國 立 勤 益 科 技 大 學

機 械 工 程 系 碩 士 班

碩士論文

自然體感機械手臂展示平台之設計與測試

Design and Testing for a Robot Arm Platform with Natural Interaction

研究生：林明毅

指導教授：邱俊智 博士

中華民國 一〇一 年 六 月

自然體感機械手臂展示平台之設計與測試

Design and Testing for a Robot Arm Platform with Natural Interaction

研究生：林明毅
指導教授：邱俊智 博士

國立勤益科技大學

機械工程系碩士班

碩士論文

A Thesis Submitted to
Department of Mechanical Engineering
National Chin-Yi University of Technology
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Mechanical Engineering

June 2012
Taiping, Taichung, Taiwan, Republic of China

中華民國 一〇一 年 六 月

國立勤益科技大學
研究所碩士班
論文口試委員會審定書

本校 機械工程系 碩士班 林明毅 君

所提論文 自然體感機械手臂展示平台之設計與測試

合於碩士資格水準，業經本委員會評審認可。

口試委員：陳正和

楊琳鏗

邱俊智

指導教授：邱俊智

系（所）主管：機械工程系 主任 蔡明義

中華民國 一〇一 年 六 月

自然體感機械手臂展示平台之設計與測試

研究生：林明毅　　**指導教授：邱俊智 博士**

國立勤益科技大學機械工程系碩士班

摘要

本研究主要在探討如何將Microsoft Kinect感測器與機械手臂整合，以快速又穩健之方式打造出一具有自然體感(Natural Interaction)之機械手臂原型平台。這個構想主要源自於救災與危險物品之處理，以及像深海探勘等這類危險的工作，如果能以自然體感直覺的操作方式遠端遙控機械手臂，而非傳統按鍵、類比搖桿這類操作複雜度較高的方式來控制機械手臂，一來可以降低操作學習門檻，二來若搭配3D影像視覺技術更能讓操作者有身歷其境的感覺。

自然體感主要強調人就是控制器這個概念，其優勢主要來自於使用者不需使用任何外部控制器，只需使用自己的身體及聲音等作為虛擬控制器，即可操縱受控體。為實現此一構想並驗證其可行性，本研究使用Kinect深度感測器，結合ROBOTIS 4-DOF機械手臂，建立一自然體感控制平台，並導入Ruby程式語言藉此提高軟體開發效率。本研究測試Kinect深度感測器的特性，同時探討機械手臂之位置控制與末端夾爪之夾取動作控制，並經由一連串的測試證實自然體感的概念與優勢。

關鍵字: Microsoft Kinect Sensor、機械手臂、Ruby Programming Language、Natural Interaction

Design and Testing for a Robot Arm Platform with Natural Interaction

Student：Ming-Yi Lin **Advisor：Dr. Chun-Chih Chiu**

National Chin-Yi University of Technology
Department of Mechanical Engineering

Abstract

The main purpose of the research is to build a prototype platform of a robot arm with Natural Interaction (NI). In a quick and robust way, a Microsoft Kinect sensor was used to integrate with a 4-axis robot arm. The idea was from the handling of hazardous materials, disaster relief management, deep-sea mining, and other dangerous tasks. Instead of using the analog stick or keypad to control the robot arm, if the operators can use their bodies as a controller to remote control the robot arm in a natural way, then the learning curve for the operation can be reduced dramatically. And, if combined with 3D-technology, the remote operator will feel personally on the scene.

NI emphasizes that human is the controller. The main benefit is that operators use their own bodies or voice and need not to carry any extra facility to manipulate the system. To achieve this concept and validate its feasibility, a Kinect depth sensor combined with a ROBOTIS 4-DOF robot arm was used to set up a NI platform. The Ruby programming language was introduced to increase the software development

efficiency. This research tested the features of the Kinect depth sensor. The research also studied the position control of the robot arm and the gripping control of the end-effector. Through a series of experiments, the NI's concepts and its merits have been verified in this research.

Keywords: Microsoft Kinect Sensor、Robot Arm、Ruby Programming Language、Natural Interaction

誌謝

這兩年的碩士生活讓我學到了很多東西，不管是在理論、實作、待人處事等各方面的能力都增進許多，也讓我理解到人真的是潛力無限，只要有心很多事情都是可以做得到的。首先要感謝我的指導教授邱俊智老師，常常在討論時互相交流許多新奇古怪的事物，激發我許許多多的靈感，也給予了我許多寶貴的建議，幫助我完成了我的研究。也要感謝我們機械碩二乙的班導師曾彥魁老師，給了我許多的機會擔任各種不同的職務，像是班代、勤益中工社社長等職務，讓我的人生有了許多截然不同的新體驗，也豐富了我的人生。還要感謝我的家人總是默默的給了我許多的支援，不僅僅只是物質上的支援，同時也是精神上的支持，讓我在研究之餘不必煩心於任何事物。還要感謝實驗室的每位同仁，智鋒、偉超、培勝、元聖、朝盛等，在我當碩士生的日子裡給了我許多的歡笑與汗水。還有好多好多感謝不完的人事物，雖然無法全部都寫完，但卻永遠記在我的腦海裡，往後的日子還很長，我會盡我所能將這份關心不斷的傳遞出去，感染周遭的每個人，並將所學貢獻給社會，期望能為社會帶來更好的福祉。

- 明毅 -

2012/6/24

目錄

圖目錄

國立勤益科技大學 National Chin-Yi University of Technology 勤益 NCUT

表目錄

國立勤益科技大學 National Chin-Yi University of Technology 勤益 NCUT

第一章 緒論

1.1 研究動機與目的

在過去許多的危險工作大多都是由人們親自上火線來執行，像是火災、水災、地震等天然災害的救援，或是像地雷、炸彈等危險物品的拆除工作，而隨著科技的進步越來越多的機器人被開發出來，並投入到了這些危險的工作，嘗試取代人們進入這些危險的地區執行任務，但由於所需執行的任務相當複雜，因此這些機器人通常需要人類的介入來順利完成這些工作，由於機器人的操作複雜，因此在過去若要手動操控機器人，就要熟悉控制器上為數眾多的按鍵，功能越強大的機器人按鍵更多，因此也同時增加了操控上的困難度。直到 2010 年 11 月 4 日，Microsoft 在美國推出 Kinect 感測器，並搭配新版 XBOX 360 遊戲主機於市場上販售，同時推出以體感為主的遊戲嘗試將自然體感技術推向大眾消費市場，以人就是控制器(You are the controller)的口號為號召吸引了無數的玩家，並在短短 60 天內就賣了 800 萬台創下當時銷售新高，Kinect 之所以能如此成功完全在於自然體感的魅力，因此 Kinect 感測器可以說是將自然體感技術帶到大眾消費市場的一大功臣。有鑑於此，本研究導入以自然體感(Natural Interaction)為主的操控方式，藉由仿效人類的動作所設計出來的機器人操控方式來控制機器人，希望能藉此改善機器人的操作體驗，雖然這個概念並不是非常新穎，但卻鮮少有人將自然體感為主的操控方式，帶入到機器人的操控上，因此本研究嘗試開發一基於 Kinect 感測器之自然體感機械手臂原型，希望能藉由此原型展示其相關概念與操控上的優勢，讓更多人了解其相關概念。

1.2 文獻回顧

機器人的應用隨著科技的發展使用率越來越高且越來越廣泛，許多高科技自動化生產線都使用機械手臂幫助上料、組裝等生產作業，而像汽車產業也大量使用機械手臂，取代人們做車體的噴塗、銲接等工作，雖然機器人可以幫助我們做大量重複的工作，但若用來處理較為複雜的工作，像是救災、危險物品之處理，以及像深海探勘等這類危險的工作，由於需要牽扯到相當複雜的影像辨識、機器人定位等，因此在使用上大多都是以手動操控的方式為主，而非使用機器人自主運作之方式。由於使用的目的與環境不同，故各領域的機器人其幾何外形與功能都不盡相同，不過大多數我們所稱的機器人幾乎都不是人形的長相，反而都是一些輪形的居多，其外形多與人類相差甚遠，不過大多數的人們卻都還是習慣稱它們為機器人，即使它們長的實在不怎麼像人類，因此在這裡我們還是把它們統稱為機器人，避免過多的名詞而混淆讀者。近幾年來各國都在研發可以投入到戰場上的機器人，由於機器人是沒有生命的，因此若不幸被摧毀也不會有任何人員傷亡，圖 1.1 即為美國陸軍所使用的 Talon 機器人，從圖中可以發現，使用者可以透過控制器，遠端的遙控機器人來執行任務。

圖 1.1 US Army Talon Robot

資料來源：US Army Official Website (http://www.army.mil/media/85672/)

2011 年 3 月 11 日在日本宮城縣東方外海發生的規模矩震級 9.0 級強震，與緊接引起的海嘯摧毀了許多人的家園，並造成福島第一核電廠的設備損毀、輻射外洩等災害。由於輻射外洩是相當嚴重的事情，因此也不太能夠隨意派遣人員進入處理，此時機器人便派上用場了。由日本核能研究機構(Japan Atomic Energy Agency, JAEA)所開發的一系列機器人，用以偵測溫度、化學物質、輻射、以及清掃污物等，圖 1.2 為 JAEA 所開發的無人直升機，下方搭載了用以量測輻射的儀器。同一時間，美國 iRobot 公司也提供給日本，其所生產的 Warrior 系列機器人，如圖 1.3 所示，用於災害的處理。

a.無人直升機　　b.遠端控制室

圖 1.2 JAEA 所開發的無人直升機
資料來源：JAEA Official Website

圖 1.3 iRobot 所開發的 Warrior 機器人
資料來源：iRobot

由於災後許多的建築物、汽車等都損毀，因此重建的工作相當漫長，而在此時日本 HITACHI 公司推出一款雙臂型建築用機械，並投入賑災以重建家園，希望能夠透過多出來的一隻手臂協助主要的手臂工作，例如剪鋼筋、分解殘骸等，其機器的外形與駕駛艙如圖 1.4 所示。

a.雙臂型建築用機械　　b.駕駛艙

圖 1.4 HITACHI Double Arm Construction Machine ASTACO Neo
資料來源：HITACHI

從這些機器的操控方式來看，可以發現幾乎都是使用相當傳統的操控方式，不外乎就是按鍵、搖桿、踏板等，雖然在使用上經過長時間的考驗，證明其相當的可靠，但在操作上不免還是複雜了許多。而自然體感或許就是解決操控複雜的一個解藥，自然體感雖然不是一個很新穎的概念，但卻是被 Microsoft 所推出的 Kinect 感測器以及其體感遊戲給發揚光大，因此 Kinect 感測器可以說是將自然體感技術帶到大眾消費市場的一大功臣。

Kinect 感測器本身說穿了只是一個深度感測器,但與多數的深度感測器不一樣的地方在於，它是使用紅外線結構光形式的感測技術，且感測到的是資料是三維的，跟一般雷射測距儀是二維的不太一樣。在過去，要量測一個場景的深度，最常見的都是使用兩個彩色攝影機，運用立體視覺的方式得到場景的深度，但是這樣的感測方式有個很大的缺點，若今天量測的對象是一個均一且完全沒有任何特徵的物件表面，問題不就大了嗎，因為根本算不出正確的距離，且在晚上時若不打光也完全無法運作，[1]因此早在 2005 年美國 NASA 的噴射推進實驗室(Jet Propulsion Laboratory, JPL)，便開發出一使用結構光感測技術的微型檢查用飛行器(Free-Flying Micro-Inspector Spacecraft)用於母船的檢查，而當中所使用的結構光感測技術與硬體架構，都與現在的 Kinect 所使用的感測技術有許多相似之處，如圖 1.5 所示。

圖 1.5 微型檢查用飛行器所使用的結構光感測技術
資料來源：文獻[1]

但 Kinect 也不是完全沒有缺點，由於 Kinect 抓取到的深度影像本身會因為視角、物件材質等的不同，而會有部分區域資料缺失的狀況發生，如圖 1.6d-f 的黑色區域所示，[2]故 S.Matyunin 等提出一方法，嘗試透過 Kinect 的彩色攝影機所提供的彩色影像來填補這些缺失的資料，並應用在前處理的階段，藉此進一步改善 Kinect 感測器所提供的深度影像之穩定性與可靠度，改善結果如圖 1.6g-i 所示。

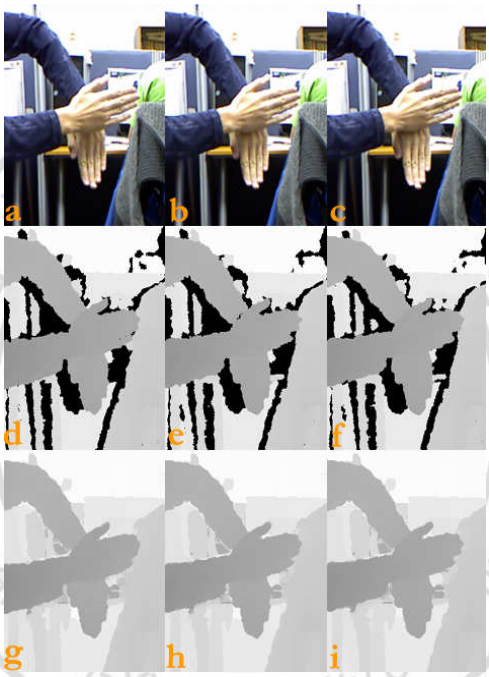


圖 1.6 Kinect 感測器部分區域資料缺失之改善
資料來源：文獻[2]

有些研究者也嘗試將 Kinect 感測器，運用到各種不同的機器人身上，如圖 1.7 所示，像是四槳直升機(Quadrotor Helicopter)的高度控制上[3]，以及強化機器人室內定位的應用上，由於一般室內機器人的定位常使用到馬達的編碼器來計算里程，但這有個致命的缺點，那就是誤差容易隨著時間而不斷累積，因此 N.Ganganath 等嘗試將 Kinect 導入到機器人的定位上，藉以強化機器人的室內定位能力[4]，

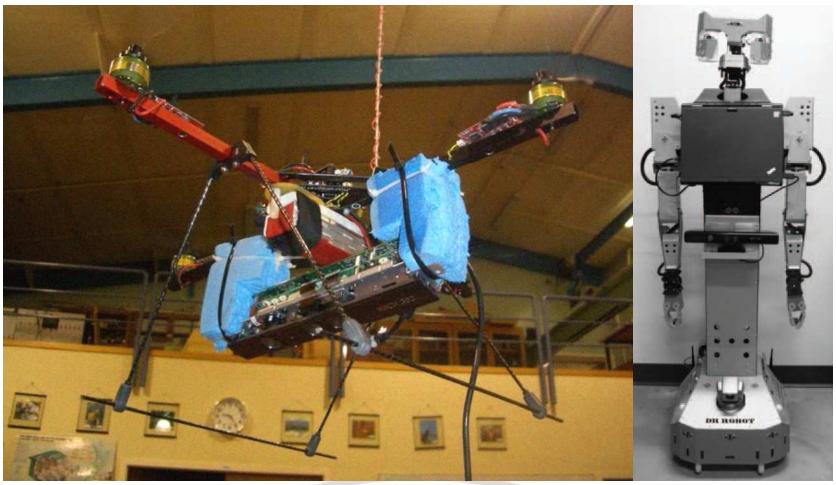


圖 1.7 Kinect 感測器運用於各種不同的機器人身上

資料來源：文獻[3, 4]

由於 Kinect 可以輕鬆的抓取到場景的深度，因此 J.Tong[5]等透過導入多部 Kinect 來開發三維掃描裝置(3D Scanning Devices)，如圖 1.8 所示，並提出一方法來強化三維模型的建置，其結果顯示，跟傳統的三維掃描裝置比較起來更為便宜，並且能夠在幾分鐘之內產生出相當專業的三維模型，這樣的模型可以用在虛擬人物等的製作上，相當具有潛力。

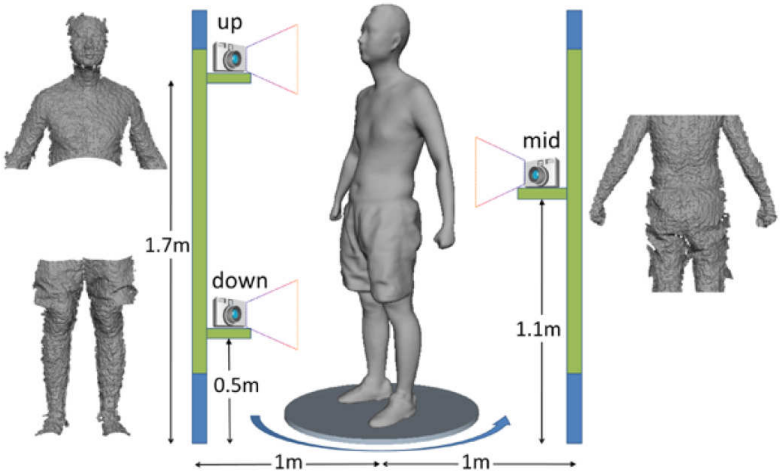


圖 1.8 由多部 Kinect 感測器所組成之三維掃描裝置

資料來源：文獻[5]

另一個相當有趣的研究，是由 C.Harrison[6]等所開發的一可穿戴式深度感測投影系統，可以讓使用者透過可穿戴式的多點觸控互動介面與應用程式互動，透過深度感測可以抓取到要投影的目標物之三維表面形狀，與使用者的手部動作辨識，在進一步透過計算單元處理後經由微型投影機，將資訊投影至目標物上如:手掌、筆記本、桌子、牆上等，如圖 1.9 所示。

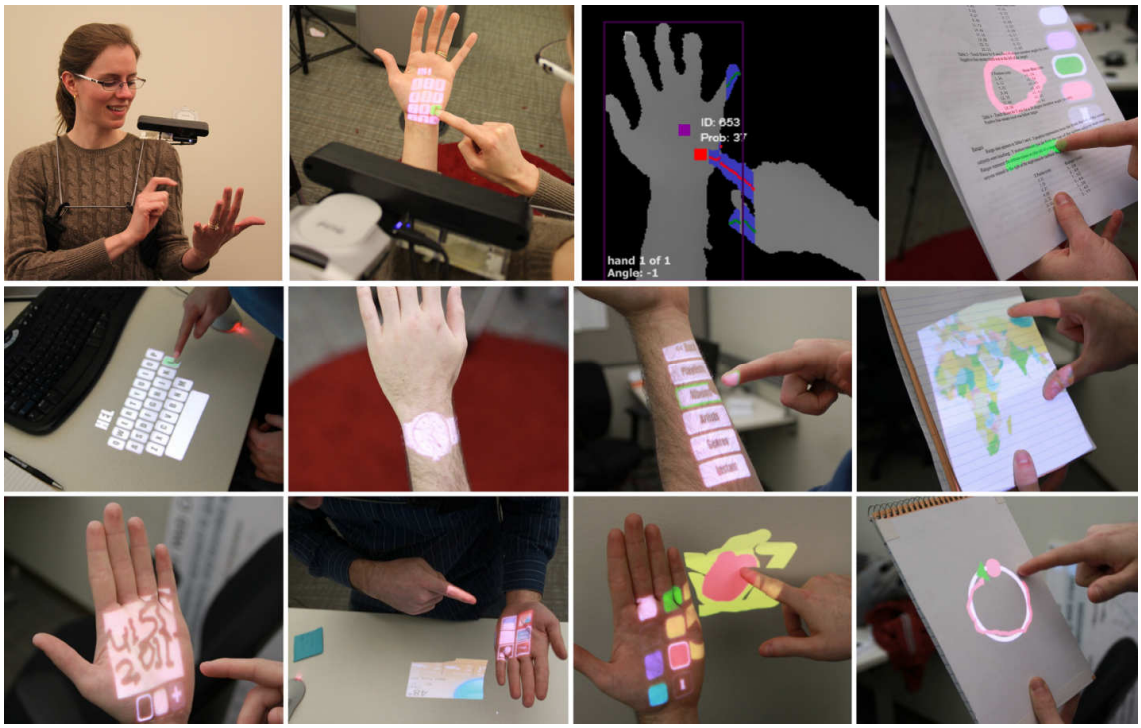


圖 1.9 可穿戴式深度感測投影系統之操作實例
資料來源：文獻[6]

由於自然體感的操控方式是以人為基礎的操控方式，因此在有些應用中若能夠透過電腦來了解使用者的各種動作，便可透過這些資訊讓電腦與使用者互動，如打籃球、高爾夫球等。有鑑於此， L.A.Schwarz[7]等提出基於深度影像資料與彩色影像資料的骨架追蹤方法，該方法甚至不需要任何的訓練資料，即可估測出

使用者各個關節的位置。有了相關的骨架追蹤方法後， L.Gallo[8]等便透過類似的技巧將其辨識技術應用到醫療方面，並提出一基於 Kinect 之非接觸式醫療影像操作系統，讓手術中的醫生可以完全不用汙染到雙手，就能輕鬆的操作醫療影像。由於在醫療領域需要非常重視衛生，因此非接觸式的醫療影像操作系統就會是一個很好的選擇，可以避免接觸感染的機會發生，其操作範例如圖 1.10 所示。而在深度影像的壓縮方面， S.Mehrotra[9]等提出一低複雜度之近無損失深度影像壓縮技術，用以壓縮 16-bit 解析度之深度影像，其結果顯示，他們所提出的壓縮技術可提供約 7:1 至 16:1 的近無損失壓縮率，而編碼(或解碼)一張深度影像只需要約 5ms。此壓縮技術可以用於深度影像資料的保存與傳輸等。

圖 1.10 基於 Kinect 之非接觸式醫療影像操作系統操作實例
資料來源：文獻[8]

由此可發現基於 Kinect 的應用與研究真的相當的多且豐富。而在軟體開發方面，由於近年來軟體系統的複雜度越來越高，故在軟體開發效率上的需求也逐漸增加，由於現在電腦效能成長非常快速，因此越來越多開發者使用像 Ruby、Python、Perl 這類動態型別語言來提高軟體系統的開發效率，舉例來說 Ruby on Rails 這套網頁開發框架就是一個相當好的例子，這幾年也因為 Ruby on Rails 的崛起讓許多人發現到 Ruby 的強大，不過這類動態型別語言也都普遍有著相同的缺點，那就是執行效率不佳的問題，為了開發效率而犧牲掉些許的執行效率這種方式，在硬體資源豐富的電腦平台上還行的通，但若想在硬體資源較為缺乏的嵌入式系統(Embedded System)上使用 Ruby 來開發軟體就會遇到瓶頸，因此這幾年 Ruby 的創作者嘗試開發精簡化的 Ruby，希望讓開發者也能在嵌入式系統上使用 Ruby 來開發程式[10]。在過去嵌入式系統的軟體開發幾乎都是 C/C++的天下，由於近年來嵌入式系統的發展越來越多元化，需求量也越來越大，複雜度也越來越高，因此若能夠導入 Ruby 到嵌入式系統軟體的開發中，絕對能夠帶來許多好處。

1.3 論文架構

第一章 緒論

此章節主要在闡述本研究的動機與目的，以及相關的文獻探討與回顧。

第二章 自然體感(Natural Interaction)

此章節主要是針對自然體感這個主題所開設的，目的在闡述自然體感這個概念，讓閱讀本論文的讀者能夠更了解整體的概念與優勢，非常值得多費些心力來仔細閱讀，也希望相關的概念能影響更多人，借以強化各種應用的使用者操作體驗。

第三章 軟體技術與硬體設備之介紹

此章節主要在介紹本研究所使用到的相關軟體技術與硬體設備，其中又以Ruby 程式語言、Microsoft Kinect Sensor 為主，由於這兩者是整個實作的核心元素，且也較為新穎，對許多人來說或許相當陌生，因此也特地花費較多篇幅與心力，嘗試將這兩者完整的介紹給讀者。

第四章 自然體感機械手臂原型之設計與實作

此章節主要針對自然體感機械手臂原型的設計與實作兩大部份，詳細敘述整個原型平台的各項開發細節與原理。文中分為 1.系統架構、2.自定義之機械手臂操作方式、3.系統之運作流程、4.機械手臂之設計與實作、5.機械手臂末端夾爪之夾取力量控制、6. Kinect 感測器於手部追蹤時之雜訊抑制等六大部分，並分別針對各主題進行詳盡的探討。

第五章 自然體感機械手臂原型之測試與分析

此章節主要針對本研究所設計並實作出來的自然體感機械手臂原型，進行一連串的效能、雜訊抑制、操作等測試，以評估整個系統的後續發展性。

第六章 結論

最後對本研究的結果進行討論，並針對未來的後續發展與需要改善的部分做些探討。

國立勤益科技大學 勤益 NCUT National Chin-Yi University of Technology

第二章 自然體感(Natural Interaction)

2.1 前言

在過去若要手動操作一台機器人，無論是人型機器人或是輪型機器人，幾乎都是使用類比搖桿、鍵盤來進行操作，而這種傳統的操作方式有著操作複雜的缺點。舉例來說，若想使用傳統的方式手動操作如圖 2.1 所示之機器人，在前、後、左、右及加減速等操作上還不太有什麼大問題，但若連機械手臂及視訊鏡頭的操作也涵蓋進來的話，操作複雜度絕對會提升許多，若沒經過適當的訓練想隨心所欲的操控機器人絕對是件相當困難的事。而若使用自然體感的方式來進行手動操作，操作者只需直覺的移動身體到想要前往的方向即可控制機器人的行進方向，而使用者移動的量則可控制機器人之加減速狀態，機械手臂的操作只需移動使用者的手臂即可控制機械手臂做出三維之相對運動，而使用者頭部的方位即可控制視訊鏡頭的方向，操作上幾乎與人的習慣一致，因此使用自然體感的方式來控制機械人絕對能夠有效降低操作複雜度並改善操作體驗。

圖 2.1 傳統使用類比搖桿、鍵盤當作輸入來手動操控機器人
資料來源：iRobot

2.2 何謂自然體感

自然體感為人機介面(Human-Machine Interface, HMI)的子集合，人機介面從以前到現在一直是一個很熱門的主題，因為人和機器之間的溝通都必須仰賴人機介面，而在這當中人機介面的友善程度將會直接影響到使用者的操作體驗。

人機介面的發展從鍵盤滑鼠與內建類比搖桿的操作手把，到 2006 年最紅的 Nintendo Wii 的體感控制器，無一不是為了增加使用者操作體驗而生，且都是以更為自然直覺的操作方式為導向所衍生出來的人機介面。在這裡不免要說明一下何謂自然且直覺的操作方式，在這世界上很多我們習以為常的行為與操作物件的方式，如:拿筷子吃飯、走路、說話，其實都是需要學習的(除了像呼吸這類自發性行為外)，因此所謂自然且直覺的操作方式，基本上都是基於人們平常生活中習以為常的行為再加以衍生出來的操作方式，而其操作方式若越接近人們平常習以為常的行為與操作物件的方式就越自然且直覺，相對的學習門檻就越低，進而提升使用者的操作體驗。自然體感就是運用了這樣子自然且直覺的特性所發展出來的新穎之人機介面。在過去若想運用非接觸式三維感測技術於自然體感這類人機介面上，幾乎都是使用兩個彩色的攝影機，以仿人眼視覺系統的方式來達到三維空間的量測，這樣子的架構下成本相對較低廉，但卻容易受到亮度、色調、物件特徵等所影響。直到 2010 年 11 月 4 日 Microsoft 在美國推出新版 XBOX360 遊戲主機與 Kinect 感測器，同時推出以體感為主的遊戲開始於零售市場上販售，當中最受矚目的就是新推出的 Kinect 感測器，創下當時銷售新高 60 天內就賣了 800 萬台，這個設備徹底顛覆了過去的遊戲機市場，以人就是控制器(You are the controller)的口號吸引了無數的玩家，當中的魔力正是前面所提及的更友善且更直覺的人機介面(關於 Kinect 感測器的介紹將於文中 3.2 節詳細說明)，而國外許多玩家與研究者

發現其潛力，開始紛紛投入其軟體開發並嘗試移植到 PC 上以拓展應用層面。本研究主要也是運用 Kinect 感測器於機械手臂的控制，期望以快速又穩健之方式打造出一具有自然體感之機械手臂原型。圖 2.2 為人機介面的演進。

自然體感主要強調人就是控制器這個概念，其優勢主要來自於使用者不需使用任何外部控制器，只需使用自己的身體及語音等人類與生俱來的直覺進行操作，即可將身體和聲音作為虛擬控制器驅動任一使用者期望控制的設備。因此，從上述的優點來看可預期在未來自然體感的應用將會越來越多也越來越廣泛。

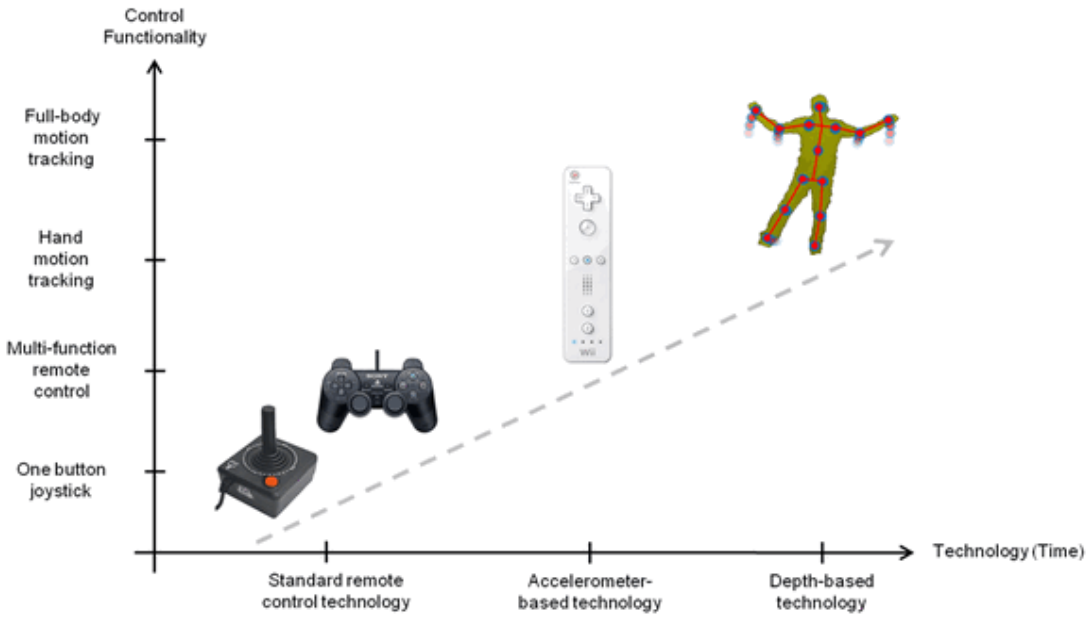


圖 2.2 人機介面的演進
資料來源：PrimeSense

2.3 Kinect 感測器於自然體感之相關應用

直至目前為止 Kinect 感測器的應用仍在不斷的快速增加當中，許多構想不斷被發展出來，應用領域也不斷在擴張，例如：機器人領域、娛樂領域、甚至是復

健醫療領域，都有人將 Kinect 導入其中。圖 2.3、圖 2.4 為 Kinect 感測器應用於機器人領域之實例。

圖 2.3 Kinect 感測器應用於無人載具之實例
資料來源：http://www.gamefront.com/10-best-kinect-hacks-so-far-list/

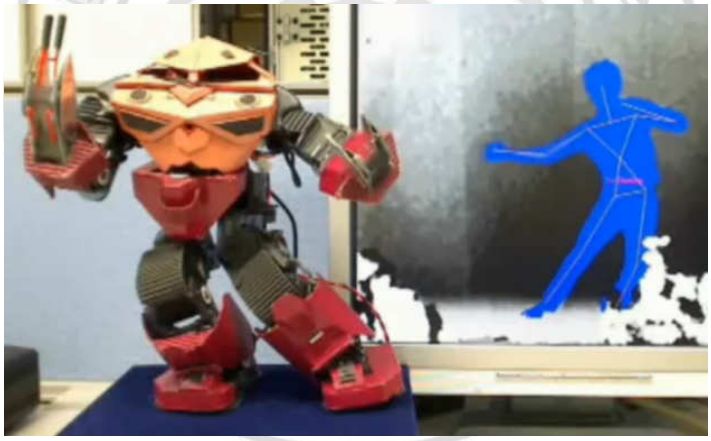


圖 2.4 Kinect 感測器應用於人形機器人之操作實例
資料來源：http://blog.esuteru.com/archives/2243459.html

在眾多應用中最早出現也相當具代表性的應用，是由美國南加州大學(University of Southern California)發展的 FAAST(Flexible Action and Articulated Skeleton)這個專案[11, 12]，它可以讓使用者以腳本的方式預先定義使用者的各種手勢、姿勢與鍵盤按鍵、滑鼠事件間的對應關係，如此便能夠以預先定義的姿勢

與手勢操作軟體或是玩遊戲。使用 FAAST 玩遊戲的例子如圖 2.5 所示。

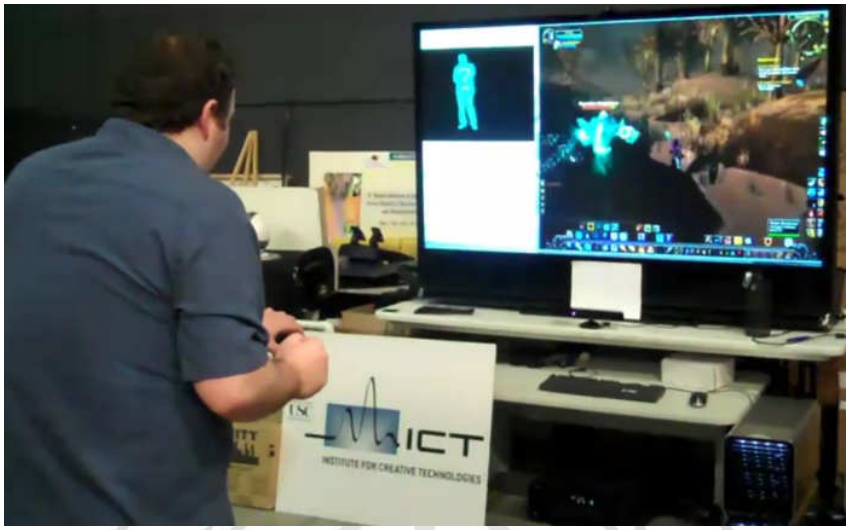


圖 2.5 使用 FAAST 玩魔獸世界
資料來源：http://projects.ict.usc.edu/mxr/faast/faast-video-gallery/

另一個也相當具代表性的應用是在英國有個電視節目叫”The Gadget Show”，他們花了將近 65 萬美元打造了一台”第一人稱射擊遊戲模擬器”(First-Person Shooter Gaming Simulator)，如圖 2.6、圖 2.7 所示。該模擬器為求擬真，因此在軟硬體上下了相當大的功夫，並以 EA 的 Battlefield 3(中文譯作"戰地風雲 3")這款 2011 當紅的第一人稱射擊遊戲為主軸，為了讓玩家感受到身歷其境的感覺而非只是單純的玩射擊遊戲，因此他們使用 Kinect 感測器追蹤使用者的動作姿態並導入漆彈槍作為遊戲角色中彈時之回饋，且因場地大小有限故他們在玩家腳下設置了一個類似跑步機的全方向原地移動機構(Omni-Directional Treadmill)，讓玩家在有限的空間下盡情的移動而不受限制。該模擬器在節目播出後得到相當大的迴響，也是 FPS 史上相當重要的里程碑。文獻[13]為 The Gadget Show 提供的 YouTube 影片連結。

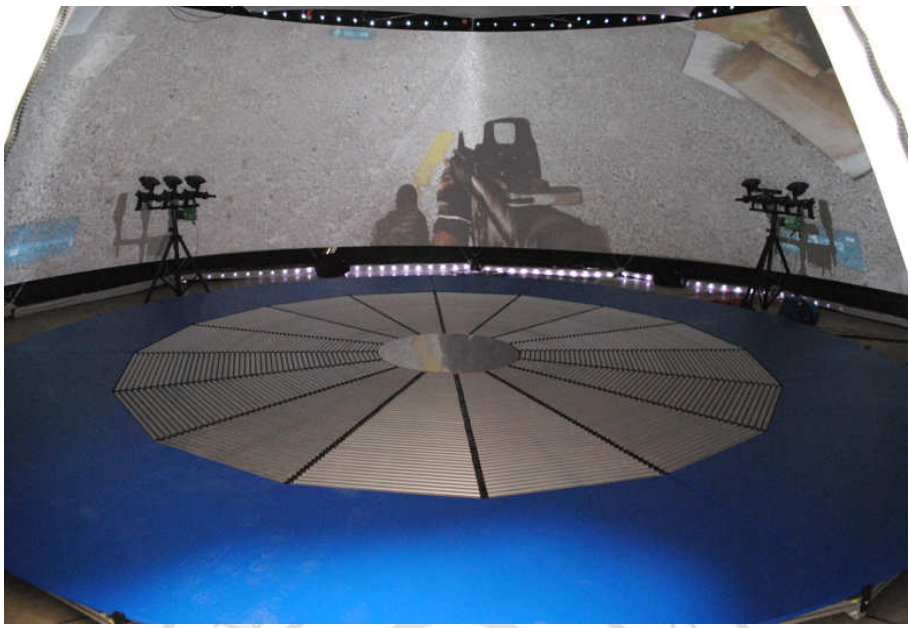


圖 2.6 FPS Gaming Simulator

資料來源：Engadget

圖 2.7 玩家於 FPS Gaming Simulator 內之操作實例

資料來源：The Gadget Show

第三章 軟體技術與硬體設備之介紹

本章將針對所使用的軟體技術與硬體設備依序做介紹，文中主要分為五大部分，分別為 1.Ruby Programming Language、2.Microsoft Kinect Sensor、3.機械手臂、4.壓力感測器、5.資料擷取卡，並針對各個主題進行詳細的介紹與說明。

3.1 Ruby Programming Language

[14, 15]Ruby 是一個來自於日本的程式語言，由一位日本人名叫 Yukihiro Matsumoto(中文譯作:松本行弘，網路暱稱:Matz)於 1993 年開始開發的程式語言，1995 年釋出第一個版本(目前穩定版本:Ruby 1.9.3-p194, Apr 2012[16])，並於 2006 年被 TIOBE 獲選為年度程式語言(Programming Language of the Year)。Ruby 的官方 Logo 如圖 3.1 所示。

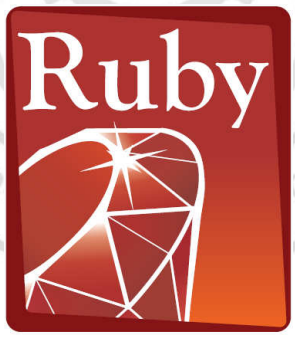


圖 3.1 The Official Ruby Logo
資料來源:Ruby Association

Ruby 屬於動態強型(Dynamic Strong Type)、純物件導向的腳本語言，其本身受到 C、Perl、Python、Smalltalk、Eiffel、Ada、Lisp 等眾多語言的影響，致力於發展成一語法簡潔清晰且又強大的通用型跨平台程式語言。在過去硬體資源較為缺

乏的年代為了不浪費硬體資源，幾乎都是使用像 C/C++這類的靜態型別程式語言，提供許多資訊(例如:型別等)以利編譯器對程式碼作完善的規劃與最佳化，藉以提高軟體效能。但在現在這個硬體資源飛快成長的年代，硬體資源早已不再是瓶頸，取而代之的是高生產力的需求。Brooks 在他的書中曾經提到[17]，一位程式設計師在一固定的時間內所產出的程式碼行數，不管他是使用哪種語言其數量基本上都是固定的，因此若一個程式設計師一天可以寫 500 行程式碼，那麼不管他是寫組合語言還是 C 語言或是 Ruby，一天之內應該都能寫出 500 行左右的程式碼，但是 500 行的 C 與 500 行的 Ruby 程式碼所能做到的事情相比實在差非常多。

Ruby 的創作者(Matz)一直希望，能夠透過 Ruby 這個語言提供更多的生產力給程式設計師們，讓各位程式設計師能脫離工作上的阻礙，找回程式設計的樂趣，雖然這樣的目標會犧牲掉許多執行效率，但別忘了我們現在可是身在硬體資源飛快成長的年代(現在連筆記型電腦都比當年的超級電腦快上許多)，因此效能問題隨著時間的增加，將會變得不再這麼為人所詬病。

3.1.1 Ruby 的特色

◎簡潔性

簡潔就是力量。為寫出更簡潔有力的程式碼，Ruby 本身以更抽象的方式嘗試提供更有力的語言架構與語法，並將許多較為低階繁雜的工作與事物都隱藏了起來(例如:記憶體的分配與回收、型別檢查等)，讓程式設計師可以把火力都集中在目標上，而不用為了多餘且非必要的事物，影響其思維。

◎擴充性

做為一個通用型程式語言，因無法預料其應用範圍(例如:氣象資料分析、網站開發、金融財務分析、生物資訊領域等)，因此語言本身必須具備良好的擴充性與

彈性，才能應付各種不同的領域和情形。因此 Ruby 本身透過提供中介編程(Metaprogramming)、開放類別等眾多語言特色，輕鬆的讓程式於執行期間動態改變程式的行為，且開發者可以依照自身需求創造新類別或修改既有類別，大幅提升了語言的擴充性與彈性。由於官方釋出的 Ruby 直譯器是以 C 語言為基礎實作出來的並提供 Ruby 的 C API，因此不管是想要突破瓶頸提高程式的執行速度，或是想在 Ruby 中使用只有 C/C++語言才能使用的資源，以及在 C/C++程式內部呼叫 Ruby 等，都可以藉由 Ruby 的 C API 達到擴充外部類別庫與提升效能等目的。且由於 Ruby 本身是開放原始碼(Open Source)的，因此若要對 Ruby 內部進行修改也是可行的。

3.1.2 為何選擇 Ruby

Ruby 相較於其它主流語言來說算是相當年輕的語言，由於它與 Python、Perl 都是相當受歡迎的腳本語言且在定位上也相當類似，因此也是最常被拿來相互比較的對象。基本上選擇 Ruby 有一部分是個人喜好(被 Ruby 的特色所吸引)，另一部分是快速開發上的需要。個人喜好的原因在於 Ruby 在許多地方上更優於 Perl 且又比 Python 更物件導向，對 Ruby 來說所有的東西都是物件，因此在 Ruby 中並沒有所謂的原生型別(Primitive Type)，包括數值也不例外，因此你可以對任何的物件呼叫方法操作它，雖然這樣做會影響到效率，但卻提高了程式碼的可讀性與一致性。Ruby 在語法上下了許多功夫，因此使用起來非常直覺，不管是開發、除錯或是維護都相當容易。選擇 Ruby 的另一個原因是快速開發上的需要，由於本研究想要快速打造自然體感機械手臂的原型，為了不花太多時間在程式開發上，因此捨棄傳統最常被使用的 C/C++、Java 語言，嘗試導入 Ruby 並以此為主力進行開發，

希望能從中獲得更高的生產力，讓更多的時間能夠專注在自然體感機械手臂原型的開發上，這就是為何本研究選擇了 Ruby。

3.1.3 計算密集型與 I/O 密集型

一般我們寫出來的程式可以分成，計算密集型(Computationally Intensive)與 I/O 密集型(I/O Intensive)兩大類，而這兩者之間最大的不同就是，一個是偏向計算居多(計算密集型，例如:大量數學計算)，另一個則是偏向 I/O 存取動作居多(I/O 密集型，例如:GUI、檔案存取、網路資料存取)，這當中最容易讓人感覺到程式執行速度差異的就是計算密集型的程式，這考驗著你寫的程式演算法好不好，電腦的硬體夠不夠快，程式執行的效率高不高，基本上由於 Ruby 是屬於直譯式的程式語言，因此在執行的效率上一般都會低於 C/C++所寫出來的程式，但這就是使用高階語言所必需要付出的代價，慶幸的是大多數的應用情況都是屬於 I/O 密集型居多，因此在許多狀況下使用 Ruby 並不會覺得程式執行的效能不佳，因為程式多數的時間都處於 I/O 的等待狀態，而 I/O 的速度快慢基本上就是取決於裝置的速度，因此大多數的狀況下使用 Ruby 來開發程式，是可以在不犧牲太多執行效能的情形下換取開發速度的。倘若還是不幸遭遇到程式執行效能上的瓶頸，可以嘗試將瓶頸的部分以 C 擴充(C Extension)的方式改寫，以提升程式整體的執行速度。

3.2 Microsoft Kinect Sensor

Kinect 是由 Microsoft 公司與以色列的 PrimeSense 公司合作，於 2010 年 11 月 4 日在美國推出的一款非接觸式三維影像感測裝置，並同時內附體感為主的遊戲軟體於市場上販售，在短短 60 天內就賣了 800 萬台創下當時銷售新高，也成功的將

非接觸式三維感測技術融入到了遊戲機內，完全顛覆了以往遊戲機的既定印象，為遊戲機市場帶來了渾然不同的操作體驗。許多玩家與開發者發現了自然體感帶來的優勢，紛紛嘗試將其優勢帶到更多領域，也因此創造了許多讓人意想不到的專案。

3.2.1 Kinect 感測器之硬體規格

Kinect 感測器是由三個主要的光學元件所組成，分別為 1.紅外線投影器(IR Projector)、2.彩色攝影機(RGB Camera)、3.紅外線攝影機(IR Camera)，如圖 3.2 所示。然而除了這三個主要的光學元件外另一個也相當重要的元件就是 PrimeSense 自己開發的 SoC 晶片(PrimeSense PS1080 SoC)，這顆 SoC 負責將感測到的紅外線訊號轉變為深度值，因此若光學元件是 Kinect 的四肢，那這顆 SoC 無疑是 Kinect 的心臟。Kinect 感測器的硬體規格如表 3.1 所示，由於感測器本身的電源設計並沒有依循標準的 USB 規格(5V)來設計，因此在使用上就必須額外透過外接電源(購買感測器時內附的變壓器, 12V)才能正常運作。詳細資料可以參閱文獻[18, 19, 20]。

圖 3.2 Kinect 感測器

資料來源：http://www.codeproject.com/Articles/317974/KinectDepthSmoothing

表3.1 Kinect感測器硬體規格

Field of View (Horizontal, Vertical) :	57° H, 43° V
Mechanized Tilt Range (Vertical) :	±28°
Image Size (Depth、Color) :	VGA (640x480), 30FPS
Spatial X/Y Resolution (@ 2m distance from sensor) :	3mm
Depth Z Resolution (@ 2m distance from sensor) :	1cm
Operation Range :	0.8m - 3.5m
Data Interface :	USB 2.0
Dimensions (Width x Height x Depth) :	14cm x 3.5cm x 5cm

資料來源：Microsoft、PrimeSense

3.2.2 Kinect 感測器之運作原理

Kinect 的感測技術主要是使用 PrimeSense 所開發，名叫 Light Coding™的感測技術，其感測方式是屬於紅外線結構光(Infrared Structured Light)的量測方式[21]，但與傳統紅外線結構光量測不同的地方在於，其投影至場景的紅外線編碼影像是由許多的點所構成，並依據不同距離時點距之變化計算出實際的場景深度值，如圖 3.3 所示。圖 3.4 為本研究所使用的 Kinect，利用光學照相機所拍得的 IR Projector 投影至場景的紅外線編碼影像(Light Coding™ IR Image)。Kinect 感測器運作流程如圖 3.5 所示，首先是以面型近紅外線雷射光(Near-IR Light, Class1)對整個場景進行編碼，再以影像感測器(Standard CMOS Image Sensor)接收已編碼過之場景紅外線影像，並將接收到之紅外線影像訊號經由 PrimeSense 的 SoC 晶片(PrimeSense PS1080 SoC)進行複雜的演算法處理，最後再將處理結果(即該場景之深度影像資訊)透過 USB2.0 介面傳回至個人電腦主機，當主機端接收到資料後即可依據需求加以客製化應用。

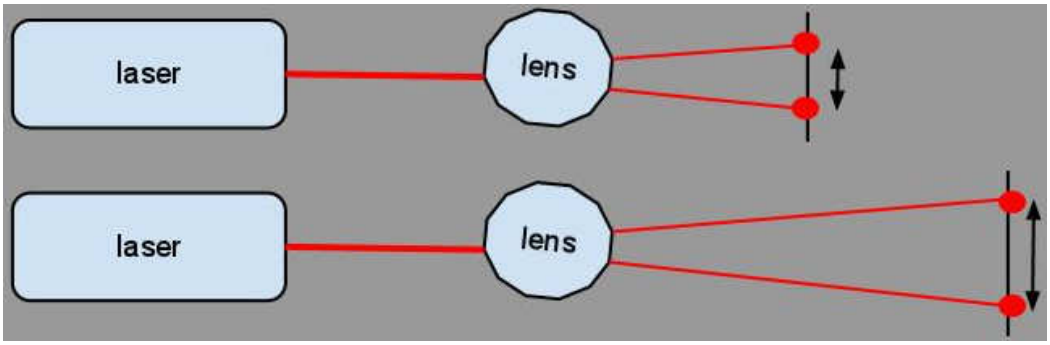


圖3.3 Light Coding™感測技術原理示意圖

資料來源：文獻[21]

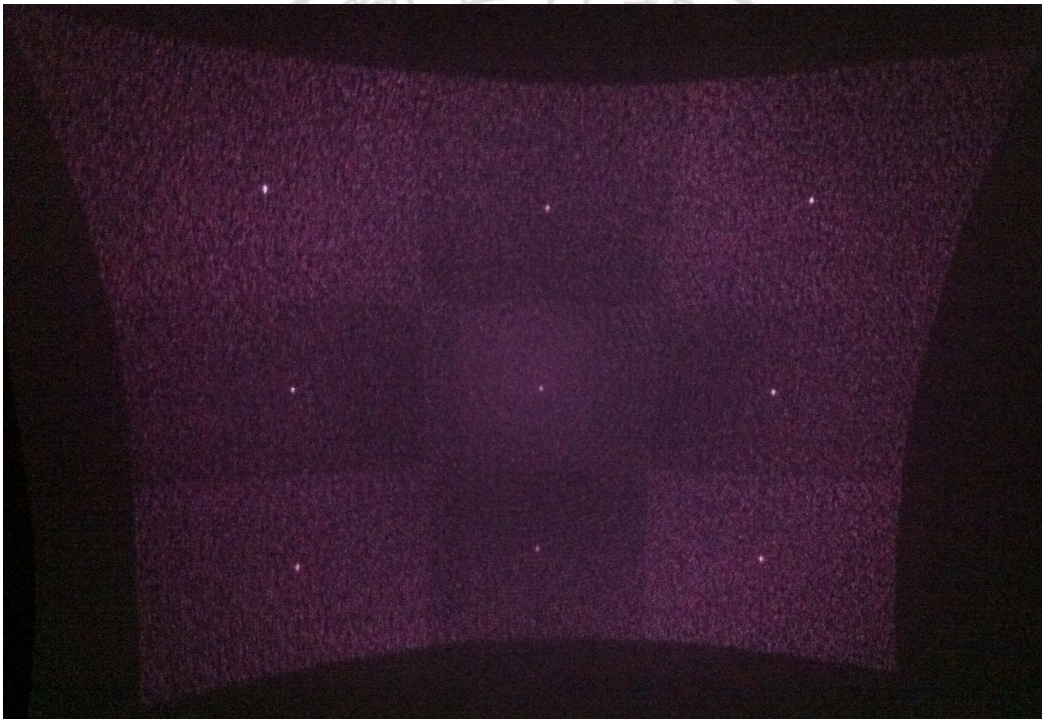


圖3.4 Light Coding™ IR Image

圖 3.5 Kinect 感測器運作流程
資料來源：PrimeSense

3.2.3 Kinect 軟體開發工具介紹

截至目前為止，若想在電腦上開發基於 Microsoft Kinect 感測器的應用程式，其主要軟體開發工具(SDK)有三種，分別為 Libfreenect、OpenNI Framework、Kinect for Windows SDK，以下將分別針對這三種開發工具進行介紹。

◎Libfreenect (Provided by OpenKinect Community)

Libfreenect 是最早出現在網路上的非 Microsoft 官方開發工具，係由 OpenKinect 這個開放原始碼社群計畫所開發，且開發工具內也一併提供該社群自行開發的 Kinect 感測器驅動程式，這樣一來解決了底層硬體的驅動問題，而且它支援 C、

C++、.NET(C#/VB.NET)、Java(JNA and JNI)、Python 等眾多程式語言，且能夠橫跨主流的三大作業系統平台(Windows、Linux、Mac)可說是相當全面，非常適合學術研究使用。由於 OpenKinect 是開放原始碼的計畫，且以社群的方式在經營，因此，若你想為其它尚未支援的程式語言撰寫操作 Kinect 的 API 也是相當歡迎的，像 MATLAB、LabVIEW 的擴充就是由社群成員所開發撰寫。但基本上 Libfreenect 只提供串流資料的讀取與硬體的控制，許多的高階功能都要由自己實作(例如：人體骨架辨識與追蹤、物件追蹤、手勢辨識等)，因此大多都會搭配 OpenCV 等額外的函式庫一起使用。若只是想單純應用而非學術研究性質的話，則使用 Libfreenect 開發 Kinect 相關應用上難度較高。圖 3.6 為使用 Libfreenect 於 Linux 平台上擷取到的深度影像資訊。

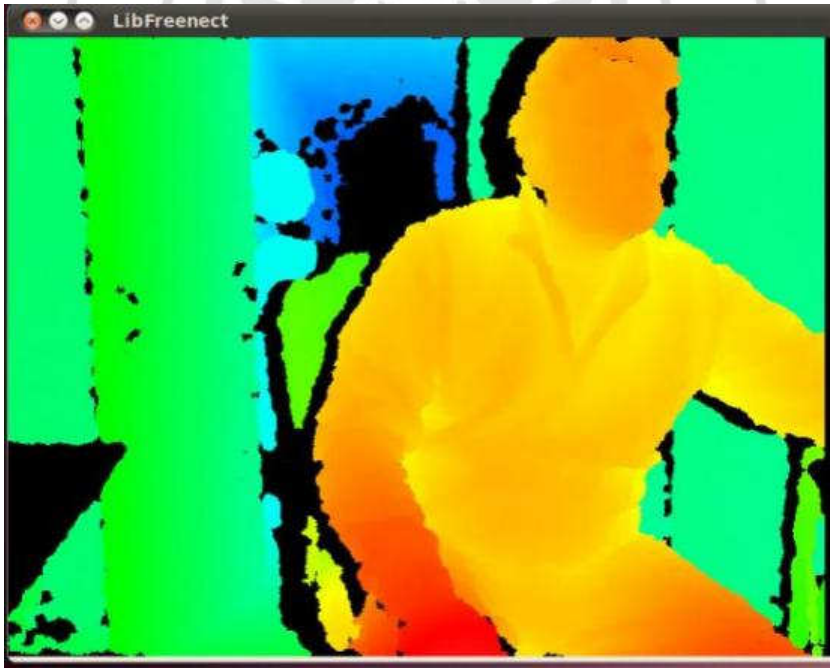


圖 3.6 使用 Libfreenect 擷取到的深度影像資訊
資料來源：http://www.linuxjournal.com/content/kinect-linux

◎OpenNI Framework (Provided by OpenNI Organization)

OpenNI 是為了推廣自然體感這塊領域所成立的一個組織，圖 3.7 為 OpenNI 的官方 Logo。目前組織成員有 PrimeSense、ASUS、Willow Garage、Side-kick、appside。該組織免費提供 OpenNI Framework 這套開放原始碼的應用程式開發框架給應用程式開發者使用，其開發框架提供 C/C++、Java、C#程式語言的支援，且同樣能夠跨平台(Windows、Linux、Mac)。該框架本身提供許多高階功能方便開發者輕鬆撰寫相關應用(例如：手勢辨識等)，同樣的也提供像是原始影像資料的讀取，這類相對來說較為低階的功能(例如：深度影像資訊、彩色影像資訊等串流之讀取)以拓展其應用層面。從圖 3.8 可以發現，開發框架本身還提供中介軟體元件(Middleware)擴充，因此開發廠商可以客製許多自己的特殊應用，像是 NITE(Natural Interaction Technology for End-user)這個中介軟體元件就是 PrimeSense 所提供，用來將這些低階的資料進行加工，進而產生人體骨架資訊的一個擴充，它本身提供了高度抽象化的 API 界面提供開發者使用，像是各個關節(例如:肩膀、手肘、膝蓋等)的座標位置與方向都能輕易的經由該擴充所提供的 API 獲得。圖 3.9 為 NITE 對人體各關節之定義。驅動程式方面，由於該組織並不直接提供相容於開發框架的 Kinect 感測器驅動程式，因此驅動程式的來源基本上是來自第三方開發者，基於 PrimeSense 官方版本並針對 Kinect 感測器進行修改後之版本，再透過 GitHub 分享其原始碼(Driver：SensorKinect，https://github.com/avin2/SensorKinect)。目前該組織所提供的開發框架相當穩定且容易使用，也是目前使用人數相當多的一套開發工具，適合想應用 Kinect 感測器於各式專案的開發者及研究人員。本研究主要也是使用這套開發框架進行 Kinect 應用程式開發。詳細的 OpenNI 相關資料請參閱文獻[22, 23, 24, 25]。

近來國內電腦廠華碩(ASUS)與 Kinect 的感測技術授權廠商 PrimeSense 合作推出兩款與 Kinect 採用同樣感測技術的感測器，分別為 Xtion PRO、Xtion PRO LIVE，因此除了 Kinect 外，現在開發者還多了兩種不一樣的選擇，且除此之外華碩推出的兩款感測器直接相容於 OpenNI 的軟體開發框架、官方驅動程式，因此等於是官方的硬體一樣，所以在支援度上相當良好。

圖 3.7 OpenNI 官方 Logo
資料來源：OpenNI

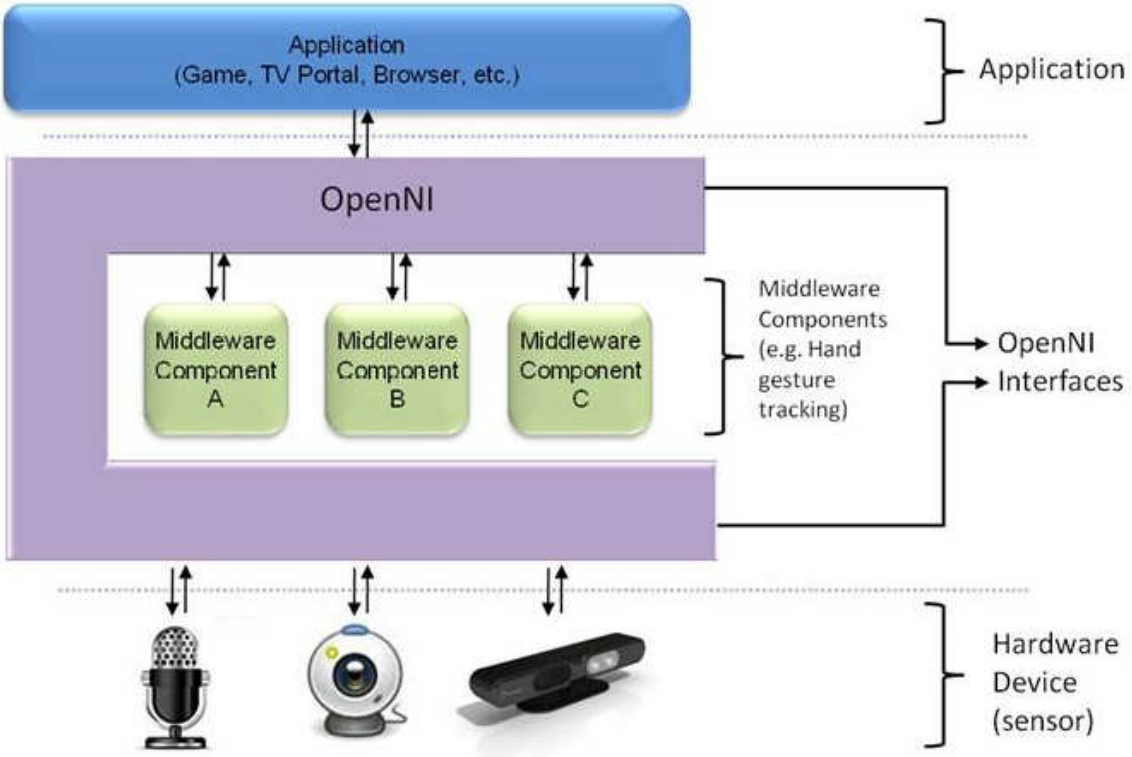


圖 3.8 OpenNI Framework 架構
資料來源："OpenNI User Guide", OpenNI, 2010

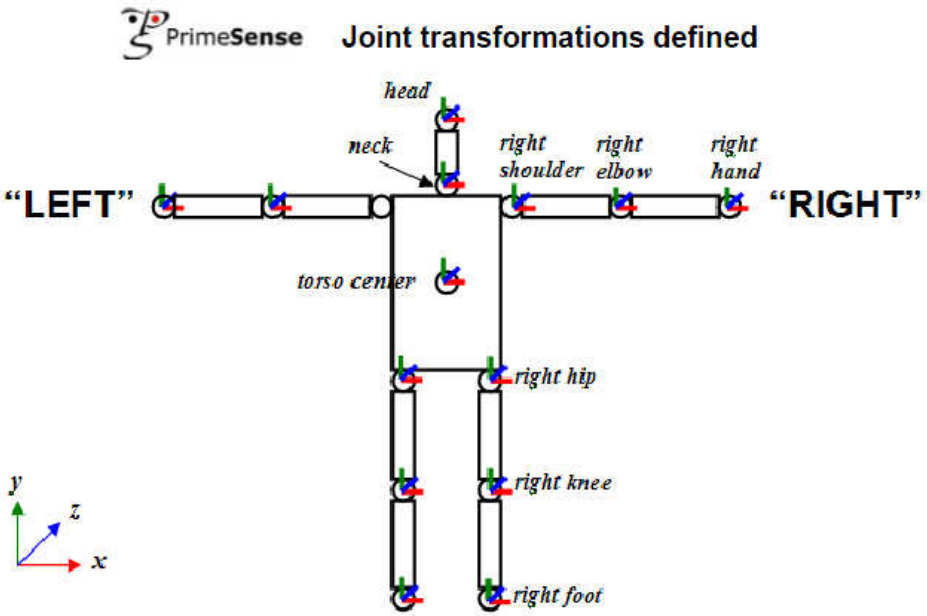


圖 3.9 NITE 關節定義

資料來源："NITE Algorithms 1.3 Note", PrimeSense Inc., 2010

◎Kinect for Windows SDK (Provided by Microsoft Corporation)

Kinect for Windows SDK 是由 Microsoft 官方正式釋出的開發工具，其開發工具本身就內含官方提供的 Kinect 感測器驅動程式，但基本上這套開發工具只能用於微軟自家 Windows 作業系統，且由於未釋出原始碼，若要用於研究領域也較為不方便，而若要使用這組開發工具開發 Kinect 的相關應用的話，開發者的作業系統就必須是 Windows 7(或以上)，且須使用 Microsoft Visual Studio 2010、Microsoft .NET Framework 4.0 才能進行開發。在程式語言的支援方面 Windows 平台的三個主力語言 C++、C#、VisualBasic 都有支援。相較於 OpenNI Framework，Microsoft 官方釋出的開發工具同樣也提供高度抽象化的 API 界面提供開發者使用，且該框架本身就內建類似 NITE 的人體骨架追蹤功能(關節之定義如圖 3.10 所示)，完全不需要再安裝任何擴充。由於作業系統平台的限制，故若想在非微軟的作業系統上開發 Kinect 相關應用則較不適合使用此開發工具。基本上這組開發工

具最大的優勢在於官方的原生支援，由於是 Microsoft 自家的硬體所以在硬體支援度與驅動能力上都較其它開發工具來的優秀許多。

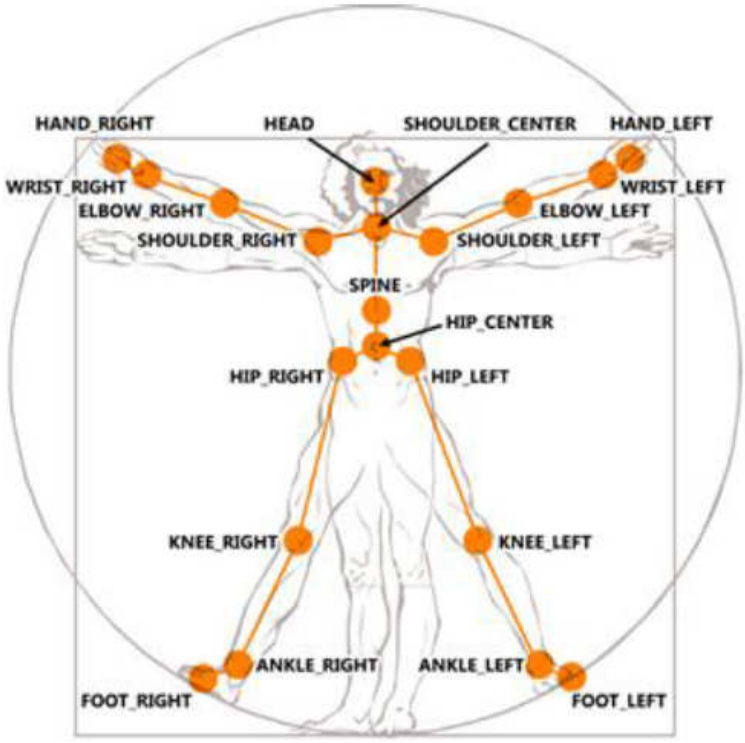


圖 3.10 Kinect for Windows SDK 關節定義

資料來源："Programming Guide : Getting Started with the Kinect for Windows SDK Beta", Microsoft Research, 2011

3.2.4 為何選擇 Kinect

一般來說若想運用非接觸式三維感測技術於自然體感人機介面上，幾乎都是使用兩個彩色攝影機以仿人眼視覺系統的方式來達到三維空間的量測[26]，這樣子的架構下成本相對較為低廉，但卻容易受到亮度、色調、物件特徵等所影響，而其它的感測方式如紅外線結構光(Infrared Structured Light)等量測方式，雖然可以精準且快速的量測出三維空間，但硬體設備一般都較為昂貴，直到 2010 年 11 月

Microsoft 推出 Kinect 感測器才改善了這個狀況。Kinect 感測器可以說是第一個把紅外線結構光感測技術推向大眾消費市場的三維影像感測裝置，由於是使用紅外線結構光感測技術，因此可以改善一般在使用彩色攝影機時容易受到的干擾，且價格也較過去同類型的裝置便宜許多，不失為一個好選擇。雖然 Kinect 感測器是相當新的東西，但其軟體開發環境的資源卻相當豐富，像是前面所提到的 OpenNI Framework、Kinect for Windows SDK、Libfreenect 都是為了 Kinect 而發展出來的軟體開發工具，若搭配上舊有的程式庫像是 OpenCV、OpenGL 等，在體感相關的應用程式開發上可謂相當便利。因此在經過一連串的評估後，本研究決定使用 Kinect 來開發自然體感機械手臂之原型，期望以較佳的硬體性能及低廉的價格打造出此一原型。

3.3 機械手臂

本研究所使用之機械手臂，是採用八顆整合式伺服馬達所組成之四軸機械手臂。整合式伺服馬達主要是使用 ROBOTIS 公司所生產的 Dynamixel AX-12+伺服馬達，來擔任驅動機械手臂的工作，這顆馬達比較特別的地方在於，它將控制電路與驅動電路都放進了伺服馬達內部，因此若要控制這顆伺服馬達，只需透過圖 3.11 的接腳定義跟馬達連接，並傳送指令進去(特定的協定)即可輕鬆的控制伺服馬達、讀取伺服馬達的狀態。馬達硬體規格如表 3.2 所示。在本研究中機械手臂主要是透過電腦來控制，因此在 PC 和伺服馬達之間的通訊上，是使用 USB2Dynamixel 通訊裝置連接，USB2Dynamixel 通訊裝置說穿了只是使用了 FTDI 的晶片，將 USB 介面模擬成 UART Serial 介面的一個裝置，讓使用者可以透過這個裝置輕鬆的跟機械手臂馬達溝通。所以跟許多工業用的機械手臂不同地方在於，本研究使用的機

械手臂並不需要運動控制卡，且馬達的控制與驅動電路為內建，因此在運作時，都是以電腦在計算路徑等各項參數，再透過 USB2Dynamixel 通訊裝置，將指令送進各軸伺服馬達內的控制器，當各軸的伺服馬達控制器接到指令後便會立刻執行指令，以驅動馬達本體做出反應。機械手臂實體及連接方式如圖 3.12 所示。詳細資訊請參考文獻[27, 28]。

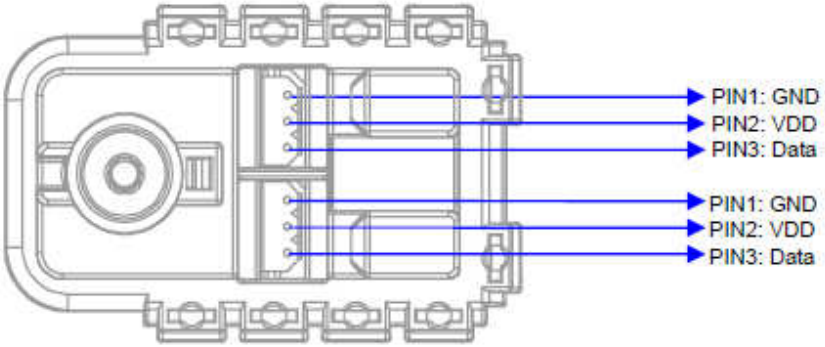


圖 3.11 Dynamixel AX-12+伺服馬達接腳定義
資料來源：ROBOTIS e-Manual v1.08.01

表 3.2 ROBOTIS Dynamixel AX-12+ 硬體規格

ROBOTIS Dynamixel AX-12+	
	
Weight :	53.5g
Dimension :	32mm × 50mm × 40mm
Resolution :	0.29°
Gear Reduction Ratio :	254 : 1
Stall Torque :	15kgf.cm(@12.0V, 1.5A)
No load speed :	59rpm (@12V)

Running Degree :	§ 0° ~ 300° § Endless Turn
Voltage :	9 ~ 12V (Recommended Voltage 11.1V)
Command Signal :	Digital Packet
Protocol Type :	Half duplex Asynchronous Serial Communication (8bit,1stop,No Parity)
Link (Physical) :	TTL Level Multi Drop (daisy chain type Connector)
Communication Speed :	7343bps ~ 1 Mbps
Feedback :	Position, Temperature, Load, Input Voltage, etc.
Material :	Engineering Plastic

資料來源：ROBOTIS e-Manual v1.08.01

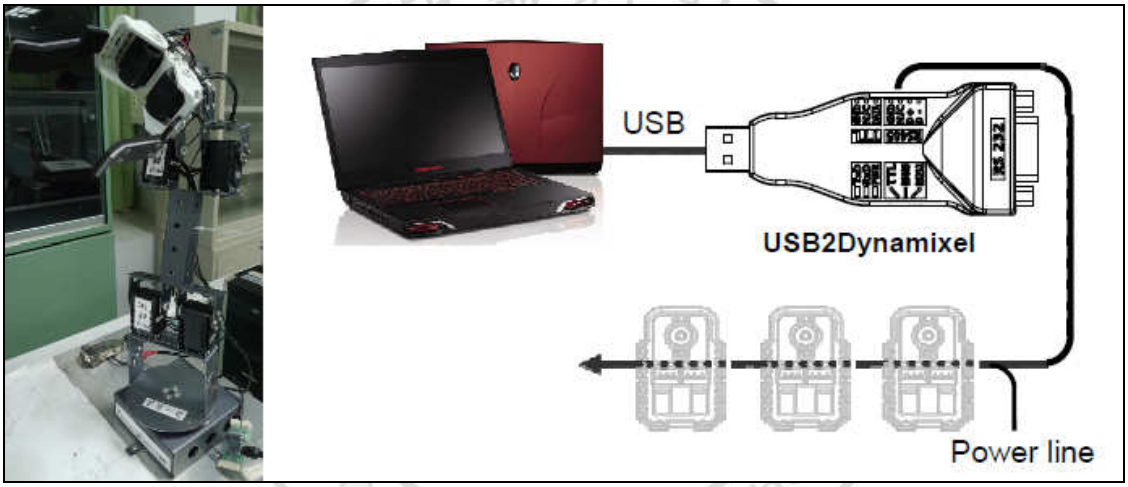


圖 3.12 機械手臂實體及連接方式

3.4 壓力感測器

本研究使用 Tekscan 所生產的可繞曲電阻式壓力感測器(Tekscan FlexiForce Sensors A201 Model, 1lb)來量測使用者手部按壓之力量，以動態控制機械手臂夾爪之夾持力量。壓力感測器硬體規格如表 3.3 所示。

表 3.3 可繞曲電阻式壓力感測器硬體規格

Tekscan FlexiForce Sensors A201 Model (1lb)	
	
Sensor Properties	
Thickness :	0.008 (0.208 mm)
Width :	0.55” (14 mm)
Sensing Area :	0.375” (9.53 mm) diameter
Connector :	3-pin male square pin (center pin is inactive)
Typical Performance	
Force Ranges :	0-1 lb (4.4 N)
Operating Temperature Range :	15°F to 140°F (-9°C to 60°C)
Linearity (Error) :	+/- 3%
Repeatability :	+/- 2.5% of full scale (conditioned sensor, 80% force applied)
Hysteresis :	<4.5% of full scale (conditioned sensor, 80% force applied)
Drift :	<5% per logarithmic time scale (constant load of 90% sensor rating)
Response Time :	<5 microseconds
Output Change/Degree F :	Up to 0.2% (~0.36% / °C) Loads <10 lbs, operating temperature can be increased to 165°F (74°C)

資料來源：Tekscan FlexiForce Sensors User Manual

3.5 資料擷取卡

本研究使用美商國家儀器(National Instruments, NI)所生產的 PCI-6024E 資料擷取卡，來擷取壓力感測器的類比電壓輸出訊號，並將其電壓輸出訊號經過公式轉換以得到對應之壓力值，才能進一步用於機械手臂夾爪之夾持力量控制上。資料擷取卡硬體規格如表 3.4 所示。

表 3.4 資料擷取卡硬體規格

NI PCI-6024E	
	
Analog Input Specification	
Channels :	Single-Ended Channels : 16 Differential Channels : 8
Resolution :	12 bits
Sample Rate :	200 kS/s
Maximum Voltage Range :	-10 V , 10 V
Maximum Voltage Range Accuracy :	16.504 mV
Minimum Voltage Range :	-50 mV , 50 mV
Minimum Voltage Range Accuracy :	0.106 mV
On-Board Memory :	512 samples

資料來源：National Instruments

National Chin-Yi University of Technology

第四章 自然體感機械手臂原型之設計與實作

本章主要探討自然體感機械手臂原型的設計與實作，詳細敘述整個原型平台的各項細節與原理，共分為六大部分：1.系統架構、2.自定義之機械手臂操作方式、3.系統之運作流程、4.機械手臂之設計與實作、5.機械手臂末端夾爪之夾取力量控制、6. Kinect 感測器於手部追蹤時之雜訊抑制。

4.1 系統架構

本研究主要目標將 Kinect 感測器與機械手臂整合，期望發展出一具有自然體感之機械手臂原型，為驗證此一構想，必須先規劃出整個系統架構。由於設定的使用情境是要在惡劣的環境下，因此遠端遙控就必須納入到整個系統架構的考量當中，這樣的設計可以讓操作者免於暴露在危險的環境當中以避免非必要之人員傷亡。如圖 4.1 所示，操作者可以在安全的地方進行機器手臂之遠端遙控，執行如"炸彈移除"等這類危險的工作，而這種主從式的架構分別可以拆成"機械手臂"、"使用者操作端"兩部分來看。

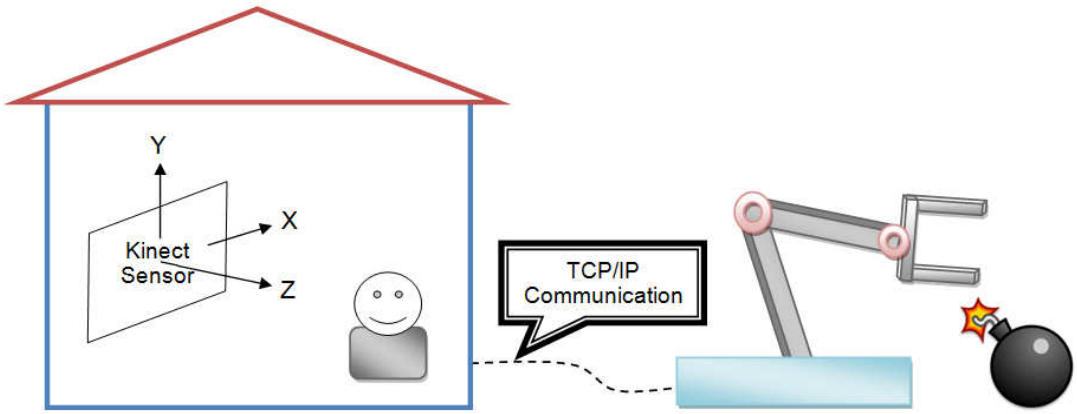


圖 4.1 遠端遙控機器人示意圖

◎遠端機械手臂：

由於想要提高程式的開發效率並專注在功能的開發上，且期望可以在未來的維護與修改上更為便利，因此本研究導入 Ruby 這個程式語言，並以此為主力進行機械手臂的程式開發。圖 4.2 為本研究所設計的機器手臂系統架構圖，其中主要分成主程式(Robot Arm Controller)、程式庫(Library)、直譯器(Ruby Interpreter)三大部分。由於 Ruby 是屬於直譯式的語言，因此完成的應用程式都需要交由直譯器執行。程式庫(Library)也分為兩類，分別是 Ruby 的標準程式庫(Ruby Standard Library)、外部程式庫(External Library)。由於我們需要使用到其它公司所生產的硬體(ROBOTIS AX-12+ Servo Motor)，因此需要特別為這些裝置各別撰寫其程式庫，讓主程式於執行期間動態載入所需的程式庫，以獲得該設備的存取能力。

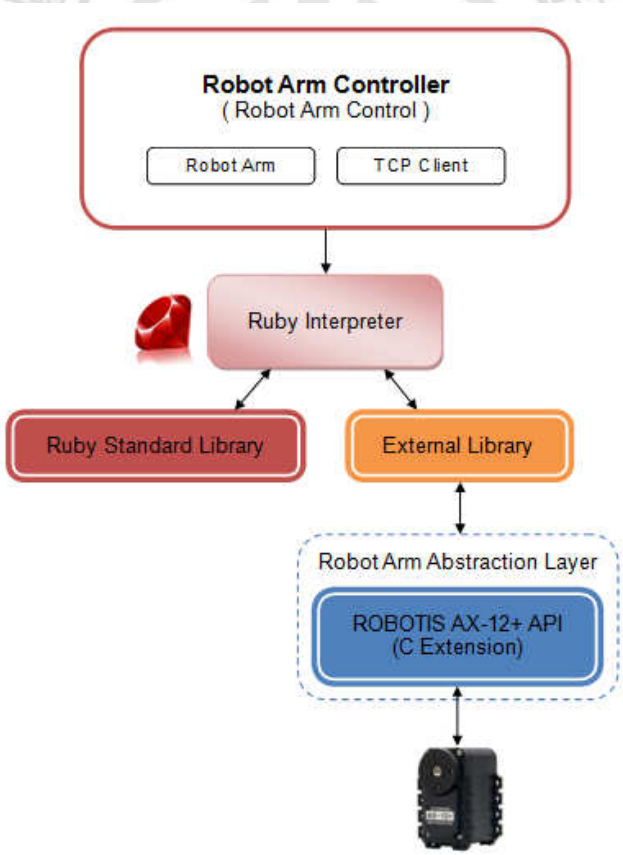


圖 4.2 機器手臂系統架構

◎使用者操作端：

主要分成兩個各自獨立運作的子系統，分別為 Kinect 感測伺服器(Kinect Sensor Server)、壓力感測伺服器(Force Sensor Server)兩部份，其主要工作為擷取使用者動作資訊，並提供給遠端的機械手臂操作使用。分成兩個子系統的主要原因在於，Kinect 感測伺服器需要大量處理影像資料，因此許多部分都需要使用執行效率較高的 C/C++語言來撰寫，以避免系統的反應速度不佳等問題產生，因此 Kinect 感測伺服器程式的開發，比較無法藉由導入 Ruby 來提高程式的開發效率，另一個原因就是分散風險，避免當其中一個系統發生問題時連累其它系統，造成整個系統崩解，因此在經過仔細的評估後，便將 Kinect 感測伺服器獨立出來並單純以 C/C++來撰寫。圖 4.3 為 Kinect 感測伺服器系統架構圖，主要是使用 OpenNI 提供的軟體開發框架、Windows SDK 所提供的 Library 來開發。

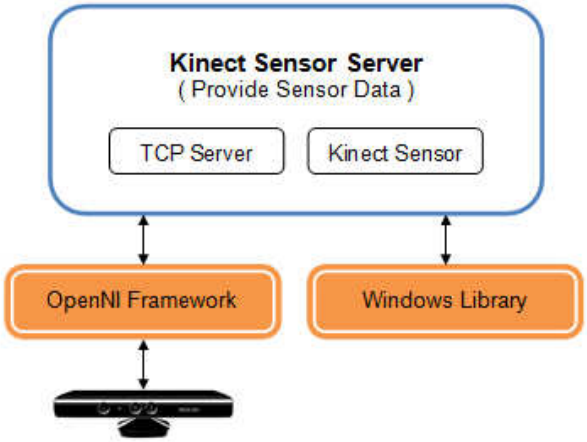


圖 4.3 Kinect 感測伺服器系統架構

而在壓力感測伺服器(Force Sensor Server)的部份，則同樣導入 Ruby 來幫助我們專注在功能上的開發。由於壓力感測器需要連接放大電路將訊號放大，再透過資料擷取卡(NI-DAQ)將放大電路所輸出的類比電壓訊號擷取到電腦內，以方便我

們使用。故基本上如同機械手臂系統架構的形式，同樣需要為我們所使用的資料擷取卡來撰寫外部程式庫，讓主程式於執行期間可以動態載入此程式庫以獲得該設備的存取能力。圖 4.4 為壓力感測伺服器系統架構圖。

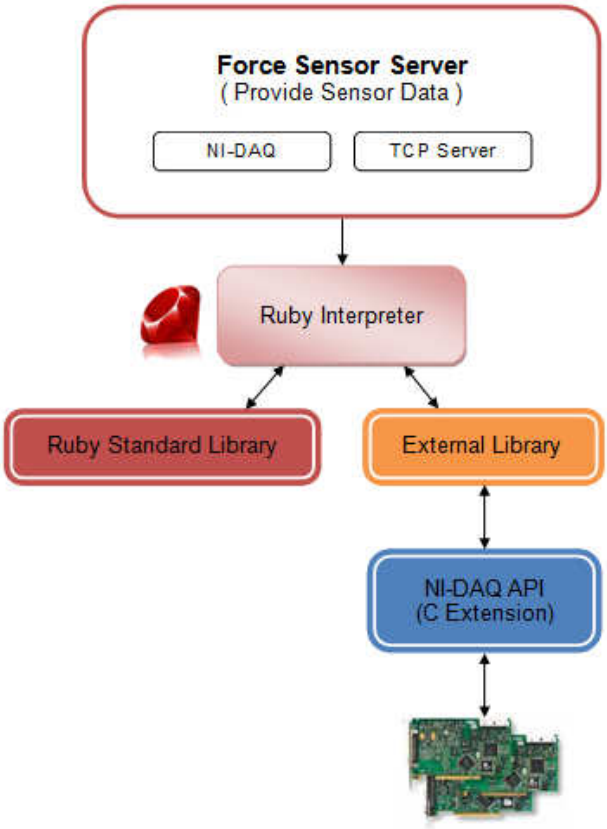


圖 4.4 壓力感測伺服器系統架構

整體自然體感機械手臂系統架構如圖 4.5 所示。透過 Kinect 感測伺服器、壓力感測伺服器，所組成的使用者操控端擷取使用者動作資訊，並將感測資料透過乙太網路(Ethernet)傳至遠端機械手臂，當遠端機械手臂的主程式接收到資料後便會依照其接收到的資料對機械手臂下達指令，進而讓機械手臂做出使用者期望的動作。選擇乙太網路作為實體傳輸介面的主要原因在於，乙太網路的發展相當成熟普遍且可靠度高。在通訊的協定上同樣選用發展相當成熟的 TCP/IP 協定，並基於

此協定傳輸自定義的資料格式做為通訊的基礎。會選用 TCP 協定還有另一個原因，由於 TCP 協定是屬於交握型式且會去確認資料是否送達，因此可以避免資料遺失的狀況發生。

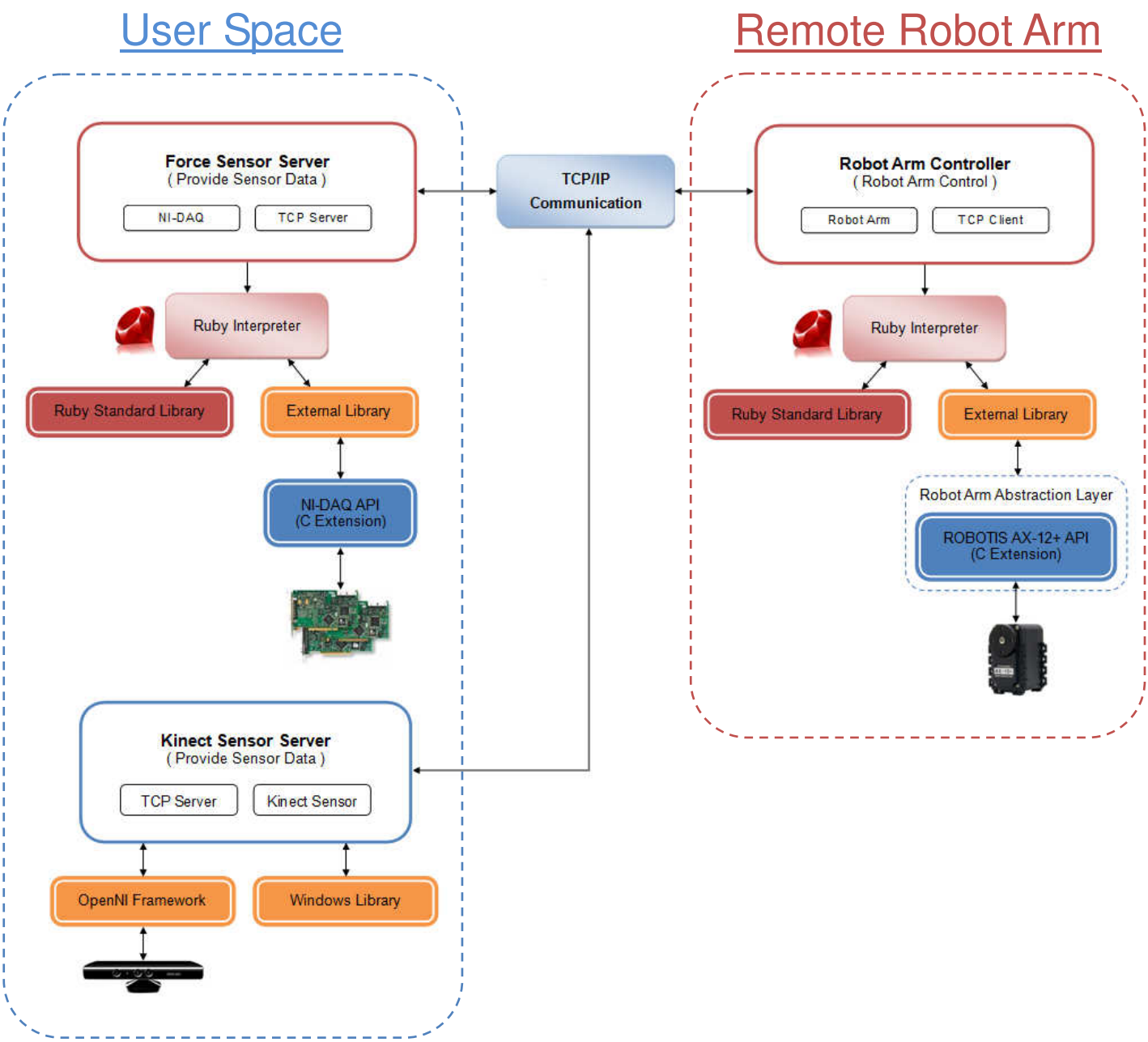
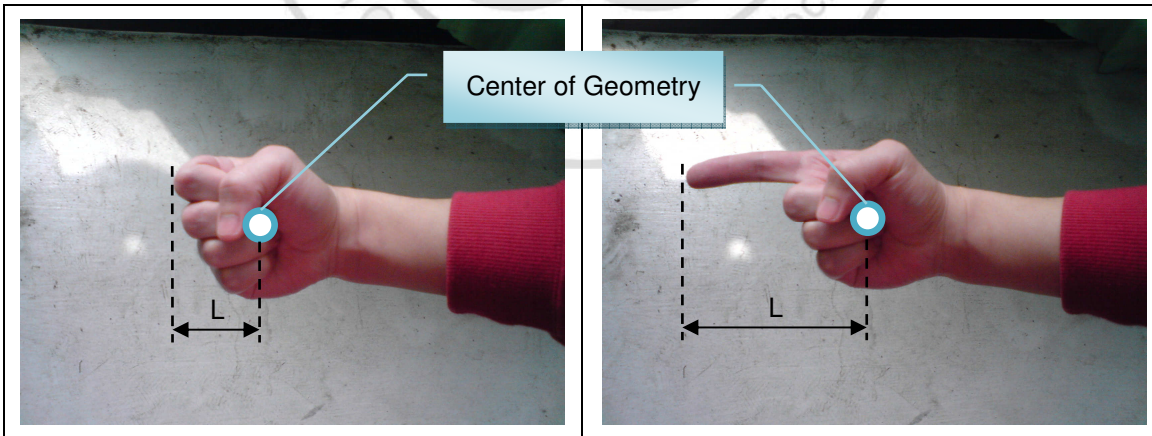


圖 4.5 自然體感機械手臂系統架構

4.2 自定義之機械手臂操作方式

由於本研究是將自然體感運用到機械手臂的控制上，因此為了讓使用者在操作機械手臂時有如使用自己的手一樣方便，故在操作上定義了基本的操作方式，如圖 4.6 所示。因為本研究是使用位置控制的方式在操作機械手臂，所以便利用使用者手掌的形心，來計算使用者手掌於三維空間中的座標位置，並對應到機械手臂夾爪末端端點的座標位置，因此當使用者將手往前移動，機械手臂也會在遠端做出相對應的運動。而夾取物件的操作方式則是依照食指的開合動作來控制機械手臂夾爪之作動。基本上用來判斷機械手臂夾爪開合的方法非常簡單，就是計算手指最前端到手掌形心的距離，當距離(L)小於所訂定的門檻值時即判定為 Close，反之則判定為 Open。當然使用這樣的方法來判斷機械手臂夾爪的開合似乎過於簡陋，若使用者伸出兩支手指以上，一樣會超過門檻值並且被判斷為 Open，但這樣做卻會因為手勢的變化太大導致手掌的幾何形心位置改變較大，而使得機械手臂出現顫抖現象。



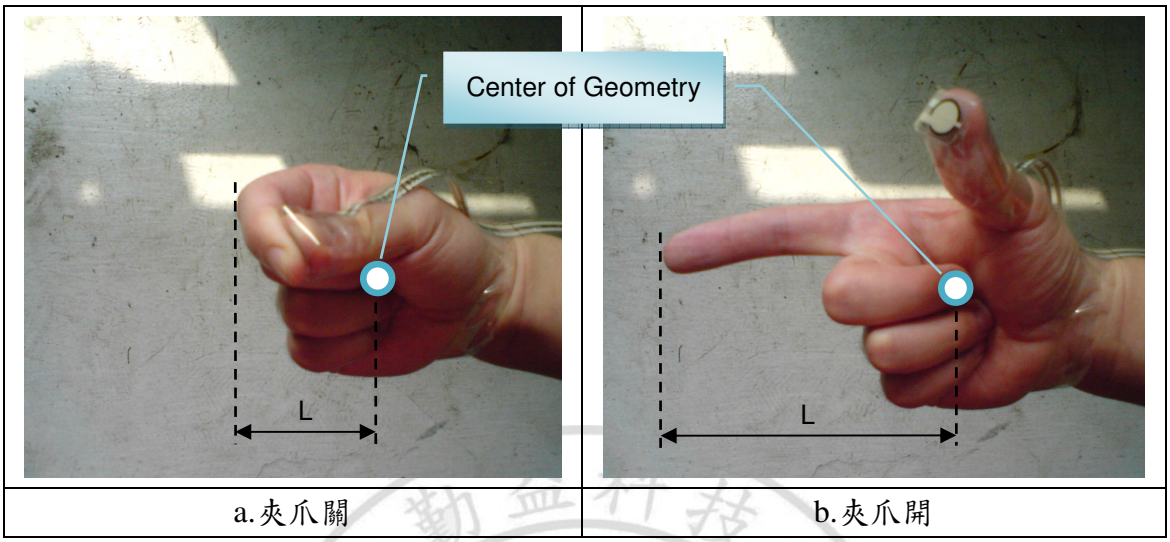
a.夾爪關　　b.夾爪開

圖 4.6 自定義之機械手臂基礎操作方式

人類之所以能夠輕易地拿起一個物件但卻又不會把它捏碎，在於人的手上有許多的感測單元能夠回授給大腦許多資訊像是壓力、溫度等，而大腦在依據這些資訊經過複雜的計算後改變手部肌肉的出力大小，使得物件能夠在被拿起的狀態下卻又不會因為肌肉的出力過大而被捏碎。因此若能控制機器手臂夾爪的夾持力，便能夠在不損及物件的前提下搬移物件。然而要做到機械手臂夾爪之夾取力道與使用者手掌之握力對應的話，便需要在使用者身上穿戴感測器才能擷取到使用者的握力變化，進而將其壓力值對應到機械手臂夾爪之夾持力控制上，藉以控制機械手臂夾爪之夾持力。在此使用的是可繞曲的電阻式壓力感測器，並將其感測器貼至使用者大拇指上方，如圖 4.7 所示。

圖 4.7 使用者配戴壓力感測器之示意圖

在操作上延續前面之操作方式，更進一步加入壓力感測器感測使用者握力大小。在操作時，當使用者合起食指並施壓期望之夾持力大小於壓力感測器上，便能夠動態即時的改變機械手臂末端夾爪的夾持力，如圖 4.8 所示。



a.夾爪關　　b.夾爪開

圖 4.8 自定義之機械手臂基礎操作方式(含壓力感測)

4.3 系統之運作流程

本研究所設計的系統，基本上是將遠端機械手臂規劃為主從架構的用戶端角色。當系統啟動時便會開始連至使用者所在的感測伺服器系統，若連線無法建立便會不斷的重試直到連線建立為止，當連線建立完成後便會開始向感測伺服器系統請求資料以驅動機械手臂作動。若運作過程中連線中斷或是出現錯誤，機械手臂控制系統便會中斷連線並嘗試重新建立連線。因此機械手臂控制系統的工作，基本上就是不斷的請求感測資料並驅動機械手臂作動以達到使用者期望的目標，當連線中斷時則主動嘗試恢復連線。其運作流程如圖 4.9 所示。

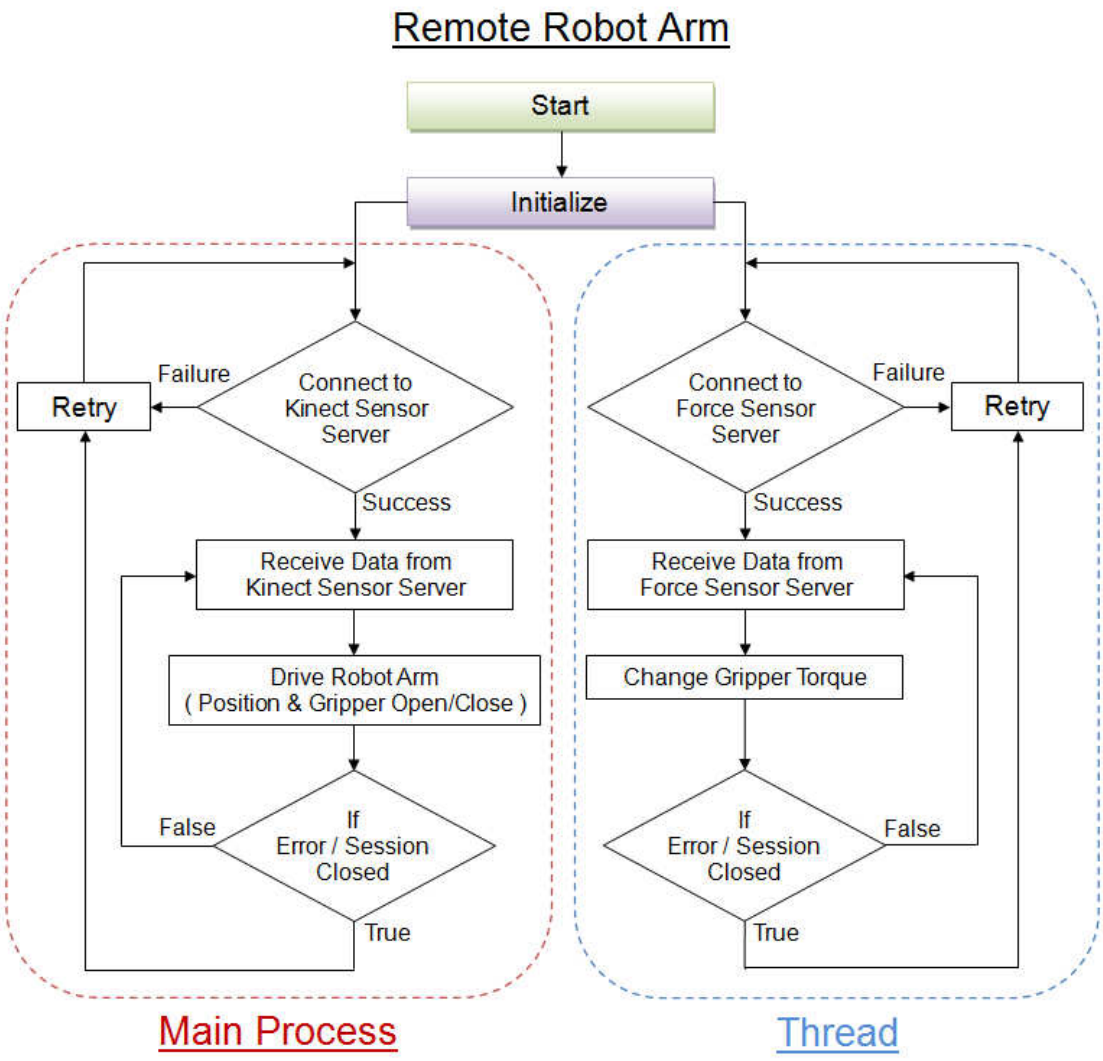


圖 4.9 機械手臂控制系統之運作流程

在使用者端的部分，則是將使用者端的感測伺服器系統規劃成主從架構的伺服端角色。且在規劃時便將每個感測器各自獨立開來，並各自擁有專屬獨立運作的伺服器且互不相干擾。這樣的做法有個好處，就是當其中一個感測伺服器當機時，不會影響到其它感測伺服器的運作，如此便可提高系統整體的可靠度。而在運作流程方面，當感測伺服器啟動時便會開始初始化，初始化結束後便會建立 TCP/IP 伺服器等待用戶端(遠端機械手臂)主動連線，當連線建立後便會依照用戶端

的資料請求將感測資料傳送至用戶端，過程中若發生連線中斷或是錯誤便會主動中斷連線並告知使用者，等待使用者排除後便可重新啟動系統恢復連線。其運作流程如圖 4.10 所示。

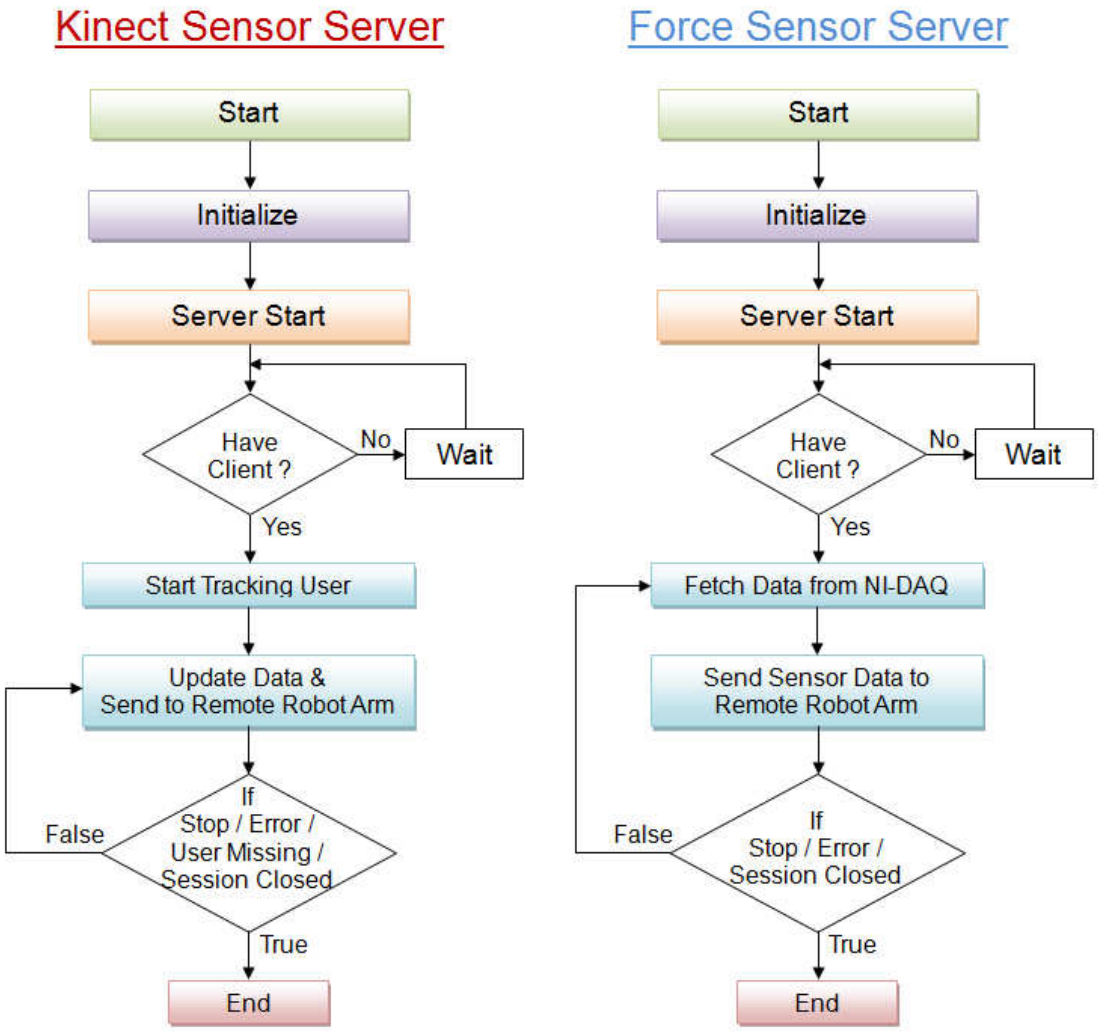


圖 4.10 感測伺服器系統之運作流程

4.4 機械手臂之設計與實作

由於電腦產業的蓬勃發展，個人電腦的計算能力也越來越強，相關軟體資源

也越來越豐富，故許多運動控制系統開始從 PLC-Based 轉向彈性更高的 PC-Based 架構，由於 PC-Based 架構可以結合 PC 的龐大軟硬體資源，來開發更具彈性的控制系統，因此近幾年來使用率越來越高。在 PC-Based 的運動控制架構裡，除了 PC 與作業系統外，另外還包括了三個相當重要的部份，依序為 1.運動控制卡(Motion Control Card)、2.馬達驅動器(Driver)、3.即時子系統(Real-Time Subsystem)，其中即時子系統的主要目的在於提高系統的即時性，由於我們常用的作業系統如 Windows、Linux、Mac OS、FreeBSD 等，幾乎都是以通用的使用目的為主，並非基於高精度控制而設計，故即時子系統可以確保程式的運作上可以達到硬即時(Hard Real-Time)的嚴苛需求，圖 4.11 為美國 IntervalZero 公司基於 Windows 作業系統所開的 RTX 硬即時平台。

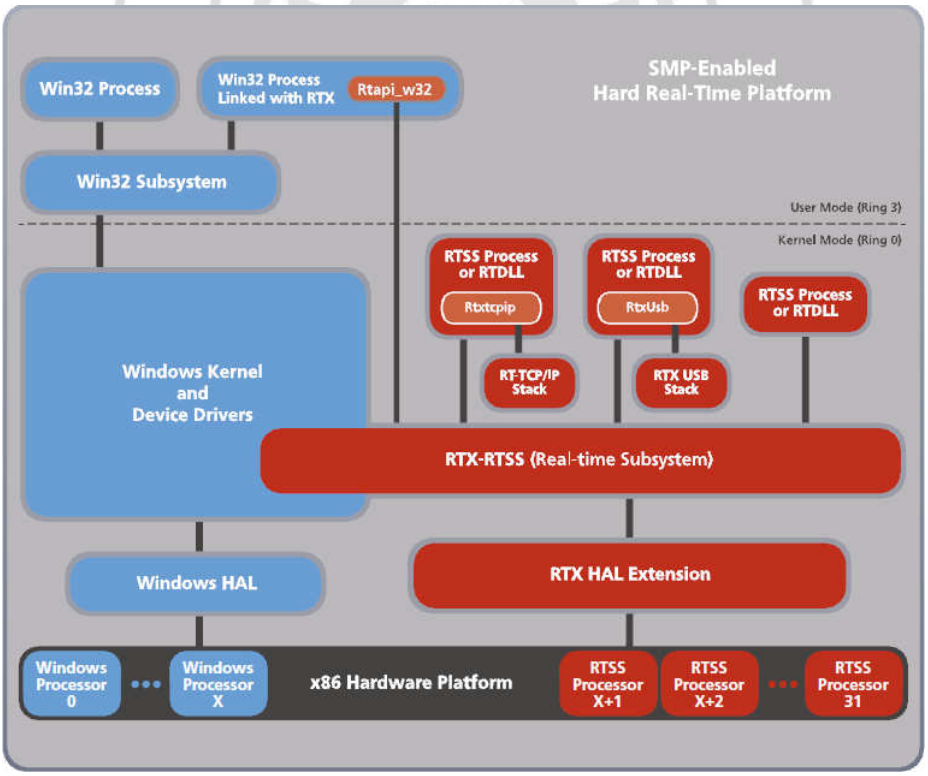


圖 4.11 IntervalZero RTX 硬即時平台架構

資料來源：IntervalZero

而運動控制卡其主要工作就是在作平台的運動控制，若以一般工業用機械手臂來說，便是控制與規劃機械手臂之路徑等各項參數。在多軸的運動平台中，運動控制卡的多軸同動能力與精度等，都是相當重要的評估指標。但在我們所要建立的原型平台當中，由於機械手臂是使用多顆整合式伺服馬達(ROBOTIS AX-12+)所組成，而非使用工業用的高精度機械手臂，故在功能上跟許多工業用機械手臂已內建之功能相比遜色不少，舉例來說，若想使用類似的機能，例如:在空間中取兩點並內插走直線等，都要自己用純軟體實作，由於受限於硬體機能，因此實作出來的功能，在許多方面都不如硬體實作來的快且穩定。雖然其性能被受限制但也因為其價格便宜且耐撞擊，故用來製作原型恰到好處。基本上製作此一機械手臂最大的困難，在於求得機械手臂末端點的座標位置與機械手臂各軸之間的轉換關係、控制程式的設計開發，本節將針對這兩個部份來進行解說。圖 4.12 為所使用的機械手臂 3D 立體模型及其各軸之座標關係。

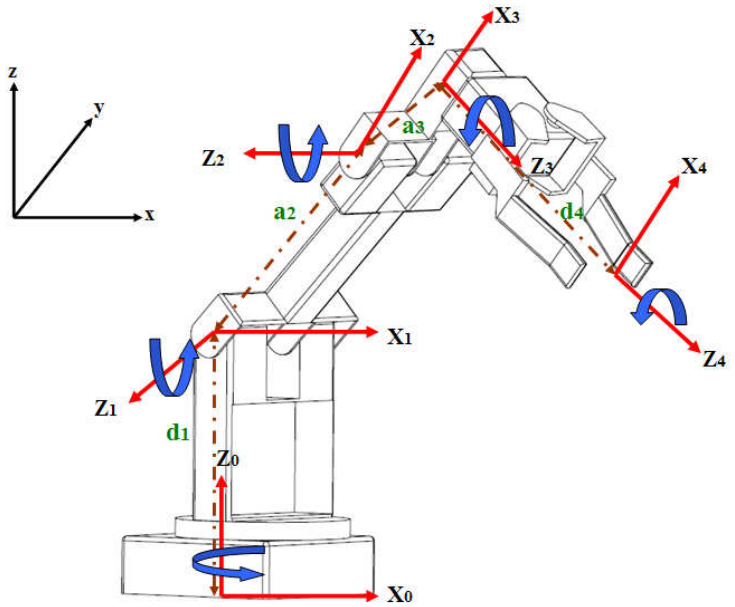


圖 4.12 機械手臂 3D 立體模型及其各軸之座標關係

資料來源：文獻[29]

4.4.1 逆向運動學

若想要在笛卡爾座標系統(Cartesian Coordinate Systems)中控制機械手臂，要求其在座標空間中任意兩點間內插走直線，則在兩點間之路徑上機械手臂各個軸的旋轉角度都將會有所不同，若想求出各軸之旋轉角度，這時就可以使用逆向運動學(Inverse Kinematics)來求出在路徑上任一點時機械手臂各軸的旋轉角度，以達到位置控制的目的。

一般求解逆向運動學的方式分為封閉解(Closed Form Solution)、數值解(Numerical Solution)兩種，而本研究使用的是封閉解的方式，來求得機械手臂末端點的座標位置與機械手臂各軸之間的轉換關係。在求逆向運動學的解之前都會先使用 D-H(Denavit-Hartenberg)的方法，先求得機械手臂各軸間的座標轉換關係，進而得到順向運動學(Forward Kinematics)的解，再利用順向運動學的解進一步求得逆向運動學的解。順向與逆向運動學之關係如圖 4.13 所示。

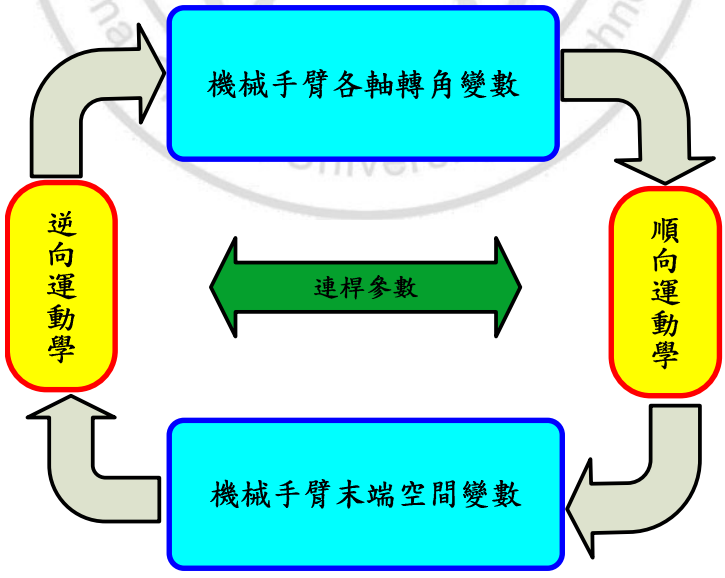


圖 4.13 順向與逆向運動學之關係
資料來源：文獻[29]

經過推導後可以得到各軸之逆向運動學的解(詳細的推導過程請參閱文獻[29])，其關係如公式(1)至公式(3)所示。當中 P_x、P_y、P_z 為機械手臂末端點的絕對座標位置(座標 4)，其座標原點則位於底座的座標上(座標 0)，而 d_1、a_2、a_3、d_4 為機械手臂各連桿之長度參數，θ_1、θ_2、θ_3 則分別對應到座標 0、座標 1、座標 2 上的旋轉軸，θ_4 (座標 3 上的旋轉軸)的部分由於只是末端夾爪的旋轉軸故與位置無關。機械手臂各軸之座標位置及各連桿之長度參數位置如圖 4.12 所示。

$$\theta_1 = \operatorname{atan2}(P_y, P_x) \tag{1}$$

$$\theta_3 = \sin^{-1}\left(\frac{\left(\frac{P_y}{\sin\theta_1}\right)^2 + (P_z - d_1)^2 - (a_2^2 + a_3^2 + d_4^2)}{2a_2\sqrt{(a_3^2 + d_4^2)}}\right) - \tan^{-1}\left(\frac{a_3}{d_4}\right) \tag{2}$$

$$\theta_2 = \sin^{-1}\left(\frac{P_z - d_1}{\sqrt{(m_1^2 + m_2^2)}}\right) - \operatorname{atan2}(m_2, m_1) \tag{3}$$

$$m_1 = a_2 + a_3\cos\theta_3 + d_4\sin\theta_3$$
$$m_2 = a_3\sin\theta_3 - d_4\cos\theta_3$$

當得到逆向運動學的解，便可將它用在機械手臂的位置控制上。我們只需要告訴機械手臂我們想要到達的位置，透過逆向運動學就可以得到該座標位置與各軸之對應關係，而機械手臂便會依據此角度移動到使用者期望之座標位置。

4.4.2 機械手臂抽象層 API 設計

機械手臂系統架構如 4-1 節圖 4.2 所示，若從程式設計的觀點來看，基本上主

要可以拆成三層，由上而下依序為機械手臂控制系統主程式、機械手臂抽象層、控制馬達用的 Ruby C 擴充，其關係如圖 4.14 所示。其中最底層的 C 擴充基本上只是將控制馬達用的 C API 做移植的動作，讓 Ruby 能夠獲得控制機械手臂各軸馬達的能力，並且再透過機械手臂抽象層把它包裝起來變成一組高度抽象化的程式庫，讓上層的機械手臂控制系統主程式可以直接使用機械手臂抽象層的 API，而不必直接存取到底層的馬達驅動介面。會把 C 擴充與抽象層分開的原因在於，抽象層內的實作在未來因應各種不同的需求是有可能被大幅變動的，因此若將抽象層直接用 C 擴充的方式來實作，雖然在執行速度上會獲得大幅度的提升，但在開發與維護上就會變得較為繁瑣，故在機械手臂原型的開發過程當中用 Ruby 來撰寫抽象層是較為經濟的選擇，且剛好控制馬達用的 C API 並沒有很多，只有 21 個函式、25 個常數，故在移植上便簡單許多。

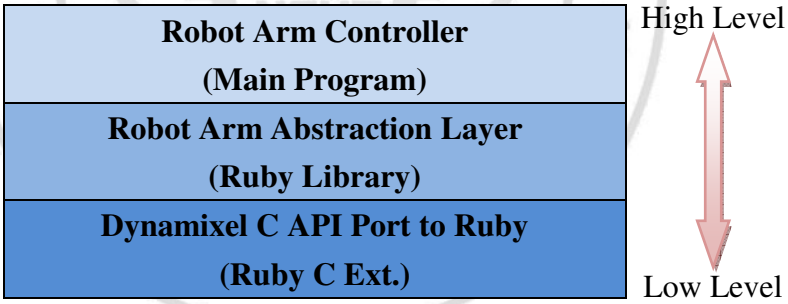


圖 4.14 機械手臂控制系統程式架構

在設計機械手臂抽象層的應用程式介面時，共定義了 16 個方法用以操作、初始化、設定機械手臂等，而這些方法是基於目前會使用到的一些機能所訂定出來的，因此未來若有新的需求只需在舊有的方法上再新增進去即可。其抽象層的 API 如表 4.1 所示。其中比較特別的是 Init. Function 裡的 initialize、init_robot_arm 這兩個方法，因為它們是私人方法(Private Methods)所以沒辦法透過實體物件直接調

用，而是需要透過 new 這個類別方法(Class Method/ Constructor)在建立物件時於程式內部所調用的私人方法。因此在調用順序上，首先是由 new 這個類別方法建立物件並自動調用 initialize 方法用於初始化物件，而後 initialize 方法會再調用 init_robot_arm 方法來初始化機械手臂，最後 initialize 方法會傳回初始化完成後的物件，這樣機械手臂的實體物件便建置完成了。

表 4.1 Robot Arm Abstraction Layer APIs

Robot Arm Abstraction Layer APIs	
# Init. Function (Private Methods)	⇨ initialize(robot_arm_config_data) ⇨ init_robot_arm
# Moving Function	⇨ calculate_inverse_kinematics(p_x, p_y, p_z) ⇨ drive_robot_arm(theta1, theta2, theta3) ⇨ mv_to(p_x, p_y, p_z)
# Gripper Function	⇨ rotate_gripper(theta4) ⇨ mv_gripper(distance) ⇨ gripper_open ⇨ gripper_close ⇨ gripper_stop ⇨ gripper_moving?
# Configure Function	⇨ ch_driver_speed(axis1_speed, axis2_speed, axis3_speed) ⇨ ch_rotate_gripper_speed(rotate_gripper_speed) ⇨ ch_gripper_speed(gripper_speed) ⇨ ch_gripper_torque(percentage_of_torque)
# Close Function	⇨ terminate

在使用上將會如圖 4.15 的範例程式碼所示。首先打上 require "AX-12 Robot Arm"告訴程式我們要使用這組程式庫，再使用 new 這個類別方法建立機械手臂的實體物件，隨後便可使用像是 mv_to、gripper_open、rotate_gripper 等相關方法來操作機械手臂作動。由此可以發現，設計抽象層的目就是隱藏實作細節並提供更

高度抽象化的介面，以利上層的程式可以寫出更容易維護的程式碼，而不會被一堆的實作細節給淹沒，因此抽象層介面設計的好壞便會直接影響到程式碼的可讀性及易用性，故在這方面也是本研究在設計這組 API 時所考慮的重點之一。機械手臂抽象層 API 的實作細節請參閱附錄五。

Example Code (Ruby)

```
require "AX-12 Robot Arm"

my_robot_arm = AX12_Robot_Arm.new()    # Create and Init. Robot Arm Instance
my_robot_arm.mv_to(10, 20, 30)    # Move to (X=10, Y=20, Z=30)
my_robot_arm.gripper_open
sleep(1)    # 1sec.
my_robot_arm.gripper_stop if my_robot_arm.gripper_moving?
```

圖 4.15 機械手臂抽象層 API 之使用範例程式碼

勤益 NCUT National Chin-Yi University of Technology

4.5 機械手臂末端夾爪之夾取力量控制

人之所以能如此自如的用手拿起任何物件，卻又不至於損壞物件的原因在於人的手上有許多的感測單元，可以幫助我們感覺到壓力、溫度等，並且將這些資訊透過神經送至大腦分析，再依據分析的結果驅動肌肉做出反應，因此人們便可動態的改變手的握力，避免力道太大或太小而傷到物件。而若想透過類似的方法來控制機械手臂夾爪的夾持力道，其中一種方法便是在人的手臂上貼上感測用的表面電極貼片用以量測肌肉訊號，另一種則是在人的手掌上貼上壓力感測器用以捕捉壓力大小，如圖 4.7 所示，本研究使用的是後者來感測使用者的握力，並且將此壓力值與機械手臂夾爪驅動馬達的扭矩值對應，進而透過動態的改變夾爪驅動馬達的扭矩值來控制機械手臂夾爪的夾持力量。

本研究所採用的壓力感測器，其訊號擷取流程流程如圖 4.16 所示。資料擷取卡使用美商國家儀器(NI)的 PCI-6024E 資料擷取卡。壓力感測器則是使用 Tekscan 所生產的電阻式可繞曲薄膜型壓力感測器(A201 Model, Force Ranges : 0~1 lb [0~453 g])。

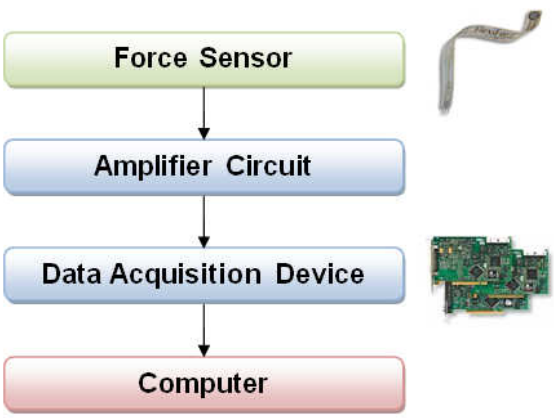


圖 4.16 壓力感測訊號擷取流程

4.5.1 壓力感測訊號放大器

本研究所使用的壓力感測訊號放大電路如圖 4.17 所示，基本上它是使用反向運算放大器(Inverting Operational Amplifier)的配置來產生類比電壓訊號的輸出，在此使用 Microchip 所生產的 MCP6004 運算放大器來製作。MCP6004 是 4 顆放大器包裝的版本，因此在實作上一顆 MCP6004 放大器便可接上 4 個壓力感測器。實作出來的壓力感測放大器如圖 4.18 所示。感測電路的靈敏度可以藉由調整 R_F 的電阻值來控制。

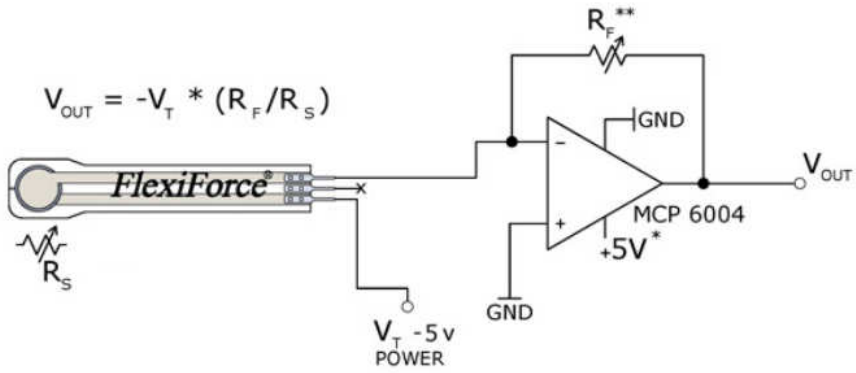


圖 4.17 壓力感測訊號放大器電路

資料來源：Tekscan

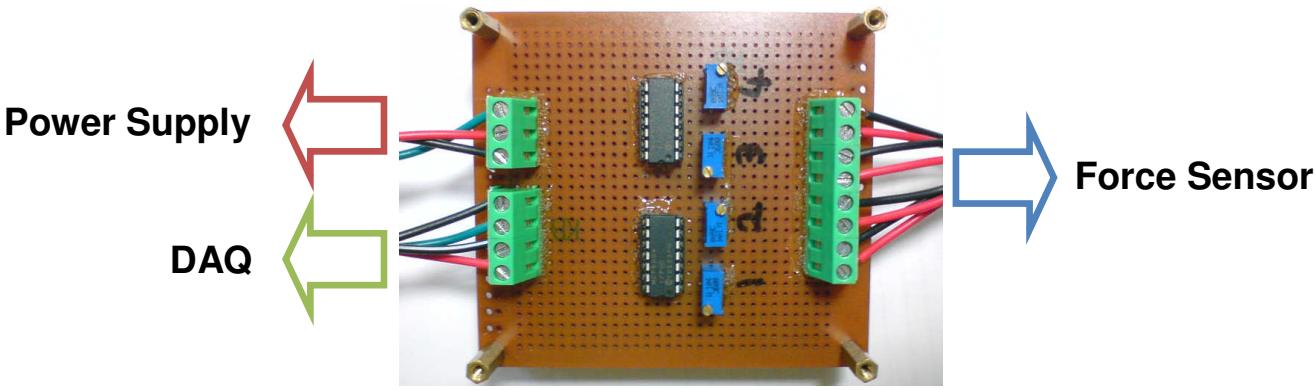


圖 4.18 壓力感測訊號放大器實體圖

4.5.2 電阻式壓力感測器校正

透過實作出來的壓力感測訊號放大器，分別連接上資料擷取卡(DAQ)、電源供應器、壓力感測器，即可在電腦上量測不同壓力時的電壓變化，量測的結果如圖 4.19 所示。先將量測輸出範圍預先固定在 0V ~ 4.5V 之間，並透過調整 R_F 的電阻值將滿載時(300g)的壓力感測輸出設定在 4.5V，再使用不同重量的砝碼施壓在壓力感測器末端的感測圓點上，並記錄不同壓力時的感測器電壓輸出。從圖 4.19 的量測結果可以看到，其壓力與電壓輸出間的對應關係並沒有非常的線性而是些微呈現二次曲線的型式，且經過重複量測後也發現資料點本身的重現性並沒有非常的精確而是稍微有些許偏差。為解決這個問題，本研究使用了二次多項式迴歸(Second-Order Polynomial Regression)，並搭配最小平方法來最小化資料點與曲線之間差異的平方和，以求出合乎資料點的二次多項式曲線方程式，並以此方程式來轉換壓力與電壓輸出間的對應關係。

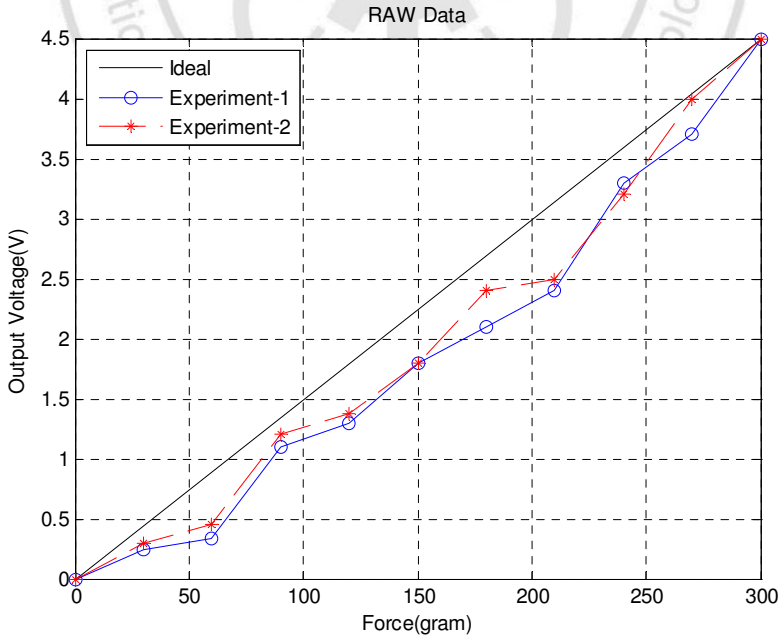


圖 4.19 壓力感測器之壓力與電壓輸出對應圖

公式(4)~公式(13)為二次多項式回歸的推導過程，其中 e 為剩餘值(Residual)，x、y 則分別為 x 軸和 y 軸的資料，a_0、a_1、a_2 則為二次多項式的各項係數為所要求的解。

$$y = a_0 + a_1 x + a_2 x^2 + e \tag{4}$$

$$S_r = \sum_{i=1}^{n} e^2 = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)^2 \tag{5}$$

$$\begin{cases} \dfrac{\partial S_r}{\partial a_0} = -2 \displaystyle\sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right) & (6) \\ \dfrac{\partial S_r}{\partial a_1} = -2 \displaystyle\sum_{i=1}^{n} x_i \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right) & (7) \\ \dfrac{\partial S_r}{\partial a_2} = -2 \displaystyle\sum_{i=1}^{n} x_i^2 \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right) & (8) \end{cases}$$

$$\begin{cases} 0 = \displaystyle\sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right) & (9) \\ 0 = \displaystyle\sum_{i=1}^{n} x_i \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right) & (10) \\ 0 = \displaystyle\sum_{i=1}^{n} x_i^2 \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right) & (11) \end{cases}$$

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix} \quad (12)$$

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix}^{-1} \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix} \quad (13)$$

從公式(9)、(10)、(11)可以發現這是一組三元一次的聯立線性方程式，因此可以透過公式(13)將表 4.2 感測器所量測到的壓力值(y)與對應的電壓輸出(x)帶入，來求得這組聯立方程式的解。

表 4.2 壓力感測器之壓力與電壓輸出對應表

	Experiment-1	Experiment-2
Force (gram)	Output Voltage (V)	
0	0	0
30	0.25	0.3
60	0.33	0.46
90	1.1	1.2
120	1.3	1.37
150	1.8	1.8
180	2.1	2.4
210	2.4	2.5
240	3.3	3.2
270	3.7	4
300	4.5	4.5

最後在將求得的解 a_0、a_1、a_2 帶入到公式(14)即可得到公式(15)，由於式中的 x 代表電壓輸出，y 則代表對應的壓力值，故可以再把公式(15)進一步寫成公式(16)的形式。若將公式(16)帶入電壓值並畫出來，則可得到如圖 4.20 的二次曲線，可以

發現配適效果相當的好。

$$y = a_0 + a_1 x + a_2 x^2 \tag{14}$$

$$y = 6.9099 + 92.0557x - 6.0417x^2 \tag{15}$$

$$\text{force(g)} = 6.9099 + 92.0557 \times \text{Voltage} - 6.0417 \times \text{Voltage}^2 \tag{16}$$

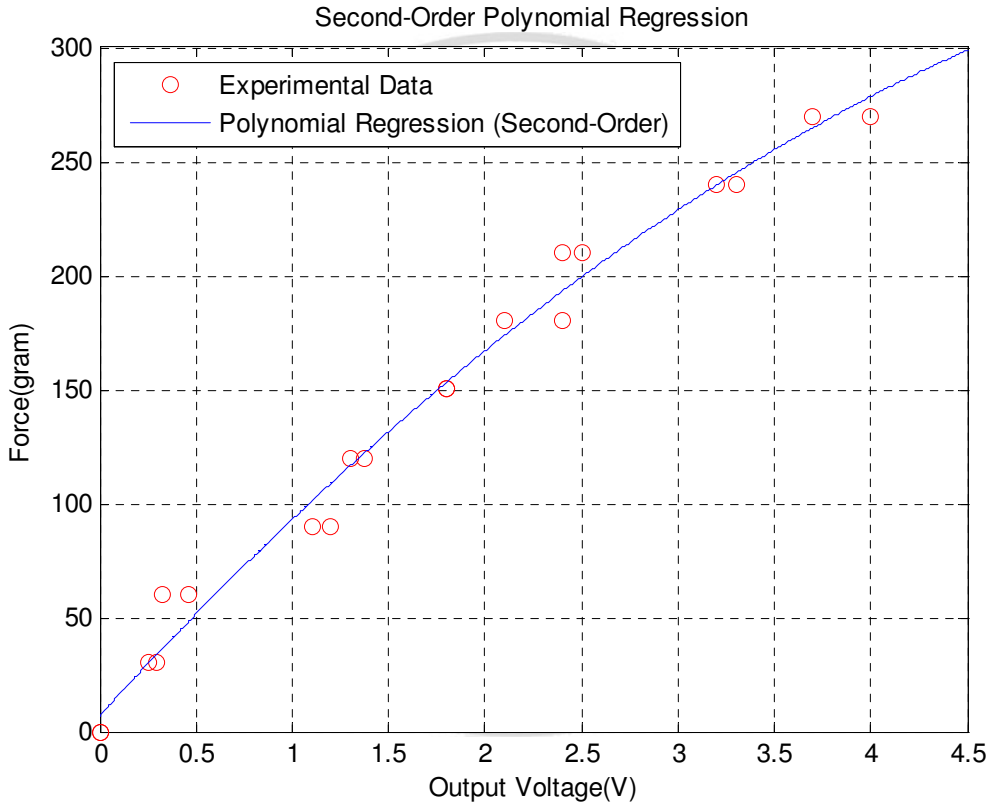


圖 4.20 二次多項式回歸之感測器壓力與電壓輸出對應圖

4.5.3 資料擷取卡的 Ruby C 擴充 API 設計

在設計資料擷取卡的擴充時與機械手臂抽象層比較不一樣的地方在於，資料擷取卡的擴充是直接將抽象層實作在 C 擴充內，如圖 4.21 所示。會這麼做的原因

在於資料擷取卡本身提供的 C API 功能相當豐富，但在此卻只會使用到其中的幾個功能，且在未來內部程式實作也不會有什麼太大的改變，故此時若將資料擷取卡的 C API 全數移植到 Ruby 上就會變得非常不經濟，但若只移植當前需要的功能並透過 Ruby 撰寫抽象層則程式碼的數量也不是很多，跟機械手臂的抽象層比起來程式碼的數量也少上許多，因此若仿造機械手臂系統的程式架構便沒有太大的意義。有鑑於此，在實作上便將抽象層直接寫在 C 擴充內部，這樣的好處是不必在額外做資料擷取卡 C API 的移植工作，且執行效率也會有顯著的提升，維護上也比較容易。

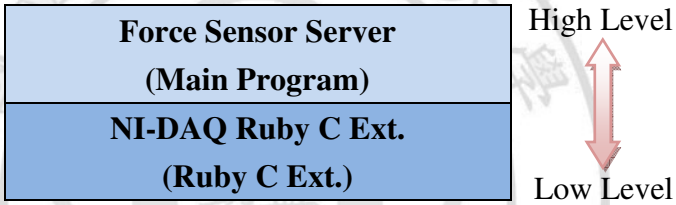


圖 4.21 壓力感測伺服器系統程式架構

在設計資料擷取卡擴充的應用程式介面時，一共只定義了 4 個簡單的方法用以操作、初始化資料擷取卡，方法如表 4.3 所示。在此並沒有定義 close 方法來釋放資源，因為在整個壓力感測伺服器的運作過程當中，並不需要關閉或重新啟動資料擷取卡，且當壓力感測伺服器關閉時，資源會自動被作業系統回收，因此是否有定義 close 方法便不是這麼的重要。但在此其實還是有將釋放資源的部分定義在 C 擴充內，不過只能透過 Ruby 的 GC(Garbage Collector)來進行回收，無法透過使用者自行釋放。資料擷取卡的 Ruby C 擴充使用範例如圖 4.22 所示。

表 4.3 NI-DAQ Ruby C Extension APIs

NI-DAQ Ruby C Extension APIs	
# Init. Function	⇨ initialize(inputChannel, minVoltage, maxVoltage)

# Task Ctrl. Function	⇨ start_task ⇨ stop_task
# Get Function	⇨ get_ai_data

Example Code (Ruby)

```
require "NIDAQ"

daq = NIDAQ.new("Dev1/ai0", 0, 5)   # Create and Init. DAQ Instance (0~5V)
daq.start_task   # Start Task
daq.get_ai_data   # Get Analog Data => Return Value : Voltage
daq.stop_task   # Stop Task
```

圖 4.22 資料擷取卡的 Ruby C 擴充使用範例程式碼

4.6 Kinect 感測器於手部追蹤時之雜訊抑制

在影像處理的領域當中有相當多的方法可以用來抑制雜訊對影像的影響，一般大多是將整個影像畫面進行去除雜訊的處理(例如:平滑法、中值法、柱狀圖等化法等)，但不管演算法在怎麼先進都還是會有些許的雜訊殘留在影像內，因此去雜訊真正只是降低雜訊對影像的影響而非完全去除，因為要完全去除雜訊實在是一件非常不容易的事。本研究之應用與一般影像處理較為不同的地方在於，要抑制的對象是圖 4.6 所示之幾何形狀中心的三維位置，與控制夾爪開關的長度 L，因此是對特定資料作雜訊抑制而非影像本身，雖然對象不同但目標一致。

由於 OpenNI 本身提供手部追蹤的功能讓我們可以輕鬆的抓取到使用者的手部位置，使用上雖然相當方便但還是有些不完美的地方，像是雜訊，由於雜訊會使輸出的手部位置不穩定的跳動，故無法直接用於機械手臂的位置控制上，因為這樣做會使得機械手臂本身因為雜訊的關係不斷抖動，進而影響其動作的精確度與正確性，故在此藉由移動平均的概念提出一簡單平滑法(Simple Smoothing

Method)來解決雜訊問題。

簡單平滑法就是運用平均值的概念，以數學來表示如公式(17)所示，其中 d 代表資料，i 則代表最新資料之索引號碼，其索引號碼會隨著新資料進來而自動加 1，而這當中可以調整的參數就是 n，其代表一次要使用多少資料來做平滑化，n 若設的太大反應速度便會降低，太小則又會不夠平滑但卻會提高反應速度，因此 n 的值會依據使用目的不同而有所差異。本研究將 n 值設定為 6，這是依據機械手臂的響應所調整出來的參數。

$$d_{i\_filter} = \sum_{x=i-n+1}^{i} \left(\frac{d_x}{n}\right) = \frac{(d_i + \cdots + d_{i-n+1})}{n} \tag{17}$$

$i \geq 1$, (i : Index number, increase with time)
$n \geq 1$, (n : How many data you want to use in filter)
if $x \leq 0$ then $d_x = 0$

簡單平滑法之示意圖如圖 4.23 所示。從圖中可以看出這樣的雜訊抑制方法除了簡單外，還能夠在不降低 Kinect 感測器的每秒輸出張數(FPS)下，即時的對資料進行去雜訊的動作。其去雜訊前與去雜訊後的比較結果將於 5.3 節中詳細敘述。

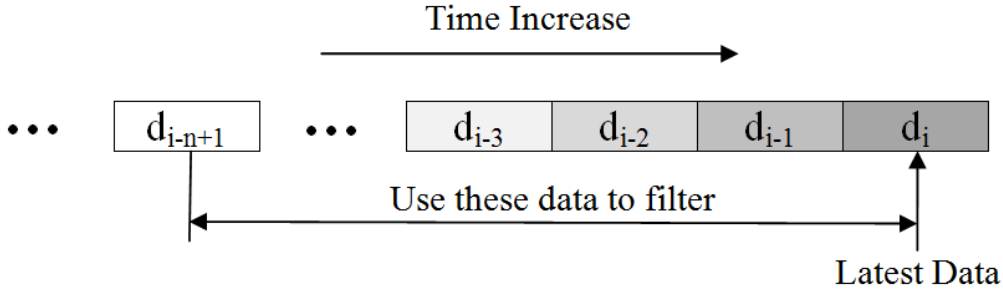


圖 4.23 簡單平滑法示意圖

在此舉個簡單平滑法的使用範例來進行解說。如圖 4.24 所示，假設 n 設為 3(代表用於平滑化的資料數量為 3)，此時當第一個資料進來時因為它是第一個資料所以前面完全沒有任何其它資料，故此時 $i = 1$ 而平均值則為 $d_1 / 3$，當下一個新的資料進來時則 $i = 2$ 平均值為 $(d_1 + d_2) / 3$，依此類推，若 $i > n$ 時，則只需將最新的三個資料拿來計算平均值即可，舊的資料則可捨棄。

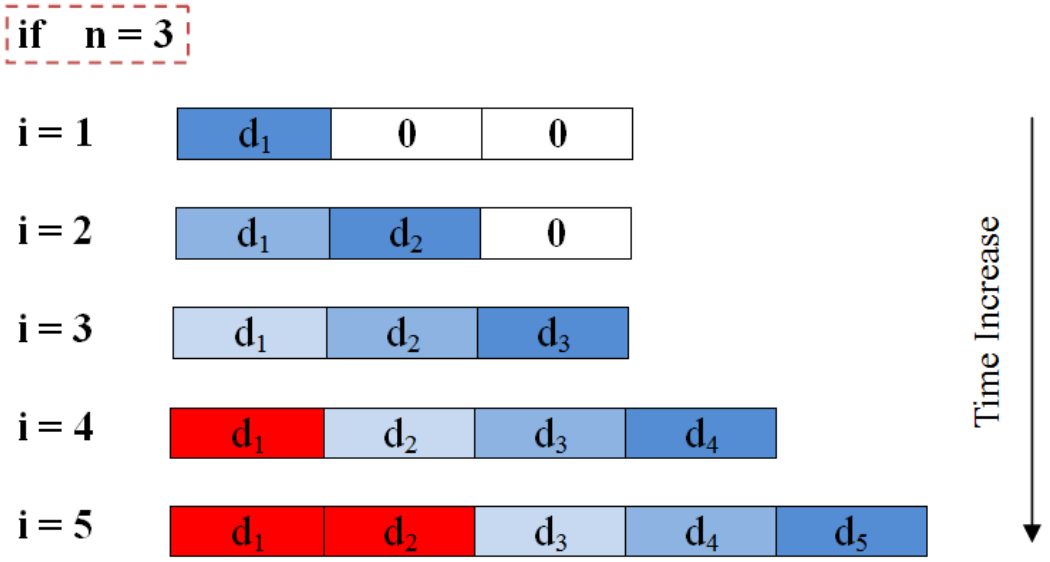


圖 4.24 簡單平滑法之使用範例

Chin-Yi University of Tech

第五章 自然體感機械手臂原型之測試與分析

本章將針對所設計的自然體感機械手臂原型進行效能、雜訊抑制、操作等各項測試，以評估整個系統的後續發展性。測試環境如表 5.1 所示。

表 5.1 系統之測試環境

Operating System :	Microsoft Windows XP Professional 32-Bit (5.1.2600 Service Pack 3 Build 2600)
Processor :	Intel Core2 CPU 6320 @1.86Ghz
Memory :	1×2048 MB (DDR2-667)
Ruby Version :	1.9.3-p194
OpenNI Version :	OpenNI-Win32-1.3.4.3-dev
NITE Version :	NITE-Win32-1.4.2.4-dev

5.1 機械手臂控制程式之效能測試

由於 Kinect 感測器本身的更新頻率為 30Hz(30FPS)，故若要讓整個系統都能夠充分運用到 Kinect 的資料，勢必要跟上 Kinect 的更新頻率才行，因此需要對機械手臂控制系統進行整體的性能測試，並且評估導入 Ruby 程式語言後的性能衰退狀況，以確保整個機械手臂控制程式的性能。

機械手臂控制程式的效能測試結果如表 5.2 所示。可以發現使用 Ruby 所實作出來的機械手臂控制程式，每執行一次需要花上約 12ms 左右，換算成頻率則為 83.333Hz，其結果完全符合當初所設定的低標(30Hz)，因此可確定目前實作出來的控制程式，足以應付整個機械手臂的控制工作，而不致於成為整個系統的瓶頸。

表 5.2 機械手臂控制程式之效能測試結果

Method Name	×1	×10	×100	×1000	×10000	×100000
calc_inv Method	0.0000s	0.0002s	0.0019s	0.0186s	0.1875s	1.8750s
drive_robot_arm Method	0.0077s	0.0781s	0.7762s	7.6875s		
mv_to Method	0.0087s	0.0817s	0.8069s	8.4531s		
Robot Arm Controller	**0.0114s**	**0.1250s**	**1.1513s**	**11.1406s**		

若深入機械手臂控制程式的運作可以發現，其主要工作是接收感測器的資料，並呼叫 move_to、gripper_open、gripper_close 等方法來控制機械手臂作動。如附錄五的機械手臂抽象層原始碼所示，其中 move_to 方法會再分別呼叫 calculate_inverse_kinematics、drive_robot_arm 這兩個方法，前者主要是計算逆向運動學，因此數學運算的比重較高，而後者則是單純驅動機械手臂作動的方法，故實作上主要以呼叫先前移植到 Ruby 上的 Dynamixel C API 為主。將表 5.2 的測試結果呈現於圖 5.1，可以看出整個控制程式耗費最多執行時間的方法為 drive_robot_arm 這個方法，它甚至在 move_to 方法裡占了 9 成左右的執行時間，可是若仔細看附錄五內的 drive_robot_arm 方法的實作程式碼可以發現，其所使用的方法幾乎只有以 C 語言實作的 Ruby 擴充，照理說以 C 語言實作出來的 Ruby 擴充執行速度應該很快才是，但為何它卻是整個機械手臂控制程式的瓶頸，經由後續的分析發現，其主要原因為 USB2Dynamixel 通訊裝置的的 I/O 速度不夠快所致，由於 USB2Dynamixel 通訊裝置是使用 FTDI 的晶片，將 USB 介面模擬成 UART Serial 介面的一個裝置，讓使用者可以透過這個裝置輕鬆的跟機械手臂馬達溝通，但重點是該裝置的 Latency Timer 最低只能設成 1ms，因此在運作上 USB2Dynamixel 通訊裝置的 I/O 速度就會變為瓶頸，成為機械手臂控制程式耗費最多執行時間的地方。

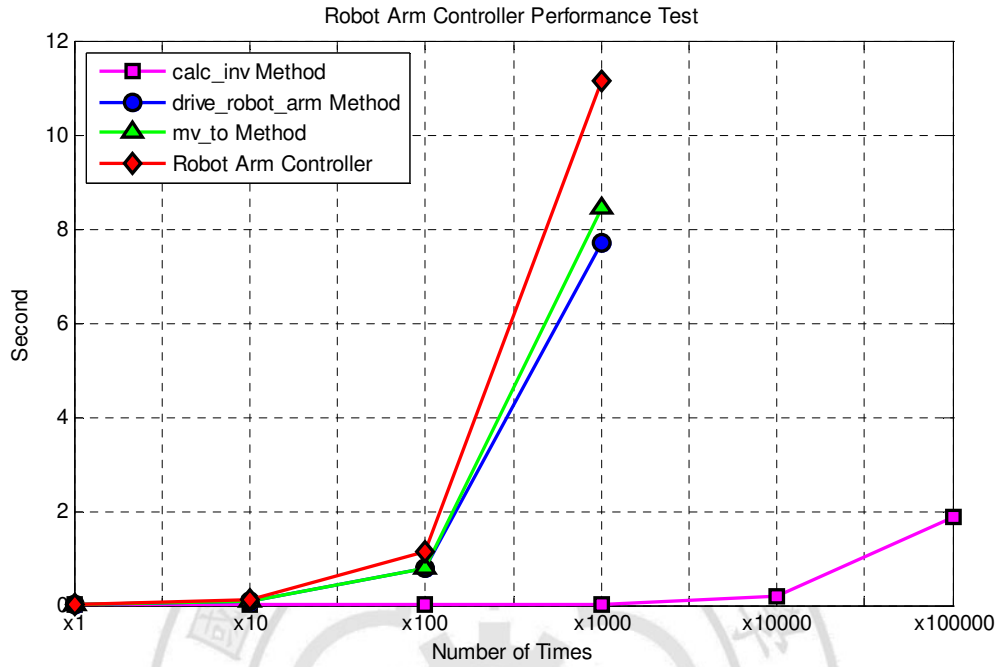


圖 5.1 機械手臂控制程式之效能測試結果

目前實作出來的機械手臂控制程式，其執行時間為 12ms/次，若從圖 5.2 來看只能排在 Mechanical Test & Analysis 的層級，但這主要是因為 USB2Dynamixel 的 I/O 速度不夠快所致，若能夠改善硬體的 I/O 性能就能夠提升到 Measurement & Control 的層級。從目前的測試結果來看可以發現，使用 Ruby 所帶來的執行效率問題並沒有想像中的嚴重，主要原因在於 Ruby 本身許多的類別與方法都是以 C 實作出來的，且演算法多經過琢磨，因此在性能上不至於相差太多，而在實作上所使用的幾乎都是以 C 實作的類別與方法為主，故在效能方面才能達到預期之水準。

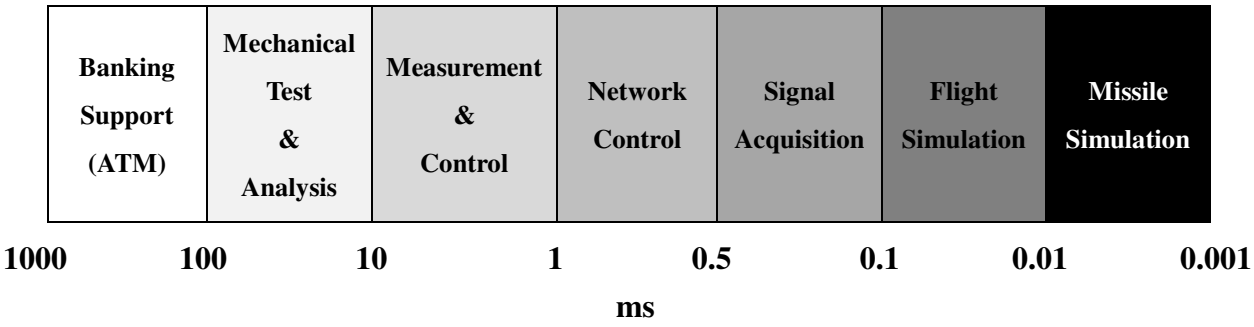


圖 5.2 各領域系統所需之響應時間(Response time)
資料來源：工業技術研究院

5.2 Kinect 感測器之深度解析度測試

由於 Kinect 感測器可輕鬆的擷取到整個場景的深度值，且售價上也相當低廉，因此許多研究者都對 Kinect 深感興趣，並嘗試將其用來取代雷射測距儀等現有之距離量測設備，而若要取代這些舊有設備，便需要透過性能測試來量化 Kinect 的性能，以評估感測器的性能是否足以勝任，這當中 Kinect 的深度感測範圍與解析度，是整個感測元件中最為重要的性能數據，因此這部分的性能測試將是本節的重點。

在過去已有研究者針對 Kinect 的深度感測能力進行量測[3, 30, 31, 32, 33]，其中 J.Stowers[3]等透過逆向工程的方式發現 Kinect 的深度解析度為 11-Bit，並經由量測得到 Kinect 感測器原始深度值(Raw Data)與真實距離(即 Kinect 與被測物間之距離)間之關係，其結果如圖 5.3 所示。從圖 5.3 中的結果可以發現，原始深度值與真實距離間之關係並沒有非常線性，且在真實距離為 1000~3000mm 時感測器的靈敏度較高，而在 4000mm 以後靈敏度下降的非常快，同樣的結果也出現在 J.Smisek [30]等的 Kinect 性能測試結果當中。

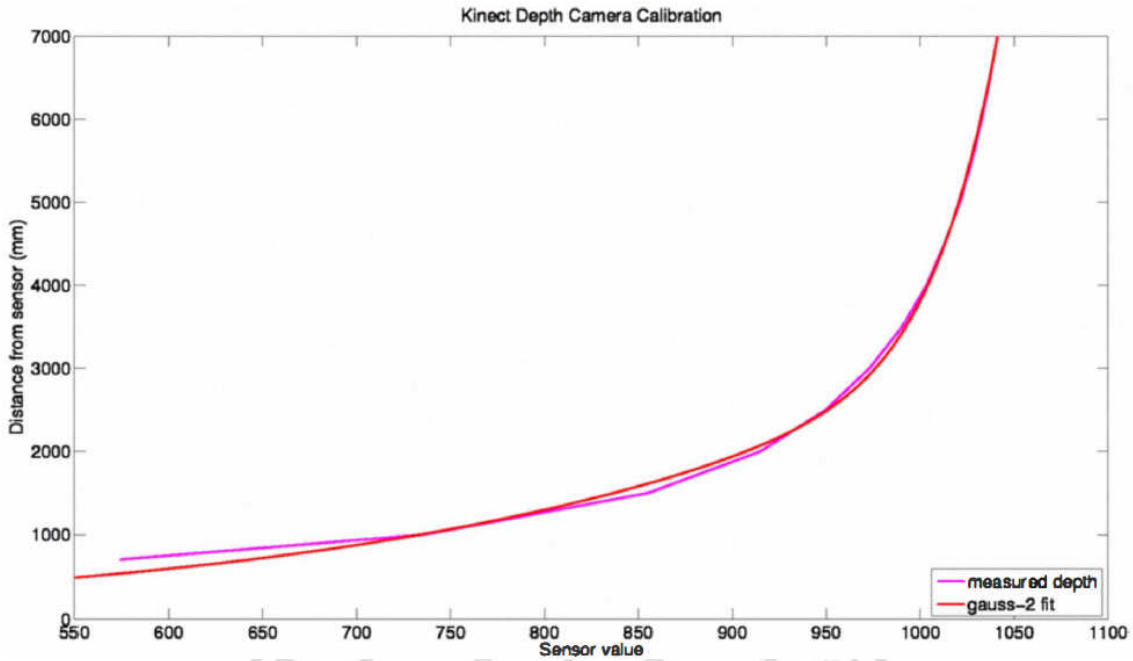


圖 5.3 Kinect 感測器原始深度值與真實距離間之關係

資料來源：文獻[3]

由於本研究是使用 OpenNI 的軟體開發框架，來開發自然體感機械手臂原型，故在此使用開發框架提供之深度影像擷取方法來擷取深度值。量測結果如圖 5.4 所示，由於 OpenNI 軟體開發框架的限制，因此量測範圍只有 50cm~10m 左右，且所提供的深度值也並非原始深度值，而是已經轉換過的真實距離，但還是可以發現其結果與文獻[3, 30]所得到的結果不謀而合，真實距離在超越 4000mm 以後解析度變得越來越差。

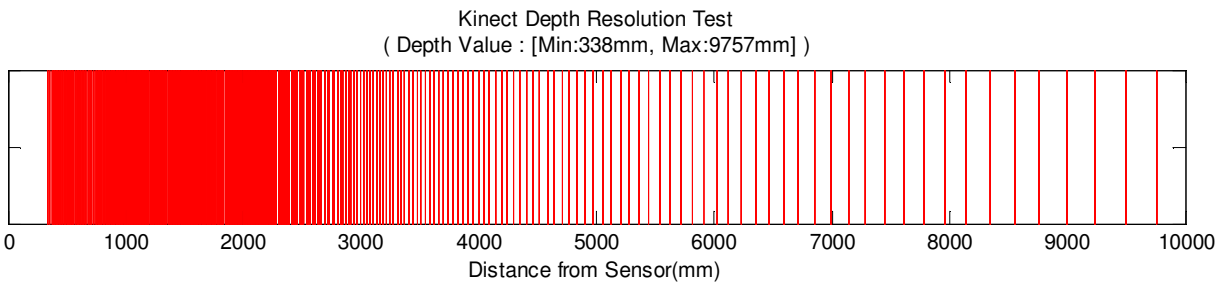


圖 5.4 Kinect 感測器之深度解析度測試結果

透過量測到的深度資料，更進一步將其轉換成 Kinect 感測器於各深度之解析度，結果如圖 5.5 所示。可以發現解析度隨著距離的增加，以二次曲線的形式不斷上升，在距離為 1m 時解析度約為 3mm，2m 時則為 11mm，但到 3m 時已經來到 26mm，而到了 4m 時則是較為無法接受的 46mm。由實驗結果可知，當量測距離超過 4m 以上時解析度相當差，若將其應用於精密量測上是非常不適合的。

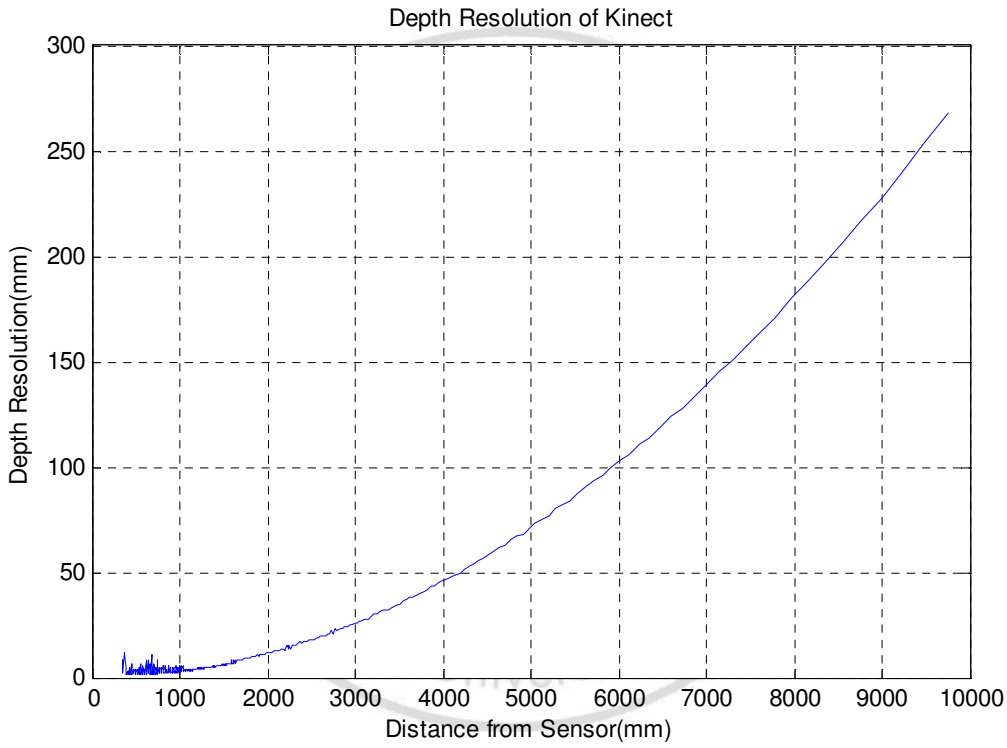


圖 5.5 Kinect 感測器之深度解析度

M.A.Livingston [31, 32, 33]等發現，距離越遠 Kinect 感測器的深度值誤差量就越大，如圖 5.6 所示，從前述的實驗結果可得知，誤差量增加的主要原因在於，感測器本身的解析度會隨著真實距離的增加而隨之下降，且衰退幅度在超越 4m 後就會變得相當大，這也是為什麼 Microsoft[20]建議的使用範圍會只有短短的 1.2~3.5m

的原因。綜合前面的各項實驗結果，本研究將自然體感機械手臂原型的感測器與使用者間之距離設定在 1.5m，並以此為原點進行機械手臂之位置控制(在此操作範圍內，Kinect 感測器解析度約在 7mm 左右)，藉此提高感測器的解析度，以降低感測器的誤差量。

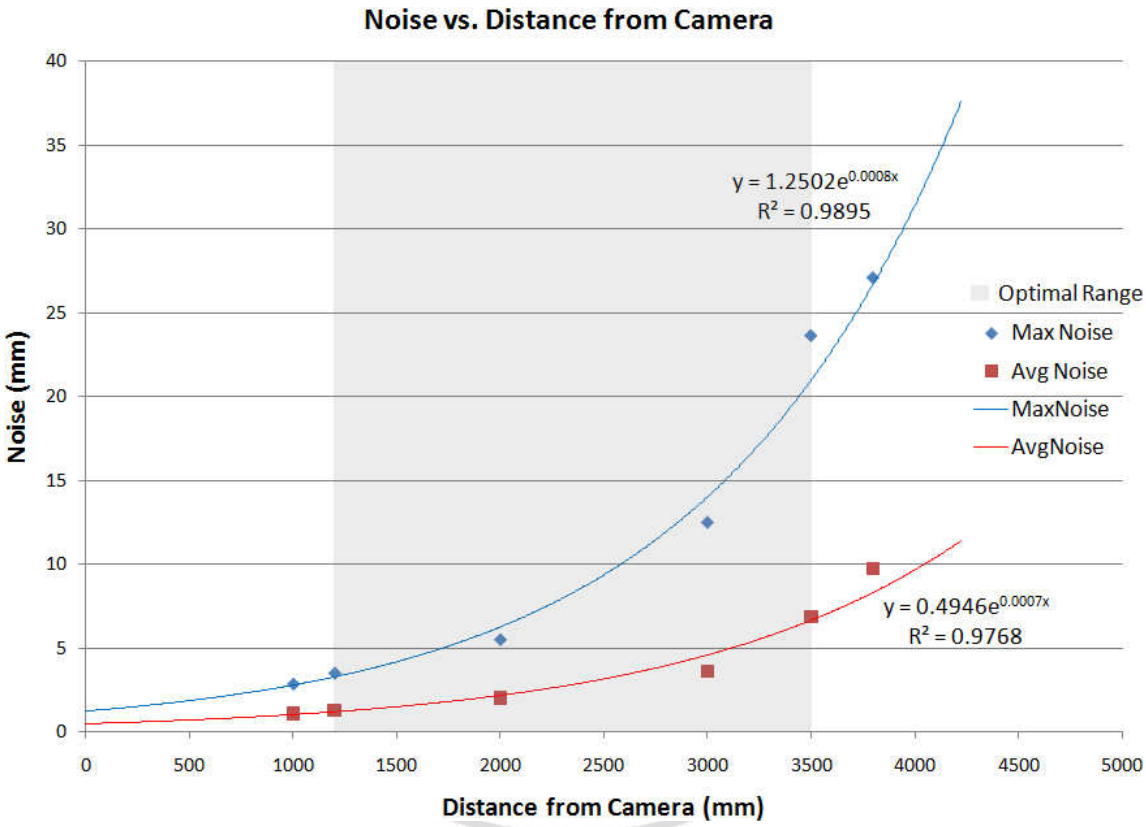


圖 5.6 Kinect 感測器之真實距離與雜訊間之關係

資料來源：文獻[32]

5.3 Kinect 感測器於手部追蹤時之雜訊抑制測試

在使用者手部追蹤辨識上，本研究是使用 OpenNI 開發框架所提供之手部追蹤功能，來抓取使用者的手部位置，但由於雜訊的關係使得輸出之手部位置不穩定的跳動，無法直接用於機械手臂的位置控制上，因此本研究提出一簡單平滑法來解決此問題(請參閱 4.6 節)，並透過測試結果來比較使用前與使用後之差異。基本上 OpenNI 開發框架所定義之座標系統，是使用左手定則來表示，而非常見的右手定則，如圖 5.7 所示，故在此測試結果均依據此座標系統來表示。

為求得精確的比較結果，本實驗透過固定使用者手部的方式，來進行重複誤差(Repeatability Error)測試，在理想的狀況下重複誤差越接近零越好。量測結果如圖 5.8 所示，可以發現在未使用簡單平滑法前雜訊較大，且跳動的程度也較高，而在使用後均獲得顯著的改善。

圖 5.7 OpenNI 座標系統

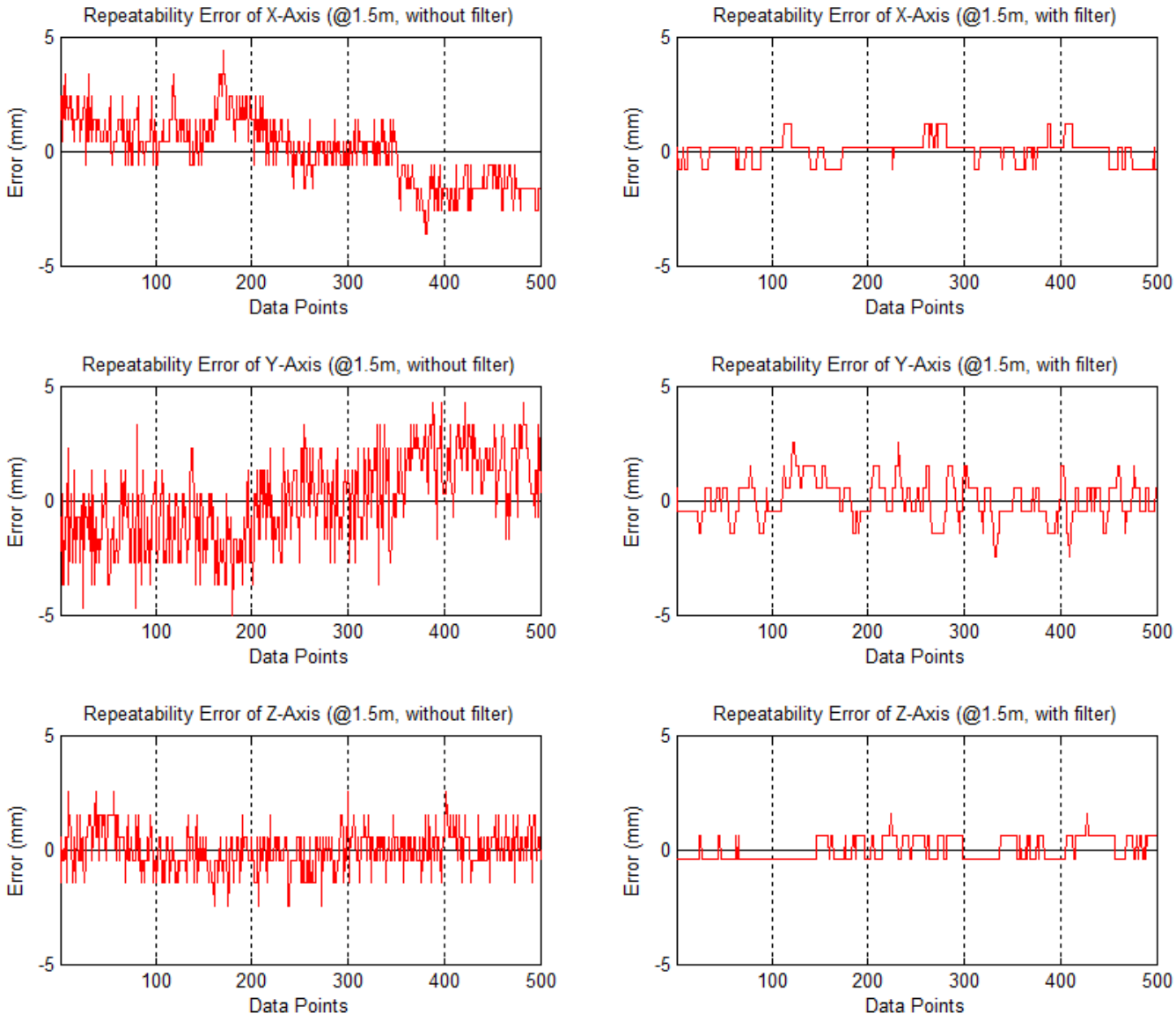


圖 5.8 Kinect 感測器於手部追蹤時之雜訊抑制測試結果

若將圖 5.8 的結果透過最大重複誤差(Maximum Repeatability Error)、標準差(Standard Deviation)來分析，則可發現在最大重複誤差方面，其改善程度相當顯著，分別有 X=73.42%、Y=41.02%、Z=37.15%的改善，而在標準差的部分其分析結果顯示，資料的集中程度提高了許多，這代表 Kinect 感測器於手部追蹤時的重複精

度有相當程度的改善，改善程度分別為 X=62.13%、Y=53.44%、Z=45.89%。其結果如圖 5.9、圖 5.10 所示。

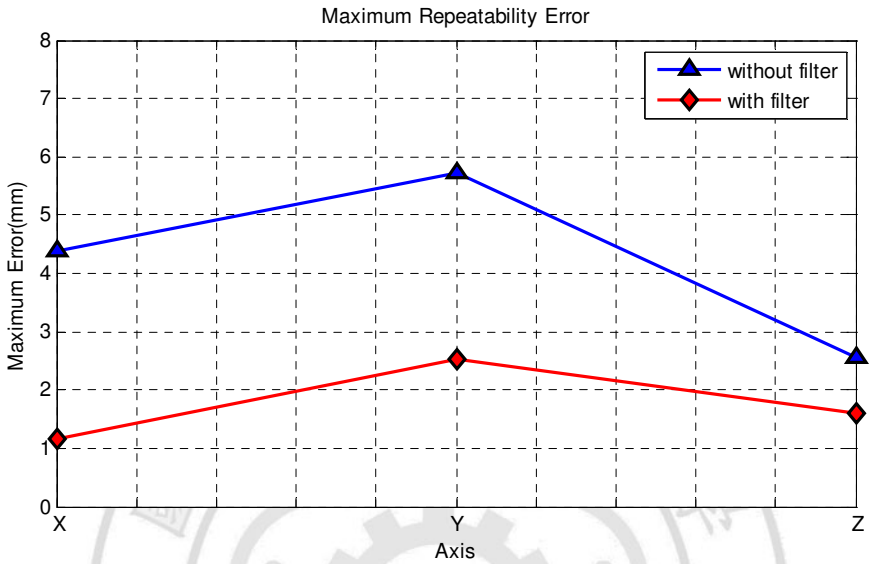


圖 5.9 Kinect 感測器於手部追蹤時之雜訊抑制測試結果(最大重複誤差)

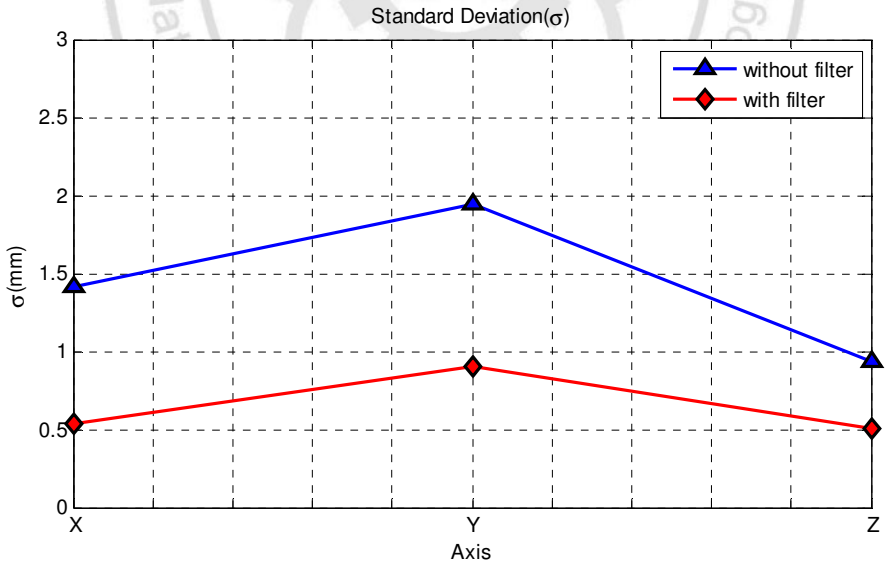


圖 5.10 Kinect 感測器於手部追蹤時之雜訊抑制測試結果(標準差 σ)

經由上述實驗結果發現，本研究所使用的雜訊抑制方法能夠有效抑制手部追蹤時的雜訊，減少機械手臂的抖動情形，進而提高動作的精確度與正確性。

5.4 自然體感機械手臂原型之操作測試

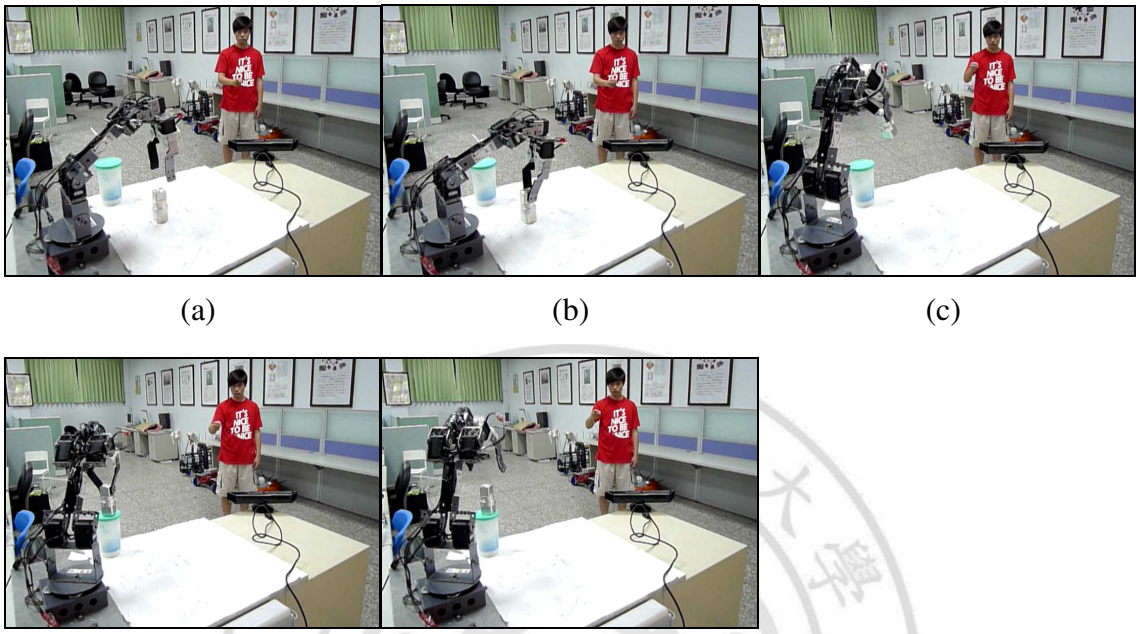
本研究使用 RJ-45 網路線(跳線)來連接使用者操作端之感測伺服器，與遠端機械手臂之控制系統，用以模擬遠距離之操作情境。經由實際之操作測試發現，在未使用簡單平滑法的情況下操作機械手臂，機械手臂本身會因為雜訊的關係震動的相當厲害，進而影響到移動與夾取的精準度，而在使用後則獲得相當大之改善，且已經可以用於許多不需要高精度的應用中，例如:物件的搬運等。在機械手臂的移動速度上，礙於伺服馬達的性能以及機械手臂整體的強度，故在設計之初便已限制各軸之伺服馬達速度於 5.55rpm，因此若使用者的手移動太快，就會造成機械手臂跟不上的情況發生，但若操作在伺服馬達的速度限制內，則可以得到即時的系統操作反應。

目前實作方面已完成機械手臂之位置控制，與夾爪之夾取動作控制。由於本研究所使用之機械手臂為小型機械手臂，全長約為 537mm，故在機械手臂的位置控制上，移動距離是採用與使用者 1:1 的對應方式在移動，也就是若使用者將手向 X、Y、Z 軸各移動+5cm，則機械手臂同樣也會向 X、Y、Z 軸各移動+5cm。

◎自然體感機械手臂原型之操作方式

首先，握拳並舉起右手，此時程式會開始追蹤使用者的拳頭，並計算其真實世界之三維位置，再將其位置當作輸入驅動機械手臂作動。因此當程式開始追蹤使用者的拳頭時，便會根據拳頭的實際位置要求機械手臂做出即時之相對運動。而當伸出食指時機械手臂的夾爪便會張開，食指收回時夾爪便會合起來。操作範

例如圖 5.11 所示。



(a) (b) (c)

(d) (e)

圖 5.11 自然體感機械手臂原型之操作範例

National Chin-Yi University of Technology

第六章 結論

6.1 結果與討論

本研究透過新技術的導入，以快速又穩健的方式發展一自然體感之機械手臂控制平台，並透過此原型展示其相關概念。綜合各項實驗結果，歸納出以下幾點結論：

1. 在自然體感的概念上，本研究透過導入 Kinect 感測器與 Ruby 程式語言來開發此一原型，在實作上可謂相當創新。
2. 在開發機械手臂之控制程式時，透過導入 Ruby 程式語言，並以它為主力語言進行開發，希望能夠藉此提高軟體之開發速率。其中機械手臂的部分，從規劃到撰寫完成共只花了兩天的時間(未包含後續之維護與新功能之追加)，生產力相當驚人，且事後的維護也相當輕鬆。
3. 在效能方面，可從 5.1 節的效能測試結果發現，原先擔心使用 Ruby 後會導致的執行效率問題，已被證實其擔心是多餘的，性能能夠如此優異的主要原因在於 Ruby 的許多類別與方法都是以 C 實作出來的，且演算法多經過琢磨，因此在性能上不致於相差太多。
4. 在 Kinect 感測器方面，由於硬體本身與過去其它的感測器較不相同，故本研究在導入初期便以評估過 Kinect 感測器之各項性能，並將自然體感機械手臂原型的感測器與使用者間之距離設定在 1.5m(感測器解析度在 1cm 以內)，以此為原點進行機械手臂之位置控制。
5. 目前實作所建立的平台可達到機械手臂之位置控制與夾爪之夾取動作控制。隨

著科技的進步與演算法的開發，精度的部分是可以再向上提升的。

本研究也透過各項競賽展示其功能與相關概念，並在建國科技大學所舉辦的2011 全國科技創意暨微電腦應用競賽，以及中國工業職業教育學會所舉辦的 2011 台灣區電腦化運動競技大賽，分別榮獲第三名(科技創意實現組)、第二名(機器人專題競賽組)的佳績，在參賽過程中也獲得一致的好評。

自然體感主要強調人就是控制器的概念，其優勢在於使用者不需使用任何的控制器，只需使用自己的身體及語音等人類與生俱來的直覺進行操作，即可將身體和聲音作為虛擬控制器，驅動任一使用者期望控制的設備。在本研究中，主要是將自然體感運用於機械手臂的位置控制，即可達到人機合一的境界。因此，我們可預期自然體感的應用將會越來越多也越來越廣泛。

6.2 未來展望

透過本研究所開發之自然體感機械手臂原型，除了展示相關概念之外，也奠定未來持續開發之基礎。且透過導入 Ruby 程式語言快速打造之方式，讓開發者迅速找到可能會遭遇到的問題與瓶頸，並在未來依照性能上的需求，可透過先前使用 Ruby 所開發之演算法，輕鬆改寫成 C/C++程式碼。實際上使用 Ruby 寫出來的程式其執行速度，都慢於 C/C++(GNU GCC)所寫出來的程式 10 ~ 100 倍之多[34]，同樣的情形也發生在 Python、Perl 身上，但這就是使用高階語言的通病，不過這樣的缺點隨著語言本身實作上的精進與改善，在未來是可以持續縮小差距的。雖然 Ruby 這類型的程式語言，在性能上還是與 C/C++有著不小的差距，但應用於開發上絕對能夠有效提升程式開發效率，而在開發後所得到之演算法，也有助於未來的移植與改寫。

本研究的主要目的在於自然體感概念的實現，若要運用在實際應用中，還是有許多地方有待克服，故本研究針對未來的改進方向提出以下幾點建議：

1. 由於感測器的硬體性能會影響到感測資料的品質，故強化感測器硬體本身，以及原始感測資料(Raw Data)前處理演算法之改善，都是相當重要的。而影像辨識演算法部分，也攸關到辨識的精確度，故也是相當值得深入探討的課題之一。
2. 在機械手臂的部分，由於需要使用在相當嚴苛的環境，故耐碰撞之特性就相當重要，才不致於在作業中因外力之撞擊而輕易的損壞，這部分也需要投入相當大的資源，來進行耐碰撞之高精度機械手臂的研發與製造。
3. 系統本身響應的即時性，也是相當重要的一塊，這會影響到系統的響應夠不夠即時，也是在整個整合過程中相當棘手的問題之一。因為影響到系統響應即時性的部份相當廣泛，像是控制程式執行速度、機械手臂響應速度、連接使用者與遠端機械手臂的網路延遲等，故這部份在開發的過程中都必須被仔細的考慮進去，且最後的測試與除錯也都相當重要。

參考文獻

[1] C.C.Liebe, C.Padgett, J.Chapsky, D.Wilson, K.Brown, S.Jerebets, H.Goldberg, J.Schroeder, "Spacecraft hazard avoidance utilizing structured light", IEEE Aerospace Conferences, 2006

[2] S.Matyunin, D.Vatolin, Y.Berdnikov, M.Smirnov, "Temporal filtering for depth maps generated by Kinect depth camera", IEEE 3DTV Conferences, 2011

[3] J.Stowers, M.Hayes, A.Bainbridge-Smith, "Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor", IEEE International Conference on Mechatronics(ICM), 2011

[4] N.Ganganath, H.Leung, "Mobile robot localization using odometry and kinect sensor", IEEE Emerging Signal Processing Applications(ESPA) International Conference, 2012

[5] J.Tong, J.Zhou, L.Liu, Z.Pan, H.Yan, "Scanning 3D Full Human Bodies Using Kinects", IEEE Visualization and Computer Graphics Magazine, VOL.18, NO.4, April 2012, Pages 643-650

[6] C.Harrison , H.Benko, A.D.Wilson, "OmniTouch: Wearable Multitouch Interaction Everywhere", ACM Symposium on User Interface Software and Technology(UIST), 2011

[7] L.A.Schwarz, A.Mkhitaryan, D.Mateus, N.Navab, "Human skeleton tracking from depth data using geodesic distances and optical flow", Image and Vision Computing,

Volume 30, Issue 3, March 2012, Pages 217-226

[8] L.Gallo, A.P.Placitelli, M.Ciampi, "Controller-free exploration of medical image data: Experiencing the Kinect", IEEE Computer-Based Medical Systems(CBMS) International Conference, 2011

[9] S.Mehrotra, Z.Zhang, Q.Cai, C.Zhang, P.A.Chou, "Low-complexity, near-lossless coding of depth maps from kinect-like depth cameras", IEEE Multimedia Signal Processing Conference(MMSP), 2011

[10] K.Tanaka, Y.Matsumoto, H.Arimori, "Embedded System Development by Lightweight Ruby", IEEE International Conference on Computational Science and Its Applications, 2011

[11] "FAAST Project Website", "http://projects.ict.usc.edu/mxr/faast/"

[12] E.A.Suma, B.Lange, A.Rizzo, D.M.Krum, M.Bolas, "FAAST : The Flexible Action and Articulated Skeleton Toolkit", IEEE Virtual Reality Conferences, 2011

[13] "Ultimate Battlefield 3 Simulator (Build & Test) – The Gadget Show", "http://www.youtube.com/watch?v=eg8Bh5iI2WY"

[14] "Programming Ruby : The Pragmatic Programmers' Guide", by Dave Thomas, with Chad Fowler and Andy Hunt

[15] "松本行弘的程式世界 : 成為一流程式設計師的 14 種思考術", 作者：松本行弘, 譯者：鄧瑋敦

[16] "Ruby Programming Language Official Website", "http://www.ruby-lang.org/"

[17] "The Mythical Man-Month : Essays on Software Engineering", by Frederick P. Brooks, Jr.

[18] "The Teardown_Teardown Kinect", IET Journals & Magazines, 2011

[19] "PrimeSense Reference Design 1.08", PrimeSense Inc., 2010

[20] "Programming Guide : Getting Started with the Kinect for Windows SDK Beta", Microsoft Research, 2011

[21] "Blog", By Eric Gregori, "http://buildsmartrobots.ning.com/profiles/blogs/one-year-anniversary-for-the-kinect-over-10-million-units-shipped"

[22] "OpenNI Organization Official Website", "http://www.openni.org/"

[23] "OpenNI User Guide", OpenNI, 2010

[24] "NITE Controls User Guide", PrimeSense Inc., 2011

[25] "NITE Algorithms 1.3 Note", PrimeSense Inc., 2010

[26] Jwu-Sheng Hu, Ming-Chih Chien, Yung-Jung Chang, Shyh-Haur Su, and Chen-Yu Kai, "A Ball-Throwing Robot with Visual Feedback", IEEE Intelligent Robots and Systems Conferences, 2010

[27] "ROBOTIS User's Manual_Dynamixel:AX-12", ROBOTIS, 2006

[28] "ROBOTIS User's Manual_USB2Dynamixel", ROBOTIS, 2006

[29] 莊修一, "AX-12 機械手臂路徑軌跡規劃與影像辨識之研究", 國立勤益科技大學機械工程系碩士論文, 2010

[30] J.Smisek, M.Jancosek, T.Pajdla, "3D with Kinect", IEEE Computer Vision Workshops International Conference(ICCV Workshops), 2011

[31] C.S.Park, S.W.Kim, D.Kim, S.R.Oh, "Comparison of plane extraction performance using laser scanner and Kinect", IEEE Ubiquitous Robots and Ambient Intelligence(URAI) International Conference, 2011

[32] M.A.Livingston, J.Sebastian, Z.Ai, J.W.Decker, "Performance measurements for the Microsoft Kinect skeleton", IEEE Virtual Reality(VR) Conference, 2012

[33] T.Dutta, "Evaluation of the Kinect™ sensor for 3-D kinematic measurement in the workplace", Applied Ergonomics, Volume 43, Issue 4, July 2012, Pages 645-649

[34] The Computer Language Benchmarks Game, "http://shootout.alioth.debian.org/"

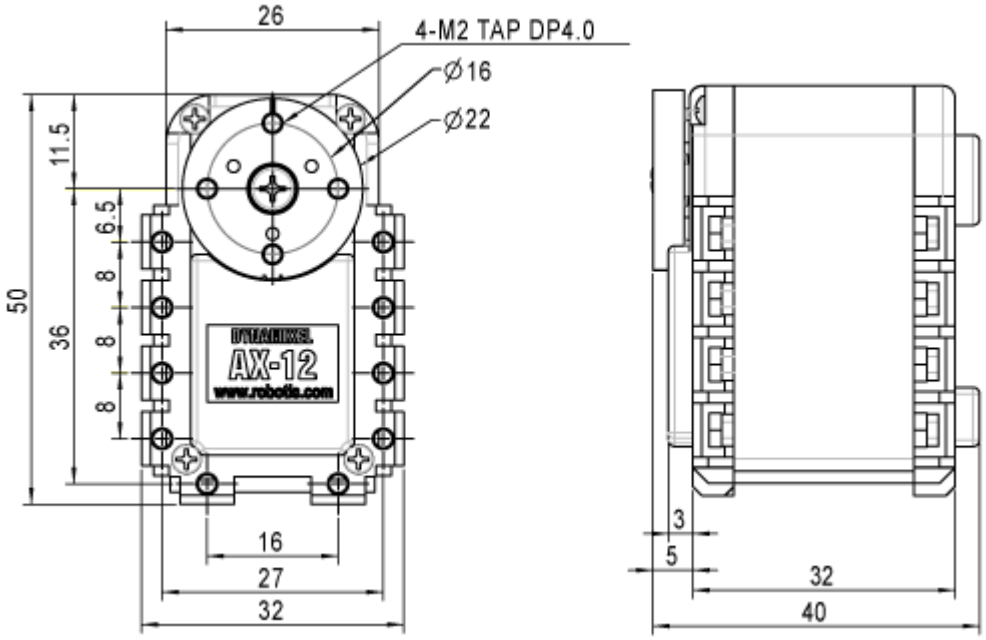
國立勤益科技大學 National Chin-Yi University of Technology 勤益 NCUT

附錄

附錄一 ：ROBOTIS AX-12+ Control Table

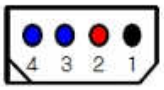
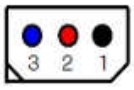
Area	Address (Hexadecimal)	Name	Description	Access	Initial Value (Hexadecimal)
EEPROM	0 (0X00)	Model Number(L)	Lowest byte of model number	R	12 (0X0C)
	1 (0X01)	Model Number(H)	Highest byte of model number	R	0 (0X00)
	2 (0X02)	Version of Firmware	Information on the version of firmware	R	-
	3 (0X03)	ID	ID of Dynamixel	RW	1 (0X01)
	4 (0X04)	Baud Rate	Baud Rate of Dynamixel	RW	1 (0X01)
	5 (0X05)	Return Delay Time	Return Delay Time	RW	250 (0XFA)
	6 (0X06)	CW Angle Limit(L)	Lowest byte of clockwise Angle Limit	RW	0 (0X00)
	7 (0X07)	CW Angle Limit(H)	Highest byte of clockwise Angle Limit	RW	0 (0X00)
	8 (0X08)	CCW Angle Limit(L)	Lowest byte of counterclockwise Angle Limit	RW	255 (0XFF)
	9 (0X09)	CCW Angle Limit(H)	Highest byte of counterclockwise Angle Limit	RW	3 (0X03)
	11 (0X0B)	the Highest Limit Temperature	Internal Limit Temperature	RW	70 (0X46)
	12 (0X0C)	the Lowest Limit Voltage	Lowest Limit Voltage	RW	60 (0X3C)
	13 (0X0D)	the Highest Limit Voltage	Highest Limit Voltage	RW	140 (0XBE)
	14 (0X0E)	Max Torque(L)	Lowest byte of Max. Torque	RW	255 (0XFF)
	15 (0X0F)	Max Torque(H)	Highest byte of Max. Torque	RW	3 (0X03)
	16 (0X10)	Status Return Level	Status Return Level	RW	2 (0X02)
	17 (0X11)	Alarm LED	LED for Alarm	RW	36(0x24)
	18 (0X12)	Alarm Shutdown	Shutdown for Alarm	RW	36(0x24)
RAM	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	CW Compliance Margin	CW Compliance margin	RW	1 (0X01)
	27 (0X1B)	CCW Compliance Margin	CCW Compliance margin	RW	1 (0X01)
	28 (0X1C)	CW Compliance Slope	CW Compliance slope	RW	32 (0X20)
	29 (0X1D)	CCW Compliance Slope	CCW Compliance slope	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Lowest byte of Goal Position	RW	-
	31 (0X1F)	Goal Position(H)	Highest byte of Goal Position	RW	-
	32 (0X20)	Moving Speed(L)	Lowest byte of Moving Speed	RW	-
	33 (0X21)	Moving Speed(H)	Highest byte of Moving Speed	RW	-
	34 (0X22)	Torque Limit(L)	Lowest byte of Torque Limit	RW	ADD14
	35 (0X23)	Torque Limit(H)	Highest byte of Torque Limit	RW	ADD15
	36 (0X24)	Present Position(L)	Lowest byte of Current Position	R	-
	37 (0X25)	Present Position(H)	Highest byte of Current Position	R	-
	38 (0X26)	Present Speed(L)	Lowest byte of Current Speed	R	-
	39 (0X27)	Present Speed(H)	Highest byte of Current Speed	R	-
	40 (0X28)	Present Load(L)	Lowest byte of Current Load	R	-
	41 (0X29)	Present Load(H)	Highest byte of Current Load	R	-
	42 (0X2A)	Present Voltage	Current Voltage	R	-
	43 (0X2B)	Present Temperature	Current Temperature	R	-
	44 (0X2C)	Registered	Means if Instruction is registered	R	0 (0X00)
	46 (0X2E)	Moving	Means if there is any movement	R	0 (0X00)
	47 (0X2F)	Lock	Locking EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Lowest byte of Punch	RW	32 (0X20)
	49 (0X31)	Punch(H)	Highest byte of Punch	RW	0 (0X00)

附錄二 ：ROBOTIS AX-12+ Dimension

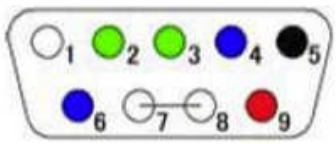


附錄三 ：ROBOTIS USB2Dynamixel Connector Pin Layout

[PIN Figure of 4P / 3P Cable Connctors]

4 Pin Cable			3 Pin Cable		
Pin No.	Signal	Pin Figure	Pin No.	Signal	Pin Figure
1	GND		1	GND	
2	NOT Connected		2	NOT Connected	
3	DATA + (RS-485)		3	DATA (TTL)	
4	DATA - (RS-485)				

[PIN Figure of Serial Connector]

Pin No.	Signal	Pin Figure
1	Data (TTL)	
2	RXD (RS-232)	
3	TXD (RS-232)	
4	D+ (RS-485)	
5	GND	
6	D- (RS-485)	
7	Short with No. 8	
8	Short with No. 7	
9	USB power (5V)	

附錄四 ：AX-12+ Robot Arm Controller Source Code (Ruby)

```
#! /usr/bin/env ruby
#  encoding: UTF-8
#
#  Program name : AX-12 Robot Arm Controller (Kinect Version)
#  Created by Louis Smith on May 9, 2011
#
#  Arguments:
#    --force_sensor_on

# [ Add Load Path  ] #-----------------------------------------#
$LOAD_PATH.push( File.dirname(__FILE__) + '/../lib' )

# [ Import Library ] #-----------------------------------------#
require "AX-12 Robot Arm"
require "socket"

# Init Robot Arm #
robot_arm = AX12_Robot_Arm.new()

# Force Sensor Daemon #
if ARGV.include? '--force_sensor_on'
  $force_sensor = Thread.new do
    begin
      puts " - Connecting to Remote Host (Force Sensor), Please Wait - "
      force_sensor_session = TCPSocket.new('127.0.0.1', 23125)

      loop {
        # Request Force Sensor Data
        force_sensor_session.print('r')

        # Receive a 32-Bits(4-Bytes) Float Number from Force Sensor Server
        raw_data = force_sensor_session.recv(4)
```

```
        raise(Exception, 'Session Closed') if raw_data.empty?
        voltage = raw_data.unpack('F').first

        # Force Sensor - Polynomial Regression Second-Order (Sensor1 + Sensor2) Reversed
        percentage_of_torque = 100 * ( 6.9099 + 92.0557 * voltage - 6.0417 * voltage**2 ) / 300.0

        if ( percentage_of_torque < 10 )
          percentage_of_torque = 10
        elsif ( percentage_of_torque > 100 )
          percentage_of_torque = 100
        end

        # Change Gripper Torque
        robot_arm.ch_gripper_torque( percentage_of_torque )

        # 5Hz(0.2 Sec) Update Frequency
        sleep(0.2)
      }

    rescue Exception => err_msg
      puts('Error: ' + err_msg.to_s, 'retry') if force_sensor_session
      if force_sensor_session.respond_to?('closed?')
        force_sensor_session.close unless force_sensor_session.closed?
      else
        force_sensor_session.close if force_sensor_session
      end
      sleep(1)
      retry

    end
  end
end
```

```
begin
  # Start TCP/IP Communication #
  puts " - Connecting to Remote Host (Kinect Sensor), Please Wait - "
  kinect_session = TCPSocket.new('127.0.0.1', 23124)

  # Main Loop #
  loop {
    # X(mm), Y(mm), Z(mm), diff(mm)
    kinect_session.recv(20) =~ /([+-]\d\d\d\d)([+-]\d\d\d\d)([+-]\d\d\d\d)([+-]\d\d\d\d)/

    x =  $1.to_i          # x(mm)
    y = ($3.to_i - 1500)  # y(mm)
    z =  $2.to_i          # z(mm)
    depth_diff = $4.to_i  #  (mm) : The difference between the hand and the nearest point of the distance of the Z axis

    # Drive the robot arm
    robot_arm.mv_to(x, y, z)

    # Open or close gripper
    if (depth_diff >= 60)  # >=60mm
      robot_arm.gripper_open
    elsif (robot_arm.gripper_status != :close)
      robot_arm.gripper_close
    end

    # Print info on screen
    printf("X%+05imm, Y%+05imm, Z%+05imm, G%+05imm\n", x, y, z, depth_diff)
  }

rescue Exception => err_msg
  puts('Error: ' + err_msg.to_s, 'retry') if kinect_session
  if kinect_session.respond_to?('closed?')
    kinect_session.close unless kinect_session.closed?
  else
```

```
    kinect_session.close if kinect_session
  end
  sleep(1)
  retry

end
```

附錄五 ：AX-12+ Robot Arm Abstraction Layer Source Code (Ruby)

```
#! /usr/bin/env ruby
#  encoding: UTF-8
#
#  Program name : AX-12+ Robot Arm
#  Created by Louis Smith on Dec. 30, 2010.

# [ Import Ext. Library ] #-----------------------------------------#
require "Dynamixel"

# [ Included Module     ] #-----------------------------------------#
include Math

class AX12_Robot_Arm
  attr_reader   :gripper_status
  attr_accessor :serial_Port_Num,
                :serial_Comm_Speed,
                :axis1_Motor_ID,
                :axis2_L_Motor_ID,
                :axis2_R_Motor_ID,
                :axis3_L_Motor_ID,
                :axis3_R_Motor_ID,
                :axis4_Motor_ID,
                :gripper_L_Motor_ID,
                :gripper_R_Motor_ID

private
  def initialize( robot_arm_config_data = { :Serial_Port_Num    => 10,
                                            :Serial_Comm_Speed  =>  1,
                                            :Axis1_Motor_ID     =>  1,
                                            :Axis2_L_Motor_ID   =>  5,
                                            :Axis2_R_Motor_ID   =>  3,
                                            :Axis3_L_Motor_ID   =>  9,
```

```
                                        :Axis3_R_Motor_ID   =>  7,
                                        :Axis4_Motor_ID     => 15,
                                        :Gripper_L_Motor_ID => 16,
                                        :Gripper_R_Motor_ID => 17 } )

  @serial_Port_Num    = robot_arm_config_data[:Serial_Port_Num]
  @serial_Comm_Speed  = robot_arm_config_data[:Serial_Comm_Speed]
  @axis1_Motor_ID     = robot_arm_config_data[:Axis1_Motor_ID]
  @axis2_L_Motor_ID   = robot_arm_config_data[:Axis2_L_Motor_ID]
  @axis2_R_Motor_ID   = robot_arm_config_data[:Axis2_R_Motor_ID]
  @axis3_L_Motor_ID   = robot_arm_config_data[:Axis3_L_Motor_ID]
  @axis3_R_Motor_ID   = robot_arm_config_data[:Axis3_R_Motor_ID]
  @axis4_Motor_ID     = robot_arm_config_data[:Axis4_Motor_ID]
  @gripper_L_Motor_ID = robot_arm_config_data[:Gripper_L_Motor_ID]
  @gripper_R_Motor_ID = robot_arm_config_data[:Gripper_R_Motor_ID]

  init_robot_arm()
end

def init_robot_arm()
  # Initializes the communication device and return status value.
  status = Dynamixel.initialize(@serial_Port_Num, @serial_Comm_Speed)

  # Display return status value of the Dynamixel communication device after Init.
  if (status == 1)
    # Success
    puts "Dynamixel Communication Device Init. 'Success':\n  " +
         "-> Device Index=#{@serial_Port_Num}\n  " +
         "-> Baud Rate Number=#{@serial_Comm_Speed}"
  else
    # Failure
    puts "- Dynamixel Communication Device Init. 'Failure' -"
    sleep(10)
    exit()
```

```
end

# AX-12+ Robot Arm Parameters
@d_1 = Float( 115 )  # mm
@a_2 = Float( 172 )  # mm
@a_3 = Float(  75 )  # mm
@d_4 = Float( 175 )  # mm

# Init. Position
@init_x = Float(    0 )
@init_y = Float( @d_4 )
@init_z = Float( @d_1 + @a_2 + @a_3 )

# Init Robot Arm
ch_driver_speed()          # Init Moving Speed of all Three Axes
ch_gripper_speed()         # Init Moving Speed of the Gripper
ch_rotate_gripper_speed()  # Init Moving Speed of Axis Four
mv_to()                    # Drive the Robot Arm to the Init Position
mv_gripper()               # Drive the Robot Arm to the Init Position
rotate_gripper()           # Drive the Robot Arm to the Init Position

# Gripper Status (Open or Close)
@gripper_status = :open

# Prevent Overload of the Gripper Motor
@gripper_protector = Thread.new do
  loop {
    if (@gripper_status == :close) and !(gripper_moving?)
      left_gripper_value  = Dynamixel.read_word(@gripper_L_Motor_ID, 36)
      right_gripper_value = Dynamixel.read_word(@gripper_R_Motor_ID, 36)
      Dynamixel.write_word(@gripper_L_Motor_ID, 30, left_gripper_value  + 10)
      Dynamixel.write_word(@gripper_R_Motor_ID, 30, right_gripper_value - 10)
    end
    sleep(0.2)  # 5Hz
```

```
      }
    end
  end

public
  def ch_driver_speed(axis1_speed=50, axis2_speed=50, axis3_speed=50)
    # Change Robot Arm Axis-1 Speed
    Dynamixel.write_word(@axis1_Motor_ID  , 32, (axis1_speed).round)

    # Change Robot Arm Axis-2 Speed
    Dynamixel.write_word(@axis2_R_Motor_ID, 32, (axis2_speed).round)
    Dynamixel.write_word(@axis2_L_Motor_ID, 32, (axis2_speed).round)

    # Change Robot Arm Axis-3 Speed
    Dynamixel.write_word(@axis3_R_Motor_ID, 32, (axis3_speed).round)
    Dynamixel.write_word(@axis3_L_Motor_ID, 32, (axis3_speed).round)
  end

  def drive_robot_arm(theta1, theta2, theta3)
    # Drive Axis-1
    Dynamixel.write_word(@axis1_Motor_ID  , 30, (     (theta1+ 60)*3.41).round)  # (theta+60)*(1023/300)

    # Drive Axis-2
    Dynamixel.write_word(@axis2_R_Motor_ID, 30, (     (theta2+150)*3.41).round)  # (theta+150)*(1023/300)
    Dynamixel.write_word(@axis2_L_Motor_ID, 30, (1023-(theta2+150)*3.41).round)

    # Drive Axis-3
    Dynamixel.write_word(@axis3_R_Motor_ID, 30, (     (theta3+150)*3.41).round)
    Dynamixel.write_word(@axis3_L_Motor_ID, 30, (1023-(theta3+150)*3.41).round)
  end

  def calculate_inverse_kinematics(p_x, p_y, p_z)
    begin
      # Note: theta1 can not be "0" and "180"
```

```
    (p_y = 1e-5) if (p_y == 0)

    # Inverse Kinematics
    theta1 = atan2(p_y, p_x)
    theta3 = asin( ((p_y/sin(theta1))**2+(p_z-@d_1)**2-(@a_2**2+@a_3**2+@d_4**2))/
             (2*@a_2*sqrt(@a_3**2+@d_4**2)) )-atan(@a_3/@d_4)
    theta2 = ( asin( (p_z-@d_1)/sqrt((@a_2+@a_3*cos(theta3)+@d_4*sin(theta3))**2+
             (@a_3*sin(theta3)-@d_4*cos(theta3))**2) )-atan2((@a_3*sin(theta3)-
             @d_4*cos(theta3)), (@a_2+@a_3*cos(theta3)+@d_4*sin(theta3))) ) - (PI/2)

    # Convert to Degree
    theta1 *= (180/PI)
    theta2 *= (180/PI)
    theta3 *= (180/PI)

    # Solve The Problem of The Third Quadrant
    ( theta1 += Float(360) ) if (-180 < theta1) and (theta1 < -90)

    return [theta1, theta2, theta3]

  rescue Exception
    STDERR.puts "-WARNING- Out of Range"
    return :ERROR
  end
end

def mv_to(p_x=@init_x, p_y=@init_y, p_z=@init_z)
  # Calculate the Inverse Kinematics
  theta = calculate_inverse_kinematics(p_x, p_y, p_z)

  # Drive the robot arm
  drive_robot_arm(*theta) if (theta != :ERROR)
end
```

國立勤益科技大學 National Chin-Yi University of Technology 勤益 NCUT

```
def rotate_gripper(theta4=0)  # Unit: Degree, Range: 150 ~ -150 degree
  Dynamixel.write_word(@axis4_Motor_ID, 30, ((theta4+150)*3.41).round)  # 3.41 = 1023 / 300
end

def ch_rotate_gripper_speed( rotate_gripper_speed = 50 )
  Dynamixel.write_word(@axis4_Motor_ID, 32, rotate_gripper_speed.round)
end

def mv_gripper(dist=45)  # Unit: cm, Range: 5 ~ 85 mm
  Dynamixel.write_word(@gripper_L_Motor_ID, 30, (512+40-(dist-5)).round)
  Dynamixel.write_word(@gripper_R_Motor_ID, 30, (512+(dist-5)-40).round)
end

def gripper_open
  Dynamixel.write_word(@gripper_L_Motor_ID, 30, 472)
  Dynamixel.write_word(@gripper_R_Motor_ID, 30, 552)
  @gripper_status = :open
end

def gripper_close
  Dynamixel.write_word(@gripper_L_Motor_ID, 30, 560)
  Dynamixel.write_word(@gripper_R_Motor_ID, 30, 464)
  @gripper_status = :close
end

def gripper_stop
  left_gripper_value  = Dynamixel.read_word(@gripper_L_Motor_ID, 36)
  right_gripper_value = Dynamixel.read_word(@gripper_R_Motor_ID, 36)
  Dynamixel.write_word(@gripper_L_Motor_ID, 30, left_gripper_value)
  Dynamixel.write_word(@gripper_R_Motor_ID, 30, right_gripper_value)
  @gripper_status = :stop
end

def ch_gripper_speed( gripper_speed = 50 )
```

```
    Dynamixel.write_word(@gripper_L_Motor_ID, 32, gripper_speed.round)
    Dynamixel.write_word(@gripper_R_Motor_ID, 32, gripper_speed.round)
  end

  def ch_gripper_torque( percentage_of_torque = 10 )
    if ( percentage_of_torque > 0 ) and ( percentage_of_torque <= 100 )
      output = (percentage_of_torque/100.0*1024-1).round
    else
      output = 0
    end
    Dynamixel.write_word(@gripper_L_Motor_ID, 34, output)
    Dynamixel.write_word(@gripper_R_Motor_ID, 34, output)
  end

  def gripper_moving?
    if ((Dynamixel.read_word(@gripper_L_Motor_ID, 46)==0) or
        (Dynamixel.read_word(@gripper_R_Motor_ID, 46)==0))
      return false
    else
      return true
    end
  end

  def terminate
    Dynamixel.terminate()
  end
end
```

附錄六 ：Dynamixel C API Port to Ruby (Ruby C Ext.)

```
/*-----------------------------------------------
#  Extension Name : Dynamixel
#  Created by Louis Smith on Oct. 10, 2011.
-----------------------------------------------*/

#if defined(__cplusplus)
extern "C" {
#endif

#include <stdio.h>
#include <stdlib.h>
#include <ruby.h>
#include <dynamixel.h>

static VALUE cDynamixel;

/* Device Control Methods */
static VALUE dynamixel_initialize( VALUE self, VALUE devIndex, VALUE baudnum ) {
     return INT2FIX( dxl_initialize( FIX2INT(devIndex), FIX2INT(baudnum) ) );
}

static VALUE dynamixel_terminate( VALUE self ) {
     dxl_terminate();
     return Qnil;
}

/* Set/Get Packet Methods */
static VALUE dynamixel_set_txpacket_id( VALUE self, VALUE id ) {
     dxl_set_txpacket_id( FIX2INT(id) );
     return Qnil;
}
```

國立勤益科技大學 National Chin-Yi University of Technology 勤益 NCUT

```
static VALUE dynamixel_set_txpacket_instruction( VALUE self, VALUE instruction ) {
     dxl_set_txpacket_instruction( FIX2INT(instruction) );
     return Qnil;
}

static VALUE dynamixel_set_txpacket_parameter( VALUE self, VALUE index, VALUE value ) {
     dxl_set_txpacket_parameter( FIX2INT(index), FIX2INT(value) );
     return Qnil;
}

static VALUE dynamixel_set_txpacket_length( VALUE self, VALUE length ) {
     dxl_set_txpacket_length( FIX2INT(length) );
     return Qnil;
}

static VALUE dynamixel_get_rxpacket_error( VALUE self, VALUE errbit ) {
     return INT2FIX( dxl_get_rxpacket_error( FIX2INT(errbit) ) );
}

static VALUE dynamixel_get_rxpacket_parameter( VALUE self, VALUE index ) {
     return INT2FIX( dxl_get_rxpacket_parameter( FIX2INT(index) ) );
}

static VALUE dynamixel_get_rxpacket_length( VALUE self ) {
     return INT2FIX( dxl_get_rxpacket_length() );
}

/* Utility for Value */
static VALUE dynamixel_makeword( VALUE self, VALUE lowbyte, VALUE highbyte ) {
     return INT2FIX( dxl_makeword( FIX2INT(lowbyte), FIX2INT(highbyte) ) );
}

static VALUE dynamixel_get_lowbyte( VALUE self, VALUE word ) {
     return INT2FIX( dxl_get_lowbyte( FIX2INT(word) ) );
```

```
}

static VALUE dynamixel_get_highbyte( VALUE self, VALUE word ) {
     return INT2FIX( dxl_get_highbyte( FIX2INT(word) ) );
}

/* Packet Communication Methods */
static VALUE dynamixel_tx_packet( VALUE self ) {
     dxl_tx_packet();
     return Qnil;
}

static VALUE dynamixel_rx_packet( VALUE self ) {
     dxl_rx_packet();
     return Qnil;
}

static VALUE dynamixel_txrx_packet( VALUE self ) {
     dxl_txrx_packet();
     return Qnil;
}

static VALUE dynamixel_get_result( VALUE self ) {
     return INT2FIX( dxl_get_result() );
}

/* High Communication Methods */
static VALUE dynamixel_ping( VALUE self, VALUE id ) {
     dxl_ping( FIX2INT(id) );
     return Qnil;
}

static VALUE dynamixel_read_byte( VALUE self, VALUE id, VALUE address ) {
     return INT2FIX( dxl_read_byte( FIX2INT(id), FIX2INT(address) ) );
```

國立勤益科技大學 National Chin-Yi University of Technology 勤益 NCUT

```
}

static VALUE dynamixel_write_byte( VALUE self, VALUE id, VALUE address, VALUE value ) {
     dxl_write_byte( FIX2INT(id), FIX2INT(address), FIX2INT(value) );
     return Qnil;
}

static VALUE dynamixel_read_word( VALUE self, VALUE id, VALUE address ) {
     return INT2FIX( dxl_read_word( FIX2INT(id), FIX2INT(address) ) );
}

static VALUE dynamixel_write_word( VALUE self, VALUE id, VALUE address, VALUE value ) {
     dxl_write_word( FIX2INT(id), FIX2INT(address), FIX2INT(value) );
     return Qnil;
}

/* Ruby Define ---------------------------------------------------------------------------------------- */
void Init_Dynamixel() {
     /* Ruby Class */
     cDynamixel = rb_define_class("Dynamixel", rb_cObject);

     /* Constant */
     rb_define_const(cDynamixel, "BROADCAST_ID"     , INT2FIX(BROADCAST_ID));
     rb_define_const(cDynamixel, "INST_PING"        , INT2FIX(INST_PING));
     rb_define_const(cDynamixel, "INST_READ"        , INT2FIX(INST_READ));
     rb_define_const(cDynamixel, "INST_WRITE"       , INT2FIX(INST_WRITE));
     rb_define_const(cDynamixel, "INST_REG_WRITE"   , INT2FIX(INST_REG_WRITE));
     rb_define_const(cDynamixel, "INST_ACTION"      , INT2FIX(INST_ACTION));
     rb_define_const(cDynamixel, "INST_RESET"       , INT2FIX(INST_RESET));
     rb_define_const(cDynamixel, "INST_SYNC_WRITE"  , INT2FIX(INST_SYNC_WRITE));
     rb_define_const(cDynamixel, "MAXNUM_TXPARAM"   , INT2FIX(MAXNUM_TXPARAM));
     rb_define_const(cDynamixel, "MAXNUM_RXPARAM"   , INT2FIX(MAXNUM_RXPARAM));
     rb_define_const(cDynamixel, "ERRBIT_VOLTAGE"   , INT2FIX(ERRBIT_VOLTAGE));
     rb_define_const(cDynamixel, "ERRBIT_ANGLE"     , INT2FIX(ERRBIT_ANGLE));
```

```
    rb_define_const(cDynamixel, "ERRBIT_OVERHEAT"   , INT2FIX(ERRBIT_OVERHEAT));
    rb_define_const(cDynamixel, "ERRBIT_RANGE"      , INT2FIX(ERRBIT_RANGE));
    rb_define_const(cDynamixel, "ERRBIT_CHECKSUM"   , INT2FIX(ERRBIT_CHECKSUM));
    rb_define_const(cDynamixel, "ERRBIT_OVERLOAD"   , INT2FIX(ERRBIT_OVERLOAD));
    rb_define_const(cDynamixel, "ERRBIT_INSTRUCTION", INT2FIX(ERRBIT_INSTRUCTION));
    rb_define_const(cDynamixel, "COMM_TXSUCCESS"    , INT2FIX(COMM_TXSUCCESS));
    rb_define_const(cDynamixel, "COMM_RXSUCCESS"    , INT2FIX(COMM_RXSUCCESS));
    rb_define_const(cDynamixel, "COMM_TXFAIL"       , INT2FIX(COMM_TXFAIL));
    rb_define_const(cDynamixel, "COMM_RXFAIL"       , INT2FIX(COMM_RXFAIL));
    rb_define_const(cDynamixel, "COMM_TXERROR"      , INT2FIX(COMM_TXERROR));
    rb_define_const(cDynamixel, "COMM_RXWAITING"    , INT2FIX(COMM_RXWAITING));
    rb_define_const(cDynamixel, "COMM_RXTIMEOUT"    , INT2FIX(COMM_RXTIMEOUT));
    rb_define_const(cDynamixel, "COMM_RXCORRUPT"    , INT2FIX(COMM_RXCORRUPT));

    /* Class Method */
    rb_define_module_function(cDynamixel, "initialize"             , RUBY_METHOD_FUNC(dynamixel_initialize             ), 2);
    rb_define_module_function(cDynamixel, "terminate"              , RUBY_METHOD_FUNC(dynamixel_terminate              ), 0);
    rb_define_module_function(cDynamixel, "set_txpacket_id"        , RUBY_METHOD_FUNC(dynamixel_set_txpacket_id        ), 1);
    rb_define_module_function(cDynamixel, "set_txpacket_instruction", RUBY_METHOD_FUNC(dynamixel_set_txpacket_instruction), 1);
    rb_define_module_function(cDynamixel, "set_txpacket_parameter"  , RUBY_METHOD_FUNC(dynamixel_set_txpacket_parameter  ), 2);
    rb_define_module_function(cDynamixel, "set_txpacket_length"     , RUBY_METHOD_FUNC(dynamixel_set_txpacket_length     ), 1);
    rb_define_module_function(cDynamixel, "get_rxpacket_error"      , RUBY_METHOD_FUNC(dynamixel_get_rxpacket_error      ), 1);
    rb_define_module_function(cDynamixel, "get_rxpacket_parameter"  , RUBY_METHOD_FUNC(dynamixel_get_rxpacket_parameter  ), 1);
    rb_define_module_function(cDynamixel, "get_rxpacket_length"     , RUBY_METHOD_FUNC(dynamixel_get_rxpacket_length     ), 0);
    rb_define_module_function(cDynamixel, "makeword"               , RUBY_METHOD_FUNC(dynamixel_makeword               ), 2);
    rb_define_module_function(cDynamixel, "get_lowbyte"            , RUBY_METHOD_FUNC(dynamixel_get_lowbyte            ), 1);
    rb_define_module_function(cDynamixel, "get_highbyte"           , RUBY_METHOD_FUNC(dynamixel_get_highbyte           ), 1);
    rb_define_module_function(cDynamixel, "tx_packet"              , RUBY_METHOD_FUNC(dynamixel_tx_packet              ), 0);
    rb_define_module_function(cDynamixel, "rx_packet"              , RUBY_METHOD_FUNC(dynamixel_rx_packet              ), 0);
    rb_define_module_function(cDynamixel, "txrx_packet"            , RUBY_METHOD_FUNC(dynamixel_txrx_packet            ), 0);
    rb_define_module_function(cDynamixel, "get_result"             , RUBY_METHOD_FUNC(dynamixel_get_result             ), 0);
    rb_define_module_function(cDynamixel, "ping"                   , RUBY_METHOD_FUNC(dynamixel_ping                   ), 1);
    rb_define_module_function(cDynamixel, "read_byte"              , RUBY_METHOD_FUNC(dynamixel_read_byte              ), 2);
    rb_define_module_function(cDynamixel, "write_byte"             , RUBY_METHOD_FUNC(dynamixel_write_byte             ), 3);
```

```
    rb_define_module_function(cDynamixel, "read_word"              , RUBY_METHOD_FUNC(dynamixel_read_word              ), 2);
    rb_define_module_function(cDynamixel, "write_word"             , RUBY_METHOD_FUNC(dynamixel_write_word             ), 3);
}

#if defined(__cplusplus)
}
#endif
```

附錄七 ：Force Sensor Server Source Code (Ruby)

```
#! /usr/bin/env ruby
#  encoding: UTF-8
#  Program name : Force Sensor Server (NI-DAQ)
#  Created by Louis Smith on May 9, 2011

# [ Add Load Path  ] #-------------------------------------------#
$LOAD_PATH.push( File.dirname(__FILE__) + '/../lib' )

# [ Import Library ] #-------------------------------------------#
require "NIDAQ"
require "socket"

force_sensor_server = TCPServer.new('127.0.0.1', 23125)
daq = NIDAQ.new("Dev1/ai0", 0, 5)
puts '- Force Sensor Server : Created -'

begin
  client = force_sensor_server.accept
  daq.start_task
  loop {
    # If Client Request Sensor Data then Send a 32-Bits(4-Bytes) Float Number Back
    client.print( [daq.get_ai_data].pack('F') ) if client.getc == 'r'
  }
rescue Exception
  puts '- Failure -'
  client.close
  sleep(1)
  retry
ensure
  force_sensor_server.close if force_sensor_server
  daq.stop_task
end
```

附錄八 ：NI-DAQ Ruby C Ext. Source Code (Ruby C Ext.)

```
/*-------------------------------------------
#  Extension Name : NI-DAQ Analog Input
#  Created by Louis Smith on Oct. 6, 2011.
-------------------------------------------*/

#if defined(__cplusplus)
extern "C" {
#endif

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ruby.h>
#include <NIDAQmx.h>

#define DAQmxErrChk(functionCall) { if( DAQmxFailed(error=(functionCall)) ) { nidaq_error( taskHandle ); } }
#define BufferSize 1000

long int error = 0;
static VALUE cNIDAQ;

// Error Handler ------------------------------------------------------------------------
static void nidaq_error( TaskHandle taskHandle ) {
     char errBuff[2048] = {'\0'};

     if( DAQmxFailed(error) ) {
          DAQmxGetExtendedErrorInfo( errBuff, 2048 );
          DAQmxStopTask( taskHandle );
          rb_raise( rb_eRuntimeError, "DAQmx Error %d: %s\n", error, errBuff );
     }
}
```

```
// Analog Input ----------------------------------------------------------------------------
static double arrayAverage( const double data[], const long int size ) {
    double avg = 0;
    long int i;

    for(i = 0; i<size; ++i) {
        avg += data[i];
    }
    avg = avg / (double)size;

    return avg;
}

static void nidaq_free( void *taskHandle ) {
     /* Close the Task */
     DAQmxClearTask( taskHandle );
}

static VALUE nidaq_alloc( VALUE klass ) {
     VALUE obj;
     TaskHandle taskHandle = 0;

     DAQmxErrChk( DAQmxCreateTask("", &taskHandle) );
     obj = Data_Wrap_Struct( klass, 0, nidaq_free, taskHandle );

     return obj;
}

static VALUE nidaq_initialize( VALUE self, VALUE ch, VALUE minVolt, VALUE maxVolt ) {
     /* Channel parameters (Convert Ruby Object to C Data Types) */
     char  *channel    = RSTRING_PTR(ch) ;  // String
     double minVoltage = NUM2DBL(minVolt);  // Numeric
     double maxVoltage = NUM2DBL(maxVolt);  // Numeric
```

```
    TaskHandle taskHandle = 0;
    Data_Get_Struct( self, TaskHandle, taskHandle );
    DAQmxErrChk( DAQmxCreateAIVoltageChan( taskHandle,
                                           channel,
                                           "",
                                           DAQmx_Val_Diff,
                                           minVoltage,
                                           maxVoltage,
                                           DAQmx_Val_Volts,
                                           NULL ));

    return self;
}

static VALUE nidaq_start_task( VALUE self ) {
    TaskHandle taskHandle = 0;
    Data_Get_Struct( self, TaskHandle, taskHandle );

    /* Start the Task */
    DAQmxErrChk( DAQmxStartTask( taskHandle ) );

    return Qnil;
}

static VALUE nidaq_stop_task( VALUE self ) {
    TaskHandle taskHandle = 0;
    Data_Get_Struct( self, TaskHandle, taskHandle );

    /* Stop the Task */
    DAQmxStopTask( taskHandle );

    return Qnil;
}
```

```
static VALUE nidaq_get_analog_input( VALUE self ) {
     // Timing parameters
     uInt32 samplesPerChan = BufferSize;  // The size of the array, in samples, into which samples are read.

     // Data read parameters
     double   data[BufferSize];           // The array to read samples into.
     long int pointsToRead = BufferSize;  // The number of samples, per channel, to read.
     long int pointsRead;                 // The actual number of samples read from each channel.
     double   timeout = 10.0;

     VALUE result;
     TaskHandle taskHandle = 0;
     Data_Get_Struct(self, TaskHandle, taskHandle);

     DAQmxErrChk( DAQmxReadAnalogF64( taskHandle,
                                      pointsToRead,
                                      timeout,
                                      DAQmx_Val_GroupByChannel,
                                      data,
                                      samplesPerChan,
                                      &pointsRead,
                                      NULL ));

     result = rb_float_new( arrayAverage(data, pointsRead) );

     if ( rb_block_given_p() ) {
          result = rb_yield( result );
     }

     return result;
}

// Ruby Define ----------------------------------------------------------------------------
void Init_NIDAQ() {
```

```
    /* Analog Input */
    cNIDAQ = rb_define_class("NIDAQ", rb_cObject);
    rb_define_alloc_func(cNIDAQ, nidaq_alloc);
    rb_define_method(cNIDAQ, "initialize" , RUBY_METHOD_FUNC(nidaq_initialize      ), 3);
    rb_define_method(cNIDAQ, "start_task" , RUBY_METHOD_FUNC(nidaq_start_task      ), 0);
    rb_define_method(cNIDAQ, "stop_task"  , RUBY_METHOD_FUNC(nidaq_stop_task       ), 0);
    rb_define_method(cNIDAQ, "get_ai_data", RUBY_METHOD_FUNC(nidaq_get_analog_input), 0);
}

#if defined(__cplusplus)
}
#endif
```

國立勤益科技大學 National Chin-Yi University of Technology 勤益 NCUT

附錄九 ：Kinect Sensor Server Source Code (C/C++)

```
/*----------------------------------------
   Program name : Kinect Sensor Server
   Created by Louis Smith on May 8, 2011.
----------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>
#include <XnCppWrapper.h>
#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>

#pragma comment (lib, "Ws2_32.lib")

#define DEFAULT_BUFLEN      512
#define DEFAULT_IP_ADDRESS "127.0.0.1"
#define DEFAULT_PORT       "23124"
#define XPIXELS             640  /* XN_VGA_X_RES */
#define YPIXELS             480  /* XN_VGA_Y_RES */
#define FPS                  30
#define GESTURE_TO_USE     "RaiseHand"
#define SMOOTHING_FACTOR      0  /* Smoothing factor, in the range 0..1.
                                    Inside the range is generator dependent.
                                     -> 0 Means no smoothing
                                     -> 1 means infinite smoothing.         */

// Global Variables //-------------------------------------------------------
// Universal //-------------------------------------------------------
XnFloat nearestPointDepth = 0;

// Kinect //-------------------------------------------------------
```

```
XnStatus eResult = XN_STATUS_OK;
xn::Context          mContext;           // Create Context
xn::DepthGenerator   mDepthGenerator;    // Create Depth   Generator
xn::GestureGenerator mGestureGenerator;  // Create Gesture Generator
xn::HandsGenerator   mHandsGenerator;    // Create Hand    Generator

// Server //-----------------------------------------------------------
int  iResult;
int  iSendResult;
char sendbuf[DEFAULT_BUFLEN];  // Send Buffer
char recvbuf[DEFAULT_BUFLEN];  // Recv Buffer
int  recvbuflen = DEFAULT_BUFLEN;
WSADATA wsaData;
SOCKET ListenSocket = INVALID_SOCKET;
SOCKET ClientSocket = INVALID_SOCKET;
struct addrinfo *result = NULL;
struct addrinfo hints;

// ERROR Check Function //---------------------------------------------------
void CheckOpenNIError( XnStatus eResult, std::string sStatus ) {
      if( eResult != XN_STATUS_OK ) {
          std::cerr << sStatus << " Error : " << xnGetStatusString( eResult ) << std::endl;
     }
}

// Find Nearest Point
XnUInt32 findNearestPoint( const XnDepthPixel *pDepthMap ) {
     int x, y, index, depth;
     int min = 10000;  // 10 Meter
     XnUInt32 nearestPointIndex = 0;

     for (y=0; y<YPIXELS; ++y) {
          for (x=0; x<XPIXELS; ++x) {
               index = y*XPIXELS+x;
```

```
                depth = *( pDepthMap + index );

                if ( (depth != 0) && (min > depth) ) {
                     nearestPointIndex = index;
                     min = depth;
                }
           }
     }
     return nearestPointIndex;
}

void average( const XnFloat dataBuffer[][4], int length, XnFloat dataAvg[4] ) {
     int index;
     for( index = 0; index<length; ++index ) {
          dataAvg[0] += dataBuffer[index][0];
          dataAvg[1] += dataBuffer[index][1];
          dataAvg[2] += dataBuffer[index][2];
          dataAvg[3] += dataBuffer[index][3];
     }
     dataAvg[0] /= length;
     dataAvg[1] /= length;
     dataAvg[2] /= length;
     dataAvg[3] /= length;
}

// Kinect Callback Function //-------------------------------------------------
void XN_CALLBACK_TYPE
Gesture_Recognized(xn::GestureGenerator &generator,
                   const XnChar         *strGesture,
                   const XnPoint3D      *pIDPosition,
                   const XnPoint3D      *pEndPosition,
                   void                 *pCookie)
{
     mGestureGenerator.RemoveGesture(strGesture);
```

```
	mHandsGenerator.StartTracking(*pEndPosition);
	printf("Gesture recognized: %s\n", strGesture);
}

void XN_CALLBACK_TYPE
Gesture_Process(xn::GestureGenerator &generator,
		const XnChar		*strGesture,
		const XnPoint3D		*pPosition,
		XnFloat			 fProgress,
		void			*pCookie)
{
}

void XN_CALLBACK_TYPE
Hand_Create(xn::HandsGenerator &generator,
		XnUserID		 nId,
		const XnPoint3D		*pPosition,
		XnFloat			 fTime,
		void			*pCookie)
{
	printf("New Hand: %d @ ( X = %8.3fmm , Y = %8.3fmm , Z = %8.3fmm )\n", nId, pPosition->X, pPosition->Y, pPosition->Z);
}

void XN_CALLBACK_TYPE
Hand_Update(xn::HandsGenerator &generator,
		XnUserID		 nId,
		const XnPoint3D		*pPosition,
		XnFloat			 fTime,
		void			*pCookie)
{
	// Initialize Static Local Variables
	static XnFloat dataAvg[4] = { 0, 0, 0, 0 };
	static short int dataBufferIndex = 0;
	static XnFloat dataBuffer[6][4] = { {0, 0, 0, 0},
```

```
                                        {0, 0, 0, 0},
                                        {0, 0, 0, 0},
                                        {0, 0, 0, 0},
                                        {0, 0, 0, 0},
                                        {0, 0, 0, 0} };

dataBuffer[dataBufferIndex][0] = pPosition->X;
dataBuffer[dataBufferIndex][1] = pPosition->Y;
dataBuffer[dataBufferIndex][2] = pPosition->Z;
dataBuffer[dataBufferIndex][3] = (pPosition->Z - nearestPointDepth);

if (dataBufferIndex<6) {
     ++dataBufferIndex;
} else {
     dataBufferIndex=0;
}

average( dataBuffer, 6, dataAvg );

// Prepare data which will be sent out.
// X(mm), Y(mm), Z(mm), diff(mm)
sprintf(sendbuf, "%+05i%+05i%+05i%+05i", (int)dataAvg[0], (int)dataAvg[1], (int)dataAvg[2], (int)dataAvg[3]);

// Send Data
iSendResult = send( ClientSocket, sendbuf, (int)strlen(sendbuf), 0 );
if (iSendResult == SOCKET_ERROR) {
     printf("send failed with error: %d\n", WSAGetLastError());
     closesocket(ClientSocket);
     WSACleanup();
}

// Print Info on Screen
printf("------------------------------------------------------------\n");
printf("%s\n", sendbuf);
```

```
	printf("Bytes Sent: %d\n", iSendResult);

	Sleep(33);  // 0.033sec
}

void XN_CALLBACK_TYPE
Hand_Destroy(xn::HandsGenerator &generator,
             XnUserID            nId,
             XnFloat             fTime,
             void               *pCookie)
{
	mGestureGenerator.AddGesture(GESTURE_TO_USE, NULL);
	printf("Lost Hand: %d\n", nId);
}

// Main Program //-------------------------------------------------------
int main( int argc, char **argv )
{
	// Server //-----------------------------------------------------
	// Initialize Winsock
	iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
	if (iResult != 0) {
		printf("WSAStartup failed with error: %d\n", iResult);
		return 1;
	}

	ZeroMemory(&hints, sizeof(hints));
	hints.ai_family   = AF_INET;
	hints.ai_socktype = SOCK_STREAM;
	hints.ai_protocol = IPPROTO_TCP;
	hints.ai_flags    = AI_PASSIVE;

	// Resolve the server address and port
	iResult = getaddrinfo(DEFAULT_IP_ADDRESS, DEFAULT_PORT, &hints, &result);
```

```
    if ( iResult != 0 ) {
        printf("getaddrinfo failed with error: %d\n", iResult);
        WSACleanup();
        return 1;
    }

    // Create a SOCKET for connecting to server
    ListenSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (ListenSocket == INVALID_SOCKET) {
        printf("socket failed with error: %ld\n", WSAGetLastError());
        freeaddrinfo(result);
        WSACleanup();
        return 1;
    }

    // Setup the TCP listening socket
    iResult = bind( ListenSocket, result->ai_addr, (int)result->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
        printf("bind failed with error: %d\n", WSAGetLastError());
        freeaddrinfo(result);
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
    }

    freeaddrinfo(result);

    iResult = listen(ListenSocket, SOMAXCONN);
    if (iResult == SOCKET_ERROR) {
        printf("listen failed with error: %d\n", WSAGetLastError());
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
    }
```

```
// Accept a client socket
ClientSocket = accept(ListenSocket, NULL, NULL);
if (ClientSocket == INVALID_SOCKET) {
     printf("accept failed with error: %d\n", WSAGetLastError());
     closesocket(ListenSocket);
     WSACleanup();
     return 1;
}

// No longer need server socket
closesocket(ListenSocket);
// Kinect //-------------------------------------------------------
// Initialize Context Object
eResult = mContext.Init();
CheckOpenNIError( eResult, "Initialize Context" );

// Create a Depth Generator Node
eResult = mDepthGenerator.Create( mContext );
CheckOpenNIError( eResult, "Create depth generator" );

// Map Output Mode
XnMapOutputMode mapMode;
mapMode.nXRes = XPIXELS;
mapMode.nYRes = YPIXELS;
mapMode.nFPS  = FPS;
eResult = mDepthGenerator.SetMapOutputMode( mapMode );
CheckOpenNIError( eResult, "Setup Depth Generator Output Mode" );

// Create the Gesture and Hands Generators
eResult = mGestureGenerator.Create( mContext );
CheckOpenNIError( eResult, "Create gesture generators" );
eResult = mHandsGenerator.Create( mContext );
CheckOpenNIError( eResult, "Create hands generators" );
```

```
// Setup Gesture and Hands Generator
mGestureGenerator.AddGesture( GESTURE_TO_USE, NULL );
mHandsGenerator.SetSmoothing( SMOOTHING_FACTOR );

// Register Callback Functions //
// Create Callback Handles
XnCallbackHandle hGestureCB, hHandCB;

// Setup Gesture Generator Callback Function
mGestureGenerator.RegisterGestureCallbacks(Gesture_Recognized, Gesture_Process, NULL, hGestureCB);

// Setup Hand Generator Callback Function
mHandsGenerator.RegisterHandCallbacks(Hand_Create, Hand_Update, Hand_Destroy, NULL, hHandCB);

// Start Generate Data //
mContext.StartGeneratingAll();

while (TRUE) {
     // Update Frame Data
     eResult = mContext.WaitAndUpdateAll();

     if ( eResult == XN_STATUS_OK ) {
          // Get current depth map.
          const XnDepthPixel* pDepthMap = mDepthGenerator.GetDepthMap();
          XnUInt32 nearestPointIndex = findNearestPoint( pDepthMap );

          // Update global variable of the nearest point.
          nearestPointDepth = pDepthMap[nearestPointIndex];
     }
}

// Stop and Shutdown //-----------------------------------------
// Server //----------------------------------------------------
```

```
    // shutdown the connection since we're done
    iResult = shutdown(ClientSocket, SD_SEND);
    if (iResult == SOCKET_ERROR) {
        printf("shutdown failed with error: %d\n", WSAGetLastError());
        closesocket(ClientSocket);
        WSACleanup();
        return 1;
    }
    // clean-up
    closesocket(ClientSocket);
    WSACleanup();

    // Kinect //-----------------------------------------------------
    // Stop and clean-up
    mContext.StopGeneratingAll();
    mContext.Release();
    return 0;
}
```

國立勤益科技大學 National Chin-Yi University of Technology 勤益 NCUT

附錄十 ：如何撰寫 Ruby 的 C 擴充

基本上想要撰寫 Ruby 的 C 擴充有三種主要的方式，如表 A.1 所示，其中 RubyInline、SWIG 是相較於 Ruby C API 來說較為特別的方式。RubyInline 可以讓你在 Ruby 的程式碼裡面直接插進 C 的程式碼，雖然很便利但卻容易造成程式碼的雜亂與混淆。SWIG 則是一套軟體開發工具，可以幫助使用者快速的移植 C/C++ 的程式碼到許多高階語言上(例如: Ruby、Python、Perl、PHP、Tcl 等)，雖然 SWIG 很方便但也容易破壞 Ruby 語法的一致性，很容易就會讓人感覺是在寫 C/C++的程式而不是在寫 Ruby，且產生出來的 C 擴充檔案大小相較於其它方法也較為大了些。使用 Ruby 本身提供的 C API 則是較為正統的做法，但相對來說在撰寫上也較其它兩者繁瑣，好處是跟 Ruby 本身的整合度較高且執行效率佳，所以在此將使用 Ruby 的 C API 來示範如何撰寫一個 Ruby 的 C 擴充。

表 A.1 撰寫 Ruby C 擴充的三種主要方式

Ruby C API	Since the Ruby interpreter is implemented in C, its API can be used.
RubyInline	Supports mixing C code into Ruby code.
SWIG	SWIG is typically used to parse C/C++ interfaces and generate the 'glue code' required for the above target languages to call into the C/C++ code.

在這裡所要寫的擴充範例是一個叫做 MyBank 的類別，其作用如同銀行一樣可以存錢、觀看帳戶資料(使用者名稱與剩餘金額)。因此若想寫出如圖 A.1 所示的 Ruby C 擴充(MyBook)需要做到兩件事，1.撰寫 C 擴充的 C/C++程式碼，並在程式碼內部定義類別與相關方法等之間的關係(依照設計時的介面來定義)、2.撰寫 extconf.rb 檔案，描述編譯檔案時需要用到的資料，讓程式自動產生 makefile 方便我們產生 C 擴充。

Ruby Code (Example.rb)

```
require 'MyBank'    # C Extension

my_bank = MyBank.new('Louis', 1000)
my_bank.get_user_name       # => Return Value: Louis
my_bank.get_user_money      # => Return Value:1000
my_bank.save_money(500)     # => Return Value:1500
my_bank.get_user_money      # => Return Value: 1500
```

圖 A.1 MyBank 的 Ruby 範例程式碼

首先來看到如何撰寫 Ruby C 擴充的 C/C++程式碼。在 Ruby 的 C API 裡是以 VALUE 這個型別來表示 Ruby 的物件，因此不管是參數或是返回值，型別都一律是 VALUE，且由於 Ruby 是純物件導向的語言，故每個 Ruby 方法的定義都要有回傳值，若只是單純計算而沒有東西可以回傳時則可回傳 Qnil 回去(即 return Qnil;)代表沒東西。參數中的 self 則是方法的呼叫者通常為物件本身，這也是必須要的參數。撰寫 Ruby C 擴充還有另一件要注意的事就是型別轉換，舉例來說，若參數傳遞一個整數進來，則必須將 VALUE 型別透過 Ruby C API 所提供的巨集轉換成 C 的 int 型別才能在 C 裡面運算，當運算完後要回傳時則須要把 C 的 int 型別再次透過 Ruby C API 所提供的巨集，轉換回 VALUE 型別才能回傳回去。當所有的 Ruby 方法都定義完成後，便可在最後定義一個 Init_開頭的函數，並在裡面定義 MyBank 這個類別的相關方法、常數等各項關係，這樣就完成了我們的 C 擴充原始碼。MyBook 的 C 擴充原始碼內容如圖 A.2 所示。

Ruby C Extension (filename:MyBank.c)

```
#include <ruby.h>

static VALUE my_bank_initialize( VALUE self, VALUE name, VALUE money ) {
    rb_iv_set(self, "@name", name);
```

```
    rb_iv_set(self, "@money", money);
    return self;
}

static VALUE my_bank_get_user_name( VALUE self ) {
    return rb_iv_get(self, "@name");
}

static VALUE my_bank_get_user_money( VALUE self ) {
    return rb_iv_get(self, "@money");
}

static VALUE my_bank_save_money( VALUE self, VALUE income ) {
    VALUE money;
    long int sum;
    money = rb_iv_get(self, "@money");
    sum = FIX2INT(money) + FIX2INT(income);
    rb_iv_set(self, "@money", INT2FIX(sum));
    return INT2FIX(sum);
}

void Init_MyBank() {
    static VALUE cMyBank;
    cMyBank = rb_define_class("MyBank", rb_cObject);
    rb_define_method(cMyBank, "initialize", my_bank_initialize, 2);
    rb_define_method(cMyBank, "get_user_name", my_bank_get_user_name, 0);
    rb_define_method(cMyBank, "get_user_money", my_bank_get_user_money, 0);
    rb_define_method(cMyBank, "save_money", my_bank_save_money, 1);
}
```

圖 A.2 MyBank 的 Ruby C 擴充原始碼

最後在撰寫 extconf.rb 檔案，如圖 A.3 所示。並將剛剛完成的 Ruby C 擴充原始碼(MyBank.c)、extconf.rb 檔案放在同一個目錄下。

Ruby Code (filename:extconf.rb)

```
# Make makefile
require 'mkmf'
extension_name = "MyBank"
create_makefile(extension_name)
```

圖 A.3 MyBank 的 extconf.rb 檔案原始碼

再打開指令視窗並移動到放置檔案的目錄下，依序執行'***ruby ./extconf.rb***'、'***make***'兩個指令即可產生 MyBank.so 這個檔案於目錄下，該檔案即為我們所要的 C 擴充。指令列操作如圖 A.4 所示。而要使用此 C 擴充於 Ruby 內只需要像標準程式庫一樣使用 require 即可(require 'MyBank')，這樣便完成了我們的 Ruby C 擴充。

Command Line

```
# ruby ./extconf.rb
creating Makefile

# make
compiling MyBank,c
linking shared-object MyBank.so
```

圖 A.4 編譯 C 擴充之指令列操作範例