# Stony Brook University

# Experimental Verification of Nonlinear System Model Parameter Identification Based On Trajectory Pattern Method

A Thesis Presented

by

**Xiao Chen**

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

**Master of Science**

in

**Mechanical Engineering**

Stony Brook University

August 2010

**Stony Brook University**

The Graduate School

## Xiao Chen

We, the thesis committee for the above candidate for the Master of Science degree, hereby recommend acceptance of this thesis.

Jahangir Rastegar – Thesis Advisor
Associate Professor, Mechanical Engineering

Q. Jeffrey Ge – Chairperson of Defense
Professor and Deputy Chair, Mechanical Engineering

Imin Kao
Professor, Mechanical Engineering

This thesis is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Thesis

# Experimental Verification of Nonlinear System Model Parameter Identification Based On Trajectory Pattern Method

by

**Xiao Chen**

**Master of Science**

in

**Mechanical Engineering**

Stony Brook University

2010

The purpose of this thesis was to verify the Trajectory Pattern Method (TPM) based nonlinear system model parameter identification method for a two degrees of freedom closed loop planar robotic manipulator. The systematic procedure and the mathematical proof of convergence of the method were introduced first. A model based feedforward control system was constructed for the control of the planar manipulator. The software of the control system was implemented. Experiments were performed to demonstrate the effectiveness of the TPM based parameter identification

method.

The experimental results, which can be extended to a large class of nonlinear mechanical systems whose models are linear with respect to their model parameters, showed that the estimated parameters were getting closer to their nominal values during the parameter updating process. As the kinematics and dynamics parameters of the system approached their nominal values, the model based feedforward control system would control the system better. The integral of the squared errors of the outputs were calculated to show that the overall control performance was improved after each parameter updating step. The principle conclusion drawn from the experiment revealed that the developed parameter identification method was very effective on the highly nonlinear dynamic system.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, I would like to express my sincere appreciation and deepest gratitude to my advisor, Professor Jahangir Rastegar, for his suggestions and support during this research. His perpetual energy and enthusiasm in research have been an inexhaustible source of encouragement and inspiration.

Second, I would like to thank Professor Imin Kao and Professor Jeffrey Ge for their valuable comments and suggestions as members of the thesis committee.

Additionally, I am grateful that Mr. Dake Feng gave me constant help during my study and research.

# Chapter 1

# Introduction

## 1.1 Nonlinear System Model Parameter Identification

Most mechanical systems such as robot manipulators have nonlinear dynamics. Although sometimes it may be possible to describe such systems linearly over a restricted range around a certain working point, in general nonlinear systems can only be adequately characterized by a nonlinear dynamics model. Since a mathematical description of a process is the prerequisite to the analysis and design of its control system, the study of system identification and model parameter estimation has become an established branch of control theory.

Identification of system kinematics and dynamics parameters is very important for accurate description of the system dynamics for design and control purposes, especially when model based controllers are involved. Some

researchers have studied the techniques for nonlinear system structure identification while others have developed new approaches of estimating unknown parameters of a known nonlinear system. Gray et. al. (1998) used Genetic Programming for nonlinear model structure identification. This is an optimization method which automatically selects model structure elements from a database. Abdelazim and Malik (2005) used Takagi-Sugeno fuzzy logic grey box modeling for identification of nonlinear systems. Li et al. (2006) developed a two-stage algorithm for identification of nonlinear dynamics systems. In this algorithm an initial model is built from a large pool of basis functions or model terms, then the significance of each term is evaluated and all insignificant terms are replaced in the second step.

One the other hand, the research on nonlinear system parameter estimation is growing recently due to the need of having a more precise model for advanced control. For example, some very sophisticated control laws used on high speed and high accuracy machinery require the parameters to be as close to the nominal values as possible. Guo et al. (1989) used an adaptive control for linearized model of robotic manipulator in which a recursive least square identification scheme is employed to perform the on-line parameter estimation for feedback gains of controllers. Ha et al. (1989) developed a method for the estimation of the model parameters with which the system does not need a predetermined trajectory to follow. Lin (1994) presented an approach to identify the minimal linear combination of manipulator parameters by least square method. Lee et al. (1997) proposed to incorporate dynamics into fuzzy logic system so the resulting fuzzy logic system posses universal approximation capability and tested the theory on a flexible single link manipulator with highly

nonlinear joint friction. Dolanc et al. (2005) applied a special excitation signal for parameter estimation of nonlinear systems using a piecewise-linear Hammerstein model. Spottswood et al. (2006) introduced a novel reduced order nonlinear identification method. Kenne et al (2006) used radial basis function networks scheme for on-line state and parameter estimation of a reasonably large class of nonlinear systems.

A review of the published literature shows the following points. Firstly that there is no systematic method that can be applied to a large class of mechanical systems while at the same time the convergence of the parameter estimation method is guaranteed over a large range of working space. In other words some of the methods rely on the response from around a local working point rather than the entire working space in which the nonlinearity gets fully exhibited. Secondly, some of the excitation signals used for identification include a large range of frequencies and might put the nonlinear system out of control. This issue becomes more important when industrial processes are concerned.

In this thesis, a systematic approach is introduced for model parameter estimation of a large class of fully controlled deterministic nonlinear mechanical systems based on Trajectory Pattern Method (TPM), Rastegar, et al. The mathematical proof for convergence and other details of this TPM based model parameter estimation method can be reffered to Rastegar and Feng [1]. A digital signal processor (DSP) based DC motor control system is built to carry out the experimantal verifacation of the TPM nonlinear system parameter estimation method. Both hardware and software of the control system are presented. This method is tested on a highly nonlinear dynamic system which is a two degree of freedom closed loop planar robot manipulator. The results

from the experiments verified that the method is very effective and can be extended to a large class of nonlinear system.

## 1.2 Trajectory Pattern Method Based Model Parameter Identification

Trajectory Pattern Method uses a number of appropriate basic time functions containing several fixed parameters to synthesize a predetermined trajectory so that the resulting trajectory can transform the dynamic model of nonlinear system from differential equations into algebraic equations. We are considering those kinds of nonlinear systems that can be written in the form of linear equations with respect to their parameters or combinations of parameters. we assume that a mathematic model that precisely describes the dynamic behavior of the nonlinear system is provided. The model parameters are constants but the values of them are unknown. Since the majority of the mechanical systems can be modeled by second order differential equations, we only consider those ones here. But this dose not mean that the method is only valid for second order systems, we can easily extend to those high order nonlinear systems. These models can be expressed or approximated by Taylor Series expansion as polynomials with parameters being coefficients of each term. A nonlinear system with these considerations can be modeled as

$$m\ddot{x} + D(x, \dot{x}, k_1, k_2, \ldots, k_n) = u \tag{1.1}$$

where $x$ is the output, $u$ is the input and $m$, $k_1$, $k_2$, ..., $k_n$ are system parameters to be identified. The function $D$ is a generic nonlinear polynomial function expressed as

$$D(x, \dot{x}, k_1, k_2, \ldots, k_n) = \sum_{j=1}^{n} k_j x^{p_j} \dot{x}^{q_j} \tag{1.2}$$

where $k_j$ is the coefficient and $p_j$, $q_j$ are the exponents of the variables $x$ and $\dot{x}$ in the $j$th term. Generally differential equations are very difficult to solve, especially when we have a multiple inputs multiple outputs high order nonlinear system. However, the Trajectory Pattern Method transform the inverse dynamic of the system into a simple algebraic equation. The trajectory pattern is determined in such a way that the desired motion is realized. By using some optimality criterion such as minimum time or minimum energy to systhesize different motions, we get different trajectory patterns with different trajectory parameters. The trajectory patterns that have been employed to date and appear to be most advantageous are those constructed using a number of basic sinusoidal time functions and their harmonics. On the other hand, the selected pattern must start from initial conditions of zero velocity, zero acceleration, zero jerk and end with zero velocity, zero acceleration and zero jerk in order to be repeatable. In addition, it is desired for a selected trajectory to have minimum number of harmonics for parameter estimatin purpose. It has been proved that the following pattern satisfies all the aforementioned conditions with a minimum number of harmonics([2],[3]).

$$x_r(t) = \lambda_0 + \lambda_1[\cos \omega t - \frac{1}{9} \cos 3\omega t] \tag{1.3}$$

where $\omega$ is the fundamental frequency of the trajectory and $\lambda_0$ and $\lambda_1$ are the trajectory parameters. This point to point motion begins at time $t_s = 0$ and ends at time $t_e = \frac{\pi}{\omega}$.

Figure (1.1) shows the block diagram representation of this method. The derivations provided in the remainder of this chapter is from Rastegar and Feng [1]. In order to make the actual output $x_a$ follow the predetermined trajectory $x_r$ as closely as possible, a model based feedforward control scheme is employed. The feedforward input $f_f$ is constructed based on the inverse dynamics model with the estimated parameters and used to compensate for the nonlinearity.

$$f_f = m'\ddot{x}_r + D(x_r, \dot{x}_r, k'_1, k'_2, \ldots, k'_n) \tag{1.4}$$

The feedback controller is then applied to compensate for errors in the estimated system parameters and errors due to input disturbances. The error is $e = x_a - x_r$. A PD controller is chosen, so the feedback can be expressed as $C(e) = G_1 e + G_2 \dot{e}$. From the control block diagram, we can get the following equation:

$$m\ddot{x}_a + D(x_a, \dot{x}_a, k_1, k_2, \ldots, k_n) = f_f + C(e) \tag{1.5}$$

Because $D$ is a polynomial function, the second term of the above equation

can be rearranged as the following

$$\begin{aligned} D(x_r + e, \dot{x}_r + \dot{e}, k_1, k_2, \ldots, k_n) &= D(x_r, \dot{x}_r, k_1, k_2, \ldots, k_n) \\ &+ D(e, \dot{e}_r, k_1, k_2, \ldots, k_n) + D'(x_r, e, \dot{x}_r, \dot{e}, k_1, k_2, \ldots, k_n) \end{aligned} \tag{1.6}$$

where $D'$ is another polynomial function that has the same order with $D$ and it is a function of the productions by $x_r$, $e$, $\dot{x}_r$ and $\dot{e}$. A very important equation can be deduced from the above three equations.

$$\begin{aligned} -(G_1 e + G_2 \dot{e}) + m\ddot{e} + \sum_{i=1}^{n} k_i e^{p_i} \dot{e}^{q_i} + \sum_{i=1}^{n} k_i \sum_{j,l=1}^{i} c_{ijl} e^j (\dot{e})^{p_i - j - 1} x_r^l (\dot{x}_r)^{q_i - l - 1} \\ = \Delta m \ddot{x}_r + \sum_{i=1}^{n} \Delta k_i x_r^{p_i} \dot{x}_r^{q_i} \end{aligned} \tag{1.7}$$

where $\Delta m = m' - m$, $\Delta k_i = k_i' - k_i$ are the errors in the estimated values of the system parameters. In feedforward control theory, it is shown that if we have a good estimation of the system parameters, the system will track the input command with a good accuracy. In other words, if $\Delta m$ and $\Delta k_i$s are small enough, the error $e$, $\dot{e}$ and $\ddot{e}$ will be very small. Comparing $C(e) = (G_1 e + G_2 \dot{e})$ with the rest of the term on the left hand side of equation (1.7), we can see that if the feedback gains $G_1$ and $G_2$ are large enough we can ignore all the terms on the left hand side except for $C(e)$, especially when the trajectory is small ($x_r$ and $\dot{x}_r$ in those terms are bounded functions and the maximum values are

small if the trajectory is small). So we can simplify the above equation to be

$$-(G_1 e + G_2 \dot{e}) = \Delta m \ddot{x}_r + \sum_{i=1}^{n} \Delta k_i x_r^{p_i} \dot{x}_r^{q_i} \tag{1.8}$$

However, even if the estimated parameters used in the feedforwad are far from the nominal values, as long as the trajectories are small enough, we can still get the above equation. This is proved in Rasteger and Feng [1].

Thus by selecting small enough trajectory parameters and large enough feedback control gains, the differential equation describing the error $e$ becomes linear. By substituting the trajectory pattern equation (1.3) into equation (1.8) and after certain amount of algebraic manipulations we get:

$$\begin{aligned} -(G_1 e + G_2 \dot{e}) = a_0 + a_1 \sin(\omega t) + a_2 \cos(\omega t) + a_3 \sin(2\omega t) + a_4 \cos(2\omega t) \\ + a_5 \sin(3\omega t) + a_6 \cos(3\omega t) + a_7 \sin(4\omega t) + a_8 \cos(4\omega t) + \ldots \end{aligned} \tag{1.9}$$

where $a_i$s are function of errors in the estimated values of the system parameters $\Delta m$, $\Delta k_i$s and the trajectory parameters $\lambda_0$, $\lambda_1$ and $\omega$

$$a_i = a_i(\lambda_0, \lambda_1, \omega, \Delta m, \Delta k_1, \Delta k_2, \ldots, \Delta k_n) \tag{1.10}$$

Thus, by measuring errors $e$ and $\dot{e}$ during point to point motions with trajectory pattern in equation (1.3) (initially with trajectory patterns with relatively small lengths and low fundamental frequency $\omega$) and then transforming the measured errors into a Fourier Series with the fundamental frequency $\omega$, then the difference between the obtained harmonic coefficients and the $a_i$ in equation (1.10) will be due to the errors in the values of the estimated system

parameters, i.e., $\Delta m, \Delta k_1, \Delta k_2, \ldots, \Delta k_n$. It is, however, shown that the latter relationship is readily derived analytically. Thus, using such a relationship, the required corrections in the values of the estimated system parameters can be calculated from the aforementioned measured error signals. By repeating the selected point to point motions and properly increasing the length of the trajectory path and its fundamental frequency $\omega$ (i.e., speed) as the calculated corrections become small enough, the present model parameter estimation procedure should allow convergence to the nominal values of the system parameter.

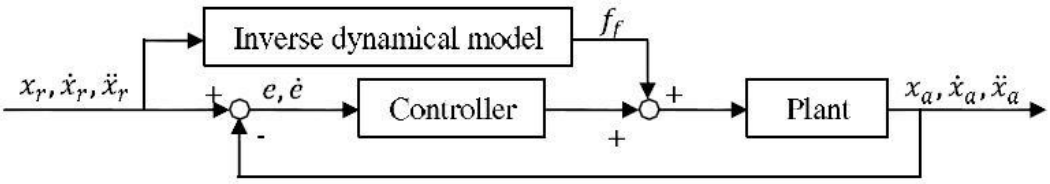


Figure 1.1: Control block diagram

# Chapter 2

# DSP based control system for parameter identification experiments

## 2.1 Hardware of the Control System

In this chapter, a DSP control system is introduced which was used in the experiments to implement the TPM based model parameter estimation theory and determine its effectiveness on a highly nonlinear dynamics system. The hardware and software of the system are presented. A M6713 DSP PCI plus-in board from Innovative Integration is used. M6713 DSP can be used for embedded data acquisition, servo control, stimulus-response and signal processing tasks. In comparison with a PC based control system, the biggest advantage of the DSP based control system is its capability to perform real-time tasks.

One of the biggest issues in using the PC for data collection, control and communication applications is the relatively poor real-time performance associated with the system. Despite the high computational power of the PC, it cannot reliably respond to real-time events at rates much faster than a few hundred hertz. The PC is really best at processing data, not collecting and/or transmitting it from/to external devices. In fact, most modern operating systems like Windows are simply not focused on real-time performance, but rather on ease of use and convenience. The solution to this problem is to provide specialized hardware assistance responsible solely for real-time tasks. Much the same as a dedicated video subsystem is required for adequate display performance, dedicated hardware for real-time data collection and transmission and signal processing is needed. This is precisely the focus of the M6713 baseboard – a high performance, dedicated digital signal processor coupled with real-time data I/O capable of flowing data via the PCI bus.

The details of the control scheme are shown in Figure (2.1). The M6713 DSP PCI-plug in baseboard resides in a host PC which downloads the parameters and programs into and communicates with the DSP board as well as receives the data from the DSP board for analysis. The M6713 is built around the powerful and C-friendly floating point TMS320C6713 DSP. To complement this core, one or two modular Omnibus I/O modules may installed into the onboard I/O sites. In this case, an A4D4 module is chosen for servo control purpose.

The M6713 receives the trajectory parameters, estimated model parameters and PD control gains from the host computer and then calculates the PD feedback and feedforward control commands and uses the A4D4 analog I/O

card to output the control signal. The feedback signal in this case is the angular displacement generated by an optical encoder. The calculated control signal is the output from the A4D4 and input to the motor power amplifiers. The A4D4 module provides the target card processor with four channels of high speed 200 kHz, 16-bit resolution analog input to digital output conversion (A/D) per module slot. In addition, four channels of high speed 200kHz, 16-bit resolution digital input to analog output conversion (D/A) are provided. The A4D4 has analog I/O that is tightly coupled with the DSP, making it well suited for controls purpose. The analog output function of the A4D4 is used in our application to output analog signal to the motor power amplifiers. Digital data is written to the D/As for output via a set of memory mapped locations. The D/A output is triggered and updated via the hardware timer and the analog trigger selection matrix on the A4D4. In this case, a host timer is programmed to run at the required sample conversion rate and the trigger selection matrix is programmed to direct the timer output to the D/A converter as a conversion strobe signal. Finally the analog output is filtered with high speed and low offset op amps.

The analog output control signals from the M6713 have to be amplified in order to drive the DC motors. We use the BD25A20AC Series PWM servo amplifiers for the system. These brushless servo amplifiers are designed to drive brushless DC motors at a high switching frequency. They are fully protected against over-voltage, over-current, over-heating and short-circuiting. The amplifiers work in current mode and the output currents is proportional to the input voltages sent from the DSP controller. So the torque generated by the DC motor becomes proportional to the DSP control signal.

The TPM based model parameter estimation algorithm was tested on a planar closed loop manipulator. The angular displacements of the arms are feedback to the digital I/O ports of the M6713 by the optical encoders. The M6713 includes 32-bits of software programmable digital I/O for acquiring digital inputs. The Digital I/O Port is a set of simple input and output latches. Outputs are enabled on a byte basis as programmed by the DIO control register. The input latch may be enabled to capture the pins whenever it is not being read, or whenever an external signal (EXT DIG CLK) is low, as enabled by the Digital I/O control register. When used in the internal enabled mode, the value read is the state of the pins immediately before the DSP reads them.

Reading the feedback correctly is a very critical problem in such a control system. The optical encoders chosen for the control system are "US digital E2 1024" and "US digital E3 2048". They are incremental rotary optical digital encoder. By counting two bits (channels), the pluses from encoder can be converted to relative angular position measurement. The signals from A and B channels can be decoded to yield the direction of rotation as shown in Figure (2.2). If the motor is rotating in clockwise (CW) direction, the phase of channel A is ahead of channel B by 90 degree. On the other hand, the phase of A falls behind channel B by 90 degree, i.e., when the motor is rotating in the counterclockwise (CCW) direction. So we can use the state of one channel as the gate, detect the changes of the state in the other channel to get the angular displacement increments. For example, taking the high state of channel B as the gate, if there is a falling edge of channel A, we know that the motor is turning a certain amount in the CW direction. If the rising edge of A appears,

we know that the motor is turning in the CCW direction.

## 2.2 Software and Programming of the Control System

The M6713 is designed for real time signal processing while the PC is good for data analysis. The programming of the system has two parts. Here we are using cross-platform graphic user interfaced (GUI) toolkits Qt4 of the Innovative Integration-authored Malibu library to design a GUI host program that runs on the host computer. The other task is to build a DSP project program for servo control.

### 2.2.1 Graphic User Interfaced Host Program

The host program takes the advantage of the Innovative Integration-authored Malibu library and uses the cross-platform GUI toolkits Qt4 to build the GUI.

Malibu is a powerful, feature-rich software library designed to meet the challenges of developing software capable of high speed data flow and real-time signal processing on the host PC. Malibu adds high performance data acquisition and data processing capabilities to C++ with a complete set of functions that solve data movement, analysis, viewing and logging and fully takes advantage of the object oriented nature of C++. Harnessing the power of the Microsoft Visual C++ and other IDE environments, Malibu offers the most powerful and flexible tools to rapidly integrate high-end data processing in applications. This class of library offers the means to the user to synchronize

the host side data movement and processing with hardware driven data that is transferred to or from DSP. Rapid application development is achieved using C++ reusable principle. Malibu is the library that allows the users to create their own real time applications running under Windows or Linux.

Qt is a cross-platform application development framework. The codes which are written on one platform (e.g. Window) can be compiled easily on another platform (e.g. Linux). Qt is actually a C++ library. The most famous applications using Qt are KDE, Google Earth and so on. Qt is dual licensed. It can be used for creating open source applications as well as proprietary ones. Qt is well established in the open source community and thousands of open source developers use Qt all over the world.

The host program communicates with DSP, sends parameters to, receives and recorded data from the DSP. Functions of the host program include:

1. Download the servo control program from PC to DSP;
2. Display the status and information of the DSP;
3. Send estimated model parameters to DSP;
4. Send trajectory parameters;
5. Send PD control gains;
6. Receive and analyze the recorded data;

The interface of the GUI host program is shown in Figure (2.3). The "open" button at the left upper corner opens the DSP project and use "COFF loading" from innovative Malibu to download the DSP codes into M6713. The progress bar at the button updates the percentage of the downloading process. Once the DSP codes are loaded, the dsp status indicator changes the "HALTED" to "RUNNING". This means the DSP control program has started and is

ready to take the parameters and update the control signal. The "parameter setting" box is divided into three parts. They are "trajectory parameters", "PD control parameters" and "model parameters", respectively, from top to bottom. After these parameters are set and updated, the user can click the "download parameters" button to send these parameters to M6713 DSP and click "display parameters" button for verification. The dialog box on the right hand side is used to display information that includes: the response and status of the DSP, the alert information, the downloaded parameters, the size of log file used to store the bulk data from M6713 DSP and so on. The "run" button is disabled as default and enabled by clicking the "download parameters" button. Once the updated parameters are received by the DSP, we can click the "run" button to execute the planned manipulators trajectory. All the aforementioned functions require the communication between the host PC and the DSP board. The M6713 has a bi-directional data channel configured such that it could be used for communication. The protocol on this channel supports sending special data blocks consisting of a small header plus an arbitrarily large data packet. The header contains two words of information designated the PeripheralId and the PacketSize. The PeripheralId is an arbitrarily large tag value stored into the header by the sender intended to allow the receiver to uniquely identify the purpose and content of the packet. The PacketSize field is automatically updated by the driver to accurately reflect the size of the data portion of the packet.

### 2.2.2 Target DSP Servo Control Program

The target DSP project is developed using the Innovative Integration Pismo Toolset. The CCstudio integrated development environment (IDE) suite consists of the TI Optimizing C++ Compiler, Assembler and Linker, the Code Composer debugger and code authoring environment. With the help of these tools it is very convenient to develop user's own target applications.

The TI DSP/BIOS operating system provides a multitasking working environment. The codes are provided in the Appendix A. The DSP program is divided in three parts. The program starts from a main thread. Figure (2.4) shows the flow chart of the main thread. Once the program is loaded from the host, the main thread begins to receive the messages from the host and by examining its head information, the DSP decides what kind of task is needed to be performed. If the "download parameters" button is clicked, the "ccDownload" message will be sent to the DSP so that the DSP program would update the values of the parameters. If the "run" button is clicked, the "ccRun" message is received and the control thread is then activated. Once the control thread is activated, it will start another thread called "servo control". The Pismo library supports the servo by defining a class called "servoBase" to perform the basics of a servo operation. The servo operation acquires the digital inputs from the optical encoders, performs the algorithm to produce an output data event, which is then sent to D/A convertors as control signal. The basic servo class handles the details of attaching the interrupt, setting up the hardware, reading the data from the hardware and delivering it to the user defined function for processing. After this event, it would then write the

modified data to the analog outputs. Figure (2.5) is the flow chart of the control thread. In order to use the servo base provided by Pismo, some initialization needs to be done. The first task is to choose the timer and setup the clock rate. The analog is setup by the servo base to drive an interrupt according the clock rate of the timer. Every time an interrupt occurs, the user defined internal function in the servo thread is executed. Second task is to enable the analog output channels and setup the DAC (digital to analog conversion) delay. After these two steps, the program initializes the variables and applies enters the data into the memory. Once the initialization task is performed, the servo thread is started and the user defined function is executed every time that the interrupt occurs. In the user defined function, the current displacement information is read from the digital inputs sent by the feedback sensor components which are optical encoders. The velocity is then calculated by taking the derivative of the displacement. The displacement and velocity errors, which are the inputs to the PD control law, are calculated and recorded. The feedback and feed-forward control signal are then added up and written to the DAC memory to update the control signal. As soon as the endpoint is reached, the servo thread is ended and the program goes back to control thread. The recorded error information is then sent to the host PC for analysis.

### 2.2.3 Read Feedback From Optical Encoder

Reading the digital inputs from the encoders is critically important because the control signal is updated according to the generated information. There

are two ways to decode the signals from the two channels of an incremental optical encoder. One is to use the hardware. Xilinx Spartan3 FPGA (field programmable gate array) is one of the many features of M6713. A counter can be constructed by FPGA to keep track of the displacement and used as the interrupt source for the user defined control signal updating function. The other method is to write a piece of code to decode the feedback signal. This method is slow compared with the first one. However, as the frequency of interrupt in our application is not extremely high, it serves our goal very well. The only problem with this method is that we have to be careful since if the speed of the dynamic system is too high, then we might lose one or two impulses from two channels of the encoder. For example, consider the situation in which the encoder is feeding back the angular displacement of a rotating shaft. The angular speed of the shaft is $n = 60rpm$. The resolution of the encoder is $CPR = 1024$. So the frequency of the pulse in the signal from channel A (Figure (2.2)) is $f_1 = \frac{n}{60} \cdot CPR = 1024Hz$. So the period is $T_1 = \frac{1}{1024} \approx 1ms$. This means that the maximum time period between two consecutive DI readings (name it $T_2$) cannot be greater than $1ms$. If we are using codes to decode signal we have to make sure those codes are called within $1ms$ from the time that they were called. There are some factors that will affect $T_2$. First one is the rotational speed. If the speed is high then the $T_2$ has to be small. Secondly, as the program is multi-threaded, if another thread with higher priority becomes active, the thread which contains the codes to read DI is then block, then the period $T_2$ become uncertain. In our case the speed is low and the codes need to be executed between two DI reads are small, therefore the time $T_2$ will not be too small.

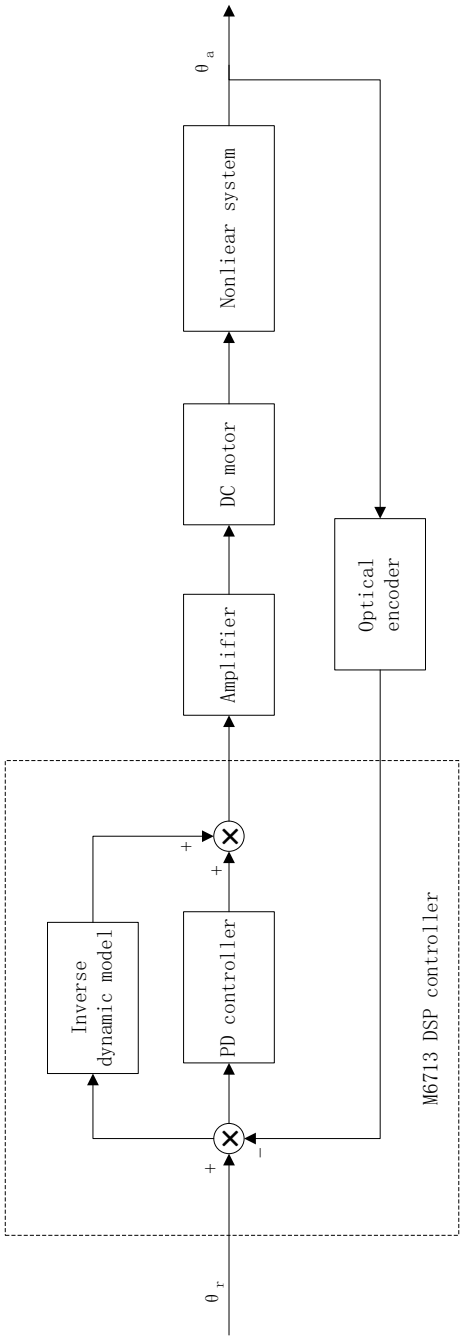


Figure 2.1: Block diagram of the control system scheme.

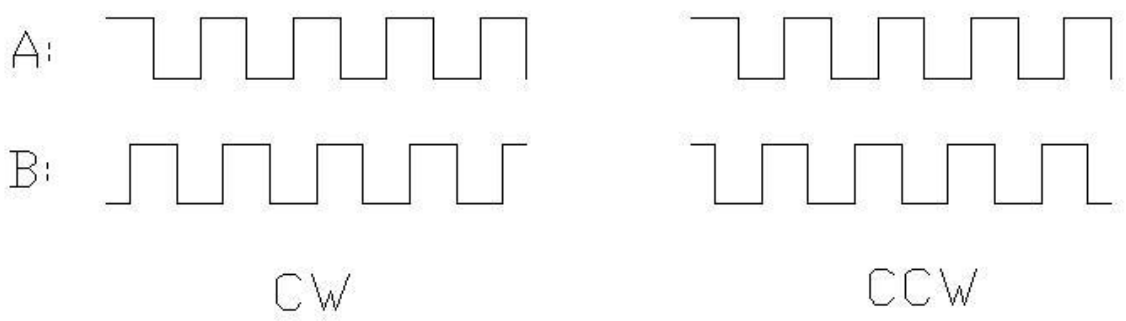


Figure 2.2: Output waveform of two channels of optical encoders.

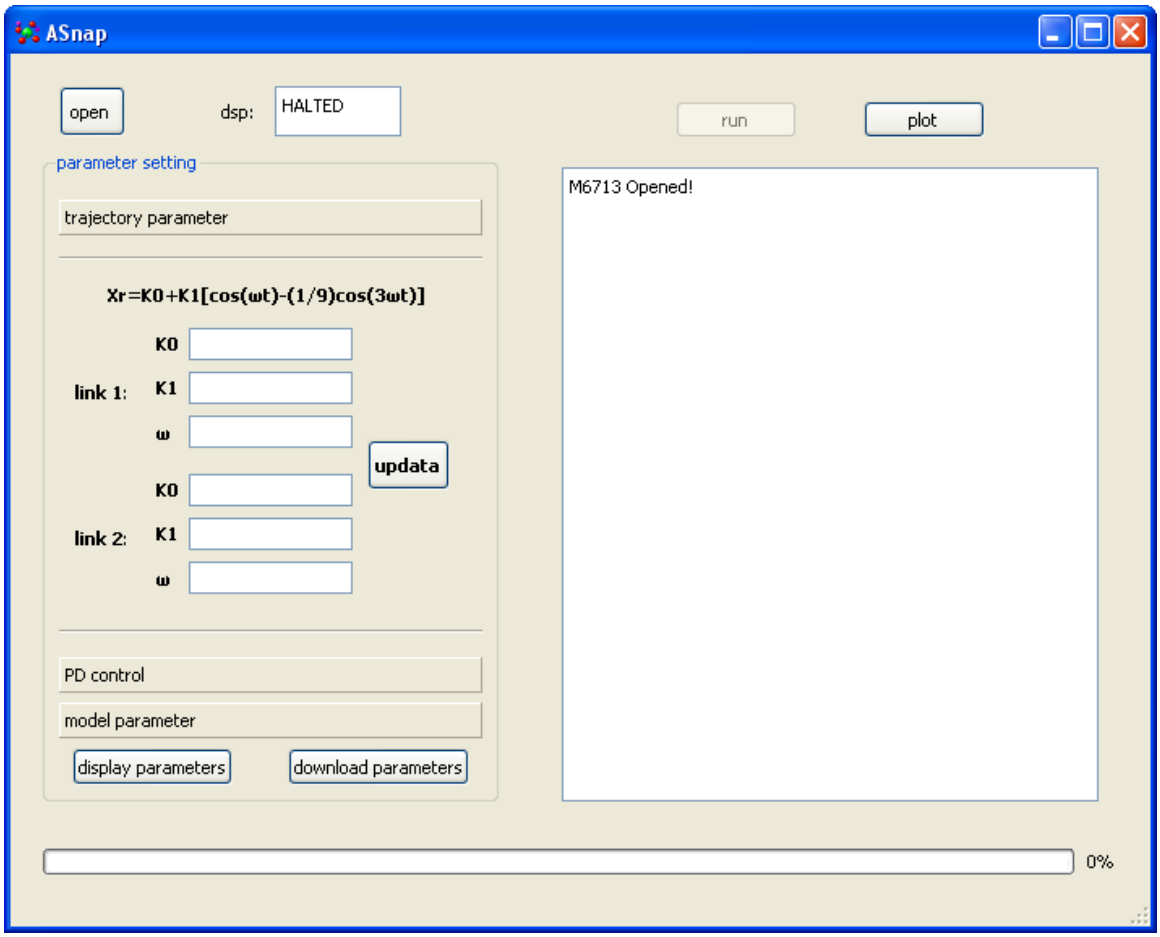


Figure 2.3: GUI host program interface.

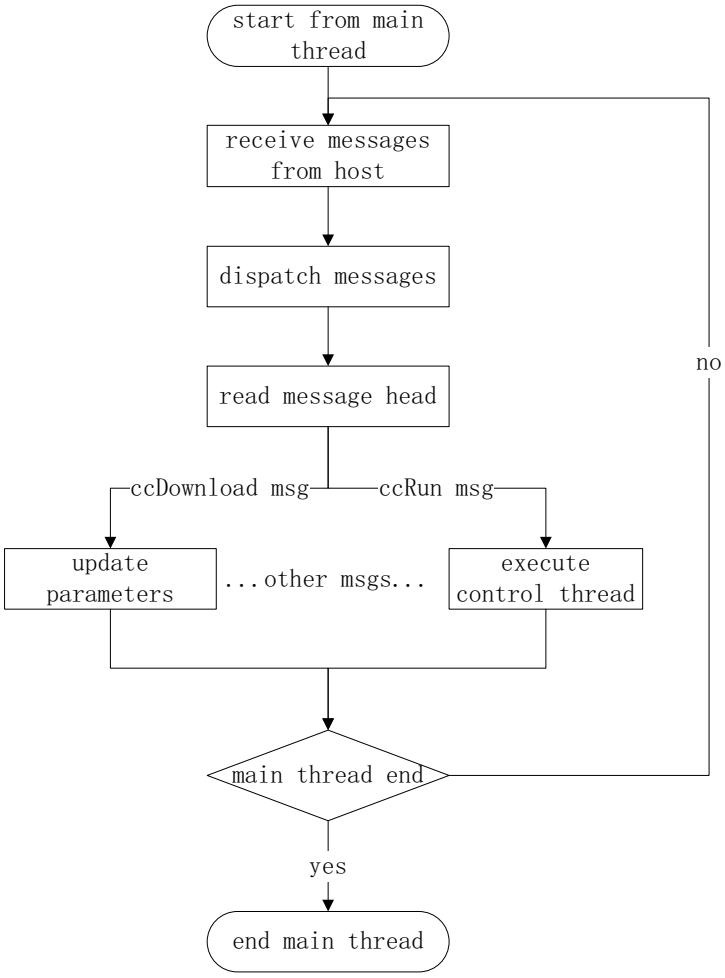


Figure 2.4: Main thread flowchart.

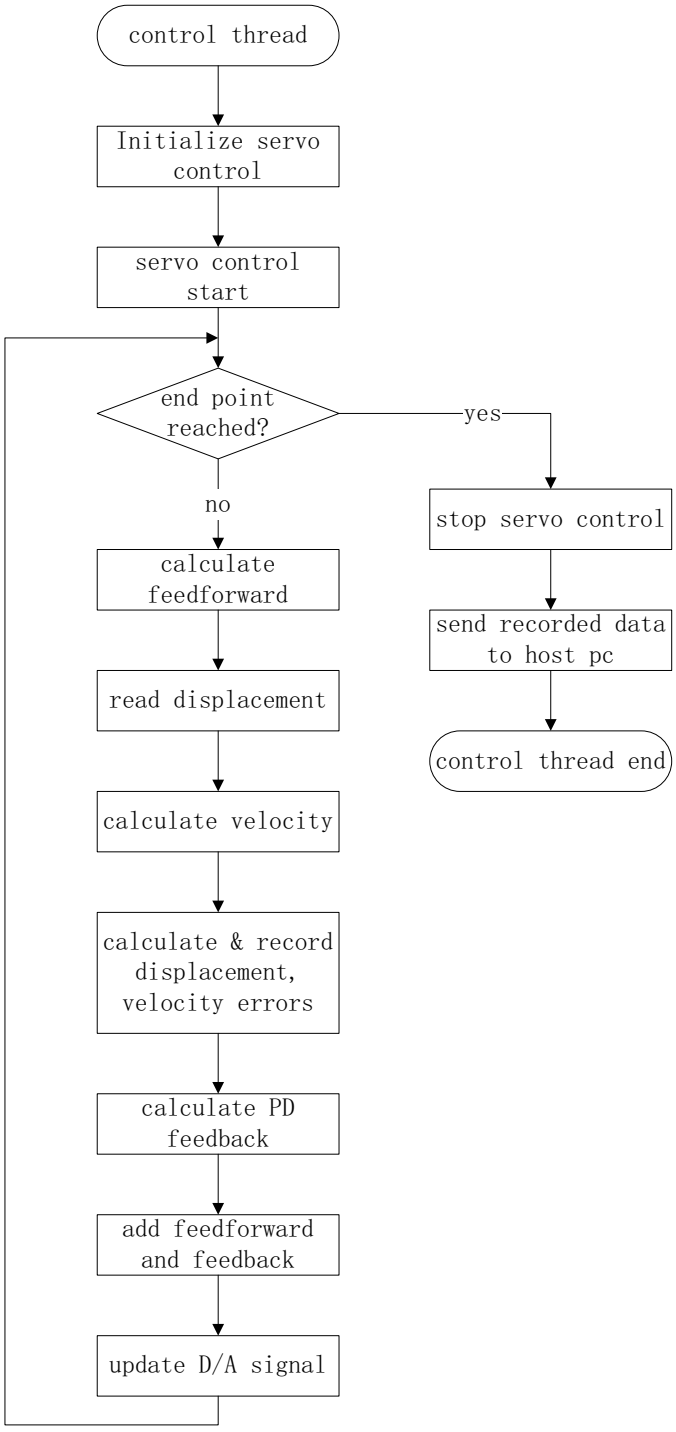


Figure 2.5: Control thread flowchart.

# Chapter 3

# Experimental Verification on a Two Degrees of Freedom Closed Loop Planar Robot Manipulator

## 3.1 Introduction

In this chapter the TPM Based Parameter Identification method is verified on a two degrees of freedom closed loop planar robot manipulator. This robot manipulator is a highly nonlinear dynamic system. The dynamic model of the system is derived. The point-to-point motion trajectories are synthesized and put into the inverse dynamics to get the FeedForward signal. Following the parameter identification procedure introduced in the first chapter, the estimated parameters are updated. The TPM based method is proved to be very effective on the nonlinear system.

## 3.2 Dynamic Model of the Two DOF Closed Loop Planar Robot Manipulator

In this section, the dynamic model of the robot manipulator is derived from Lagrange method. The manipulator is shown in Figure (3.1). In this figure, $\theta_1$ and $\theta_2$ are the angular displacements of joint 1 and joint 2 respectively. The actuating torques $\tau_1$ and $\tau_2$ are applied at the two joints. It is a fully controlled system. The dimensions of the closed loop are known. Therefore, the angles $\theta_3$ and $\theta_4$ can be expressed as functions of $\theta_2$. The model parameters $ms$, $Is$, $rs$ and $ls$ represent masses, moments of inertia, distances from each joint to their mass centers and lengths of the links.

Angular displacements $\theta_1$ and $\theta_2$ are chosen as the generalized coordinates of the 2DOF system. Torques applied at joint 1 and joint 2 are the generalized forces. As it is a planar manipulator and the links are rigid, there is no potential energy in the system. The kinetic energy of the manipulator can be expressed as

$$T = T_{12} + T_3 + T_4 + T_{56} \tag{3.1}$$

where:

$$T_{12} = \frac{1}{2} m_{12} (r_1 \dot{\theta}_1)^2 + \frac{1}{2} I_{12} \dot{\theta}_1^2$$

$$T_3 = \frac{1}{2} m_3 v_3^2 + \frac{1}{2} I_3 (\dot{\theta}_1 + \dot{\theta}_4)^2$$

$$T_4 = \frac{1}{2} m_4 v_4^2 + \frac{1}{2} I_4 (\dot{\theta}_1 + \dot{\theta}_2)^2$$

$$T_{56} = \frac{1}{2} m_{56} v_{56}^2 + \frac{1}{2} I_{56} (\dot{\theta}_1 + \dot{\theta}_3)^2$$

$$v_3^2 = (l_1 + l_2)^2 \dot{\theta}_1^2 + r_3^2 (\dot{\theta}_1 + \dot{\theta}_4)^2 + 2(l_1 + l_2) r_3 \cos\theta_4 \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_4)$$

$$v_4^2 = l_1^2 \dot{\theta}_1^2 + r_4^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 + 2 l_1 r_4 \cos\theta_2 \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2)$$

$$\begin{aligned} v_{56}^2 = {} & l_1^2 \dot{\theta}_1^2 + l_4^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 + r_5^2 (\dot{\theta}_1 + \dot{\theta}_3)^2 + 2 l_1 l_4 \cos\theta_2 \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2) \\ & + 2 l_1 r_5 \cos\theta_3 \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_3) + 2 l_4 r_5 \cos(\theta_2 - \theta_3)(\dot{\theta}_1 + \dot{\theta}_2)(\dot{\theta}_1 + \dot{\theta}_3) \end{aligned}$$

The angles $\theta_3$ and $\theta_4$ are functions of $\theta_2$

$$\theta_3 = \arctan\left(\frac{-l_4 \sin\theta_2}{l_2 - l_4 \cos\theta_2}\right) + \arccos\left(\frac{l_4^2 + l_2^2 - l_3^2 + l_5^2 - 2 l_2 l_4 \cos\theta_2}{2 l_5 \sqrt{l_4^2 + l_2^2 - 2 l_2 l_4 \cos\theta_2}}\right)$$

$$\theta_4 = \arctan\left(\frac{-l_4 \sin\theta_2}{l_2 - l_4 \cos\theta_2}\right) + \arccos\left(\frac{l_4^2 + l_2^2 + l_3^2 - l_5^2 - 2 l_2 l_4 \cos\theta_2}{-2 l_3 \sqrt{l_4^2 + l_2^2 - 2 l_2 l_4 \cos\theta_2}}\right)$$

$$\dot{\theta}_3 = \frac{l_4 \sin(\theta_4 - \theta_2)}{l_5 \sin(\theta_3 - \theta_4)} \dot{\theta}_2$$

$$\dot{\theta}_4 = \frac{l_4 \sin(\theta_3 - \theta_2)}{l_3 \sin(\theta_3 - \theta_4)} \dot{\theta}_2$$

The total kinetic energy can be simplified as

$$T = f_1(\theta_2) \dot{\theta}_1^2 + f_2(\theta_2) \dot{\theta}_2^2 + f_3(\theta_2) \dot{\theta}_1 \dot{\theta}_2 \tag{3.2}$$

where:

$$f_1(\theta_2) = \frac{A_1}{2} + A_2 \cos\theta_2 + A_3 \cos\theta_3 + A_4 \cos\theta_4 + A_5 \cos(\theta_2 - \theta_3)$$

$$f_2(\theta_2) = \frac{A_6}{2} + \frac{A_7}{2}[\beta\frac{\sin(\theta_4-\theta_2)}{\sin(\theta_3-\theta_4)}]^2 + \frac{A_8}{2}[\alpha\frac{\sin(\theta_3-\theta_2)}{\sin(\theta_3-\theta_4)}]^2 + A_5\beta\cos(\theta_2-\theta_3)\frac{\sin(\theta_4-\theta_2)}{\sin(\theta_3-\theta_4)}$$

$$f_3(\theta_2) = A_6 + A_2\cos\theta_2 + A_5\cos(\theta_2-\theta_3) + [A_7 + A_3\cos\theta_3 + A_5\cos(\theta_2-\theta_3)]\beta\frac{\sin(\theta_4-\theta_2)}{\sin(\theta_3-\theta_4)} + (A_8 + A_4\cos\theta_4)\alpha\frac{\sin(\theta_3-\theta_2)}{\sin(\theta_3-\theta_4)}$$

where:

$$A_1 = m_{12}r_1^2 + I_{12} + m_3(l_1+l_2)^2 + m_3r_3^2 + I_3 + m_4(l_1^2+r_4^2) + I_4 + m_{56}(l_1^2+l_4^2+r_5^2) + I_{56}$$

$$A_2 = m_4l_1r_4 + m_{56}l_1l_4$$

$$A_3 = m_{56}l_1r_5$$

$$A_4 = m_3(l_1+l_2)r_3$$

$$A_5 = m_{56}l_4r_5$$

$$A_6 = m_4r_4^2 + I_4 + m_{56}l_4^2$$

$$A_7 = m_{56}r_5^2 + I_{56}$$

$$A_8 = m_3r_3^2 + I_3$$

and

$$\alpha = \frac{l_4}{l_3}$$

$$\beta = \frac{l_4}{l_5}$$

Apply the Lagrange equations

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{\theta}_1}) - \frac{\partial T}{\partial \theta_1} = \tau_1$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{\theta}_2}) - \frac{\partial T}{\partial \theta_2} = \tau_2$$

We get the dynamic equations as

$$2f_1(\theta_2)\ddot{\theta}_1 + f_3(\theta_2)\ddot{\theta}_2 + 2\frac{df_1(\theta_2)}{d\theta_2}\dot{\theta}_1\dot{\theta}_2 + \frac{df_3(\theta_2)}{d\theta_2}\dot{\theta}_2^2 = \tau_1 \tag{3.3}$$

$$f_3(\theta_2)\ddot{\theta}_1 + 2f_2(\theta_2)\ddot{\theta}_2 - \frac{df_1(\theta_2)}{d\theta_2}\dot{\theta}_1^2 + \frac{df_2(\theta_2)}{d\theta_2}\dot{\theta}_2^2 = \tau_2 \tag{3.4}$$

From the above equations we can see that this 2DOF planar closed loop manipulator is a highly nonlinear mechanical system. The closed loop kinematics chain increases the nonlinearity greatly. The point-to-point trajectories introduced in chapter 1 (equation (1.3)) are put into the equations to get the inverse dynamic model. The frequencies of the fundamental harmonics of the joint trajectories may be different from joint 1 to joint 2. However, without loss of generality and to keep the number of harmonic components in the actuating toruqes low, we use the same fundamental frequency for both joint trajectory.

## 3.3 Approximating the Inverse Dynamic Model

If we apply the excitation forces to the above manipulator's dynamic equations, it is very difficult to get the analytical solution of the angular displacements of $\theta_1$ and $\theta_2$. It is pointed out in the introduction chapter that the TPM can transform the problem into algebraic form which is relatively easy to solve. Synthesize the point-to-point trajectories to be

$$\theta_{r1} = \lambda_1[\cos(\omega t) - \frac{\cos(3\omega t)}{9} - \frac{8}{9}] \tag{3.5}$$

$$\theta_{r2} = k_0 + \lambda_2[\cos(\omega t) - \frac{\cos(3\omega t)}{9}] \tag{3.6}$$

The motion starts at time $t = 0$ and ends at $t = \frac{\pi}{\omega}$. Put the trajectories (equation (3.5) and equation (3.6)) into equations (3.3) and (3.4) to get the inverse dynamic model. The inverse dynamic forces contain very high order harmonic components of the input trajectories. Therefore, the FeedForward signal is very hard to generated directly from the above equations. We use $n + 1$ Taylor Series terms to approximate the $f_1(\theta_2)$, $f_2(\theta_2)$, $f_3(\theta_2)$, $\frac{df_1(\theta_2)}{d\theta_2}$, $\frac{df_2(\theta_2)}{d\theta_2}$ and $\frac{df_3(\theta_2)}{d\theta_2}$. The resulting actuating forces contain $(3n + 5)th$ order of harmonics and can be expressed as

$$\tau_1 = \sum_{i=0}^{3n+6} a_{1,i} \cos(i\omega t) \tag{3.7}$$

$$\tau_2 = \sum_{j=0}^{3n+6} a_{2,j} \cos(j\omega t) \tag{3.8}$$

where $a_{1,i}$ and $a_{2,j}$ are the corresponding harmonic amplitudes and are functions of the dynamic, kinematic parameters of the robot manipulator as well as the trajectory parameters.

The number of Taylor Series terms $n+1$ used to accurately describe the FeedForward torques depends on the trajectories as well as the estimated parameters. For example, let us consider the following trajectories

$$\theta_{r1} = 0.23[\cos(0.5t) - \frac{\cos(1.5t)}{9} - \frac{8}{9}]$$

$$\theta_{r2} = -0.06 + 0.12[\cos(0.5t) - \frac{\cos(1.5t)}{9}]$$

The unknown model parameters are estimated to make all the $A_i$s in equation (3.7) and equation (3.8) to be 1. Figure (3.2) shows the accurate torques with the above trajectory parameters at joint 1 and joint 2. Figure (3.3) presents the approximated torques and errors at joint 1 with $n = 2$ and $n = 3$ respectively. Figure (3.4) contains the approximated torques and errors at joint 2. From these figures we can see that if we want the error of approximated torque to be within 10% of the accurate value, the approximated $\tau_1$ calculated from equation (3.7) with $n = 2$ is good enough. However, we have to set $n = 3$ in equation (3.8) to approximate $\tau_2$. In short, if we choose small trajectories and the values of the estimated parameters are not too big, 3 terms of Taylor series expansion may be good enough to approximate the torques. However, if

the estimated parameters and the trajectories are not small enough, we may need more terms to approximate the actuating torques.

## 3.4 Model Parameter Identification Process

Once we calculate the FeedForward model, following the control scheme introduced in Chapter 2 (Figure (2.1)), we can build the control system. The feedback commands applied at joint 1 and joint 2 are expressed as

$$C_1(e_1, \dot{e}_1) = k_{p1}e_1 + k_{d1}\dot{e}_1 \tag{3.9}$$

$$C_2(e_2, \dot{e}_2) = k_{p2}e_2 + k_{d2}\dot{e}_2 \tag{3.10}$$

From the TPM based model parameter estimation method(Rastegar and Feng) introduced in chapter 1, we know that if the errors in actual displacements and velocities are small enough, we can ignore those high order terms of $e_1$, $\dot{e}_1$, $e_2$ and $\dot{e}_2$. We get the following equations

$$-C_1(e_1, \dot{e}_1) = \sum_{i=0}^{3n+6} a_{1,i}(\Delta P_1, \Delta P_2, \Delta P_3 \cdots) \cos(i\omega t) \tag{3.11}$$

$$-C_2(e_2, \dot{e}_2) = \sum_{j=0}^{3n+6} a_{2,j}(\Delta P_1, \Delta P_2, \Delta P_3 \cdots) \cos(j\omega t) \tag{3.12}$$

where $\Delta P_i$s are the deviation in the estimated parameters.

The TPM based model parameter identification method is proved to be valid for updating those parameters with respect to which the system is linear. In other words, in order to maintain the linear relationship between the deviation of the estimated parameters $\Delta P_i$s and the amplitudes of harmonic components of the feedback signals in the above equations (3.11) and (3.12), we have to choose the estimated parameters that appear in the inverse dynamic equations as coefficient of terms. For example, we cannot use the parameter identification method to update the parameter $r_5$ directly. However, since $m_5 r_5$ and $m_5 r_5^2$ appear as the coefficient of nonlinear terms in the inverse dynamics, we can use the parameter identification method to get these two "combined parameters" and then calculate $m_5$ and $r_5$. In this chapter we choose the coefficients in the inverse dynamic model which are functions of the dynamic, kinematic paramter of the manipulator as the unknown parameters. Later on these parameters will be referred to as "combined parameters". We use the TPM based parameter identification method to update the values of the combined parameter and then calculate the manipulator's parameters.

Since the amplitudes of harmonic components of the feedback signals maintain a linear relationship with the deviation of estimated parameter $\Delta P_i$s. We can write equations (3.11) and (3.12) into a matrix form

$$E = M \times P \tag{3.13}$$

where $E = [E_1, E_2, \cdots, E_m]$ are the amplitudes of harmonic components of the feedbacks and can be obtained from the experiments using Fourier integration;

$P = [\Delta P_1, \Delta P_2, \cdots, \Delta P_n]$ are the deviations of the estimated parameters. The matrix $M_{m\times n}$, which is a function of trajectory parameters and can be obtained analytically, needs not to be squared. In fact, the size of the matrix $M$ is chosen to keep those harmonic terms with large amplitudes. We name the two terms respectively from the Fourier expansion of $C_1(e_1, \dot{e}_1)$ and $C_2(e_2, \dot{e}_2)$ with the largest amplitudes as the dominant terms. Any other term from $C_1$ or $C_2$ with the amplitude less than 1% of the dominant terms can be ignored. Even if we ignore those small harmonics, the number of the equations is still larger than the number of the unknown parameters. A least-squares method is adopted to solve this problem. We use the optimization method to minimize the following object function.

$$F = \sum_{i=1}^{n_h} (\sum_{j=1}^{n} m_{ij}\Delta P_j - E_i)^2 \tag{3.14}$$

Here $n_h$ is the number of harmonics with large amplitudes we kept for analysis and $m_{ij}$ are the elements in mapping matrix $M$.

The difficulty of solving this kind of least-squares problem lies in the system itself. If the mapping matrix $M_{m\times n}$ is ill-conditioned or singular, it is very difficult to solve. And the $M$ is determined by the very nature of the system. In our case, the rank of $M$ is no larger than 6. This means there are only 6 estimated parameters that are linearly independent of each other. If there are more than 6 parameters needed to be updated, we may have to include more information about these parameters, such as the regularity properties and the relationship between some of the parameters. Otherwise the problem will not

be solved satisfactorily. We choose the following 6 combined parameters.

$$P_1 = m_{12}r_1^2 + I_{12}$$

$$P_2 = I_3$$

$$P_3 = I_4$$

$$P_4 = m_{56}$$

$$P_5 = m_{56}r_5$$

$$P_6 = m_{56}r_5^2 + I_{56}$$

Therefore, these 6 model parameters in figure (3.1): $I'_{12} = m_{12}r_1^2 + I_{12}$ (moment of inertia of link 1 about joint 1), $I_3$ (moment of inertia of link 3 about its mass center), $I_4$ (moment of inertia of link 4 about its mass center), $m_{56}$ (mass of link 56), $r_5$ ( distance from mass center of link 56 to the joint) and $I_{56}$ (moment of inertia of link 56) can be calculated from the above combined parameter $P_i$s. In the experiment, the parameters are updated by adding the resulting corrections $\Delta P_i$s to their current values after each step. The entire updating process is repeated until the estimated parameters reach the satisfactory accuracy.

## 3.5 Experimental Results and Analysis

Table (3.1) contains the experimental results. The initial estimated values of the combined parameters are selected to be $[P]_0 = [1.000, 1.000, 1.000, 1.000,$

| Step No. | trajectory | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ |
|---|---|---|---|---|---|---|---|
| 0 | | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 1 | $\lambda_0 = 0.2$<br>$k_0 = 0$<br>$\lambda_1 = 0.196$<br>$\omega = 0.5$ | 0.2418 | 0.1534 | 0.1509 | 0.6651 | 0.2724 | 0.1866 |
| 2 | $\lambda_0 = 0.295$<br>$k_0 = -0.262$<br>$\lambda_1 = 0.295$<br>$\omega = 1$ | 0.1187 | 0.006023 | 0.01119 | 0.6106 | 0.1546 | 0.05058 |
| 3 | $\lambda_0 = 0.3$<br>$k_0 = -0.262$<br>$\lambda_1 = 0.295$<br>$\omega = 2$ | 0.1080 | 0.005422 | 0.01029 | 0.6059 | 0.1442 | 0.04350 |

Table 3.1: Updated combined parameters after each step

$1.000, 1.000]$. After three steps the parameters are updated to be $[P]_3 = [0.1080, 0.005422, 0.01029,\ 0.6059, 0.1442, 0.04350]$, and the model parameters are calculated to be $I'_{12} = 1.080 \times 10^{-1} kg\cdot m^2$, $I_3 = 5.422 \times 10^{-3} kg\cdot m^2$, $I_4 = 1.029^{-2} kg\cdot m^2$, $m_{56} = 6.059 \times 10^{-1} kg$, $r_5 = 2.380 \times 10^{-1} m$ and $I_{56} = 9.179 \times 10^{-3} kg\cdot m^2$.

From table (3.1) we can see that the same trajectory can only update the parameters once. In order to update the estimated parameters further, we have to enlarge the trajectory after each step. Ideally each trajectory can be used several times to update the parameters. However, with the physical limitation of the system and other reasons that will be discussed in the next section, we use each trajectory once and end the update process after the third step. The biggest improvement of all the parameters in the last step is within 14% of its previous value.

Initially, the estimated parameters are far from their nominated values, so

we have to choose small trajectories to keep the errors small. However, it is found in the experiment that it is very difficult to reduce the velocity errors when the joints move very slowly. As a result, we cannot select too small trajectories. The feedback control gains are turned after several trial and error runs. With the relative large feedback gains, the system can be stabilized and keep the output errors very small even if the estimated parameters are far from their nominal values. After we update the parameters the error will be reduced and become too small to detect, so we have to enlarge the trajectories in the following steps.

From Figure (3.5) to (3.8), the trajectories in step 1 are used. Figure (3.5) and (3.6) represent the angular displacements, velocities of the two joints and the harmonic contents of the feedback $C_1$, $C_2$ applied at both joints with the initial estimated parameters shown in Table (3.1). Figure (3.7) and (3.8) shows the motions of joint 1 and 2 with the updated parameter values after step 1. It is shown that using the updated parameters, the errors are reduced. However, the output response in Figure (3.7) and (3.8) could not further update the parameters. From Figure (3.9) to (3.12), the trajectories in step 2 are used. Figure (3.9) and (3.10) showns the outputs before the improvement; Figure (3.11) and (3.12) showns the outputs after the improvement. From Figure (3.13) to (3.16), the trajectories in step 3 are used. The outputs before and after parameter improvements in step 3 are shown.

## 3.6 Results Evaluation

In order to show that the estimated parameters are getting closer to their nominal values after each updating step, we check the errors in the system output and calculate the integral performance indices. As the parameters approach the nominal values, the model based feedforward controller plus the feedback controller control the manipulator system better and the errors will be reduced.

Two of the most common indices used to measure the overall performance of a control system are the integral of error and integral of squared error. Since the sign of the error changes, we use the integral of squared error.

$$I = \int_0^{\frac{\pi}{\omega}} e^2(t)dt \tag{3.15}$$

The integral of squared displacement error in joint 1 ($I_{d1}$) and joint 2 ($I_{d2}$) as well as the integral of squared velocity error in both joints ($I_{v1}$ and $I_{v2}$) are all calculated. In each updating step, for the same set of trajectories, we use the estimated parameters before and after this step and then compare the two outputs (see Figure (3.5) to Figure (3.16)). By showing in Table 3.2 that using updated parameter values, the error integrals are smaller than before, we know that the parameters are better estimated.

## 3.7 Discussion and Conclusions

In this chapter, the TPM based nonlinear system model parameter estimation is verified on a two degrees of freedom closed loop planar robot manipula-

| trajectory paras | used in step 1 | | used in step 2 | | used in step 3 | |
|---|---|---|---|---|---|---|
| estimated paras | before updated | after updated | before updated | after updated | before updated | after updated |
| $I_{d1}$ | $7.34 \times 10^{-4}$ | $1.89 \times 10^{-4}$ | $2.58 \times 10^{-4}$ | $7.73 \times 10^{-5}$ | $4.94 \times 10^{-4}$ | $2.14 \times 10^{-4}$ |
| $I_{v1}$ | $1.99 \times 10^{-3}$ | $1.81 \times 10^{-3}$ | $7.98 \times 10^{-3}$ | $6.76 \times 10^{-3}$ | $4.25 \times 10^{-2}$ | $2.54 \times 10^{-2}$ |
| $I_{d2}$ | $2.53 \times 10^{-3}$ | $2.08 \times 10^{-4}$ | $6.40 \times 10^{-4}$ | $1.04 \times 10^{-4}$ | $4.01 \times 10^{-4}$ | $1.99 \times 10^{-4}$ |
| $I_{v2}$ | $4.18 \times 10^{-3}$ | $1.17 \times 10^{-3}$ | $1.06 \times 10^{-2}$ | $7.05 \times 10^{-3}$ | $2.32 \times 10^{-2}$ | $1.66 \times 10^{-2}$ |

Table 3.2: Squared error integrals

tor. The results presented in the chapter shows that the developed method is very effective on a highly nonlinear system. It only takes three steps to reach a very good estimation. In this case when the improvements are within 14% of the previous values, the updating procedure ends. However, if the application requires a higher speed and precision, one can keep the estimate iterations until the desired accuracy or some other requirements are attained.

If we want to further improve the accuracy, there are other factors that need to be taken into consideration. First, there is model inaccuracy. For example, we assume this is a planar manipulator. However, it is found out in the experiment that the desk where the manipulator is mounted is a little tilted. A more accurate model should take some gravitational forces into consideration. Second, the optical encoders have limited resolutions. As the accuracy of the estimated parameters reaches certain extent, errors related to the instruments will affect our analysis. Third, in order to further improve the parameters, we need to enlarge the trajectories. Larger trajectory means larger torques applied at the joint. The limit of the output torque of the big motor is set to $2Nm$ and the small motor is set to $1Nm$ in order to keep a linear relationship between the control signals and the motor torques. If we need to further increase the trajectory, we may exceed the limitation of the

motors.

It is proved that this method makes use of slow and small trajectories to provide the system stability when the estimated parameters are not accurately known and faster and larger motions are synthesized to better estimate the parameters. This method is shown to be convergent and very effective.

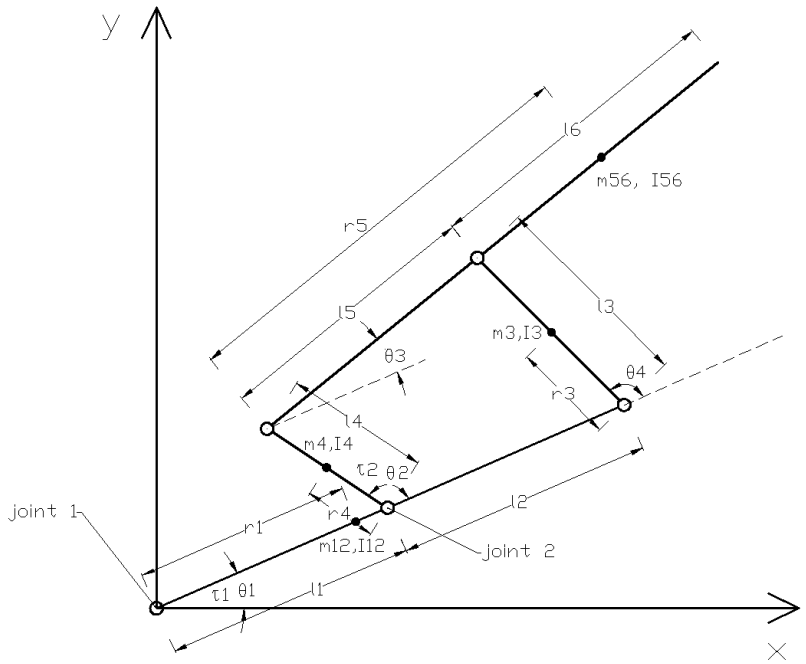


Figure 3.1: Two DOF closed loop planar manipulator

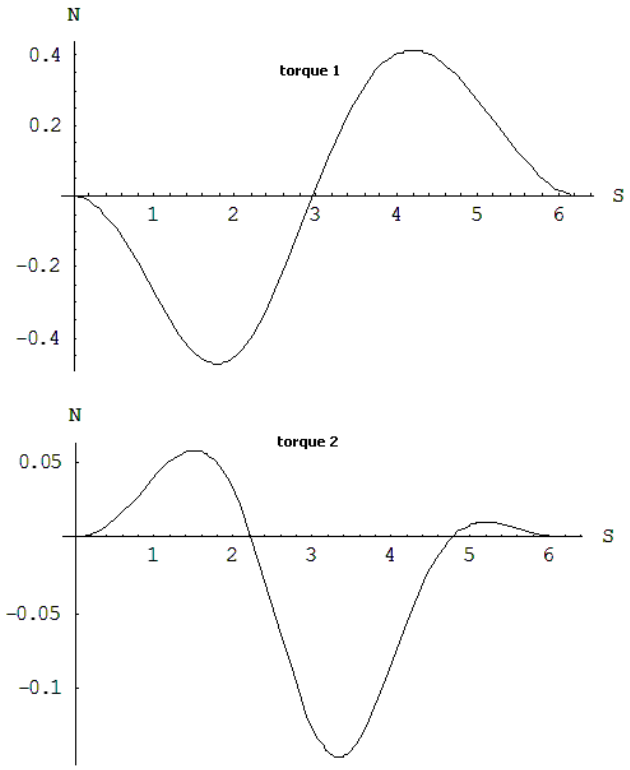


Figure 3.2: Accurate torques at joint 1 and 2 from the example in section 3.3

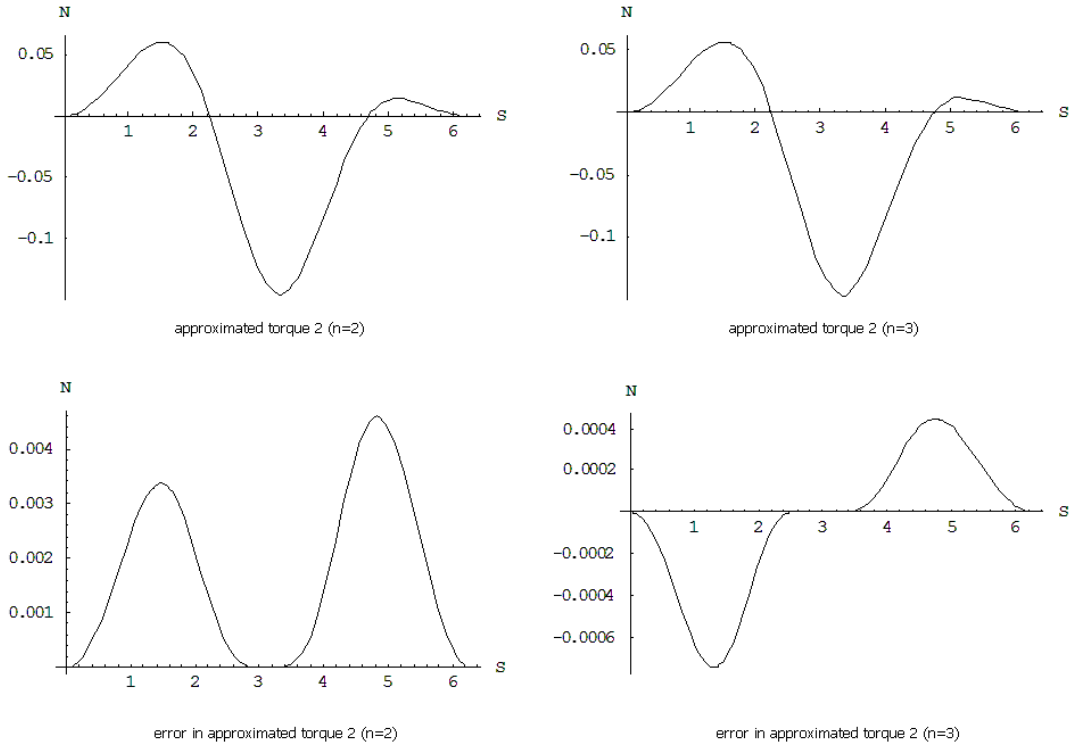


Figure 3.3: Approximated torques and errors at joint 1 from the example in section 3.3 with $n = 2$ and $n = 3$ respectively

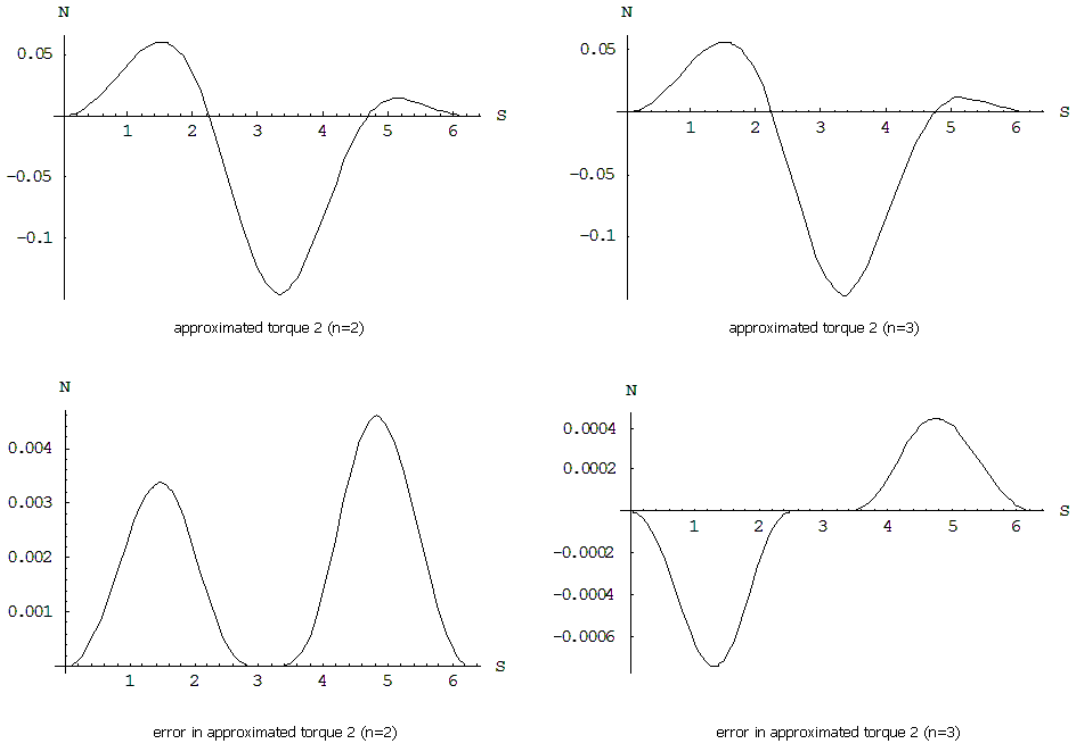


Figure 3.4: Approximated torques and errors at joint 2 from the example in section 3.3 with $n = 2$ and $n = 3$ respectively

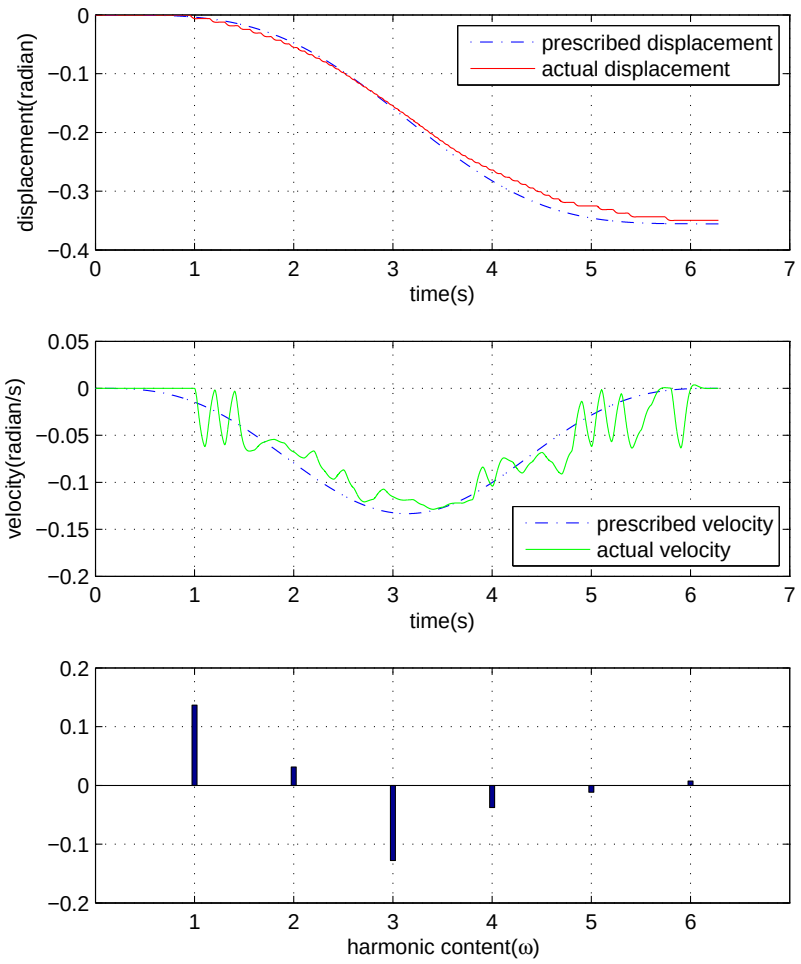


Figure 3.5: Joint motion $\theta_1$ in step 1 with the initially estimated parameters and the harmonic content of feedback applied at joint 1

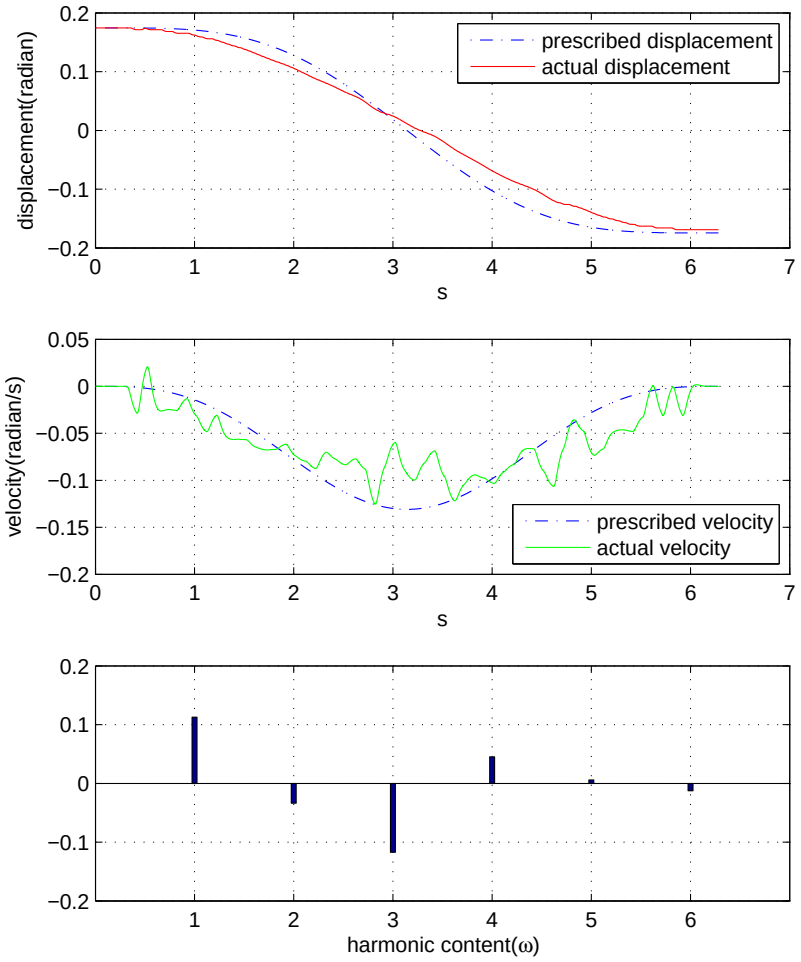


Figure 3.6: Joint motion $\theta_2$ in step 1 with the initially estimated parameters and the harmonic content of feedback applied at joint 2

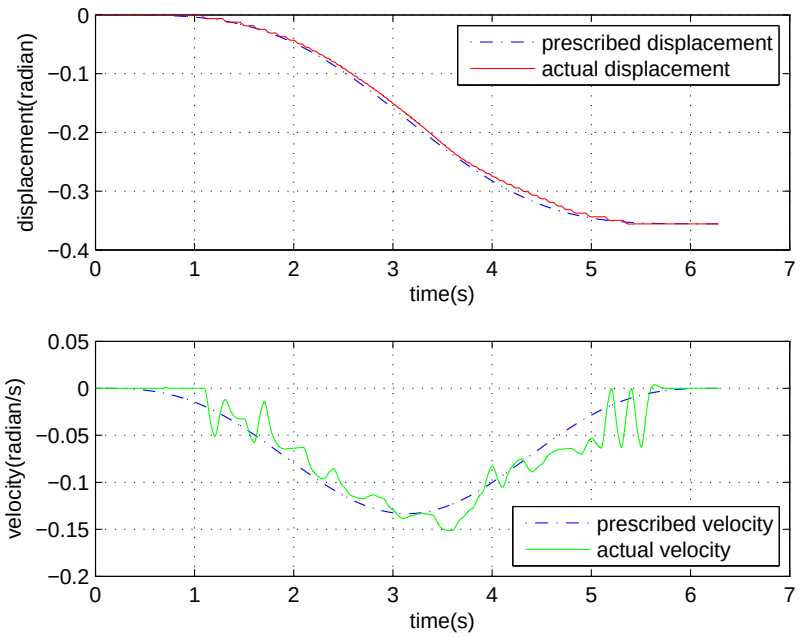


Figure 3.7: Joint motion of $\theta_1$ in step 1 with the updated parameters obtained after step 1

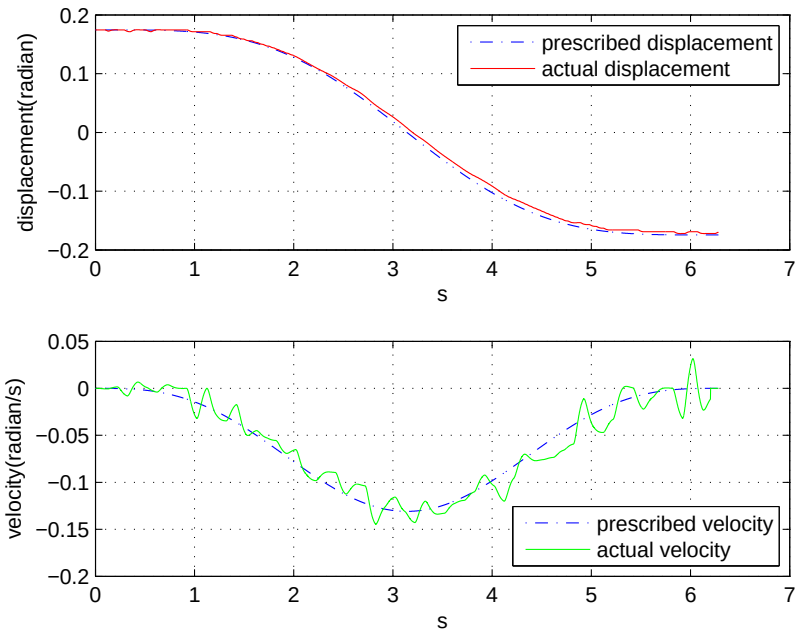


Figure 3.8: Joint motion of $\theta_2$ in step 1 with the updated parameters obtained after step 1

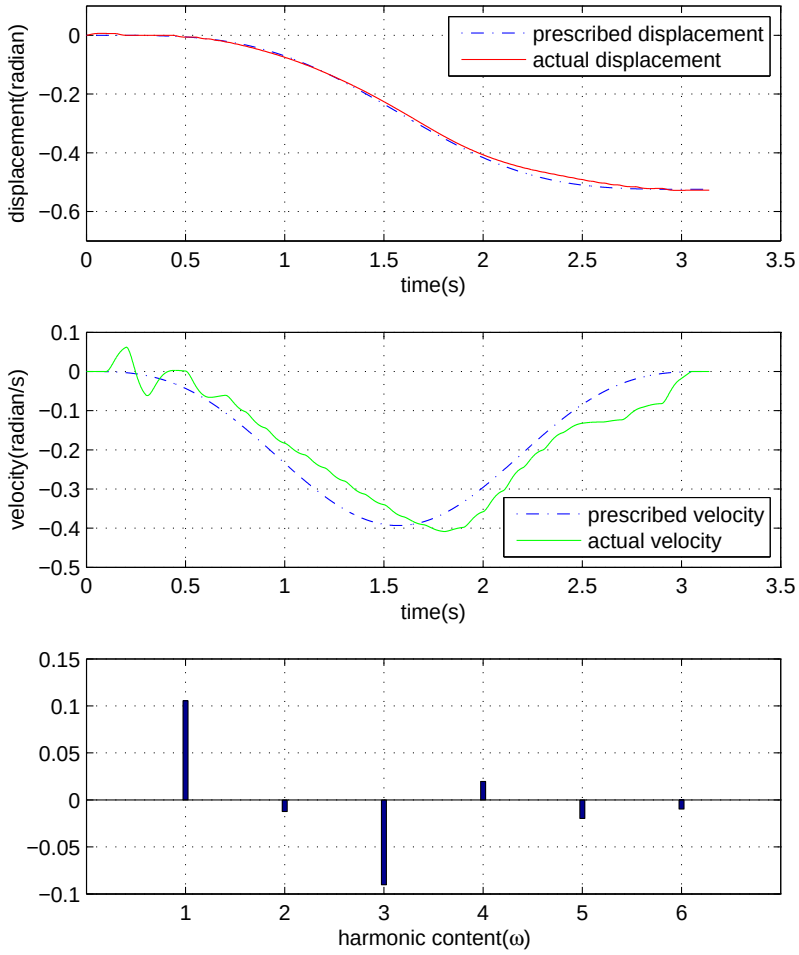


Figure 3.9: Joint motion $\theta_1$ in step 2 with the updated parameters obtained after step 1 and the harmonic content of feedback applied at joint 1

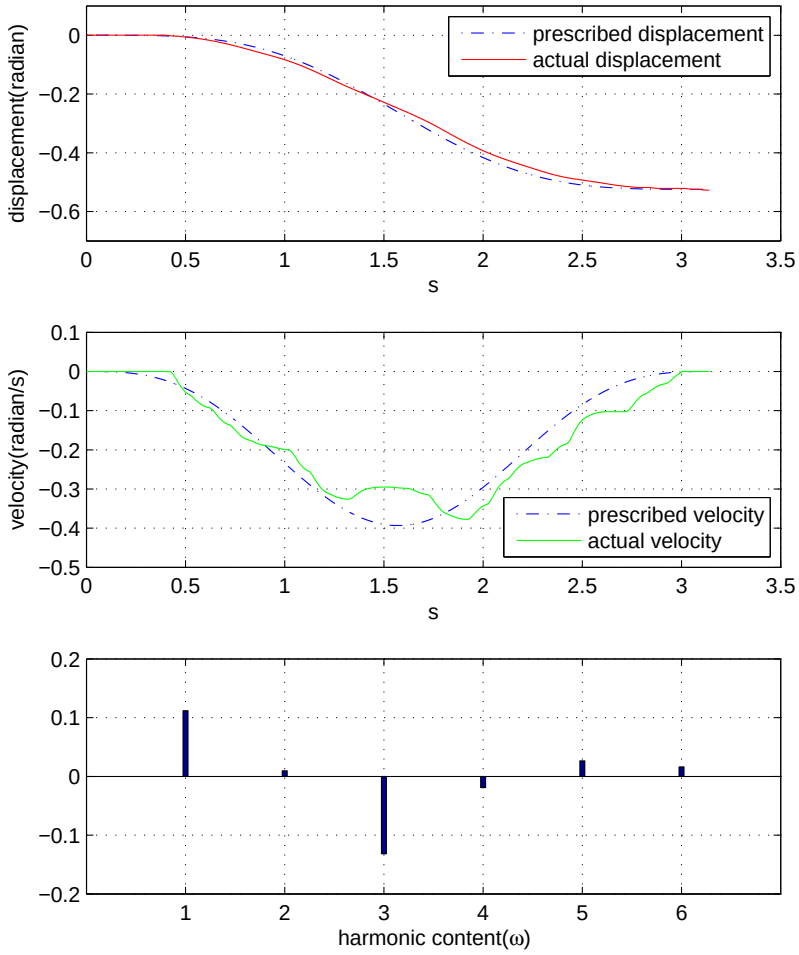


Figure 3.10: Joint motion $\theta_2$ in step 2 with the updated parameters obtained after step 1 and the harmonic content of feedback applied at joint 2

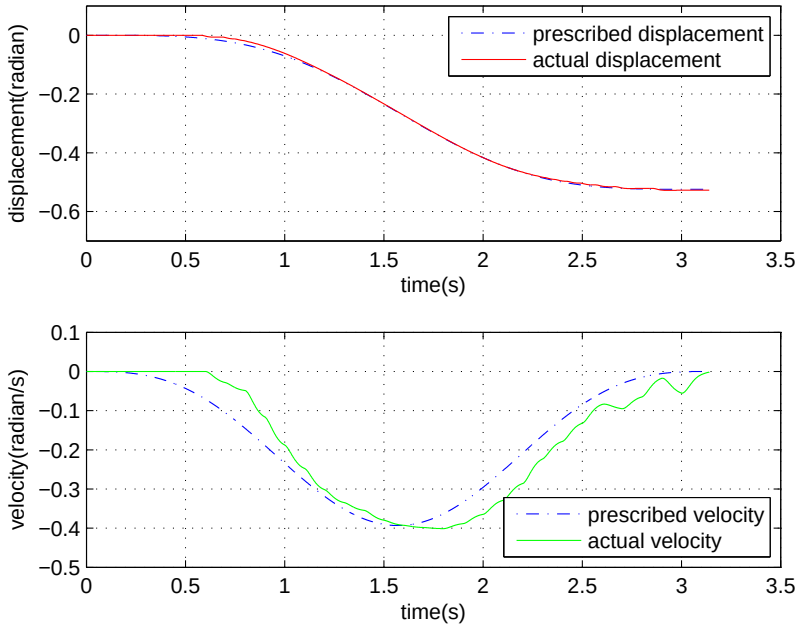


Figure 3.11: Joint motion of $\theta_1$ in step 2 with the updated parameters obtained after step 2

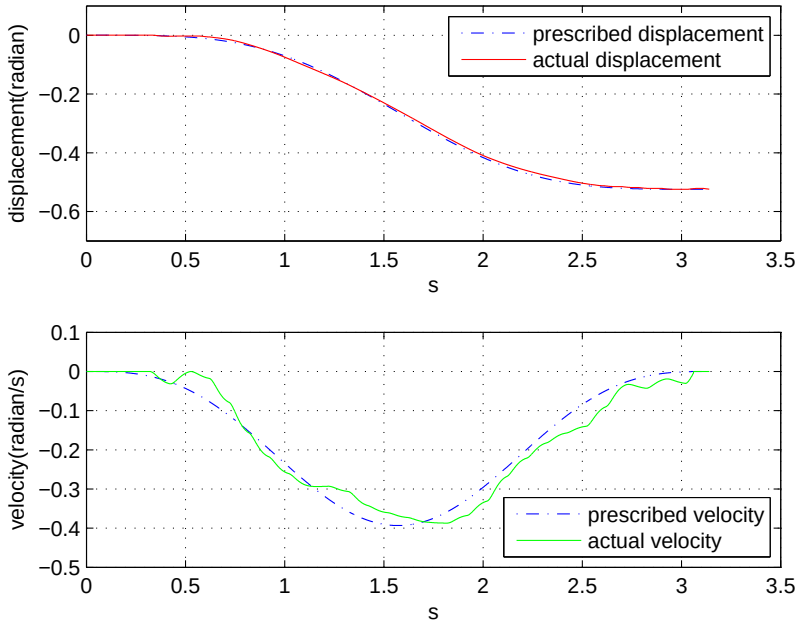


Figure 3.12: Joint motion of $\theta_2$ in step 2 with the updated parameters obtained after step 2

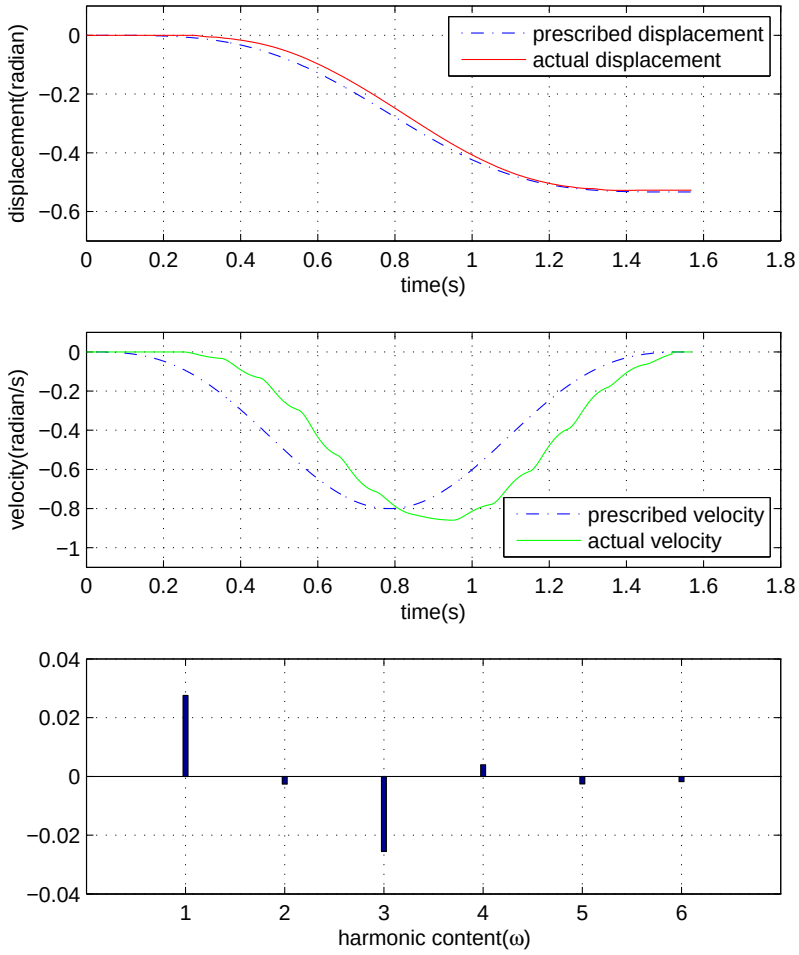


Figure 3.13: Joint motion $\theta_1$ in step 3 with the updated parameters obtained after step 2 and the harmonic content of feedback applied at joint 1

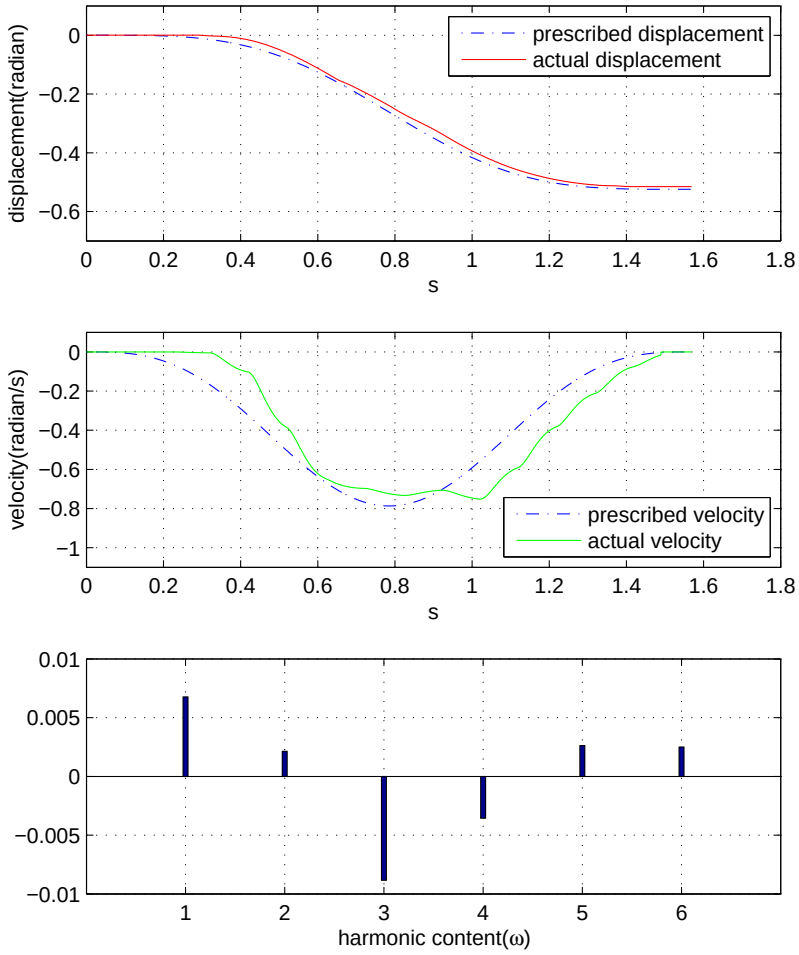


Figure 3.14: Joint motion $\theta_2$ in step 3 with the updated parameters obtained after step 2 and the harmonic content of feedback applied at joint 2

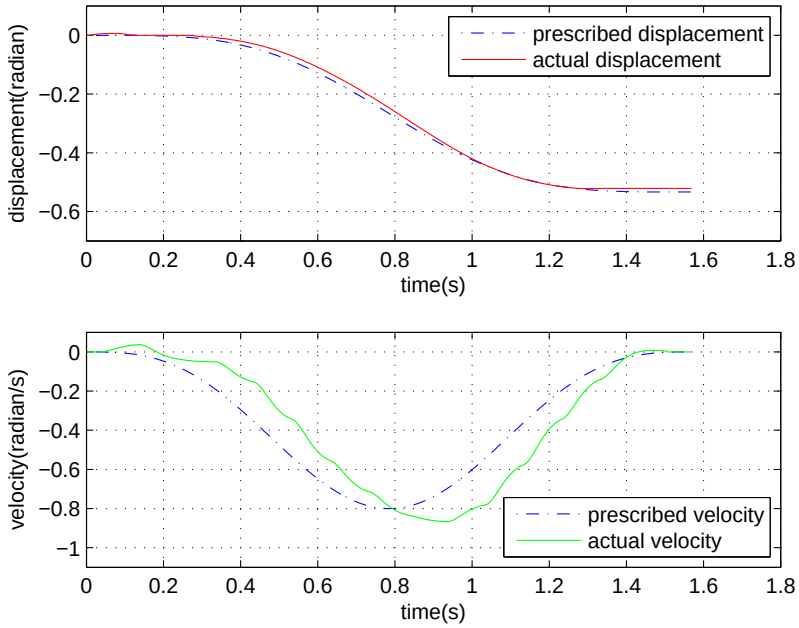


Figure 3.15: Joint motion of $\theta_1$ in step 3 with the updated parameters obtained after step 3

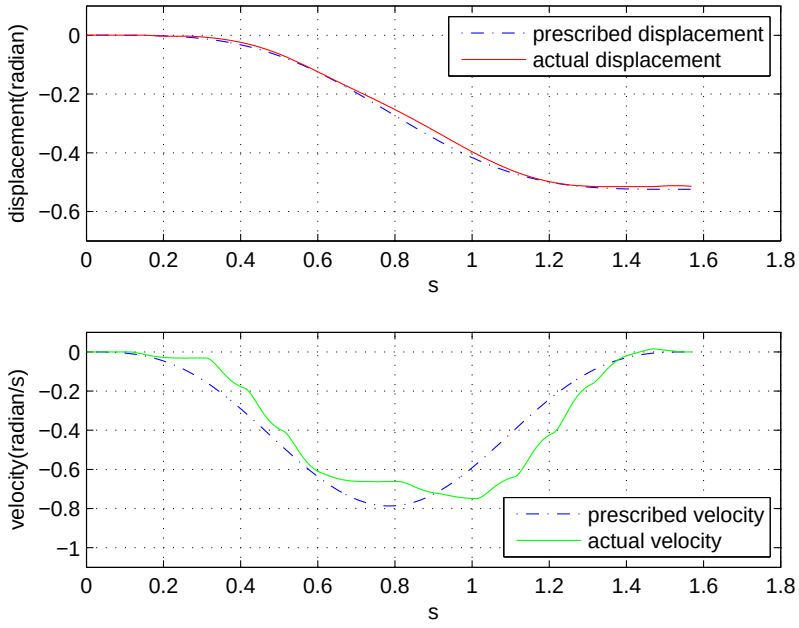


Figure 3.16: Joint motion of $\theta_2$ in step 3 with the updated parameters obtained after step 3

# Bibliography


[1] Rastegar, J., and Feng, D., 2010. "Model parameter identification of nonlinear dynamic systems by trajectory pattern method". In ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, Montreal, Canada.

[2] Tu, Q., and Rastegar, J., 1991. "On the inherent characteristics of the dynamics of robot manipulators". In 13th Biennial ASME Conference on Mechanical Vibration and Noise, Miami.

[3] Rastegar, J., Liu, L., and Yin, D., 1999. "Task–specific optimal simultaneous kinematic, dynamic, and control design of high–performance robotic systems". *IEEE/ASME Transacions on Mechatronics,* **4**(4), pp. 387–395.

[4] Gray, G., Murray-Smith, D., and Weibrenner, T., 1998. "Nonlinear model structure identification using genetic programming". *Control Engineering Practice,* **6**(11), pp. 1341–1352.

[5] Abdelazim, T., and Malik, O., 2005. "Identification of nonlinear systems by takagi–sugeno fuzzy logic grey box modeling for real time control". *Control Engineering Practice,* **13**(12), pp. 1489–1498.

[6] Li, K., Peng, J., and Bai, E., 2006. "A tow–stage algorithm for identification of nonliear dynamic system". *Automatica,* **42**(7), pp. 1189–1197.

[7] Guo, L., and Angeles, J., 1989. "Controller estimation for the adaptive control of robotic manipulator". *Robotics And Automation,* **5**(3), pp. 315–323.

[8] Ha, I., Ko, M., and Kwon, S., 1989. "An efficient estimation algorithm for the model parameters of robotic manipulators". *Robotics and Automation,* **5**(3), pp. 384–394.

[9] Lin, S., 1994. "An approach to the identifiable parameters of a manipulator". *Robotic System,* **11**(7), pp. 641–656.

[10] Lee, J., and Vukovich, G., 1997. "The dynamic fuzzy logic system: Nonlinear system identification and application to robotic manipulators". *Robotic Systems,* **14**(6), pp. 391–405.

[11] Dolanc, G., and Strmcnik, S., 2005. "Identification of nonlinear systems using a piecewise linear hammerstein model". *Systems and Control Letters,* **54**(2), pp. 145–158.

[12] Spottswood, S., and Allemang, R., 2006. "Identification of nonlinear parameters for reduced order models". *Sound and Vibration,* **295**(1–2), pp. 226–245.

[13] Kenne, G., Ahmed-Ali, A., Lamnabhi-Lagarrigue, F., and Nkwawo, H., 2006. "Nonlinear system parameters estimation using radial basis function network". *Control Engineering Practice,* **14**(7), pp. 819–832.

[14] Tai, M., and Rastegar, J., 1996. "Trajectory pattern based simultaneous kinematic and dynamic optimal design of high speed manipulators". In ASME Mechanisms Conference, Irvine, CA, pp. 519–527.

[15] Rastegar, J., and Tu, Q., 1997. "The effects of the manipulator type on the vibrational excitation during motion". *Mechanism Mach. Theory,* **32**(2), pp. 221–234.

[16] Rastegar, J., and Fardanesh, B., 1991. "Inverse dynamics model of robot manipulators using trajectory patterns – with application to learning controllers". In Eighth World Congress on the Theory of Machines and Mechanisms, Czechoslovakia.

[17] Swevers, J., Ganseman, C., Tukel, D., de Schutter, J., and Van Brussel, H., 1997. "Optimal robot excitation and identification". *Robotics and Automation,* **13**(5), pp. 730–740.

[18] Lu, Z., Shimaga, K., and Goldenberg, A., 1993. "Experimental determination of dynamic parameters of robotic arms". *Robotic Systems,* **10**(8), pp. 1009–1029.

[19] Lin, S., 1992. "An identification method for estimating the inertia parameters of a manipulator". *Robotic Systems,* **9**(4), pp. 505–528.

[20] Zak, G., Benhabib, B., Fenton, R., and Saban, C., 1994. "Application of the weighted least squares parameter estimation method to the robot calibration". *Mechanisms, Transmissions and Automation in Design,* **116**(3), pp. 890–893.

[21] Goswami, A., and Bosnik, J., 1993. "On the relationship between the physical features of robotic manipulators and the kinematic parameters produced by numerical calibration". *Mechanical Design,* **115**(4), pp. 892–900.

# Appendix A

# The Code for Target DSP Control Program

```
// DSP/BIOS header files
#include "Pismo.h"
#include "Commands.h"
#include "..\OmniBusModules.h"
#include "A4D4.h"
#include <cmath>

#define MaxPayload 0x14000
#define Pi 3.14159265
#define Scale 3276.8
#define NumOfPeriod 0x1
#define K_1 0.114 //big motor coefficient
#define K_2 0.0832  //small motor coefficient

using namespace II;

DigitalIo Io;
const int numOfParas=28;
float paras[numOfParas];

//filter coefficients (second order IIR filter, these
   numbers can be changed for different filters)
float filterCoA[3]={0.00094469,0.0018894,0.00094469};
float filterCoB[3]={1,-1.9112,0.91498};
float filterCoA_v[3]={0.00024136,0.00048272,0.00024136};
```

```
float filterCoB_v[3]={1,-1.9556, 0.95654};

//message dispatch function
void DispatchPacket(IntBuffer & Buf);

//servo class
class TPMServo : public ServoBase
{
public:
  TPMServo(Omnibus::ModuleSite site): ServoBase(site,
     false){ }

  bool Halt;
  unsigned int ADCcounts;
  unsigned int DACcounts;
  FloatBuffer  RecordChannel1;
  FloatBuffer  RecordChannel2;
  int Cursor[2];

  //memory used for filters
  float ud1[2];
  float ud2[2];
  float uv1[2];
  float uv2[2];
  float uc1[2];
  float uc2[2];

  void ServoSetup(float sampleRate)
  {
    Io.Config(0xFF); //read digital input setting
    for (int i=0;i<2;i++)
    {
      CurrentAngle[i] = 0;
      PrevAngle[i] = 0;
      FiltedAngle[i] = 0;
      Velocity[i] = 0;
      PrevVelocity[i] = 0;
      CurrentData[i] = 0;
      OriginalData[i] = 0;
      Index = 0;
```

```
    Error[i] = 0;
    FeedBack[i] = 0;
    Cursor[i] =0;
    ud1[i] = 0;
    ud2[i] = 0;
    uv1[i] = 0;
    uv2[i] = 0;
    uc1[i] = 0;
    uc2[i] = 0;
}
CurrentData[1] = CurrentData[0] = Io.Data();//
    initial value assigned to first OriginalData
MaxIndex[0] = sampleRate * Pi / paras[2] + 0.5;
MaxIndex[1] = sampleRate * Pi / paras[5] + 0.5;
Halt = false;
Ts = 1. / sampleRate;
ActualSampleRate = sampleRate;

RecordChannel1.Bytes(sizeof(float) * 2 * 40000);
RecordChannel1.Clean();
RecordChannel2.Bytes(sizeof(float) * 2 * 40000);
RecordChannel2.Clean();

//assign downloaded parameters
k1=paras[1];
k0_=paras[3];
k1_=paras[4];
omega1=paras[2];
omega2=paras[5];
kp1=paras[6];
kd1=paras[7];
kp2=paras[8];
kd2=paras[9];
f1=paras[10];
df1=paras[11];
secondf1=paras[12];
thirdf1=paras[13];
fourthf1=paras[14];
fifthf1=paras[15];
f2=paras[16];
```

```
  df2=paras[17];
  secondf2=paras[18];
  thirdf2=paras[19];
  fourthf2=paras[20];
  fifthf2=paras[21];
  f3=paras[22];
  df3=paras[23];
  secondf3=paras[24];
  thirdf3=paras[25];
  fourthf3=paras[26];
  fifthf3=paras[27];
}

// Overrides
#pragma CODE_SECTION(".hwi");
virtual void Execute(volatile short * event, int
   inputs, int outputs)
{
  for (int i=0; i<inputs; ++i)
  {
    event[i] += ADCcounts;
  }

  for (int i=0; i<outputs; ++i)
  {
    event[i] += DACcounts;
    if (i<2)
    {
      // read angular displacement
      OriginalData[i] = CurrentData[i];
      CurrentData[i] = Io.Data();
      angularDisplayDecoding(OriginalData[i],
         CurrentData[i], i);
      if (i==0)
        FiltedAngle[0] = filter((float)CurrentAngle[0]
            / 1024. * 2 * Pi,ud1,0);
      if (i==1)
        FiltedAngle[1] = filter((float)CurrentAngle[1]
            / 2048. * 2 * Pi,ud2,0);
```

```
//calculate velocity
Velocity[i] = velocityCalculation(FiltedAngle[i
    ],PrevAngle[i],Index,100,i);
if (i==0)
  Velocity[i] = filter(Velocity[i],uv1,1);
if (i==1)
  Velocity[i] = filter(Velocity[i],uv2,1);

//calculated prescribed angular positions and
    their derivatives
static float theta1, dTheta1, ddTheta1, theta2,
    dTheta2, ddTheta2;
preCalculation(Index, MaxIndex[i], theta1,
    dTheta1, ddTheta1, theta2, dTheta2, ddTheta2)
    ;

//feed forward calculation
feedForwardCalculation(Index, i, theta1, dTheta1
    , ddTheta1, theta2, dTheta2, ddTheta2);

// feed back calculation
feedBackCalculation(Index,i,theta1,dTheta1,
    theta2,dTheta2);

//  record the uploading informatin for analysis
if ((Cursor[i] < RecordChannel2.Elements()))
{
  FloatBuffer temp(0x6);
  temp[0] = Index;
  temp[1] = CurrentAngle[i];
  temp[2] = FiltedAngle[i];
  temp[3] = Velocity[i];
  if (i==0)
  {
    temp[4] = Trajectory[i] * K_1 / Scale;
    temp[5] = FeedBack[i] * K_1 / Scale;
    RecordChannel1.Copy(temp,Cursor[i],6);
  }
  else
  {
```

```
            temp[4] = Trajectory[i] * K_2 / Scale;
            temp[5] = FeedBack[i] * K_2 / Scale;
            RecordChannel2.Copy(temp,Cursor[i],6);
          }
          Cursor[i] += 6;
        }

        float controlSignal = Trajectory[i] + FeedBack[i
           ];
        if ( CurrentAngle[0] >= 512 || - CurrentAngle[0]
            >= 512 || CurrentAngle[1] >= 800 || -
           CurrentAngle[1] >= 800 )
          Halt=true;
        if (!Halt)
          event[i] += controlSignal;
      }
    }
    Index++;
}

//decode angular position from optical encoders
void angularDisplayDecoding(int originalDate, int
   currentDate, int motorChannel)
{
  int movebit =1;
  //big motor displaycement feedback input to digital
     Io bits 0 1;
  if (motorChannel==0)
  {
    originalDate = originalDate & 3;
    currentDate  = currentDate  & 3;
    if ((originalDate - currentDate)!= 0)
    {
      movebit=1;
      movebit<<=1;
      if (((movebit & currentDate) != 0) && ((movebit
         & originalDate) != 0))
      {
        movebit>>=1;
        if (((movebit & originalDate)==0) && ((movebit
```

```
            & currentDate)!=0))
          CurrentAngle[0]++;
        else if (((movebit & originalDate)!=0) && ((
           movebit & currentDate)==0))
          CurrentAngle[0]--;
      }
    }
  }
  //small motor displaycement feedback input to
     digital Io bits 2 3;
  if (motorChannel==1)
  {
    originalDate = originalDate & 12;
    currentDate  = currentDate  & 12;
    if ((originalDate - currentDate)!= 0)
    {
      movebit=1;
      movebit<<=3;
      if (((movebit & currentDate) != 0) && ((movebit
         & originalDate) != 0))
      {
        movebit>>=1;
        if (((movebit & originalDate)==0) && ((movebit
            & currentDate)!=0))
          CurrentAngle[1]--;
        else if (((movebit & originalDate)!=0) && ((
           movebit & currentDate)==0))
          CurrentAngle[1]++;
      }
    }
  }

}

//velocity calculation function
float velocityCalculation(float currentDisplay, float
   prevDisplay, int index, int freqAdjust, int channel
   )
{
  float velocity=0;
```

```
  if (index % freqAdjust == 0)
  {
    velocity = (currentDisplay - prevDisplay) *
       ActualSampleRate / freqAdjust;
    if (channel == 0)
    {
      PrevVelocity[0] = velocity;
      PrevAngle[0] = currentDisplay;
    }
    if (channel == 1)
    {
      PrevVelocity[1] = velocity;
      PrevAngle[1] = currentDisplay;
    }
  }
  else
  {
    if (channel == 0)
      velocity = PrevVelocity[0];
    if (channel == 1)
      velocity = PrevVelocity[1];
  }
  return velocity;
}

//IIR second order filter
float filter(float inputValue, float u[], int DorV)
{
  float FiltedValue=0;
  float un=0;

  if (DorV==0)
  {
    un = inputValue - filterCoB[1] * u[0] - filterCoB
       [2] * u[1];
    FiltedValue = filterCoA[0] * un + filterCoA[1] * u
       [0] + filterCoA[2] * u[1];
  }
  if (DorV==1)
```

```
  {
    un = inputValue - filterCoB_v[1] * u[0] -
       filterCoB_v[2] * u[1];
    FiltedValue = filterCoA_v[0] * un + filterCoA_v[1]
        * u[0] + filterCoA_v[2] * u[1];
  }
  u[1] = u[0];
  u[0] = un;
  return FiltedValue;
}

//calculate important values for feedforward torques
void preCalculation(int index, int maxIndex, float &
   theta1, float & dTheta1, float & ddTheta1, float &
   theta2, float & dTheta2, float & ddTheta2)
{
  static float c, c_, c3, c3_, s, s_, s3, s3_;
  if (index <= NumOfPeriod * maxIndex)
  {
    c = std::cos(Pi * (((float)index) / MaxIndex[0]));
    c3 = std::cos(3 * Pi * (((float)index) / MaxIndex
       [0]));
    s = std::sin(Pi * (((float)index) / MaxIndex[0]));
    s3 = std::sin(3 * Pi * (((float)index) / MaxIndex
       [0]));
    c_ = std::cos(Pi * (((float)index) / MaxIndex[1]))
       ;
    c3_ = std::cos(3 * Pi * (((float)index) / MaxIndex
       [1]));
    s_ = std::sin(Pi * (((float)index) / MaxIndex[1]))
       ;
    s3_ = std::sin(3 * Pi * (((float)index) / MaxIndex
       [1]));

    theta1 = k1 * (c - c3 / 9 - 8. / 9);
    theta2 = k0_ + k1_ * (c_ - c3_ / 9);
    dTheta1 = k1 * omega1 * (s3 / 3 - s);
    ddTheta1 = k1 * omega1 * omega1 * (c3 - c);
    dTheta2 = k1_ * omega2 * (s3_ / 3 - s_);
    ddTheta2 = k1_ * omega2 * omega2 * (c3_ - c_);
```

```
  }
  else
  {
    c = std::cos(Pi * NumOfPeriod);
    c3 = std::cos(3 * Pi * NumOfPeriod);
    s = std::sin(Pi * NumOfPeriod);
    s3 = std::sin(3 * Pi * NumOfPeriod);
    c_ = std::cos(Pi * NumOfPeriod);
    c3_ = std::cos(3 * Pi * NumOfPeriod);
    s_ = std::sin(Pi * NumOfPeriod);
    s3_ = std::sin(3 * Pi * NumOfPeriod);

    theta1 = paras[1] * (c - c3 / 9 - 8. / 9);
    theta2 = paras[3] + paras[4] * (c_ - c3_ / 9);
    dTheta1 = paras[1] * paras[2] * (s3 / 3 - s);
    ddTheta1 = paras[1] * paras[2] * paras[2] * (c3 -
       c);
    dTheta2 = paras[4] * paras[5] * (s3_ / 3 - s_);
    ddTheta2 = paras[4] * paras[5] * paras[5] * (c3_ -
        c_);
  }
}

//feedforward controller
void feedForwardCalculation(int index, int channel,
   float theta1, float dTheta1, float ddTheta1, float
   theta2, float dTheta2, float ddTheta2)
{
  float temp,temp1,temp2,temp3,temp4;
  temp = theta2 - k0_ ;
  if (channel==0)
  {
    if (index <= NumOfPeriod * MaxIndex[0])
    {
      temp1 = f1 + df1 * temp + 0.5 * secondf1 * temp
         * temp + 1.0 / 6 * thirdf1 * temp * temp *
         temp + 1.0 / 24 * fourthf1 * temp * temp *
         temp * temp;
      temp2 = f3 + df3 * temp + 0.5 * secondf3 * temp
         * temp + 1.0 / 6 * thirdf3 * temp * temp *
```

```
        temp + 1.0 / 24 * fourthf3 * temp * temp *
        temp * temp;
      temp3 = df1 + secondf1 * temp + 0.5 * thirdf1 *
        temp * temp + 1.0 / 6 * fourthf1 * temp *
        temp * temp + 1.0 / 24 * fifthf1 * temp *
        temp * temp * temp;
      temp4 = df3 + secondf3 * temp + 0.5 * thirdf3 *
        temp * temp + 1.0 / 6 * fourthf3 * temp *
        temp * temp + 1.0 / 24 * fifthf3 * temp *
        temp * temp * temp;
      Trajectory[0] = 2 * temp1 * ddTheta1 + temp2 *
        ddTheta2 + 2 * temp3 * dTheta1 * dTheta2 +
        temp4 * dTheta2 * dTheta2;
      Trajectory[0] = Trajectory[0] / K_1 * Scale;
      Trajectory[0] = Trajectory[0];
    }
    else
      Trajectory[0] = 0;
  }
  if (channel==1)
  {
    if (index <= NumOfPeriod * MaxIndex[1])
    {
      temp1 = f3 + df3 * temp + 0.5 * secondf3 * temp
        * temp + 1.0 / 6 * thirdf3 * temp * temp *
        temp + 1.0 / 24 * fourthf3 * temp * temp *
        temp * temp;
      temp2 = f2 + df2 * temp + 0.5 * secondf2 * temp
        * temp + 1.0 / 6 * thirdf2 * temp * temp *
        temp + 1.0 / 24 * fourthf2 * temp * temp *
        temp * temp;
      temp3 = df1 + secondf1 * temp + 0.5 * thirdf1 *
        temp * temp + 1.0 / 6 * fourthf1 * temp *
        temp * temp + 1.0 / 24 * fifthf1 * temp *
        temp * temp * temp;
      temp4 = df2 + secondf2 * temp + 0.5 * thirdf2 *
        temp * temp + 1.0 / 6 * fourthf2 * temp *
        temp * temp + 1.0 / 24 * fifthf2 * temp *
        temp * temp * temp;
      Trajectory[1] = temp1 * ddTheta1 + 2 * temp2 *
```

```
            ddTheta2 - temp3 * dTheta1 * dTheta1 + temp4
            * dTheta2 * dTheta2;
        Trajectory[1] = - Trajectory[1] / K_2 * Scale;
        Trajectory[1] = Trajectory[1];
      }
      else
        Trajectory[1] = 0;
    }
}

//feed back controller
void feedBackCalculation(int index, int channel, float
    theta1, float dTheta1, float theta2, float dTheta2
   )
{
  if (channel == 0)
  {
    Error[0] = 0;
    if (index <= NumOfPeriod * MaxIndex[0])
        Error[0] = theta1 - FiltedAngle[0];
    else
      {
        if ((NumOfPeriod % 2) == 0)
          Error[0] = 0 - FiltedAngle[0];
        else
          Error[0] = 0 - paras[1] * 16 / 9 -
             FiltedAngle[0];
      }
    FeedBack[0] = Error[0] * paras[6] + (dTheta1 -
       Velocity[0]) * paras[7];
    FeedBack[0] = FeedBack[0] / K_1 * Scale;
  }
  if (channel == 1)
  {
    Error[1] = 0;
    if (Index <= NumOfPeriod * MaxIndex[1])
        Error[1] = theta2 - (paras[3]+paras[4] * 8. /
           9) - FiltedAngle[1];
    else
      {
```

```
            if ((NumOfPeriod % 2) == 0)
              Error[1] = 0 - FiltedAngle[1];
            else
              Error[1] =  0 - paras[4] * 16. / 9 -
                 FiltedAngle[1];
          }
        FeedBack[1] = Error[1] * paras[8] + (dTheta2 -
           Velocity[1]) * paras[9];
        FeedBack[1] = 0 - FeedBack[1] / K_2 * Scale;
      }
    }

protected:
    int CurrentAngle[2];  //displaycements
    float FiltedAngle[2];
    float PrevAngle[2];

    float Velocity[2];    //velocities
    float PrevVelocity[2];

    int CurrentData[2];   //digital input from encoders
    int OriginalData[2];

    int Index;
    int MaxIndex[2];
    float Trajectory[2];

    float Error[2];      //displaycement and velocity
       errors
    float FeedBack[2];

    float Ts;         //sample period
    float ActualSampleRate; //sample rate

    //downloaded parameters
    float k1;    //indicate range of theta 1
    float omega1;
    float k0_;     //midpoint of theta 2
    float k1_;     //indicate range of theta 2
    float omega2;
```

```
  float kp1;     //pd control p for link 1
  float kd1;     //pd control d for link 1
  float kp2;     //pd control p for link 2
  float kd2;     //pd control d for link 2

  //values calculated from estimated model paras
  float f1;
  float df1;
  float secondf1;
  float thirdf1;
  float fourthf1;
  float fifthf1;
  float f2;
  float df2;
  float secondf2;
  float thirdf2;
  float fourthf2;
  float fifthf2;
  float f3;
  float df3;
  float secondf3;
  float thirdf3;
  float fourthf3;
  float fifthf3;
};

// servo control thread
class DriveThread : public Thread
{
public:
  DriveThread(IIPriority priority)
    : Thread(priority), Xfr1(new Transfer), Once(false)
      {}
  ~DriveThread()
  {
    delete Xfr1;
  }

  void Acquire()
  {
```

```
  Start.Release();
  Available.Acquire();
}

void WriteBuffer(FloatBuffer & Buffer, int index, int
   s)
{
  int Elements = index;
  int Cursor = 0;
  while (Cursor < Elements)
    {
    int Residual = Elements - Cursor;
    int Chunk = std::min(Residual,MaxPayload);
    IntBuffer Num(0x4);
    Num[0] = Chunk;
    Num[1] = s;
    Xfr1->Send(ccNumOfAcqusition, Num);
    // Copy buffer to payload
    FloatBuffer Dst(Chunk);
    Dst.Copy(Buffer, Cursor, Chunk);
        Xfr1->Send(ccAcqusition, Dst);
    Cursor += Chunk;
    }
  IntBuffer Cmd(0x4);
  Xfr1->Send(ccDataReceiveComplete, Cmd);
}

void Controler1()
{
  int DacDelay;

  TPMServo ServoIo(Omnibus::mSite0);

  if (!Once)
  {
      LoadModule(Omnibus::mSite0, Omnibus::mtA4D4);
      Once = true;
  }

  ServoIo.Open();
```

```
// Locate event buffer onchip
ServoIo.SegId(1);

ClockRateUI * Clock = ClockRateUIPtr(ServoIo.Clock()
    );
Clock->Rate(1000);
float actualSampleRate =  Clock->RateActual();

// Enable all analog input and output channels
ServoIo.InputChannels().EnableChannels(2);
ServoIo.OutputChannels().EnableChannels(2);

//  ...Configure delay of DAC clock
DacDelay = 5.e3;
ServoIo.Delay(DacDelay);

ServoIo.ADCcounts = ServoIo.Module()->Info().Adc().
    OffsetCounts();
ServoIo.DACcounts = ServoIo.Module()->Info().Dac().
    OffsetCounts();
ServoIo.ServoSetup(actualSampleRate);

Sleep(5000);
// Start processing
ServoIo.Start();

float Period = (NumOfPeriod * Pi / paras[2] + 0.1) *
     1000;
Sleep(Period);

ServoIo.Halt = true;
Sleep(100);

// Stop processing
ServoIo.Stop();

FloatBuffer Dst1(ServoIo.RecordChannel1.Ints());
FloatBuffer Dst2(ServoIo.RecordChannel2.Ints());
Dst1.Copy(ServoIo.RecordChannel1);
Dst2.Copy(ServoIo.RecordChannel2);
```

```
    int index1 = ServoIo.Cursor[0];
    int index2 = ServoIo.Cursor[1];

    // Close driver
    ServoIo.Close();

    //send angle record data to host
    WriteBuffer(Dst1,index1,1);
    WriteBuffer(Dst2,index2,2);
  }

protected:
  Semaphore       Available;
  Semaphore       Start;
  Transfer *        Xfr1;
  bool          Once;

  //
    ------------------------------------------------------------------

  // Methods
  //
    ------------------------------------------------------------------

  void Execute()
  {
    for (int i = 0; i < 1; ++i)
      {
        IntBuffer Cmd(0x4);
        Cmd[0] = i;
        Cmd[1] = 100;   Cmd[2] = 101;  Cmd[3] = 102;
        Xfr1->Send(ccControler, Cmd);
      }
    while(!Terminated())
      {
        Start.Acquire();
        Controler1();
        Available.Release();
      }
  }
```

```
};
DriveThread Controler(tpNormal);

//
   ------------------------------------------------------------------------

//  Main Program
//
   ------------------------------------------------------------------------

void IIMain()
{
  Controler.Resume();
  Transfer Xfr;
  IntBuffer C;
  IntBuffer Cmd(0x4);
  Cmd[0] = 0;
  Cmd[1] = 0;
  Cmd[2] = 0;
  Cmd[3] = 0;
  Xfr.Send(ccLogin, Cmd);

  //wait for message to arrive from host
  for (;;)
  {
    Xfr.Recv(C);
      DispatchPacket(C);
  }
}

//
   ------------------------------------------------------------------------

//  DispatchPacket() --  Dispatch a Host command
//
   ------------------------------------------------------------------------

void DispatchPacket(IntBuffer & Buf)
{
    //  Configure Header
```

```
    TransferBufferHeader TBH(Buf.Header());

    //  Use the Peripheral ID to separate packet types
    switch (TBH.PeripheralId())
    {
    //receive downloaded parameters
    case ccDownloadParas:
    {
      float paraNum=0;
      memcpy(&paraNum, Buf.Addr(), sizeof(float));
      for (int i=0; i<(int)paraNum; i++)
        memcpy(paras+i, Buf.Addr()+i+1, sizeof(float));
      FloatBuffer Cmd(0x30);
      Cmd[0] = (int)paraNum;
      for (int j=0;j<numOfParas;j++)
        Cmd[j+1] = paras[j];
      Transfer    Xfr1;
      Xfr1.Send(ccDownloadParas, Cmd);  //for test
    }
      break;

    //execute servo control
    case ccRun:
      Controler.Acquire();
      break;

    case ccInit:
      break;

    default:
      break;
  }
}
```