การออกแบบระบบบนชิพสำหรับตัวควบคุมหุ่นยนต์หกแกนด้วยเอฟพีจีเอต้นทุนต่ำ

นาย อากัส บีโจ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2551

A SYSTEM ON CHIP DESIGN OF A 6-AXIS ROBOTIC ARM CONTROLLER IMPLEMENTED ON A LOW-COST FPGA

Mr. Agus Bejo

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2008

Thesis Title	A SYSTEM ON CHIP DESIGN OF A 6-AXIS ROBOTIC ARM CONTROLLER IMPLEMENTED ON A LOW-COST FPGA
By	Mr. Agus Bejo
Field of Study	Electrical Engineering
Advisor	Assistant Professor Wanchalerm Pora, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

[illegible] Dean of the Faculty of Engineering
(Associate Professor Boonsom Lerdhirunwong, Dr.Ing.)

THESIS COMMITTEE

[illegible] Chairman
(Assistant Professor Manop Wongsaisuwan, Ph.D.)

[illegible] Advisor
(Assistant Professor Wanchalerm Pora, Ph.D.)

[illegible] Examiner
(Suree Pumrin, Ph.D.)

[illegible] External Examiner
(Wisuwat Plodpradista, Ph.D.)

อากัส บีโจ : การออกแบบระบบบนชิพสำหรับตัวควบคุมหุ่นยนต์หกแกนด้วยเอฟพีจีเอต้นทุนต่ำ (A System On Chip Design of a 6-Axis Robotic Arm Controller Implemented on a Low-Cost FPGA) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร.วันเฉลิม โปรา, 101 หน้า.

วิทยานิพนธ์ฉบับนี้นำเสนอการพัฒนาตัวควบคุมแขนกลหุ่นยนต์ 6 แกน ซึ่งใช้พลังงานจากเครื่องสูบไฮโดรลิก โดยแขนกลแต่ละแกนจะถูกขับเคลื่อนด้วยลิ้นเซอร์โว ตัวควบคุมนี้สร้างขึ้นโดยใช้ FPGA ราคาต่ำ หน่วยประมวลผลหลักของตัวควบคุมรองรับอุปกรณ์รอบข้างหลายประเภท เช่น ช่องทางอนุกรม (UART) วงจรควบคุมสัญญาณนาฬิกา (Timer/Counter) ช่องทางเข้าออกอเนกประสงค์ (I/O) สัญญาณขัดจังหวะ (Interrupt) ชุดสร้างสัญญาณกล้ำพัลส์ (PWM) และช่องทาง SPI ถูกรวบรวมและออกแบบไว้ใน FPGA

วิทยานิพนธ์นี้แบ่งออกเป็น 4 ส่วน โดยส่วนแรกจะกล่าวถึงการแก้สมการผกผันจลนศาสตร์ ส่วนที่สองจะกล่าวถึงการออกแบบตัวควบคุม ส่วนที่สามจะกล่าวถึงการพัฒนาส่วนเครื่องและส่วนซอฟต์แวร์ส่วนสุดท้ายจะกล่าวถึงการพัฒนาซอฟต์แวร์เชื่อมประสานแบบกราฟิก ด้วยการกำหนดเงื่อนไขบางประการ ทำให้แก้สมการผกผันจลนศาสตร์โดยใช้วิธีแบบปิด (closed-form method) ได้ ตัวควบคุมชนิด 2-DOF PID ถูกออกแบบขึ้นโดยใช้แบบจำลองที่ได้จากวิธีการระบุแบบจำลองของระบบ วิธีการวนซ้ำกำลังสองน้อยที่สุด และวิธีการวางตำแหน่งขั้ว ถูกนำมาใช้ในการระบุแบบจำลองของระบบ และการออกแบบตัวควบคุม ส่วนเครื่องถูกสร้างขึ้นเพื่อทำหน้าที่เป็นตัวควบคุมหุ่นยนต์หกแกน ท้ายที่สุดซอฟต์แวร์เชื่อมประสานแบบกราฟิกถูกพัฒนาขึ้นสำหรับประสานการทำงานระหว่างผู้ใช้เครื่องของตัวควบคุม

ในวิทยานิพนธ์นี้ตัวควบคุมหุ่นยนต์ใหม่ถูกสร้างขึ้นเพื่อใช้แทนตัวควบคุมหุ่นยนต์เดิมซึ่งมีข้อจำกัดหลายประการเช่น หน่วยความจำน้อย การใช้งานยากลำบาก การติดตั้งไม่สะดวก และไม่สามารถติดต่อกับช่องทางสื่อสารสมัยใหม่ได้ โดยรวมแล้ว ตัวควบคุมใหม่นี้มีคุณลักษณะดีกว่าตัวควบคุมเดิม ซึ่งจะช่วยเพิ่มผลิตภาพให้ดียิ่งขึ้นได้

ภาควิชา.........วิศวกรรมไฟฟ้า.......... ลายมือชื่อนิสิต [illegible]
สาขาวิชา.........วิศวกรรมไฟฟ้า.......... ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก [illegible]
ปีการศึกษา...........2551

##507 06550 21: MAJOR ELECTRICAL ENGINEERING

KEYWORDS: ROBOTIC ARM / KINEMATICS / PID CONTROLLER DESIGN / MODELING SYSTEM / SYSTEM-ON-CHIP / FPGA

AGUS BEJO: A SYSTEM ON CHIP DESIGN OF A 6-AXIS ROBOTIC ARM CONTROLLER IMPLEMENTED ON A LOW-COST FPGA. ADVISOR: ASST. PROF. WANCHALERM PORA, Ph.D., 101 pp.

In this work, we developed a 6-axis robotic arm controller. The robotic arm is powered by hydraulic pump. At each axis its arm is driven by a servo valve. This controller is implemented on a low-cost FPGA. The main control processor supported by some peripherals such as UART, Timer/Counter, Digital I/O, Interrupt, PWM Generator, SPI and SPI Reader is embedded on a FPGA chip.

This work is composed of four parts: solving the inverse kinematics problem, designing proper controller, developing required hardwares and developing GUI user application software. With some constraints, a set of formulae is developed to solve the inverse kinematics problem using closed-form method. A 2-DOF PID controller is designed using obtained model from system identification step. The Recursive Least Square method and Pole Placement method are respectively employed for the system identification and the controller design tasks. Required hardwares are then implemented to realize the controller system. Finally, a GUI user application software is developed to interface the controller system with human operator.

Through this work, a new robotic controller system is created to replace the old existing controller system which has many constraints such as limited storage space, uneasy usage, time consuming setup and incompatibility with modern communication channel. Overall, this new controller system provides better features which is possible to increase the productivity rate.

Department:	Electrical Engineering	**Student's Signature:**	[illegible]
Field of Study:	Electrical Engineering	**Advisor's Signature:**	[illegible]
Academic Year:	2008		

Acknowledgments

Firstly, I would like to express my profound gratitude to my advisor, Asst. Prof. Dr. Wanchalerm Pora, for his patient guidance, encouragement and excellent advice as well as giving me extraordinary experiences throughout the work. He has given me abundant attention and offered invaluable assistance during my study at Chulalongkorn University. I am very sure that this thesis would not have been successful without his total supervision from the very early stage until the end of the work.

My sincere thanks to Assoc. Prof. Dr. Ekachai Leelarasmee as a former head of Embedded System and IC Design (ESID) Laboratory who give me some encouragement to accomplish my master study as well as to pursue my further study.

Special thanks in particular to Sakonpong Buranawit, Torkiat Taithongchai, Kampanard Suwannawut and all of my friends in ESID Laboratory for their kindness and very good friendship. They always help me much on keeping the spirit up, sharing the literature during the work, discussing and joking in our spare time. The unforgotten thing is that they advise me much on scheduling my working time so I can accomplish my work on time.

Another thanks is addressed to Stainless Steel Home Equipment Manufacturing Co, Ltd for their generosity of letting me use their resources for conducting my experiments. I am indebted to Chaiwat Srivongchareon, Samnoek, Manote, and other factory staffs who have introduced me to this work, guided me and accompanied me during my work. I can not imagine how to accomplish this work without their assistance and facilities support. It is a very valuable thing that help me much on finishing the work properly.

I would also like to convey many thanks to the AUN/Seed-Net scholarship program under JICA who has given me a great opportunity to pursue my advanced study abroad and provided entire financial support during my study.

Finally, I wishes to express my love and gratitude to my beloved families; for their understanding and endless love, through the duration of my study. Their moral support has brought me to an earnest work and a desire to do the best for them.

Contents

List of Tables

List of Figures

CHAPTER I

INTRODUCTION

1.1 Introduction

The term *robot* was first introduced by the Czech Karel Capek in 1920 [1]. It means an artificial humanoids - biped robots - which help human beings in physically difficult tasks [2]. Recently, the term *robot* is used to denote animated autonomous machines. Robot can be classified into two main classes, namely robot manipulators and mobile robots. This work will concern only on the robot manipulators which is defined as a computer-controlled industrial manipulator [1]. A robot manipulator is composed of links connected by joints into an open kinematic chain. Joints can be either rotary (revolute) or linear (prismatic). The number of joints determines the degrees-of-freedom (DOF). A manipulator with 6 axes posses 6 DOF, generally three for positioning and three for orientation. The joints may be actuated electrically, hydraulically or pneumatically.

Through this writing, the term of *robotic arms* will be used rather than *robot manipulators*. Robotic arms, especially the 6-axis ones, have been widespreadly used in several industrial processes [3] for many years. The robotic arm that will be emphasized in this work is a painting machine.

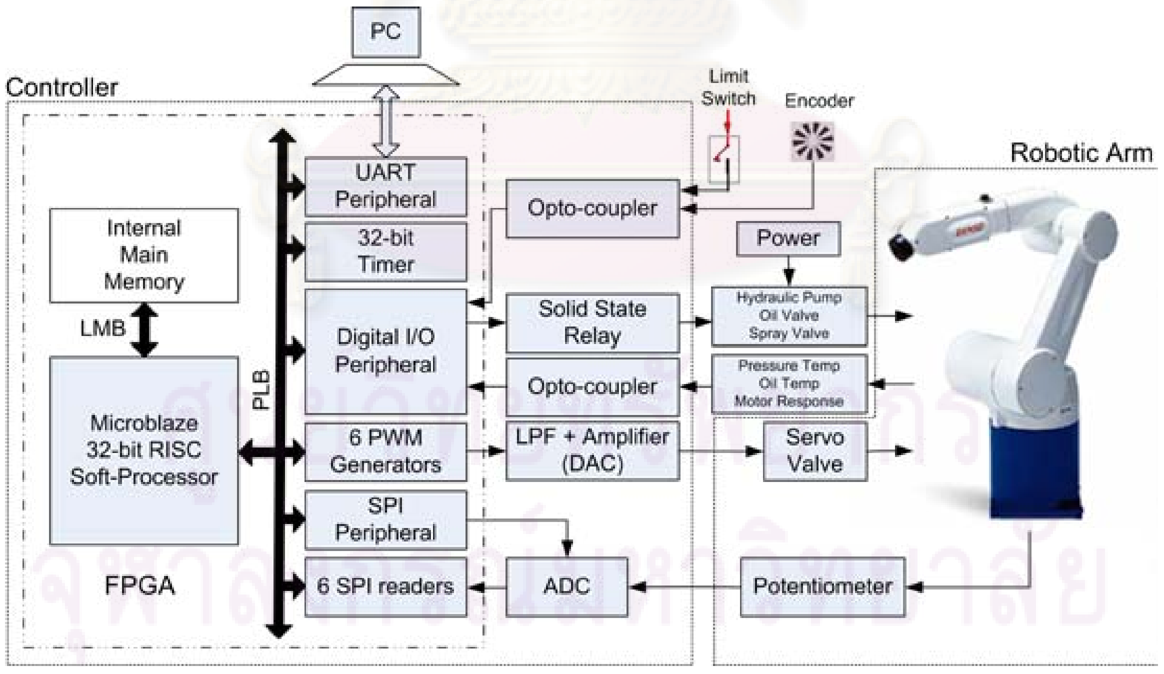


Figure 1.1: Robot system architecture

Nowadays there are many types of robotic arms that is functioned as painting machines [4–6]. Some of them have been employed for a long time. Perhaps, their service support cannot be extended. Usually, just the electronic parts of the machines which have been in service ten years or more are obsolete. On the other hands, most factory support the mechanic part of the machines themselves. It is a dilemma to choose whether to continue using the old machine with high risk and high maintenance cost due to lack of standard spare parts, or to invest in new machines which are costly. Moreover, most old machines have restrictions [7–9] such as limited storage space, incompatibility with modern technology media or communication channel.

This research was conducted to offer a solution towards some disadvantages above by designing a System-on-Chip (SoC) embedded controller based on a low-cost FPGA. The system developed in this research is shown in Figure 1.1. It is composed of a Microblaze module (32-bit RISC soft-core processor), an internal memory module, a UART module, a 32-bit timer/counter module, a digital I/O module, a SPI module, 6 custom SPI readers for interfacing ADC and 6 custom PWM generators as DACs. By using FPGA, it will improve the computation power compared with that of 8051 microcontrollers. FPGA's processing power is closed to that of DSP, however its price is much cheaper than that of DSP. The other advantages of FPGA is that it has hardware-level reconfigurable, so any module with I/O terminal can be designed, modified or adjusted very flexibly. Overall, with integrating all mentioned module on chip this yields a high-performance system.

The controller controls angular movement of each axis, both direction and velocity, by sending digital data to the DAC modules in the form of PWM signals which connected to the servo valve via low-pass filters and amplifiers. The signals that are produced by the DAC modules represent the outputs of the controller. Their polarity and magnitude respectively determines the direction and velocity of movement. The position of each axis is measured from its potentiometer that is attached at each joint. This represents the input of the controller. A PC, supported by GUI application software, can be connected to the controller system via a USB/UART to make it more interactive and user-friendly.

1.2 Literature Review

For a few decades, some researches regarding to the development of robotic arms controller have been done, and are still being done. Inverse kinematics settlement and controller design are common issues to face.

Closed-form method, that is solving the inverse kinematics analytically, gives preferable solution [3]. Nonetheless, the closed-form method requires complex calculation and rely on the robot structure. An improvement of closed-form method was proposed to reduce the computational cost by avoiding a large amount of inverse matrix multiplication [10]. Another approach of closed-form method was Product-of-Exponentials (POE) [11]. The other methods were numerical methods (nonclosed-form) such as geometrical solution [12] and iterative solution [13]. Nonclosed-form methods involve numerical iteration until a desired end-point position and orientation is reached to within a maximum allowable error. Nonclosed-form methods have some weaknesses as mentioned in chapter 2.1.3.

Apart from the kinematics problem, robotic arm controllers also have been developed in various ways, both of algorithm and its hardware. PI controller [14], PID controller [15–17] or adaptive controller [18,19] have been implemented by using PC [18], Microcontroller [17,20] and FPGA [15]. It is a bargain to choose which algorithm is suitable to solve inverse kinematics and controller regarding to the hardware requirement and the desired system performance. A new approach to improve the end-effector of robotic arm motion has been proposed by incorporating a controller algorithm such as PID [16] or MRAC [19] and an optimization algorithm such as Least Mean Square.

This work did entire mentioned issues both solving kinematics and designing controller from software up to hardware to replicate or re-design a 6-axis robotic arm controller system as a study case. In this case, a set of formulae to solve specific inverse kinematics problem was developed and 2-DOF PID controller was designed. A preliminary research to realize that system has been done by author using microcontroller [17].

1.3 Objectives

The main objective of this research is to develop a system-on-chip of a 6-axis robotic arm controller. To achieve said objective, one must fulfill the following aims:

1. Solving the inverse kinematics problem
2. Modeling the servo valves system of the robotic arm
3. Designing 2-DOF PID controller
4. Designing required hardwares
5. Implementing the controller on FPGA
6. Developing a GUI user application software

1.4 Scope of Thesis

The system designed by this research has specification as follows:

- It has an embedded PID controller
- It can communicate with PC via Serial UART/USB port
- All required modules/peripherals are integrated on chip
- It has GUI interface to control the robot through PC
- It has two mode, those are teaching mode and play back mode
- It can load or store the trajectory file from or to PC

1.5 Methodology

In this work, the inverse kinematics problem is solved analytically (closed-form method). After the desired trajectory pattern that contains the sequence of the robot movement in Cartesian coordinate had specified by an operator, corresponding file then is generated in Microsoft Excel format called trajectory file. This file is generated by solving the inverse kinematics problem with some (angular to digital value) conversions. This file is then loaded into memory of the controller system. At playback mode, this file will became the set-points of the controller to drive each position of joint so that the end-point of the robot moves to the position and orientation such the trajectory pattern had been specified.

The robot is powered by a hydraulic pump which is controlled through digital I/O connections. Its motion is driven by signals coming to the servo valves. To design proper controller, a good knowledge regarding to this servo valves must be known. The robotic servo valves are identified by using Recursive Least Square method. Some signals are given as input to the servo valves for driving the robot while the output signals are measured from the potentiometer attached in every joint. Both input and output signals are recorded and then processed by using mentioned method to obtain the model. This model represents a transfer function that states the relation between position toward input signal. Three types of input signals i.e. step, triangle and random signals are employed. The effect of the position and the load toward the model was studied. If the model does not change too much because of its load, a PID controller can be chosen as the controller. However, when the model change so much, it will be better to design an adaptive controller. By using adaptive controller, it can compensate the change of the plant model so that the controller performance kept well.

Having the servo valves model were obtained, proper 2-DOF PID controller then was designed based on the knowledge of that model. This controller with its apparatus peripheral will be implemented on FPGA. Before being implemented on FPGA, the controller will be applied by using microcontroller. This way is useful for observing the performance of the controller. If the controller performance is not good enough, it can be modified or can be improved easily without time consuming step. Eventually, after the controller performance has achieved the desired one then the controller can be implemented on FPGA. It can be said that an imitation of microcontroller system with fix features will be built on FPGA to replace the microcontroller as the main processing for the controller.

For the reason of interactivity, a GUI user application software is developed on a PC. This one can be used to re-adjust or to modify the controller system behavior such as updating the controller parameter, importing or exporting the trajectory file, updating the movement speed and so on by an operator easily.

1.6 Contributions

Through this research it is wished to yield a replicate of the old robotic arm controller which has some superiority such as:

1. Its hardware is easy to operate.

2. Its software is easy to use.
3. It can be modified to suit software needs quickly.
4. It provides better features and better capabilities.

1.7 Publications

1. Agus Bejo and Wanchalerm Pora, *Combination of Model Reference Adaptive Control and Least Mean Square Algorithms for Robotic Arm Controllers*, Asia International Symposium on Mechatronics 2008 (AISM 2008), Aug. 27-31, 2008, Hokkaido University, Japan.
2. Agus Bejo and Wanchalerm Pora, *An Improvement of The End-point Error for Multiple-axis Robotic Arms Using The LMS Algorithm*, 31^{st} Electrical Engineering Conference (EECON-31), Oct 29-30, 2008, Nakhon Nayok, Thailand.
3. Agus Bejo, Wanchalerm Pora and Hiroaki Kunieda, *Development of a 6-Axis Robotic Arm Controller Implemented on a Low-Cost Microcontroller*, Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology Conference 2009 (ECTI-Con 2009), May 6-9, 2009, Pattaya, Thailand.

CHAPTER II

BASIC THEORY AND PROPOSED METHODS

This chapter provides basic theory and some proposed methods which are used through this work.

2.1 Robot Kinematics

Robot arm kinematics deals with analytical study of the geometry of its motion with respect to a fixed reference coordinate system, particularly the relations between the joint-variable space and the position and orientation of the end-effector [1]. The robot arm used in this work has 6-DOF, which are all revolute-type. Hence, we use *robotic arm* term through this writing to refer a 6-axis robot arm.

2.1.1 Homogeneous Transformation Matrix

The homogeneous transformation matrix $\mathbb{T}$ is a 4×4 matrix which maps a position vector expressed in homogeneous coordinate system to another coordinate system. A large part of robot kinematics is concerned with the establishment of various coordinate systems to represent the positions and orientation of rigid object, and with transformations among these coordinate systems. Homogeneous transformations combine the operations of rotation and translation into a single matrix multiplication.

Suppose two coordinate systems are assigned to each link of a robot arm at link *i-1* and link *i* respectively. The link *i-1* coordinate system is the reference coordinate system and the link *i* coordinate system is the moving coordinate system when joint *i* is activated. Using the homogeneous transformation matrix $\mathbb{T}$, a point p_i can be expressed in the link *i* coordinate system in terms of the link *i-1* coordinate system as

$$p_{i-1} =^{i-1} \mathbb{T}_i p_i \tag{2.1}$$

where

p_i	=	position vector $(x_i, y_i, z_i, 1)^T$ representing a point in the link *i* coordinate system
p_{i-1}	=	position vector $(x_{i-1}, y_{i-1}, z_{i-1}, 1)^T$ representing the same point p_i in term of the link *i-1* coordinate system
${}^{i-1}\mathbb{T}_i$	=	homogeneous transformation matrix which specifies the location of the i^{th} coordinate frame with respect to the base coordinate system *i-1*th

A homogeneous transformation matrix can be considered to consist of four submatrices:

$$\mathbb{T} = \left[\begin{array}{c|c} \mathbb{R} & \vec{T} \\ \hline \vec{P} & s \end{array}\right] = \left[\begin{array}{c|c} Rotation & Translation \\ \hline Perspective & ScaleFactor \end{array}\right] \tag{2.2}$$

$\mathbb{R}$ is a 3×3 matrix that represents the relative rotation (orientation). Three basic rotation matrices $\mathbb{R}$ are given in (2.3) to (2.5). $\mathbb{R}_{x,\alpha}$, $\mathbb{R}_{y,\alpha}$ and $\mathbb{R}_{z,\alpha}$ respectively represent the rotation through an angel α about the x-axis, y-axis and z-axis.

$$\mathbb{R}_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{2.3}$$

$$\mathbb{R}_{y,\alpha} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \tag{2.4}$$

$$\mathbb{R}_{z,\alpha} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$\vec{T}$ is a 3×1 column vector that represents the relative translation (position). Three basic translation vectors $\vec{T}$ are given in (2.6). $\vec{T}_{x,a}$, $\vec{T}_{y,b}$ and $\vec{T}_{z,c}$ respectively represent the translation in a distance through the x-axis, b distance through the y-axis and c distance through the z-axis.

$$\vec{T}_{x,a} = \begin{bmatrix} a \\ 0 \\ 0 \end{bmatrix}, \vec{T}_{y,b} = \begin{bmatrix} 0 \\ b \\ 0 \end{bmatrix}, \vec{T}_{z,c} = \begin{bmatrix} 0 \\ 0 \\ c \end{bmatrix} \tag{2.6}$$

A 1×3 row vector $\vec{P}$ and a scalar s are the perspective and scale factor parameter that generally given by $\vec{P}$=[0 0 0] and s=1, when the joint is a rotary type.

2.1.2 The Denavit-Hartenberg Representation

In 1955, Denavit and Hartenberg [21] proposed a systematic and generalized approach of utilizing matrix algebra to describe and represent the spatial geometry of the links of a robot arm with respect to a fixed reference frame . This method uses a homogeneous transformation matrix $\mathbb{T}$ to describe the spatial relationship between two adjacent rigid mechanical links. The Denavit-Hartenberg (D-H) representation results a homogeneous transformation matrix representing each link's coordinate system at the joint with respect to the previous link's coordinate system.

Suppose A_i is the homogeneous matrix that transforms the coordinate of a point from frame *i* to frame *i-1* i.e $A_i =^{i-1} \mathbb{T}_i$. In the D-H convention, each homogeneous transformation A_i is represented as a product of four basic transformations

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \tag{2.7}$$

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & \cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

where the four quantities θ_i, d_i, a_i and α_i are parameters of link *i* and joint *i* as depicted in Figure 2.1. These parameters are obtained by step 7 in the D-H general procedure below. Since all joints of the robot used in this thesis are revolute joint so the three of above four quantities are constant while the fourth parameter θ_i is the joint variable.

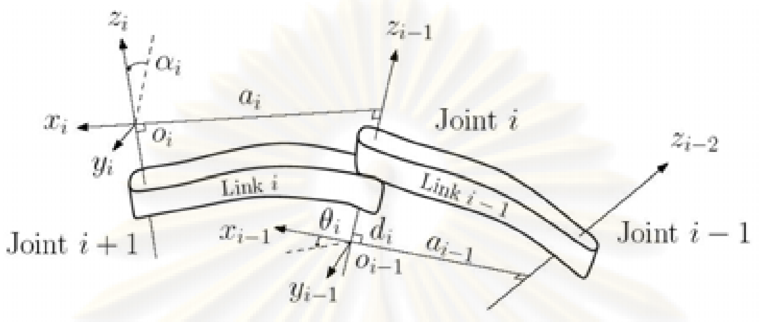


Figure 2.1: Denavit-Hartenberg frame assignment

Through sequential transformations, the end-effector can be transformed and expressed in the base coordinates system. The homogeneous matrix ${}^0\mathbb{T}_i$ which specifies the location of the i^{th} coordinate frame with respect to the base coordinate system is the chain product of successive coordinate transformation matrices of A_i and is expressed as

$$ {}^0\mathbb{T}_i = A_1 A_2 A_3 ... A_i \tag{2.9}$$

This is the general procedure to derive the link parameters based on the D-H convention:

step 1: Locate and label the joint axes $z_0, ..., z_{n-1}$.

step 2: Establish the base frame. Set the origin anywhere on the z_0-axis.
The x_0 and y_0 axes are chosen conveniently to form a right-hand frame.
For $i = 1, ..., n-1$ perform steps 3 to 5

step 3: Locate the origin o_i where the common normal to z_i and z_{i-1} intersects z_i.
If z_i intersects z_{i-1} locate o_i at this intersection.
If z_i and z_{i-1} are parallel, locate o_i at joint *i*.

step 4: Establish x_i along the common normal between z_{i-1} and z_i through 0_i or
in the direction normal to the z_{i-1} - z_i plane if z_{i-1} and z_i intersect.

step 5: Establish y_i to complete a right-hand frame.

step 6: Establish the end-effector $o_n x_n y_n z_n$

step 7: Create a table of link parameters θ_i, α_i, d_i and a_i
θ_i = the angle between x_{i-1} and x_i measured about z_{i-1}.
α_i = the angle between z_{i-1} and z_i measured about x_i.
d_i = the distance along z_{i-1} from o_{i-1} to the intersection of the x_i and z_{i-1} axes.
a_i = the distance along x_i from o_i to the intersection of the x_i and z_{i-1} axes.

step 8: Form the homogeneous transformation matrix A_i by substituting the above parameters into (2.8)

step 9: Form ${}^{0}\mathbb{T}_n = A_1A_2A_3...A_n$. This gives the position and orientation of the end-effector expressed in base coordinates.

2.1.3 Inverse Kinematics

There are two fundamental problems in robot arm kinematics namely forward kinematics and inverse kinematics. Fig 2.2 shows a simple block diagram of both forward and inverse kinematics problems. Forward kinematics problem is defined as how to compute the end-point position and orientation using known link parameters when joint variables are given . Conversely, inverse kinematics problem is defined as how to compute the joint variables using known link parameters when the end-point position and orientation is given.

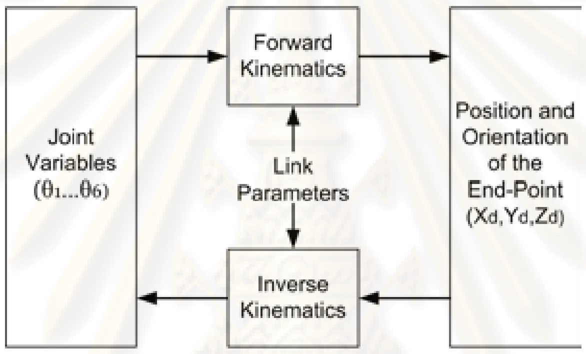

Forward Kinematics
Joint Variables ($\theta_1...\theta_6$)
Link Parameters
Position and Orientation of the End-Point (X_d,Y_d,Z_d)
Inverse Kinematics

Figure 2.2: Forward and Inverse Kinematics Problems

The general procedure described in section 2.1.2 yields the end-point position and orientation when the link parameters are known and a set of joint variables is given. This procedure is called forward kinematics solution. Generally, forward kinematics problem can be solved easier than inverse kinematics one. Nonetheless, inverse kinematics is more common faced in real application such as trajectory tracing.

S. Kucuk and Z. Bingul [3] mentioned that there are three types of method for solving the inverse kinematics problem as follows:

1. Complete analytical solution (closed-form method). The solutions of all joint variables solved analytically are derived from the basic equations. This method gives very accurate result but it requires complex calculation.

2. Numerical solution (nonclosed-form method). The solutions of joint variables solved numerically are obtained by iterative computational procedures. This method gives worse result than that of the analytical method but it can be applied generally. With bad initial values, correct solutions can not be guaranteed. It may yield only one solution or no answer at all.

3. Semi-analytical solutions. This solution combines analytical and numerical method. Some of the joint variables are determined analytically and some of them are computed numerically.

This thesis proposed a set of formulae to solve the inverse kinematics problem by using closed-form method as shown in section 3.1.

2.2 System Identification Method

System identification is the method that estimate the mathematical model of a system from experimental data [22]. Estimate model obtained by identification is then used for designing the controller mathematically. The following are general procedures of system identification steps:

1. Select and define a model structure (containing unknown parameters).
2. Exciting the system by known signals (step, sinusoidal, pseudo-random signals).
3. Collect and record both inputs and outputs data over a certain interval time.
4. Estimate the parameters using some statistically based methods.
5. Validate the obtained model, repeat 1-4 if required.

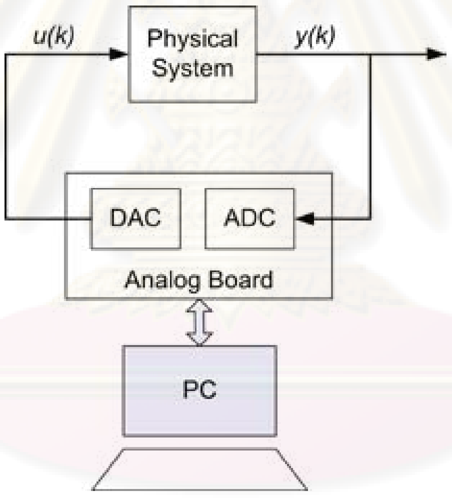


Figure 2.3: System identification setup

The specific methods used at each step depend on the type of model desired (parametric or non-parametric, continuous-time or discrete-time) and on the experimental conditions. The validation is the verification step to decide whether the identified model is acceptable or not.

The equipment setup of system identification for discrete time models is illustrated in Figure 2.3. A sampled input sequence $u(k)$ (where k is the discrete time) is applied to the physical system by means of a digital-to-analog converter (DAC). The measured sampled output sequence $y(k)$ is obtained by means of an analog-to-digital converter (ADC). An estimate model with adjustable parameters $\hat{\theta}$ is calculated on the computer. The error e, that is the difference between output $y(k)$ and the predicted output $\hat{y}(k)$, is used to update the estimate model parameters in order to minimize this error. The simple formula of adaptation algorithm is expressed:

$$\hat{\theta}^T_{k+1} = \hat{\theta}^T_k + \mu\phi^T e_k \tag{2.10}$$

where $\hat{\theta}^T, \mu, \phi^T$ and e are the estimated parameter, adaptation gain, measured data and error respectively. A non-recursive identification algorithms proceed a bulk of data input and output over a certain interval time to obtain the estimated parameter. The bigger data being processed the better estimated parameter is yielded. However, the increase of used data will be followed by the increase of computation cost. Recursive method is an alternative technique to counter that problem. Recursive method offers some advantages such as:

- Reducing the data processing since this algorithm only proceed a shape of data instead of the whole data.
- Estimating the real current model.
- Reducing the memory and computation requirement.
- Making possible to be applied as a real-time identification (on-line identification).

2.2.1 Least-Squares Method

The Least-Squares method was first introduced by Karl Gauss [23] for determining the orbits of planets. This method then becomes a major tool and the most popular estimation technique in parameter estimation because it is easy to comprehend and easy to be implemented. This algorithm is objected to minimize the sum of the squares of error between the actual data and that predicted by the model.

Let the process that will be estimated be described by an *n*th order difference equation as in (2.11) or be expressed as transfer function $G(q)$ as in (2.12). Assume that *N* samples of measurements of u_k, ..., u_{k-N} and y_k, ..., y_{k-N} are obtained from experiment. Where y_k and u_k are the output and input sample at time *k* respectively and $a_1, a_2, ..., a_n, b_1, b_2, ..., b_n$ is a set of constant parameters.

$$y_k + a_1 y_{k-1} + ... + a_n y_{k-n} = b_1 u_{k-1} + b_2 u_{k-2} + ... + b_n u_{k-n} \tag{2.11}$$

$$G(q) = \frac{B(q)}{A(q)} \tag{2.12}$$

where
$A(q) = 1 + a_1 q^{-1} + ... + a_n q^{-n}$
$B(q) = b_1 q^{-1} + ... + b_n q^{-n}$

To estimate the parameters a_i and b_i we introduce the *residual error* as

$$y_k + a_1 y_{k-1} + ... + a_n y_{k-n} = b_1 u_{k-1} + b_2 u_{k-2} + ... + b_n u_{k-n} + \epsilon_k \tag{2.13}$$

or

$$A(q)y_k = B(q)u_k + \epsilon_k \tag{2.14}$$

This equation is called *AutoRegresive with eXoganious* (ARX) and will be used through this writing. Filling the data samples into (2.13) results a set of linear equation that can be expressed as a simple matrix form in (2.15)

$$Y = \Phi\theta + \epsilon \tag{2.15}$$

where

$$Y = \begin{bmatrix} y_k \\ y_{k-1} \\ \cdots \\ y_{k-N} \end{bmatrix}$$

$$\Phi = \begin{bmatrix} -y_{k-1} & \cdots & -y_{k-n} & x_k & x_{k-1} & \cdots & x_{k-n} \\ -y_{k-2} & \cdots & -y_{k-n-1} & x_{k-1} & x_{k-2} & \cdots & x_{k-n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ -y_{k-N} & \cdots & -y_{k-n-N} & x_{k-N} & x_{k-1} & \cdots & x_{k-n-N} \end{bmatrix}$$

$$\theta = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ b_n \end{bmatrix}$$

$$\epsilon = \begin{bmatrix} \epsilon(k) \\ \epsilon(k-1) \\ \cdots \\ \epsilon(k-N) \end{bmatrix}$$

The estimate of θ, denoted by $\hat{\theta}$, is obtained by solving (2.15). A necessary condition to solve this set of equations is $N \geq n$. When $N = n$, it has a unique solution given in (2.16).

$$\hat{\theta} = \Phi^{-1}Y \tag{2.16}$$

where Φ^{-1} is the inverse of the square matrix Φ. However, when $N > n$, generally it is not possible to find a $\hat{\theta}$ vector which can fit the data samples exactly. It may be caused by too low order of the model or a wrong model structure. One way to determine the parameters is to estimate them based on least-square error.

Let the error $\epsilon(k)$ and the sum of the squares error V_{LS} are stated by (2.17) and (2.18) respectively.

$$\epsilon(k) = y(k) - \hat{y}(k) = y(t) - \Phi\hat{\theta} \tag{2.17}$$

$$V_{LS} = \frac{1}{N}\sum_{j=k-N}^{k} \epsilon(j)^2 = \frac{1}{N}\epsilon^T\epsilon \tag{2.18}$$

To carry out the minimization, we express

$$V_{LS}(\theta) = \frac{1}{N}(Y - \Phi\theta)^T(Y - \Phi\theta) = \frac{1}{N}\left[Y^TY - \theta^T\Phi^TY - Y^T\Phi\theta - \theta^T\Phi^T\Phi\theta\right] \qquad (2.19)$$

Taking the first derivative of $V_{LS}(\theta)$ with respect to θ and equating the result to zero, we have

$$\frac{\partial V_{LS}(\theta)}{\partial\theta}|_{\theta=\hat{\theta}} = \frac{1}{N}\left[-2\Phi^TY + 2\Phi^T\Phi\hat{\theta}\right] = 0 \qquad (2.20)$$

Hence the solution is given by the equation

$$\hat{\theta} = \left[\Phi^T\Phi\right]^{-1}\Phi^TY \qquad (2.21)$$

2.2.2 Recursive Least Square Method

In some cases, such as adaptive control, it may be necessary to estimate a model on-line while the process is in operation. The model will be updated when the new observations are available. Hence for computation efficiency, it is desirable to arrange the algorithms in such a way that the results obtained previously can be used for on-line updating. This way of algorithm is called recursive. The recursive version of Least-Squares method is called Recursive Least-Square methods. Because of some advantages mentioned above, this thesis will concern on this method only.

In this methods, a square matrix P_k is introduced as in (2.22). Suppose that we have found an estimate of parameters at time *k*th, that is $\hat{\theta_k}$. Then we could update those estimated parameters using a new data obtained at time *k+1th* (y_{k+1}) by formulae (2.23), (2.24) and (2.25).

$$P_k = \left[\Phi^T\Phi\right] \qquad (2.22)$$

$$\epsilon_{k+1} = y_{k+1} - \Phi^T\hat{\theta}_k \qquad (2.23)$$

$$P_{k+1} = P_k - \left[\frac{P_k\Phi\Phi^TP_k}{F + \Phi^TP_k\Phi}\right] \qquad (2.24)$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + P_{k+1}\Phi\epsilon_{k+1} \qquad (2.25)$$

F is forgetting factor which should be less than or equal to 1. When the process is slowly time varying, the measurements obtained a long time ago contain less information than the recent one. In order to let the estimator follow the change of the process, it is desirable to truncate the old measurements in the estimation algorithm.

2.2.3 Order Selection

The purpose of order selection is to find the model order that is most accurate for its use in control. There are some order selection methods that do not need complete parameter estimaton of model like checking the rank of covariance matrices and plotting the singular value of the Hankel matrix [22].

However, these methods are not accurate for noisy data. In this thesis, models will be estimated with increasing orders and then the best order will be chosen using an error criterion. This method is so-called cross-validation.

A number of criteria exist for order selection. In identification literature [23], the prediction error criterion is often used. Denote prediction error criterion using a validation data set as in (2.26) and using the estimate data set as in (2.27).

$$V_{PE}^{V} = \frac{1}{N}\sum_{k=1}^{N} \hat{H}^{-1}(q)\left[y_k^V - \hat{G}(q)u_k^V\right]^2 \tag{2.26}$$

$$V_{PE}^{E} = \frac{1}{N}\sum_{k=1}^{N} \hat{H}^{-1}(q)\left[y_k - \hat{G}(q)u_k\right]^2 \tag{2.27}$$

where u_k^V, y_k^V and u_k, y_k are the input-output from the validation data set and the input-output from the estimate data set respectively. Assume that the white noise e_k is Gaussian and the model order is correct, then Akaike (1974,1981) has derived an asymptotically unbiased estimate of V_{PE}^V using V_{PE}^E:

$$FPE = V_{PE}^{V} = \frac{N+d}{N-d}V_{PE}^{E} \tag{2.28}$$

where N is the number of estimation data samples and d is the number of model paramters. This is the famous *Akaike's Final Prediction Error* criterion (FPE). Here the factor $\frac{N+d}{N-d}$ is used to correct for the over-fit effect.

Remember that $H(q)$=1 for ARX model, therefore the term *lost function* V_{OE} can be used instead of FPE.

$$V_{OE} = \frac{1}{N}\sum_{k=1}^{N} \hat{\epsilon}_{OEk} \tag{2.29}$$

Note that *output error* $\hat{\epsilon}_{OE(k)}$ is not the same as *residual error* $\hat{\epsilon}_{(k)}$. The relation between them is

$$y_k = \frac{\hat{B}(q)}{\hat{A}(q)}u_k + \frac{\hat{\epsilon}_k}{\hat{A}(q)} \tag{2.30}$$

$$y_k = \frac{\hat{B}(q)}{\hat{A}(q)}u_k + \hat{\epsilon}_{OEk} \tag{2.31}$$

$$\hat{\epsilon}_k = \hat{A}(q)\hat{\epsilon}_{OEk} \tag{2.32}$$

In general, the lost function V_{OE} decreases as the order n increases. The reduction of V_{OE} will be insignificant when the order of the model is high enough for simulation purposes. Based on this principle, a procedure for order selection is so simple. The appropriate model order can be chosen as the one for which V_{OE} stops decreasing significantly.

2.2.4 Validation

The goal of model validation is to check if the identifed model is good enough for use in control. This is done by carrying out a *whiteness test* on the sequence of *residual* errors. An acceptable identified model should verify the condition

$$|RN(i)| \leq \frac{2.17}{\sqrt{N}}; \quad i = 1, 2, ..., max(n_A, n_B + d) \tag{2.33}$$

$$RN(i) = \frac{\frac{1}{N}\sum_{k=1}^{N} \epsilon_k \epsilon_{k-i}}{\frac{1}{N}\sum_{k=1}^{N} \epsilon_k^2} \tag{2.34}$$

where N is the number of samples.

2.3 Control Design Method

Controllers can be classified into feedback and feed forward controllers. The feedback controllers itself are divided into one degree of freedom and two degree of fredom controllers (abbreviated as 1-DOF and 2-DOF respectively) [24]. The structure of both controllers (1-DOF and 2-DOF) in discrete time are shown in Figure 2.4.

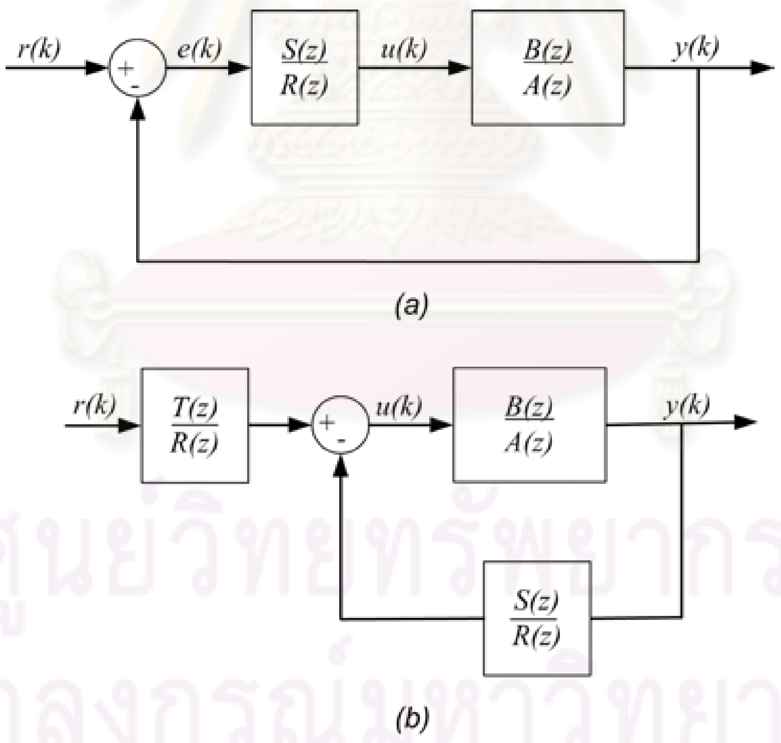


Figure 2.4: Controller Structure (a) 1-DOF (b) 2-DOF

1-DOF controller is simpler to design than 2-DOF controller because it has fewer parameters. However, 2-DOF controller has more capabilities to shape the responses both reference and disturbance signals simultaneously and independently.

Figure 2.5 shows the general principle of controller design. In order to design and tune a controller correctly, one needs:

1. Determination of the plant model.
2. Specification of the performance.
3. Computation of the controller parameters (the coefficients of the polynomials $R(z)$, $S(z)$ and $T(z)$.
4. Verification of the achieved robustness margins and sensitivity functions.

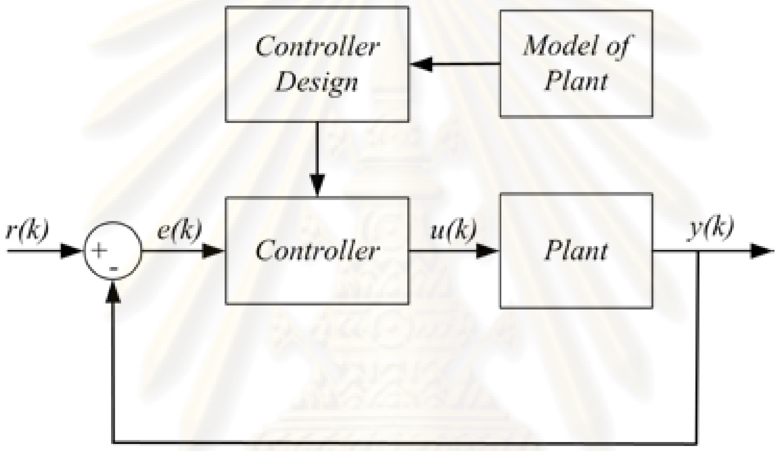


Figure 2.5: Controller design principle

Step no.1, that is system identification, has been detailed previously in chapter 2.2 . This section will detail the other points.

2.3.1 2-DOF PID Controller

Proportional, integral and derivative (PID) controllers are widely used in industrial practice. Because of its merit and its simplicity to design, this thesis will be focused on 2-DOF Discrete PID Controller.

Consider the 2-DOF discrete control structure presented in Figure 2.4 (b). The control law is implemented by:

$$u(k) = \frac{T(z)}{R(z)}r(k) - \frac{S(z)}{R(z)}y(k) \tag{2.35}$$

It is easy to arrive at the following relation between $r(k)$ and $y(k)$:

$$y(k) = \frac{T(z)}{R(z)} \frac{B(z)/A(z)}{(1 + B(z)S(z)/A(z)R(z))} r(k) = \frac{B(z)T(z)}{A(z)R(z) + B(z)S(z)} r(k) \tag{2.36}$$

The error, that is the difference between the reference signal $r(k)$ and the actual value $y(k)$, is given by:

$$e(k) = r(k) - y(k) = \left(1 - \frac{B(z)T(z)}{A(z)R(z) + B(z)S(z)}\right) r(k) \tag{2.37}$$

Simplifying, we arrive at

$$e(k) = \frac{A(z)R(z) + B(z)S(z) - B(z)T(z)}{A(z)R(z) + B(z)S(z)} r(k) \tag{2.38}$$

The discrete time form of the error expression in (2.38) is given by:

$$E(z) = \frac{A(z)R(z) + B(z)S(z) - B(z)T(z)}{A(z)R(z) + B(z)S(z)} r(z) \tag{2.39}$$

where $E(z)$ and $r(z)$ are, respectively, the Z-transforms of $e(k)$ and $r(k)$. Using the final value theorem from discrete time systems, we obtain:

$$\lim_{k\to\infty} e_k = \lim_{z\to 1} \frac{z-1}{z} E(z) \tag{2.40}$$

Substituting the expression for $E(z)$ in (2.39) with step function input, the above equation becomes

$$\lim_{k\to\infty} e_k = \lim_{z\to 1} \frac{z-1}{z} \frac{A(z)R(z) + B(z)S(z) - B(z)T(z)}{A(z)R(z) + B(z)S(z)} \frac{z}{z-1} \tag{2.41}$$

where we have made use of the fact that $R(z)$ is the transfer function of unit step. When the controller has an integral action, $R(1) = 0$. Using this, the above equation reduces to:

$$e(\infty) = \frac{S(z) - T(z)}{S(z)}|_{z=1} = \frac{S(1) - T(1)}{S(1)} \tag{2.42}$$

The above condition can be satisfied if one of the following condition is met:

$$S = T \tag{2.43}$$

$$S(1) = T \quad or \quad S = T(1) \tag{2.44}$$

$$S(1) = T(1) \tag{2.45}$$

If we use $S = T$ to solve (2.42) it mean that we use 1-DOF controller. We also need to assume that $S(1)$ is nonzero otherwise it is a 1-DOF controller too.

2.3.2 Pole Placement

The *pole placement* method allows the design of a 2-DOF discrete controller both for stable and unstable systems without restriction on the degrees of the polynomials $A(z)$ and $B(z)$, without restriction on the time delay and without restriction on the plant zeros (stable or unstable) because this method does not simplify the system zeros. The only restriction concerns the possible common factors of $A(z)$ and $B(z)$, which must be simplified before the computations are carried out.

Let the controlled plant is characterized by the transfer function:

$$H(z) = \frac{z^{-d}B(z)}{A(z)} \tag{2.46}$$

where
d : the integer number of sampling periods contained in the time delay
$A(z) : 1 + a_1 z^{-1} + ... + a_{nA} z^{-nA}$
$B(z) : b_1 z^{-1} + ... + b_{nB} z^{-nB}$

The closed loop transfer function is given by:

$$H_{CL}(z) = \frac{z^{-d}T(z)B(z)}{A(z)R(z) + z^{-d}B(z)S(z)} = \frac{z^{-d}T(z)B(z)}{P(z)} \tag{2.47}$$

where
$P(z) = 1 + p_1 z^{-1} + ... + p_{n_P} z^{-n_P}$
$R(z) = 1 + r_1 z^{-1} + ... + r_{n_P} z^{-n_R}$
$S(z) = s_0 + s_1 z^{-1} + ... + s_{n_P} z^{-n_S}$

Polynomial $P(z)$ can be specified directly by defining the desired closed loop poles such as $P(z) = (1+p'_1 z^{-1})(1+p'_2 z^{-1})...(1+p'_{nP} z^{-1})$. Generally, $P(z)$ is chosen in the form of a second-order polynomial by discretization of a second-order continuous time system, specifying ω_0, ζ and assuring that the condition in (2.48) is satisfied.

$$0.25 \leq \omega_0 T_s \leq 1.5; \quad 0.7 \leq \zeta \leq 1 \tag{2.48}$$

Once $P(z)$ is specified, the following equation (known as *Bezout Identity equation*) must be solved to compute $R(z)$ and $S(z)$ in (2.47):

$$A(z)R(z) + z^{-d}B(z)S(z) = P(z) \tag{2.49}$$

Defining

$$n_A = degA(z); \quad n_B = degB(z) \tag{2.50}$$

This polynomial equation has a unique solution with minimal degree (when $A(z)$ and $B(z)$) do not have common factors) for

$$n_P = degP(z) \leq n_A + n_B + d - 1 \tag{2.51}$$

$$n_R = degR(z) = n_B + d - 1 \tag{2.52}$$

$$n_S = degS(z) = n_A - 1 \tag{2.53}$$

In order to solve equation (2.49) effectively, this is often represented in the matrix form

$$Mx = p \tag{2.54}$$

where

$$x = [1, r_1, ..., r_{n_R}, s_0, ..., s_{n_S}]^T \tag{2.55}$$

$$p = [1, p_1, ..., p_i, p_{n_P}, 0, ..., 0]^T \tag{2.56}$$

and the matrix M has the following form

$$M = \underbrace{\begin{bmatrix} 1 & 0 & ... & 0 & 0 & ... & 0 \\ a_1 & 1 & ... & ... & b'_1 & ... & ... \\ a_2 & a_1 & ... & 0 & b'_2 & ... & b'_1 \\ ... & ... & ... & 1 & ... & ... & b'_2 \\ ... & ... & ... & a_1 & ... & ... & ... \\ a_{n_A} & ... & ... & a_2 & b'_{n_B} & ... & ... \\ 0 & ... & ... & ... & 0 & ... & ... \\ 0 & ... & ... & a_{n_A} & 0 & ... & b'_{n_B} \end{bmatrix}}_{n_A+n_B+d} \tag{2.57}$$

where:
$b'_i = 0$ for $i = 0, 1, ..., d$; $\quad b'_i = b_{i-d}$ for $i \geq d + 1$

The vector x, contains the coefficients of the polynomial $S(z)$ and $R(z)$, is obtained using matrix inversion M by the formula

$$x = M^{-1}p \tag{2.58}$$

where M^{-1} is the matrix inverse of M. This inverse exists if the determinant of the matrix M is not zero. One can prove that this is verified if and only if $A(z)$ and $B(z)$ are coprime polynomials (no simplifications between zeros and poles). Finally, parameters of polynomial $R(z)$ are obtained by equation (2.43), (2.44) or (2.45).

CHAPTER III

IMPLEMENTATION, HARDWARE AND SOFTWARE DESIGN

This chapter explains the implementation and the hardware design of controller system. At first, a set of formulae for solving the inverse kinematics problem and how to generate the trajectory files are described. The 2-DOF PID controllers then are conveyed based on servo valves models obtained from system identification. Finally, the required hardwares are constructed to realize the controller system.

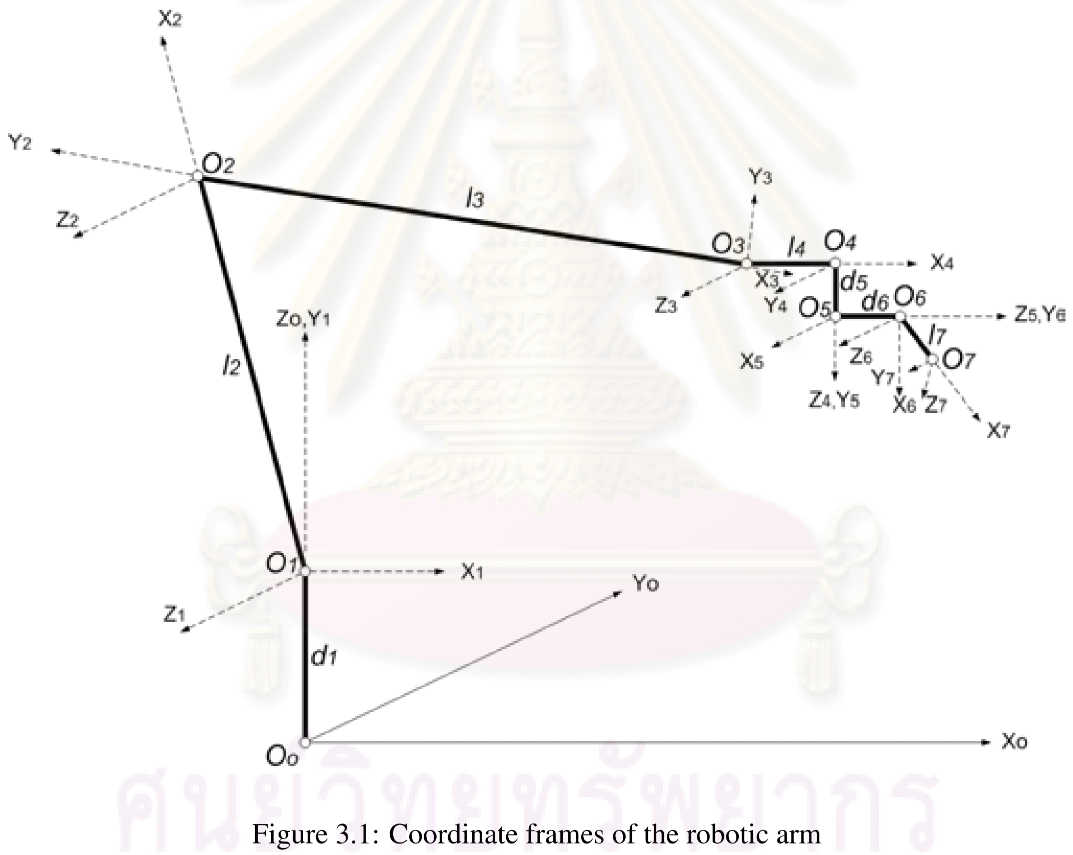


Figure 3.1: Coordinate frames of the robotic arm

3.1 Closed-form Solution of Inverse Kinematics

In order to develop formulae for solving the inverse kinematics problem described in section 2.1.3, a set of joint parameter represents the mechanical structure of the robot must be known. The mechanical structure of the robot used in this case study is shown in Figure 3.1. The coordinate Oi, $i = 0, .., 7$ was specified. Then the parameters l_i,α_i, d_i, and θ_i can be extracted following the steps in section 2.1.2 as shown in Table 3.1.

Table 3.1: Denavit-Hartenberg Parameters

Axis	l_i	α_i	d_i	θ_i
1	0	$\frac{\pi}{2}$	63.2	θ_1
2	100	0	0	θ_2
3	164	0	0	θ_3
4	10.16	$\frac{\pi}{2}$	0	θ_4
5	0	$\frac{\pi}{2}$	7.6	θ_5
6	0	$\frac{\pi}{2}$	8.5	θ_6
7	11	$\frac{\pi}{2}$	0	0.7π

3.1.1 General inverse kinematics formulae when O_7 and its orientation are given

In this section, we develop a general inverse kinematics formulae when a desired end-point position (O_7) and its orientation are given. Figure 3.2 and Figure 3.3 show the coordinate frame example and the zoomed area of end-point respectively.

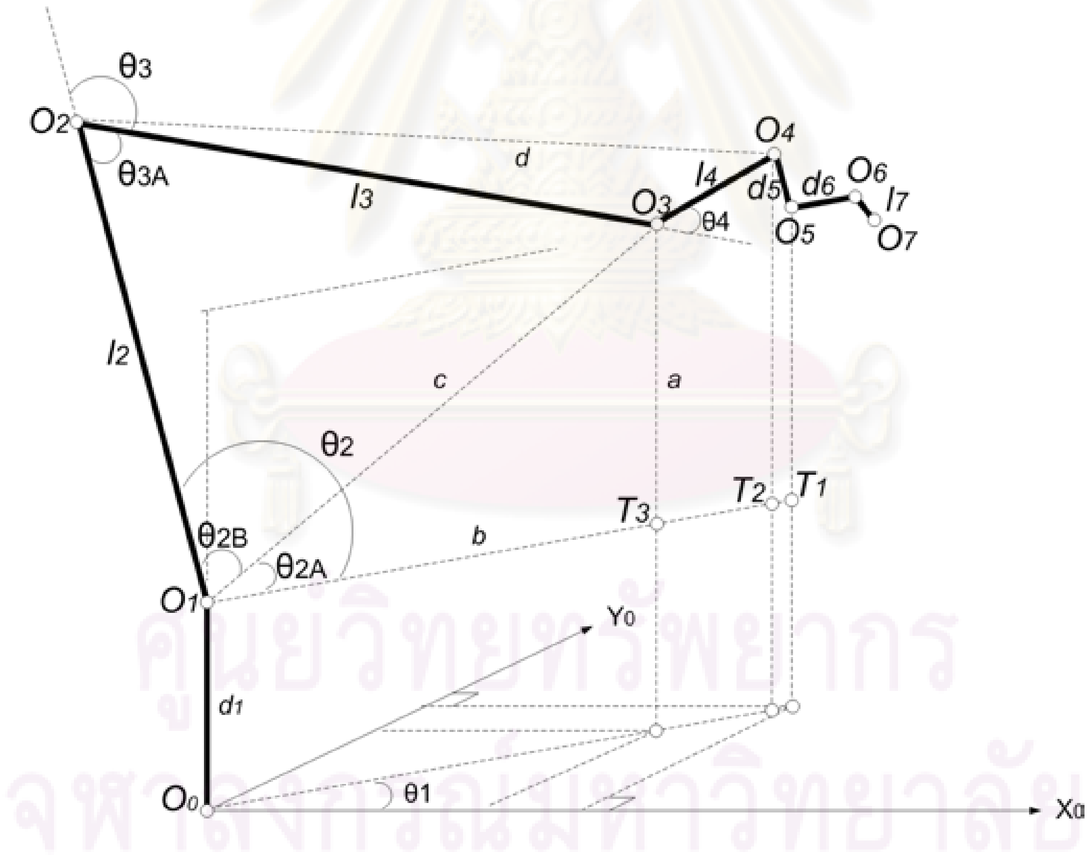


Figure 3.2: Coordinate frame example for section 3.1.1

The end-point position is specified in original coordinate direction O_0 ($O_7 = [x_7\ y_7\ z_7]^T$) whereas the orientation is specified by α and β as shown in Figure 3.3.

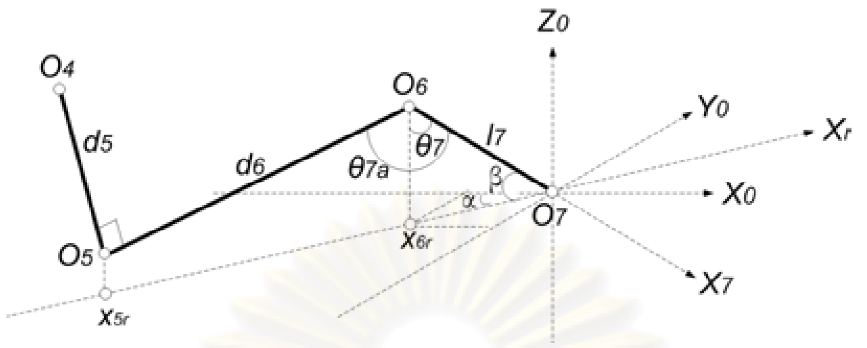


Figure 3.3: End-point position and orientation

Assume that O_7, α and β are given, where O_i=$[x_i\ y_i\ z_i]^T$ is the position coordinate of joint-i in X_0-axis, Y_0-axis and Z_0-axis respectively. The corresponding homogeneous matrix transformation of desired position and orientation ($^0\mathbb{T}_7$) is stated by a rotation of α degrees about Z_0 axis followed by a rotation α degrees about Y_7 axis, a rotation 180 degrees about X_7 axis and translated by translation O_7=$[x_7\ y_7\ z_7]^T$.

$$^0\mathbb{T}_7 = Rot_{z,\alpha}\ Rot_{y,\beta}\ Rot_{x,180}\ Trans_{x,x_7}\ Trans_{y,y_7}\ Trans_{z,z_7} \tag{3.1}$$

$$^0\mathbb{T}_7 = \begin{bmatrix} \cos\alpha\cos\beta & \sin\alpha & -\cos\alpha\sin\beta & x_7 \\ \sin\alpha\cos\beta & -\cos\alpha & -\sin\alpha\sin\beta & y_7 \\ -\sin\beta & 0 & -\cos\beta & z_7 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

From Figure 3.3 we can easily obtain O_6:

$$O_6 = O_7 + \begin{bmatrix} -l_7\cos\beta\cos\alpha \\ -l_7\cos\beta\sin\alpha \\ l_7\sin\beta \end{bmatrix} \tag{3.3}$$

θ_{7a} is a constant which is determined by the mechanical structure of the robotic arm. The relation between θ_{7a} and θ_7 is given in (3.4).

$$\theta_{7a} = \pi/2 + \theta_7 \tag{3.4}$$

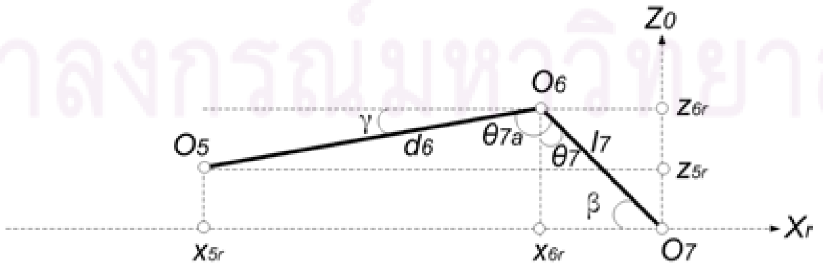


Figure 3.4: X_r-Z_0 plane view

There are many possible points of O_5 so that the angle between link-6 and link-7 satisfies θ_{7a}. In order to get a unique position of O_5, we constrains this so that it is in the X_r-Z_0 plane and O_5 always in the left side of O_6. Figure 3.4 shows the X_r-Z_0 plane view. An auxiliary variable γ is denoted by:

$$\gamma = \pi - \beta - \theta_{7a} \tag{3.5}$$

Then the position of O_5 in the original base coordinate is:

$$O_5 = O_6 + \begin{bmatrix} -d_6 \cos\gamma \cos\alpha \\ -d_6 \cos\gamma \sin\alpha \\ -d_6 \sin\gamma \end{bmatrix} \tag{3.6}$$

Because link-5 and link-6 always be perpendicular, it satisfies the following rule:

$$(x_4 - x_5)(x_6 - x_5) + (y_4 - y_5)(y_6 - y_5) + (z_4 - z_5)(z_6 - z_5) = 0 \tag{3.7}$$

Using known position of O_6 and O_5 in (3.3) and (3.6), we introduce constants $c_1 = x_6 - x_5$, $c_2 = y_6 - y_5$ and $c_3 = z_6 - z_5$. Then equation (3.7) becomes:

$$(x_4 - x_5)c_1 + (y_4 - y_5)c_2 + (z_4 - z_5)c_3 = 0 \tag{3.8}$$

$$c_1 x_4 + c_2 y_4 + c_3 z_4 = c_4 \tag{3.9}$$

where $c_4 = c_1 x_5 + c_2 y_5 + c_3 z_5$.

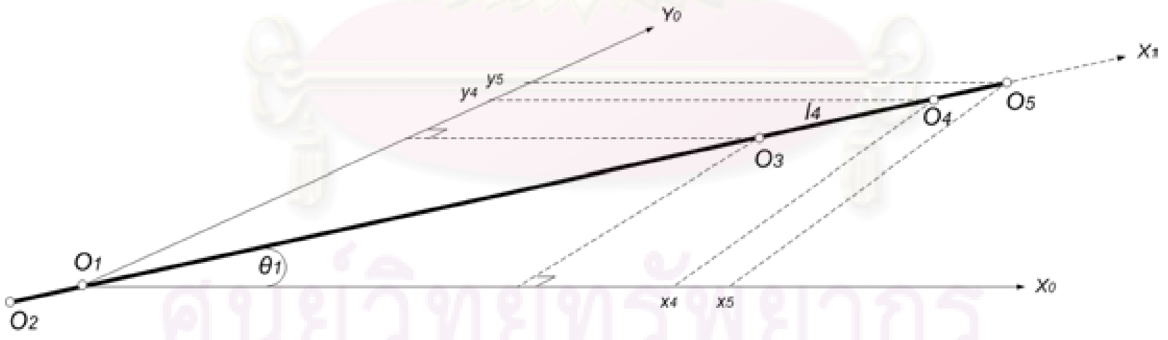


Figure 3.5: Top view of X_0-Y_0 plane

Figure 3.5 shows the top view of X_0-Y_0 plane. From this figure we can compute θ_1 using formula in (3.10). We also have correlation between x_4, y_4 and x_{41} as shown in (3.11) and (3.12), where x_{i1} $(i = 2, 3, 4, 5)$ is the projection of corresponding position O_i in x_1 direction.

$$\theta_1 = \tan^{-1}\left(\frac{y_5}{x_5}\right) \tag{3.10}$$

$$x_4 = x_{41} cos\theta_1 \tag{3.11}$$

$$y_4 = x_{41} sin\theta_1 \tag{3.12}$$

Substituting (3.11) and (3.12) into (3.9) yields

$$c_1 x_{41} cos\theta_1 + c_2 x_{41} sin\theta_1 + c_3 z_4 = c_4 \tag{3.13}$$

$$(c_1 cos\theta_1 + c_2 sin\theta_1) x_{41} = c_4 - c_3 z_4 \tag{3.14}$$

$$x_{41} = \frac{c_4 - c_3 z_4}{c_1 cos\theta_1 + c_2 sin\theta_1} \tag{3.15}$$

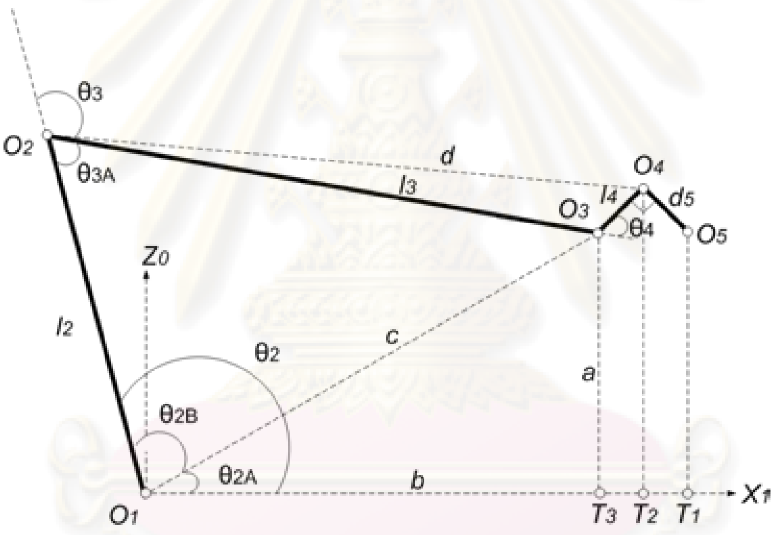


Figure 3.6: Side view of X_1-Z_0 plane

Figure 3.6 shows the side view of X_1-Z_0 plane. Because link-4 and link-5 is perpendicular in this plane, it satisfies:

$$(x_{31} - x_{41})(x_{51} - x_{41}) + (z_{31} - z_{41})(z_{51} - z_{41}) = 0 \tag{3.16}$$

$$(x_{41} - x_{51})^2 + (z_{41} - z_{51})^2 = d_5^2 \tag{3.17}$$

$$(x_{41} - x_{31})^2 + (z_{41} - z_{31})^2 = l_4^2 \tag{3.18}$$

Substituting (3.15) into (3.17) yields

$$(\frac{c_4 - c_3 z_4}{c_1 cos\theta_1 + c_2 sin\theta_1} - x_{51})^2 + (z_{41} - z_{51})^2 = d_5^2 \tag{3.19}$$

Because z_4=z_{41}+d_1 then

$$(\frac{c_4 - c_3(z_{41}+d_1)}{c_1 cos\theta_1 + c_2 sin\theta_1} - x_{51})^2 + (z_{41}-z_{51})^2 = d_5^2 \tag{3.20}$$

Simplifying this equation yields

$$pz_{41}^2 + qz_{41} + r = 0 \tag{3.21}$$

where
$p = \left[\frac{c_3^2}{(c_1 cos\theta_1 + c_2 sin\theta_1)^2}\right] + 1$
$q = -2\left[\frac{c_3}{c_1 cos\theta_1 + c_2 sin\theta_1}(\frac{c_4 - c_3 d_1}{c_1 cos\theta_1 + c_2 sin\theta_1} - x_{51}) + z_{51}\right]$
$r = (\frac{c_4 - c_3 d_1}{c_1 cos\theta_1 + c_2 sin\theta_1} - x_{51})^2 + z_{51}^2 - d_5^2$

Using quadratic formula, we have 2 solutions for the problem in (3.21). However, we only choose the bigger one because we constrain that the position of O_4 always higher than the position of O_5. Therefore, the solution of z_{41} is given by:

$$z_{41} = \frac{-q + \sqrt{q^2 - 4pr}}{2p} \tag{3.22}$$

After z_{41} is known then we can obtain x_{41} using formula in (3.15). Now we can simplify formula in (3.16)

$$(x_{31} - x_{41})c_6 + (z_{31} - z_{41})c_7 = 0 \tag{3.23}$$

$$c_6 x_{31} + c_7 z_{31} = c_6 x_{41} + c_7 z_{41} \tag{3.24}$$

$$z_{31} = \frac{(c_6 x_{41} + c_7 z_{41}) - c_6 x_{31}}{c_7} \tag{3.25}$$

where c_6=$x_{51} - x_{41}$ and c_7=$z_{51} - z_{41}$. Substituting (3.25) into (3.18) yields

$$sx_{31}^2 + tx_{31} + u = 0 \tag{3.26}$$

where
$s = (c_6^2/c_7^2) + 1$
$t = 2\left[\frac{c_6}{7}(z_{41} - \frac{c_6 x_{41} + c_7 z_{41}}{c_7}) - x_{41}\right]$
$u = \left[(z_{41} - \frac{c_6 x_{41} + c_7 z_{41}}{c_7})^2 + x_{41}^2\right] - l_5^2$

Problems in (3.26) also has 2 solutions. Because the position of O_3 always in the left of O_4 then we chose the smaller one. The solution of x_{31} is given by:

$$x_{31} = \frac{-t - \sqrt{t^2 - 4su}}{2s} \tag{3.27}$$

Then z_{31} can be solved using formula in (3.25). From Figure 3.6 we can derive a formula to obtain θ_2, θ_3 and θ_4 as in (3.29), (3.30) and (3.31) respectively, where a=z_{31}, b=x_{31} and d=$\|O_2 - O_4\|_2$:

$$c = \sqrt{a^2 + b^2} \tag{3.28}$$

$$\theta_2 = \tan^{-1}\left(\frac{a}{b}\right) + \cos^{-1}\left(-\frac{l_3^2 - l_2^2 - c^2}{2l_2 c}\right) \tag{3.29}$$

$$\theta_3 = -\pi + \cos^{-1}\left(-\frac{c^2 - l_2^2 - l_3^2}{2l_2 l_3}\right) \tag{3.30}$$

$$\theta_4 = \pi - \cos^{-1}\left(-\frac{d^2 - l_3^2 - l_4^2}{2l_3 l_4}\right) \tag{3.31}$$

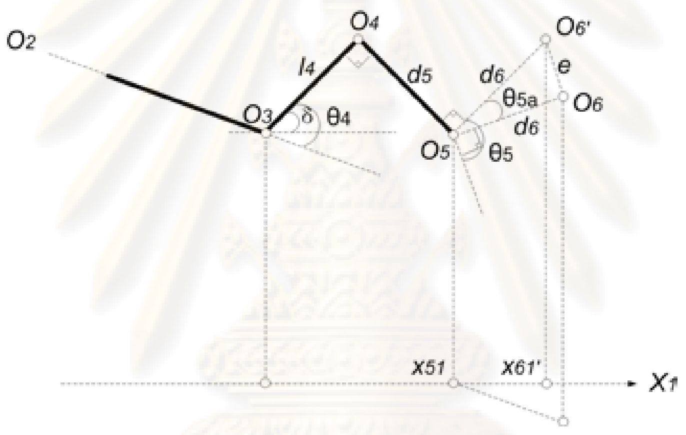


Figure 3.7: Zoomed area of O_3, O_4, O_5 and O_6' in $X_1 - Z_0$ plane

Figure 3.7 shows the zoomed area of O_3, O_4, O_5 and O_6' in $X_1 - Z_0$ plane. O_6' is the projection of O_6 in X_1 coordinate direction. From this figure, we derive the following formulae, where δ=$\theta_4 + \theta_3 + \theta_2$:

$$x_{61'} = x_{51} + d_6 \cos\delta \tag{3.32}$$

$$z_{61'} = z_{51} + d_6 \sin\delta \tag{3.33}$$

$$x_{6'} = x_{61'} \cos\theta_1 \tag{3.34}$$

$$y_{6'} = x_{61'} \sin\theta_1 \tag{3.35}$$

$$z_{6'} = z_{61'} + d_1 \tag{3.36}$$

Denotes vectors $\vec{v_6}$ and $\vec{v_{6'}}$ are the relative position of O_6 and O_6' toward O_5 respectively.

$$\vec{v_6} = [(x_6 - x_5) \quad (y_6 - y5) \quad (z_6 - z_5)] \tag{3.37}$$

$$\vec{v_{6'}} = [(x_{6'} - x_5) \quad (y_{6'} - y_5) \quad (z_{6'} - z_5)] \tag{3.38}$$

By using dot product of two vector formula we obtain:

$$\theta_{5a} = cos^{-1}\left(\frac{\vec{v_6}\vec{v_{6'}}^T}{\|\vec{v_6}\| \, \|\vec{v_{6'}}\|}\right) = cos^{-1}\left(\frac{\vec{v_6}\vec{v_{6'}}^T}{d_6^2}\right) \tag{3.39}$$

Then we get θ_5:

$$\theta_5 = \frac{\pi}{2} \pm \theta_{5a} \tag{3.40}$$

We introduce $\Delta\vec{v}$=$\vec{v_6} - \vec{v_{6'}}$. The sign in (3.40) is plus (+) when $\Delta\vec{v}_{(2)}$ is negative, otherwise is minus (−). Now, θ_1...θ_5 have been known and θ_7 is a constant. Therefore we can obtain θ_6 using the following formula:

$$^0\mathbb{T}_7 =^0 \mathbb{T}_5 \, ^5\mathbb{T}_6 \, ^6\mathbb{T}_7 \tag{3.41}$$

where $^5\mathbb{T}_6$=A_6 and $^6\mathbb{T}_7$=A_7 as defined in (2.8). Because the unknown parameter θ_6 is stated in A_6, so the solution of problem (3.41) is:

$$A_6 =^0 \mathbb{T}_5^{-1} \, ^0\mathbb{T}_7 \, A_7^{-1} \tag{3.42}$$

Finally, we get θ_6 from known matrix A_6

$$\theta_6 = \cos^{-1} A_{6(1,1)} \tag{3.43}$$

3.1.2 Specific inverse kinematic formulae when O_7 is given and link-6 is perpendicular to the object plane (Y_0-Z_0 plane)

This is a specific case for section 3.1.1. We develop another formulae of inverse kinematics when the end-point position (O_7) is given and the orientation is fixed determined so that the link-6 always be perpendicular to the object plane (Y_0-Z_0 plane).

In fact, this case can be solved using general inverse kinematics formulae as in previous section. However, the general inverse kinematics formulae requires much complex computation. Therefore, we develop another simplified formulae to satisfy this case. Figure 3.8 shows the coordinate frame example of section 3.1.2.

In this case, to make the end-point orientation constant, θ_6 should be determined previously. If the end-point position is given then the appropriate angle θ_1 to θ_6 can be obtained by the formulae in (3.44) to (3.57). Assume that an end-point $O_7 = [x_7 \; y_7 \; z_7]^T$ is given and θ_6 is defined previously, where x_i, y_i and z_i are the position coordinate of joint-*i* in X_0-axis, Y_0-axis and Z_0-axis respectively.

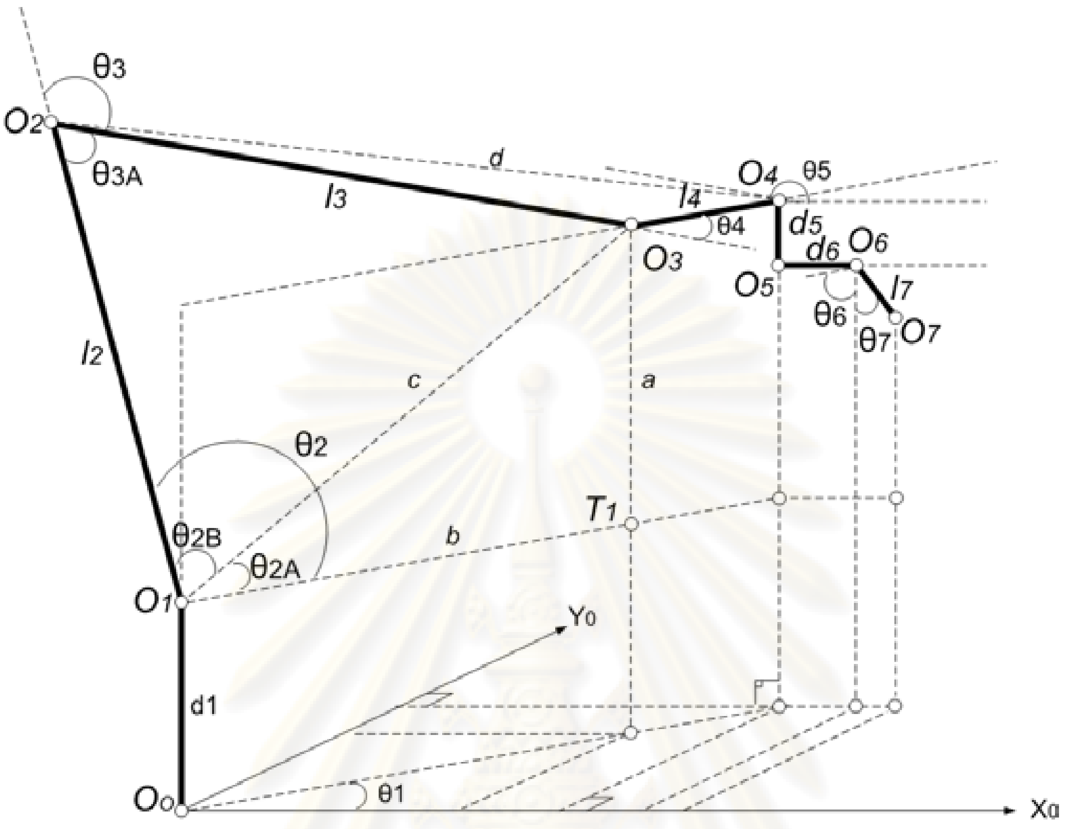


Figure 3.8: Coordinate frame example for section 3.1.2

Note that θ_7 is also a constant that defined by its mechanical structure. From Figure 3.8, O_6, O_5 and O_4 can be computed by equation in (3.44) to (3.46)

$$O_6 = O_7 + \begin{bmatrix} -l_7 \sin\theta_7 \\ \cos\theta_6 \cos\theta_7 \\ -l_7 \sin\theta_6 \cos\theta_7 \end{bmatrix} \tag{3.44}$$

$$O_5 = O_6 + \begin{bmatrix} -d_6 \\ 0 \\ 0 \end{bmatrix} \tag{3.45}$$

$$O_4 = O_5 + \begin{bmatrix} 0 \\ 0 \\ -d_5 \end{bmatrix} \tag{3.46}$$

Using known position of O_4, then θ_1 can be computed:

$$\theta_1 = \tan^{-1}\left(\frac{y_4}{x_4}\right) \tag{3.47}$$

If the area around O_1, O_3 and O_4 is zoomed in with top view as shown in Figure 3.9, the next formula can be derived to compute θ_5 and position O_3.

$$\theta_5 = \frac{\pi}{2} + \theta_1 \tag{3.48}$$

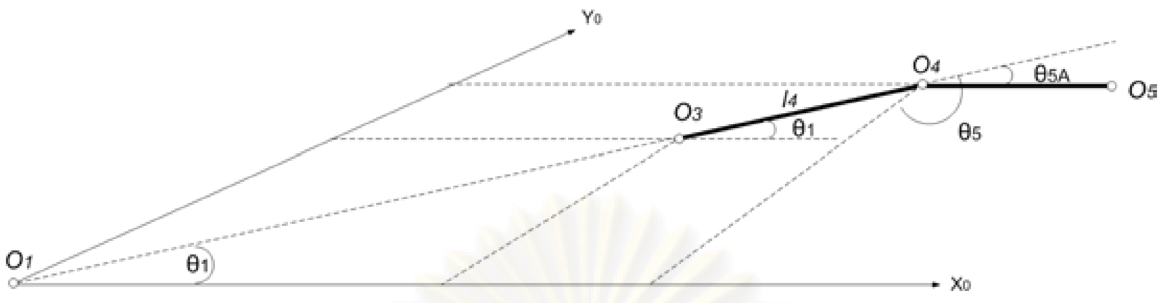


Figure 3.9: Zooming in the area O_1, O_3 and O_4 with top view

$$O_3 = O_4 + \begin{bmatrix} -l_4 \cos\theta_1 \\ -l_4 \sin\theta_1 \\ 0 \end{bmatrix} \tag{3.49}$$

An auxiliary position, called T_1 and variables a, b, c and d are introduced. Figure 3.10 is the zoomed area of O_1, O_2 and O_3 in X_1-Z_0 plane. From this figure we can derive formulae to compute θ_2 and θ_3 as in (3.54) and (3.55), where $\|\cdot\|$ denotes norm order 2.

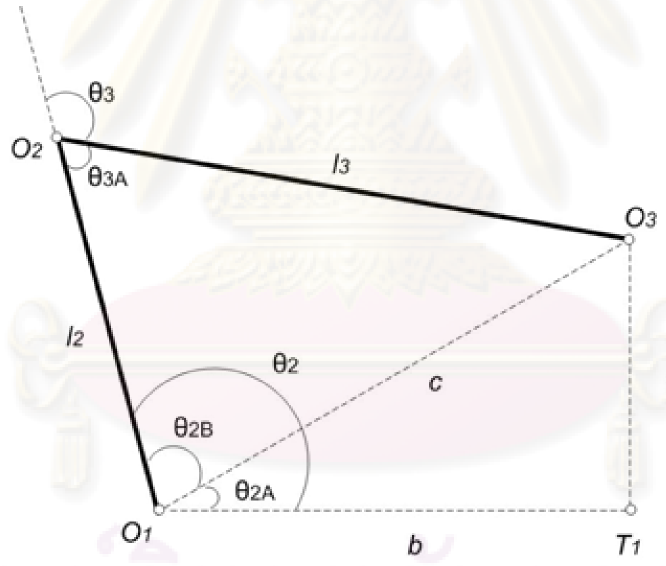


Figure 3.10: Zooming in the area O_1, O_2 and O_3 with side view

$$T_1 = O_3 + \begin{bmatrix} 0 \\ 0 \\ -d_1 \end{bmatrix} \tag{3.50}$$

$$a = \|O_3 - T_1\| = z_3 - d_1 \tag{3.51}$$

$$b = \|T_1 - O_1\| = \sqrt{x_3^2 + y_3^2} \tag{3.52}$$

$$c = \|O_3 - O_1\| = \sqrt{a^2 + b^2} \tag{3.53}$$

$$\theta_2 = \tan^{-1}\left(\frac{a}{b}\right) + \cos^{-1}\left(-\frac{l_3^2 - l_2^2 - c^2}{2l_2c}\right) \tag{3.54}$$

$$\theta_3 = -\pi + \cos^{-1}\left(-\frac{c^2 - l_2^2 - l_3^2}{2l_2l_3}\right) \tag{3.55}$$

Figure 3.11 shows the zoomed area of O_2, O_3 and O_4. θ_4 can be obtained by the following formula:

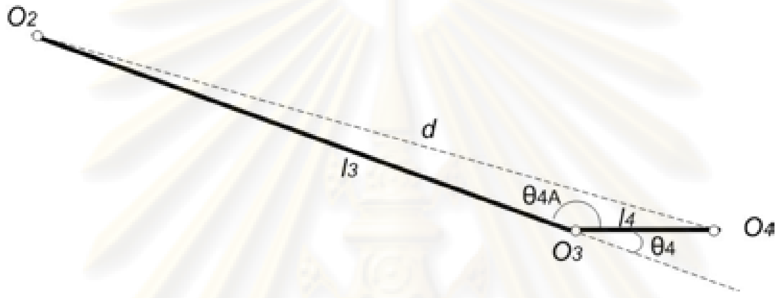


Figure 3.11: Zooming in the area O_2, O_3 and O_4 with side view

$$d = \|O_2 - O_4\|_2 = \sqrt{(x_2 - x_4)^2 + (y_2 - y_4)^2 + (z_2 - z_4)^2} \tag{3.56}$$

$$\theta_4 = \pi - \cos^{-1}\left(-\frac{d^2 - l_3^2 - l_4^2}{2l_3l_4}\right) \tag{3.57}$$

3.1.3 Moving end-point in O_0 coordinate direction with keeping its orientation

In this section, we design a strategy for moving the end-point of the robotic arm. The end-point will be moved left-right, up-down and in-out in Y_0, Z_0 and X_0 direction respectively while the orientation is kept constant. This strategy requires formulae which is almost the same with section 3.1.1. The only difference is that to keep its orientation, we move O_5, O_6 and O_7 parallelly with the same relative movement from the previous position. Figure 3.12 shows the coordinate frame example for this strategy.

Assume that present positions $O_{iP} = [x_{iP}\ y_{iP}\ z_{iP}]^T$ are known by using forward kinematics formula $O_{iP} = ForwardKinematics(\vec{\theta}_P)$, where $i = 5, 6, 7$ and $\vec{\theta}_P$ are obtained from the measurement of potentiometers. If desired relative movement, $\Delta\vec{P}$=$[\Delta P_x\ \Delta P_y\ \Delta P_z]^T$, in original base ($O_0$) coordinate direction is given then we can easily find the next position of O_{7N}, O_{6N} and O_{5N}:

$$O_{7N} = O_{7P} + \Delta\vec{P} \tag{3.58}$$

$$O_{6N} = O_{6P} + \Delta\vec{P} \tag{3.59}$$

$$O_{5N} = O_{5P} + \Delta\vec{P} \tag{3.60}$$

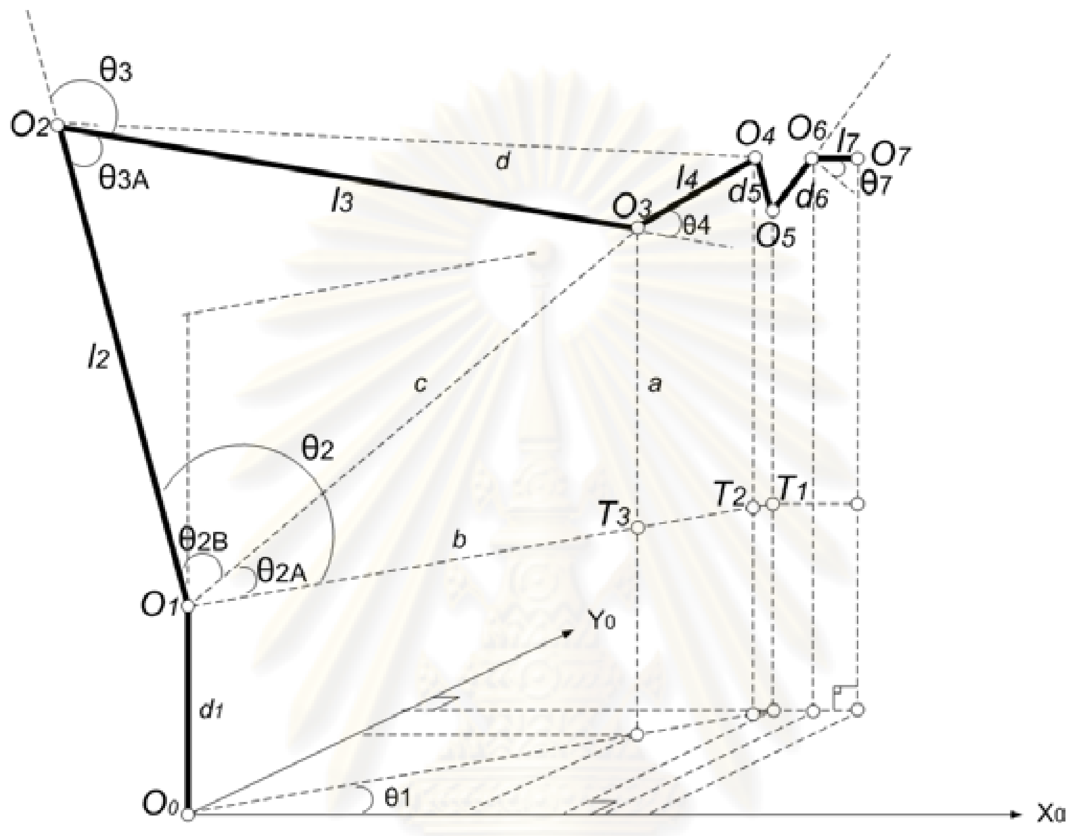


Figure 3.12: Coordinate frame example for section 3.1.3 strategy

Using known position of O_5, O_6 and O_7 then we can compute $\theta_1 ... \theta_5$ with formulae in (3.10) to (3.40). In this case, there is a difference formula on how to compute θ_6 compare to that one in section 3.1.1 because here we do not specify the orientation by α and β. Therefore we just need previous orientation as the next orientation.

Suppose that corresponding homogeneous matrix transformations of the present position ${}^0\mathbb{T}_7$ is denoted by ${}^0\mathbb{T}_{7P}$ whereas the next position ${}^0\mathbb{T}_7$, ${}^0\mathbb{T}_5$, ${}^5\mathbb{T}_6$ and ${}^6\mathbb{T}_7$ are denoted by ${}^0\mathbb{T}_{7N}$, ${}^0\mathbb{T}_{5N}$, A_{6N} and A_{7N} respectively. The next homogeneous matrix transformation ${}^0\mathbb{T}_{7N}$ is obtained from the present homogeneous matrix transformation ${}^0\mathbb{T}_{7P}$ using the following relation:

$${}^0\mathbb{T}_{7N} = {}^0\,\mathbb{T}_{7P} + \begin{bmatrix} 0 & 0 & 0 & \Delta\vec{P}_x \\ 0 & 0 & 0 & \Delta\vec{P}_y \\ 0 & 0 & 0 & \Delta\vec{P}_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.61}$$

To find θ_6, we use the following formula:

$${}^0\mathbb{T}_{7N} = {}^0\,\mathbb{T}_{5N}\,A_{6N}\,A_{7N} \tag{3.62}$$

$$A_{6N} =^0 \mathbb{T}_{5N}^{-1}\ {}^0\mathbb{T}_{7N}\ A_{7N}^{-1} \tag{3.63}$$

Then, we get θ_6 from this known matrix A_{6N}

$$\theta_6 = \cos^{-1} A_{6N(1,1)} \tag{3.64}$$

This is the general procedure for moving the end-point of robotic arm in coordinate O_0 direction with keeping its orientation:

1. Obtain the present Polar position $\vec{\theta}_P$ by measuring the potentiometer attached at every axis.
2. Using known $\vec{\theta}_P$ and general homogeneous transformation matrix in (2.8), compute: ${}^0\mathbb{T}_{7P} = A_{1P}A_{2P}A_{3P}...A_{7P}$.
3. Obtain the next position and its orientation ${}^0\mathbb{T}_{7N}$ using formula in (3.61).
4. Compute $\theta_{1N}...\theta_{5N}$ using formulae in (3.10) to (3.40).
5. Solve θ_{6N} using formula in (3.64).
6. Drive the robotic arm following the new Polar position $\vec{\theta}_N$.

3.1.4 Moving end-point in O_7 coordinate Direction with keeping its orientation

In this section, we design a similar strategy for moving the end-point of the robotic arm as in section 3.1.3. The same formulae and the same procedure are also employed. However, the relative movement will be specified in O_7 coordinate direction (not in O_0 coordinate direction). As addition, we constrain the movement only in X_7 or Y_7 or Z_7 exclusively.

The following procedure is a general step for moving the end-point of robotic arm in coordinate O_7 direction with keeping its orientation:

1. Obtain the present Polar position $\vec{\theta}_P$ by measuring the potentiometer attached at every axis.
2. Using known $\vec{\theta}_P$ and general homogeneous transformation matrix in (2.8), compute: ${}^0\mathbb{T}_{7P} = A_{1P}A_{2P}A_{3P}...A_{7P}$.
3. Assume that we have link-8 so that the position and orientation of O_8 is the same with O_7 exactly. Now we can derive a movement in O_7 coordinate direction using this pseudo link-8. Moving ΔP_x distance in X_7 direction means specifies $\alpha_8 = 0$, $d_8 = 0$, $l_8 = \Delta P_x$, and $\theta_8 = 0$. Corresponding homogeneous transformation matrix of axis-8 can be stated by $A_8 = A_i(\alpha = 0, d = 0, l = \Delta P_x, \theta = 0)$. Likewise, ΔP_y and ΔP_z distance movement in Y_7 and Z_7 direction respectively can be stated by $A_8 = A_i(\alpha = 0, d = 0, l = \Delta P_y, \theta = \pi/2)$ and $A_8 = A_i(\alpha = 0, d = \Delta P_z, l = 0, \theta = 0)$.
4. Compute the pseudo-position O_8=${}^0\mathbb{T}_{7P}\ A_8$.

5. Obtain the next position and its orientation ${}^{0}\mathbb{T}_{7N}$ using formula in (3.61), where ΔP_x=x_8-x_7, ΔP_y=y_8-y_7 and ΔP_z=z_8-X_z.

6. Compute $\theta_{1N}...\theta_{5N}$ using formulae in (3.10) to (3.40).

7. Solve θ_{6N} using formula in (3.64).

8. Drive the robotic arm following the new Polar position $\vec{\theta}_N$.

3.2 Trajectory File Generator

A trajectory pattern that contains a sequence of the robot movement in Cartesian coordinate is obtained by solving the inverse kinematics problem with some angular-to-digital conversions. This trajectory pattern is specified by an human operator in Cartesian coordinate. Corresponding file represents the sequence of the polar coordinate in digital value then is generated. This file is stored in Microsoft Excel format and so-called trajectory file.

Using inverse kinematics formulae in (3.44) to (3.57), a Cartesian coordinate will be converted into polar coordinate. Since the controller is a digital system, the polar coordinate value needs to be converted into digital value before proceeded by the controller. The relation between the angle in polar coordinate and the digital value is regressed by Linear Least Square method using data from experiment. This is called angular to digital value conversion.

The following is the least squares regression line at joint-i:

$$\hat{y_i} = m_i\theta_i + c_i \tag{3.65}$$

$$m_i = \frac{\sum_{j=1}^{N}(\theta_{i,j} - \bar{\theta}_i)(y_{i,j} - \bar{y}_i)}{\sum_{j=1}^{N}(\theta_{i,j} - \bar{\theta}_i)^2} \tag{3.66}$$

$$c_i = \bar{y}_i - m_i\bar{\theta}_i \tag{3.67}$$

where $\hat{y_i}$, θ_i, m_i, c_i, $\bar{\theta}_i$ and $\bar{y}_i$ denote the estimate of the angle in digital value, the angle, regression coefficient, a constant, the mean of θ_i and the mean of y at axis-i.

In this robotic arm, all of joints using rotary potentiometers to measure the angle in every joint except joint-3 which uses linear potentiometer. Generally, the linear regression line in (3.65) is used to state the relation between the angle in polar coordinate and the digital value. Particularly in joint-3, this linear regression line is used to state the relation between the length of L_p and the digital value as shown in Figure 3.13. The relation between the angle and the length of L_p is given in (3.68).

$$L_p(\theta_2,\theta_3) = \sqrt{(l_2cos\theta_2 + l_{31}cos(-(\theta_3+\theta_2)) - l_{21})^2 + ((l_2sin\theta_2 - l_{31}sin(-(\theta_3+\theta_2)))^2} \tag{3.68}$$

where l_{21} = 18 cm and l_{31} = 20 cm.

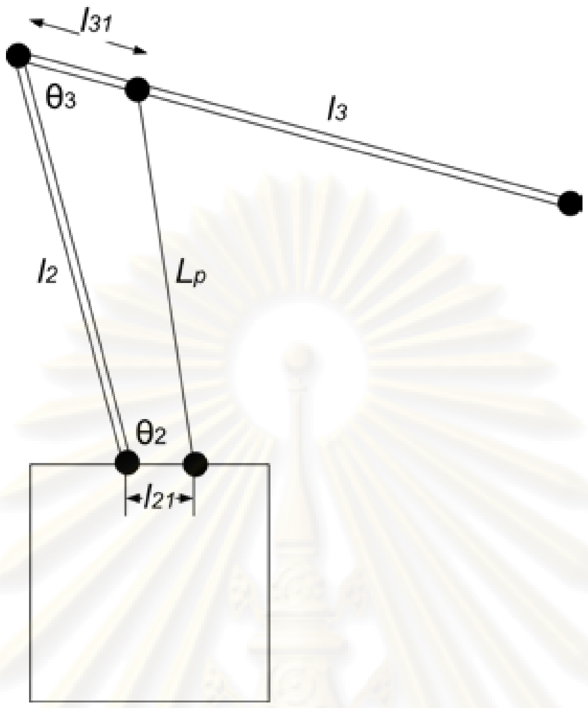


Figure 3.13: The structure of linear potentiometer at joint-3

The experiment was set up to find m_i and c_i as explain in (3.66) and (3.67). After measuring $\theta_{i,j}$ and $y_{i,j}$, parameters m_i and c_i are obtained and shown in Table 3.2.

Table 3.2: Regression Parameters

Axis-i	mi	c_i
1	33.7477	1950
2	50	-2400
3	-165.1325	17914
4	-12.5	2170
5	-11.4892	3080
6	-5.8889	2500

3.3 System Identification

The following section describes the system identification steps. The first step is usually the selection of a model structure. There are various possibilities of structure state-space and polynomial forms such as ARX, ARMAX, OE, BJ etc. The more complex the structure, the better estimate model is yielded. In this work, a high-accuracy model is not required, therefore ARX structure is chosen because of its simplicity. For a given choice of structure, the order of the model needs to be specified before the corresponding parameters are estimated.

In order to estimate the servo valves of the robot, a set of data input and output from experiment are required. As an illustration, the 6-th axis servo valve estimation is given to explain its step using System Identification Toolbox provided by Matlab.

3.3.1 Delay Estimation

The amount of delay (d) as shown in (2.46) also infulence the selection of the model order. Therefore we need to estimate the best delay before selecting the model order. There are various options available for determining the time delay from input to output. However, this work will focus only on one of them, that is using *delayest* utility. This function evaluates an ARX structure as in (3.69), where u, y, na, nb and d respectively are the input signal, the output signal, the order of input, the order of output and the time delay.

$$y_k + a_1 y_{k-1} + ... + a_{na} y_{k-na} = b_1 u_{k-d} + ... + b_{nb} u_{k-nb-d+1} \tag{3.69}$$

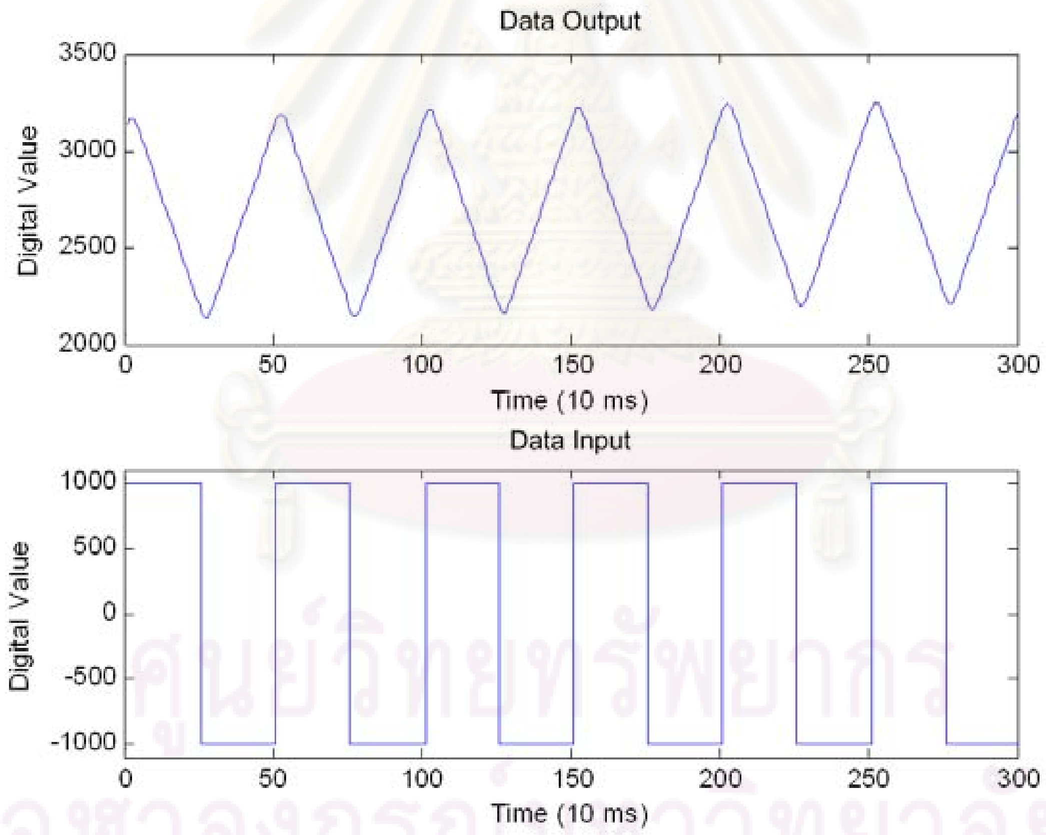


Figure 3.14: The dataset measured at axis-6

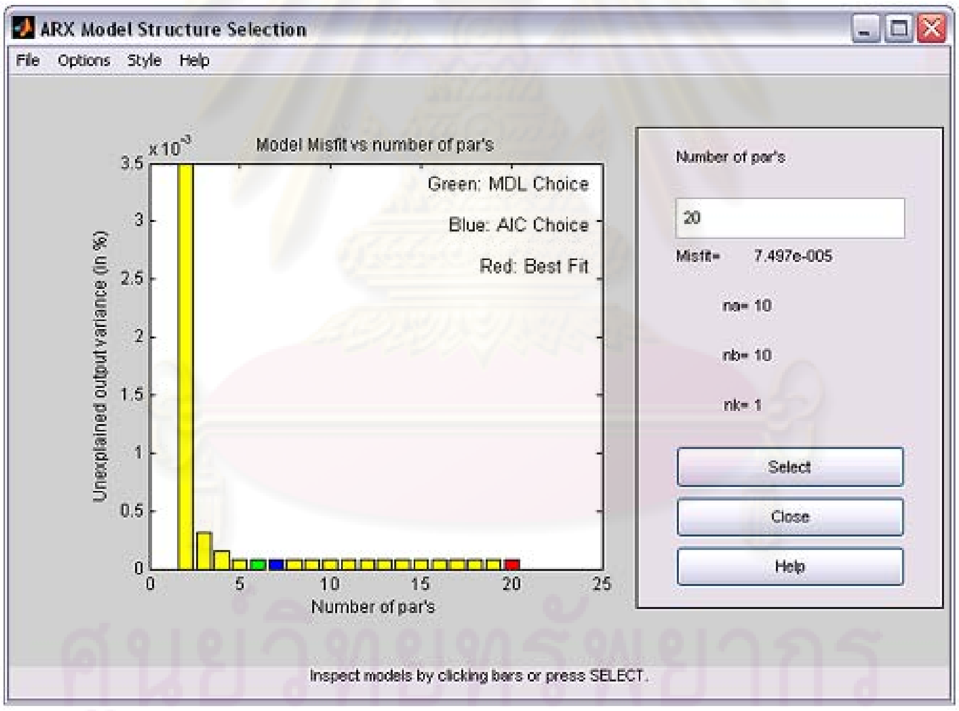
Assume that we have two set of data input and output ($x1$,$y1$ and $x2$,$y2$) which are obtained from measurement at axis-6 as shown in Figure 3.14. Here is the Matlab syntax to estimate the delay using *delayest* toolbox.

```
>>  cdata = iddata(y1,x1);        %  data test
>>  vdata = iddata(y2,x2);        %  data validation
>>  delay = delayest(cdata);
    delay = 1
```

This result shows that the best time delay (d) that should be selected is 1.

3.3.2 Order Selection

Once we have decided the model structure to use, the next task is to determine the order(s). In general, the selected order should give as close as possible to the real model. However, it should be not higher than necessary. This can be determined by analyzing the improvement in fit as a function of model order. When doing this, it is advisable to use a separate, independent dataset for validation. Choosing an independent validation dataset (*vdata* in this example) would improve the detection of over-fitting.


ARX Model Structure Selection
File Options Style Help
Model Misfit vs number of par's
x10^-3
Unexplained output variance (in %)
Number of par's
Green: MDL Choice
Blue: AIC Choice
Red: Best Fit
Select
Close
Help
Inspect models by clicking bars or press SELECT.

Number of par's

20

Misfit= 7.497e-005

na= 10

nb= 10

nk= 1

Figure 3.15: The ARX model structure selection

In addition to a progressive analysis of multiple model orders, explicit determination of optimum orders can be performed for some model structures. Functions *arxstruc* and *selstruc* are used for choosing the best order for ARX models. In this work, MATLAB 7.0 is employed to facilitate system identification. We check the fit for all 100 combinations of up to 10 $B(z)$-parameters and up to 10 $A(z)$-parameters as in (2.46), all with a delay value of 1:

```
>>  V = arxstruc(cdata,vdata,struc(1:10,1:10,1));
```

The best fit for the validation data set is obtained by:

```
>>  nn = selstruc(V,0);
    nn = 10    10    1
```

For choosing the model order interactively, the following syntax can be used:

```
>>  nns = selstruc(V);
```

Figure 3.15 shows the ARX model structure selection panel. From this figure we know that the best fit model is performed when na=10, nb=10 and the time delay d=1 as shown by the red color bar. If we use *Akaike Criterion*, we get the best order model when na=1, nb=6 and d=1 as shown by the blue color bar. A simple 2nd order ARX yields only 3.5 x 10^{-3} *unexpected output variance*. This structure has good enough approximate of the model and nowever can be well exploited to estimate the controller as proven by the experiment result in section 5.7.

3.4 Controller Design

Due to result in previous section, 2-DOF PID controller will be designed based on the approximate model of the robot. For the reason of simplicity, 2nd order of plant model and 2nd order of 2-DOF PID controller with pole placement method are chosen. Assume that the approximate model, $H(z)$, obtained by identification is given in (3.70) and the controller structure is as shown in Figure 2.4 (b).

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{3.70}$$

A 2-nd order controller parameter is stated by:

$$S(z) = s_0 + s_1 z^{-1} + s_2 z^{-2} \tag{3.71}$$

$$R(z) = (1 - z^{-1})(1 + r z^{-1}) = 1 + (r - 1) z^{-1} - r z^{-2} \tag{3.72}$$

A polynomial $P(z)$ represents the desired pole locations is specified by:

$$P(z) = 1 + p_1 z^{-1} + p_2 z^{-2} + p_3 z^{-3} + p_4 z^{-4} \tag{3.73}$$

In this design, polynomial $R(z)$ is not in general form. By reconfiguring (2.54) we have

$$x = [s_0, s_1, s_2, r]^T \tag{3.74}$$

$$M = \begin{bmatrix} b_1 & 0 & 0 & 1 \\ b_2 & b_1 & 0 & a_1 - 1 \\ 0 & b_2 & b_1 & a_2 - a_1 \\ 0 & 0 & b_2 & -a_2 \end{bmatrix} \tag{3.75}$$

$$K = \begin{bmatrix} p_1 + 1 - a_1 \\ p_2 + a_1 - a_2 \\ p_3 + a_2 \\ p_4 \end{bmatrix} \tag{3.76}$$

$$x = M^{-1}K \tag{3.77}$$

The following is an example of 2-DOF controller design for joint-1. The estimate model of plant is given in (3.78) and the desired pole locations are multiple pole at 0.8 as shown in (3.79). In order to guarantee that the controller system is stable, the poles must be chosen so that they are located inside unity circle. The closer of the poles to the zero, the faster of settling time to be stable. However the closer of the poles to the zero will affect to the vulnerability of its stability. Therefore, locating pole at 0.8 is one of good choices regarding to those reasons.

$$H(z) = \frac{B(z)}{A(z)} = \frac{-0.0026z^{-1} - 0.0116z^{-2}}{1 - 1.4825z^{-1} + 0.4821z^{-2}} \tag{3.78}$$

$$P(z) = (1 - 0.8z^{-1})^4 = 1 - 3.2z^{-1} + 3.84z^{-2} - 2.048z^{-3} + 0.4096z^{-4} \tag{3.79}$$

Then the matrix M and vector K are stated by:

$$M = \begin{bmatrix} -0.0026 & 0 & 0 & 1 \\ -0.0116 & -0.0026 & 0 & -2.4825 \\ 0 & -0.0116 & -0.0026 & 1.9646 \\ 0 & 0 & -0.0116 & -0.4821 \end{bmatrix} \tag{3.80}$$

$$K = \begin{bmatrix} -0.7175 \\ 1.8754 \\ -1.5659 \\ 0.4096 \end{bmatrix} \tag{3.81}$$

The solution is:

$$x = M^{-1}K = [-6.8755 \;\; 11.5107 \; -4.7479 \; -0.7354]^T \tag{3.82}$$

The parameters of the controller are

$S(z) = -6.8755 + 11.5107z^{-1} - 4.7479z^{-2}$

$R(z) = 1 - 1.7354z^{-1} + 0.7354z^{-2}$

Finally, $T(z)$ can be computed using (2.44)

$$T = S(1) = -0.1127 \tag{3.83}$$

3.5 Hardware Design

This section explains the hardware requirement. In general, the hardware is classified into four parts i.e. main processor board, analog board, digital I/O board and power supply board.

3.5.1 Main Processor Board

The main processor board is the central processing unit of the controller system. This part is responsible on receiving, processing and transmitting data from and to external devices. This part composed of several internal peripherals such as 32-bit soft-core processor, UART, Digital I/O, Timer/Counter, SPI, SPI reader and PWM generator. All of mentioned peripherals are embedded in a Xilinx FPGA XC3S400.

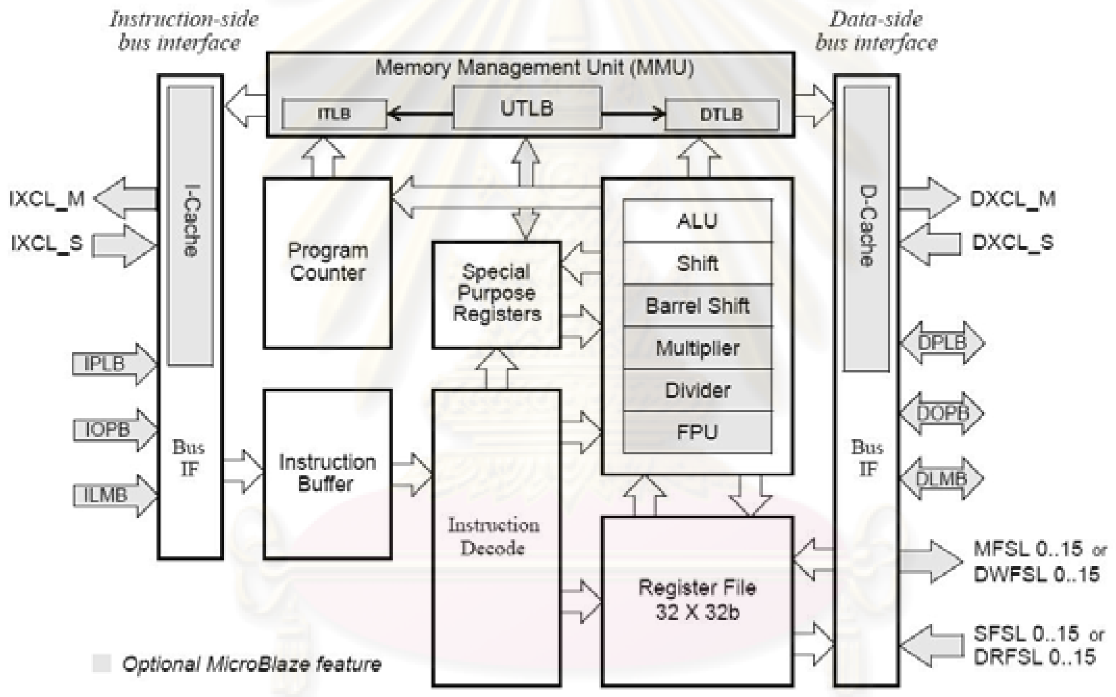


Figure 3.16: MicroBlaze core block diagram

3.5.1.1 MicroBlaze 32-bit Soft-core Processor

Microblaze is a 32-bit soft-core processor developed by Xilinx. This is a reduced instruction set computer (RISC) optimized for implementation in Xilinx Field Programmable Gate Arrays (FPGAs). Figure 3.16 shows a functional block diagram of the MicroBlaze core.

The basic MicroBlaze architecture is consists of 32 general-purpose registers, an Arithmetic Logic Unit (ALU), a shift unit, and two levels of interrupt. This basic design can then be configured with more advanced features such as: memory management unit (MMU), barrel shifter, memory management/memory protection unit, floating-point unit (FPU), caches, exception handling, debug

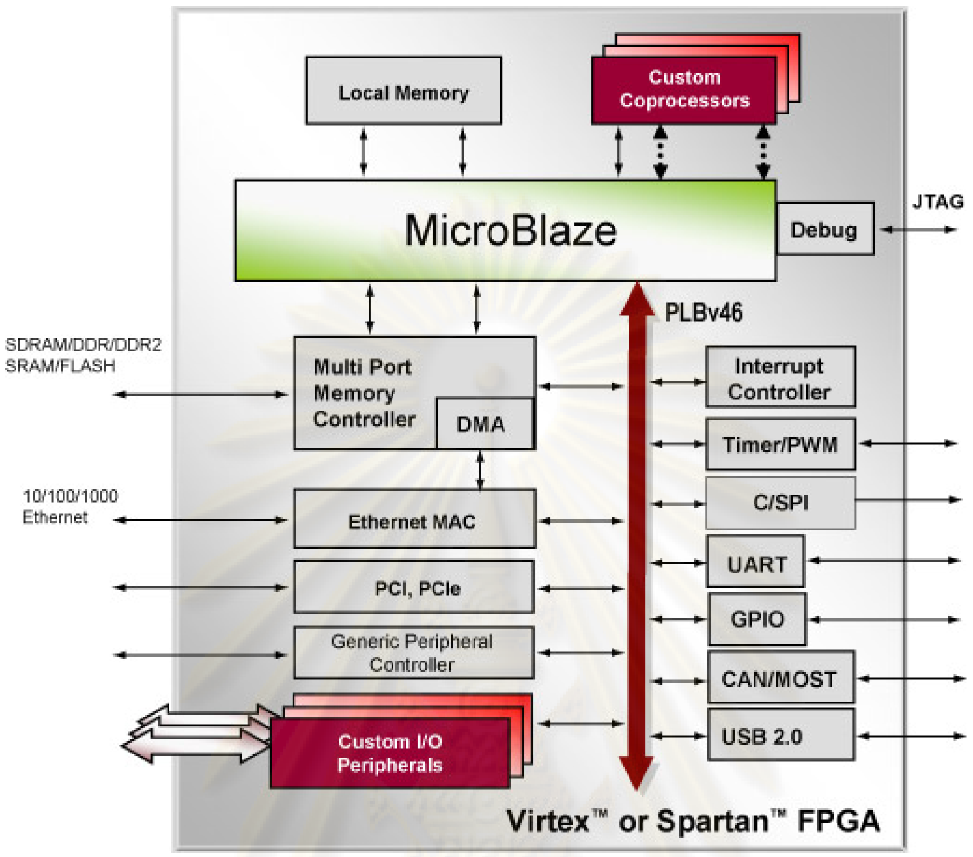


Figure 3.17: MicroBlaze core with advanced features

logic, and others. This flexibility allows the user to balance the required performance of the target application against the logic area cost of the soft processor. Figure 3.17 shows a MicroBlaze system configured with advanced features.

Three types of memory interfaces are supported: Local Memory Bus, Processor Local Bus (PLB) / On-Chip Peripheral Bus (OPB) and Xilinx CacheLink (XCL). In this design, Microblaze system is configured with 16 KB local/internal memory, active low reset, 50 MHz bus clock frequency and several advanced feautures which are connected via PLB Bus as described in subsection 3.5.1.2 to 3.5.1.7.

3.5.1.2 UART

The UART peripheral is connected to the PLB Bus to provide the controller interface for asynchronous serial data transfer as shown in Figure 3.18.

This peripheral performs parallel to serial conversion on character reveived through PLB and serial to parallel conversion on characters received from external serial devices. The UART peripheral is capable of transmitting and receiving 8,7,6 or 5 bit characters with 1 bit stop and 1 bit parity.

There are three sections inside the UART peripheral. First, the PLB interface module provides the interface to the PLB and implements PLB protocol logic. Second, the register module consist

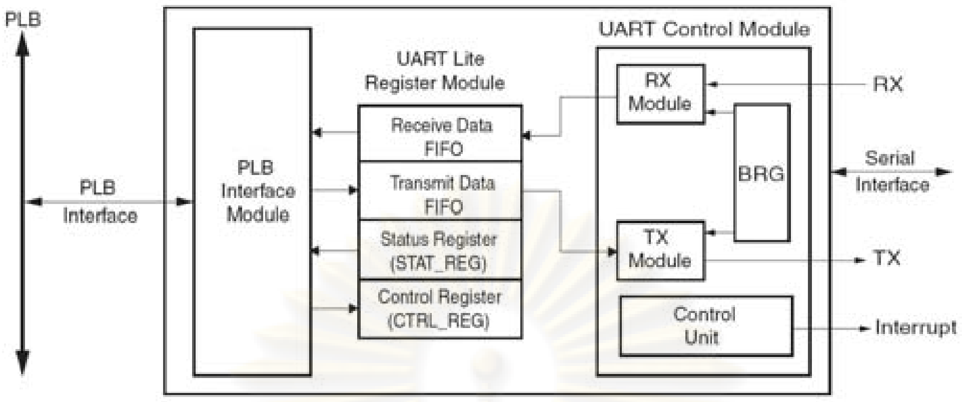


Figure 3.18: Block diagram of UART peripheral

of an 8-bit status register, an 8-bit control register and a pair of 8-bit Transmit/Receive registers. All registers are accessed directly from the PLB using the PLB module. Third, the control module consists of a RX module, a TX module, a parameterized baud rate generator (BRG) and a control unit.

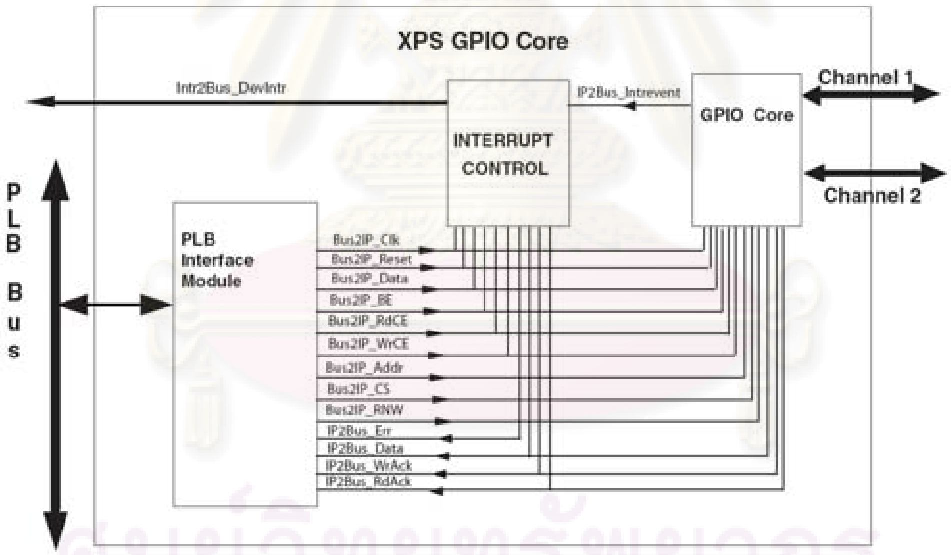


Figure 3.19: Block diagram of GPIO peripheral

3.5.1.3 Digital I/O

The digital I/O peripheral is a general purpose Input/Output (GPIO) to interface the PLB Bus with external devices. It is a 32-bit bi-directional data transfer which can be configured either as input or as output at a time.

An input port may be configured to take its external input either from bidirectional 3-state pin or from the dedcated input only pins. As a ouput port, the data is driven out through a 3-state buffer as well as to an input only pin. The port can be configured dynamically for input or output by enabling or disabling the 3-state buffer.

The bock diagram of GPIO is shown in Figure 3.19. It is composed of 3 modules i.e PLB interface, Interrupt Control and GPIO core. It can be configured as a single or a dual channel device. The channel width and the direction are configurable and can be enabled individually.

3.5.1.4 Timer/Counter

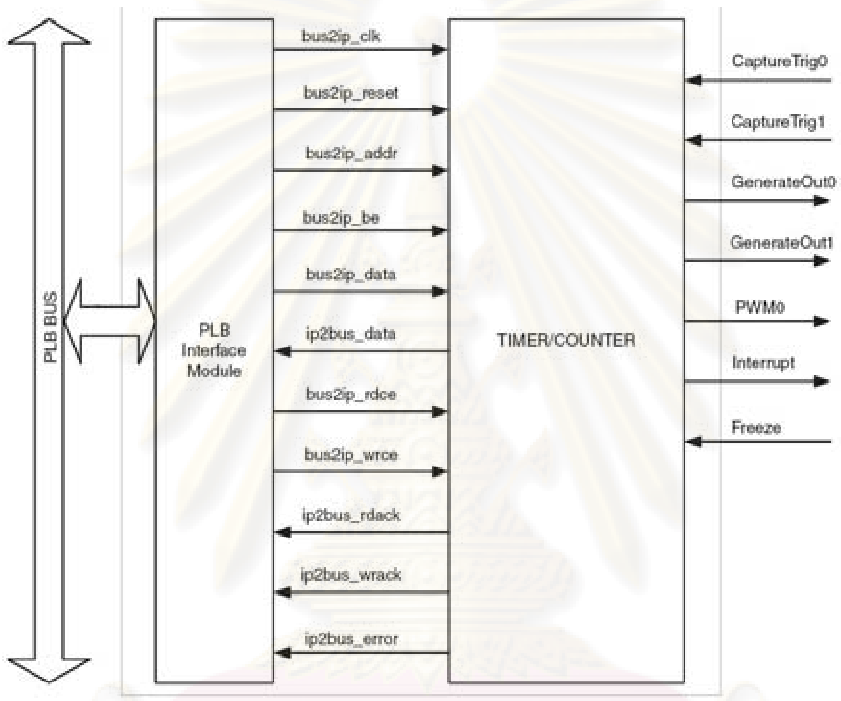


Figure 3.20: Block diagram of Timer/Counter peripheral

This Timer/Counter peripheral is a 32-bit timer module that attaches to the PLB bus. It is organized as two identical timer modules as shown in Figure 3.20. Each timer module has an associated load register that is used to hold either the initial value for the counter for event generation or a capture value depending on the mode of the timer.

There are three modes that can be used with the two Timer/Counter modules:

- Generate Mode. The value in the load register is loaded into the counter. When enabled, the counter beings to count up or down depending on the selection of the Timer Control Status Register. This mode is useful for generating repetitive interrupts or external signals with a specified interval.

- Capture Mode. The value of the counter is stored in the load register when the external capture signal is asserted. This mode is useful for time tagging external events while simultaneously generating an interrupt.

- Pulse Width Modulation (PWM) Mode. Two timer/counters are used as a pair to produce an output signal (PWMo) with a spefified frequency and duty factor. Timer0 and timer1 are used to set the period and the high interval time of the PWM respectively.

3.5.1.5 SPI

The SPI peripheral connects to the PLB bus and provides a serial interface to SPI devices. The SPI protocol provides a simple method for a master and as selected slave to exchange data. SPI is a full-duplex synchronous channel that supports four-wire interface (transmit, receive, clock and slave-select) between a master and a selected slave. The block diagram of SPI peripheral is shown in Figure 3.21.

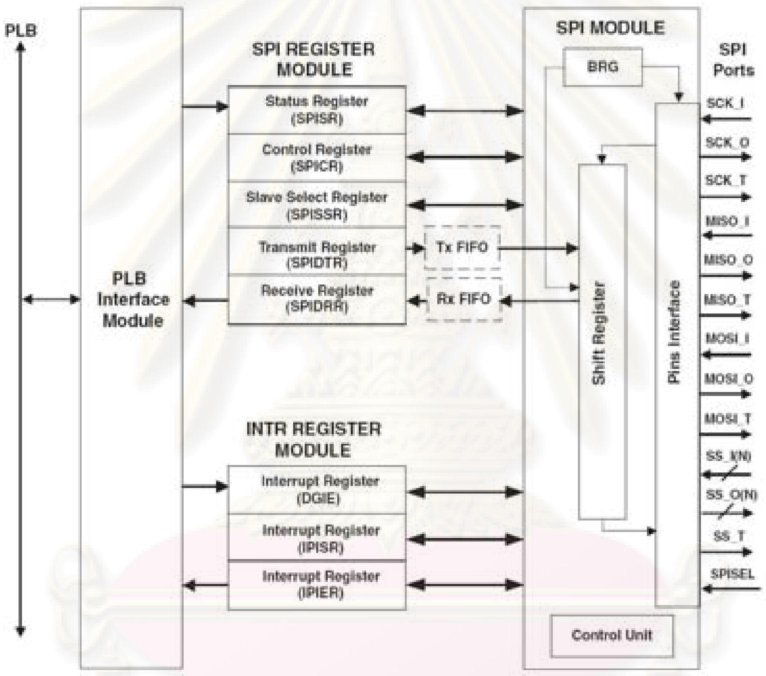


Figure 3.21: Block diagram of SPI peripheral

When configured as a master, the SPI can communicate with both off-chip and on-chip slaves. However, when configured as a slave, it can communicate only with on-chip masters. The number of slaves is limited to 32 by the size of the *Slave Select Register*. However, the number of slaves and masters will impact the achievable performance.

3.5.1.6 SPI reader

The SPI reader peripheral is a custom logic 16-bit shift register which is compatible with SPI protocol. However it is only able to perform read access and act as a slave module. As shown in Figure 1.1, there are 6 channels of ADC accessed by SPI protocol. The ADCs are desired to be accessed concurrently. It is redundant to use 6 SPI peripheral because it consumes more resources. Therefore a custom SPI reader is developed to optimize and reduce the resources usage. A single command can be sent to all

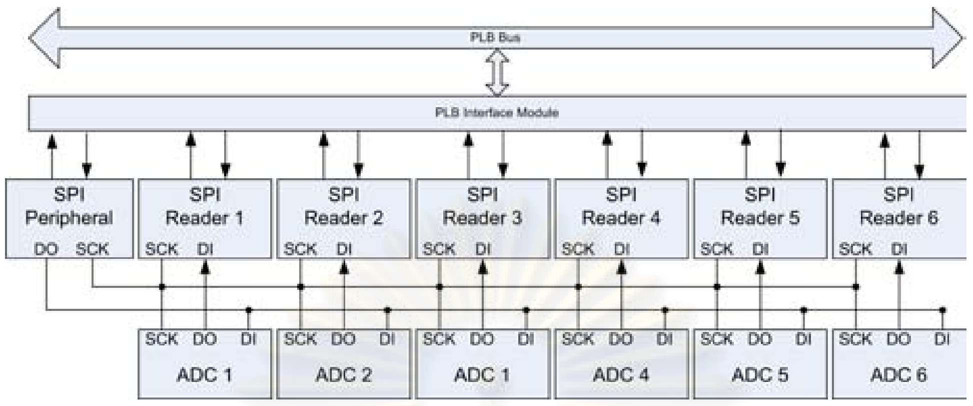


Figure 3.22: Block diagram of SPI reader

of ADCs while 6 SPI readers are used to read the data from every channel concurrently. Figure 3.22 shows the block diagram of SPI reader module with their connections to ADCs.

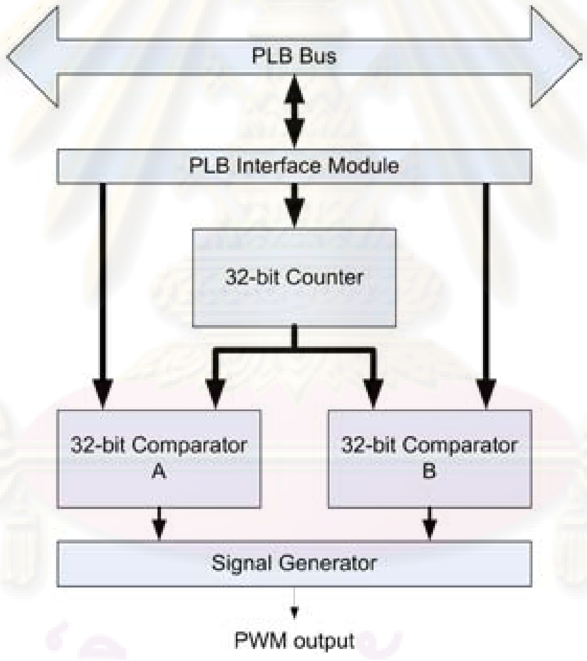


Figure 3.23: Block diagram of PWM generator

3.5.1.7 PWM Generator

The PWM generator peripheral is a custom logic hardware which is dedicated specifically to generate PWM signal. Figure 3.23 shows the block diagram of PWM generator.

The PWM generator composed of a 32-bit counter register and two 32-bit comparator registers. The counter data and the comparators data are defined by sending data through PLB bus. The PLB interface module then store that data into counter register and comparator registers. When rising clock is happened, the counter register value will be incremented by 1 and at the same time this counter value is compared with both comparator registers value. When the value of counter register is greater than comparator register A value then PWM ouput signal is forced to be ’0’. When the value of counter register is greater than comparator register B value then PWM output signal is forced to ’1’ and register counter value is reset to 0 (zero). The PWM signal generation can be illustrated as shown in Figure 3.24.

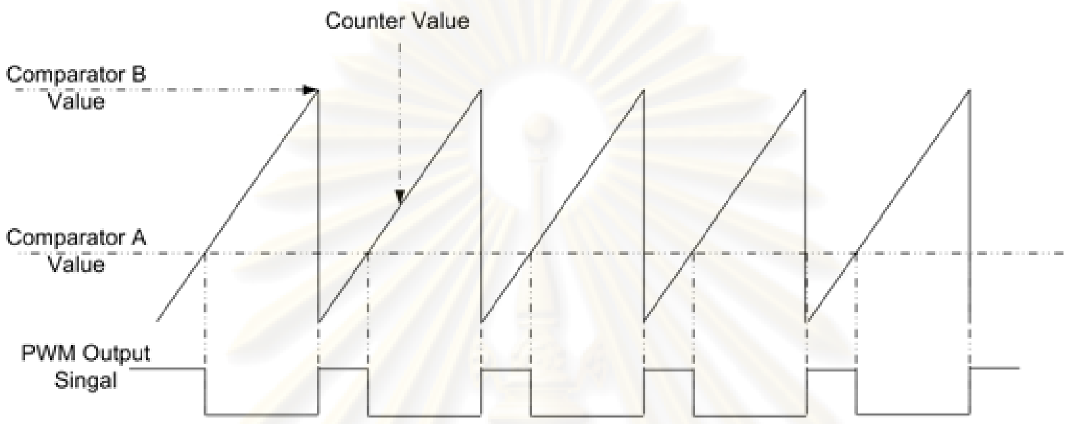


Figure 3.24: PWM output signal

3.5.2 Analog Board

Analog board composed of three parts: digital isolator, filter-amplifier and analog to digital converter (ADC). The block diagram of Analog board is shown in Figure 3.25.

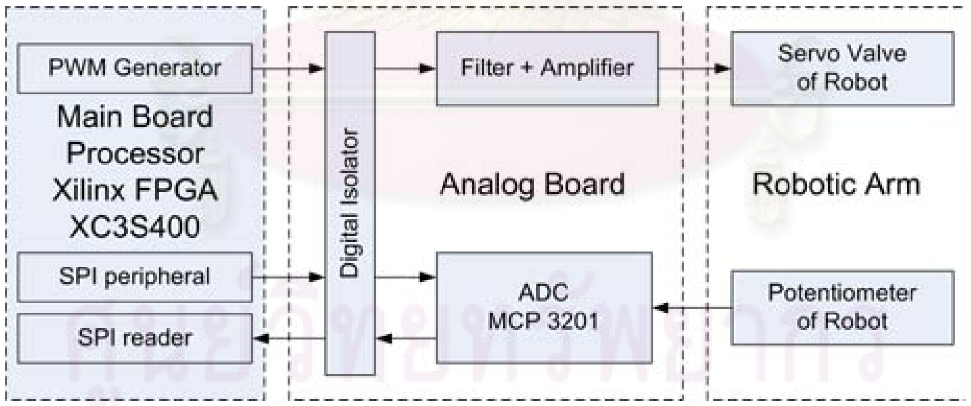


Figure 3.25: Block diagram of Analog Board

3.5.2.1 Digital Isolator

The digital isolator is used to isolate the power supply between the main processor board and the analog board. Those power supply must be separated in order to reduce and avoid the noise propagation from or to external devices. ADuM14xx digital isolator is chosen because it has very high switching speed (up to 90 Mbps), low pulse width distortion (less than 2 ns) and provides four independent isolation channels in a variety of channel configurations and data rates. Figure 3.26 shows the block diagram of ADuM14xx.

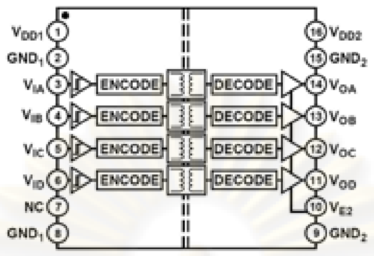


Figure 3.26: Block diagram of ADuM14xx

3.5.2.2 Filter and Amplifier

The filter and amplifier circuit are respectively used to convert the PWM signal into analog signal and to amplify the analog signal. The filter is designed specifically to satisfy the PWM characteristics. In a typical PWM signal, the base frequency is fixed but the pulse width is a variable. The produced analog signal is directly proportional to the duty cycle (D) of the PWM signal. A typical PWM signal is shown in Figure 3.27.

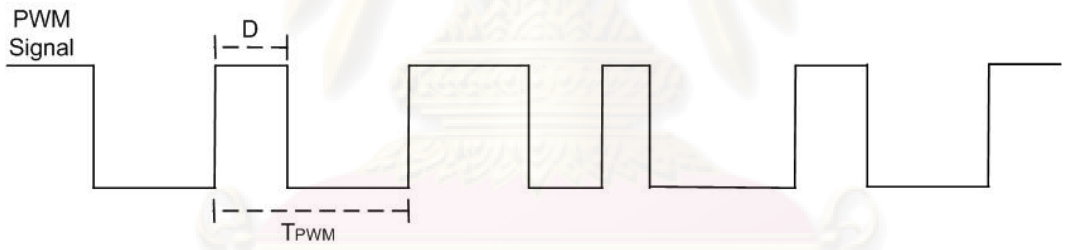


Figure 3.27: A typical PWM signal

A Fourier analysis shows that there is a strong peak at frequency f_{PWM} and other strongs harmonics at frequency nf_{PWM} as shown in Figure 3.28, where n is an integer and f_{PWM} is the base frequency of PWM signal. These peaks are unwanted noise and should be eliminated. To eliminate these noise a low-pass filter with frequency cut off (f_c) lower than f_{PWM} must be employed.

In this design, the PWM base frequency (f_{PWM}) is fix determined by loading 4096 at register comparator B in Figure 3.24. With 100 MHz source clock, it yields 40.96 us periode of PWM (T_{PWM}) or equal to 25 KHz frequency of PWM (f_{PWM}). Therefore the low-pass filter must have frequency cut off less than 25 KHz. The lower f_c the better analog signal is yielded. However, one must be considered that a sinusoidal waveform will be generated at frequency 200 Hz. Hence the f_c of the low-pass filter must be selected so that it is less than 25 KHz but greater than 200 Hz. In this case, f_c = 232.27 Hz is chosen. This low-pass filter is realized by using 2-nd order passive filter with R_1= 10 KOhm, R_2=100 KOhm, C_1=0.1 uF and C_2= 4700 pF as shown in Figure 3.29 and equation in (3.84).

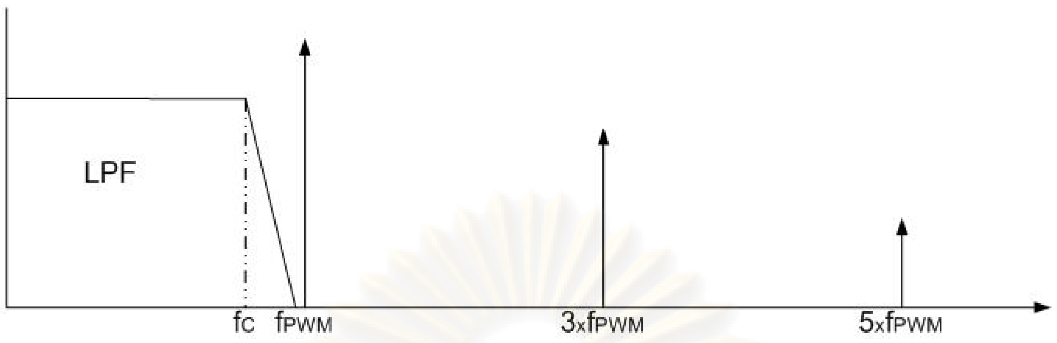


Figure 3.28: The spectrum frequency of PWM

$$f_c = \frac{1}{2\pi\sqrt{R_1 R_2 C_1 C_2}} = \frac{1}{2\pi\sqrt{10 * 100 * 0.1 * 4700 * 10^{-12}}} = 232.27 \ \ Hz \tag{3.84}$$

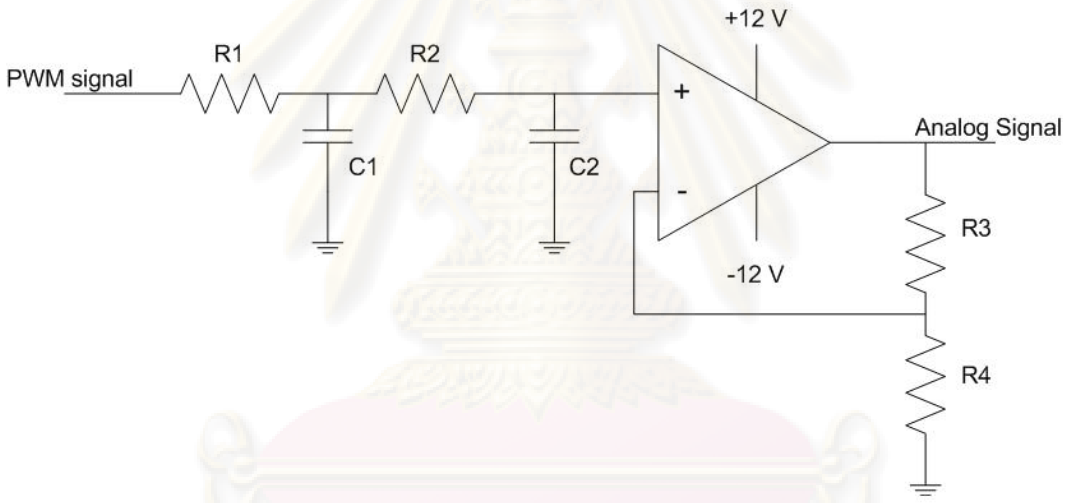


Figure 3.29: The circuit of filter and amplifier

The analog signal produced by low-pass filter will be used to drive the servo valve of the robotic arm. Since the PWM signal level are -2.5 V and +2.5 V, the output of analog signal needs to be amplified so that it is powerful enough to drive the servo valve. In this design, analog signal is amplified 4.6 times by choosing R_3=36 KOhm and R_4=10 KOhm as computed in equation (3.85), where V_i and V_o are the input and the output signal respectively. Therefore the analog signal output span becomes -11.5 V to +11.5 V.

$$V_o = (1 + \frac{R_3}{R_4})V_i = (1 + \frac{36}{10})V_i = 4.6V_i \tag{3.85}$$

3.5.2.3 Analog to Digital Converter

The analog to digital converter is used to convert the analog signal obtained by measuring the potentiometer voltage into digital value. The potentiometer is attached in every joint of robotic arm to represent the angle. In other word, the ADC is used to convert the measured angle of joint into digital value.

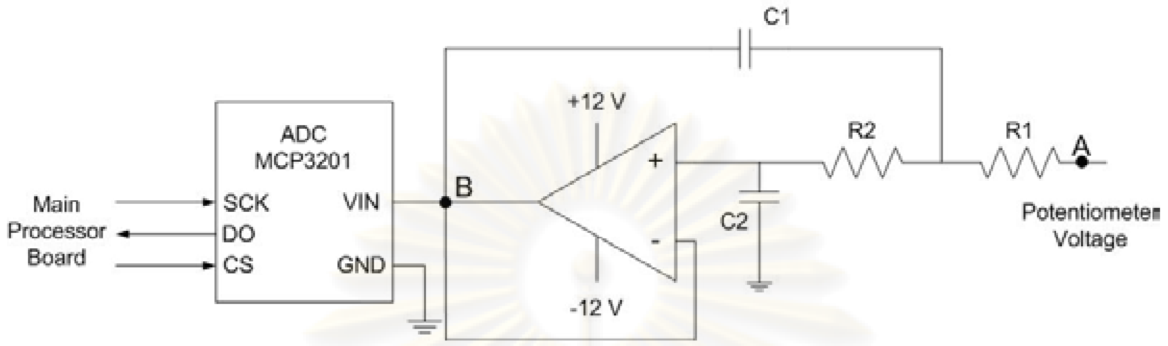


Figure 3.30: The circuit of analog to digital converter

A single channel ADC (MCP3201) is chosen in this design because its features satisfy the system requirement. Moreover, it has very low price compare to other multi-channels ADCs. MCP3201 has 12-bit resolution, 100 KSPS maximum sampling rate, SPI protocol compatible and single supply operation up to +5 V. The ADC circuit is shown in Figure 3.30.

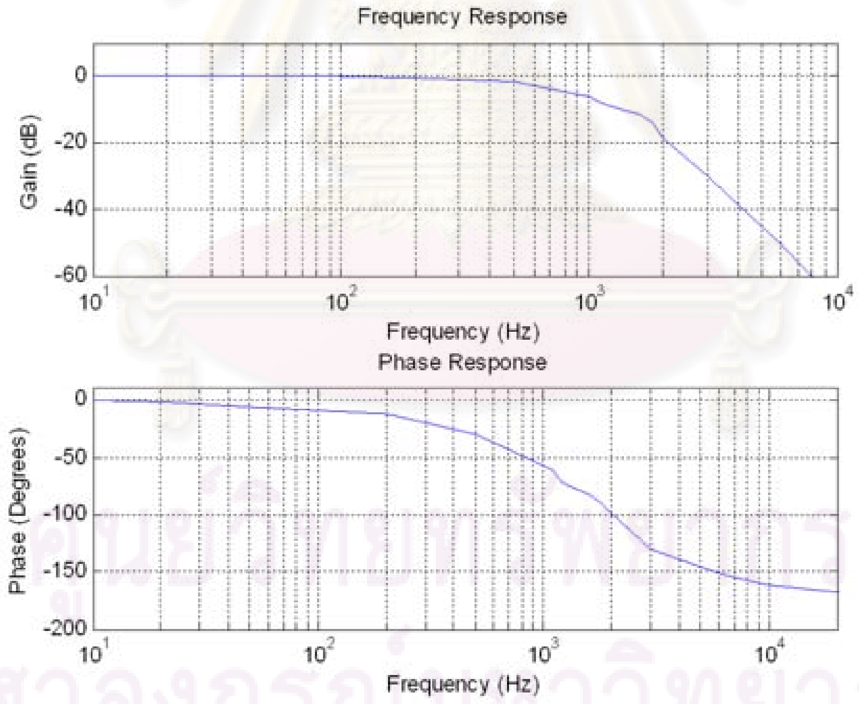


Figure 3.31: The active filter frequency response

A 2-nd order active low-pass filter with unity gain is placed between measured potentiometer voltage and the ADC. This filter is used to filter high frequency noise from external and functioned as a buffer to isolate the input load. Because the measured signal is changed very slow, frequency cut off f_c=1592.4 Hz is a good value to be selected. This filter is realized by choosing R_1=R_2= 1 KOhm and C_1=C_2=0.1 uF as computed in (3.86).

$$f_c = \frac{1}{2\pi\sqrt{R_1R_2C_1C_2}} = \frac{1}{2\pi\sqrt{1*1*0.1*0.1*10^{-12}}} = 1592.4\ \ Hz \qquad (3.86)$$

The frequency response and the phase response of the designed filter are shown in Figure 3.31. This data is obtained by giving some sinusoidal waveforms wit at point A and then measure the output at point B

3.5.3 Digital I/O Board

The digital I/O part is used to interface the digital input-output signal between main processor and external devices. Solid state relays LH150 and opto-couplers CNY173 are applied as output and input interfaces respectively as shown in Figure 3.32. This interface solves the DC level mismatch problem (main processor voltage is 0-3.3 V; whereas those of I/Os are either 0-12V or 0-24V) and noisy environment problem.

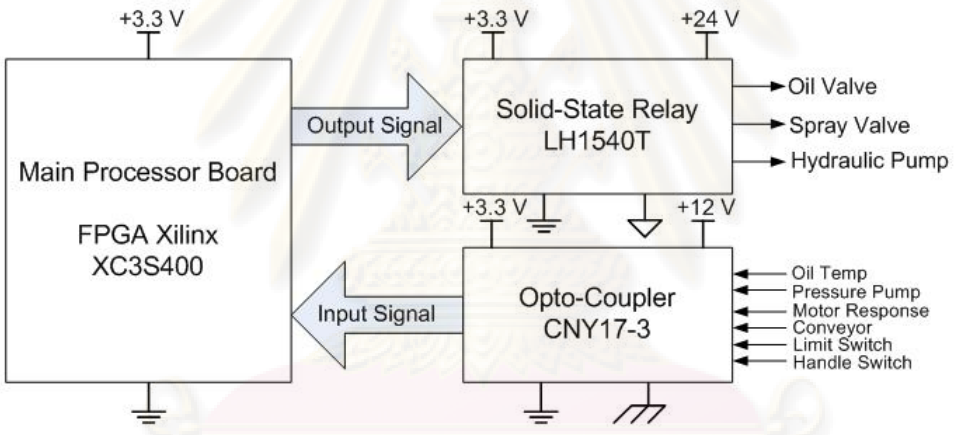


Figure 3.32: Block diagram of Digital I/O

3.5.4 Power Supply Board

Several DC voltages are required to supply power for main processor board, analog board and digital I/O board. In order to eliminate the noise interference among those board, the DC voltages should be isolated from each others. In this design, a multi secondary side transformer was constructed to produce eight constant voltage supplies with four isolately ground as shown in Fig. 3.33. Furthermore, the current requirement for all supplies are less than 0.5 A, that linear regulation therefore seems to be a good selection for this purpose. By isolating the power supply, it will reduce the noise and interference from both external noisy factory environment and among internal parts.

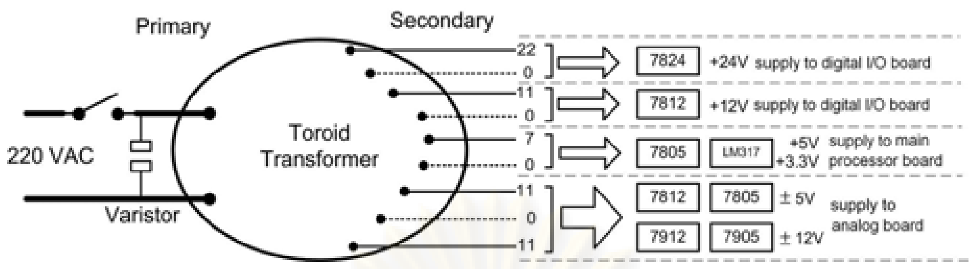


Figure 3.33: Power supply unit

3.6 Software

This section concern with the software of the controller system. Physically, the hardware provides 3 buttons as input interface used to command the controller system software: 2 general buttons and 1 emergency button as shown in Figure 5.14.

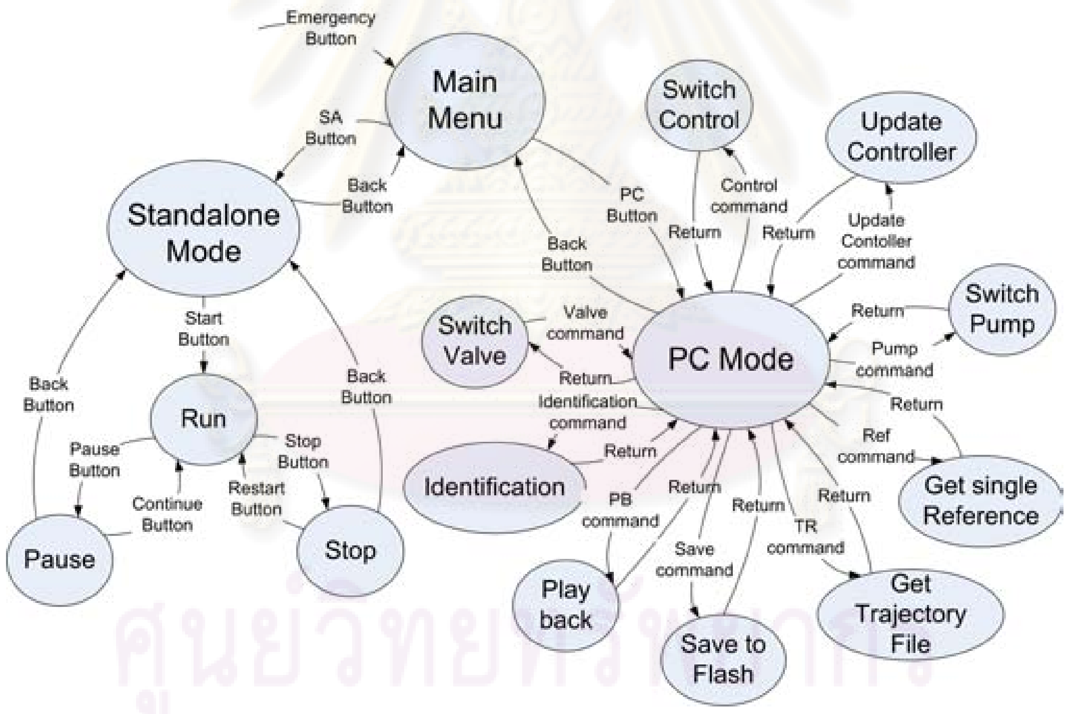


Figure 3.34: The state machine diagram of the software

3.6.1 Main Menu

Figure 3.34 shows the state machine diagram of the software. The main software consist of two main states: *standalone mode* and *PC mode*. At the *main menu* state, the state when the controller system just ran, some peripheral initializations are done. Initializations are required to set how each peripheral works. *UART* initializes the serial communication between controller system and PC such as port, baudrate, stop bit and so on. *SPI* initializes the communication between controller system and ADC. *PWM* initializes the duty cycle and the base frequency of PWMs. *I/O* initializes general input and output pins. *Controller* initializes the PID controllers parameter in their initial value. *LCD* initializes the LCD appearance. *Interrupt* initializes the interrupt service requests.

Having finished the initialization, the software come to a continuous-looping. The looping will never end unless one of command buttons is pressed. Pressing *SD* button brings the state to *standalone mode*. Likewise, pressing *PC* button brings the state to *PC mode*. Conversely, pressing *back* button will back its state to the *main menu*.

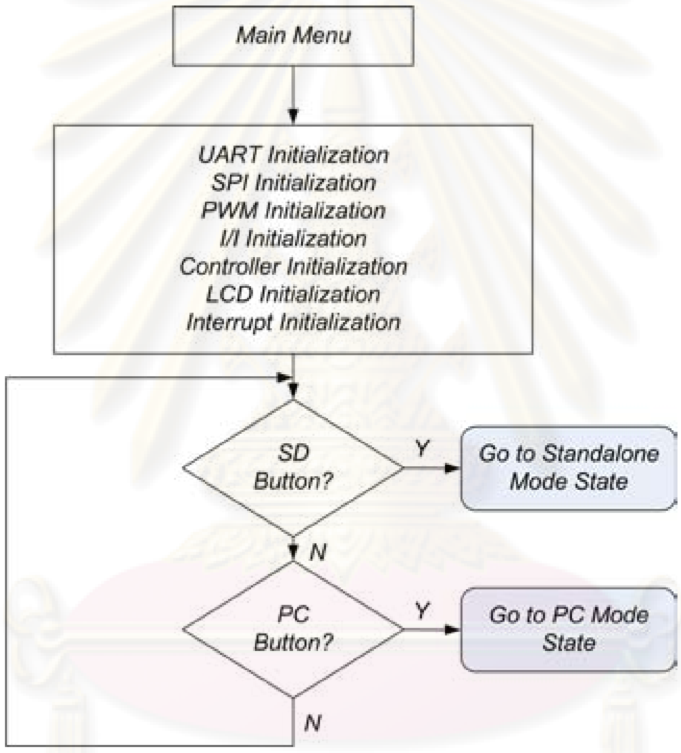


Figure 3.35: Flowchart diagram of software main menu

3.6.2 Standalone Mode State

Figure 3.36 shows the flowchart diagram of *standalone mode* state. A second after *standalone mode* is chosen then the system will check the existence of the trajectory file. If there is no trajectory file available in the Flash ROM then it shows an empty message on LCD display then goes to *main menu* state. Otherwise, it goes to a continuous looping until either *start* button or *back* button is pressed. Pressing *start* button brings its state to *start* whereas pressing *back* button brings its state to the *main menu*.

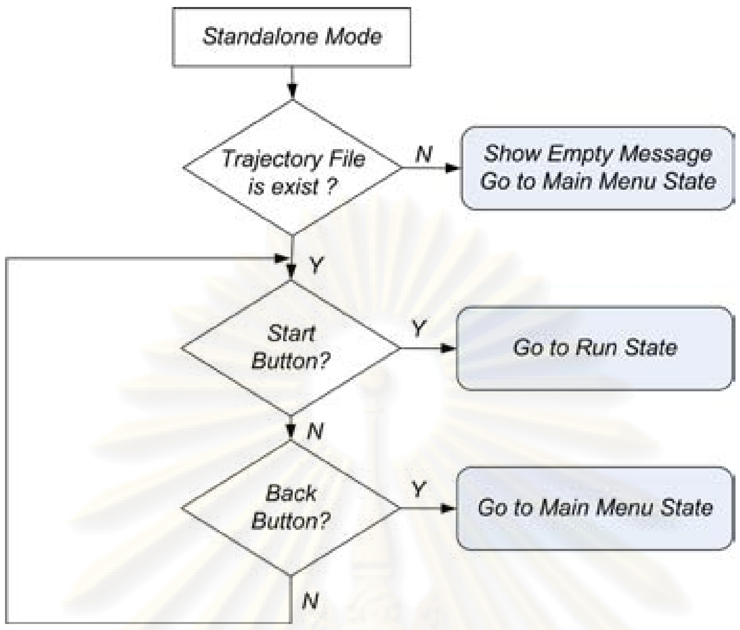


Figure 3.36: Flowchart diagram of standalone mode

3.6.3 Run State

In *run* state, the trajectory file will be loaded from the Flash ROM into main memory of the system. Then the robot will be driven so that it moves following the pattern stored in the trajectory file. Pressing *stop* button brings its state to *stop* whereas pressing *pause* button brings its state to *pause*.

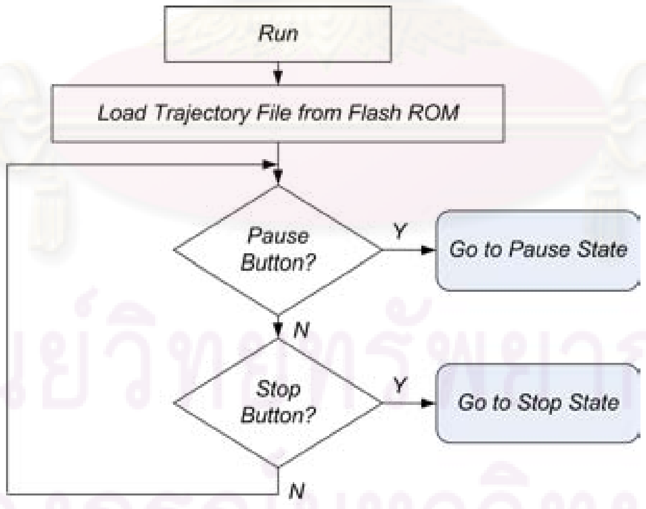


Figure 3.37: Flowchart diagram of run state

3.6.4 Pause State

When *pause* button is pressed, controller system holds the present position and waits until the next command is given. Pressing *continue* button lets the robot continuing its motion and brings its state back to *run* state whereas pressing *back* button brings it to *standalone mode* state.

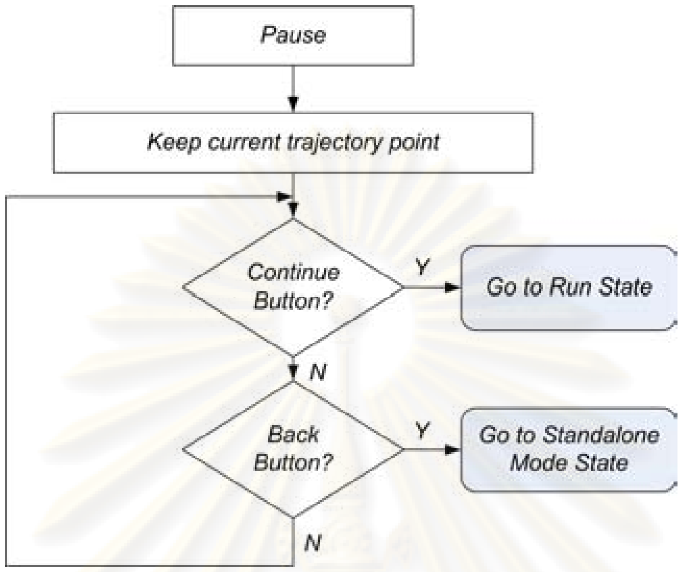


Figure 3.38: Flowchart diagram of pause state

3.6.5 Stop State

Stop state is similar with *pause* state. If *stop* button is pressed, controller system holds the present position. While the present position is hold, however, its trajectory sequence is resetted to the initial position of the trajectory file. Then it waits until the next command is given. If *restart* button is pressed, it goes to *run* state again. If back button is pressed, it backs to *standalone mode* state.

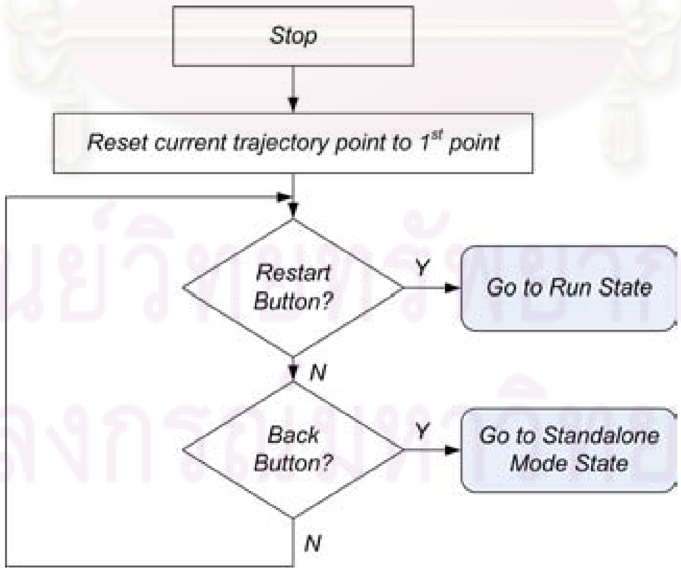
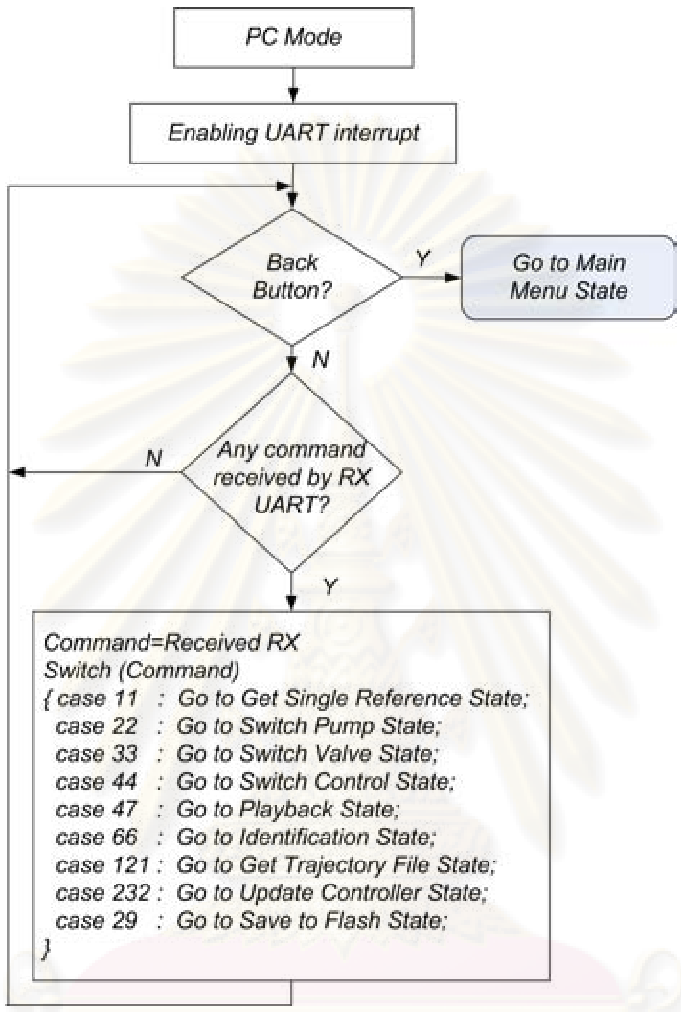

Stop
Reset current trajectory point to 1st point
Restart Button?
Y
Go to Run State
N
Back Button?
Y
Go to Standalone Mode State
N

Figure 3.39: Flowchart diagram of stop state

3.6.6 PC Mode State



```
Command=Received RX
Switch (Command)
{ case 11   :  Go to Get Single Reference State;
  case 22   :  Go to Switch Pump State;
  case 33   :  Go to Switch Valve State;
  case 44   :  Go to Switch Control State;
  case 47   :  Go to Playback State;
  case 66   :  Go to Identification State;
  case 121 :  Go to Get Trajectory File State;
  case 232 :  Go to Update Controller State;
  case 29   :  Go to Save to Flash State;
}
```

Figure 3.40: Flowchart diagram of PC mode state

When *PC Mode* is chosen, controller system will activate the UART communication channel by enabling UART interrupt. The controller system can communicate with PC, either receiving command from PC or transmitting data to PC, using UART protocol. During *PC mode*, all of robotic operations are controlled through PC. When a certain command is received, its state goes to corresponding state as shown in Figure 3.34. Every command is known by a unique identity number called ID command. If there is no command received and *back* button is pressed then it brings the state back to the *main menu*.

3.6.7 Switch Pump State

This state is done when a command with ID=22 is received. After the a byte command is received then it waits to receive the next byte for determining the next instruction. If the next received byte is '1' then controller system switch the pump ON. Conversely, the controller switch the pump OFF when it receives '0' as the next received byte.

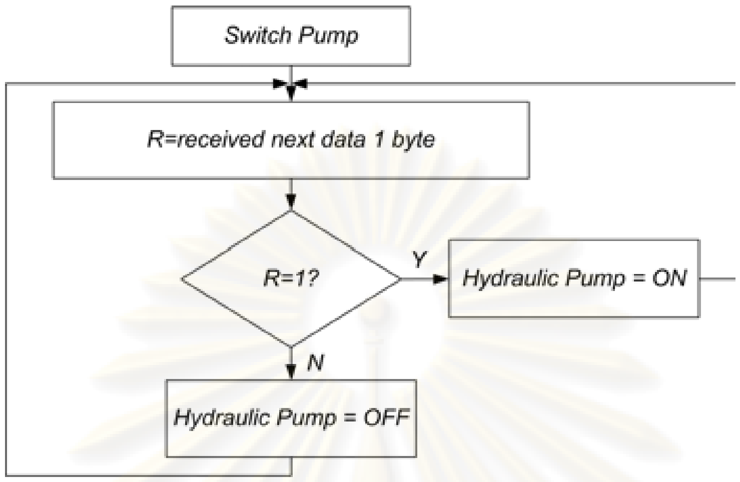


Figure 3.41: Flowchart diagram of switch pump state

3.6.8 Switch Valve State

This is a similar state with *switch pump*. The state comes to *switch valve* state when a command with ID=33 is received. The next received byte determines the next instruction, either switch on the oil valve or switch off the oil valve. The next received byte = '1' means switch the oil valve ON, otherwise switch the oil valve OFF.

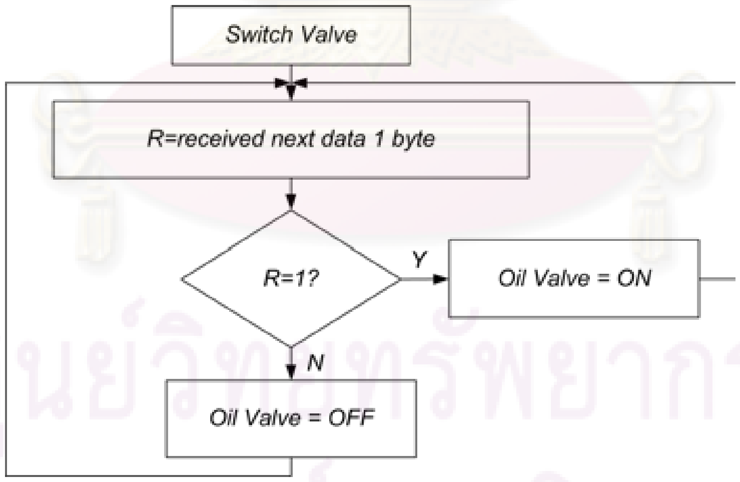


Figure 3.42: Flowchart diagram of switch valve state

3.6.9 Switch Control State

The same way as *switch pump* and *switch valve* is also used in *switch control* state. The state comes to *switch control* when a command with ID=44 is received. The next received byte = '1' means the controller status is ON, otherwise is OFF.

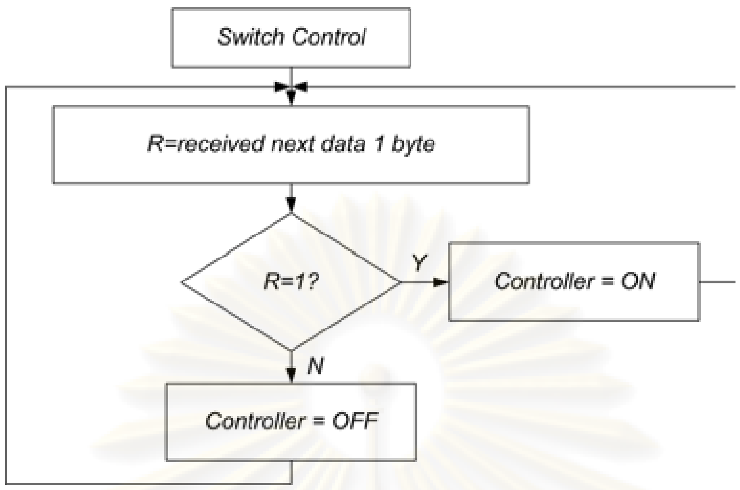


Figure 3.43: Flowchart diagram of switch control state

3.6.10 Playback State

If a command with ID=47 is received then it goes to playback state. Playback means the controller system drives the robot moving with certain pattern automatically. The second received byte determines the playback status. If the second received byte is '1' then playback status is ON, otherwise is OFF. When the playback status is ON, the controller system drives the robot following a sequence of data represents its position from the trajectory file.

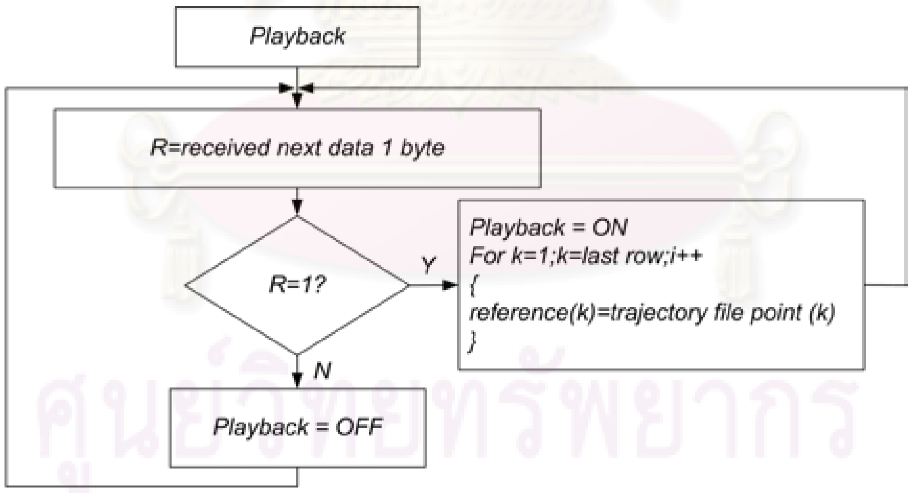

Playback
R=received next data 1 byte
R=1?
Y
Playback = ON
For k=1;k=last row;i++
{
reference(k)=trajectory file point (k)
}
N
Playback = OFF

Figure 3.44: Flowchart diagram of playback state

3.6.11 Update Controller State

After a corresponding ID command of update controller is received, then series of data are received following the ID command. These series of data represent the coefficient of controller parameters $R(z)$, $S(z)$ and $T(z)$ as shown in (3.71), (3.72) and (3.83). These data is begun by 2 bytes represents the coefficient of R_0, followed by 2 bytes represents the coefficient of R_1 and so on up to T_2 as shown in flowchart diagram 3.45.

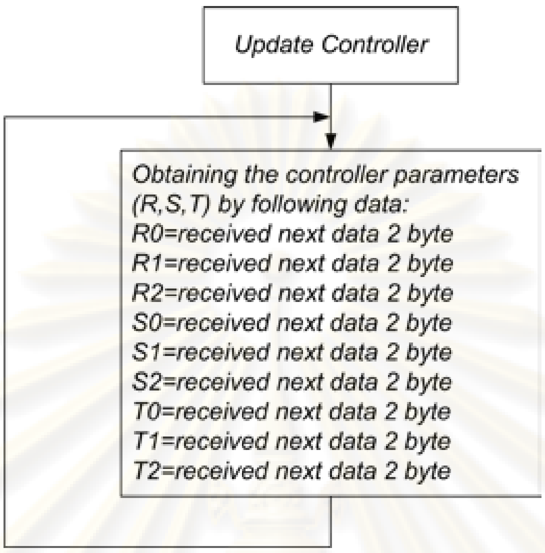

Update Controller
Obtaining the controller parameters
(R,S,T) by following data:
R0=received next data 2 byte
R1=received next data 2 byte
R2=received next data 2 byte
S0=received next data 2 byte
S1=received next data 2 byte
S2=received next data 2 byte
T0=received next data 2 byte
T1=received next data 2 byte
T2=received next data 2 byte

Figure 3.45: Flowchart diagram of update controller state

3.6.12 Get Single Reference State

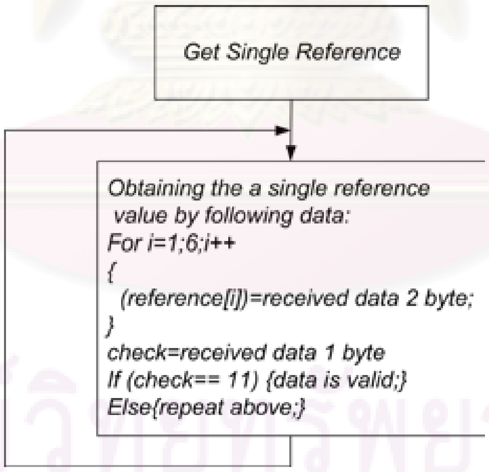

Get Single Reference
Obtaining the a single reference
value by following data:
For i=1;6;i++
{
(reference[i])=received data 2 byte;
}
check=received data 1 byte
If (check== 11) {data is valid;}
Else{repeat above;}

Figure 3.46: Flowchart diagram of get single reference state

The *get single reference* state is begun by receiving a byte of corresponding ID command then followed by 12 bytes data and ended by a byte of closing ID. These 12 bytes data divides into 6 parts which every part (2 bytes data) represents the reference value of angle/position in every axis of robotic arm. A closing ID is required to guarantee that these sequence of data are received properly. When the closing ID does not received correctly then the received data will never used to update the reference value.

3.6.13 Save to Flash State

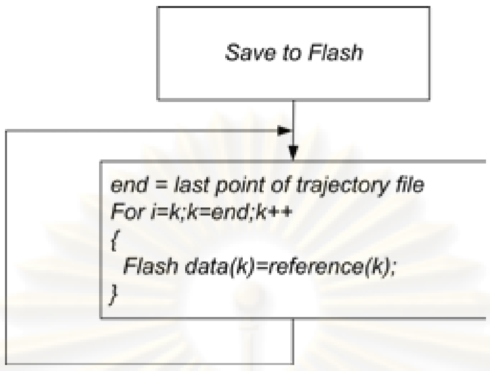

Save to Flash
end = last point of trajectory file
For i=k;k=end;k++
{
Flash data(k)=reference(k);
}

Figure 3.47: Flowchart diagram of save to flash

Save to flash means a present trajectory file used by the controller system will be stored in the Flash ROM. Storing the trajectory file into Flash ROM enables the *standalone mode* menu because the controller system can load this trajectory file anytime without connected to a PC. As shown in Figure 3.47, when a command ID of *save to flash* is received then it does a looping to copy the trajectory data occupies the volatile memory at certain address to the Flash ROM of the controller system.

3.6.14 Identification State

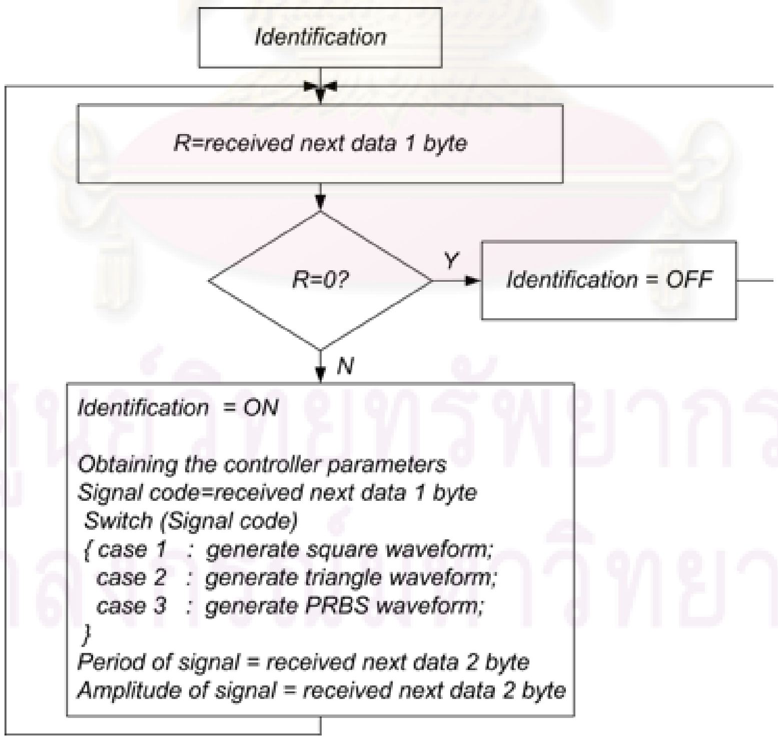
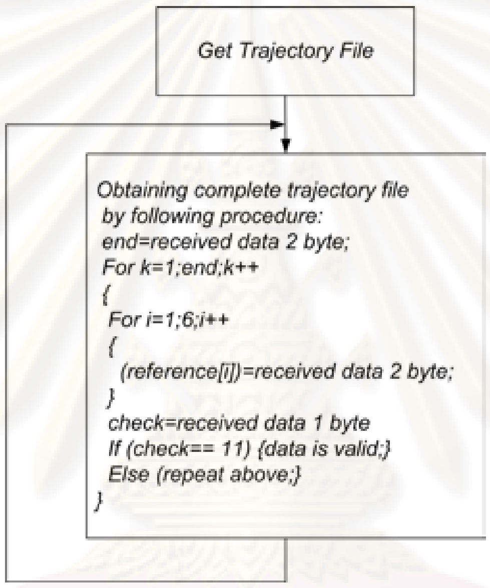

Identification
R=received next data 1 byte
R=0?
Y
Identification = OFF
N
Identification = ON
Obtaining the controller parameters
Signal code=received next data 1 byte
Switch (Signal code)
{ case 1 : generate square waveform;
case 2 : generate triangle waveform;
case 3 : generate PRBS waveform;
}
Period of signal = received next data 2 byte
Amplitude of signal = received next data 2 byte

Figure 3.48: Flowchart diagram of identification state

In *identification* state, two bytes of data are received firstly. The first byte is a corresponding ID command of identification and the second command determines the *identification* status. If the second byte is '0' then *identification* status is OFF, otherwise is ON. When the *identification* status is ON, 5 bytes of data are received following the previous command. These data consist of 3 parts: 1 byte to define the desired signal waveform, 2 bytes represents the desired period of signal and last 2 bytes represents the desired amplitude of signal that is used as input signal in system identification process.

3.6.15 Get Trajectory File State



```
Obtaining complete trajectory file
 by following procedure:
 end=received data 2 byte;
 For k=1;end;k++
 {
  For i=1;6;i++
  {
   (reference[i])=received data 2 byte;
  }
  check=received data 1 byte
  If (check== 11) {data is valid;}
  Else (repeat above;}
}
```

Figure 3.49: Flowchart diagram of get trajectory file state

Get trajectory file state consists of multiple routines of *get single reference* and some additional routines as shown in flowchart diagram 3.49. Firstly, a byte ID command is received then followed by 2 bytes data represents the number of data rows that will be received later. Every row of received data composed of 12 bytes data represents the reference value of angle/position and 1 byte of closing ID as used in *get single reference* state.

3.6.16 Interrupt Routine

There are three available interrupts provided by the software: *emergency interrupt*, *timer interrupt* and *UART interrupt*.

- *Emergency interrupt* handles the emergency condition by pressing the *emergency* button. When the *emergency* button is pressed, the controller system forces to switch the servo valve and the oil valve OFF, re-initializes their peripherals and back to the *main menu* state. Figure 3.50 shows the flowchart diagram of emergency interrupt.

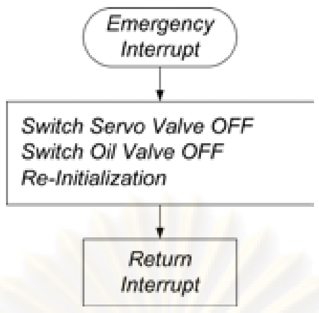


Figure 3.50: Flowchart diagram of interrupt timer routine

- *Timer interrupt* generates a certain interval time (in this case 10 ms) that becomes the controller sampling rate. Every single sampling rate, a timer interrupt routine will be performed. This routine covers updating ADC value, getting current reference value, computing PID algorithm and generating signal for system identification. When the *controller status* is ON, the system generates control signal using PID algorithm and send it to DAC module for driving the robotic arm. The same way is treated to the *identification status*. When the *identification status* is ON, the system generates certain waveform based on its command and send the measured data to PC. In order to avoid an interleaving it must be guaranteed that this routine will be accomplished before the coming timer interrupt is occurred. Figure 3.51 shows the flowchart diagram of timer interrupt routine.

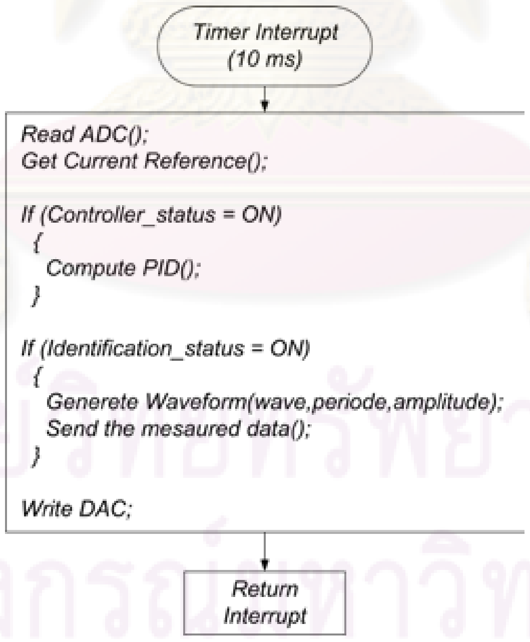

Timer Interrupt (10 ms)
Read ADC();
Get Current Reference();
If (Controller_status = ON)
{
Compute PID();
}
If (Identification_status = ON)
{
Generete Waveform(wave,periode,amplitude);
Send the mesaured data();
}
Write DAC;
Return Interrupt

Figure 3.51: Flowchart diagram of timer interrupt routine

- *UART interrupt* handles a command that is sent from PC. This interrupt will be enabled only when *PC mode* is selected. There are 9 commands known by the controller system: switch control, switch pump, switch valve, get single reference, get trajectory file, playback, update controller, save to flash and identification. Each command has unique command code to differentiate each other. For example command for getting single reference has command code = 11, switch pump has command code = 22 and so on. Each command has a specific routine that will be executed immediately after the command is received.

CHAPTER IV

GUI USER APPLICATION SOFTWARE

This chapter concern with GUI user application software as an interface between the operator (human) and the controller system. This software is developed by using Delphi Integrated Development Equipment program. Basically, it has five main menus: *Controller System*, *Identification System*, *Trajectory Generator*, *Teaching Mode* and *Simulation*. The state machine diagram of the *main* menu and its screenshot display are shown in Figure 4.1 and Figure 4.2 respectively.

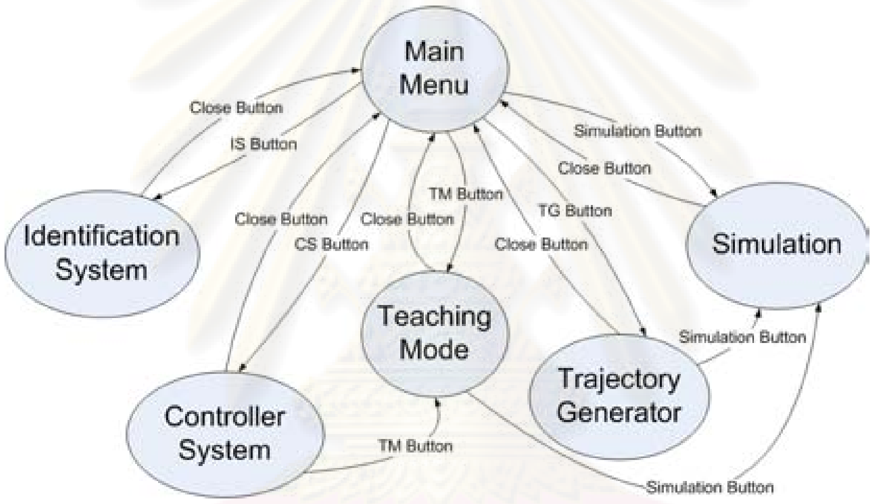


Figure 4.1: The state machine diagram of main menu

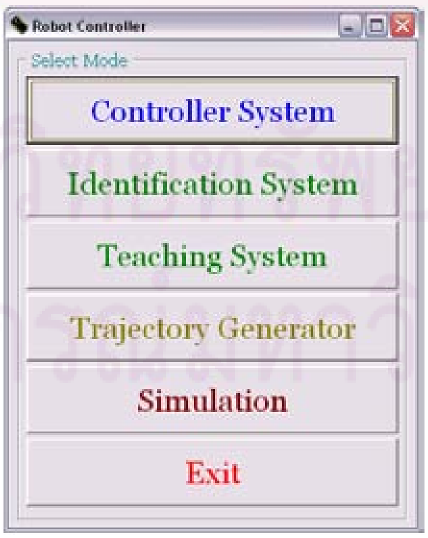


Figure 4.2: Main menu

Clicking one of five available menus above brings the state to the corresponding menu state and shows a new window panel. Then, closing the menu window panel will back its state to the *main menu* state. Particularly for the windows panel *Controller System*, *Teaching Mode* and *Trajectory Generator* respectively, they have a command to bring its state to *Teaching Mode* and *Simulation* state directly. Subsection 4.1 to 4.5 describe the detail of each those menu. Complete flowchart diagrams for each state in this state machine diagram are described in appendix.

4.1 Controller System

Controller system controls entire behavior of the controller system: how to operate the robot both manually and automatically, how to setup the controller parameter and so on. This menu has 3 tab panels: *Controller*, *Setup Controller* and *Data Record*. The state machine diagram of this menu is shown in Figure 4.3 whereas their screenshot displays are shown in Figure 4.4, 4.5 and 4.6.

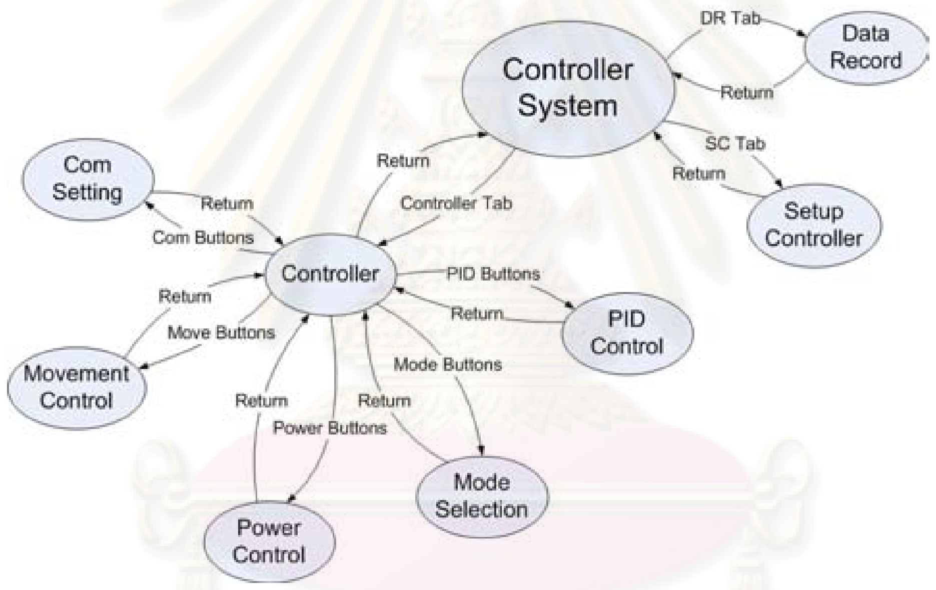


Figure 4.3: The state machine diagram of controller system menu

4.1.1 Controller

- *Communication Setting* sets the serial UART communication parameters such as port, baudrate and open-close communication status.
- *Power Control* controls the hydraulic pump and oil valve. The oil valve can be turned on only when the hydraulic pump has been turned on already. When the hydraulic pump is turned off, the oil valve will be turned off as well automatically.
- *PID Control* controls the PID controller status. When PID controller status is ON, the robotic arm moves automatically following the given position either single position or sequence of positions as in trajectory file that has been sent to the controller system previously.

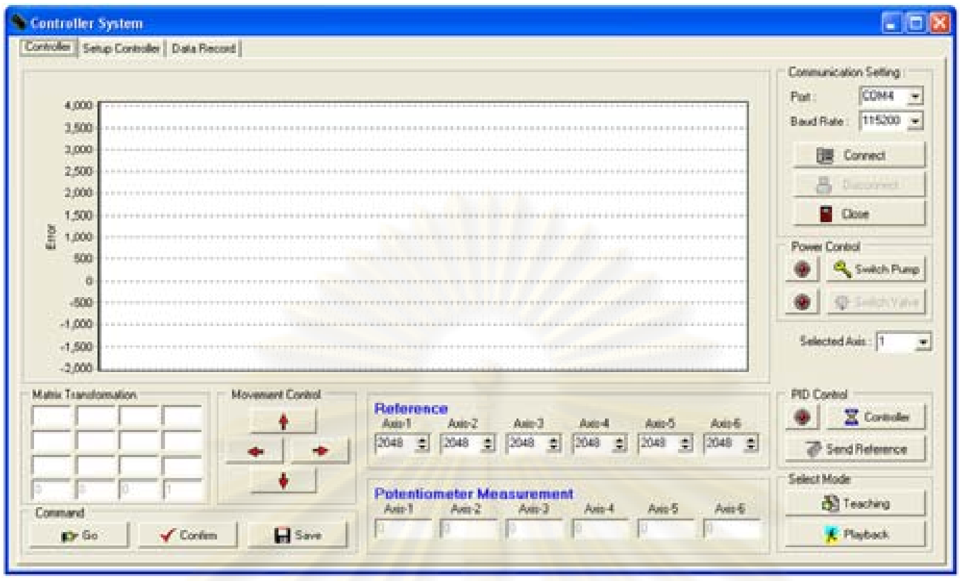


Figure 4.4: Controller tab

- *Select Mode* provides two operation modes: *Teaching Mode* and *Playback Mode*. *Teaching mode* means the robot is operated under an human operator control. The trajectory yielded during teaching mode can be stored as a trajectory file and later can be loaded and played back. *Playback mode* means the robot operates automatically doing the trajectory file that has been stored in the controller system previously.

- *Reference* are the references value of the PID controller system. When the playback status is ON, these reference values will change following the trajectory file. Conversely, these one can be set manually and can be sent to the controller system by pressing *Send Reference* button.

- *Potentiometer Measurement* is the measured values that represent the angle in every joint of robotic arm.

- *Movement Control* control the relative movement of the robotic arm in Cartesian coordinate.

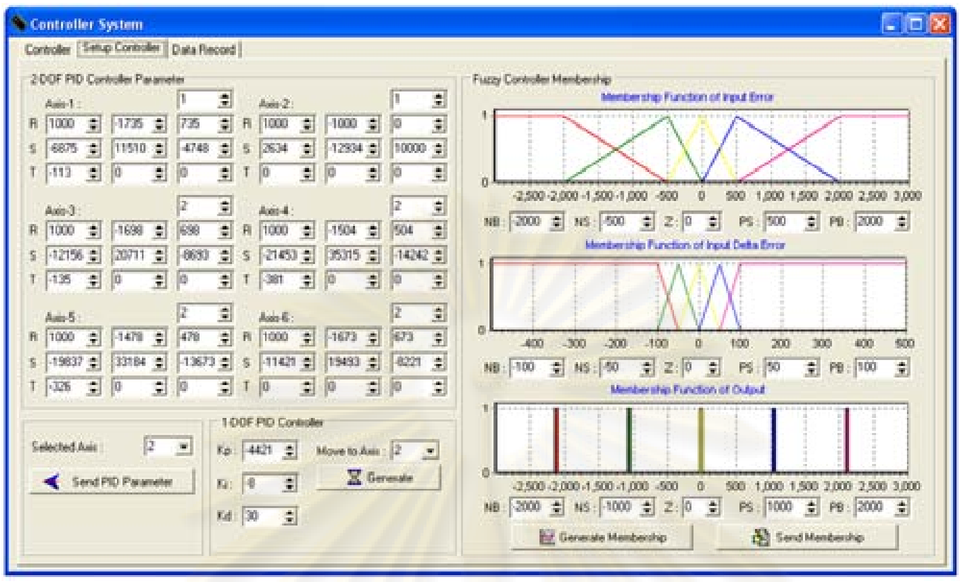
- *Matrix Transformation* represents the position and orientation of the end-point of robotic arm.

4.1.2 Setup Controller

- *2-DOF PID Controller Parameters* adjusts the controller parameters. R, S and T are the polinomials of controller with structure as shown in Figure 2.4 (b).

- *1-DOF PID Controller Parameters* adjusts the controller parameters with structure as shown in Figure 2.4 (a). *Send PID Parameters* button is used to send the controller parameter at joint as in selected axis.

- *Fuzzy Controller Membership* adjusts the membership function of parameter input error, delta error and output if fuzzy algorithm is chosen. *Generate Membership* button and *Send Member-*


Controller System
Controller
Setup Controller
Data Record
Fuzzy Controller Membership
Membership Function of Input Error
Membership Function of Input Delta Error
Membership Function of Output
Send PID Parameter
Generate
Generate Membership
Send Membership

2-DOF PID Controller Parameter

	Axis-1 :		1		Axis-2 :		1
R	1000	-1735	735	R	1000	-1000	0
S	-6875	11510	-4748	S	2634	-12934	10000
T	-113	0	0	T	0	0	0
	Axis-3 :		2		Axis-4 :		2
R	1000	-1698	698	R	1000	-1504	504
S	-12156	20711	-8693	S	-21453	35315	-14242
T	-135	0	0	T	-381	0	0
	Axis-5 :		2		Axis-6 :		2
R	1000	-1478	478	R	1000	-1673	673
S	-19837	33184	-13673	S	-11421	19493	-8221
T	-326	0	0	T	0	0	0

Selected Axis : 2

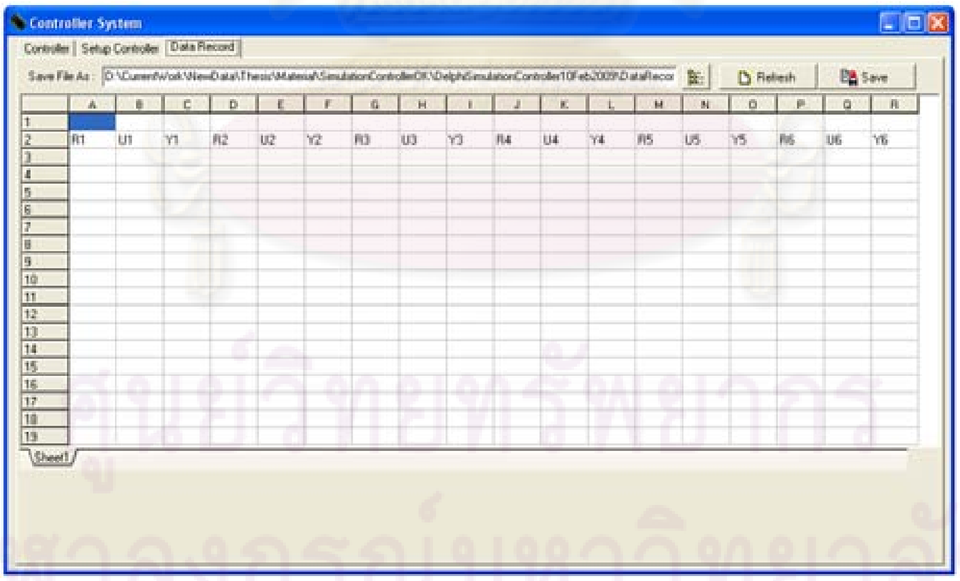
1-DOF PID Controller
Kp : -4421
Ki : -8
Kd : 30
Move to Axis : 2

	NB	NS	Z	PS	PB
Membership Function of Input Error	-2000	-500	0	500	2000
Membership Function of Input Delta Error	-100	-50	0	50	100
Membership Function of Output	-2000	-1000	0	1000	2000

Figure 4.5: Setup controller tab

ship button are used to generate the membership function and send that membership function to the controller system respectively.

4.1.3 Data Record



Save File As : D:\CurrentWork\NewData\Thesis\Material\SimulationControllerOK\DelphiSimulationController10Feb2009\DataRecor

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1																		
2	R1	U1	Y1	R2	U2	Y2	R3	U3	Y3	R4	U4	Y4	R5	U5	Y5	R6	U6	Y6
3																		
4																		
5																		
6																		
7																		
8																		
9																		
10																		
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		

Figure 4.6: Data record menu

Data record shows the recorded data during the robot operation. These data are stored in a Microsoft Excel file format. Pressing *Refresh* button will clear the recorded data and *save* button will save the recorded data into PC storage.

4.2 Identification System

Identification system is provided to support the data collection during experiment. The collected data is composed of data input and data output of the servo valve in every joint of robotic arm. In advance, these data are processed to yield the estimate model of the servo valve of robot. The state machine diagram of the *identification system* menu is shown in Figure 4.7.

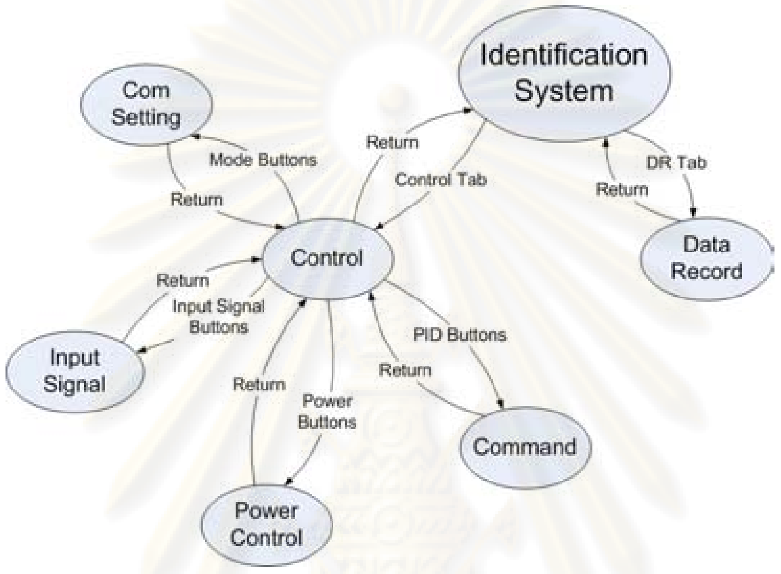


Figure 4.7: The state machine diagram of identification system menu

4.2.1 Control

This menu controls the data collection process: when the data is began to be collected, when the data is stopped to be collected, what kind of input data is applied and so on. Figure 4.8 shows the screenshot of *control* submenu.

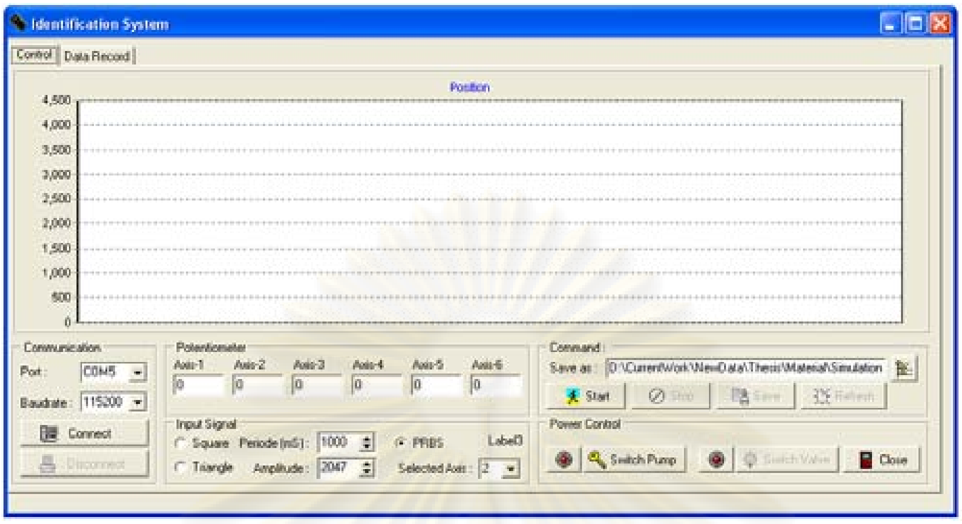
- *Command* is a panel to control the identification status. *Start* button and *Stop* button are used to begin and end the data collection process.

- *Power Control* controls the hydraulic pump and oil valve. The oil valve can be turned on only when the hydraulic pump has been turned on. However the oil valve will be turned off automatically when the hydraulic pump is turned off.

- *Communication* sets the serial UART communication parameters such as port, baudrate and open-close communication status.

- *Potentiometer* shows the measured values represent the angle in every joint of robotic arm.

- *Input Signal* gives choices; what kind of input signal will be employed in the identification system. There are three types of input signal: Square, Triangle and Pseudo Random Binary Se-



Port : COM5
Baudrate : 115200
Axis-1 0
Axis-2 0
Axis-3 0
Axis-4 0
Axis-5 0
Axis-6 0
Save as : D:\CurrentWork\NewData\Thesis\Material\Simulation
Square Periode (mS) : 1000
Triangle Amplitude : 2047
[✓ PRBS]
Selected Axis : 2

Figure 4.8: Control in Identification System

quence (PRBS). For Square and Triangle, its amplitude and period of the signal can be adjusted manually.

4.2.2 Data Record

Data record shows the collected data during the system identification process. These data will be stored in PC storage as a Microsoft Excel file format by pressing *save* button.

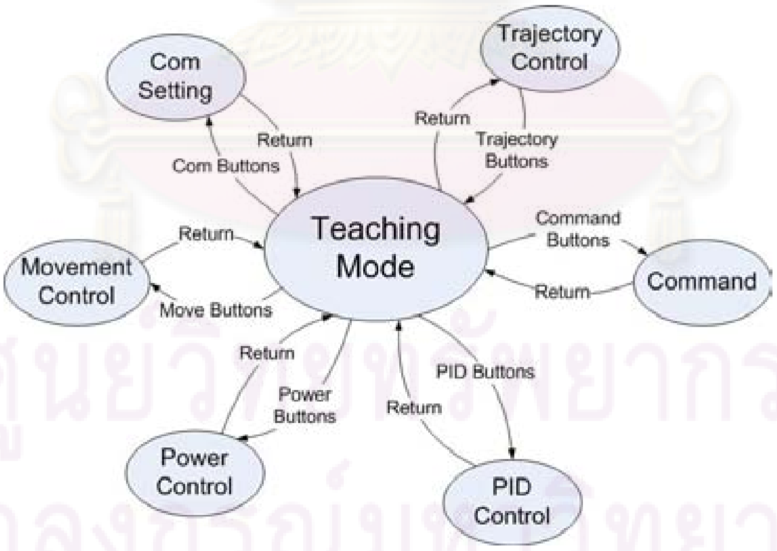
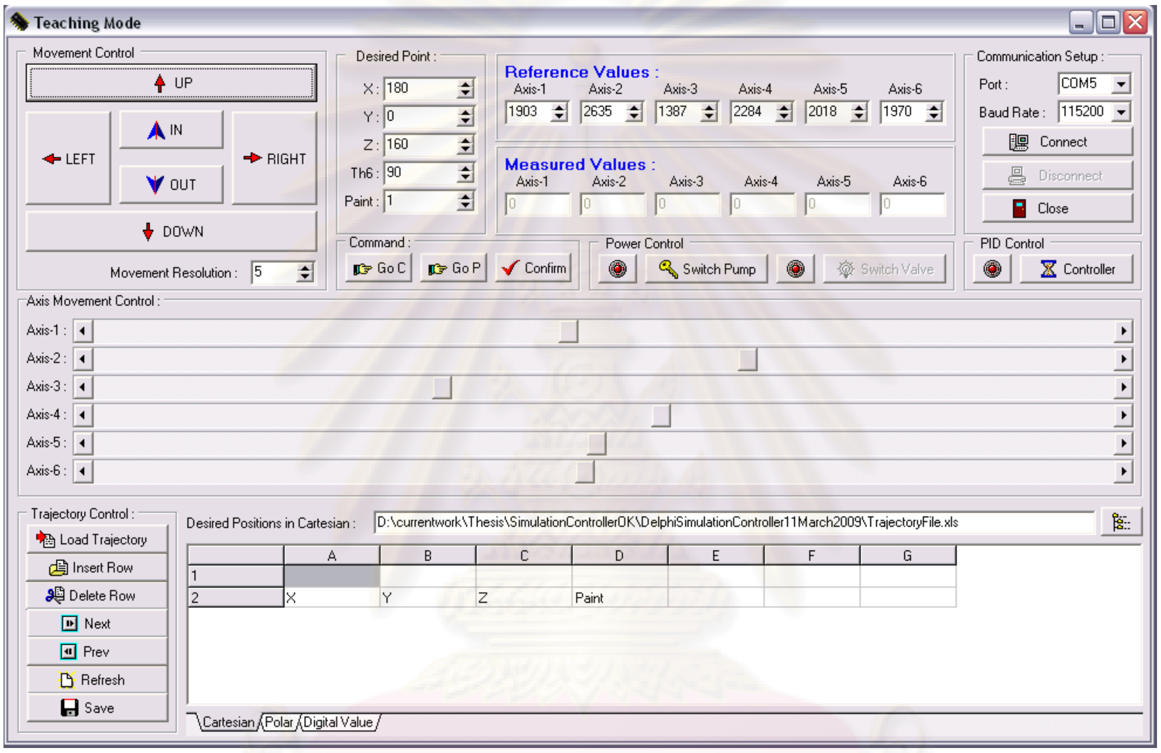


Figure 4.9: The state machine diagram of teaching mode menu

4.3 Teaching Mode

This menu provides a panel to control the robot in *teaching* mode. In this mode, the robot is fully controlled by an human operator. The robot position can be specified either in Cartesian or Polar coordinate. During teaching process, the positions of the robot can be stored as a trajectory file. This trajectory file then can be loaded and executed in *playback* mode. The state machine diagram and the sreenshot display of *teaching mode* menu are shown in Figure 4.9 and Figure 4.10 respectively.



Movement Resolution : 5

X : 180
Y : 0
Z : 160
Th6 : 90
Paint : 1

	Axis-1	Axis-2	Axis-3	Axis-4	Axis-5	Axis-6
Reference Values	1903	2635	1387	2284	2018	1970
Measured Values	0	0	0	0	0	0

Port : COM5
Baud Rate : 115200

Desired Positions in Cartesian : D:\currentwork\Thesis\SimulationControllerOK\DelphiSimulationController11March2009\TrajectoryFile.xls

	A	B	C	D	E	F	G
1							
2	X	Y	Z	Paint			

Figure 4.10: Teaching mode menu

- *Movement Control* control the relative movement of the robotic arm in Cartesian coordinate. The relative movement distance and the direction are respectively determined by the *movement resolution* and the direction button.
- *Communication Setup* sets the serial UART communication parameters such as port, baudrate and open-close communication status.
- *Command* is a panel to control the robot moving to the specified position. Button *Go C* and *Go P* respectively forces the robot moves to a specified point in Cartesian and in Polar coordinate. Pressing *confirm* button will save current position to the trajectory file.
- *Power Control* controls the hydraulic pump and oil valve. The oil valve can be turned on only when the hydraulic pump has been turned on already. When the hydraulic pump is turned off, the oil valve will be turned off as well automatically.

- *PID Control* controls the PID controller status. When PID controller status is ON, the robotic arm moves automatically following the given position either single position or sequence of positions as in trajectory file that has been sent to the controller system previously.

- *Trajectory Control* modifies current trajectory file during *teaching* mode such as loading trajectory file, inserting and deleting a data.

4.4 Trajectory Generator

This panel provides an interface to do everything related to the trajectory file. Creating a new trajectory file, loading trajectory file, editing trajectory file, sending trajectory file and saving trajectory file are some available commands in this menu. Figure 4.11 shows the state machine diagram of *trajectory generator* menu whereas Figure 4.12 shows its screenshot display.

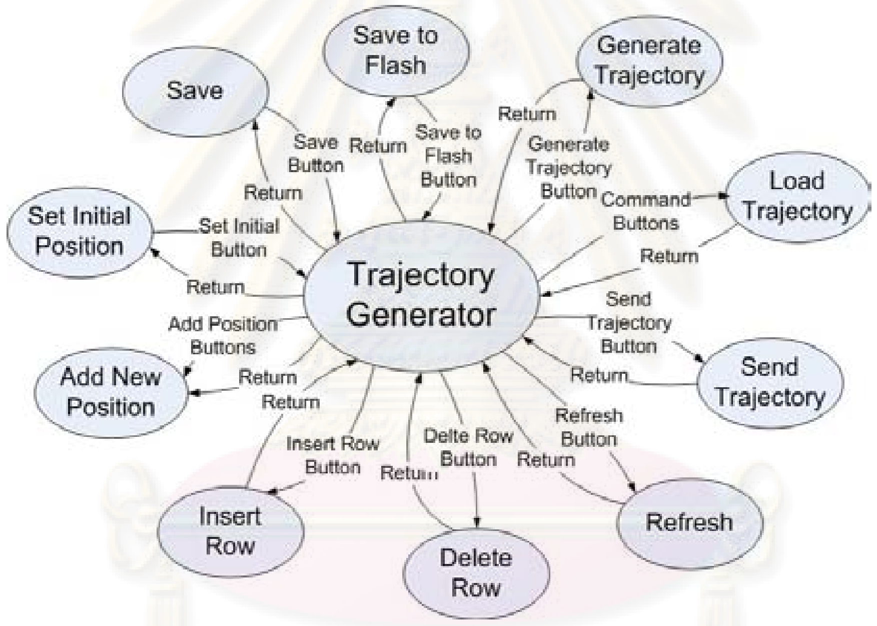


Figure 4.11: The state machine of trajectory generator menu

- *Set Initial Position* specifies the initial position of the robotic arm end-point in Cartesian coordinate.

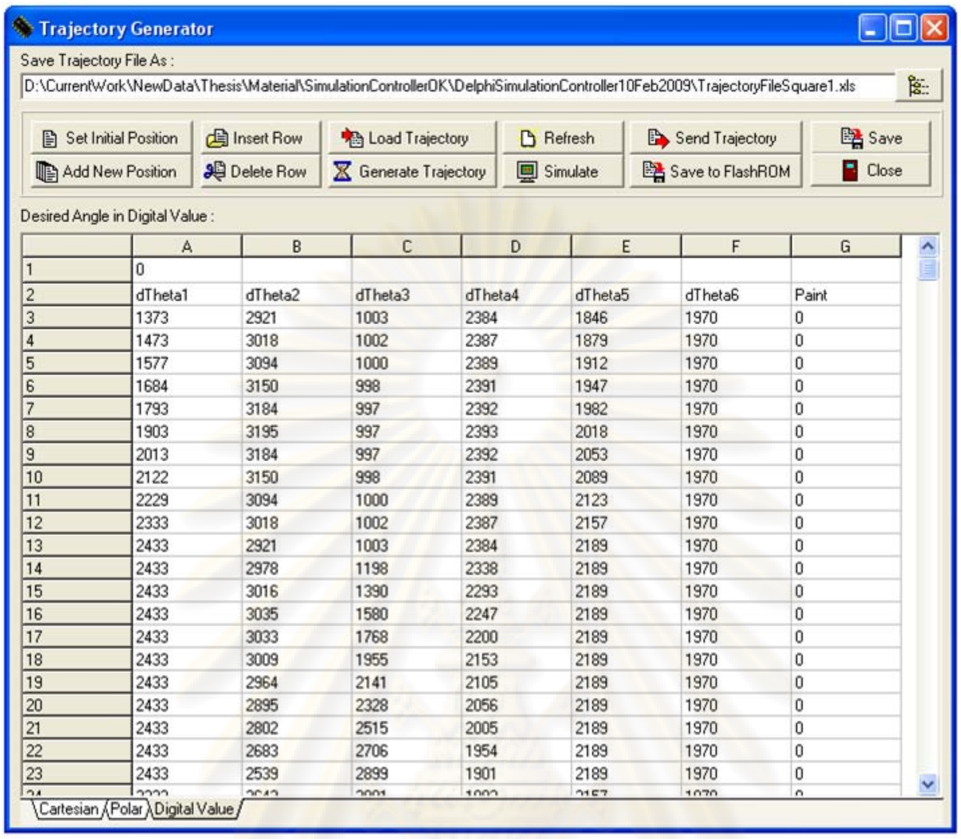
- *Add New Position* adds a new position in trajectory file.

- *Insert Row* inserts a new data in selected row of trajectory file.

- *Delete Row* deletes a data in selected row of trajectory file.

- *Load Trajectory* opens the trajectory file that has been stored in the PC storage previously.

- *Generate Trajectory* converts the sequences of position in Cartesian coordinate into polar coordinate (angle in every joint) and converts the polar coordinate into digital value.



Save Trajectory File As :
D:\CurrentWork\NewData\Thesis\Material\SimulationControllerOK\DelphiSimulationController10Feb2009\TrajectoryFileSquare1.xls

Desired Angle in Digital Value :

	A	B	C	D	E	F	G
1	0						
2	dTheta1	dTheta2	dTheta3	dTheta4	dTheta5	dTheta6	Paint
3	1373	2921	1003	2384	1846	1970	0
4	1473	3018	1002	2387	1879	1970	0
5	1577	3094	1000	2389	1912	1970	0
6	1684	3150	998	2391	1947	1970	0
7	1793	3184	997	2392	1982	1970	0
8	1903	3195	997	2393	2018	1970	0
9	2013	3184	997	2392	2053	1970	0
10	2122	3150	998	2391	2089	1970	0
11	2229	3094	1000	2389	2123	1970	0
12	2333	3018	1002	2387	2157	1970	0
13	2433	2921	1003	2384	2189	1970	0
14	2433	2978	1198	2338	2189	1970	0
15	2433	3016	1390	2293	2189	1970	0
16	2433	3035	1580	2247	2189	1970	0
17	2433	3033	1768	2200	2189	1970	0
18	2433	3009	1955	2153	2189	1970	0
19	2433	2964	2141	2105	2189	1970	0
20	2433	2895	2328	2056	2189	1970	0
21	2433	2802	2515	2005	2189	1970	0
22	2433	2683	2706	1954	2189	1970	0
23	2433	2539	2899	1901	2189	1970	0

Figure 4.12: Trajectory generator

- *Refresh* clears the current trajectory data.
- *Simulate* calls the simulation panel.
- *Send Trajectory* sends the trajectory file to the controller system. This trajectory file will be stored at the RAM of controller system.
- *Save to Flash ROM* saves the trajectory file that is stored in RAM into Flash ROM of the controller system. When a trajectory file has been sent to the RAM and its trajectory has been run well then it can be stored into flash ROM to keep it reside in the controller system permanently.
- *Save* stores current trajectory file to the PC storage as a Microsoft Excel file format.

4.5 Simulation

This menu acts as a tool to simulate the trajectory file before loaded in the real controller system. It is highly recommended to do a simulation by this tool to guarantee that the trajectory file will drive the robotic arm following the desired pattern properly. Figure 4.13 shows the state machine diagram of *simulation* menu whereas Figure 4.14 shows its screenshot display.

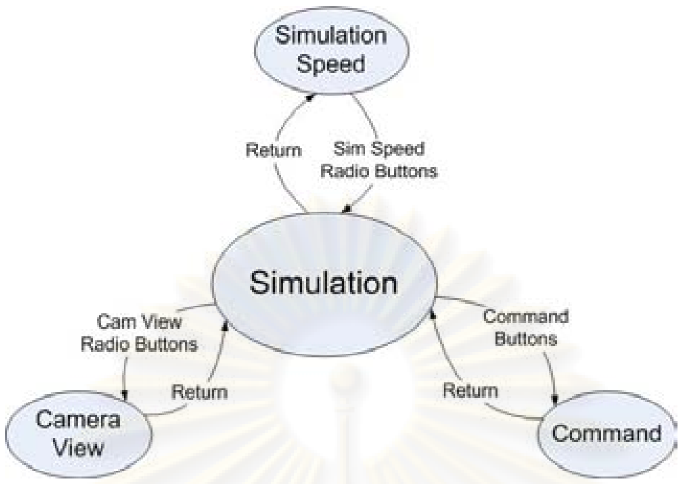
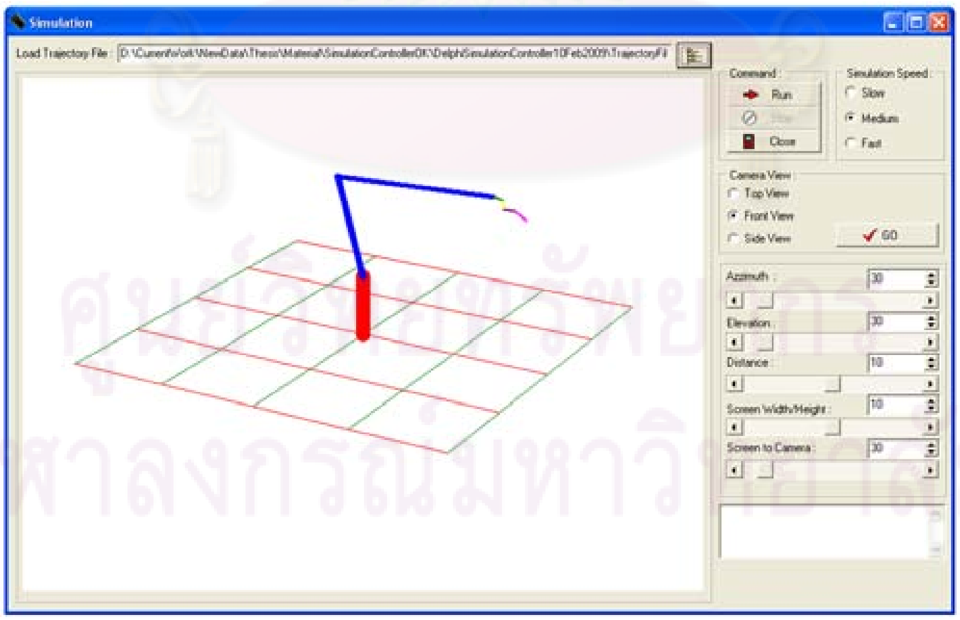


Figure 4.13: The state machine diagram of simulation menu

- *Command* is a panel to control the simulation status. *Run* button and *Stop* button are used to begin and stop the simulation respectively. *Run* button only available when the simulated trajectory file has been loaded.
- *Simulation Speed* is used to tune the speed of robotic arm motion in simulation.
- *Camera View* varies the way of view in many styles. Three basic views: top view, front view and side view are available. It also can be tuned manually by adjusting the value in each parameters such as azimuth, elevation and distance.



Load Trajectory File: D:\CurrentWork\NewData\Thesis\Material\SimulationControllerOK\Delphi\SimulationController10Feb2009\TrajectoryFi

Simulation Speed: Slow [✓ Medium] Fast

Camera View: Top View [✓ Front View] Side View

Azimuth: 30
Elevation: 30
Distance: 10
Screen Width/Height: 10
Screen to Camera: 30

Figure 4.14: Inverse kinematics simulation with IDE Delphi

CHAPTER V

TEST AND EXPERIMENT RESULTS

5.1 End-point Position Calibration

This experiment is purposed to calibrate the end-point position of the robotic arm by comparing a targeted position and the real one obtained from measurement. Because a specific tool to measure the real position is not available then the real position is measured by the method described below.

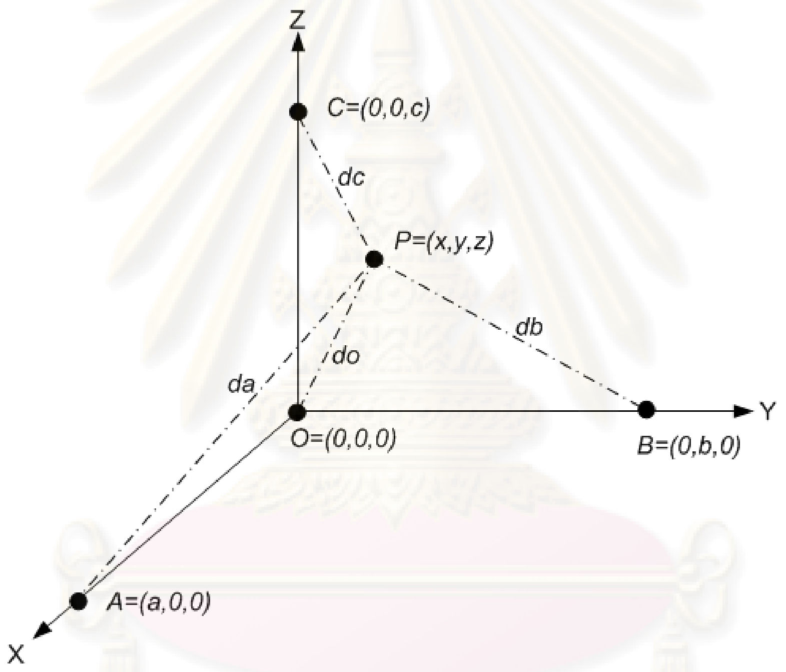


Figure 5.1: End-point measurement method

Suppose a base point O (0,0,0) and three auxiliary points with known coordinate are given: point A (a,0,0), point B (0,b,0) and point C (0,0,c) as shown in Figure 5.1. Any point P with coordinate (x,y,z) can be known by measuring the distance between point P and each point of O, A, B and C namely do, da, db and dc. There are four equations that represent each of those distance as in (5.1) to (5.4).

$$x^2 + y^2 + z^2 = do^2 \tag{5.1}$$

$$(x - a)^2 + y^2 + z^2 = da^2 \tag{5.2}$$

$$x^2 + (y - b)^2 + z^2 = db^2 \tag{5.3}$$

$$x^2 + y^2 + (z - c)^2 = dc^2 \tag{5.4}$$

By substituting equation (5.1) into equation (5.2), (5.3) and (5.4) it yields:

$$\Delta x = \frac{da^2 - do^2 - a^2}{-2a} \tag{5.5}$$

$$\Delta y = \frac{db^2 - do^2 - b^2}{-2b} \tag{5.6}$$

$$\Delta z = \frac{dc^2 - do^2 - c^2}{-2c} \tag{5.7}$$

Practically, we use O=[46 37 0]^T, A=[268 37 0]^T, B=[46 $-$ 37 0]^T, and C=[46 37 49]^T in this measurement. Therefore we have variables of a, b and c are 222, -74 and 49 respectively. Using equation (5.5), (5.6) and (5.7) we obtain the measured end-point position P by formula in (5.8). Table 5.1 shows the complete measurement results for this experiment.

$$P = O + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{5.8}$$

Table 5.1: End-point measurement results in cm unit

Target Point			Measurement				Measured Point			Error		
x_d	y_d	z_d	do	da	db	dc	x_m	y_m	z_m	x	y	z
180	0	170	220	220	185	193	182.11	0	169.14	-2.11	0	0.85
180	-60	170	238	218	206	213	182.39	-61.62	169.47	-2.39	1.62	0.52
180	60	170	218	238	183	190	182.72	61.62	167.71	-2.72	-1.62	2.28
190	0	170	226	226	192	188	192.43	0	169.52	-2.43	0	0.47
190	0	150	212	212	180	171	192.36	0	152.5	-2.36	0	-2.5
190	0	120	194	194	168	149	191.76	0	120.54	-1.76	0	-0.54
190	0	80	170	170	154	115	192.30	0	77.39	-2.30	0	2.60
160	0	170	208	207	171	202	162.54	-2.80	167.59	-2.54	2.80	2.40
140	0	170	197	197	157	214	141.26	0	165.77	-1.26	0	1.01
170	0	170	213	213	176	197	171.77	0	171.36	-1.77	0	-1.36

5.2 Analog Board Tests

These experiments test whether the analog board has worked properly or not. The experiment setup is shown in Figure 5.2.

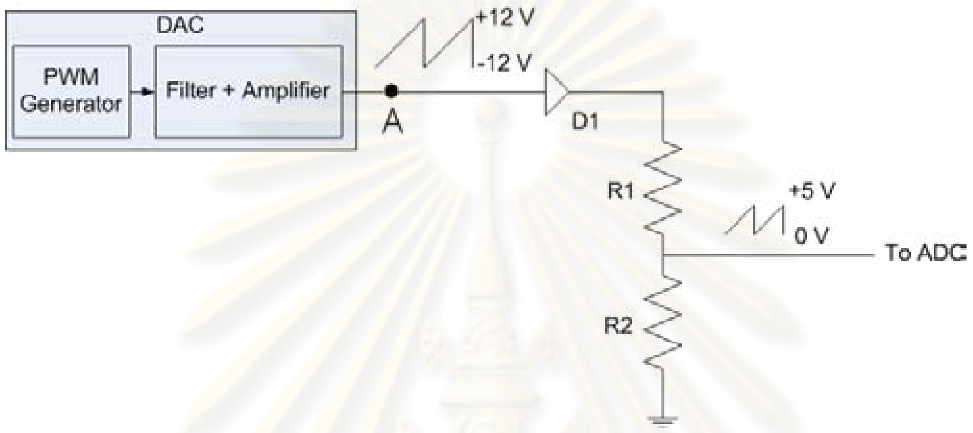


Figure 5.2: The circuit for matching the ADC and DAC level

The first test is done by generating sinusoidal and sawtooth waveform independently. Microblaze is programmed to produce digital values represent the sinusoidal and sawtooth waveform. Those signals then converted into analog signal by DAC part which is composed of PWM generator, filter and amplifier circuit. Analog signals at point A are observed by oscilloscope. The results are shown in Figure 5.3 and 5.4.

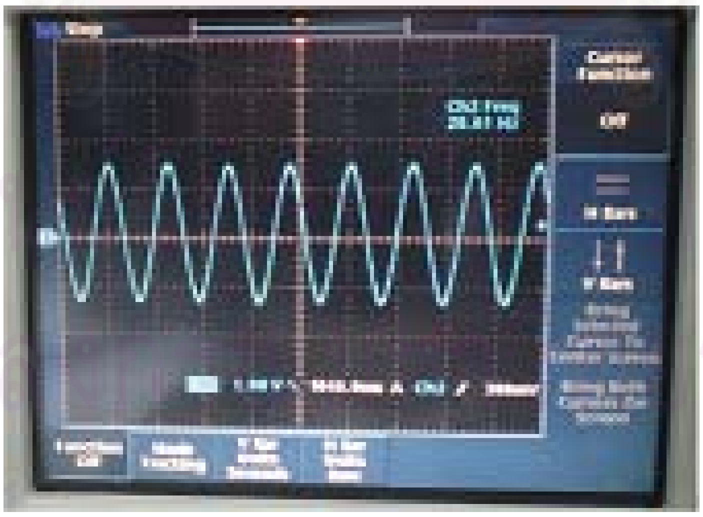


Figure 5.3: Generating sinusoidal waveform

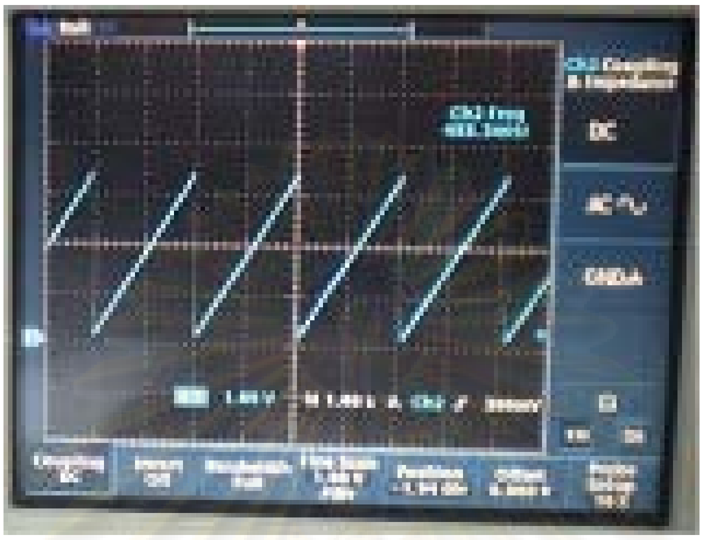
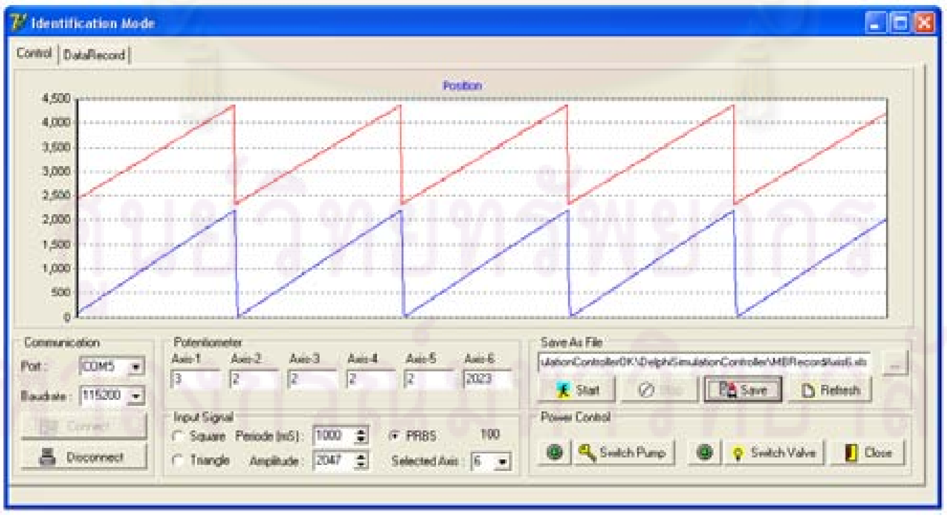


Figure 5.4: Generating sawtooth waveform

The second test is done by looping back the signal that has been generated by DAC part into ADC part. Both of DAC and ADC values are recorded by a PC and displayed in the screen. Since the DAC level (-12 V to +12V) is unequal to the ADC level (0V to +5 V), it needs an adjustment to make it works at proper operation point for each. Figure 5.2 is a simple voltage divider circuit that can be used for this purpose. By choosing R_1= 6.8 KOhm and R_2=5 KOhm it can adjust the DAC level (-12 V to +12V) to the lower level (-5 V to +5 V). The diode $D1$ is functioned to protect the negative level so that it only has 0 V to +5 V level which is match with the ADC level. Figure 5.5 and 5.6 show the test results. The red line is the DAC data whereas the blue line is the ADC data.



Communication — Port : COM5; Baudrate : 115200

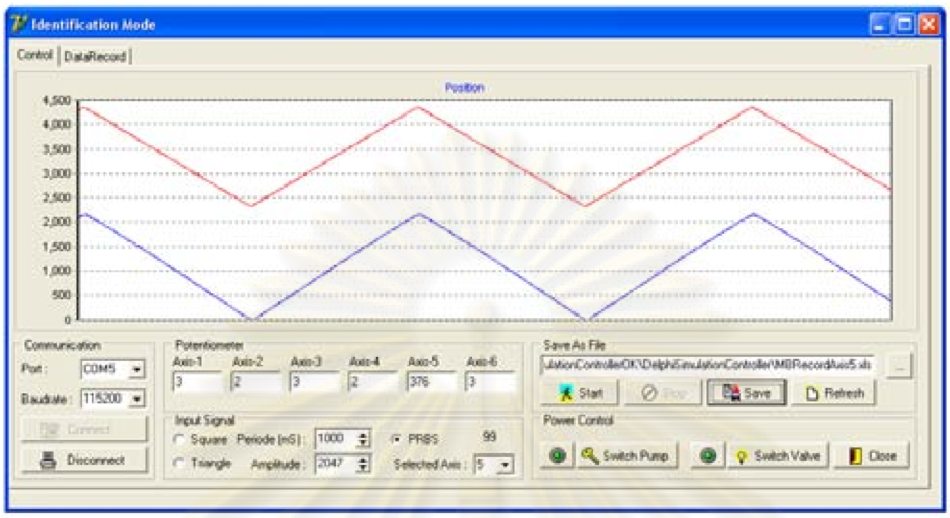
Potentiometer — Axis-1: 3; Axis-2: 2; Axis-3: 2; Axis-4: 2; Axis-5: 2; Axis-6: 2023

Input Signal — Square; Triangle; Periode (mS) : 1000; Amplitude : 2047; [✓ PRBS] 100; Selected Axis : 6

Save As File — ulationControllerOK\DelphiSimulationController\MBRecord\axis6.xls

Power Control

Figure 5.5: ADC and DAC looping back test with sawtooth waveform



Communication
Port : COM5
Baudrate : 115200

Potentiometer

Axis-1	Axis-2	Axis-3	Axis-4	Axis-5	Axis-6
3	2	3	2	376	3

Input Signal
Square Periode (mS) : 1000
Triangle Amplitude : 2047
PRBS 99
Selected Axis : 5

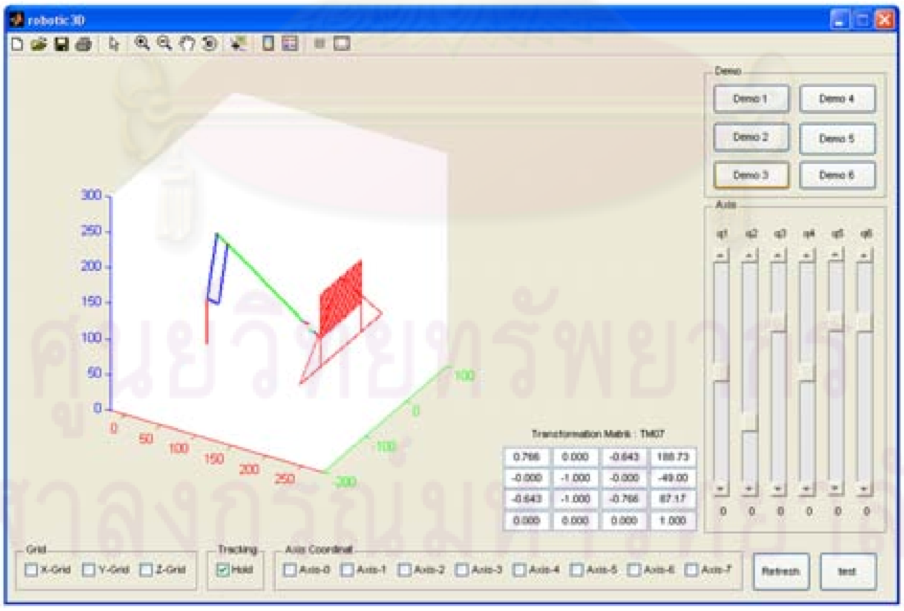
Save As File
...ulationControllerOK\DelphiSimulationController\MBRecordAxis5.xls

Power Control

Figure 5.6: ADC and DAC looping back test with triangle waveform

5.3 Inverse Kinematics Simulation

Figure 5.7 and Figure 4.14 show the simulation of inverse kinematics using Matlab and Delphi. Those simulation demonstrate how the robotic arm move following a given trajectory pattern in Cartesian coordinate.



Transformation Matrix : TM07

0.766	0.000	-0.643	188.73
-0.000	-1.000	-0.000	-49.00
-0.643	-1.000	-0.766	87.17
0.000	0.000	0.000	1.000

q1	q2	q3	q4	q5	q6
0	0	0	0	0	0

Figure 5.7: Inverse kinematics simulation with Matlab

5.4 Playback Tests

In order to observe the controller performance, we also test the robotic operation at playback mode. Firstly, a sequence of targeted positions in Cartesian coordinate is specified. In this experiment we specify a sequence of points forming a square pattern. By using trajectory generator menu, the corresponding trajectory file is then generated. After being simulated by simulation tool, finally the trajectory file is transferred to the controller system and the controller is operated based on the given trajectory file. Figure 5.8 shows the experiment result. The red-line square represents the ideal trajectory pattern whereas the blue-line square represents the real trajectory pattern.

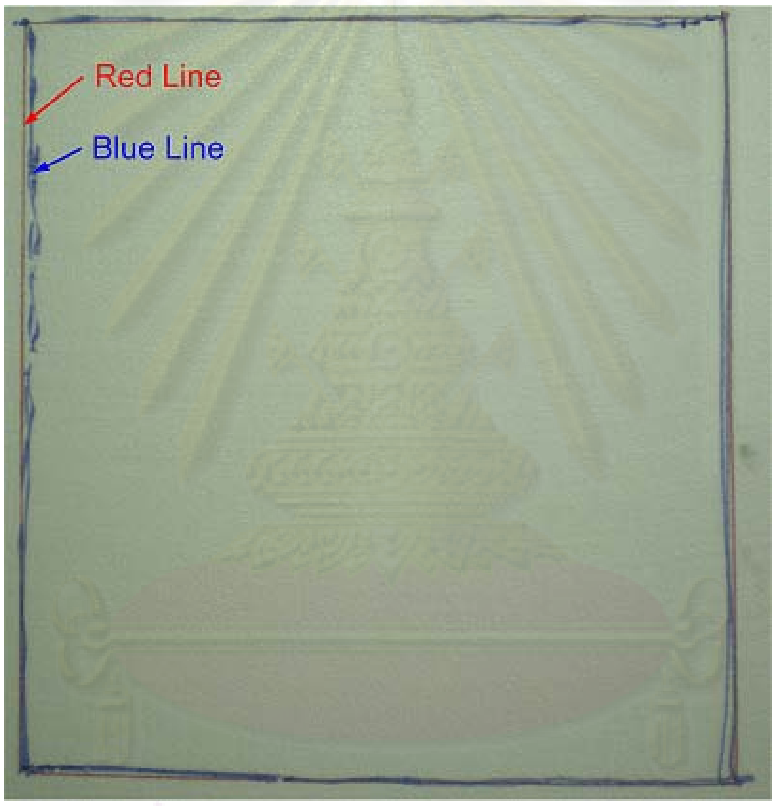


Figure 5.8: Playback test result

5.5 Servo Valves Model

The servo valve in every axis is identified using Recursive Least Square (RLS) Algorithm. Square signals is given as the inputs and the outputs are recorded as shown in Figure 3.14. The estimate model has structure as shown in (3.70). Complete estimated models of the six servo valves are shown in Table 5.2.

Table 5.2: Estimate Model of Servo Valves

Axis	Estimate Model	
	$A(z^{-1})$	$B(z^{-1})$
1	$1 - 1.4825z^{-1} + 0.4821z^{-2}$	$-0.0026z^{-1} - 0.0116z^{-2}$
2	$1 - 1.6972z^{-1} + 0.6973z^{-2}$	$-0.0008z^{-1} - 0.0036z^{-2}$
3	$1 - 1.715z^{-1} + 0.715z^{-2}$	$-0.0011z^{-1} - 0.026z^{-2}$
4	$1 - 1.7023z^{-1} + 0.7023z^{-2}$	$-0.0003z^{-1} - 0.0039z^{-2}$
5	$1 - 1.7339z^{-1} + 0.7339z^{-2}$	$-0.0006z^{-1} - 0.0043z^{-2}$
6	$1 - 1.9267z^{-1} + 0.9268z^{-2}$	$-0.0004z^{-1} - 0.0011z^{-2}$

5.6 Controller Coefficient

The PID controller structure is described in Figure 2.4 (b). By solving (3.77) and (3.83), the controller coeficients of $R(z^{-1})$, $S(z^{-1})$ and $T(z^{-1})$ are obtained as shown in Table 5.3.

Table 5.3: Controller Coefficient

Axis	Controller Coefficient	
1	$R(z^{-1})$	$1 - 1.7354z^{-1} + 0.7354z^{-2}$
	$S(z^{-1})$	$-6.8755 + 11.5107z^{-1} - 4.7479z^{-2}$
	$T(z^{-1})$	-0.1127
2	$R(z^{-1})$	$1 - 1.201z^{-1} + 0.201z^{-2}$
	$S(z^{-1})$	$6.243 - 14.678z^{-1} + 8.365z^{-2}$
	$T(z^{-1})$	-0.072
3	$R(z^{-1})$	$1 - 1.6984z^{-1} + 0.6984z^{-2}$
	$S(z^{-1})$	$-12.1557 + 20.7115z^{-1} - 8.6927z^{-2}$
	$T(z^{-1})$	-0.1368
4	$R(z^{-1})$	$1 - 1.5041z^{-1} + 0.5041z^{-2}$
	$S(z^{-1})$	$-21.4533 + 35.3147z^{-1} - 14.2424z^{-2}$
	$T(z^{-1})$	-0.381
5	$R(z^{-1})$	$1 - 1.478z^{-1} + 0.478z^{-2}$
	$S(z^{-1})$	$-19.8371 + 33.1837z^{-1} - 13.6731z^{-2}$
	$T(z^{-1})$	-0.3265
6	$R(z^{-1})$	$1 - 1.501z^{-1} + 0.501z^{-2}$
	$S(z^{-1})$	$-69.3273 + 127.8178z^{-1} - 58.978z^{-2}$
	$T(z^{-1})$	-0.1492

5.7 Controller Performance

The performance of PID controller at every axis is shown in Figure 5.9. Step input is employed to observe the PID controller performance individually. Solid line curve is the response of PID controller from simulation based on the estimate model obtained by identification whereas dotted line curve is the measured response of PID controller from the experiment. Figure 5.10 shows the error curves, that is the difference between measured values and reference values.

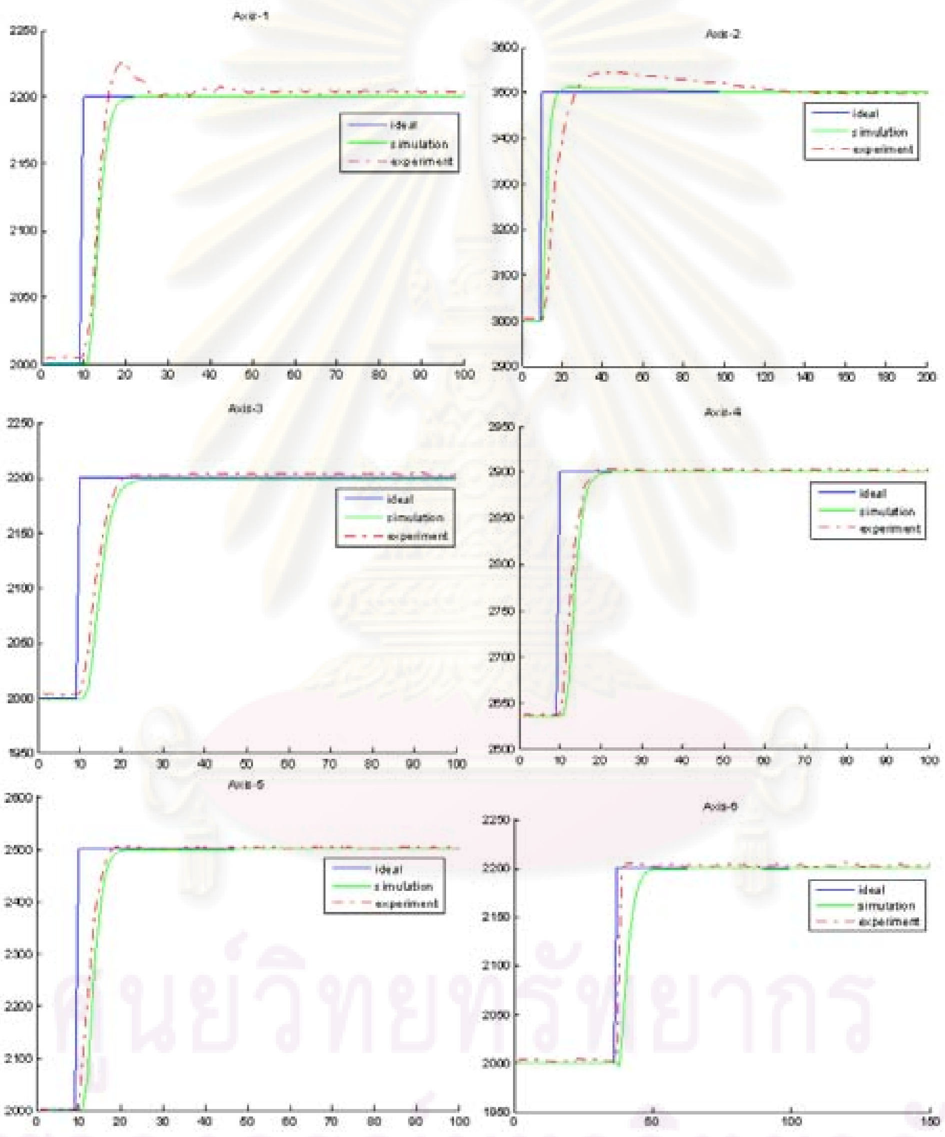


Figure 5.9: PID controller performances

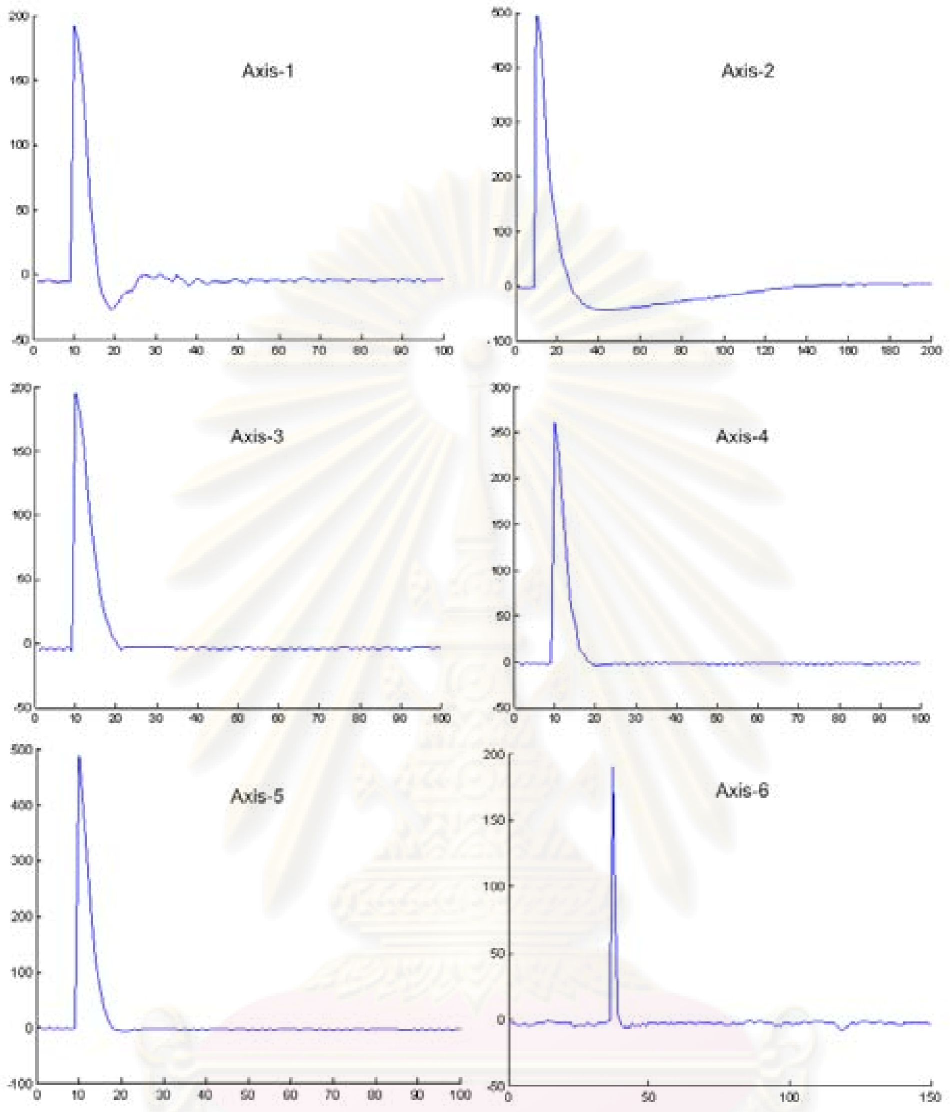


Figure 5.10: The error curves

Figure 5.11 shows the end-point error curves when it moves in every axis. Experiments were done by moving the end-point position for every axis individually. The positions are moved respectively from (191,6,149) to (189,23,149), (154,0,206) to (124,0,233), (180,0,135) to (187,0,121), (173,0,134) to (161,0,132), (191,1,148) to (187,-11,148) and (180,0,135) to (187,0,121).

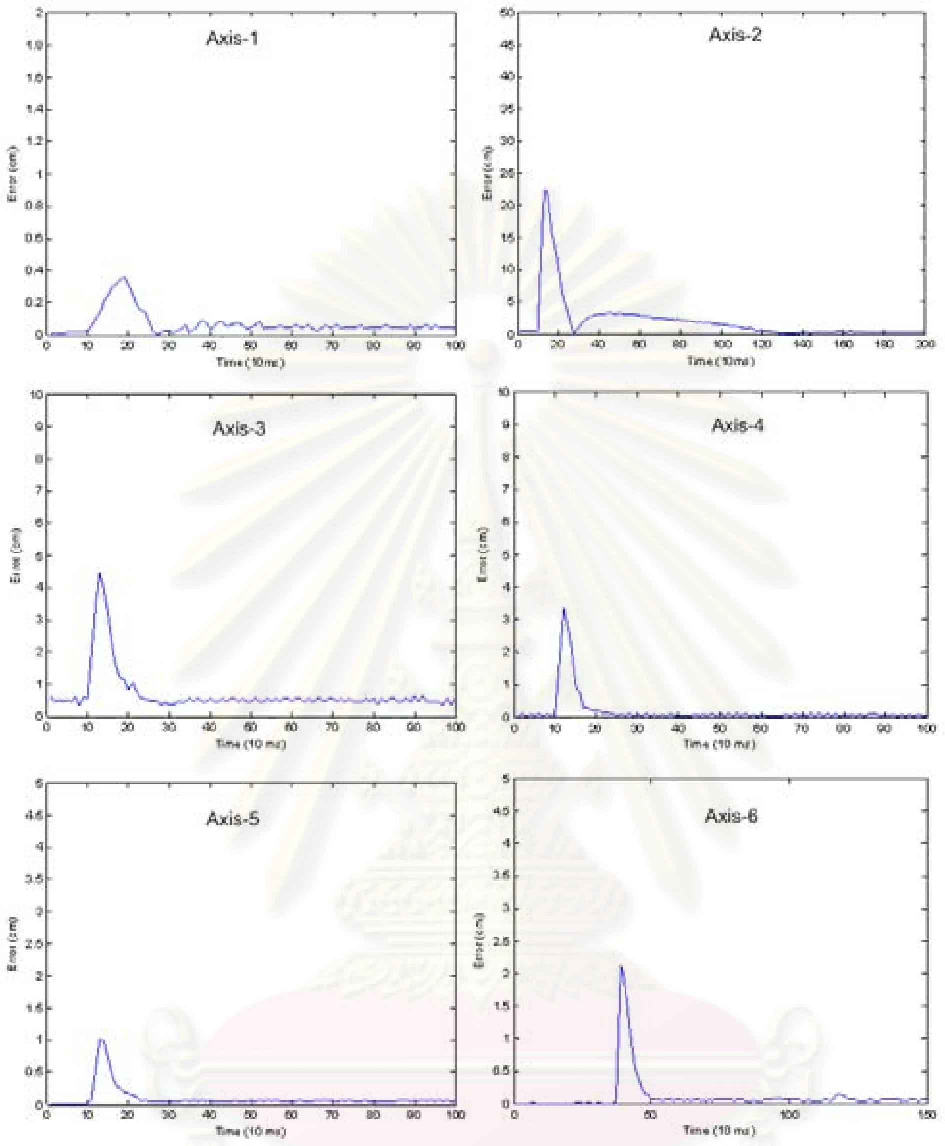


Figure 5.11: The end-point position error curves

5.8 Hardware Synthesis

Table 5.4 shows the synthesis report of the device utilization.

Table 5.4: Device utilization summary

Item	Number	Percentage
Number of Slices	3149 out of 3584	87
Number of Slice Flip Flops	3670 out of 7168	51
Number of 4 input LUTs	4283 out of 7168	59
Number of bonded IOBs	35 out of 97	36
Number of BRAMs	8 out of 16	50
Number of MULT18X18s	3 out of 16	18
Number of GCLKs	8 out of 8	100
Number of DCMs	2 out of 4	50

5.9 Demonstration Video

In order to show the experiment results on controlling the robotic arm clearly, demonstration videos were taken. These videos are also shared to public at http://www.youtube.com/user/arumdapta98.

5.10 Hardware Assembly

Figure 5.12 to 5.14 show the developed hardwares to realize the controller system.

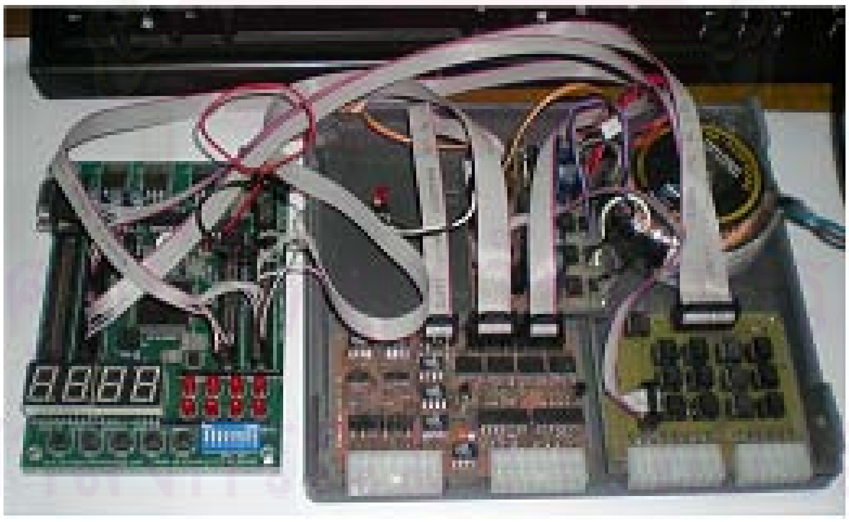


Figure 5.12: The developed hardware of controller system based on FPGA

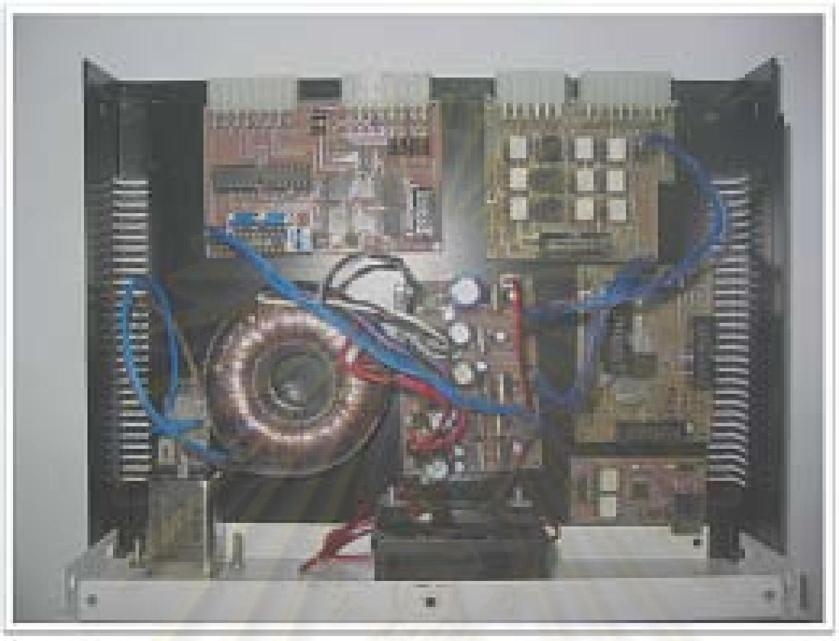


Figure 5.13: The developed hardware of controller system based on microcontroller

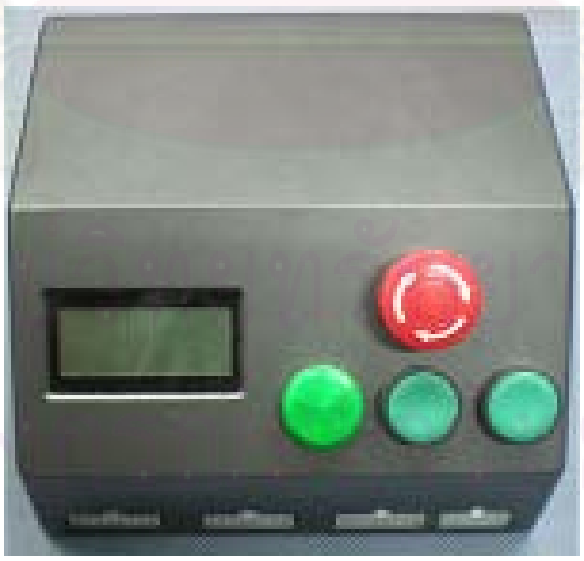


Figure 5.14: The assembled hardware

CHAPTER VI

DISCUSSION AND CONCLUSIONS

6.1 Discussion

This section discuss some experiment results both hardware and software as depicted in chapter 5.

6.1.1 Hardware

In this work, a 6-axis robotic arm controller is developed. Each axis has its own 2-DOF PID controller which control each servo valve at every axis independently. A 32-bit RISC soft-core processor (microblaze) with some peripherals and custom peripherals are constructed to realize those controller on a FPGA chip. The device utilization summary of hardware synthesis for controller implementation on Xilinx FPGA XC3S400 is shown in Table 5.4. From this table it looks that the developed hardware spends 87% of total resources provided by Xilinx FPGA XC3S400. The two biggest procentage of resource usage is allocated for custom peripheral and microblaze circuit which spend 21% and 19% respectively. The 47% procentage left is allocated for other peripherals such as timer/counter, SPI, UART, I/O etc. Custom peripheral is a dedicated 6 PWM generators and 6 SPI readers. This peripheral is designed to handle a specific job that is generating PWM signal and reading SPI data input. Using this costum peripheral we can save much more resources compare to when we use general 6 timer/counter and 6 SPI peripheral to handle the same job. This can reduce resources usage from 96% to only 21%.

The main clock of the controller system is driven by an external 25 MHz source clock. The microblaze processor clock, bus clock and most of peripheral clock use this 25 MHz source clock. Only PWM generator peripheral which uses 100 MHz source clock. This clock is obtained by multiplying the 25 MHz source clock 4 times through Digital Clock Manager. PWM generator needs faster clock because the faster clock is used the better analog output is produced.

Analog Board is an external device which needs more attention on hardware readiness test compare to other external devices such as I/O board and power supply board. Basically, analog board test consist of two tests: DAC test and ADC test. The DAC test can be done by generating any signals such as square, sawtooth, triangle or sinusoidal waveform then observe the analog output through oscilloscope. Whereas the ADC test is done by providing certain analog signal then read and display it on PC. Figure 5.3, 5.4, 5.5 and 5.6 results show that the analog board has work properly. It can produce analog signal through DAC and read analog signal through ADC satisfically.

A specific test is also done to check the performance of low pass filter circuit. Figure 3.31 shows that the frequency response and phase response of the designed filter has satisfied response as desired one in section 3.5.2.3.

Finally an overall hardware test is conducted to ensure that all hardwares are ready to use. The overall test and some partially tests above show that the designed hardwares are work well and ready in use. Unfortunately, those hardware never have tried to control the robotic arm in real because of the limited time on working schedule. However the 2-DOF PID controller design ever tried and implemented on microcontroller at preliminary work. The result as shown in Figure 5.9 has proved that the designed controller work properly even using microcontroller. Therefore we are sure that those designed controller will work properly as well if it is implemented on the FPGA based.

6.1.2 Controller

An inverse kinematics simulation must be performed before loading and running the trajectory file at the controller system. In this experiments, 3 kind of trajectory files are generated and simulated: square pattern, triangle pattern, and zig-zag pattern. To generate this trajectory files, some points must be defined previously in Cartesian coordinate by an operator. For example, points P_1(160,-50,180), P_2(160,-50,130), P_3(160,50,130) and P_4(160,50,180) are determined as a via points to generate square trajectory pattern in cm unit length. Every 5 cm movement in any direction, the Cartesian coordinate will be converted into Polar coordinate using closed-form inverse kinematics formula as given in 3.44 to 3.57. An polar to digital conversion then is employed to convert the corresponding Polar coordinate into digital value. Finally, the sequence of the digital value are stored in a Microsoft Excel format namely trajectory file.

This simulation is required to guarantee that the trajectory file can drive the robotic arm move following the desired pattern properly. By assuming that the controller performance is ideal, this simulation provides a good enough testing tool represents the real motion of robotic arm. Chosen trajectory file then can be loaded, ran and stored in the controller system safely. Figure 5.7 and 4.14 show an example of this simulation. It looks that desired trajectory file could drive the robotic arm satisfically.

Having generated and simulated, the trajectory file is loaded into controller system. Now it is the controller responsibility to drive the robot so that the angle in every axis following the sequence values in the trajectory file. The performance of 2-DOF PID controller in each axis is shown in Figure 5.9. Step input is employed to observe these performance individually and independently. Solid line curve represents the response of the controller system from simulation which is simulated by the model obtained from identification whereas dotted line curve represents the real response of the controller system which is obtained from. From this figure it looks that the controller responses both from simulation and from experiment resemble one another. It means that the estimate model of the servo valves are closed enough to the real model.

Controller error, that is the difference between measured angle and the reference input in each axis, is less than 6 (digital value) or less than 0.5^o at steady state condition as shown in Figure 5.10. End-point error is the difference between the targeted position and the measured position in Cartesian coordinate. The measured end-point error is less than 3 cm as shown in Table 5.1. This error is also proven by experiment result as shown in Figure 5.8. The error yielded by that experiment (the difference between the real square pattern at playback mode and the ideal one) is less than 1.5 cm. This error can be said small enough when this error can be tolerated and acceptable such that the painting target do not need rework.

As shown in Figure 5.9, the axis-2 requires much more time to reach steady state. Most axes require 10 up to 30 sampling time to be stable. Only axis-2 that requires more than 100 sampling time or equal with more than 1 second to achieve the steady state. From the experiments, axis-2 can not be forced to reach the reference too fast. If this is done, it yields oscillation and it never be stable. This problem is happened because the center of mass of the whole machine, i.e weight of the robot is on this axis, hence the velocity of this axis is not directly proportional to the DC component of the servo valve. PID controller will think that the weight is the disturbance of the system i.e that have not formulated in the model. Moreover when the angle of the axis changes, the center of mass moves and the force that counter weight changes too. It mean that the model may change dependent on the angle at a particular time. Therefore we have to choose slower rise-time for this axis.

6.2 Conclusions

This work is conducted to develop an embedded system of 6-axis robotic arm controller. This means that both hardware and software are designed here. The controller system itself is integrated and embedded in a single chip. This is usually called a system on chip. Xilinx FPGA XC3S400 is chosen to implement that controller system because its features satisfy the requirement and its price is so cheap.

The angle of robotic arm in every axis is measured from the potentiometer attached in every joint. It then be converted from analog value to digital value by ADC part. This measured angle in digital value will be compared with reference values. The difference between measured value and the reference value, namely error, is proceeded by PID algorithm to produce control signal. This control signal is then used to drive the servo valve of robotic arm such that the angle in every joint following its reference value.

To design proper PID controller, the servo valves were identified. The estimate model of servo valves and the PID controller are simulated using matlab tool. After it satisfied the desired specification, the controller is then implemented on microcontroller. Finally, the microcontroller will be replaced by FPGA. Experiment results show that the designed controller has worked very well.

A set of formulas were developed to solve the inverse kinematics problem. With some constraints such as by keeping the end-point orientation constant, the inverse kinematics problem can be solved with closed-form method. This method has been worked correctly as shown in the simulation and the experiment results.

In this work, a GUI user application software is developed as well. This GUI application software can be used to setup and control the robotic arm controller system interactively. Through this GUI application software, an operator can command the robot easily. Some interactive menus are designed as easy as possible to use. Moreover, with its user-friendliness it can increase the productivity because the operator can modify and adjust anything regarding the robotic arm operation easily and quickly.

6.3 Suggestion for future work

In this work, we have not considered about the disturbance effect against the servo valves model. As mentioned in the discussion session, particularly in axis-2, its models may change depend on the end-point position. When its model change too much, it makes the PID controller unstable and oscillate. Therefore other identification methods which include the disturbance variable could be considered in the future future work to get better estimate model and to get better controller performance. Adaptive controllers such as MRAC or repetitive-path optimization are other points that can be further considered.

REFERENCES

[1] M. W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* Canada: John Wiley & Sons. 1989.

[2] R. Kelly, V. Santibazez, and A. Loria. *Control of Robot Manipulators in Joint Space.* London: Springer. 2005.

[3] S. Kucuk and Z. Bingul. The inverse kinematics solutions of industrial robot manipulators. in *Proceedings of the IEEE International Conference on Mechatronics*. (2004): 274-279.

[4] B. Naticchia, A. Girreti, and A. Carbonari. Set up of an automated multi-colour system for interior wall painting. *International Journal of Advanced Robotic Systems*. 4. 4. (2007): 407-416.

[5] Y. S. Kim, M. yung, H. ung, Y. K. Cho, J. Lee, and U. Jung. Conceptual design and feasibility analysis of a robotic system for automated exterior wall painting. *International Journal of Advanced Robotic Systems*. 4. 4. (2007): 417-430.

[6] F. You and G. Shao. Painting brush control techniques in chinese painting robot. in *IEEE International Workshop on Robots and Human Interactive Communication*. (2005): 462-467.

[7] Toxico. *Painting Robot Maintenance Manual.* Toxico Ltd. 1987.

[8] Toxico. *Standard Electric Connection Instruction of TOXICO Spray Painting Hydroulic Robot System.* Toxico Ltd. 1987.

[9] Toxico. *Software Robotic.* Toxico Ltd. 1990.

[10] S. Y. J. Xie and W. Qiang. A method for solving the inverse kinematics problem of 6-dof space manipulator. in *International Symposium on Systems and Control in Aerospace and Astronautics ISSCAA*. (2006): 382.

[11] G. Z. Grudic and P. D. Lawrence. Closed-form inverse kinematics solver for reconfigurable robots. in *IEEE International Conference on Robotics and Automation*. (2001): 2395-2400.

[12] M. Zoppi. Effective backward kinematics for an industrial 6r robot. in *IASME Design Engineering Technical Conferences Computers and Information*. (2002): 1-7.

[13] G. Z. Grudic and P. D. Lawrence. Iterative inverse kinematis with manipulator configuration. in *IEEE Transactions on Robotics and Automation*. (1993): 476-483.

[14] Y. S. Kung, K. H. Tseng, C. S. Chen, H. Z. Sze, and A. P. Wang. Fpga-implementation of inverse kinematics and servo controller for robot manipulator. in *IEEE International Conference on Robotics and Biomimetics*. (2006): 1163-1168.

[15] D. G. Bihn and T. C. S. Hsia. Universal six-joint robot controller. in *IEEE Control Systems Magazine*. vol. 8. (1988): 31-36.

[16] A. Bejo and W. Pora. An improvement of the end-point error for multiple-axis robotic arms using the lms algorithm. in *31st Electrical Engineering Conference (EECON-31)*. (2008): 807-810.

[17] A. Bejo, W. Pora, and H. Kunieda. Development of a 6-axis robotic arm controller implemented on a low-cost microcontroller. in *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology Conference 2009 (ECTI-Con 2009)*.

[18] R. S. G. Honegger and M. Brega. Application of a nonlinear adaptive controller to a 6 dof parallel manipulator. in *IEEE International Conference on Robotics and Automation*. vol. 2. (2000): 1930-1935.

[19] A. Bejo and W. Pora. Combination of model reference adaptive control and least mean square algorithms for robotic arm controllers. in *Asia International Sysmposium on Mechatronics*. (2008): 235-238.

[20] J. Olawale, A. Oludele, A. Ayodele, and N. M. Alejendro. Development of a microcontroller based robotic arm. in *Proceedings of the 2007 Computer Science and IT Education Conference*. (2007): 549-557.

[21] K. S. Fu, R. C. Gonzalez, and C. Lee. *Robotics: Control, Sensing, Vision, and Intelligence*. Mcgraw-Hill Book Company. 1987.

[22] L. Ljung. *System Identification: Theory for the User, 2nd ed.* Prentice-Hall. 1999.

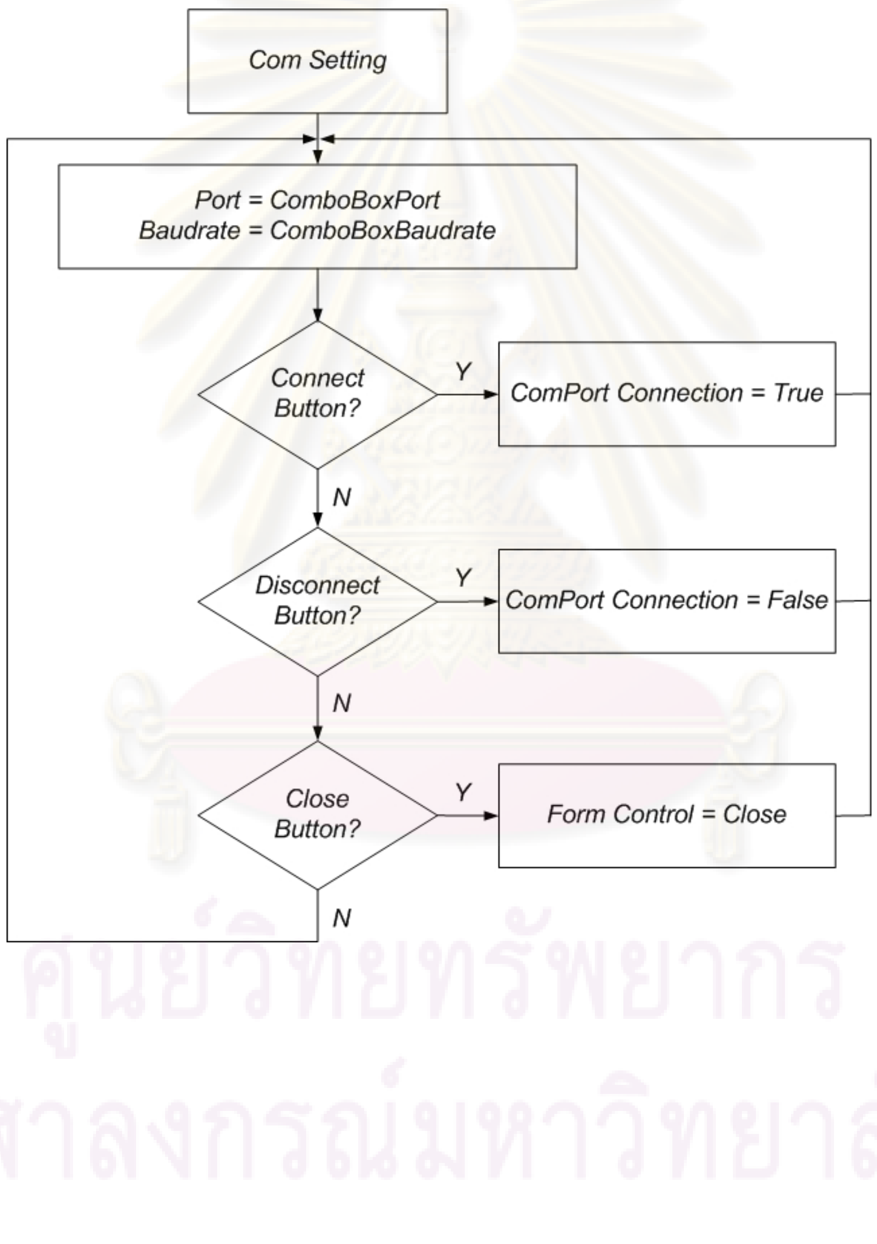
[23] Y. Zhu. *Multivariable System Identification for Process Control*. Netherlands: Pergamon. 2001.

[24] K. M. Moudgalya. *Digital Control*. John Wiley and Sons, Ltd. 2007.

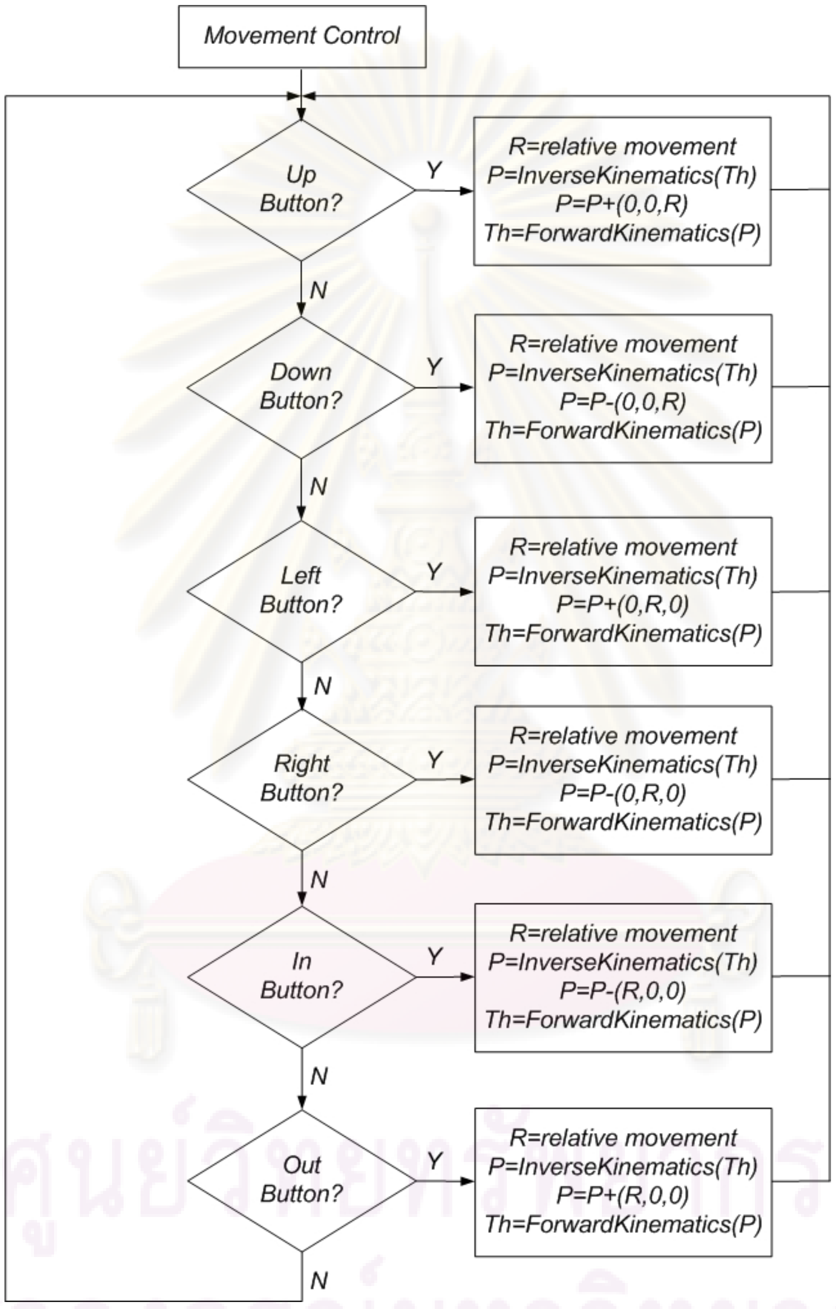
APPENDIX

Flowchart Diagram of GUI User Application Software

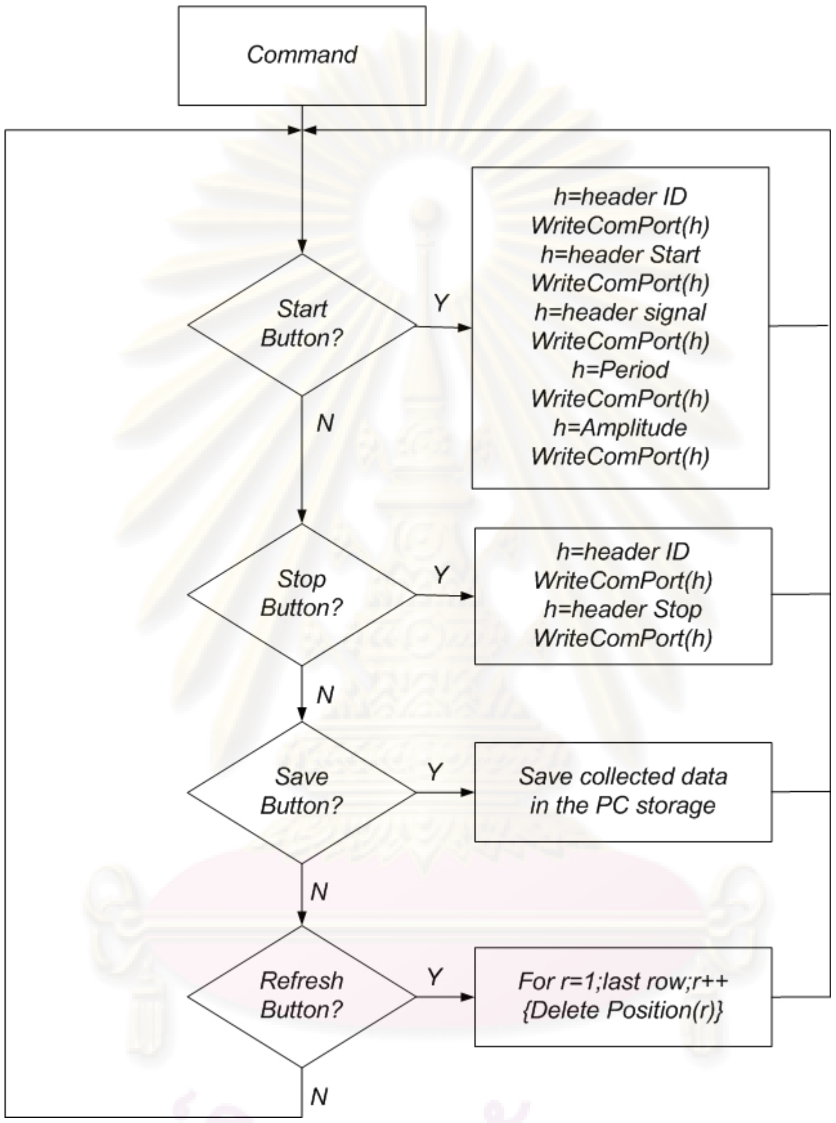
1. Communication Setting



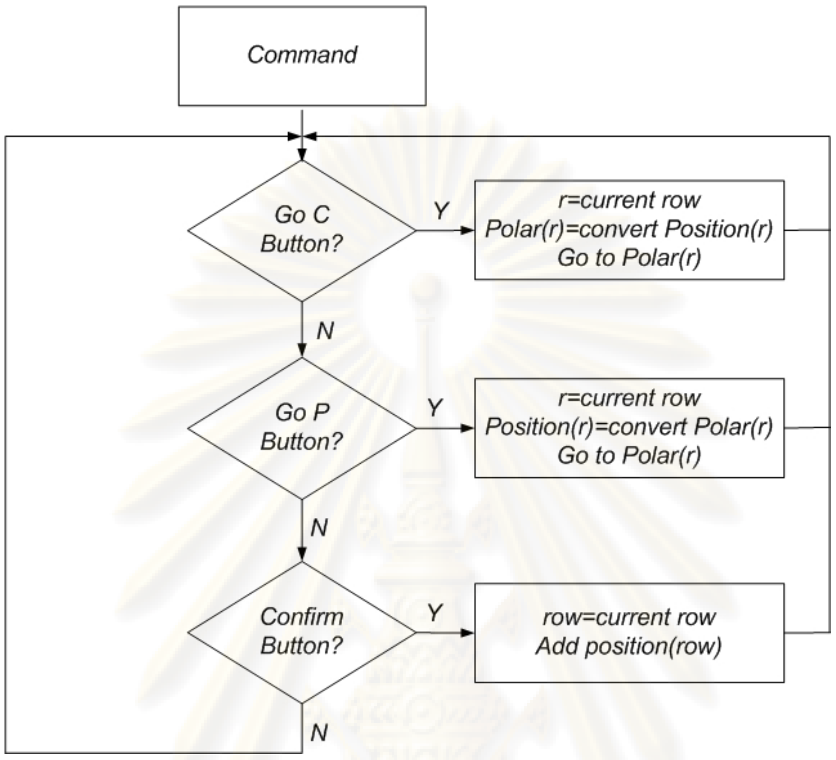
2. Movement Control



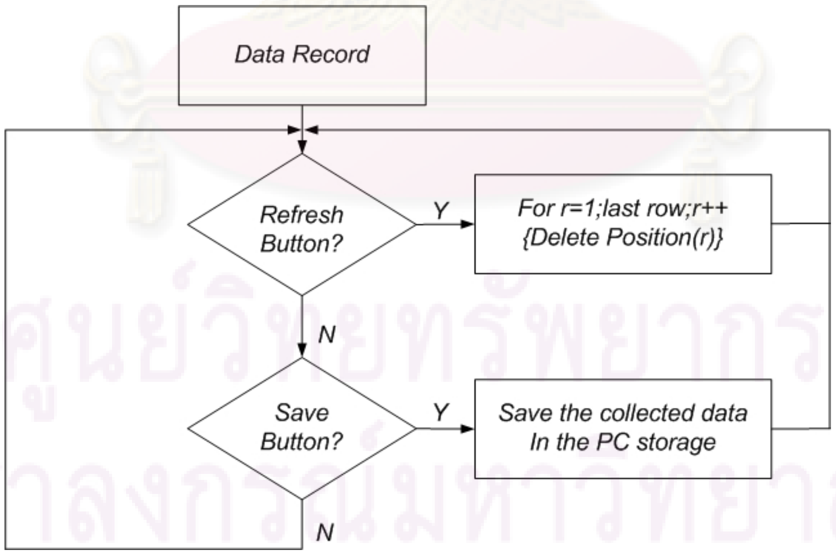
3. Command in System Identification



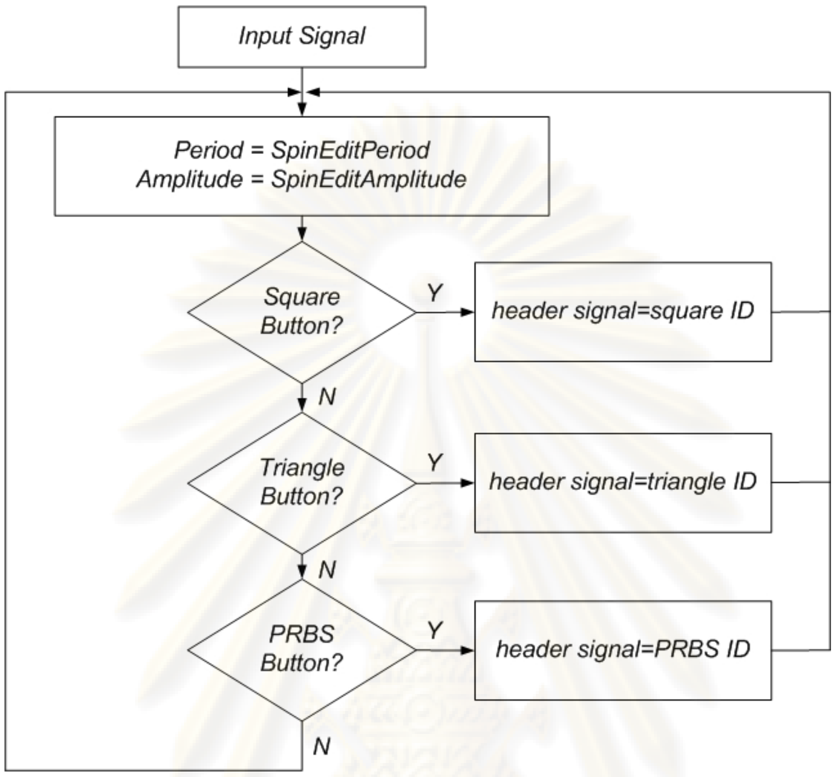
4. Command in Teaching Mode



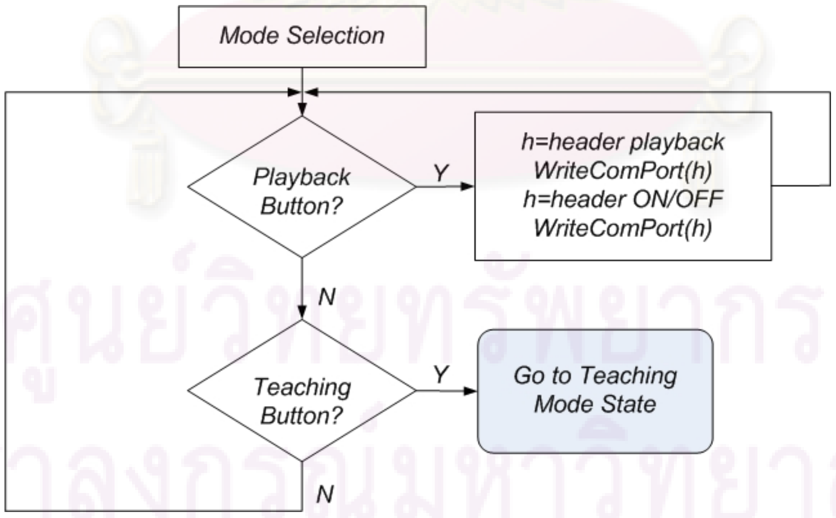
5. Data Record



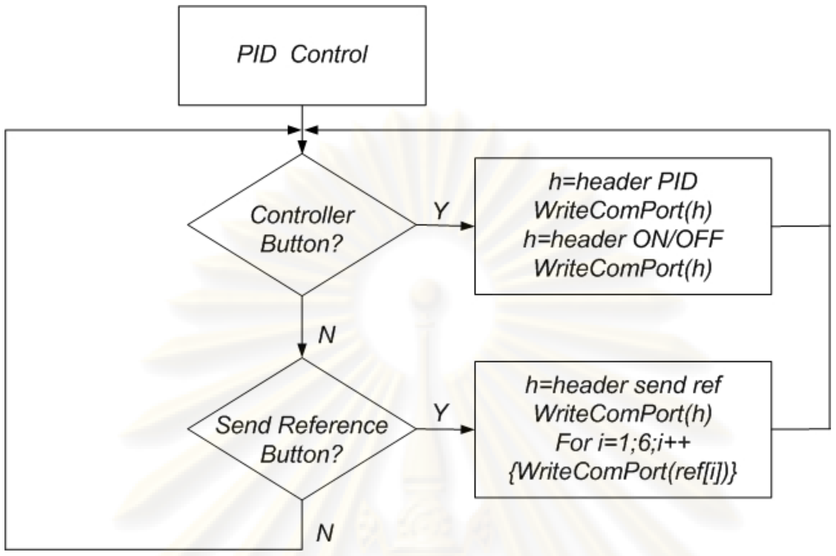
6. Input Signal



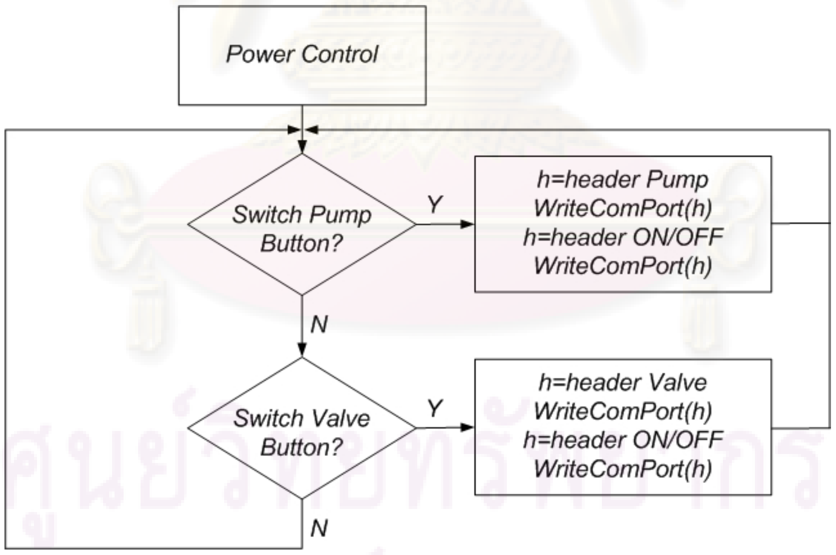
7. Mode Selection



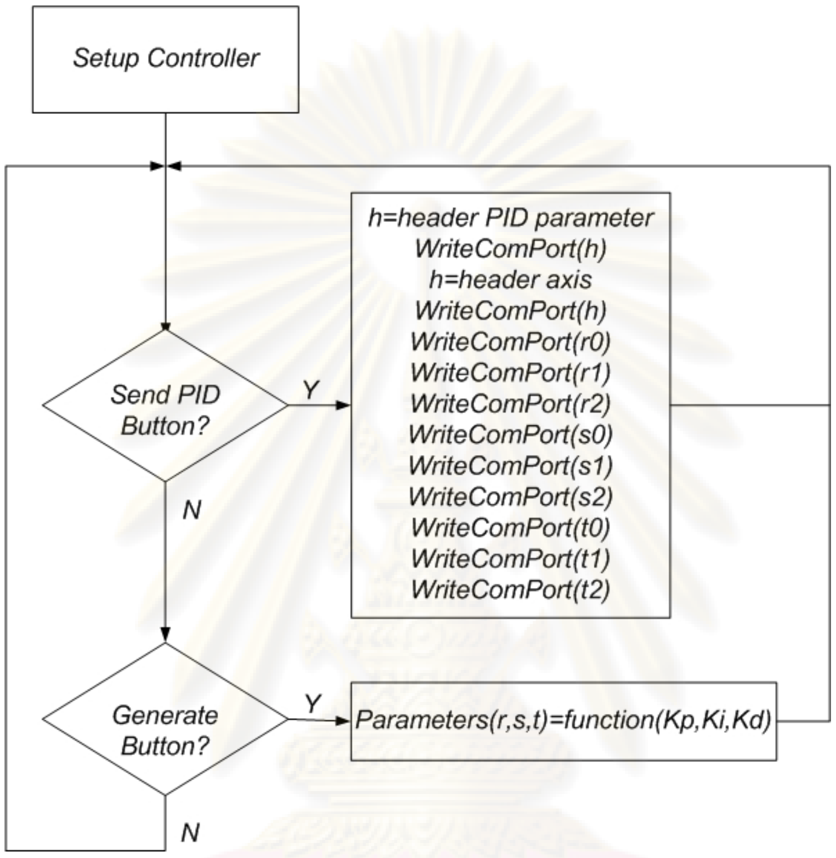
8. PID Control



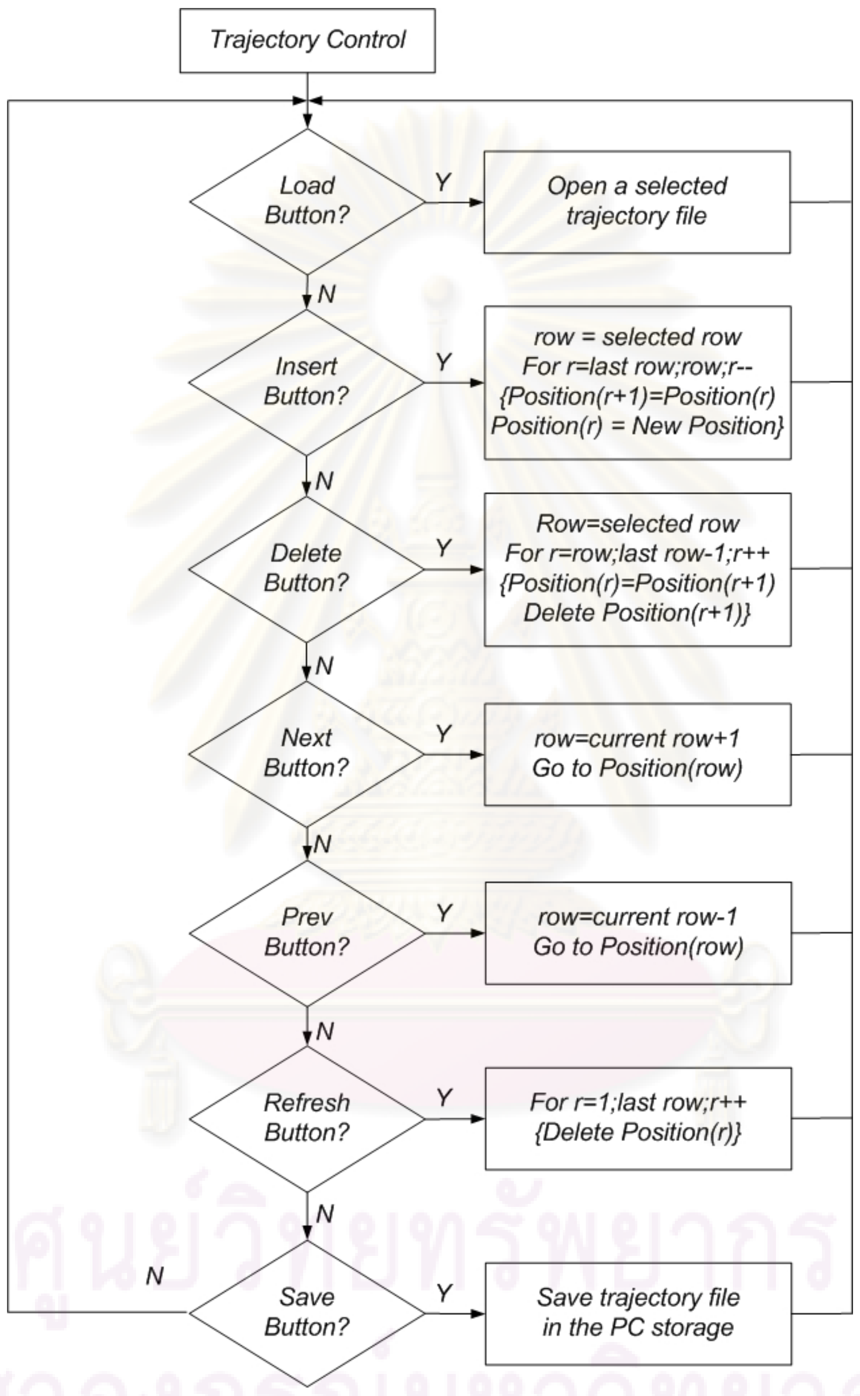
9. Power Control



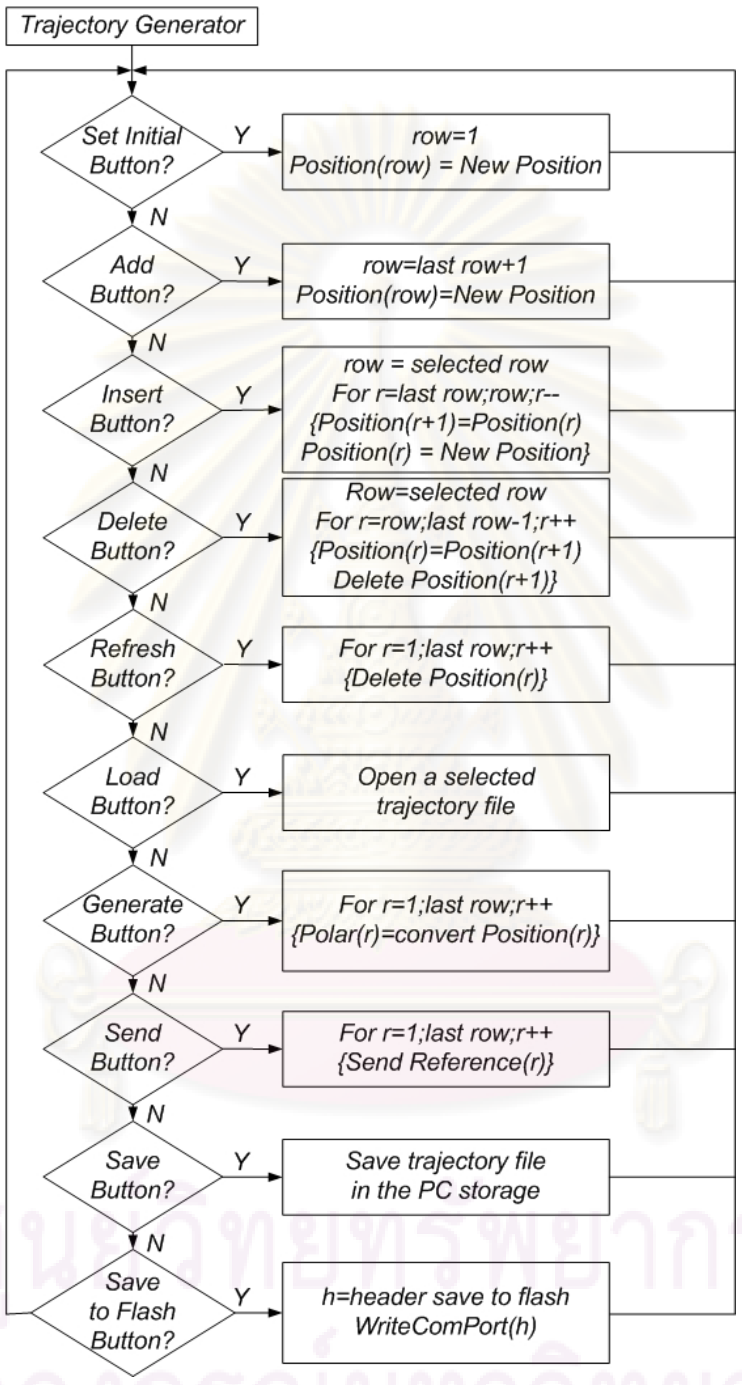
10. Setup Controller



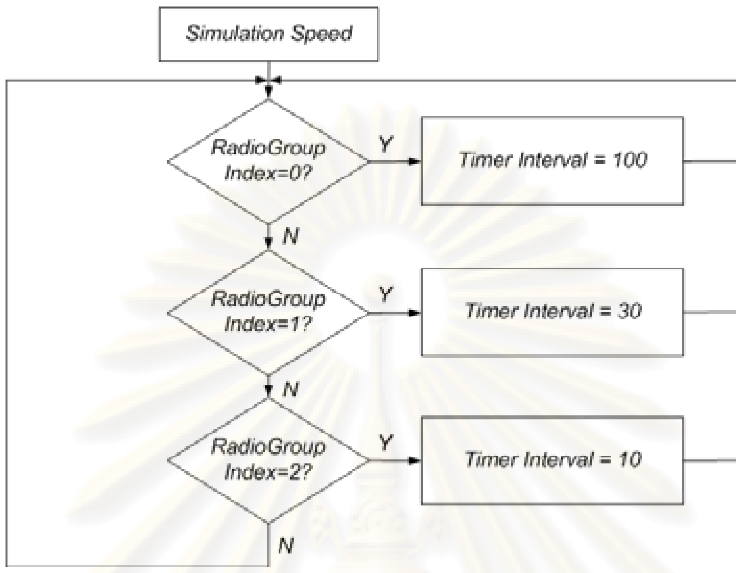
11. Trajectory Control



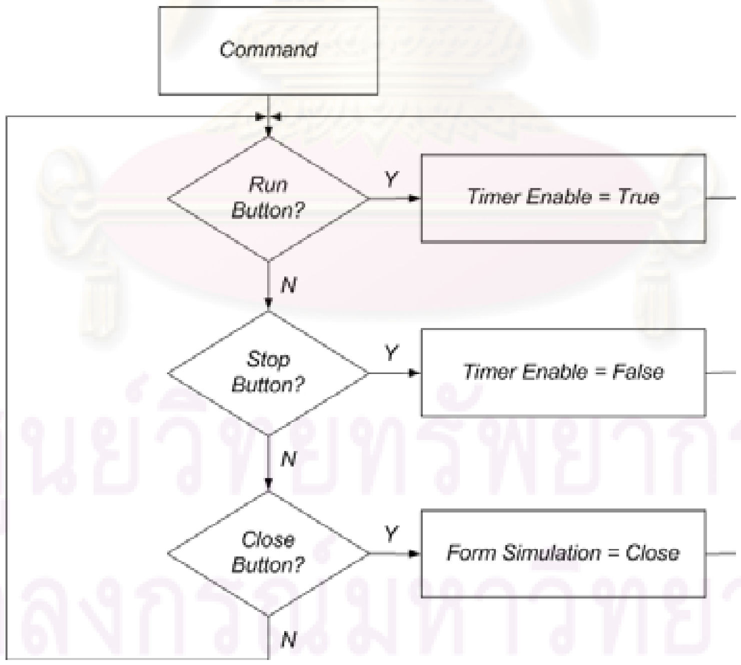
12. Trajectory Generator



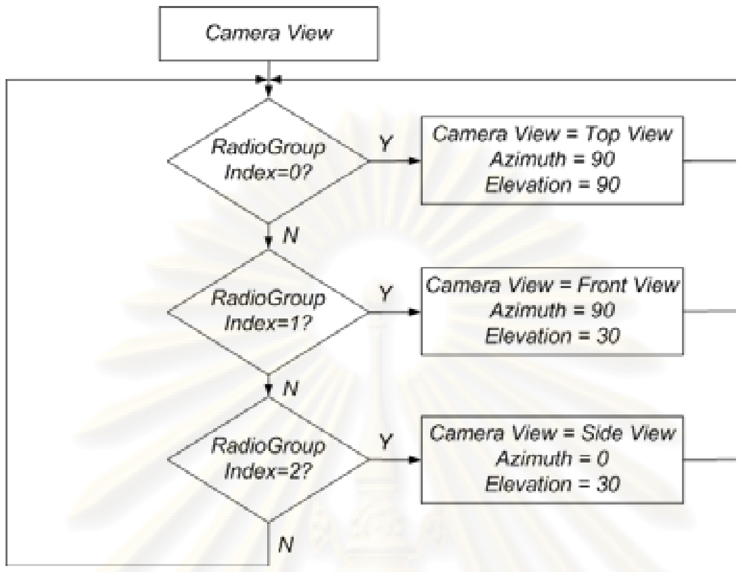
13. Simulation Speed



14. Command in Simulation



15. Camera View



Biography

Agus Bejo was born in Sleman, Indonesia, in 1980. He entered Gadjah Mada University, Indonesia in 1998 and received his Bachelor's degree majoring in Electrical Engineering in 2003. He joined LG Electronics Company as a research and development staff in 2003. In the end of year 2005, he decided to revert to Gadjah Mada University as a faculty staff. Since 2007, he has been granted a scholarship from the AUN/SEED-Net to pursue his Master's degree in Electrical Engineering Department at Chulalongkorn University, Thailand. Now He is a member of Embedded System and Digital IC Design Laboratory, Chulalongkorn University. His current research is related to the design a system on chip of robotic arm controller.