別 ID 分別為 1、2、3，當 C 要傳給 A 節點資料的時候，只要將資料欄框裡面的識別碼設定為 1 再將資料傳到 CAN 網路上面，由於各節點都有用 MASK 設定各自可接受的 ID，因此該筆訊息只有節點 A 可以接收到。其他節點因為訊息被 MASK 濾掉，這樣也不需要先將訊息讀入再判斷該筆資料是否是給他的。

除了唯一的識別 ID，每個節點再多規劃一個廣播用 ID，當訊息上面的識別 ID 使廣播用的 ID 時，每個節點都可以同時收到該筆訊息，這是為了當某個點要對多數節點傳送訊息而準備。例如，當有個節點發生錯誤的時候，可以利用廣播用 ID 通知 CAN 網路上每個節點系統有錯誤發生，應該採取對應的措施，而不用再對每個節點一個一個通知，以節省時間。

由上面的規劃，本論文中 CAN 網路系統每個節點最少都可以接收兩種不同的 ID，一種為系統本身的識別 ID，另外一種為廣播用 ID。

### 3-1-2 使用者自訂欄框

先前有提過 CAN 網路沒有對資料欄框中資料的部份做分配，因次在本論文的系統當中，我們自行規劃三種自訂欄框，節點跟節點之間溝通的訊息都靠這三種欄框，圖 3.1 分別為三種欄框的格式和大小。其中三個欄框都有個共通點，就是最前面會帶有 Type 和 Device 的欄位，大小均為 1Byte，Type 指的是該筆

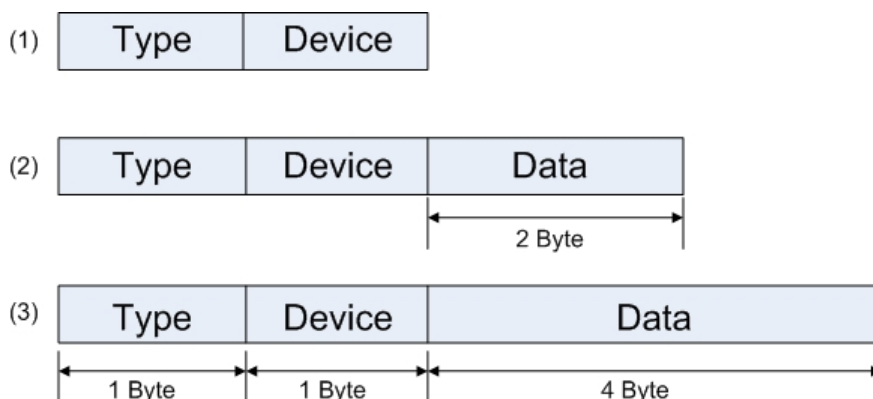


圖 3.1 三種自訂欄框

訊息代表的事件，Device 表示由哪個節點傳送，由於我們系統的訊息事件和節點不多，所以 Type 和 Device 都取 1Byte，1Byte 可以代表 256 筆訊息事件和 256 個節點。三個欄框唯一不同之處就是 Data 部份，依據不同需求來選擇不同長度的 Data 大小。設定好自訂欄框之後，將此欄框封裝到 CAN 封包 Data 的部份，在將封包傳送到對應的節點，對應的節點收到封包將自訂欄框取出後，解析該欄框後做出對應的事件。

## 3-2 CAN 網路監控程式介紹

CAN 網路上的訊息傳輸都是利用廣播的方式將訊息傳到網路上去，然後利用個節點的 MASK 設定將想要收到訊息給收進來，我們網路監控程式就利用此特性，來監看網路上所有訊息，並且可以利用遮罩(Mask)的設定，將不想要看的訊息給濾除掉。本監控程式要搭配 USB CAN 使得 PC 變成 CAN 網路上的一個節點才能監看網路上的訊息。3-2-1 介紹 USB CAN，3-2-2 介紹 USB CAN 軟體介面的建立，0 介紹 CAN 網路監控程式。

### 3-2-1 USB CAN 介紹[17]

USB CAN 可以讓 PC 透過 USB 介面連接到 CAN 網路上面，使得 PC 也變成 CAN 網路上面的一個節點，透過 USB CAN 方便讓電腦收集和分析 CAN 網路上面的訊息，或是透過 USB CAN 傳達訊息到 CAN 網路上各節點。本章採用的 USB CAN 為周立功單片機發展有限公司所開發出來的 USBCANII 智能 CAN 接口卡，USBCANII 有兩個 CAN 通道，可讓使用者同時連到兩個不同的 CAN 網路上面，並且附有 ZLGVCII 接口函式庫，方便使用者搭配不同開發程式開發符合自己需求的產品，以下為 USBCANII 外觀和硬體規格：



(a)



(b)

圖 3.2 USBCANII 外觀 (a)正面 (b)背面

硬體規格：

- PC 接口：USB1.1
- CAN 控制器：PHILIPS SJA1000
- CAN 收發器： PHILIPS PCA82C250
- CAN 網路傳輸速率：5Kbps~1Mbps
- CAN 通訊接口：DB9，符合 DeviceNET 和 CANOpen 標準
- 支援 CAN 網路協定：支援 CAN 2.0B(兼容 CAN2.0A 協定)，符合 ISO/IS 11898
- 最高 Frame 流量：每個通道 5000 Frame/sec
- 供電方式：USB 供電或是外接+9V~+25V，400mA 的電源

### 3-2-2 USB CAN 軟體介面的建立

USBCANII 有提供接口函式庫方便使用者開發符合自己所需要的程式，要使用這些函式需將原廠所附的三個文件 ControlCAN.h、ControllCAN.lib、ControlCAN.dll 和一個資料夾 kerneldlls 加到開發的程式中。

通常要使用廠商所提供的動態連結檔有兩個方式[18]：

(1) Implicitly Link (隱式連結) 又稱靜態載入，靜態載入方式所使用的 dll 檔

會在應用程式執行時載入，然後就可以呼叫所有由 dll 檔中匯出的函式，就好像是包含在程式一般，此方法較簡單，載入方式是由編譯器負責處理，但是編譯時需額外的 import library (.lib)，本節一開始的 dll 檔使用方式即為此方式。

- (2) Explicitly Link (顯式連結) 又稱動態載入，動態載入就是在程式需要時候才將 dll 檔載入，不需要的時候即釋放 dll 檔，此方法可以更有效的使用記憶體，應用程式載入時的速度較快，不需額外加入 lib 檔，但是使用方式複雜。

當我們依照一開始介紹的方法在 Borland C++ Builder 上面使用靜態連結的方式將 dll 檔載入，則會產生問題無法使用。會產生問題的原因出在靜態載入的時候需要使用到 lib 檔，lib 檔分為兩種格式：

- (1) COFF 格式：Microsoft 所使用。
- (2) OMF 格式：Borland 所使用。

USB CAN 所附 lib 檔是給 Visual C++ 使用，所以直接拿來 Borland C++ Builder 上面使用會出現問題，由於 lib 檔無法使用我們只好採用動態連結的方式來載入 dll 檔。

接下來介紹動態載入的流程，首先要使用 WINDOWS API 裡面的 LoadLibrary 手動載入 dll 檔，並且使用 GetProcAddress 來取得所要使用的函式的位址，最後不需要此 dll 檔的時候，使用 FreeLibrary 將 dll 釋放掉，下面就使用接口函式庫所提供的一個函式當作例子：

```
DWORD __stdcall VCI_OpenDevice(DWORD DevType, DWORD DevIndex,  DWORD Reserved)
```

功能：用來打開 USB CAN 設備

(1) 使用 LoadLibrary 載入 dll 檔：

```
HINSTANCE hDLL;  
hDLL = ::LoadLibrary("ControlCAN.dll");
```

(2) 取得函式位址：首先用 typedef 把 VCI\_OpenDevice 定義成一個自訂函式指標的型別然後在宣告一個 vciOpenDevice 函數指標，最後利用 GetProcAddress 取得函式位址並指定給 vciOpenDevice，如此一來就可以使用 vciOpenDevice 函式就像是使用靜態載入的 VCI\_OpenDevice。

```
DWORD __fastcall OpenDevice(DWORD DeviceType,DWORD DeviceInd,DWORD Reserved);  
VCI_OpenDevice vciOpenDevice;  
vciOpenDevice = (VCI_OpenDevice)GetProcAddress(hDLL,"VCI_OpenDevice");
```

(3) 釋放 dll 檔：假如程式不再使用 dll 檔可用 FreeLibrary 將不使用的 dll 檔釋放掉。

```
FreeLibrary(hDLL);
```

使用動態載入的方式可以順利使用 USB CAN 所提供的接口函式，但是由上面的例子不難發現，使用 USB CAN 接口函式庫的其中一個函式要先經過複雜設定之後才可以使用，若要再使用其他的函式還要經過相同複雜的設定，會使得原本的程式更加複雜難懂，為了減低程式的複雜度和使用方便，我們建立一個 USBCAN 的類別，將上面設定複雜的動作和接口函式庫都包在這個類別內，只要宣告此物件就可以直接使用接口函式庫，不需要再去做複雜的設定。下面簡單介紹物件的建立過程，並且只介紹 VCI\_OpenDevice 如何加入此物件，若要使用其他函式可用相同方法加到此物件（其他函式用...來代替）：

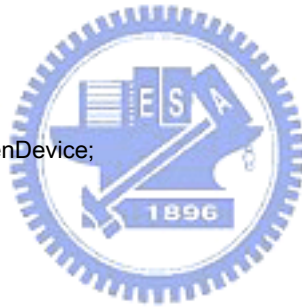


### [1]用 typedef 自訂函式指標的型別

```
DWORD __fastcall OpenDevice(DWORD DeviceType,DWORD DeviceInd,DWORD Reserved);  
...  
...
```

### [2]定義 USBCAN 類別

```
class USBCAN  
{  
public:  
    __fastcall USBCAN(void);  
    __fastcall ~USBCAN(void);  
    DWORD __fastcall OpenDevice(DWORD DeviceType,DWORD DeviceInd,DWORD Reserved);  
    ...  
    ...  
private:  
    HINSTANCE hDLL;  
    VCI_OpenDevice vciOpenDevice;  
    ...  
    ...  
}
```



### [3]dll 的載入和函式位址的取得寫在 USBCAN 的建構子

```
__fastcall USBCAN::USBCAN()  
{  
    hDLL = ::LoadLibrary("ControlCAN.dll");  
    vciOpenDevice = (VCI_OpenDevice)GetProcAddress(hDLL,"VCI_OpenDevice");  
    ...  
    ...  
}
```

[4]dll 檔的釋放寫在 USBCAN 的解構子，如此當物件消失的時候會順便將 dll 檔釋放出來，避免 dll 檔一直佔用資源。

```
__fastcall USBCAN::~USBCAN()  
{  
    ::FreeLibrary(hDLL);  
}
```

[5]設定物件的方法:

```
DWORD __fastcall USBCAN::OpenDevice(DWORD DeviceType,DWORD DeviceInd,DWORD  
Reserved)  
{  
    return vciOpenDevice(DeviceType,DeviceInd,Reserved);  
}  
...  
...
```

經過以上設定後，就可以完成 USBCAN 類別的建立。下面將介紹如何使用此類別來建立物件開啟 USB CAN 設備（此類別其他的方法將放在附錄介紹）。

```
USBCAN UsbCanObj; //建立 UsbCanObj 物件
```

```
UsbCanObj.OpenDevice(VCI_USBCAN2,this->DevIndex,0) //開啟 USB CAN Device
```

### 3-2-3 網路監控程式

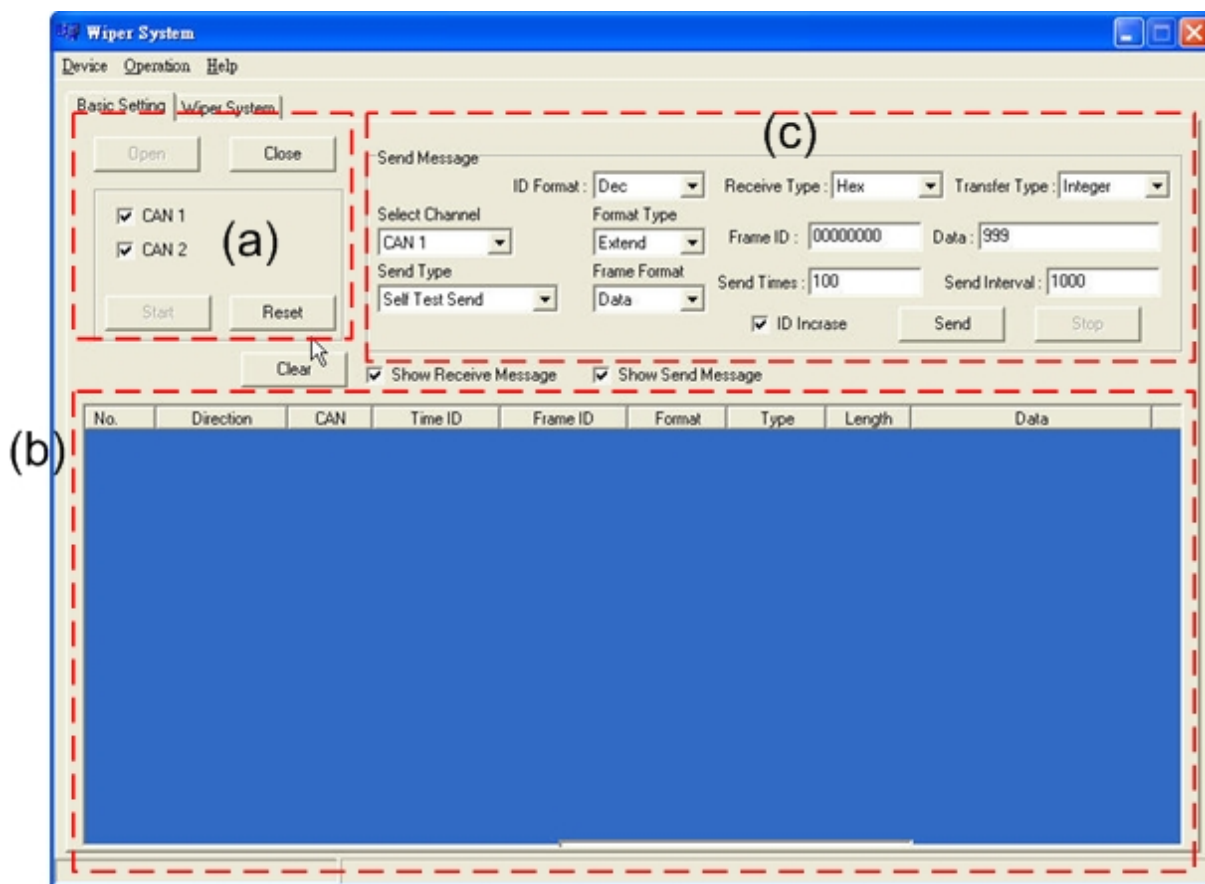


圖 3.3 網路監控程式主畫面

圖 3.3 為我們利用 3-2-2 提到的軟體介面所建立的網路監控程式的主要畫面，總共分成三個部份：

(a) 控制 USBCAN 節點屬性：

當要開始使用監控程式的時候，需先將 USBCAN 相關參數設定好，首先，先點下 Open 按鈕後會跳出參數設定視窗如圖 3.4，此畫面可以讓我們設定 ID 遮罩，濾波器形式，工作模式還有傳輸速度，由於本 USBCAN 使用的 CAN Controller 為 PHILIPS SJA-1000，因此參數的詳細設定，可以直接參考 SJA-1000 的說明文件。此 USBCNA 有兩組 CAN 節點，所以有兩組參數可供設定，Device Index 是當 PC 不只接上一台 USBCAN

時，可以讓我們選擇想要控制的裝置。當設定好參數之後，選擇 OK 跳回主畫面，此時再按 Start 按鈕即可開始監聽 CAN 網路上面的訊息。

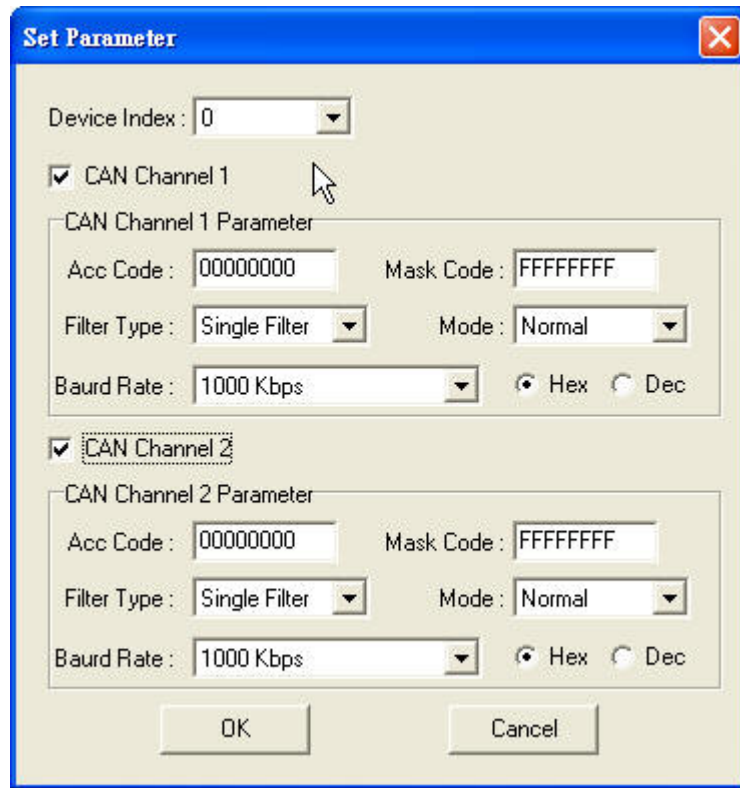


圖 3.4 USB CAN 系統參數設定畫面

(b) 監聽 CAN BUS 訊息的顯示部份：

監控程式將在 CAN Bus 上面監聽到的訊息會在此部份顯示出來，在此我們可以看到訊息的

- (1) direction：傳遞方向，讓我們知道該訊息是由 PC 端接收或是發送。
- (2) CAN：接收或傳送節點，由於 USBCANII 可以讓 PC 擁有兩個 CAN 節點，此部份可以顯示出是哪個節點收發該訊息。
- (3) Time ID：顯示時間 Tag
- (4) Frame ID：顯示該訊息的識別碼，根據 3-1 節的規劃此 ID 是將

接收此訊息的識別 ID，可以藉由 Frame ID 來斷該訊息的去處。

(5) Format：顯示該訊息是資料欄框(Data Frame)或是遙控欄框  
(Remote Frame)

(6) Type：資料欄框的訊息格式，該看訊息是 Extend 或是 Standard  
的格式

(7) Length：該訊息所帶的資料長度

(8) Data：該訊息所攜帶的資料內容，可搭配(c)部份的 Receive  
Type 來顯示該資料對應的格式。

(c) 傳送訊息部份：

傳送部份可以讓 PC 節點不只扮演監聽的角色，也可讓 PC 主動  
發出訊息，藉由 PC 發出訊息來看對應的節點是否有對該訊息做出正常  
的反應。在此部份可以設定訊息傳送的種類，由哪個節點發出，等相關  
設定。同時也可以指定該節點，每隔多久時間所送多少筆訊息。

介紹完程式各部份之後，我們來看看實際使用的狀況，圖 3.5 為網監控程  
式監聽訊息的部份擷取畫面，系統訊息和 ID 有照 3-1 節所規劃的話，則麼我們  
可很方便快速的解讀此三筆訊息：

- (a) 第一筆為圖 3.1 的第一種欄框，訊息目的地是識別 ID 為 2 的節點，傳  
送該筆訊息節點是識別 ID 為 4 的節點，事件內容代號為 3。
- (b) 第一筆為圖 3.1 的第二種欄框，訊息目的地是識別 ID 為 5 的節點，傳  
送該筆訊息節點是識別 ID 為 2 的節點，事件內容代號為 5，後面 2Byte  
為該訊息帶有的 Data。
- (c) 第一筆為圖 3.1 的第三種欄框，訊息目的地是識別 ID 為 1 的節點，傳  
送該筆訊息節點是識別 ID 為 8 的節點，事件內容代號為 7，後面 4Byte

為該訊息帶有的 Data。

No.	Direction	CAN	Time ID	Frame ID	Format	Type	Length	Data
1	Receive	1	21:08.182.6	2	Data	Extend	2	03 04
2	Receive	1	21:33.989.7	5	Data	Extend	4	05 02 01 10
3	Receive	1	22:30.682.7	1	Data	Extend	6	07 08 00 00 12 35

圖 3.5 監控程式擷取畫面

利用有系統的建立系統識別 ID 和規劃好的自訂欄框，再搭配網路鍵控程式可以讓我們在比較簡單的方式掌握分散式控制 CAN 網路上面的訊息傳遞，當系統出問題時，也可以藉由觀測訊息傳遞狀況，來把問題找出。



## 第四章 網路延遲時間與量測

### 4-1 網路控制系統

回授控制系統若是其控制迴路及回授迴路都經由網路來做傳輸媒介，則這些控制系統稱為網路控制系統(networked control systems, NCSs)[2]，其中使用的網路可能是有線網路如 Ethernet，無線網路如 IEEE802.11，或是工業用網路如 CAN。透過網路來做控制，有許多有許多優點，例如[1]：

- (1) 節省配線，只需備妥網路環境，即可將不同的授控端連結在一起，網路上各個元件不需在個別拉導線回到控制端，就可以將資訊傳回主控端，如此也可以節省建制的時間
- (2) 模組化，每個連接在網路上的控制端或是授控端都可以被視為物件(object)，在一個系統裡面要增減一個物件，只需在軟體做設定即可。
- (3) 故障易於診斷及排除，通常網路化的元件，都會有自我診斷的功能，當元件故障，這些訊息能夠透過網路傳回主控台，然後只需將故障的模組汰換掉，換上相同的模組即可。
- (4) 增加控制的距離，可以使操作人員遠離危險環境，在遠端即可控制機器。

由上面可以知道，網路控制系統可以為我們帶來許多便利，但是將控制系統網路化之後，也帶來幾個缺點，例如時間延遲(delay time)，資料遺失(data dropout)，封包碰撞，取樣週期降低，網路排程(network scheduling)等問題，這些問題沒有處理好，輕則降低系統效能，重則整個系統產生不穩定或是無法控制，這些問題都是網路控制上的重要課題。

在本章我們主要探討時間延遲對系統造成的影響，4-2 節討論時間延遲對網

路系統的影響，並於第五章應用於 Smith predictor 消除時間延遲對系統的不良影響。

## 4-2 網路延遲時間(delay time)探討

圖 4.1 為一般網路控制方塊圖，在控制端與授控端之間的命令和回授訊號的傳輸是透過各種網路來傳輸，由網路來當作傳輸媒介就會產生延遲時間(delay time)。網路控制所產生的時間延遲分成兩類，一種為命令延遲時間(command time delay)是由控制端將命令傳到授控端的所產生的延遲時間，另外一種則是回授延遲時間( feedback time delay) ，回授延遲時間是指授控端將感測器的資料傳回控制端所產生的延遲時間[2]。

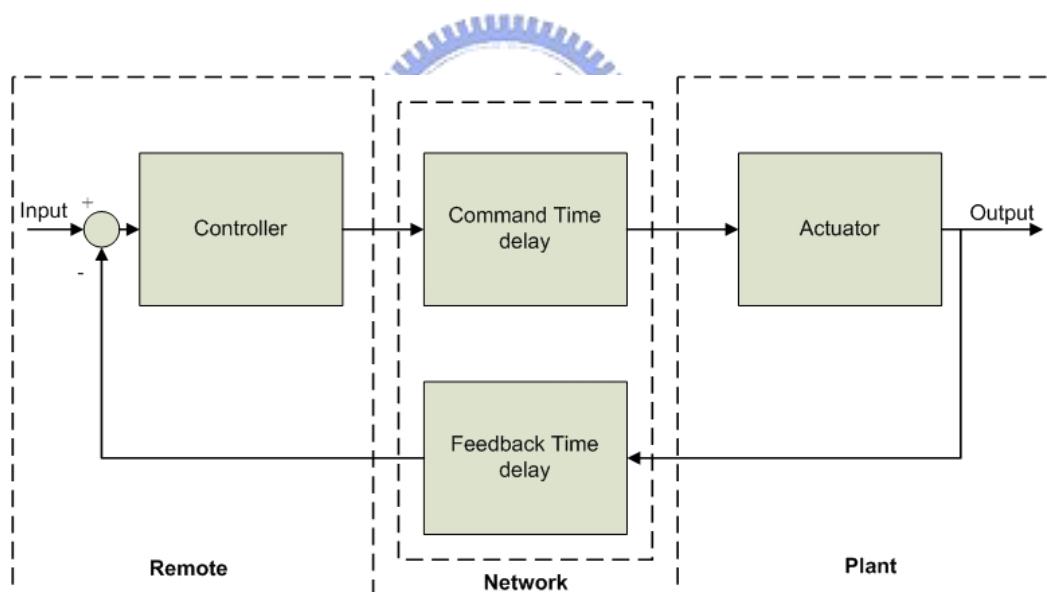


圖 4.1 網路控制系統方塊圖

延遲時間若小於系統的取樣週期，則我們可以將延遲時間忽略掉，但是延遲時間大於取樣週期，此時延遲時間對整個控制系統的影響，輕則會降低系統的效能，嚴重則會造成整個控制系統的不穩定。圖 4.2 和圖 4.3 分別為實際的網路控制系統在取樣頻率為 5 ms 時的響應圖，兩張圖中虛線部份是不經過網路控制的系統跑來的響應，實線部份則是經過網路來做控制的響應圖，圖 4.2 的



延遲時間為  $106.968\text{ ms}$  此時響應圖跟未受網路控制做比較，可以看出因延遲時間造成的系統震盪，若我們再將時間延遲加大到  $255.196\text{ ms}$ ，整個網路控制系統已經發散不受控制。

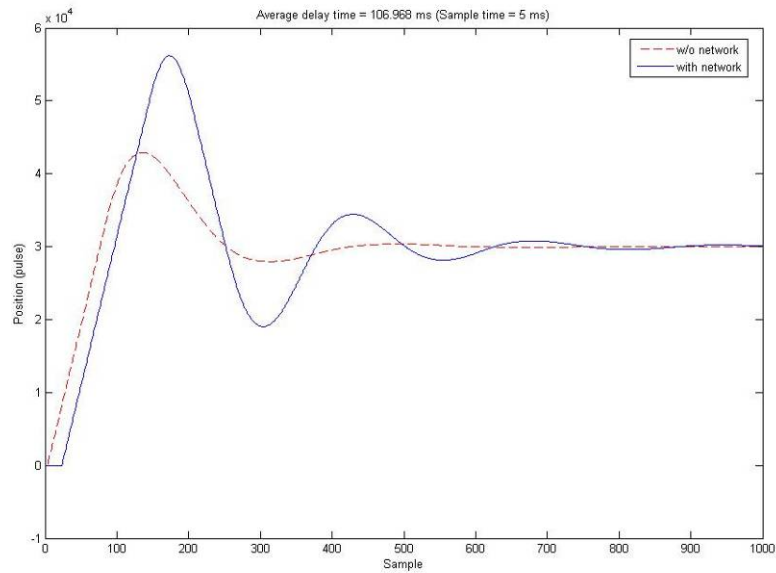


圖 4.2 延遲時間對系統影響圖(delay time =  $106.968\text{ ms}$ )

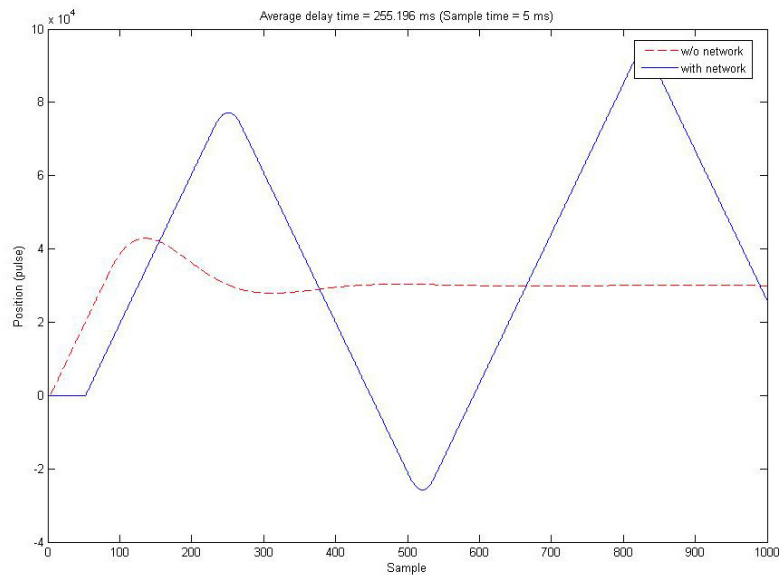


圖 4.3 延遲時間對系統影響圖(delay time =  $255.196\text{ ms}$ )

由上面兩張圖可以看出，在網路控制系統若不將延遲時間做妥善處理，則可能對我們系統造成無法控制和降低效能。

## 4-3 網路延遲時間測定實驗

### 4-3-1 網路延遲時間量測方法

在做 Smith predictor 實驗之前，我們必須先取得網路上面的延遲時間，我們這邊說的網路延遲時間是指圖 4.4 中  $T1+T2$  的總和， $T1$  就是 4-2 節所提到的命令延遲時間， $T2$  為回授延遲時間。其中  $T1$  和  $T2$  延遲時間為每個節點的應用層到另外一邊應用層加起來的總和，而不是單純指硬體層對硬體層之間的傳輸時間[6]，如圖 4.5。

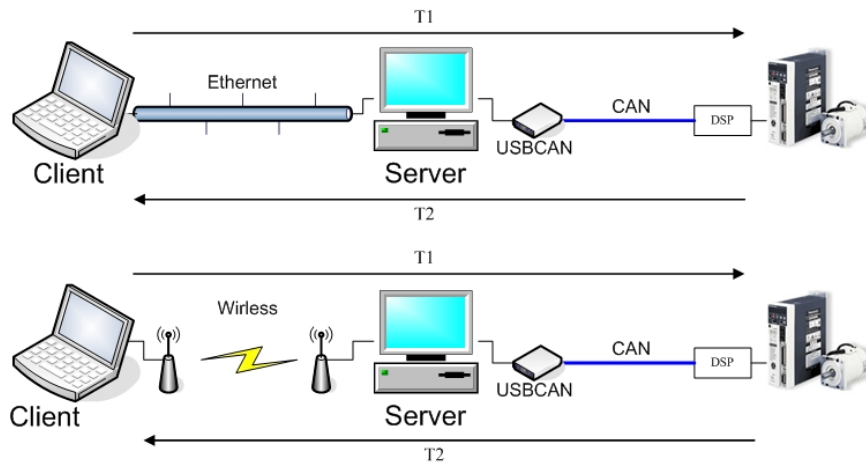


圖 4.4 網路系統延遲時間示意圖

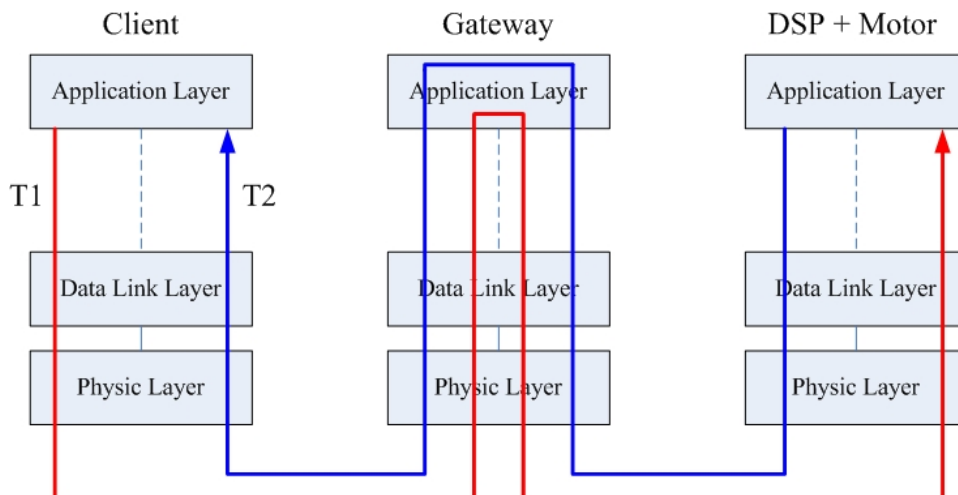


圖 4.5 延遲時間計算

量測延遲時間的方法是由 DSP 端固定一段時間傳送帶有 Index 的資料封包經由 Gateway 傳送到 Client 端，這裡 Index 指的是該封包的編號，封包傳送出

去前會記錄封包發送的起始時間，當 Client 端接收封包之後，立刻將封包原封不動傳回 DSP，DSP 接到封包後記錄到達時間，並且根據封包上面的 Index 找出發送時的起始時間，將到達時間跟起始時間相減即可求得延遲時間。

我們選擇 DSP 當作量測起始點，而不選用 Client 當作起始點，是因為 Windows 作業系統的 timer 只有 10 ms 的精確度，若由 Client 當起始點就得用作業系統 timer 來計數，10 ms 對我們控制系統而言，精確度略顯不足，為了求得更準確的時間，我們選用 DSP 來計算延遲時間，DSP 計數精確度可以達到 1 ms 以下，在本次實驗我們選用 1ms 為我們量測延遲時間最小的單位。

這次網路延遲時間實驗會包括以下的項目：(1) 網路控制系統延遲時間量測，(2) 固定延遲時間 (Constant delay) 的產生及量測，(3) 長距離的網路延遲時間量測。

#### 4-3-2 量測延遲時間人機介面

圖 4.6 為量測延遲時間的人機介面，可以設定每隔多久由 DSP 傳送一筆訊息和總共要儲存多少筆訊息，DSP 可以儲存的資料空間有限，假如傳送頻率很快的時候，很難做長時間的記錄，在此程式可以設定，每隔多久紀錄一筆資料，藉由這樣的方式來延長紀錄訊息的時間。以下為程式各部份的說明。

- (a) 設定想要連結 Server 的 IP 和 Port
- (b) 當延遲時間量測完成之後會顯示在此，或者是設定 Smith Predictor 裡面的延遲時間。單位為 *ms*
- (c) 測試的次數
- (d) 設定想要增加多少系統的延遲時間
- (e) 連結 Server 按鈕
- (f) 開始量測延遲時間

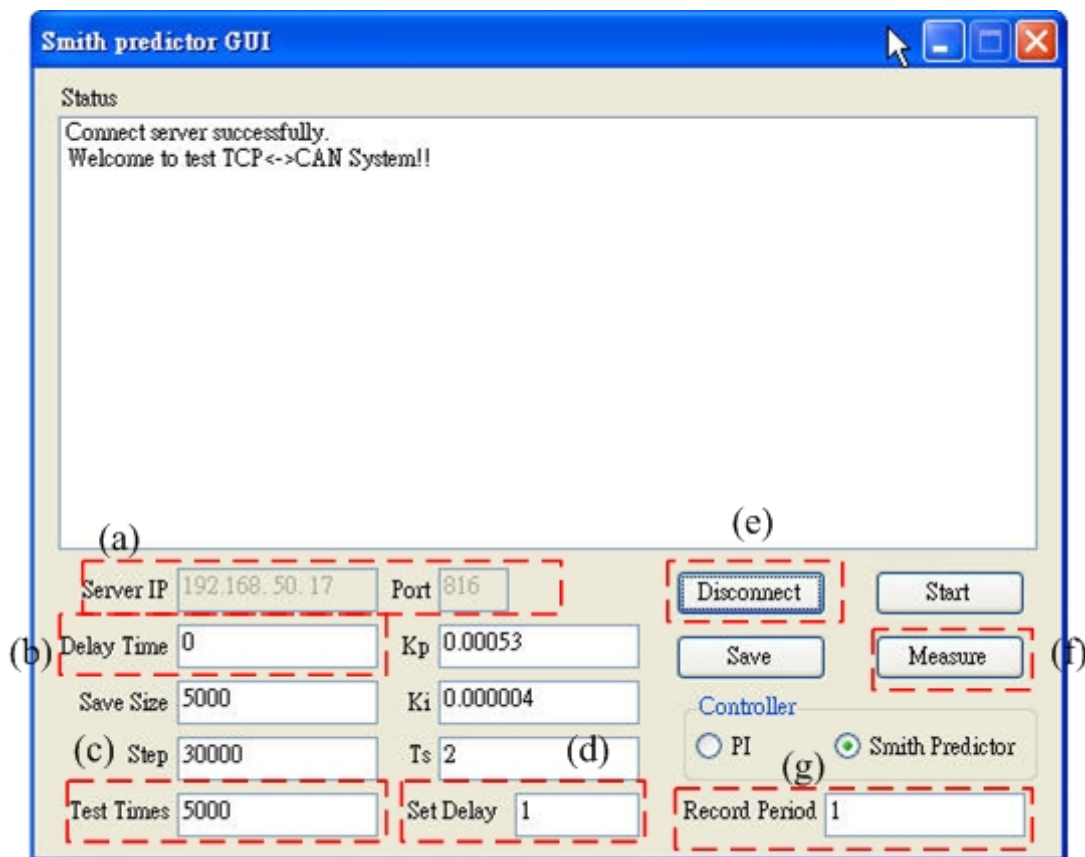


圖 4.6 量測延遲時間人機介面

### 4-3-3 網路控制系統的延遲時間

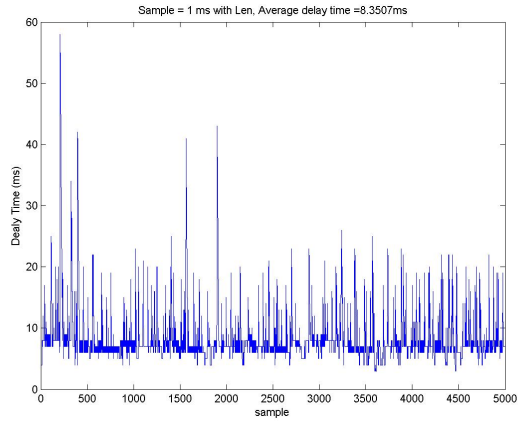
本次實驗是在 30 秒內固定時間傳送固定大小的訊息來量測我們網路控制系統的平均延遲時間，不同的傳送時間各做五次同樣的實驗。網路傳輸部份，Server 連接 DSP 是用 CAN 網路。Server 連接 Client 我們分別會使用 Ethernet 和無線網路，傳輸協定是使用 TCP/IP，Ethernet 傳輸速度為 100Mbps，Client 和 Server 之間透過一個 HUB 相連接。無線部份為 IEEE 802.11b，傳輸速率為 11Mbps，連結模式採用 ad-hoc 點對點的傳輸模式。

表 4-1 為使用有線網路 Ethernet 加上 CAN 網路在不同傳送頻率下所量測的平均延遲時間，由此表可以看出，在不同傳輸頻率下，平均的延遲時間也跟著不同，在傳輸頻率較低的時候，延遲時間較小平均值大約在 2~3 ms 之間，當網

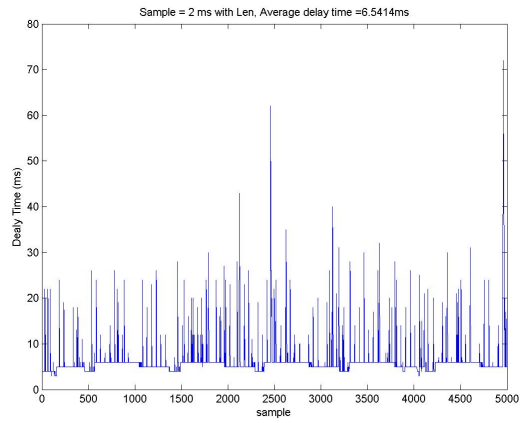
路傳送頻率變高的時候，延遲時間也跟著變大。若由同樣傳送頻率來看，此時平均延遲時間其實是差不多的，顯示還沒有到達頻寬限制的時候，在同樣傳送頻率的傳輸之下有相同的平均延遲時間，圖 4.7 為不同頻率下的時間分佈，傳送頻率高的時候延遲時間的變異較大，但還在一定範圍內，傳送頻率低的時候，變異較小，偶爾會有比較大的延遲時間可能是因為跟其他電腦的網路封包產生碰撞，使得延遲時間變大的關係

表 4-1 Ethernet+CAN 的平均延遲時間 (單位:ms)

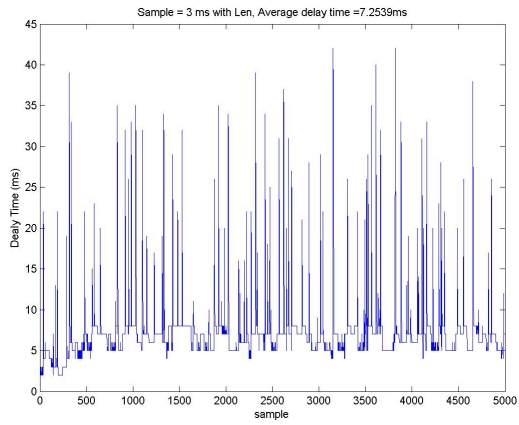
實驗群組 No. 傳送時間間隔 T	1	2	3	4	5
300 ms	2.707071	2.909091	2.757576	2.676768	2.626262
200 ms	2.966443	2.664428	2.610738	2.577181	2.409396
50 ms	2.560935	2.707846	2.679466	2.681135	2.961603
20 ms	2.731154	2.945964	2.72515	2.586391	2.639093
5 ms	7.380276	7.477095	7.411882	7.353271	7.35307
3 ms	7.465493	7.469894	7.291658	7.682536	7.35307
2 ms	6.58016	6.871774	6.741745	6.579316	6.607121
1 ms	8.653331	8.464092	8.388078	8.35447	8.90958



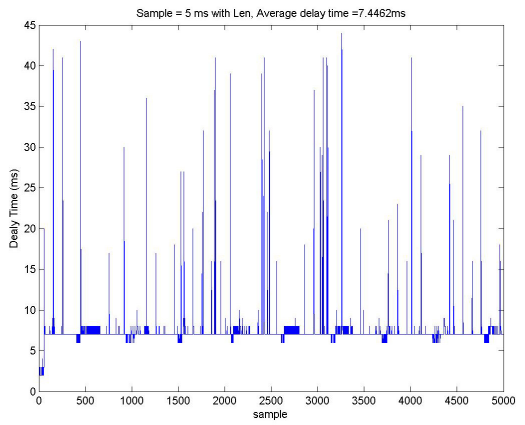
(a)



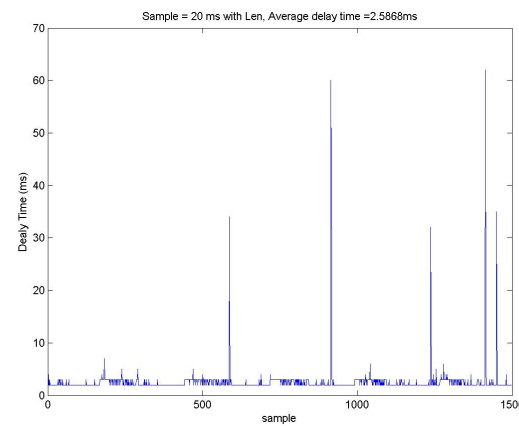
(b)



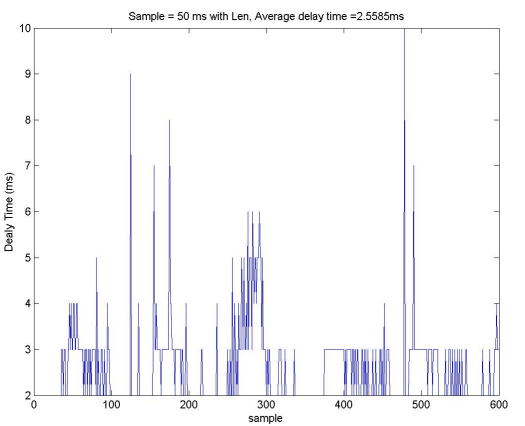
(c)



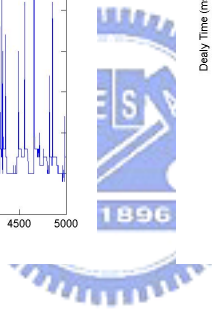
(d)

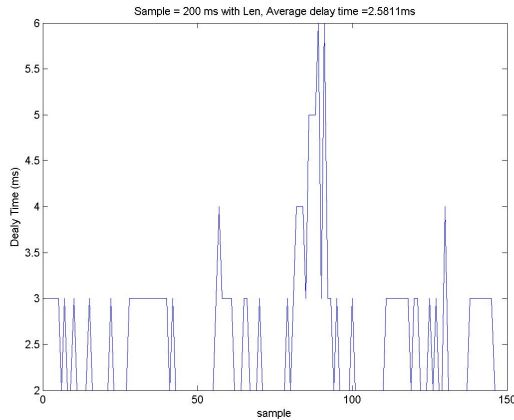


(e)

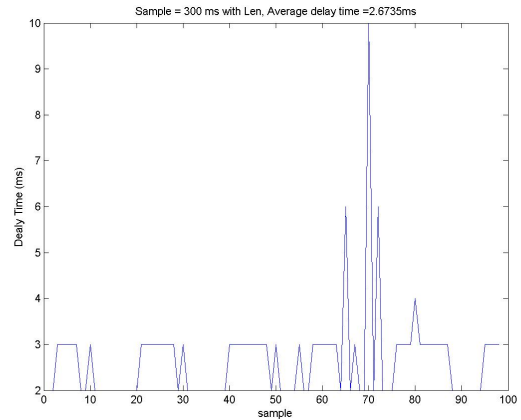


(f)





(g)



(h)

圖 4.7 Ethernet + CAN 之延遲時間量測 (a)  $T=1ms$  (b)  $T=2ms$  (c)  $T=3ms$  (d)  $T=5ms$  (e)  $T=20ms$  (f)  $T=50ms$  (g)  $T=200ms$  (h)  $T=300ms$

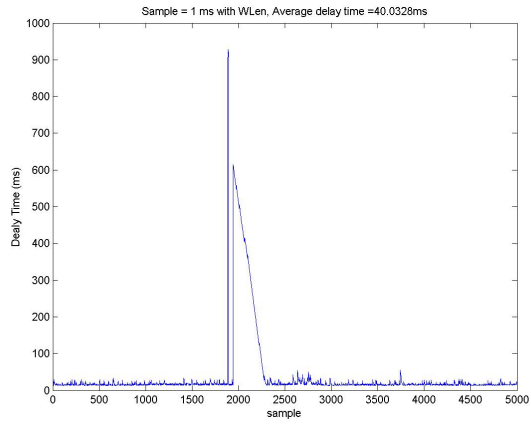
表 4-2 為 IEEE 802.11b 無線網路加上 CAN 網路所量測到的平均延遲時間，由此表格我們會發現，量測出來的結果跟有線網路的現象比起來有很大的差別，在同一傳送時間間隔下平均延遲時間不再近似，且在不同傳送頻率下不再是較低的傳送頻率擁有較短的延遲時間，量測出來的延遲時間呈現不規則分佈，會造成這種現象的發生是因為使用無線網路的關係，由圖 4.8 可以發現某些時刻，延遲時間會異常增加，造成此問題的原因是無線網路容易受到環境干擾造成封包的遺失，但是我們是使用 TCP/IP 的傳輸協定，此傳輸協定為一連線型的傳輸協定，當有訊息沒有傳到目的地時，系統會自動重傳遺失的封包，在封包遺失到重傳的過程當中會造成延遲時間大量的增加，這些大量增加的延遲時間便會對平均延遲時間造成影響，但是這些大量增加的延遲時間對所有傳輸來說只佔少部份，所以我們在估測平均延遲時間時，可以忽略掉這些大量增加的延遲時間來計算平均延遲時間。

表 4-2 802.11 + CAN 的平均延遲時間(處理前) (單位:ms)

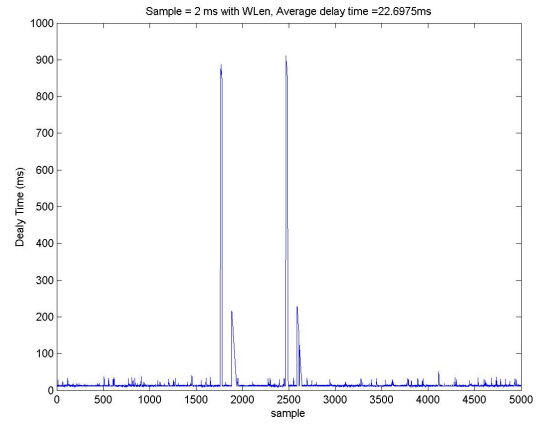
實驗群組 No. 傳送時間間隔 T	1	2	3	4	5
300 ms	27.8889	37.9697	39.41414	6.2222	6.40404
200 ms	34.40367	35.76313	45.828	34.1893	52.53128
50 ms	7.3050	7.3533	6.1167	6.0900	6.0483
20 ms	15.9173	20.1453	7.2667	14.9700	7.1947
5 ms	28.93519	29.58452	24.93579	25.97939	27.5125
3 ms	22.06001	20.93599	21.19504	20.95439	23.4943
2 ms	19.0274	13.5275	19.78836	22.73075	24.60612
1 ms	40.06301	17.61272	17.67153	16.95779	17.27065



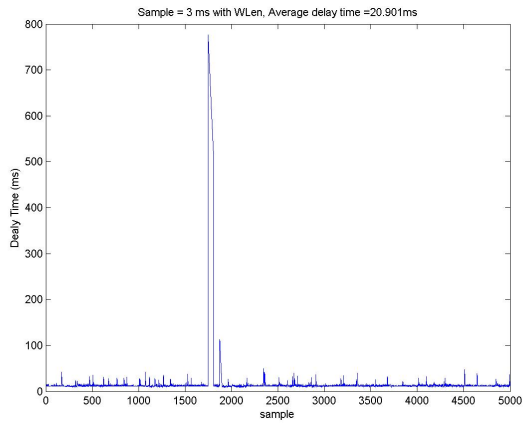




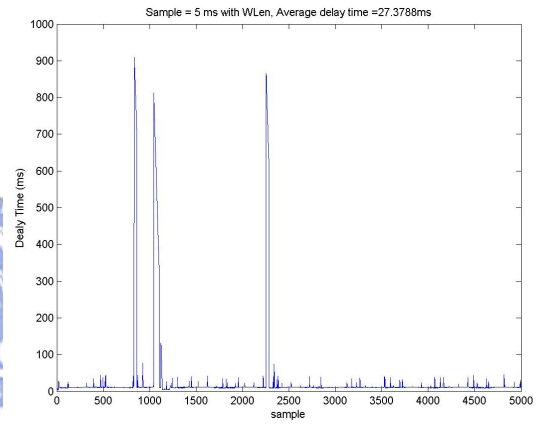
(a)



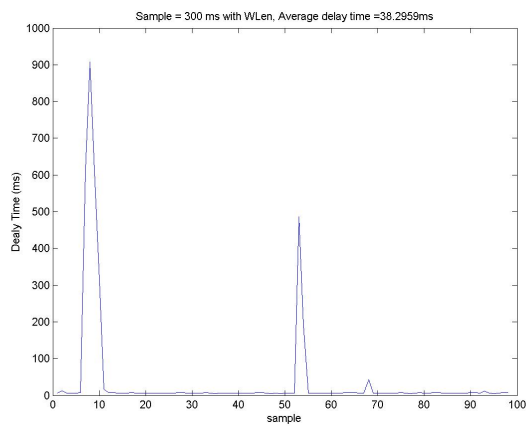
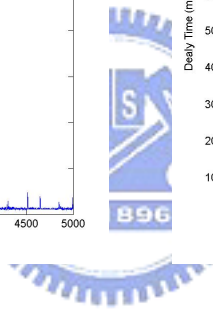
(b)



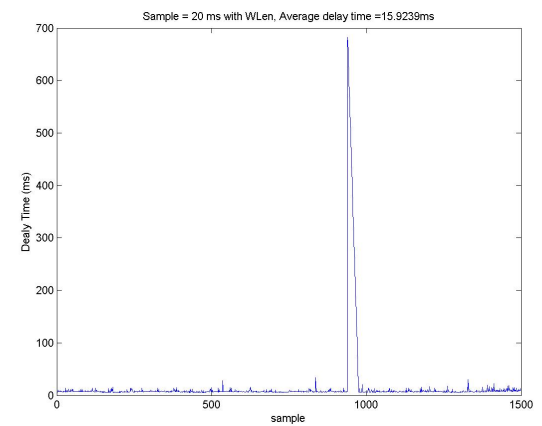
(c)



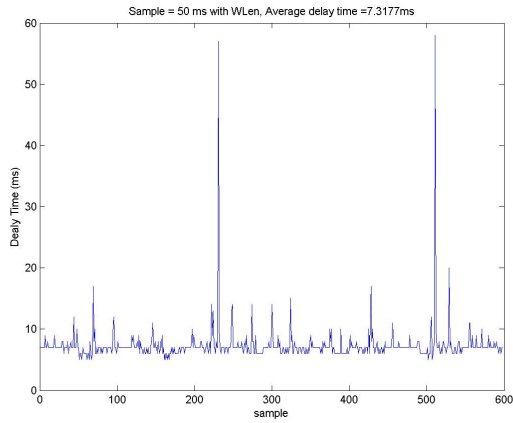
(d)



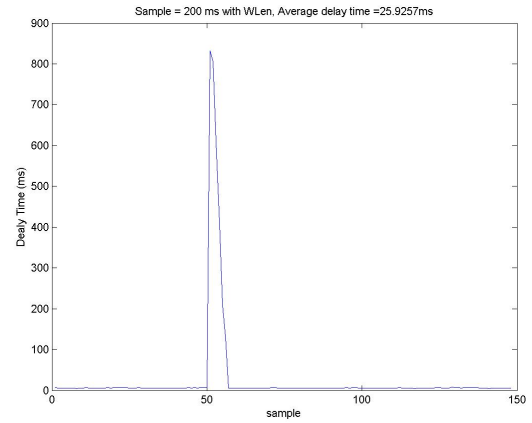
(e)



(f)



(g)



(h)

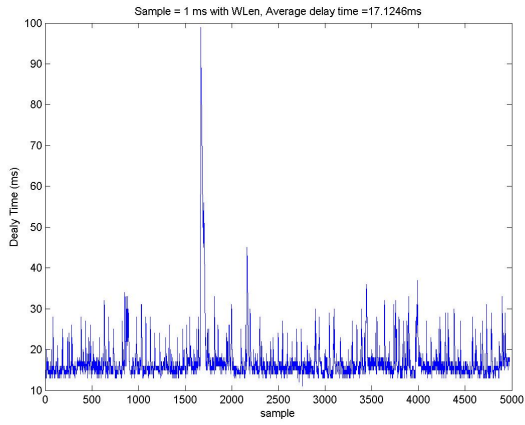
圖 4.8 802.11 + CAN 之延遲時間量測 (處理前) (a)  $T = 1ms$  (b)  $T = 2ms$   
 (c)  $T = 3ms$  (d)  $T = 5ms$  (e)  $T = 20ms$  (f)  $T = 50ms$  (g)  $T = 200ms$  (h)  $T = 300ms$

表 4-2 是將表 4-3 各平均延遲時間對應的資料中延遲時間大於  $100ms$  的資料去除後所求得平均值，在處理過後的資料我們可以發現，現在資料跟使用有線網路所量測到的延遲時間情況差不多，在傳送頻率較低的時候，延遲時間也較小約在  $6\sim 8ms$  左右，傳送頻率高時，延遲時間也跟著上升。若由同頻率來看，在同樣頻率情況下，有差不多的平均延遲時間，藉由將過大的延遲時間替除掉，來還原原本網路時間延遲的狀況，有無此處理，也會對我們後面的 Smith predictor 帶來影響，此方法使用時機是當過大的延遲時間佔正常延遲時間的一小部份時才可以使用，圖 4.9 為處理過後的延遲時間分佈圖。

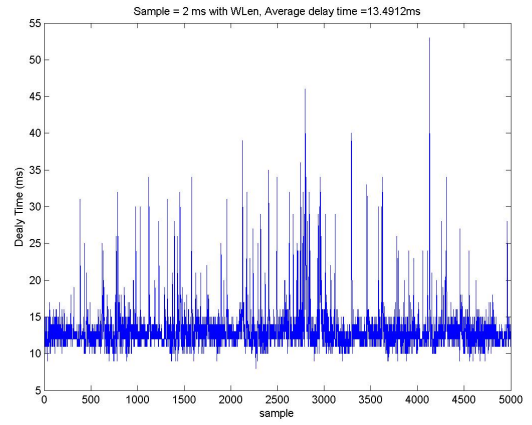
表 4-3 802.11 + CAN 的平均延遲時間(處理後) (單位: *ms*)

實驗群組 No. 傳送時間間隔 T	1	2	3	4	5
300 <i>ms</i>	6.1667	6.8587	7.0879	6.2143	6.482
200 <i>ms</i>	6.0362	5.9225	6.0154	5.9489	5.9262
50 <i>ms</i>	7.3050	7.3533	6.1167	6.0900	6.0483
20 <i>ms</i>	7.6044	7.5303	7.261	7.6501	7.1947
5 <i>ms</i>	11.2694	10.9093	10.6374	10.8904	11.3131
3 <i>ms</i>	13.2678	13.135	13.3383	13.2172	13.2845
2 <i>ms</i>	13.7171	13.4912	13.8011	13.901	13.8893
1 <i>ms</i>	17.6999	17.1248	17.3583	16.921	17.235

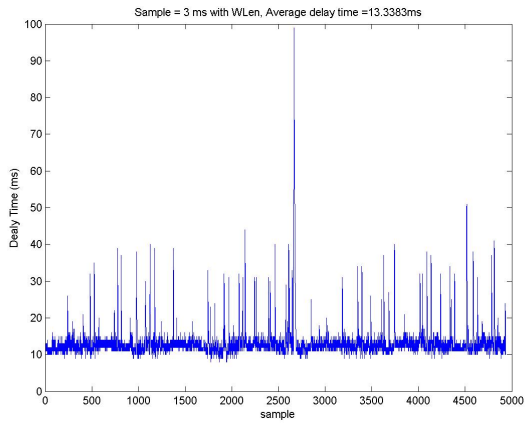




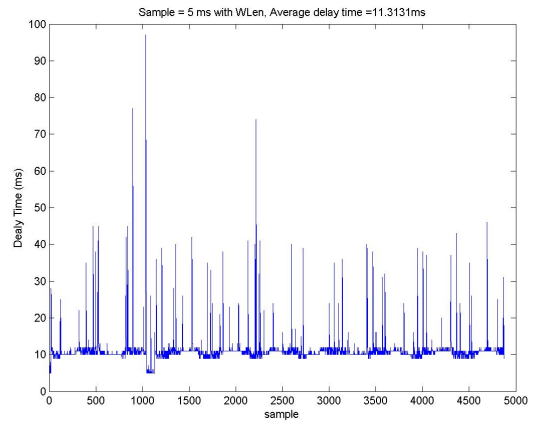
(a)



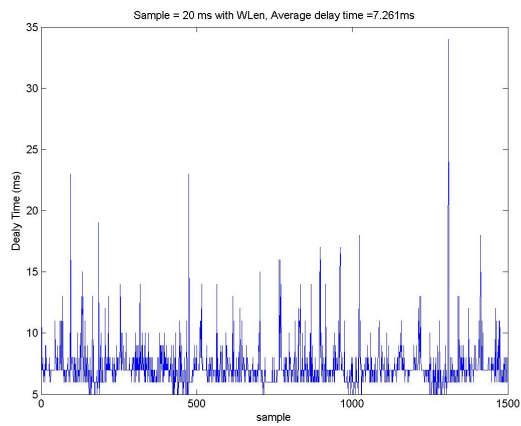
(b)



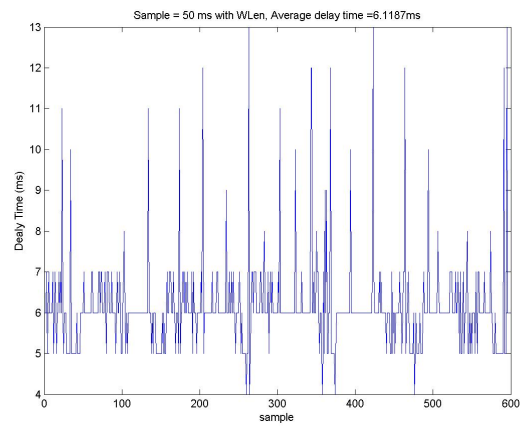
(c)



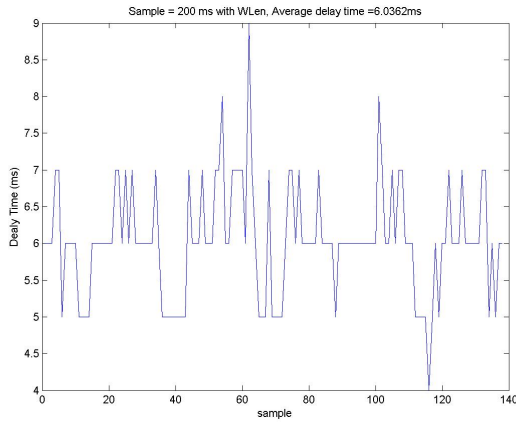
(d)



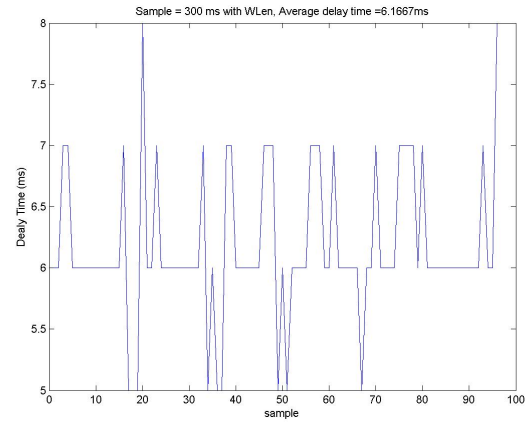
(e)



(f)



(g)



(h)

圖 4.9 802.11 + CAN 之延遲時間量測(處理後) (a)  $T = 1ms$  (b)  $T = 2ms$   
(c)  $T = 3ms$  (d)  $T = 5ms$  (e)  $T = 20ms$  (f)  $T = 50ms$  (g)  $T = 200ms$  (h)  $T = 300ms$

#### 4-3-4 固定延遲時間的產生及量測

由前一節我們可以知道我們建立的網路控制系統的延遲時間，不管是使用有線網路或是無線網路，所量測的平均延遲時間均在  $20ms$  以下，因為有線網路的環境下 Server 和 Client 之間僅透過一個 hub 來連結，在無線網路的環境下是使用點對點的方式讓兩台電腦互相連結，除非傳輸量超過網路負載，否則很難取得較大的延遲時間，但是若故意讓傳輸量高過網路負載，其延遲時間會一直增加並無法取得一個固定的平均延遲時間如圖 4.10，此情況嚴重時會讓整個

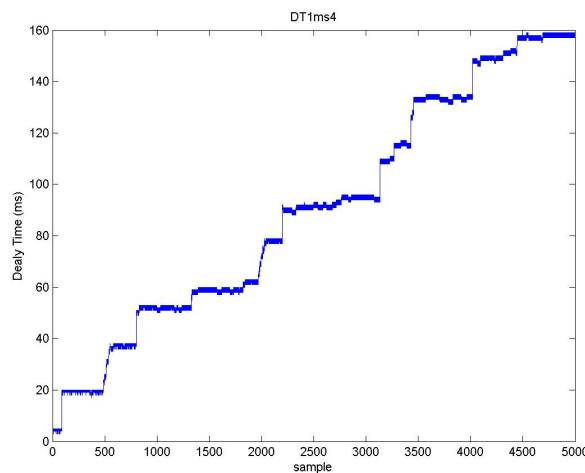


圖 4.10 超過網路覆載的延遲時間

系統故障停擺，例如 CAN 網路具有錯誤處理機制，當傳輸超過覆載，訊息一直發送不出去時，錯誤計數器超過一定量時，會自動將該節點與 CAN 網路隔離。

若我們想分析較長的時間延遲會對我們系統有什麼影響，或想測試我們設計的控制器，在長延遲時間情況下的效能，一個方法是直接將 Client 端拿到較遠的地方，中間透過幾台路由器和 Server 或是跨過不同的網域區段才有可能取得較長的時間延遲，但是這樣實驗非常不方便。另外一個方式我們可以在命令接收端或是回授端建立一個 Buffer 如圖 4.11 所示，當有回授訊號要傳回 Client 時，回授訊號會先進入 Buffer，然後每隔一次取樣週期位移一個位置，直到離開 Buffer 之後 Client 才會取得回授訊號，如此每筆回授訊號依序進入 Buffer 再出來，每筆訊息的延遲時間會等量的增加，增加後的延遲時間和 Buffer 大小關係如下：

$$T_{d\_new} = (Size_{Buffer} - 1) \cdot T_s + T_d \quad (4-1)$$

其中  $T_{d\_new}$  為延長過後的延遲時間， $Size_{Buffer}$  為 Buffer 大小， $T_s$  為系統取樣週期， $T_d$  為原本網路的延遲時間。圖 4.12 為使用 Ethernet 加 CAN 網路，每隔 2ms 傳送一筆訊息再加上 Buffer 產生出來的延遲時間，左邊為 Buffer Size = 50 的結果，此時網路延遲時間為 104.4584ms，右邊為 Buffer Size = 10 的結果，其網路延遲時間為 204.1869ms，兩個延遲時間結果和用式子(4-4)算出來的結果相近，因此想要獲取長時間的網路延遲，選用此方法即可達成。

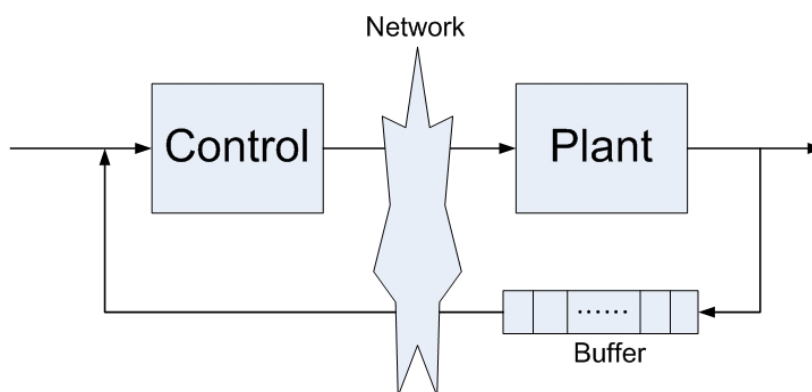
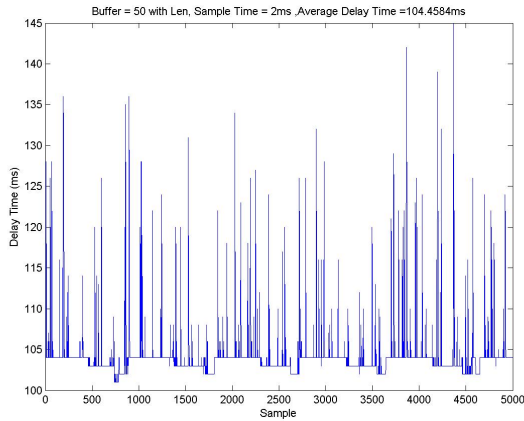
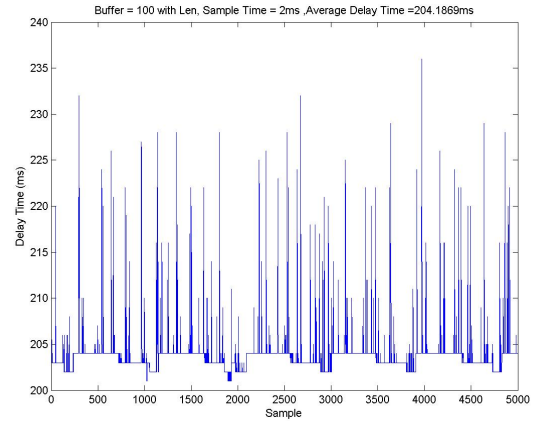


圖 4.11 回授端加 Buffer 示意圖



(a)



(b)

圖 4.12 加 Buffer 後之延遲時間量測(a) Buffer Size = 50, delay time =104.4584ms (b) Buffer Size = 100, delay time = 204.1869 ms

#### 4-3-5 遠距離的網路延遲時間

前面兩個實驗都是在較短距離且干擾較小的環境下做實驗，接下來看看若 Client 和 Server 的距離拉遠，且中間可能會經過其他主機 Server 或是跨網域的時間延遲情況。此實驗分別由高雄、屏東、交大男七舍連線到交大工五館的 LAB816 實驗室來量測其延遲時間，其中高雄部份有兩個量測地點且有不同的網路系統，這些實驗都是經過有線網路來傳輸，表 4-4 為在不同傳輸間隔時間量測實驗結果：

表 4-4 遠距離延遲時間 (單位：ms)

地點 傳送時間間隔 T	高雄 1	高雄 2	屏東	交大七舍
2ms	281.4133	<b>5123.573</b>	<b>12731.93</b>	6.481896
5ms	175.3095	302.0402	373.1072	7.289458
10ms	153.6207	282.9744	340.3327	2.39748
15ms	148.0080	220.7217	270.7242	2.334467
20ms	142.6101	199.0654	249.5145	2.603921

由表 4-4 很明顯看出訊息的傳送時間間隔對網路延遲時間有很大的影響，傳送間隔越長延遲時間越短，相反地傳送時間越短延遲時間越長，且可能超過網路負載，造成延遲時間急遽增加如圖 4.13。

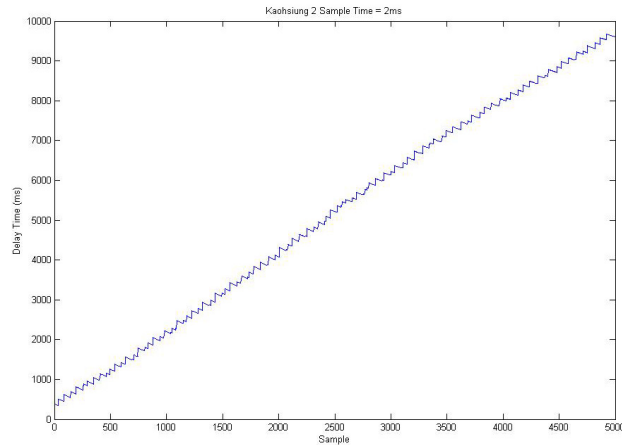
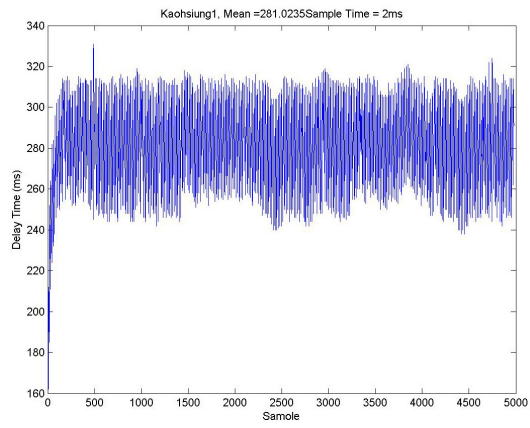


圖 4.13 高雄 2 傳輸間隔 2ms 的不正常延遲時間

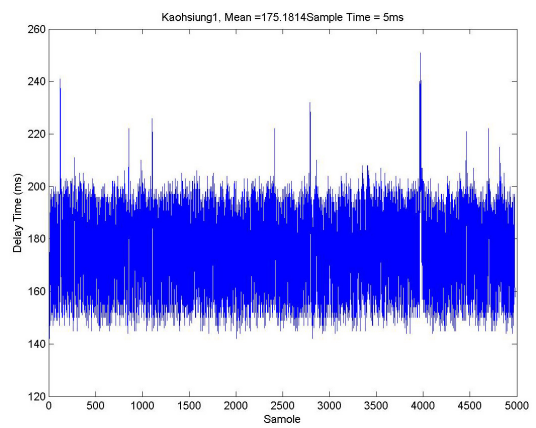
影響長距離的延遲時間，有許多因素，並不是說距離長延遲時間就一定跟著變長，在傳輸過程經過多少路由器或多少 Server 還有末端使用的網路速度，這些都會對延遲時間產生影響，像由交大七舍連到實驗室，所得到的延遲時間，其實跟在實驗室量測到的延遲時間是差不多的，這是因為交大校內都是由 100Mbps 的乙太網路所連結，網路傳輸速度跟在實驗室差不多，所以對延遲時間影響不大，因此單就距離來判斷延遲時間是不準確的，距離跟延遲時間沒有絕對關係，還要多方考量其他因素才行。

雖然在長距離的傳輸下，只要不超過網路負荷量，在同樣的網路和同樣的時間間隔傳輸下，其平均的延遲時間是差不多的，但是看時間分佈圖會發現，其變異量可能會變大許多如圖 4.14 所示，圖 4.14 為高雄 1 在不同傳輸間隔所得到的時間延遲分佈圖。至於變異量變大對使用 Smith predictor 有什麼影響，將在後面章節再另外討論。

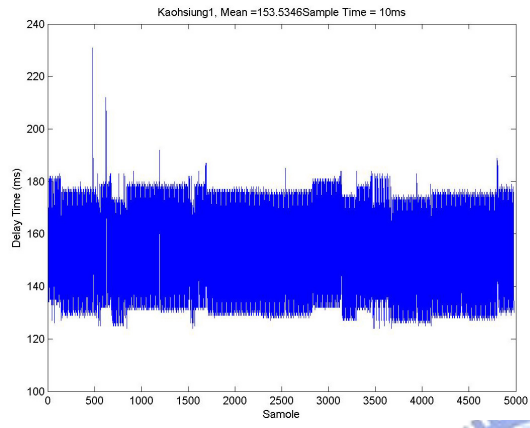




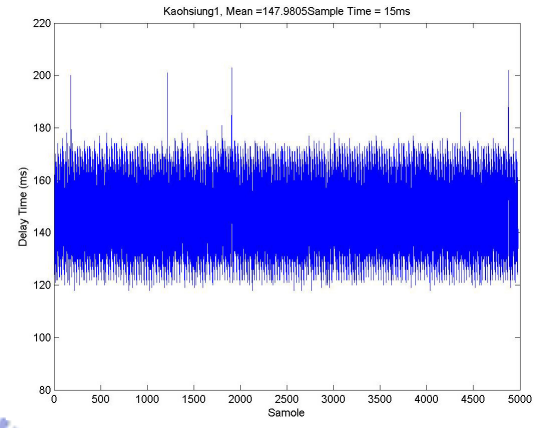
(a)



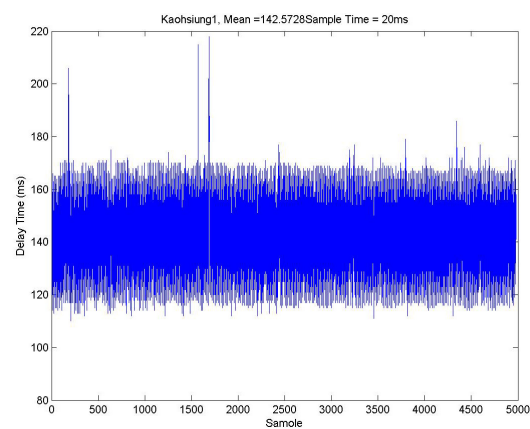
(b)



(c)



(d)



(e)

圖 4.14 高雄 1 之延遲時間量測，傳送間隔為(a)2ms (b)5ms (c)10ms (d)15ms (e) 20ms

## 第五章 網路遠端控制系統設計

### 5-1 Smith predictor

圖 5.1 為網路控制系統的控制方塊圖， $t_d$  為圖 4.1 中命令延遲時間和回授延遲時間的總和。Smith 提出一個控制架構來補償一個具有時間延遲的系統(圖 5.1)，利用此架構來消除時間延遲對系統的不良影響，此架構如圖 5.2 所示，其中  $G_c(s)$  為控制器， $G_p(s)$  為不具時間延遲的系統轉移函數， $\hat{G}_p(s)$  為估測得到的

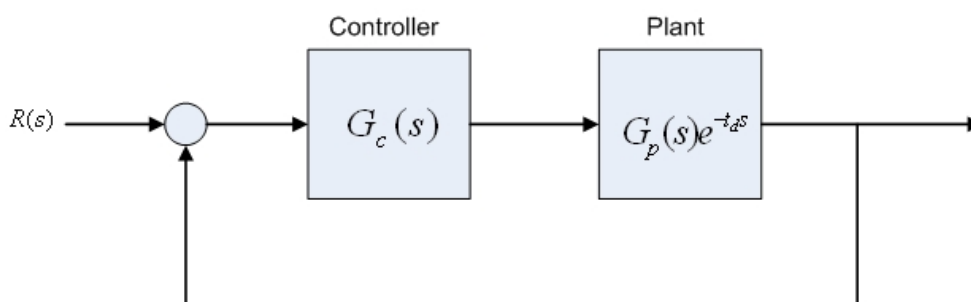


圖 5.1 網路控制系統的控制方塊圖

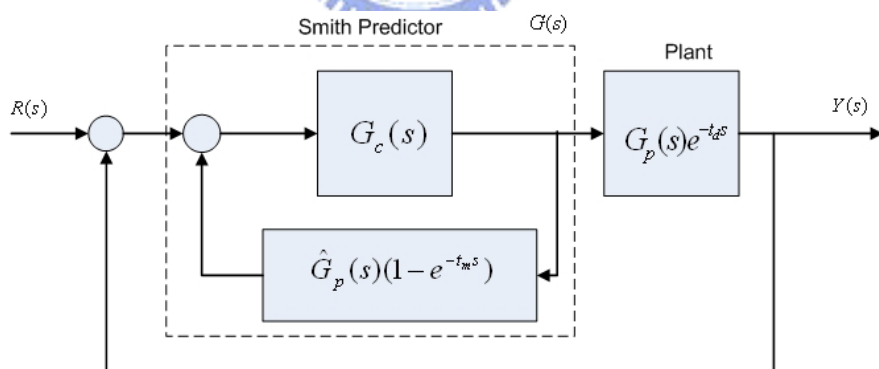


圖 5.2 Smith predictor 方塊圖

不具時間延遲的系統轉移函數， $t_p$  是系統延遲的時間， $t_m$  是估測所得到的系統延遲時間，由輸入  $R(s)$  到  $Y(s)$  的轉移函數，可寫成下面的(4-1)式：

$$\frac{Y(s)}{R(s)} = \frac{G_p(s)G_c(s)e^{-t_p s}}{1 + \hat{G}_p(s)G_c(s) + G_p(s)G_c(s)e^{-t_p s} - \hat{G}_p(s)G_c(s)e^{-t_m s}} \quad (5-1)$$

圖 5.2 虛線部份  $G(s)$  為 Smith predictor，其轉移函式如下：

$$G(s) = \frac{G_c(s)}{1 + (1 - e^{-t_m s})\hat{G}_p(s)G_c(s)} \quad (5-2)$$

當  $\hat{G}_p(s) = G_p(s)$ ， $t_m = t_d$  的時候，(4-1)式變成(4-3)式：

$$\frac{Y(s)}{R(s)} = \frac{G_p(s)G_c(s)}{1 + G_p(s)G_c(s)} e^{-t_d s} \quad (5-3)$$

由(4-3)式我們可以發現，原本因為加上時間延遲的複雜轉移函式，變成單純的兩個部份，一部分是原本沒有受到時間延遲影響的系統轉移函式，另一部份只是單純的時間延遲項，將(4-3)式化成方塊圖如圖 5.3 所示，原本複雜的系統等效成未受到時間延遲的系統加上一個時間延遲項。由此可知，搭配 Smith predictor 我們可以先在沒有時間延遲的系統下面設計好控制器，然後再套用在有時間延遲的系統上面，控制效果理論上可以跟沒有時間延遲的系統一樣，和原本系統唯一的差別就是要經過  $T_d$  之後才會反應。

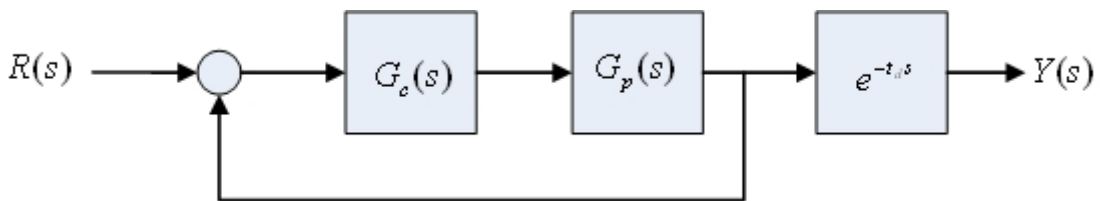


圖 5.3 加上 Smith predictor 的等效方塊圖

## 5-2 實驗系統架構

圖 5.4 為這次實驗的系統架構圖，此架構由三個部份所組成，第一個部份為 Ti 的 F2812 DSP 和 Panasonic AC 伺服馬達所組成，馬達工作模式調整為速度模式，第二部份為一台電腦和 USB CAN 所組成的 Server 端，Server 端是連結 CAN 網路和乙太網路或是無線網路的溝通橋樑，具有閘道器(gateway)的功用，並且具有簡單的管理功能，第三部份為負責網路控制的電腦。DSP 跟 Server 端的 USBCAN 之間的聯繫是透過 CAN 來連結，Server 端和負責遠端控制的 Client 端中間透過乙太網路或是無線網路來連結，資料傳輸的協定則是使用 TCP/IP。把圖 5.4 和圖 4.1 拿來做對應，圖 5.4 中的 Client 相當是圖 4.1 裡面的 Remote 部份，控制器採用 PI 控制器，DSP 加上伺服馬達相當於 Plant，剩下的 Server 和 Ethernet 和 CAN 都屬於 Network 部份，整個控制迴路為位置迴路控制。

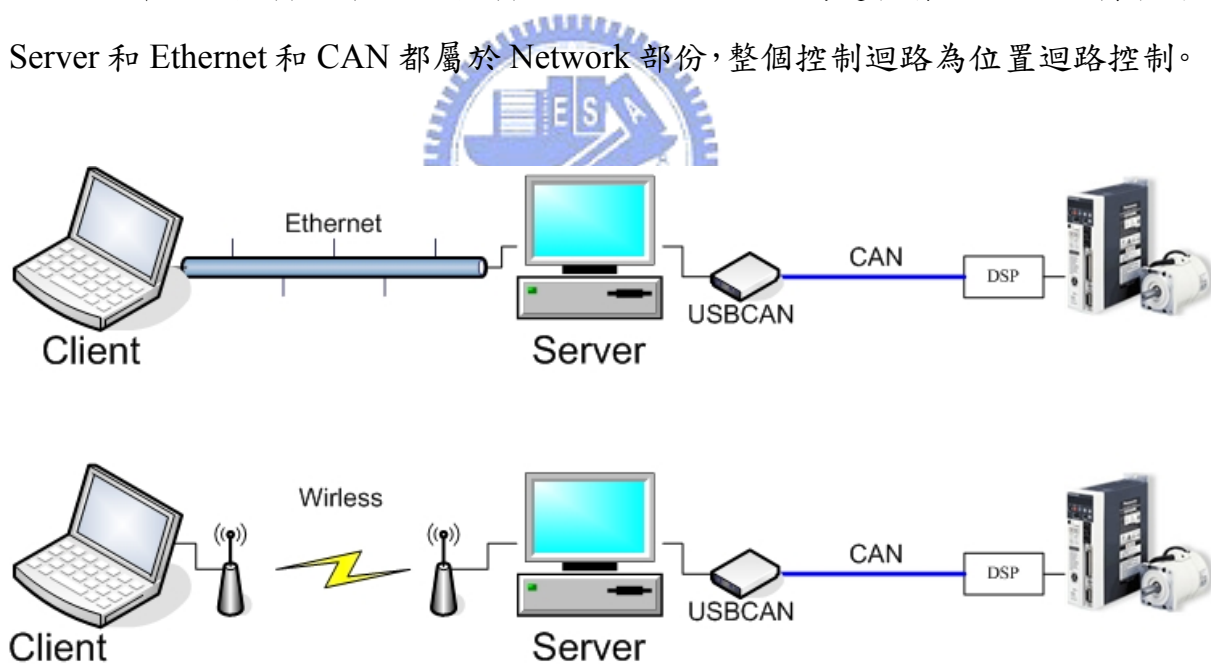


圖 5.4 實驗系統架構圖

由於本實驗系統是透過網路來控制，Client 端和 DSP 端很難達到 Clock 的同步，假如兩端要採用時間觸發(time-triggered)的方式來做控制將是非常困難的事，因此本實驗的架構 DSP 端採用時間觸發方式，而 Client 則是採用事件觸發(Event-triggered)的方式，DSP 每隔固定的時間會將馬達 Encoder 讀到的資料回

傳給 Client，當 Client 有接收到 DSP 傳過來的回授訊號才經由控制器計算出命令回傳給 DSP。

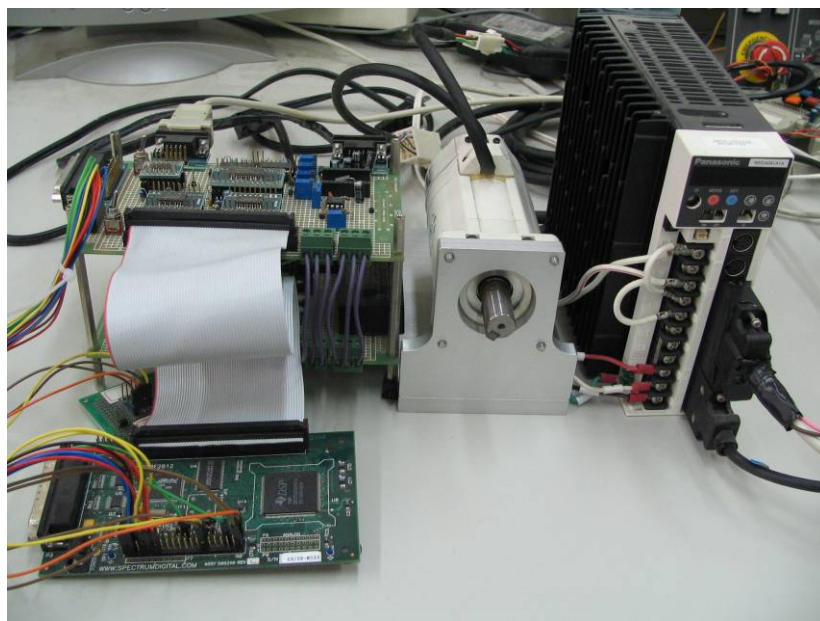


圖 5.5 Ti F2812 ezDSP DSK 板+自製週邊電路+Panasonic AC 伺服馬達



圖 5.6 本次實驗 Server+USBCAN

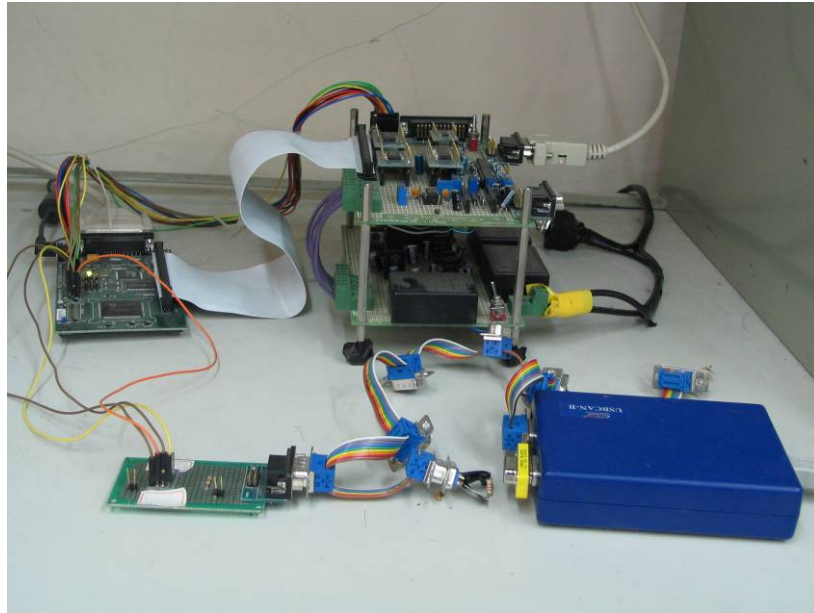


圖 5.7 DSP 和 Server 之間透過 CAN 來做連接

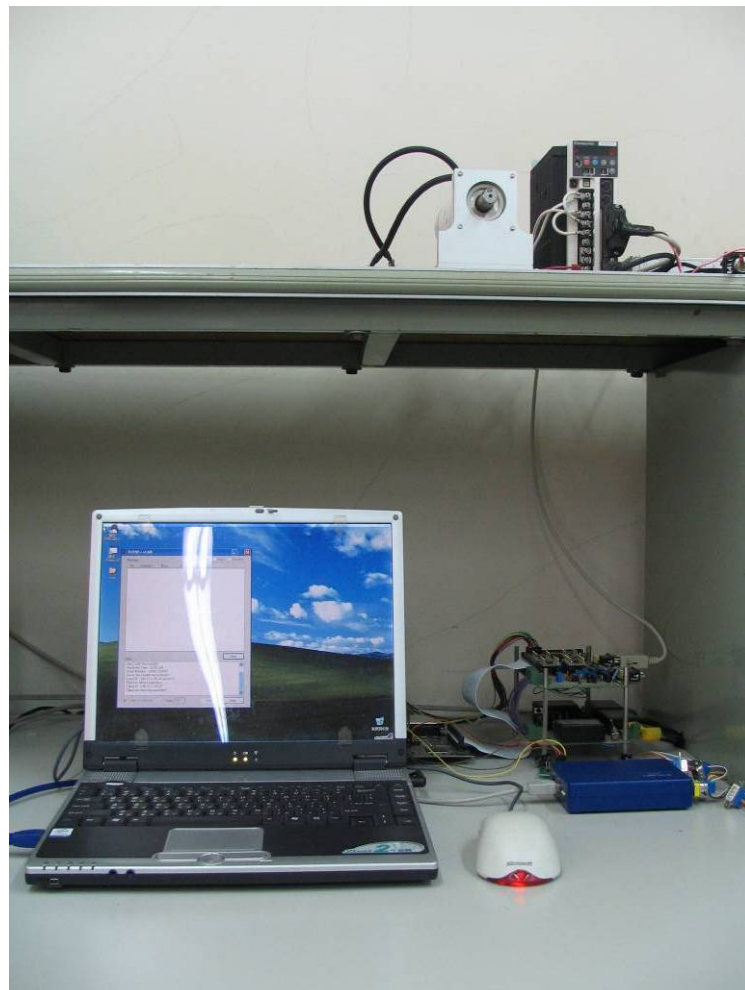


圖 5.8 Server 和馬達系統

## 5-3 TCP/IP 與 CAN 封包轉換

### 5-3-1 Server 架構介紹

由於 DSP 上面只有 CAN 介面，而遠端控制是使用乙太網路或是無線網路透過 TCP/IP 通訊協定來傳遞資料，兩種不同協定的網路無法直接做溝通，所以我們中間需要透過閘道器來轉換兩種不同型式的封包，在上一節提到的 Server 就是扮演這種角色。

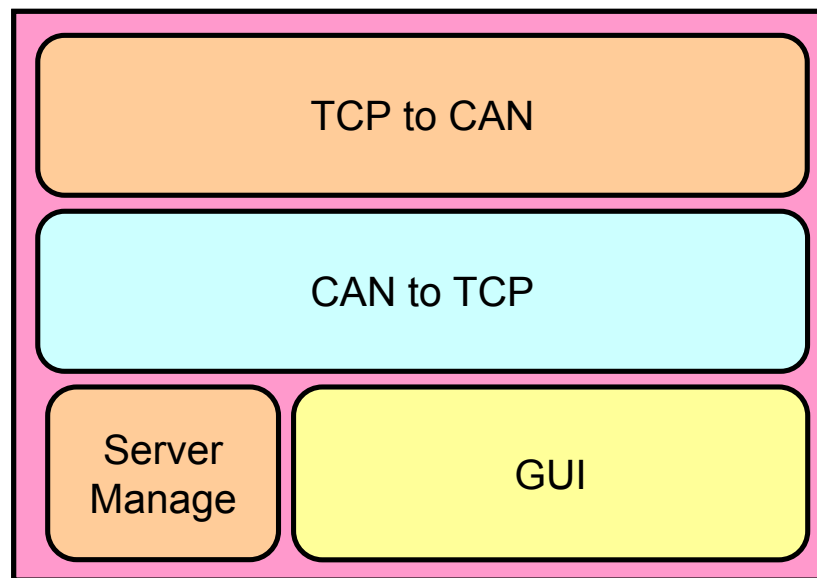


圖 5.9 Server 架構圖

圖 5.9 為 Server 的架構圖，Server 主要分成四個部份，第一部份是由 CAN 網路接受 CAN 封包，然後將該封包轉成 TCP/IP 格式的封包透過 Ethernet 或是無線網路，將封包送到遠端的控制端，第二部份為由 Ethernet 或無線網路接收到 TCP/IP 封包，轉成 CAN 的封包然後經由 CAN 網路將資料傳給 DSP。第三部份為 Server 管理程式，Server 一次僅允許一台 Client 連結到 Server，以防止在做網路控制時，還有其他 Client 端進來干擾，嚴重將使系統不穩定，當已經有一台 Client 已經建立好連線，此時若有其他 Client 嘗試連線，Server 會拒絕該 Client 的連線並且給予警告的訊息。第四部份則為使用者介面。

### 5-3-2 封包轉換方式及過程

當有資料要從 DSP 傳送到 Client 的控制端，首先將訊息種類(Type)及傳送資料(Data)依照第三章所提的訊息 Frame 設定好，設定好訊息 Frame 之後將他放到 CAN 封包的 Data 部份透過 CAN 網路傳到 Server，此步驟如圖 5.10 的(1)。當 Server 接收到 CAN 送過來的封包之後，將整個 CAN 封包放到 TCP 封包的 Data 部份，再利用 Ethernet 或無線網路傳送到 Client 端，此步驟如圖 5.10 的(2)。Client 收到由 Ethernet 或是無線網路送過來的封包後，先從 TCP 封包 Data 部份取出 CAN 的封包，再由 CAN 封包 Data 部份取出使用者自己定義的資料 Frame，最後再分析資料 Frame 即可完成 DSP 傳送到 Client 的工作，如圖 5.10 的(3)。照著圖 5.10 中(1)→(2)→(3)的順序即可將訊息完整的傳送到 Client 控制端。

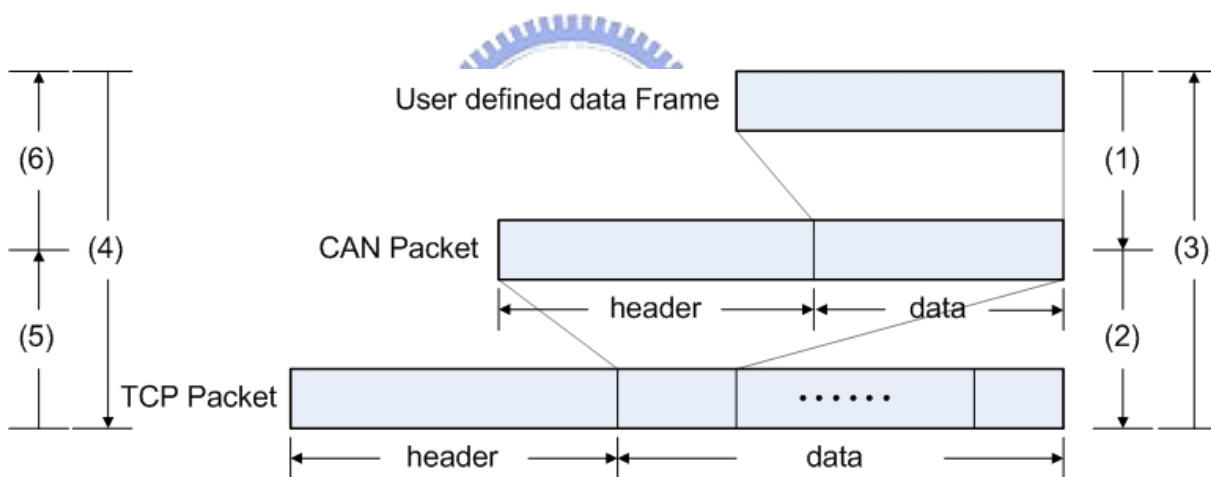


圖 5.10 封包轉換圖

若要將資料由 Client 控制端傳送到 DSP，先將要傳送的訊息種類(Type)和傳送資料(Data) 依照第三章所提的訊息 Frame 設定好，再把該 Frame 放到 CAN 封包 Data 部份，最後把 CAN 封包整個放到 TCP 封包的 Data 部份，利用 Ethernet 或無線網路將封包傳送出去，如圖 5.10 的(4)。Server 收到封包之後，從 TCP 封包 Data 部份取出 CAN 封包，將 CAN 封包透過 CAN 網路將封包傳送到 DSP，如圖 5.10 的(5)。DSP 收到 CAN 封包後，將訊息 Frame 由 CAN 封包 Data 部份取出，再進行解析，如圖 5.10 的(6)。只要照著圖 5.10 中(4)→(5)→(6)的訊息即



可完成 Client 控制端傳送訊息到 DSP 的動作。

### 5-3-3 程式介面及程式流程圖

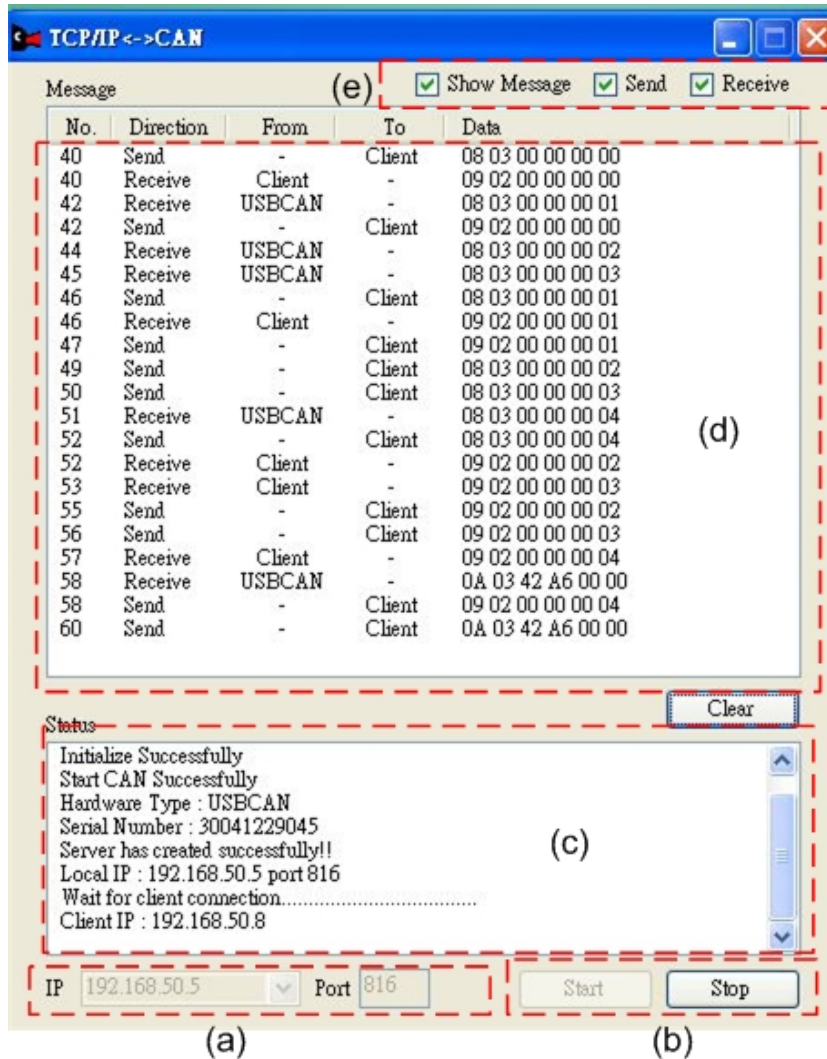


圖 5.11 Server 人機介面

圖 5.11 為 Server 的人機介面，具有將 CAN 和 TCP/IP 兩種不同網路協定做轉換的功能，同時具有連線管理功能和簡單的系統異常偵測功能。在 Ethernet 或無線網路連線發生錯誤的時候，會自動停止程式並且停止 DSP 及馬達運轉，以防止嚴重錯誤的發生。以下為程式個部份的功能介紹：

- (a) 該主機可獲得的 IP，當該主機有多張網卡的時候，會把該主機所有可以得到的 IP 列出來，供使用者選擇欲建立連線的 IP。Port 為建立連線的埠號。
- (b) Start 為建立網路連線，等待使用者跟主機建立連線，並且啟動 USBCAN 的連線。Stop 為停止網路連線，並且關閉 USBCAN 的連線。
- (c) 狀態顯示。
- (d) 目前主機接收或傳出訊息的狀態，需選取(e)中的 Show Message 才會有畫面顯示在上面，此畫面可以提供訊息傳收狀態、傳送目的地目接收來源還有該訊息所帶的資料。此功能是方便使用者監看訊息傳遞還有 debug 而設計，但實際需要大量傳輸的時候，建議不要將 Show Message 勾選起來，因為可能拖累整個系統的性能。
- (e) 選擇是否顯示訊息和選擇顯示訊息的模式。

介紹完人機介面，後面再附上 Server 的程式流程圖，圖 5.12 為 Server 連線管理流程圖，圖 5.13 為 CAN 轉 TCP/IP 的流程圖，圖 5.14 為 TCP/IP 轉 CAN 的流程圖。

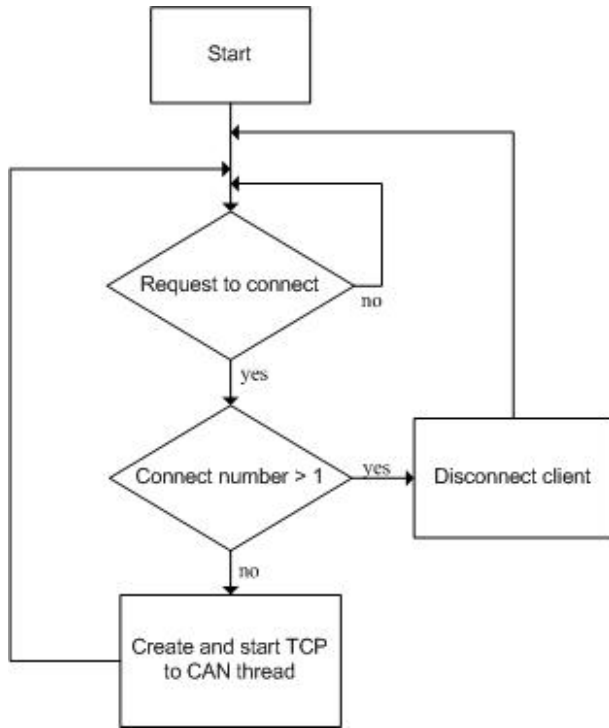


圖 5.12 Server 連線管理流程圖

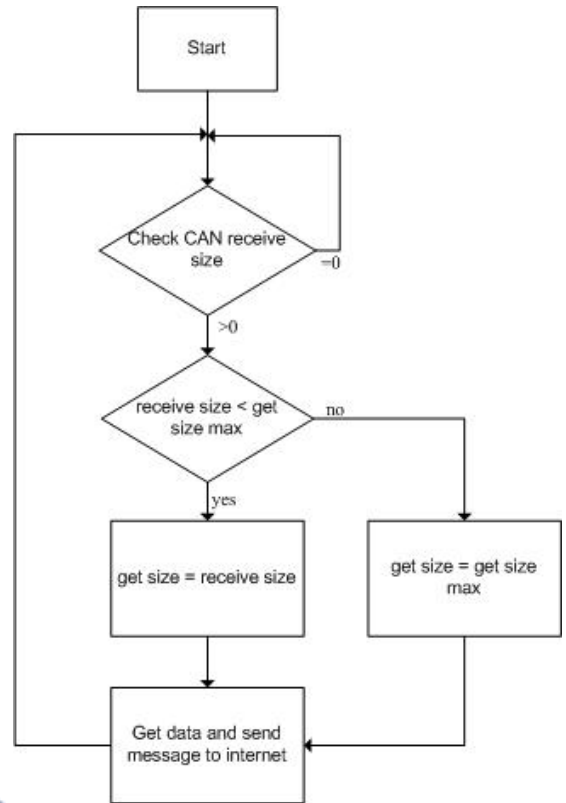


圖 5.13 CAN 轉 TCP/IP 流程圖

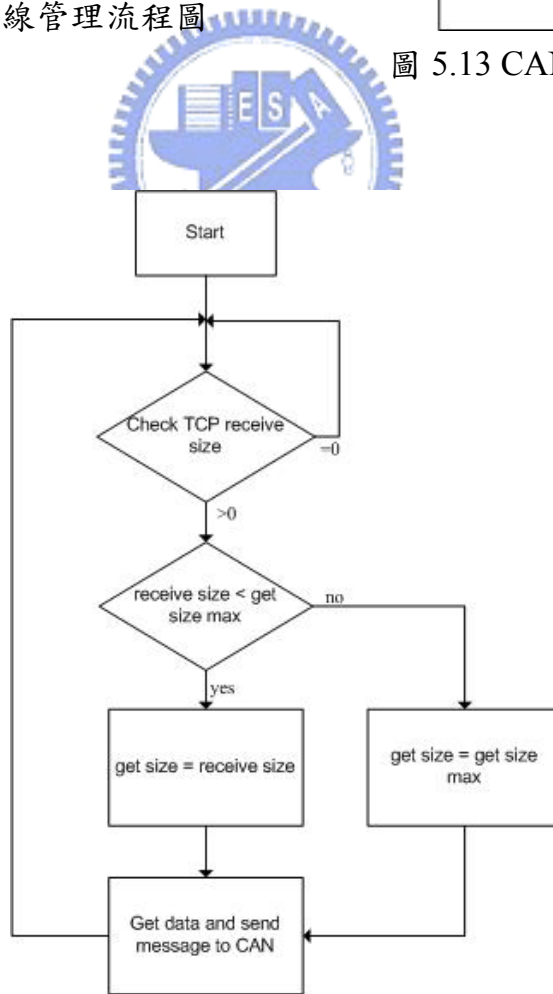


圖 5.14 TCP 轉 CAN 流程圖

## 5-4 系統識別實驗

在設計 Smith predictor 之前，必須先取得系統的識別資料。本實驗系統模型(model)參數的取得，是對系統輸入虛擬隨機二進位訊號(Pseudo Random Binary Sequence, PRBS) 的方式獲取輸出的響應，利用 ARX(Auto-Regresive Exogeneous)模型作為本實驗系統的模型結構，然後對 ARX 模型的參數做最小平方推測以取得系統模型[19,20]。ARX 的數學模式為下

$$A(q^{-1})y(k) = B(q^{-1})u(k - n_k) \quad (5-4)$$

其中  $n_k$  延遲時間， $A(\cdot)$  和  $B(\cdot)$  分別為  $q^{-1}$  的多項式：

$$A(q^{-1}) = 1 + a_1q^{-1} + a_2q^{-2} + \dots + a_{na}q^{-na} \quad (5-5)$$

$$B(q^{-1}) = b_0 + b_1q^{-1} + a_2q^{-2} + \dots + a_{nb}q^{-nb} \quad (5-6)$$

圖 5.15 為系統識別流程圖，圖 5.16 是利用自己開發的程式讓識別出來的系統模型自動產生對應的 C#程式碼以方便後面 Smith predictor GUI 使用。

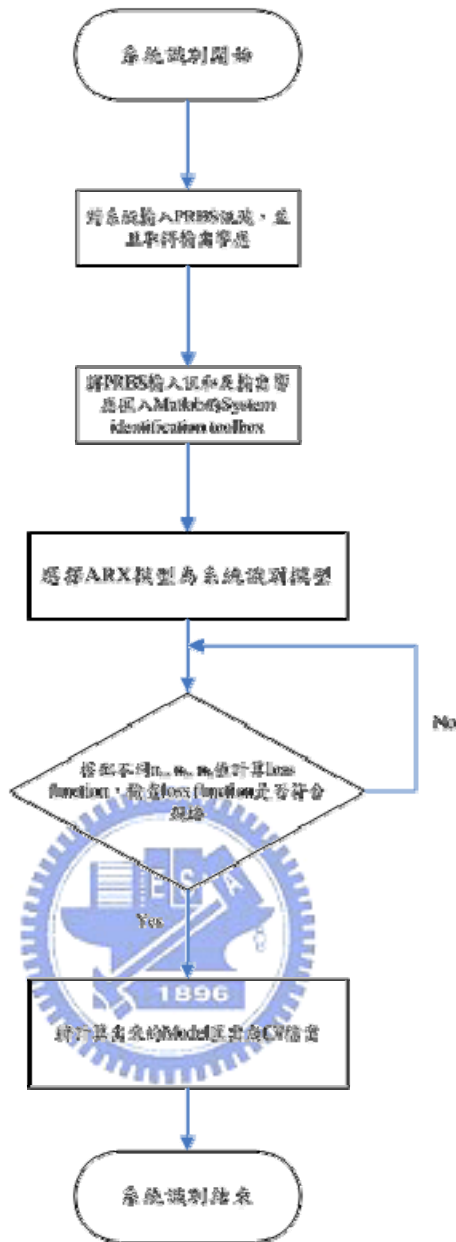


圖 5.15 系統識別流程圖

```

/* ----- File Generation Date :25-Sep-2006 20:51:15 ----- */
using System;

namespace ControlSys
{
    class SysModel
    {
        public static Single[] num = {0.000000F,5.277227F,14.989393F,-8.787528F,0.000000F};
        public static Single[] den = {1.000000F,-1.180290F,0.185197F,-0.135699F,0.130793F};
    }
}
  
```

圖 5.16 自動產生 C# 系統模型程式片段

## 5-4-1 系統識別人機使用介面

圖 5.17 為系統識別程式有兩種模式供選用，一個模式為系統識別模式，另一個模式為線上調整 DSP 上面 PI 控制器的  $K_p$  和  $K_i$ ，以下為程式各部份的說明：

(a) 連結 USBCAN

(b) 開始執行系統識別或是調整 PI 控制器。

(c) 相關參數輸入

$K_p$ ：PI 控制器的  $K_p$  控制器

$K_i$ ：PI 控制器的  $K_i$  控制器

Position：調整 PI 控制器時所輸入 Step 的大小，單位 pulse

Amplitude：系統識別時 PRBS 訊號輸入的大小，單位 Voltage

$T_s$ ：系統識別或調整 PI 控制器時的系統取樣週期，單位  $ms$

(d) 程式執行模式

(e) 訊息顯示介面

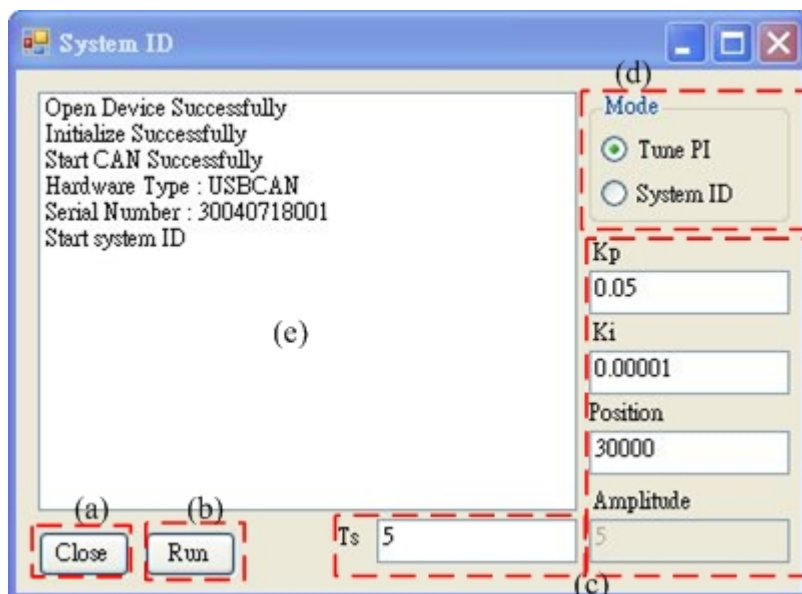


圖 5.17 系統識別人機使用介面

## 5-4-2 實驗結果

本次實驗用的馬達輸入電壓範圍為 $\pm 10V$ ，工作模式為速度模式，馬達轉一圈輸出 10000 個 pulse。系統識別輸入為振幅 5V 的 PRBS 訊號，輸出為馬達位移角度單位為 pulse，取樣時間為 2 ms。圖 5.18 為系統識別的輸入和輸出。

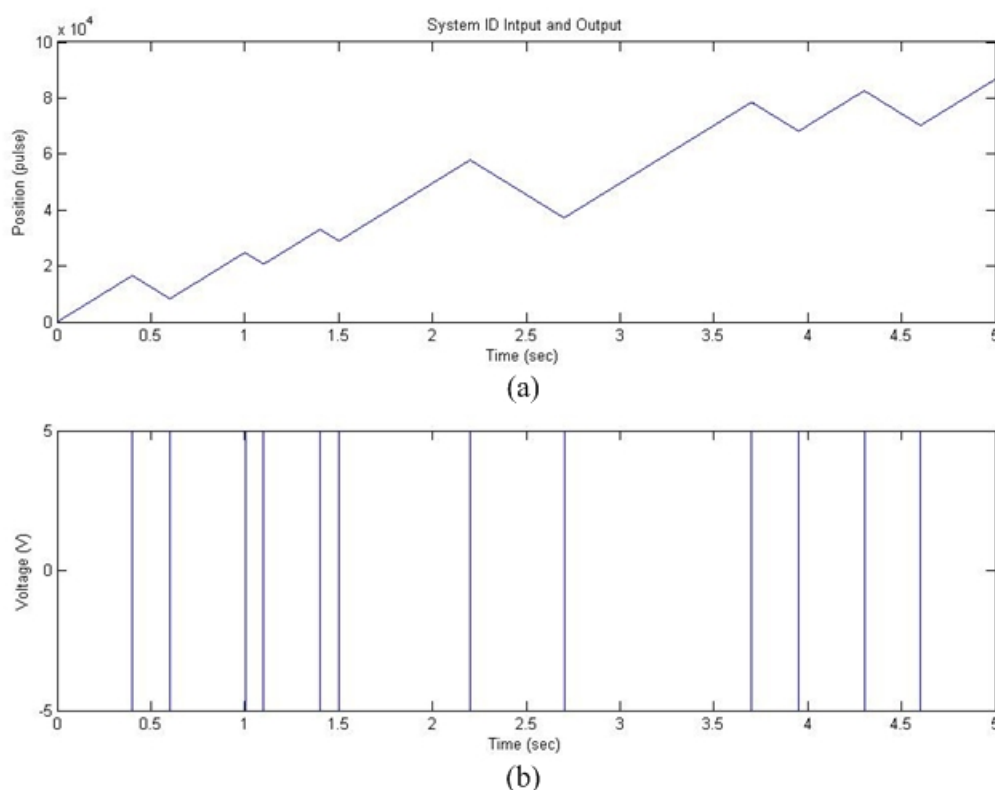


圖 5.18 系統識別的輸入(a)和輸出(b)

取得系統識別的輸入和輸出之後，再利用 Matlab 的 System Identification Toolbox (圖 5.19)，將系統識別的輸入及輸出響應讀入，識別模型選擇參數型的 ARX 模型，na, nb, nk 的階數範圍階設為 1~10 後執行程式，當執行完後會跳出圖 5.20，圖 5.20 為估測不同階數模型的誤差圖，我們選擇虛線圈起來的部來當作是我們馬達系統的模型，雖然階數較高的時候有較少的誤差，但此時誤差已經非常小了，沒有必要選擇更高的階數來增加計算上面的複雜度。

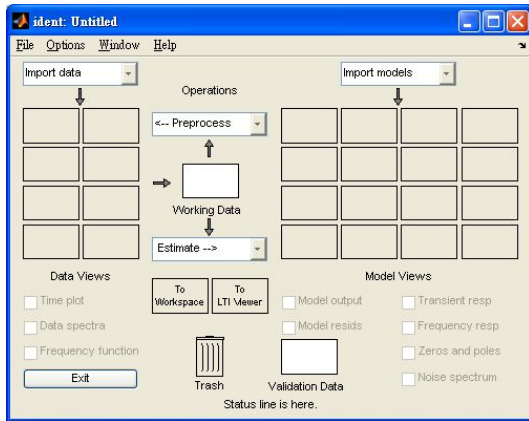


圖 5.19 System Identification Toolbox GUI

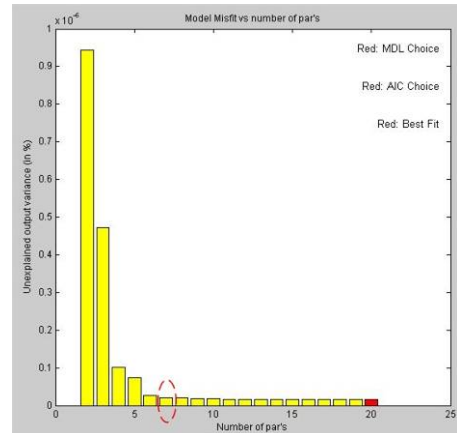


圖 5.20 不同階數估測模型的誤差

我們估測出來的系統模型如下：

$$\frac{\theta}{V} = \frac{5.277z^3 + 14.99z^2 - 8.788z}{z^4 - 1.18z^3 + 0.1852z^2 - 0.1357z + 0.1308} \quad (5-7)$$

Sample time : 0.002 ms

Loss function = 0.577863 ms

為了驗證我們估測出來的系統模型是否準確，我們將原本 PRBS 輸入訊號加到估測出來的模型當中所得到的輸出和原本實際系統輸出做比較，由圖 12 和圖 5.22 可以看出估測模型的輸出和實際系統輸出非常吻合，根據計算吻合程度達 99.96%。

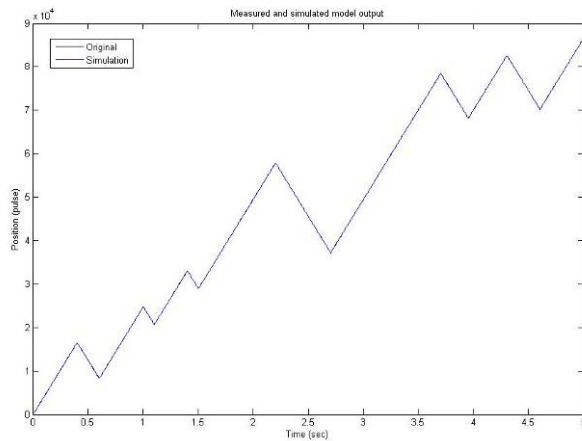


圖 5.21 估測模型和實際系統比較圖

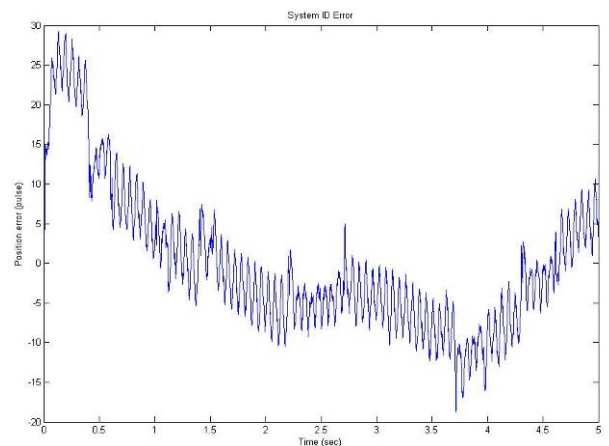


圖 5.22 系統判別位置誤差



## 5-5 DSP 端程式

本實驗的 DSP 端程式分為五大部份分別為(a)系統識別，(b)調整 PI 控制器，(c)測試延遲時間，(d)Smith predictor 接收端，(e)訊息管理程式，以下為 DSP 各部份程式的簡單介紹：

- (a) 系統識別：在訊息管理程式接到做系統識別的訊息之後，等到人機介面將取樣頻率，PRBS 振幅設定好之後進入此程式，開始對馬達系統送出 PRBS 訊號，送完 PRBS 訊號之後，回傳完成系統識別的訊號後即完成 DSP 系統識別程式。
- (b) 調整 PI 控制器：當訊息管理程式接收到調整 PI 控制器命令之後，等到人機介面設定好 PI 控制器的  $K_p$ ,  $K_i$  值還有輸入步階響應的大小後，開始對馬達做 PI 控制，等 PC 端送停止命令，DSP 端停止控制馬達，並且對馬達輸出 0V 電壓。
- (c) 測試延遲時間：當接收到測試命令時，等到收到傳送次數，儲存間隔，取樣週期和延遲時間 Buffer 大小之後，開始傳送帶有 Index 的訊息，之間的工作過程就像 0 所提，這邊不再重述，當收到最後一筆訊息之後，計算平均網路延遲時間，並將此數據傳回遠端電腦，傳回遠端電腦之後即完成此程式。
- (d) Smith predictor 接收端：當收到啟動 Smith predictor 控制，等待設定好延遲時間 Buffer 和取樣週期之後，即進入此程式，此程式非常簡單，只要把遠端傳來的電壓訊號傳到馬達即可，由於 DSP 端是時間觸發，當取樣週期到的時候，此時若遠端的命令還沒傳到，或是訊息遺失，我們只採取最簡單的措施，就是延續前一個取樣週期的命令當作現在的命令。當遠端傳來停止的命令時，則結束整個程式，並且輸出 0V 給馬達。
- (e) 訊息管理程式：負責解析接收到的訊息，並做出對應動作。

每個部份都有對應的人機介面程式，其對應程式已經在其他小節提過，再此不再贅述。DSP 端程式和人機介面的之間的溝通方式就像第 2 章所提，都是透過事先定義好的事件還有處理方法，只要將想傳遞的訊息照格式封裝起來傳到對應的地方，接受端只要在接到封包之後，照著事先訂好的格式解開封包，即可做出對應的動作，表 5-1 為所有系統事件定義表，其中 No.為該事件的代號，Type 為該事件的名稱，Device 為訊息傳送節點，Data 為該訊息攜帶的資料，Data Size 為攜帶資料的大小。利用此方法，當有新功能加入時，只要增加需要的訊息事件，再寫入對應的程式即可。

表 5-1 網路控制系統系統事件定義表

No.	Type	Device	Data	Data Size (Byte)	說明
1	RUN_SYS	PC	操作模式	2	啟動系統，並且設定操作模式。 1：線上調整 PI 模式 2：系統識別模式 3：量測延遲時間模式 4：Smith predictor 控制模式
2	PAUSH_SYS	PC	無	0	暫停系統
3	STOP_SYS	PC DSP	無	0	停止系統
4	POSITION_DATA	PC	輸入大小 (pulse)	4	線上調整 PI 模式下的輸入命令
5	KP_SET	PC	Kp 值	4	設定系統 PI 控制器的 Kp 值
6	KI_SET	PC	Ki 值	4	設定系統 PI 控制器 Ki 值
7	SET_ID_INPUT_MAX	PC	系統識別輸入最大值(V)	4	設定系統識別 PRBS 訊號最大值

8	DELAY_TIME_SEND	DSP	Index	4	發送帶有 Index 測試延遲時間訊息
9	DELAY_TIME_FEEDBACK	PC	Index	4	接收帶有 Index 測試延遲時間訊息
10	DELAY_TIME_FINISH	DSP	平均延遲時間(ms)	4	量測延遲時間結束訊息，並回傳平均延遲時間
11	CMD	PC	電壓(V)	2	Smith predictor 操作模式下，傳送給 DSP 的電壓命令
12	FEEDBACK	DSP	位置(pulse)	4	Smith predictor 操作模式下，傳回馬達位置
13	SETTING_TS	PC	取樣週期(ms)	2	設定取樣週期
14	SETTING_TEST_TIMES	PC	量測次數	2	設定單次量測延遲時間的次數
15	SETTING_DELAY_TIME	PC	Buffer 大小	2	設定延遲時間 Buffer 大小
16	RECORD_PERIOD	PC	記錄週期	2	在量測延遲時間模式下，記錄訊息的時間間隔

## 5-6 Smith predictor 設計

### 5-6-1 Smith predictor 在數位系統的實現

在 5-1 節中提到的 Smith predictor 是在連續系統下所推導出來的，若我們想在數位系統下使用，我們需將 Smith predictor 做一些調整，圖 5.23 為一個 Smith predictor 方塊圖，首先先將  $\hat{G}_p(s)$  換成我們前面完成的系統辨識的數位模型， $G_c(s)$  換成想要使用的數位控制器， $e^{-t_m s}$  延遲時間項則用  $z^{-k}$  來代替，其中  $k$  為整數， $k$  的決定方式如下：

$$k = \text{ceil}\left(\frac{t_m}{T_s}\right) \quad (5-8)$$

其中  $t_m$  為網路平均延遲時間， $T_s$  為取樣週期， $k$  的決定方式就是將  $t_m$  除以  $T_s$  之後再無條件進位，經過上述動作後，Smith predictor 即可在數位系統下使用。

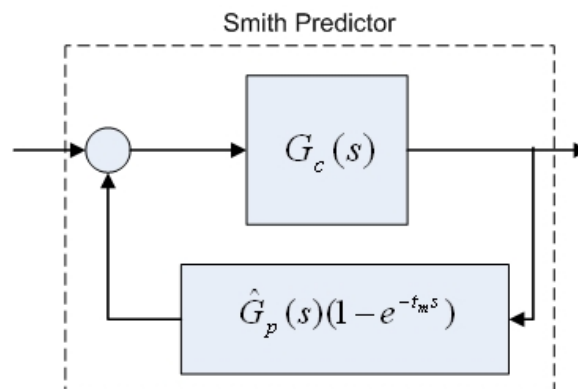


圖 5.23 Smith predictor 架構圖

## 5-6-2 實驗環境

在本次實驗，我們會使用兩組 PI 控制器分別當作我們 Smith predictor 裡面的  $G_c$ ，一組是反應速度較快，另外一組反應速度較慢，如此可以看延遲系統對反應速度不同的影響，反應速度較慢的 PI 控制器 A： $K_p = 0.00053$ ， $K_i = 0.000004$ ，反應速度較快的 PI 控制器 B： $K_p = 0.01$ ， $K_i = 0.0000001$ ，圖 5.24 為兩組 PI 控制器在 2ms 取樣週期，輸入命令為步階大小為 30000 個 pulse 下沒有透過網路直接由 DSP 對馬達控制的響應圖：

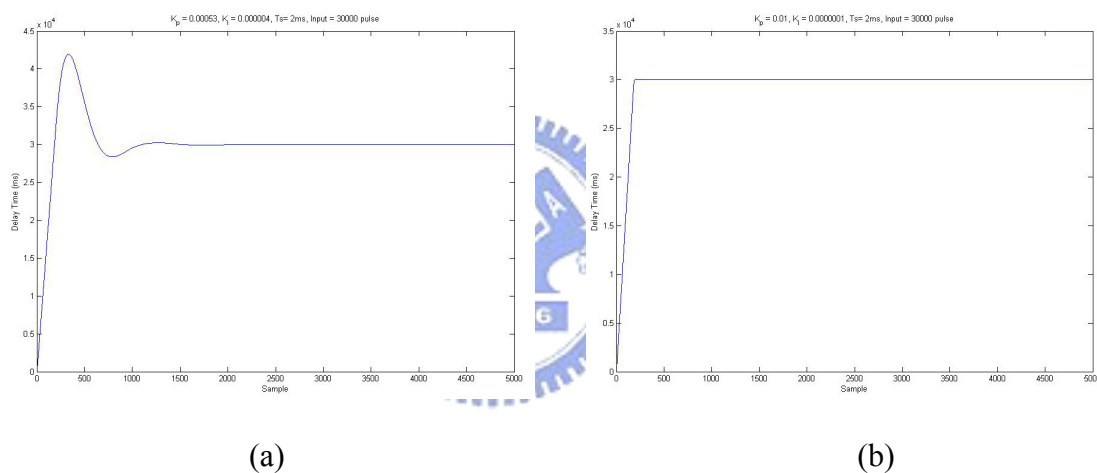


圖 5.24 DSP 對馬達控制的系統響應圖(a) 控制器 A  $K_p = 0.00053$ ，

$K_i = 0.000004$  (b) 控制器 B  $K_p = 0.01$ ， $K_i = 0.0000001$

接下來的實驗分為兩個部份，第一部份為在不同延遲時間下，延遲時間對網路控制系統的影響，第二部份為長距離的網路控制，並且看延遲時間的變異量對 Smith predictor 是否造成影響。每部份實驗都會用有線網路和無線網路來傳輸。

### 5-6-3 Smith predictor 人機介面介紹

圖 5.25 為 Smith predictor 人機介面，此介面是跟延遲時間人機介面共用同一個程式，要操作此程式非常簡單，首先先連線到 Server，再來將相關數值設定好之後先按 Measure 量測延遲時間，假如已經知道延遲時間不需要量測，直接將延遲時間填到適當的位置即可，然後就可以按 Start 開始下命令給馬達，此時 Start 按鈕會變成 Stop 按鈕，只要按下 Stop 即可停止馬達，再按 Save 按鈕可將實驗所得到的數值存起來。以下為程式各部份的解說：

(a) 連線 Server 的 IP 和 Port

(b) 設定補償的延遲時間，可以手動輸入，或是按 measure 量測完後自動將數值填上時間，單位為 *ms*

(c) 儲存資料容量

(d) 步階輸入的大小，單位為 pulse

(e)  $K_p$  和  $K_i$  的設定

(f) 設定取樣週期，單位 *ms*

(g) 設定產生固定延遲時間 Buffer 的大小

(h) 解除跟 Server 連線的按鈕

(i) 啟動馬達的按鈕

(j) 存檔按鈕



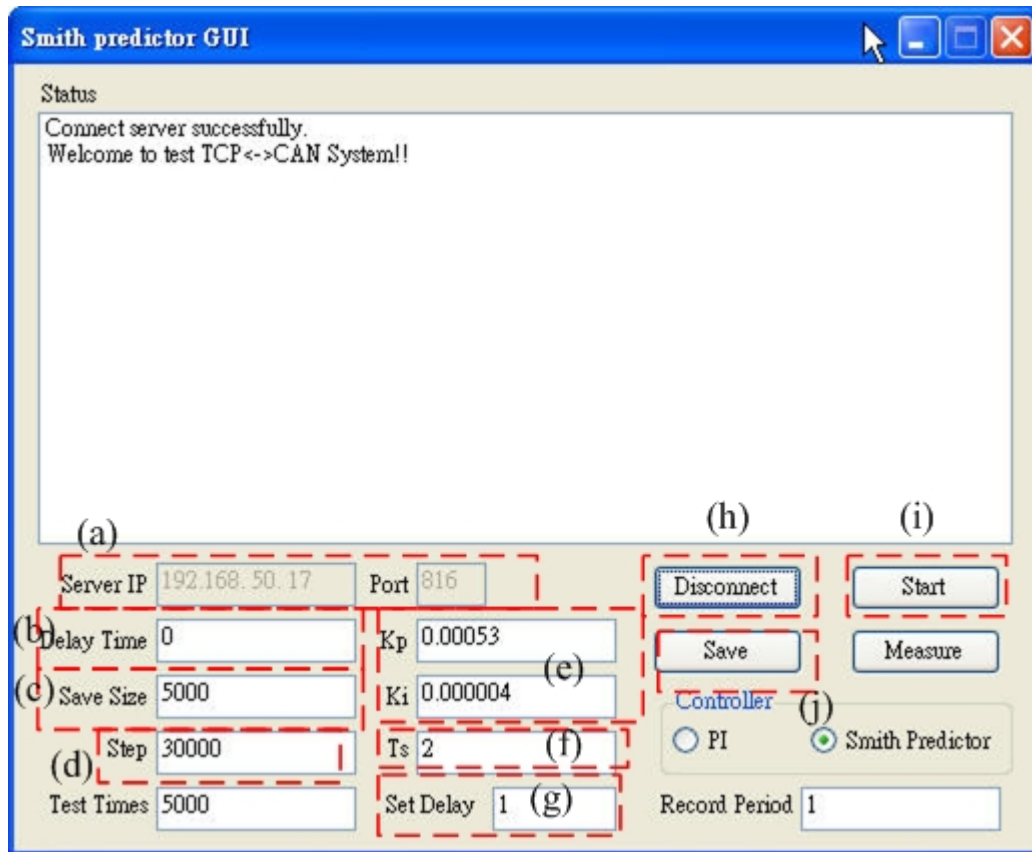


圖 5.25 Smith predictor 人機械面

#### 5-6-4 不同延遲時間對網路控制的影響

此實驗給系統不同的延遲時間來看在不同延遲時間下會對系統帶來什麼影響，並且測試 Smith predictor 是否真的可以消除延遲時間對系統造成的影響，本系統取樣週期為  $2ms$ ，步階輸入大小為 30000 pulse，圖 5.26、圖 5.27、圖 5.28、圖 5.29、圖 5.30 和圖 5.31 為實驗結果，圖中的虛線為直接使用 DSP 不透過網路控制馬達的響應，畫出來以供對照之用。

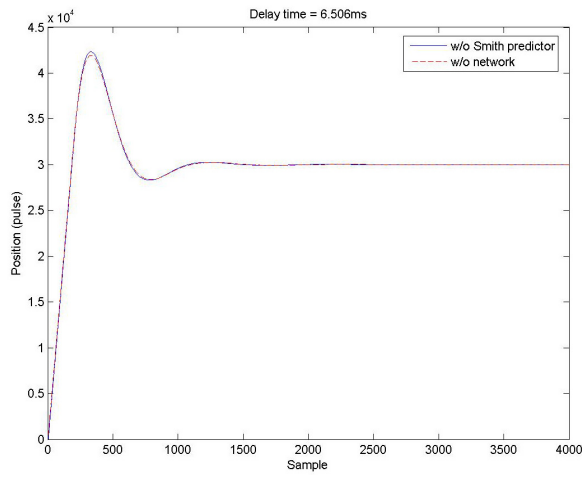
圖 5.26、圖 5.27、圖 5.28、圖 5.29 左半邊的圖均沒有加上 Smith predictor，右半邊則是加上 Smith predictor 以後的結果，圖 5.26 和圖 5.27 是使用 Ethernet 來做傳輸，圖 5.28 和圖 5.29 則是用無線網路來傳輸。由這些圖可以看出，隨著延遲時間變大，延遲時間對系統影響也越大，但是大部分不穩定的系統再加上 Smith predictor 以後，都變回跟原來沒有接上網路的效能差不多，差別只差在加

了 Smith predictor 反應比沒有加網路的系統慢了一些時間，由此可知，Smith predictor 對抗延遲時間有明顯的效果。

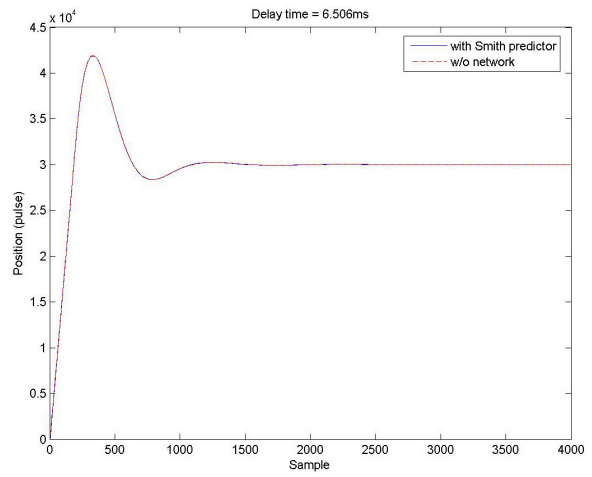
分別拿圖 5.26 跟圖 5.27 還有圖 5.28 跟圖 5.29 來做比較，我們可以發現，反應速度較快的系統，若在還沒使用 Smith predictor 補償的時候，在較小的延遲時間 (56.2405ms) 的時候，系統就已經不穩定，不像反應較慢的系統，在較小的延遲時間 (56.2405ms) 時只是效能變差還不至於不穩定，由此可知，反應速度快的系統比反應速度慢的系統更容易受到延遲時間的影響。

再看圖 5.28 跟圖 5.29，這兩張圖均由無線網路來傳輸，雖然用 Smith predictor 有改善不穩定的情況，但是效果跟有線網路比起來還是有一段差距，尤其是圖 5.29 右下角那一張圖在穩態的時候產生不規則的振動，這是因為我們在算平均延遲時間的時候，連很大的延遲時間也加進去造成我們平均延遲時間大量上升，這種未處理的平均延遲時間直接拿來設計 Smith predictor，會有過度的補償現象發生。因此要在無線網路上面使用 Smith predictor 時，求平均延遲時間要注意是否有很大的延遲時間產生，假如有而且這很大的延遲時間只佔所有傳輸的一小部份的時候，則可以將之忽略不計，如此求得的平均延遲時間比較合乎我們需求，圖 5.30 和圖 5.31 左半邊是使用原始的平均延遲時間右半邊使用處理過的平均延遲時間，很明顯的，使用處理過的平均延遲時間系統效能恢復到跟有線網路一樣的水準。

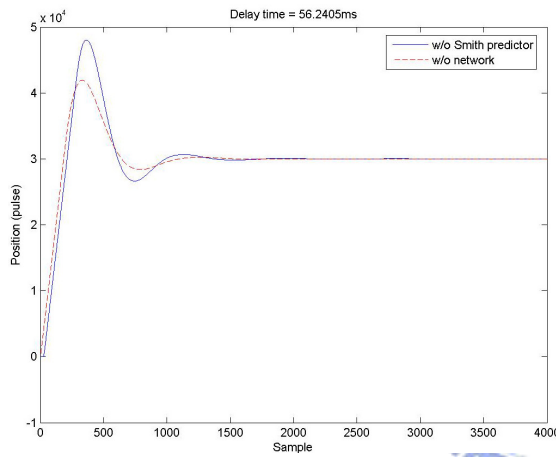




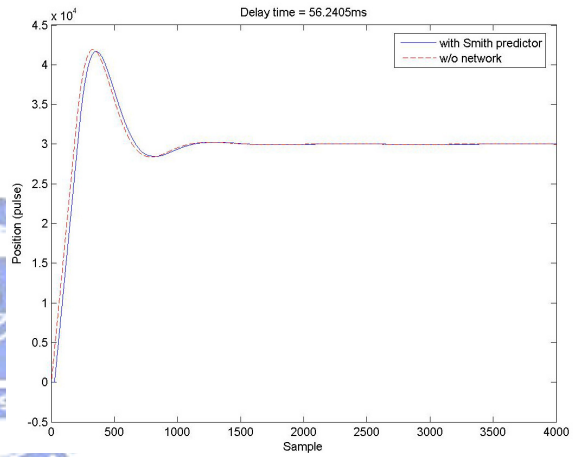
(a-1)



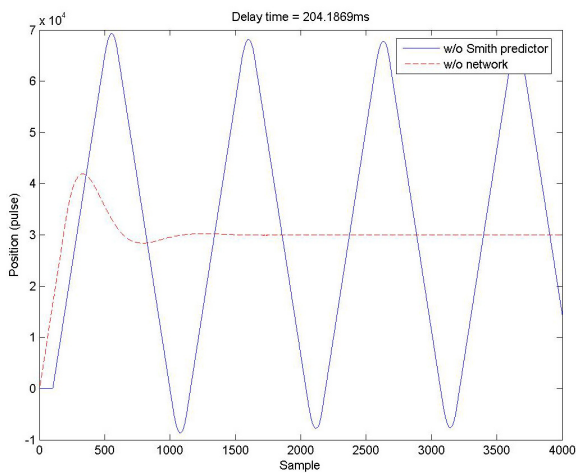
(a-2)



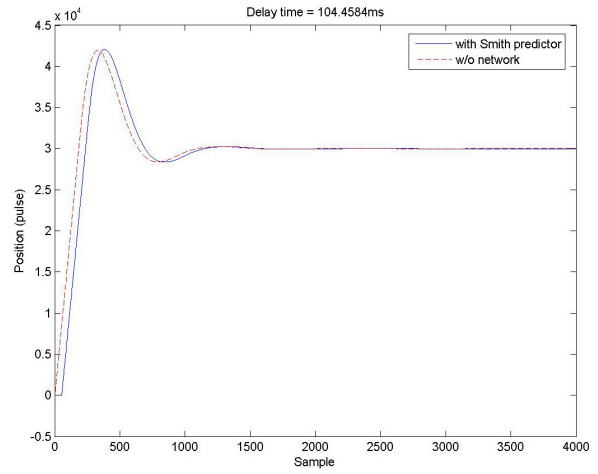
(b-1)



(b-2)

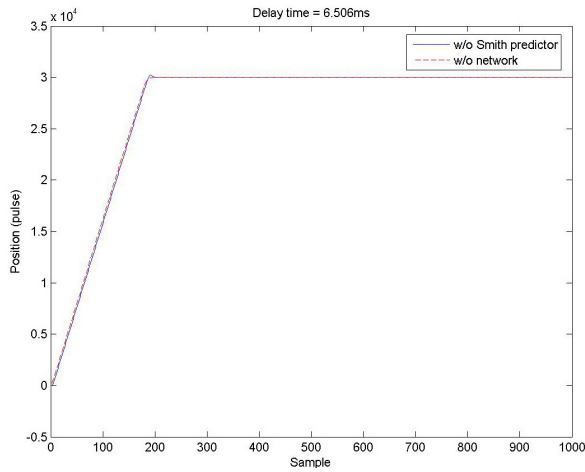


(c-1)

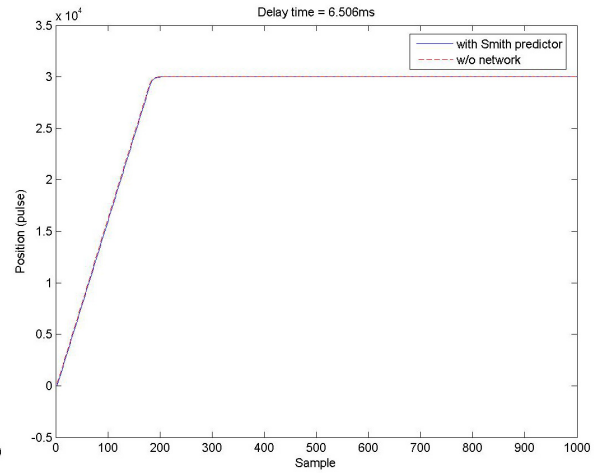


(c-2)

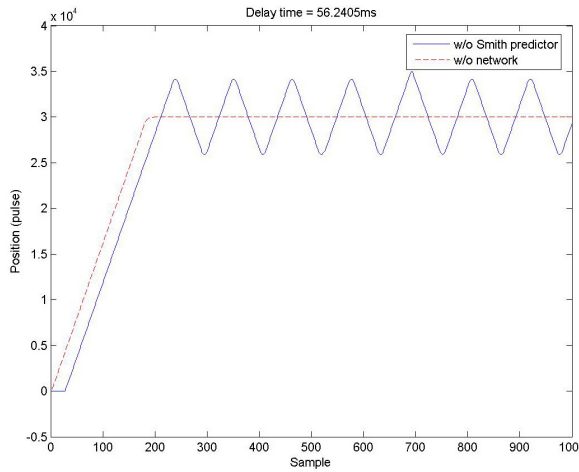
圖 5.26 Ethernet + CAN 之 Smith Predictor 改善效果， $T_{d\_new}$  設為(a) 6.506ms (b) 56.2405ms (c) 204.1869ms (控制器 A)



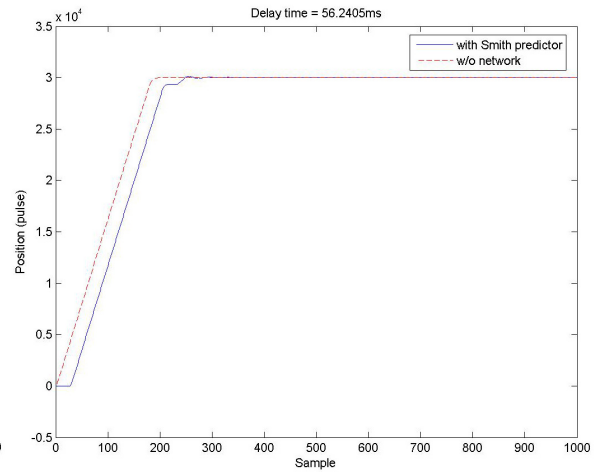
(a-1)



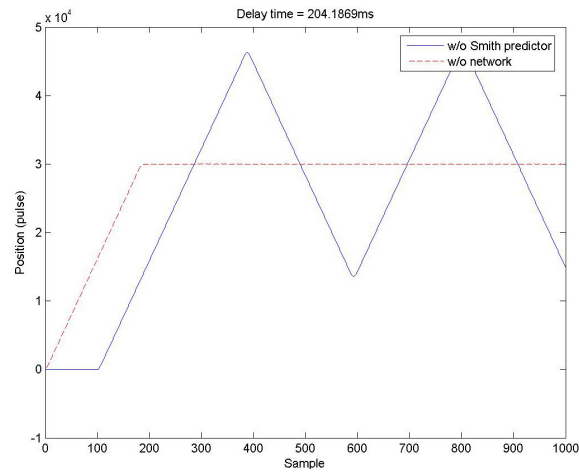
(a-2)



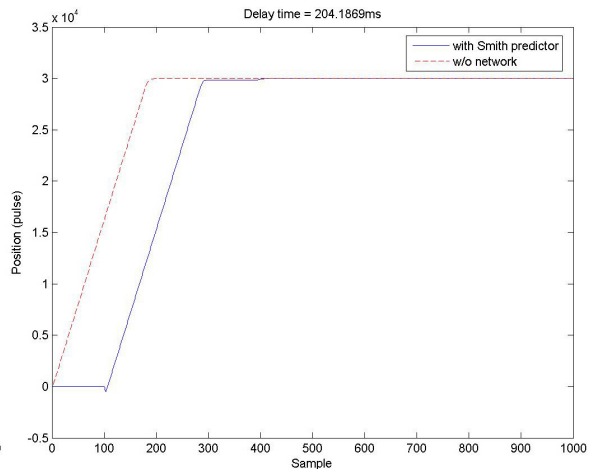
(b-1)



(b-2)

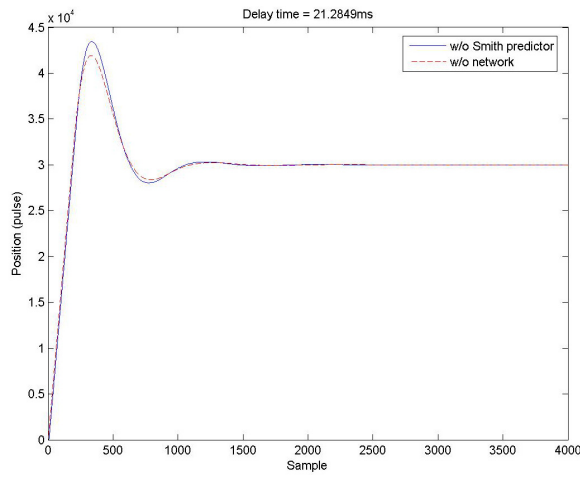


(c-1)

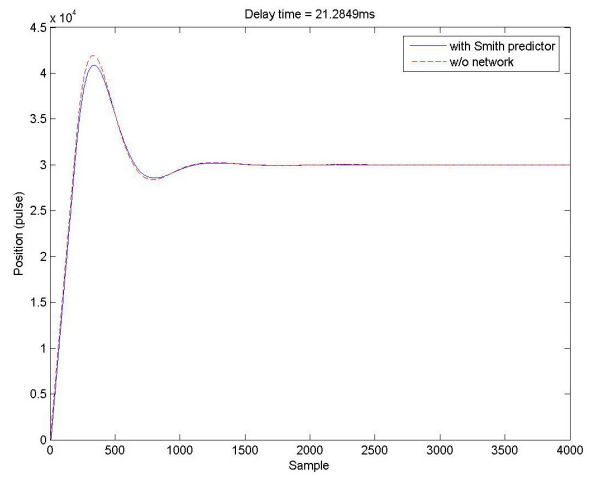


(c-2)

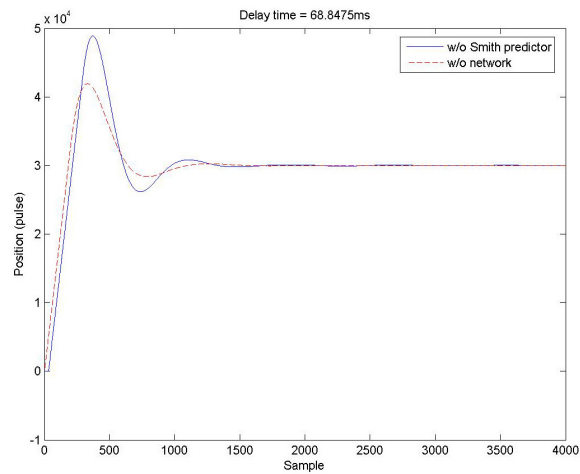
圖 5.27 Ethernet + CAN 之 Smith Predictor 改善效果,  $T_{d\_new}$  設為(a)  $6.506ms$  (b)  $56.2405ms$  (c)  $204.1869ms$  (控制器 B)



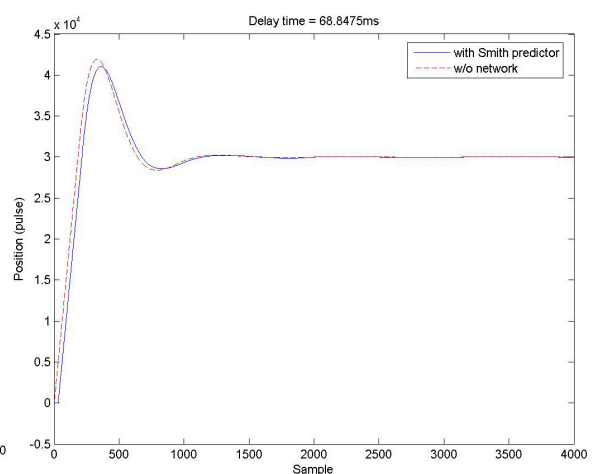
(a-1)



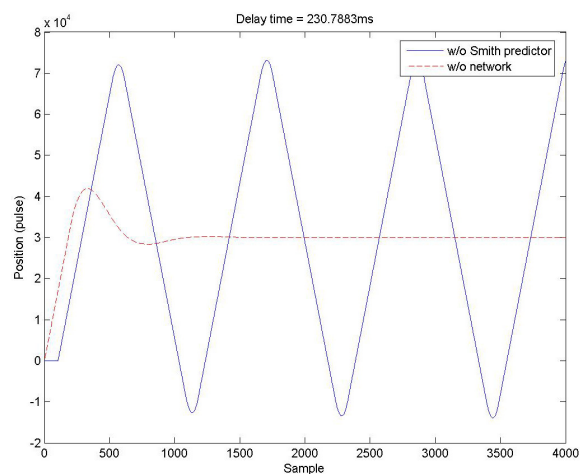
(a-2)



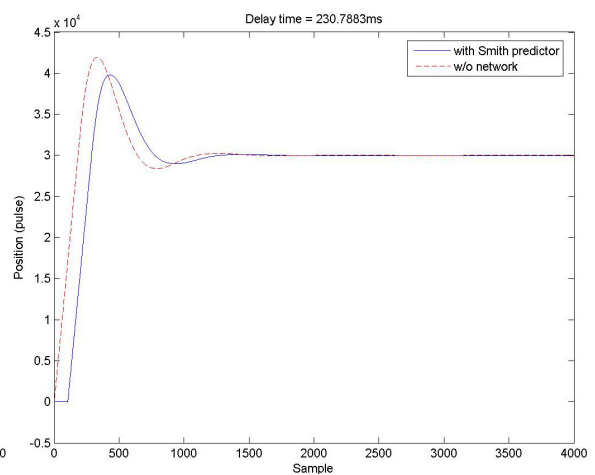
(b-1)



(b-2)

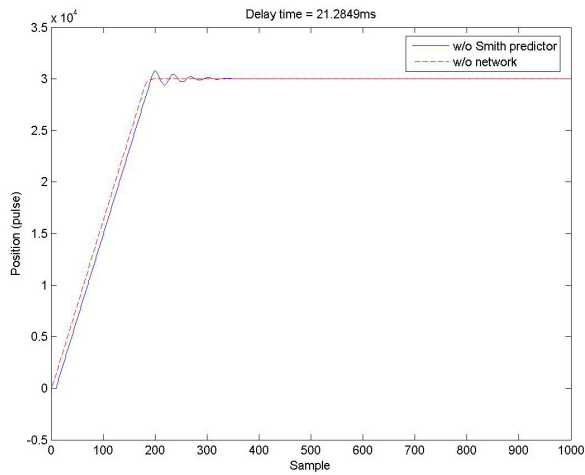


(c-1)

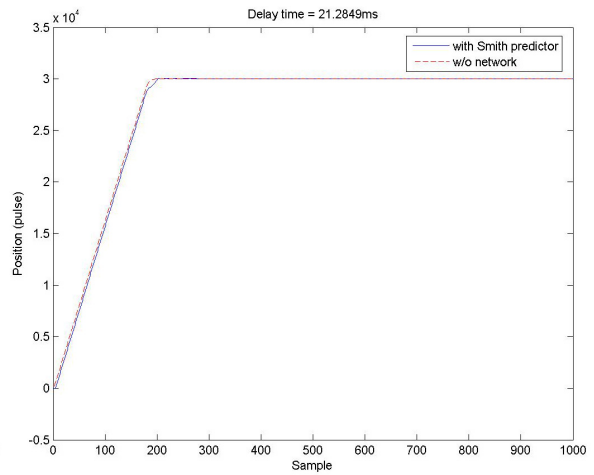


(c-2)

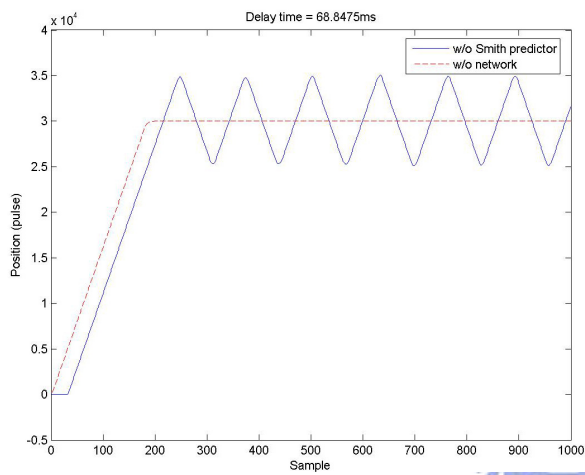
圖 5.28 802.11 + CAN 之 Smith Predictor 改善效果， $T_{d\_new}$  設為(a) 21.2849ms (b) 68.8475ms (c) 230.7883ms (未處理之時間延遲)(控制器 A)



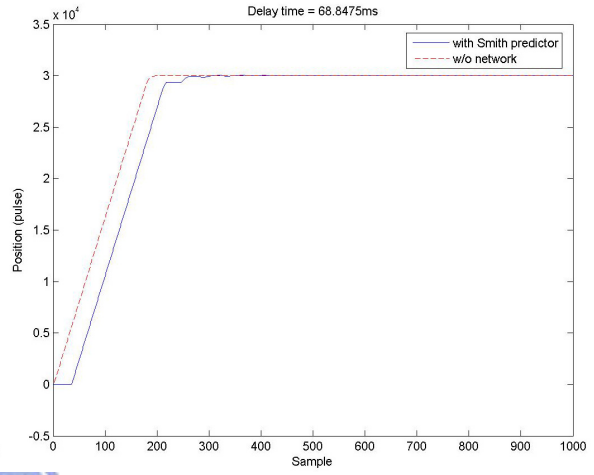
(a-1)



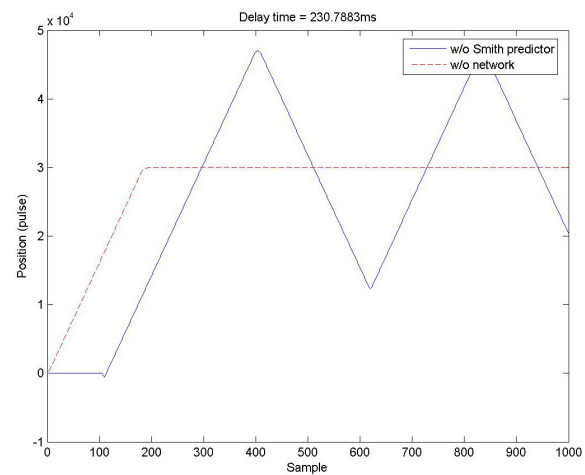
(a-2)



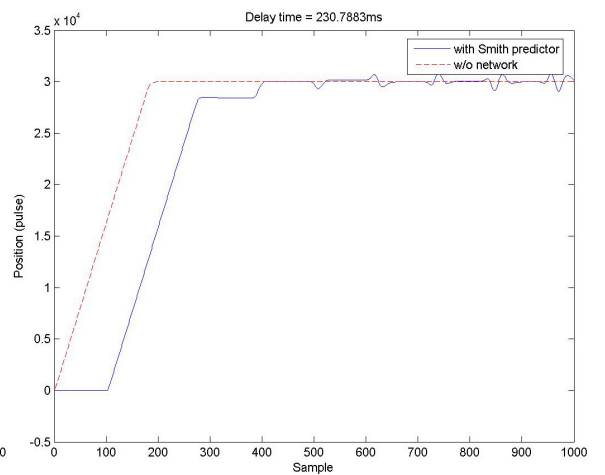
(b-1)



(b-2)

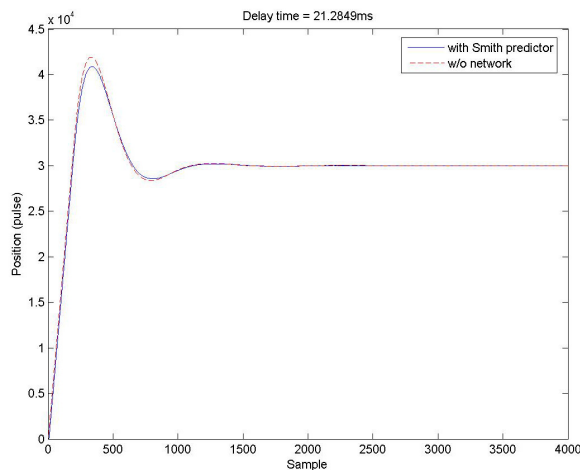


(c-1)

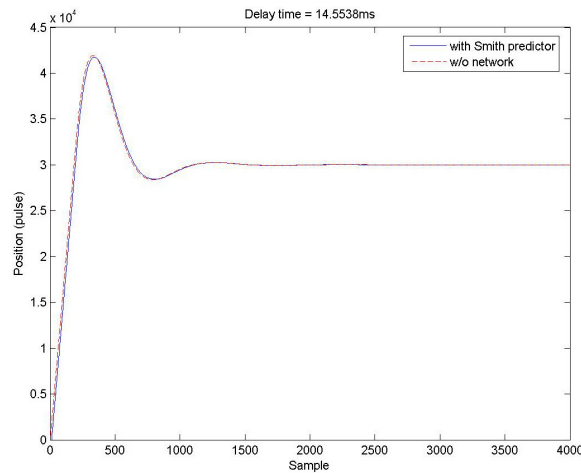


(c-2)

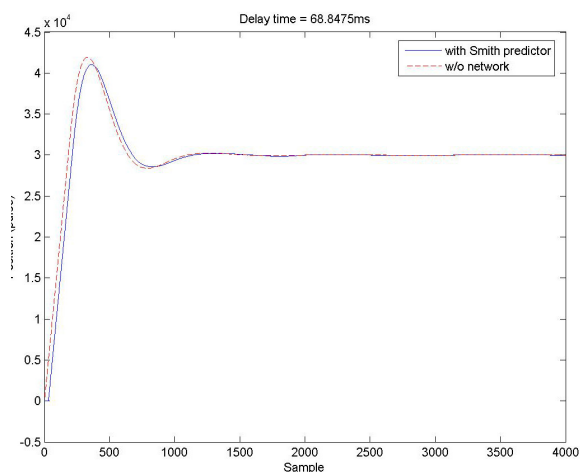
圖 5.29 802.11 + CAN 之 Smith Predictor 改善效果， $T_{d\_new}$  設為(a) 21.2849ms (b) 68.8475ms (c) 230.7883ms (未處理之時間延遲) (控制器 B)



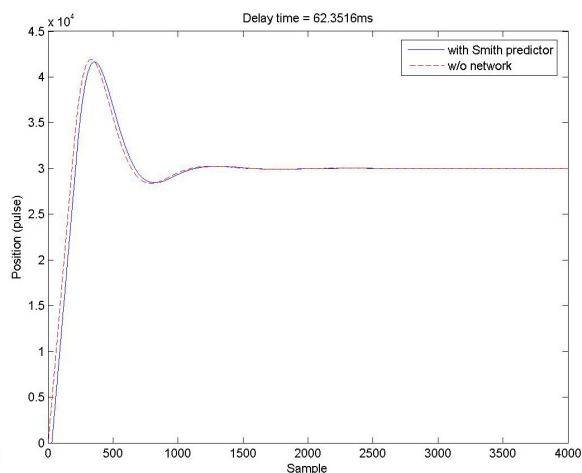
(a-1)



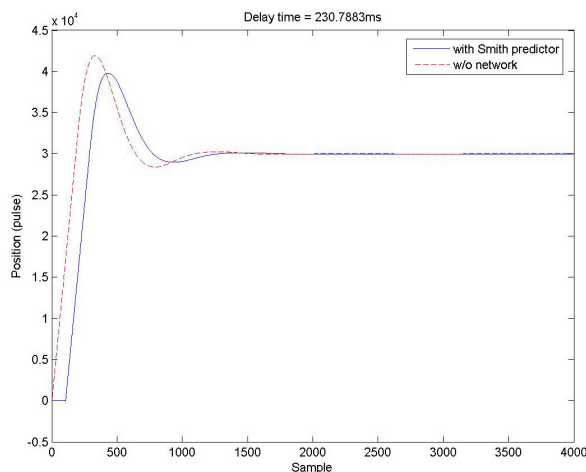
(a-2)



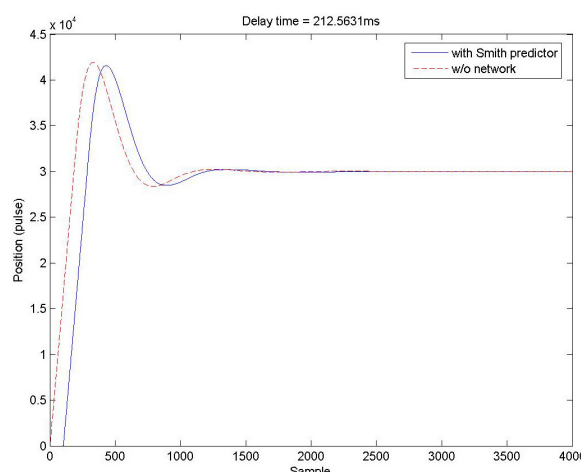
(b-1)



(b-2)

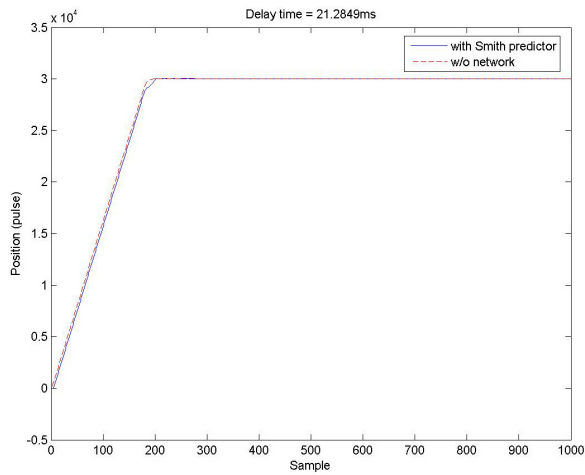


(c-1)

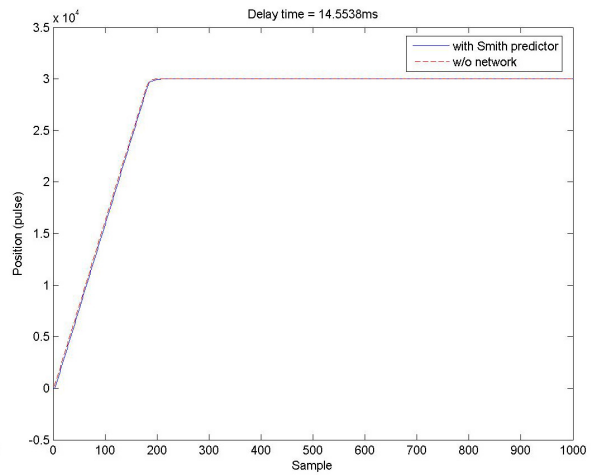


(c-2)

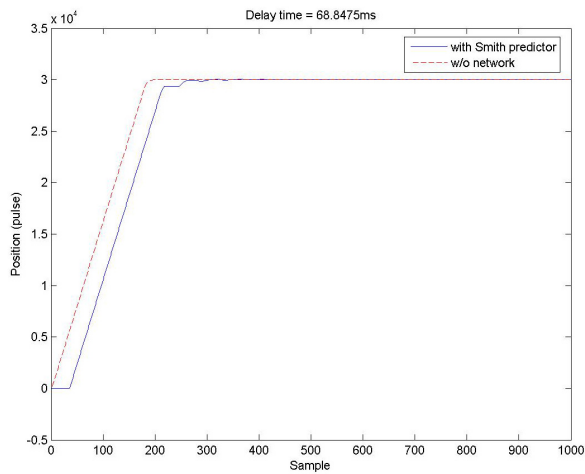
圖 5.30 802.11 + CAN 經過延遲時間處理之控制效能  
 (a)  $21.2849ms \rightarrow 14.5538ms$  (b)  $68.8475ms \rightarrow 62.3516ms$   
 (c)  $230.7883ms \rightarrow 212.5631ms$  (控制器 A)



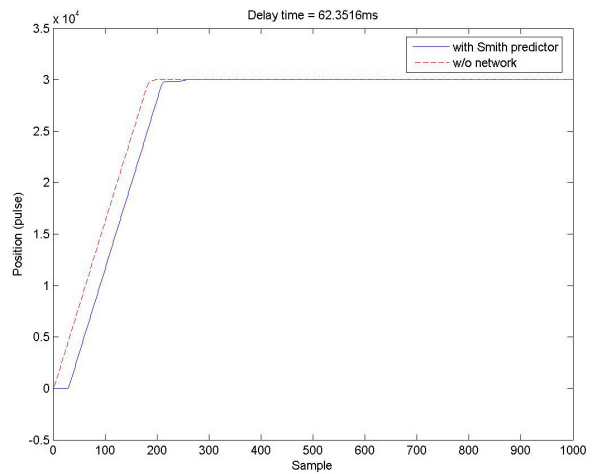
(a-1)



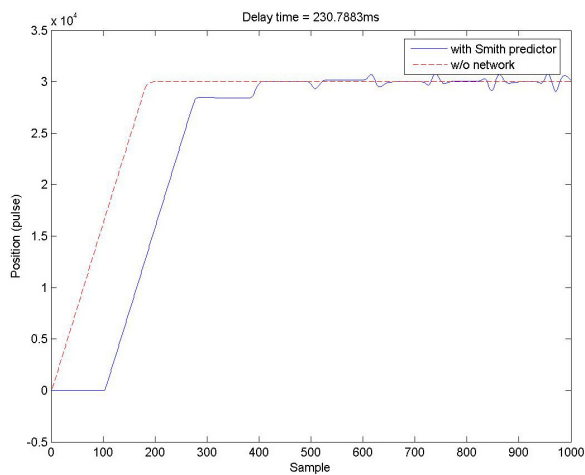
(a-2)



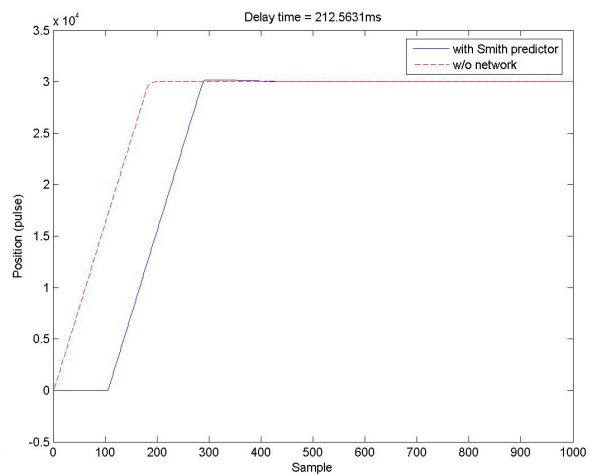
(b-1)



(b-2)



(c-1)



(c-2)

圖 5.31 802.11 + CAN 經過延遲時間處理之控制效能  
 (a)  $21.2849ms \rightarrow 14.5538ms$  (b)  $68.8475ms \rightarrow 62.3516ms$   
 (c)  $230.7883ms \rightarrow 212.5631ms$  (控制器 B)

## 5-7 遠距離網路控制實驗

在本章最後，我們來看由 4-3-5 裡面所提到的從高雄 1 客戶端利用 Smith predictor 控制實驗室馬達的情形，實驗取樣週期為  $2ms$ ，輸入為步階大小 30000 pulse，分別做兩組 PI 控制，一組響應較快其  $K_p = 0.01$ ， $K_i = 0.0000001$  (PI\_A)，另一組響應較慢的  $K_p = 0.0053$ ， $K_i = 0.000004$  (PI\_B)。做完長距離遠端控制之後，再利用 4-3-4 所提到的方法，在近端產生和高雄 1 差不多的延遲時間，並且利用 Smith predictor 來控制，取得結果之後再拿來做比較。

圖 5.32 (a) 為高雄 1 到實驗室的延遲時間分佈圖平均為  $281.4133ms$ ，右邊則為在實驗室近端產生出來的延遲時間分佈圖，平均為  $282.4935ms$ ，兩個平均延遲時間差不多，但是高雄 1 的平均變化似乎比較大，我們來看兩者的標準差，高雄 1 的標準差高達  $21.7946ms$ ，我們自己產生出來的延遲時則為  $3.58ms$ ，雖然兩者平均差不多，但是標準差相差很多，接下來我們來看看不同的標準差會對我們利用 Smith predictor 來做控制會帶來什麼影響。

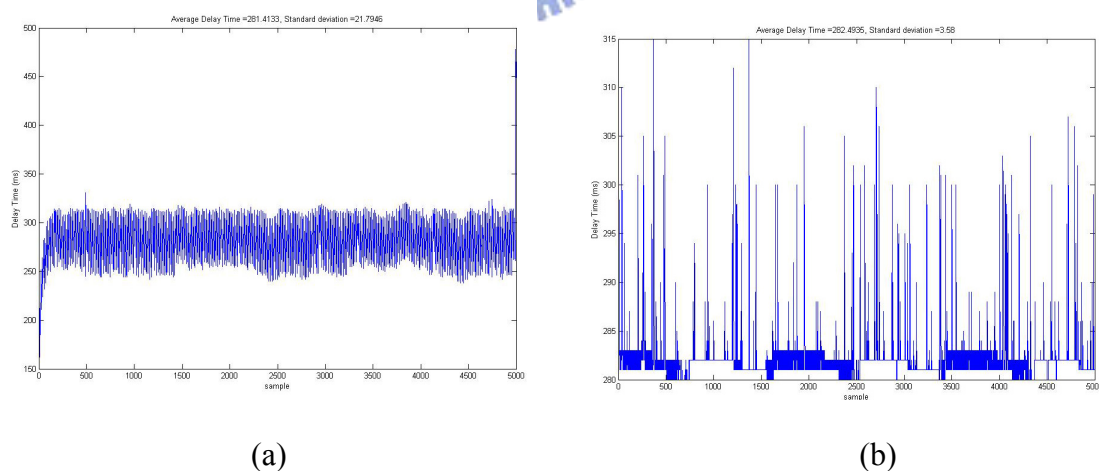


圖 5.32 延遲時間分佈圖(a)高雄 1 遠端傳送，(b)近端自行產生延遲時間

圖 5.33 為  $K_p = 0.00053$ ， $K_i = 0.000004$  時利用 Smith predictor 來補償的響應圖，(a) 為高雄 1 產生的響應圖，(b) 為近端的時間響應圖，由圖形可以明顯發現，

高雄 1 的控制效能比自行產生延遲時間的系統還要差，虛線為沒有透過網路由 DSP 直接控制的響應圖。再來看圖 5.34，圖 5.34 為  $K_p = 0.01$ ， $K_i = 0.0000001$  時利用 Smith predictor 來補償的響應圖，在這張圖可以更明顯的發現高雄 1 有不穩定的現象產生。

由圖 5.33 和圖 5.34 可以看出，當平均延遲時間差不多的時候，並不表示系統經過 Smith predictor 補償後跑出來的響應圖是一樣的，除了平均延遲時間外，我們還需要考慮系統的變異量，系統的變異量可以由標準差看出，當系統的變異量大時，其 Smith predictor 還是有效果存在，但是效果不如變異量小的時候，再來響應速度快的系統似乎比較容易受到變異量的影響而變成不穩定，當有系統因變應量變不穩定的時候，可以考慮將系統放慢下來。

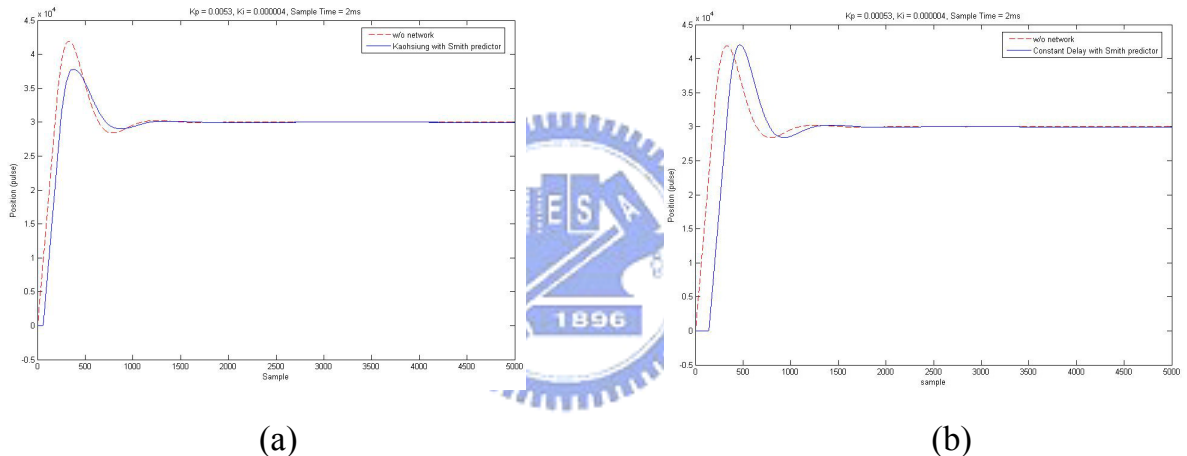


圖 5.33 Ethernet + CAN + Smith predictor 補償的響應圖(a)高雄 1 (b)自行產生延遲時間，(控制器 A)

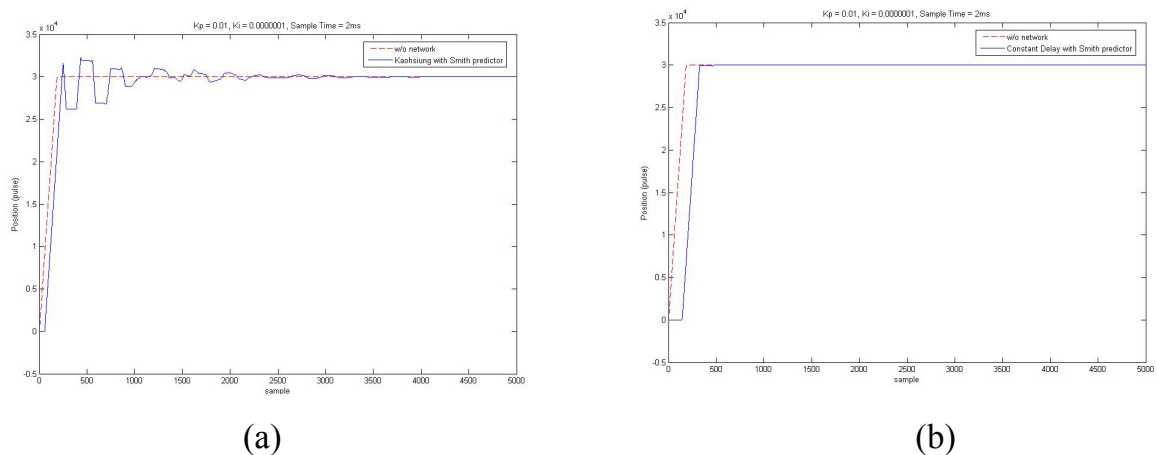


圖 5.34 Ethernet + CAN + Smith predictor 補償的響應圖(a)高雄 1 (b)近端自行產生延遲時間，(控制器 B)



## 第六章 結論與未來展望

由以上的實驗結果，和網路控制系統的建立，本研究達成下列成果：

### 1、 建立 CAN 網路監控系統

本研究在 CAN 訊息資料方面，自訂資料封包，上面包含事件代號，傳送封包節點和資料，再搭配所建立的網路監控系統，可以方便監控網路上面的訊息和除錯。為了讓 DSP 端可以和客戶端互相通訊，CAN 與 Ethernet，以及 CAN 與 802.11 在之間，分別建立可讓 TCP/IP 做封包轉換的 gateway，藉由此 gateway 可以讓所建立的系統透過 Ethernet 和無線網路達到遠端監控的目的。

### 2、 網路時間延遲量測與分析

本研究利用 DSP 傳送帶有 Index 的封包，可以計算客戶端到 DSP 端間應用層對應用層的網路平均延遲時間，在量測過程中發現利用無線網路做量測時，較容易受到干擾，有時後會造成封包的遺失，因本研究使用 TCP/IP 的方式做傳輸，當遇到封包遺失的時候會再重傳遺失的封包，在這重傳過程當中，會造成延遲時間的大量增加，但這情況是偶爾會出現的，因此在計算平均延遲時間的時候，需忽略這些大量增加的延遲時間，來獲取較精確的值，作為控制系統設計的參數，圖 5.30 和圖 5.31 指出透過處理過的量測時間延遲，可達到較正確的遠端監控。

在延遲時間量測結果可以發現，傳送資料的時間間隔越短，會造成系統延遲時間的增加。由交大校園和實驗室量測的數據下發現近距離的變化不會造成網路延遲的增加，在量測遠距離時候可以發現由於傳輸過程中經過多台 Server 跨過不同網域，以及客戶端的網路速度，會使得網路平均延遲時間大量增加，且延遲時間的變動量也跟著變大。

### 3、 建立網路遠端監控系統之延遲時間補償

本研究利用 Smith predictor 成功的消除網路延遲時間對網路控制系統帶來的不良影響。Smith predictor 在具有固定的時間延遲系統下，對抗延遲時間影響有不錯的效果。但在遠距離控制延遲時間發現，遠距離控制的平均延遲時間是有較大變異性的，雖然如此，應用 Smith predictor 還是有效的改進遠端控制性能，但是效果卻比延遲時間變動量小的系統要差。因此，在要評估 Smith predictor 在控制系統效果的時候，除了要考慮是否有固定的延遲時間外，還要考慮延遲時間變異性的大小。

本研究成功的藉由 CAN + Ethernet 及 CAN + 802.11 完成遠端監控系統的設計實驗，除了在交大校園內整合以 CAN 工業網路及 802.11 無線網路的馬達伺服控制，並測試由高雄透過 Ethernet 控制在交大的 CAN 網路馬達伺服控制系統，實現遠端監控系統的設計與應用。



## 參考文獻

- [1] 施浩仁, “網路技術在馬達驅動器的應用”, 機械工業雜誌, Vol. 253, pp197-253, 2004.
- [2] J. Yu, S. Yu and H. Wang, “Survey on the Performance Analysis of Networked Control Systems”, *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 6, pp. 5068-5073, October 2004.
- [3] L. Samaranayake, S. Alahakoon and K. Walgama, “Speed Controller Strategies for Distributed Motion Control via Ethernet”, *Proceedings of the 2003 IEEE International Symposium on Intelligent Control*, pp. 322-327, October 2003.
- [4] 黃祖磊, “以 CAN 為網路協定之分散式資料擷取系統”, 國立交通大學, 碩士論文, 中華民國 87 年 6 月.
- [5] 謝鎮洲, “以 CAN bus 建構出高速精密之多軸運動控制器”, 國立交通大學, 碩士論文, 中華民國 91 年 8 月.
- [6] F. L. Lian, J. Moyne, D. Tilbury, “Network Design Consideration for Distributed Control Systems”, *IEEE Transactions on Control Systems Technology*, Vol. 10, NO. 2, pp. 297-307, March 2002.

- [7] C. C. Hsieh, P. L. Hsu and B. C. Wang “The Motion Message Estimator in Real-Time Network Control Systems”, *Conference of the IEEE Industrial Electronics Society*, November 2006, (to be published).
- [8] 薛伊婷, “以硬體實現主動可調式 CAN 網路排程系統”, 國立交通大學, 碩士論文, 中華民國 93 年 7 月.
- [9] R. LUCK and A. RAY, “An Observer-based Compensator for Distributed Delays”, *Automatica*, Vol. 26, No. 5, pp. 903-908, 1990.
- [10] B. A. Forouzan, “TCP/IP Protocol Suit”, *Mc Graw Hill*, 2001.
- [11] P. Sourdille and A. O’wyer, “A New Modified Smith Predictor Design”, *Proceedings of the 1st international symposium on Information and communication technologies*, Vol. 49, pp. 385-390, September 2003.
- [12] O. J. M. Smith, “Closer Control of loops with dead time”, *Chemical Engineering Progress Transactions*, Vol. 53, pp. 217-219, 1957.
- [13] “CAN Specification Version 2.0”, *BOSCH*, 1991.

- [14] 林宗翰, “802.11 無線點對點網路傳輸率之改善及延遲時間之分析”, 國立交通大學, 碩士論文, 民國 95 年 6 月.
- [15] C. Hunt, “TCP/IP Network Administration, 2/e”, O'REILLY, 1998.
- [16] 吳燦銘, 鄭苑鳳, “TCP/IP 概論”, 五南出版社, 2002.
- [17] “USBCAN-III 智能 CAN 接口卡用戶手冊 V1.2”, 廣州周立功單片機發展有限公司, 2003.
- [18] 蕭永哲, “深入 C++ Builder 探訪動態連結函式庫 (Dynamic Linking Libraries, DLLs)”, Borland Taiwan 網站, 1994.
- [19] 蔡政宏, “發展先進運動控制之 CAD 及其於精密 CNC 之實現”, 國立交通大學, 碩士論文, 中華民國 93 年 7 月.
- [20] 趙清風, “控制系統之識別”, 全華科技圖書有限公司, 2001.