ABSTRACT

ANALYSIS AND CONTROL OF MONOLITHIC PIEZOELECTRIC NANO-ACTUATOR

by
Xuemei Sun

The study of the monolithic piezoelectric actuator, the dominant type of micro-positioner is an attractive and challenging area, where real-time control theory and digital signal processing are effectively applied. The actuator can be applied in precision instruments and precision control, such as microscopes, medical and optics instruments because of the piezoelectric ceramic's high resolution, fast transient response, and potential low cost. However, hysteresis nonlinearity and lightly damped vibration exist in the system, which limit the actuator applications.

This work focuses on the hysteresis characteristics in time and frequency domains along with experimental and simulated results to verify the effectiveness of the model in describing the hysteresis phenomena. The analytic expressions of the hysteresis harmonics are further applied in hysteresis parameter estimation.

A reduced order nonlinear hysteresis observer compensator is proposed, and the stability of the compensated system is discussed. The compensator reduces the hysteresis effect significantly under simulated and experimental conditions. Furthermore, an adaptive hysteresis observer compensator is further presented to compensate the slowly changed hysteresis parameters. Time division multi-control strategy is proposed to implement fast transient response, low vibration and high resolution.

Extensive numerical simulation and real-time experiment are carried out to verify the control strategies. GUI is developed to implement the communication between the code in DSP memory and Labview, which improves the efficiency in system test.

ANALYSIS AND CONTROL OF MONOLITHIC PIEZOELECTRIC NANO-ACTUATOR

by
Xuemei Sun

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

Department of Electrical and Computer Engineering

January 2001

APPROVAL PAGE

ANALYSIS AND CONTROL OF MONOLITHIC PIEZOELECTRIC NANO-ACTUATOR

Xuemei Sun

Dr. Timothy N. Chang, Dissertation Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Andrew Meyer, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Bernard Friedland, Committee Member — Date
Distinguished Professor of Electrical and Computer Engineering, NJIT

Dr. Marshall Kuo, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Zhiming Ji, Committee Member — Date
Associate Professor of Mechanical Engineering, NJIT

Dr. Haim Baruh, Committee Member — Date
Associate Professor of Mechanical and Aerospace Engineering, Rutgers University

BIOGRAPHICAL SKETCH

Author: Xuemei Sun

Degree: Doctor of Philosophy

Date: January 2001

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, January 2001

- Master of Science in Electrical and Computer Engineering,
  Chengdu University of Science and Technology, Chengdu, P. R. China, 1992

- Bachelor of Science in Electrical and Computer Engineering,
  Chengdu University of Science and Technology, Chengdu, P. R. China, 1989

Major: Electrical Engineering

Presentations and Publications:

Timothy Chang, Xuemei Sun,
"Analysis and control of monolithic piezoelectric nano-actuator,"
To appear in IEEE Transactions on Control System Technology, January.

Timothy Chang, Xuemei Sun, Zhiming Ji, and Reggie Caudill,
"Analysis and control of monolithic piezoelectric nano-actuator,"
Proceedings of the 2000 ACC, Chicago, IL, pp. 3086-3090.

Timothy Chang, Xuemei Sun, Vincenzo Pappano, Zhiming Ji, and Reggie Caudill,
"High-precision, multiple degree-of-freedom piezoelectric actuator,"
Proceedings of SPIE, vol. 3519, Nov. 1998.

To my beloved parents

ACKNOWLEDGMENT

I wish to express my deepest gratefulness and respect to my advisor Dr. Timothy N. Chang for his guidance and care and financial support during my study and research.

Many thanks to Dr. Andrew Meyer, Dr. Bernard Friedland, Dr. Marshall Kuo, Dr. Zhiming Ji, Dr. Haim Baruh for serving as members of the committee.

For their expertise and technical assistance, I'd like to thank Vincenzo Pappano, Tahir Nazir, Biao Zhen, Chung-Hsiang Wang, Puttiphong Jaroonsiriphan and other fellow classmates.

Finally I would like to thank my parents and my sister for their great encouragement and love.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter Page

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter Page

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure Page

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Objective

Precision manufacturing technology has received more and more attention in recent years because of the development of high-density semiconductors and optoelectronics. The dominant type of micropositioners is piezoelectric due to the improvement of piezo-ceramic characteristics (scale factor, linearity, stability, etc.) and the inherent high resolution of displacement. A number of piezoelectric micropositioners or actuators have been invented, such as two-axis motion apparatus [1], piezo-translation stage [2]. The applications cover the areas of microscopes, medical and optics instruments.

The monolithic piezoelectric positioner [3] is applied in this work, which has the following advantages:

1. Monolithic structure
2. Wide bandwidth and high resolution
3. Two degree-of-freedom, with extension to six degree-of-freedom
4. Multiple layers stackable to increase range
5. Low cost

However, due to the geometry and material properties, nonlinearities such as hysteresis and nonlinear scale factor exist in the system. They can cause a number of undesirable effects, including loss of stability robustness, limit cycles and steady state error. Furthermore, the characteristic of under-damping also brings the side effect of vibration [25][38]. This study is concentrated on hysteresis characteristics analysis, hysteresis modeling and compensation, and vibration control and precision control.

1.2 Literature Survey on Hysteresis Model

Hysteresis effect between the displacement and the electric field is an intrinsic characteristic of piezoelectric ceramics. This effect becomes increasing noticeable when the electric field strength or the piezoelectric sensitivity of the material is increased. Chen and Montgomery [36] gave a physical explanation for the hysteresis phenomenon from a macroscopic viewpoint: the external applied cyclic electric field generates domain switching which lead to the hysteresis loop. The domain is made up of parallel dipoles, and the effective number of the dipoles aligned in the direction of the external electric field. The domain switches as the electric field changes. The delay of the switching causes the hysteresis loop.

The objectives of modeling the hysteresis in the piezoelectric actuator are to accurately tract the hysteresis behavior, analyze the hysteresis phenomena, and eliminate the hysteresis effect.

Unlike saturation and dead zone, the description of hysteresis is not unique. In the following, five most popular models: hysteron, Bouc-Wen, Chua-Stromsmoe, Preisach and Dahl, and their applications on piezoelectric actuators will be briefly discussed.

1.2.1 Hysteron Model

Hysteron model [4] is defined on piecewise monotone continuous inputs. For each linear piece, the description is shown in Figure 1.1, where x and u are the output and input of the hysteron, respectively; $\Omega_0(\omega)$ is the area bounded by Γ_- and Γ_+; and u_2, u_1 define the boundary of the hysteron. The model describing the single hysteron is expressed in four areas: $u \notin (\ u_2 \quad u_1\ )$, $\Omega_0(\omega)$, Γ_- and Γ_+. The smooth hysteresis is expressed by the combination of all the pieces of hysteron.

Hysteron Model in Piezoelectric Actuators – Maxwell Model [5, 6]

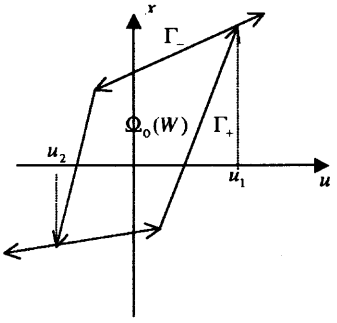

Figure 1.1 Hysteron model

In piezoelectric actuators, the hysteron is simplified and has the shape of parallel quadrilateral, as shown in Figure 1.2. The mathematical expression is,

$$F = \begin{cases} k(x - x_b) & if \quad |k(x - x_b)| < f \\ f\ sgn(\dot{x}) \quad and \quad x_b = x - \frac{f}{k} sgn(\dot{x}) & else \end{cases} \tag{1.1}$$

and

$$sgn(x) = \begin{cases} 1 & for \quad x \geq 0 \\ -1 & for \quad x < 0 \end{cases} \tag{1.2}$$

where x is the input displacement, F is the output force, k is the stiffness of the spring, $f = \mu N$ is the breakaway friction force of the block, and x_b is the position of the block.

Figure 1.3 shows the smooth hysteresis behavior. The hysteresis model consists of several elasto-slide elements in parallel, each having an incrementally larger breakaway force. Hence, the simple relationship of Figure 1.2 becomes a piecewise linear approximation of rate-independent hysteresis. The set of parameters, such as stiffness k_n and breakaway force f_n, are obtained by piece-wise linear fit of the rising curve [5, 6]. The model has been applied by Choi and Kim [7] to compensate the hysteresis.

1.2.2 Bouc-Wen Model

Bouc-Wen model [4] can represent a large class of hysteresis behavior, such as inelastic stress-strain relationships in structures, and magnetorheological behavior.

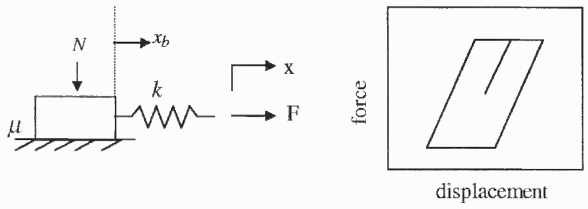

Figure 1.2 Single elasto-slide element

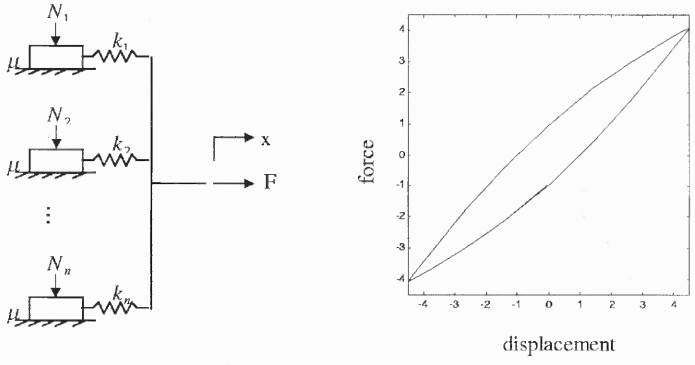

Figure 1.3 Maxwell model of a smooth hysteresis curve: Parallel several elasto-slide elements, each subjected to increasing normal force

It has an appealing mathematical expression, given as,

$$\begin{array}{ll} \ddot{x}(t) = f(x, \dot{x}, z, u) & x(t_0) = x_0,\ \dot{x}(t_0) = \dot{x}_0 \\ \dot{z} = A\dot{x} - \beta\dot{x}\,|z|^n - \gamma\,|\dot{x}|\,|z|^{n-1}\,z & z(t_0) = z_0 \end{array} \tag{1.3}$$

where x and z stand for position and hysteresis restoring force, respectively. The model can be reduced to,

$$\frac{dz}{dx} = A \pm (\alpha \pm \beta)z^n \tag{1.4}$$

where the choice of positive or negative signum depends on the values of $\dot{x}$ and z to be positive or negative. The scale and general shape of the hysteresis loop are governed by A, α and β, and n controls the smoothness of the force-displacement curve. For $n = 1$, $A = 1$, Wen and Asce [8] showed the shape of hysteresis for six sets of combination of α and β.

Application of Bouc-Wen Model in Piezoelectric Actuators

Jouaneh and Tian [9] modified Bouc-Wen model by setting $n = 1$, and replacing position $x(t)$ by input $u(t)$, as following,

$$\dot{z} = A\dot{u} - \beta |\dot{u}| z - r\dot{u} |z| \tag{1.5}$$

The state variables of the linear system, z and hysteresis parameters are estimated using the least squares method. One-step-ahead predictor of state z is calculated and applied as feedback to compensate the hysteresis nonlinearity.

1.2.3 Chua-Stromsmoe Model

Chua-Stromsmoe model [4] has a S shape and is used to model ferromagnetic hysteresis that has saturation characteristics. It is described as

$$\dot{x} = \omega(\dot{u})h(x)\{g[u(t)] - f(x)\}, \quad x(t_0) = x_0 \tag{1.6}$$

where u and x stand for input and output, respectively; f and g stand for dissipative and restorative phenomena, respectively; h controls the loop width, enabling to generate asymmetric loops, and ω provides frequency-dependent loop widening. This model can express extensive frequency-dependent behavior, such as loop widening or narrowing for different frequency ranges. It has some proven properties, such as conditions for existence, uniqueness, boundedness and periodicity of a solution x. An algorithm was proposed to determine functions f, g, h and ω.

Since Chua-Stromsmoe model describes the saturated hysteresis phenomenon, while piezoelectric ceramic devices never work on saturated areas, this model has not been applied in piezoelectric actuators.

1.2.4 Preisach Model

Preisach model is expressed as double integral of the output of an ideal relay [10, 11], as below.

$$f(t) = \iint_{\alpha \geq \beta} \mu(\alpha, \beta)\gamma_{\alpha\beta}(u(t))\, d\alpha d\beta \tag{1.7}$$

where $\gamma_{\alpha\beta}(u(t))$ is the hysteresis operator shown in Figure 1.4, $\mu(\alpha, \beta)$ is a weighted function, and α and β stand for up and down switching values of the input $u(t)$. Hence the model can be interpreted by a summation of an infinite set of simplest hysteresis operators and weighting functions.

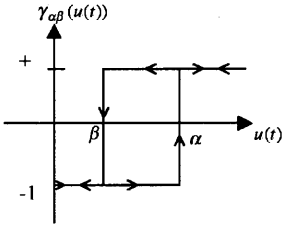

Figure 1.4 Hysteresis operator

In the following, the numerical implementation of equation (1.7) is discussed. Figure 1.5 shows the definitions of the major hysteresis loop, as well as the first and the second order reversal curves, where f_α and f_β are the outputs corresponding to $u(t) = \alpha$ and $u(t) = \beta$, respectively. The first order reversal curves are used to obtain a mesh of α and β values following the procedure below:

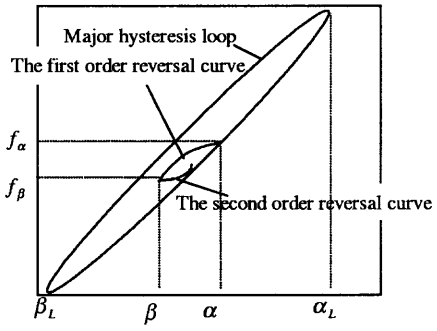

Figure 1.5 The first order reversal curve

1. Obtain a set of first-order reversal curves from experimental data. Let's say, the reversal curves start from the ascending path of major hysteresis. Hence for fixed β as β_L, a set of α_i is obtained.

2. Changing the size of main hysteresis loop, i.e., the values of α_L and β_L to be α'_L and β'_L , and repeating step 1, we can get another set of α_i corresponding

to β_L'. Therefore, a table of (α_i, β_i) can be obtained, which covers the area of the triangle constrained by $\alpha \geq \beta$, $\alpha_{\max}$ and $\beta_{\min}$.

Figure 1.6 shows the hysteresis loop with several switching input values α_i and β_i. β_L and α_0 define the lowest curve, and β_1 and α_0 define the highest one, and so on. Figure 1.7 shows the corresponding interface link, which passes the points of (β_L, α_0), (β_1, α_0), (β_1, α_1), (β_N, α_1), (β_N, α_N), $(u(t), \alpha_N)$. For the case of a horizontal initial link and a vertical final link, the numerical implementation of equation (1.7) is,

$$\begin{aligned} f(t) &= 2\{F(\alpha_1, \beta_L) - F(\alpha_1, \beta_1)\} - F(\alpha_L, \beta_L) \\ &\quad +2\sum_{i=1}^{N-1} F(\alpha_{i+1}, \beta_i) - F(\alpha_{i+1}, \beta_{i+1}) \\ F(\alpha, \beta) &= \frac{1}{2}(f_\alpha - f_{\alpha\beta}) \end{aligned} \tag{1.8}$$

where $F(\alpha, \beta)$ represents the change of $f(t)$ as $u(t)$ changes from α to β. Since the initial link and the final link can be either horizontal or vertical lines, the hysteresis model has four expressions.

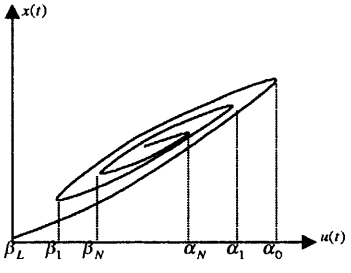

Figure 1.6 Hysteresis loop with several α and β, and final input on a descending branch

Preisach Model in Piezoelectric Actuators

The model adapted to a piezoelectric actuator is discussed in [12, 13, 14], with the value of $\gamma_{\alpha\beta}(u(t))$ modified to 0 or 1 instead of -1 or +1.

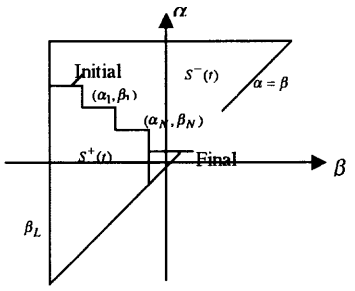

Figure 1.7 Interface link to the hysteresis curves

1.2.5 Dahl Model

P. R. Dahl [15, 16] proposed a solid friction model, as

$$\frac{dF}{dx} = \sigma \left|1 - \frac{F}{F_c} sgn(\dot{x})\right|^i sgn(1 - \frac{F}{F_c} sgn(\dot{x})) \tag{1.9}$$

where the function sgn is defined in equation (1.2); F is the solid friction force, x describes the displacement, F_c is the Coulomb friction force or "running friction force", σ is the rest stiffness or the slope of the force deflection curve at $F = 0$, i stands for the type of different crystals. The effect of σ and i on the shape of hysteresis is fully discussed in [17].

Application of Dahl Model in Piezoelectric Actuators

For most lead zirconate - lead titanate (PZT) materials, $\sigma = 1$ and $i = 1$. Hong and Chang [18] proposed an adaptive dither method to compensate the hysteresis in a piezoelectric actuator based on the simplified Dahl model

$$\frac{d^2x}{dt^2} + \gamma \frac{dx}{dt} + k_n x = k_v u - k_1 F \tag{1.10}$$

$$\frac{dF}{dt} = \frac{dx}{dt} - \frac{F}{F_c}\left|\frac{dx}{dt}\right| \tag{1.11}$$

where x and u are the actuator end displacement in meters and applied voltage in Volts, γ, k_n, k_v and k_1 are system coefficients. Regardless the second term in the

right hand side, equation (1.10) describes the linear part of the system. $k_n = \omega_n^2$ with ω_n as natural frequency in rad/sec; $\gamma = 2\varsigma\omega_n$ with ς as damping ratio; and k_v is the input scale factor. Equation (1.11) describes the nonlinear hysteresis. k_1 and F_c govern the scale and the shape of the hysteresis loop.

1.3 The Hysteresis Model in This Work

To describe the hysteresis phenomena in piezoelectric actuators, the analytic Bouc-Wen model and Dahl model, the lumped parameter Maxwell model, and the non-analytic Preisach model have been applied. In this work, to implement the objectives of:

1. Describing the hysteresis phenomena in piezoelectric actuators
2. Analyzing the characteristics of the hysteresis in time and frequency domains
3. Deriving corresponding compensated strategies

the simplified Dahl model in equations (1.10) and (1.11) is applied. It has the advantages of:

1. Analytic formulation which is suited to analyze hysteresis characteristics
2. Only five plant parameters are adequate to describe the characteristics of the piezoelectric actuators up to the first resonance
3. As special cases of Dahl model as well as Bouc-Wen model
   The model can be obtained from the Bouc-Wen model by setting $\beta = 0$ and $n = 1$.

1.4 Dissertation Organization

Chapter 2 of this dissertation describes the experimental setup and the hardware used in this work. Chapter 3 discusses the real-time operating system including

the architecture of the TMS320C31 microprocessor, software platform and the graphical user interface development. The hysteresis characteristics in the piezoelectric actuator are discussed in Chapter 4. The hysteresis harmonic expressions are derived in Chapter 5. Hysteresis parameter estimation and compensation are proposed in Chapters 6 and 7 respectively. Chapter 8 proposes an adaptive hysteresis compensation scheme for input sinusoids. Chapter 9 proposes a time-division multi-control scheme to implement vibration and precision control. Chapter 10 summarizes the work presented along with the future work recommendations.

CHAPTER 2
HARDWARE DESCRIPTIONS

Figure 2.1 shows the physical system of the monolithic piezoelectric actuator. The white cruciform object in the middle is the piezoelectric actuator. The motions in x and y axes are measured by the two capacitance sensors, whose probes are shown in the picture. The experimental setup for the control system is shown in Figure 2.2. It consists of two Model 601B-PCB amplifiers, two ADE 3800 capacitance sensors, and a real-time operating system including Dalanco Spry Model 310 digital signal processing board and PC. The properties of these hardware components are described below.

2.1 Model 601B-PCB High-Voltage Amplifiers

Model 601B-PCB high voltage amplifier is made by TREK Incorporated. It has the following properties: the gain is 100, the voltage range is $[0\ \pm 1000] Volts$ or $[-500\ 500] Volts$; the current range is $[0\ \pm 10] mA$; and the bandwidth is greater than $3MHz$. The detail information is described in the Operator's Manual [19].

2.2 ADE 3800 Capacitance Sensors

The ADE capacitance sensor measures and transfers the displacement into voltage. The linear relationship between the displacement and the voltage is that for $2.5\mu m$ displacement, one volt is generated.

The sensor has the following properties: The resolution is $1nm$; The bandwidth can be changed among 10, 100, 1000 and $5000Hz$ by plugging in proper jumpers. In the experiment setup, the bandwidth is chosen as $1000Hz$ since the resonance frequency of the PZT system is around $500Hz$.

When more than one sensors are applied in the same system, multi-unit connection should be applied. One ADE sensor is connected as master and the

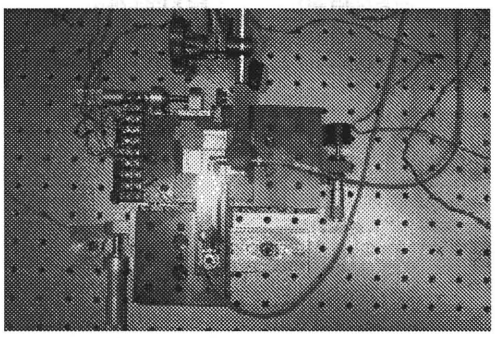

Figure 2.1 The system of the monolithic piezoelectric actuator

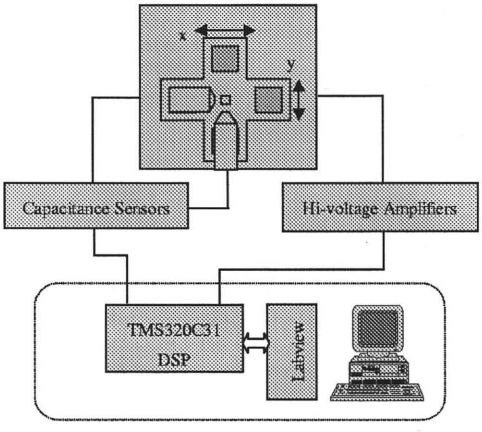

Figure 2.2 Experimental setup for the piezoelectric actuator control system

others are connected as slaves. For detail information, please see the User's Guide [20].

2.3 Real Time Operating System

The real time operating system consists of the hardware of Dalanco Spry Model 310B data acquisition and signal processing board (DSP) and PC, and the software of C/C++ and Labview.

The DSP includes a Texas Instrument TMS320 C31 microprocessor, dual ported memory, a four-channel A/D converter and a two-channel D/A converter.

Because of the $50MHz$ clock on all devices, it is an ideal system in real time data acquisition, data processing and digital control.

Labview is a graphical programming language and the user's programs are created in block diagram forms. Labview simulates the panel of a physical instrument, like knobs, buttons, graphs and indicators, and has powerful data acquisition and data processing libraries.

To monitor the status of the PZT plant, and to operate the DSP from PC, the graphical user interface has been developed in this work. It has the following advantages:

1. Easy to operate complex control strategies

   The system can be operated without much understanding the control principles.

2. Convenient in system testing

   The coefficients, such as PID coefficients, and the magnitude and frequency of the reference signals can be easily changed using a keyboard or a mouse while the system is running.

3. Safer to operate

   The system is safer since the closed-loop control can be switched to open loop control quickly and easily.

Chapter 3 will discuss the real-time operating system in detail.

2.4 Monolithic Piezoelectric Actuator

The cruciform piezoelectric actuator is one part of the six-degree-of-freedom actuator patented by Timothy Chang [3]. It is a positioning device capable of providing two degrees of freedom: x and y axes. It is constructed from a single piezoelectric plate with the material of lead zirconate-lead titanate (PZT). The top and bottom

faces of the cruciform are covered by silver, an electric conductive material to form electrodes.

2.4.1 Form the Electrodes

The top and bottom faces of the piezoelectric actuator have the same structure, as shown in Figure 2.3. The four shadow parts on the sides are four electrodes, and the shadow part in the middle is the target for measuring the displacements in x and y axes. The procedure of forming the electrodes is as following:

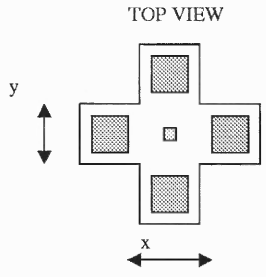

Figure 2.3 Cruciform piezoelectric actuator

1. Deposit photoresistor (red liquid) on the areas of the top face where the silver should be kept. Wait until the photoresistor is dry. Do the same thing on the bottom face.

2. Put the cruciform PZT proceeded in step 1 into a container, and bake in an oven with temperature of 70^0F for 3 to $3\frac{1}{2} hours$

3. Eliminate the unwanted silver on the PZT using nitric acid with density of 50%

4. Clean the acid using water, and the photoresistor using acetone.

2.4.2 Theory of Operation

The linear motion in the x and y directions can be generated by applying suitable drive voltages to the electrodes, as shown in Figure 2.4. In the following, the operation theory in x direction is explained, and that in y direction is the same

except that the symbol of the applied voltage is V_y. Apply V_x to electrodes 1 and 2, and $-V_x$ is simultaneously applied to electrodes 3 and 4. The resultant net displacement in x direction is given by:

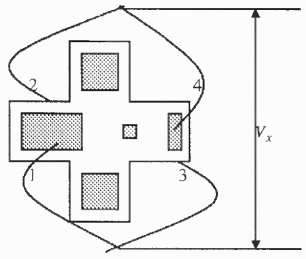

Figure 2.4 Electrode connection of the piezoelectric actuator

$$\Delta x = d_{31}\frac{L}{T}V_x \tag{2.1}$$

where L and T are the length and thickness of the cruciform section covered by electrodes 1 and 2 (or electrodes 3 and 4), and d_{31} is the piezoelectric voltage constant with typical value of $-250 \times 10^{-12} m/V$.

It can be seen that when voltage is applied, one piezo-element contracts while the other extends. Thus the displacement of the target is approximate to twice of that of one piece only. Central to the base-plate design is that the orthogonal member (e.g. the y-axis element when x-axis is actuated) acts as a soft nonlinear spring that stabilizes the unenergized equilibrium. The pressure produced by the deformation is given by:

$$P_x = d_{31}Y_{11}^E\frac{V_x}{T} \tag{2.2}$$

where Y_{11}^E is the Young's modulus with typical value of $6 \times 10^{10} N/m^2$.

2.4.3 Nominal Performance Characteristics

For a devise with one layer cruciform PZT, the nominal performance characteristics are summarized as below:

1. Accuracy: $2.5nm$

2. Maximum load: about $0.5kg$ with $500Volts$ drive

3. Normal range of displacement: vary from $5\mu m$ to $15\mu m$

4. Bandwidth: about $500Hz$

5. Size: $10cm \times 10cm \times 1mm$

2.5 Summary

This chapter briefly describes the hardware of the experimental system, including high voltage amplifiers, capacitance sensors, real-time operating system, and the PZT actuator. The real time operating system will be discussed in Chapter 3.

CHAPTER 3

REAL TIME OPERATING SYSTEM

The real-time operating system consists of both hardware and software parts. The hardware includes Dalanco Spry Model 310B data acquisition and signal processing board and PC, and the software includes C code for DSP, Labview and C++ code for graphical user interface. In the following, the architecture of the DSP is first discussed. The driver programs for DSP, and Labview platform are then explained. The development of graphical user interface on Labview platform is described at last.

3.1 Hardware Description

The block diagram of the Dalanco Spry Model 310B data acquisition and signal processing board is shown in Figure 3.1. It consists of a TMS320C31 Microprocessor, 512K words memory, a four-channel A/D converter and a two-channel D/A converter. The main properties of each component are described below.

3.1.1 Key Features of TMS320C31 Microprocessor

Figure 3.2 shows the block diagram of TMS320C31. Some of the key features are listed below as references in programming.

- CPU
  - $40ns$ single-cycle instruction execution time with $50MHz$ clock on all devices
  - 32-bit instruction words, 32-bit data words, and 24-bit addresses
  - parallel ALU and multiplier instructions in a single cycle
- peripheral

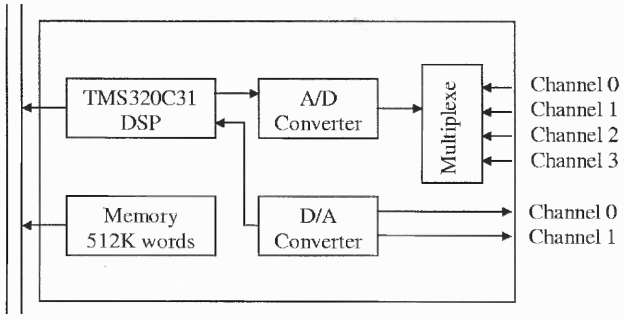

Figure 3.1 Block diagram of Dalanco Spry Model 310 data acquisition and signal processing board

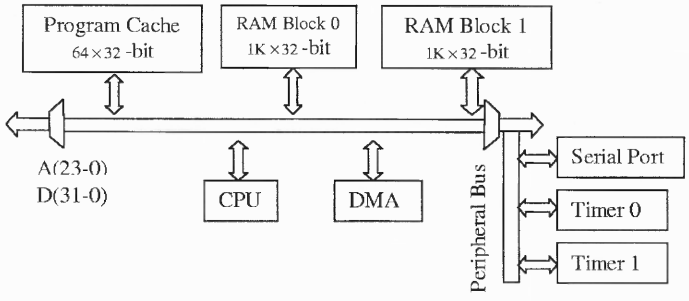

Figure 3.2 TMS320C31 block diagram

- One memory-mapped serial port to support 8, 16, 24 or 32-bit full-duplex transfer

- Memory
  - 64 × 32-bit instruction cache
  - Two 1K 32-bit-word single-cycle dual-access on-chip RAM block
  - 16M words addressing space
  - Preprogrammed bootloader

- Memory interface
  - One external memory bus

memory address (hex)	functions
0-03F	reset, interrupt, trap vectors
040-7FFFFF	external memory
800000-807FFF	reserved
808000-8097FF	peripheral bus, memory mapped
809800-809BFF	RAM block 0
809C00-809FFF	RAM block 1
80A000-FFFFFF	external memory

Table 3.1 Memory map

3.1.2 Memory Distribution

TMS320C31 has 24-bit addresses, and the address starts from $0h$ to $FFFFFFh$. The h appended indicates that the number is in hexadecimal. As shown in Table 3.1, the usable area for the user program is $040h - 7FFFFFh$, which is about $8.4M$ words. In our system only the first $512K$ words memory size can be used. The memory distribution of the variables used in user programs is listed in Appendix A.

3.1.3 Analog to Digital (A/D) Converter

- A/D converter type: Maxim121
- Maximum rate: $300KHz$
- Input voltage range: $\pm 5v$
- Number of bits: 14
- The microprocessor controls the parameters of:
  - The next A/D input channel to be sampled (channels 0-3)
  - The gain of the programmable gain amplifier

  by writing to a latch mapped at memory location 0FFFFFFh. The control word is 4-bit, with bits 0 and 1 control the gain, and bits 2 and 3 control the A/D channel. The gain of the programmable gain amplifier is equal to one in

Analog voltage (v)	-5	...	0	...	4.99939
Digital value (hexadecimal)	2000	...	0	...	1FFF

Table 3.2 Analog input and its corresponding digital value

Analog voltage (v)	-5	...	0	...	4.99939
Digital value (hexadecimal)	800	...	0	...	7FF

Table 3.3 Digital input and its corresponding analog output

this system. Table 3.2 shows the linear relationship between the analog and the digital signals.

3.1.4 Digital to Analog (D/A) Converter

- D/A converter type: 2 AD7243
- Rate: $250kHz$ single/$140kHz$ dual channel
- Output voltage range: $\pm 5v$

Table 3.3 shows the linear relationship between the analog and the digital signals.

3.2 Software Description

This section describes the DSP driver programs, and the basic components in Labview programs.

3.2.1 DSP Driver Programs

Three functions defined in file "$d310biox.h$" are used in our programs. They are explained below.

InitDsp()

Function *InitDsp*(*void*) implements the initialization of the DSP board: set up the serial port, timer and latch.

ReadAdc()

Function *ReadAdc*(*int channel*) starts the A/D converter, and reads the digital value from the ADC output buffer. The linear relationship between the analog and the digital signals is shown in Table 3.2. It should be noted that to recover the true value of the analog signal, the digital signal should multiply the scale factor of 4.99939/8191.

WriteDAC() **and** *WriteDACs*()

Function *WriteDAC*(*int value*, *int channel*) starts the D/A converter and output an analog signal. The digital data is 32-bit word in which the lower 16-bit is the digital value in channel 0, and the higher 16-bit is that in channel 1.

Similar to *WriteDAC*, function *WriteDACs*(*int value*0, *int value*1) starts the D/A converter to convert a 32-bit digital value to two analog voltages. The higher 16-bit is *value*1 and the lower 16-bit is *value*0

The linear relationship between the analog and the digital signals is shown in Table 3.3. It should be noted that to convert a digital signal to its corresponding analog value, the digital signal should multiply 2047/4.9976.

Sampling Rate

The sampling rate of the A/D converter is defined using constant $TIMPER0$. The value of $TIMPER0$ is calculated from the following equation:

$$TIMPER0 = \frac{50 \times 10^6}{sampling\ rate \times numcalls \times 8} \tag{3.1}$$

where *numcalls* = the number of function calls of *ReadAdc*(). It should be noted that the definition of $TIMPER0$ should be placed before the statement of "*#include d*310*biox.h*".

3.2.2 DSP Memory Access from PC

The memory on the DSP board is dual ported, i.e., it is accessible at any time to the DSP as well as to the PC via the bus interface. To access the DSP memory from PC, the page value and address counter should be set first.

Set Page Value The 64K-page value is bits $a16 - a19$ of the desired address value. It is latched and need not be changed once set unless accesses are made across page boundaries. *port_value* is 306h and the command is,

$$outword16(0x306, page\_value);$$

Set Address Counter

The address counter within a page is determined by $a0 - a15$. After each memory access, the address counter is incremental. A string of data may thus be read or written with only one address 'setup'. *port_value* is 302h and the command is,

$$outword16(0x302, address\_value);$$

Access Actual Data Two reads (or writes) are needed because the data is 32 bits wide. *port_value* is 300h and the commands are,

$$outword16(0x300, data\_lower\_16\_bits);$$

$$outword16(0x300, data\_upper\_16\_bits);$$

The port values are summarized in Table 3.4. For further information about the DSP, please refer to Dalanco Spry manual [21].

Example: Codes for Data Acquisition

```
/*————————— Data Acquisition ————————————————————
  Objective: obtain the data stored in the DSP memory
```

Port value = IO Base + Offset		
Offset	Read	Write
0	RAM address (word)	
1		
2		RAM address (word)
3		
4	interrupt acknowledge	allow interrupt
5	interrupt TMS320 (byte)	disallow interrupt (byte)
6	set TMS320 (byte)	address page (byte)
7	reset TMS320 (byte)	

Table 3.4 Host PC IO assignments (Base IO Address = $300h$)

```
  starting address: 0x1388
  length: 5000
-------------------------------------------------------------*/
# include < stdio.h >
# include < math.h >
# include < conio.h >
# define Y0 x1388
# define DATASIZE 5000
void main(void) {
  int longl, longh;
  long int longv;
  unsigned short i;
  FILE *fp;
  fp=fopen("grabi.out","wt");                    //file name=grabi.out
  (void) _outp(0x306,0);                          //set page value
  for(i=0; i<DATASIZE; i++)  {
        (void)_outpw(0x302,(Y+i));                //set address
        longl = _inpw(0x300) & 0x0000FFFF; //access actual data
        longh = _inpw(0x300) & 0x0000FFFF;
```

```
            longv = longl | (longh << 16);
            printf(fp, "%ld \n",longv);
        }
    fclose(fp);
    } //end of file
```

3.2.3 Labview – Graphical Programming Language (G)

Labview is a program development environment like C and Basic language. Instead of using text-based language, however, it uses a graphical programming language (G) to create programs in block diagram form. The Labview program, named virtual instrument (VI) programs, consists of three parts, as described below.

Front Panel - Interactive User Interface

It is named since it simulates the panel of a physical instrument. The front panel shows controls such as knobs and push buttons, and graphs and indicators. A mouse and keyboard are used to enter data, and graphs or indicators show results.

Block Diagram - Source Code

The block diagram is the source code and constructed in graphical programming language. Similar to Simulink or Visual Simulation (VisSim), it is a pictorial solution to a programming problem.

Icon - Function Definition

Icon is used to embed a VI as a subVI. Similar to the functions in other software language, icon is the place that the inputs and outputs of a function are defined. The icon and the connector of a VI work like a graphical parameter list so that other VIs can pass data to it. The property makes VIs to be hierarchical and modular.

Labview includes libraries for data acquisition, GPIB and serial communications, data analysis, data presentation and data storage. For more information on Labview, please refer to Labview manual [22].

3.2.4 Code Interface Reference (CIN)

A CIN is a block diagram node associated with a section of source code written in a conventional programming language. It appears on the diagram as an icon with input and output terminals. When the CIN is called, the input terminals pass the specified data to the codes; and after the code finishes executing, the output terminals return data. In the following, the commonly used functions in CIN source code are explained. For detail information of CIN node, please refer to Code Interface Reference Manual [23].

The basic CIN source file looks like this,

```
# include "extcode.h"
CIN MgErr CINRun(variables);
CIN MgErr CINRun(variables);{
  /* users source code */
  return noErr;
}
```

extcode.h

It is a file that defines basic data types and a number of routines that can be used by CINs and external subroutines. It should always be included at the beginning of the source code.

CINRun ***routine***

Labview calls the *CINRun* routine when it is time for the node to be executed. *CINRun* receives the input and output values as parameters.

MgErr

MgErr is a Labview data type that corresponds to a set of error codes that the manager routines return.

NumericaArrayResize

It resizes a data handle that refers to a numeric array.

3.3 Graphical User Interface Development

The objective of developing graphical user interface is to make it easier to operate and test the complex control strategies. The advantages of using the interface are as following

1. Different control strategies, such as open-loop, input shaping and PI control, can be chosen or switched easily

2. The coefficients of PID, magnitudes or frequencies of reference inputs can be changed conveniently

3. Labview's powerful data processing libraries and virtual instruments can be applied directly

Developing the interface involves the following steps:

1. Add a CIN node to the block diagram

2. Create input and output connections

3. Develop and compile an interface C++ source code

4. Link the code to CIN

In the following, an example is used to illustrate the development of the graphical user interface. The requirement of the code is that Labview should send and receive data from the DSP memory through a CIN node.

Interface C++ Source Code

```
/*————————— CIN source file: example.c —————————————————

Objective: Demonstrate how to access the DSP memory

Input of a CIN node: *freq2dsp

Output of a CIN node: *output

————————————————————————————————————————————————————*/

# include "extcode.h"

# include < stdio.h >

#include < math.h >

#include <conio.h>

#define FREQ1 0x12F5

#define Y 0x1383

#define FSCALING 5000

CIN MgErr CINRun (float64 *freq2dsp, float64 *output);

CIN MgErr CINRun (float64 *freq2dsp, float64 *output){

/* ENTER YOUR CODE HERE */

        long int longv, longh, longl;

        outp(0x306,0x0);                        //page value

//Write Freq to DSP memory

        longv=(long int)((*freq2dsp) * FSCALING);

        outpw(0x302, FREQ1);            //DSP memory address

        outpw(0x300, longv);            //low 16-bit data

        outpw(0x300, (longv) >> 16);    //high 16-bit data

//Read output from DSP memory

        outpw(0x302, Y);            //DSP memory address

        longl=inpw(0x300) & 0x0000FFFF; //low 16-bit data
```

```
    longh=inpw(0x300) & 0x0000FFFF; //high 16-bit data
    longv=longl | (longh << 16);          //32-bit data
    *output=(float64)(longv)/8191.0 * 4.99939; //data to PC
    return noErr;
} //end of file
```

Compile the source code

The source code for the CIN must be compiled in a format that Labview can use. To compile the above source codes, write the following *makefile*.

```
/*————— makefile: example.lvm —————*/
IDE=VC
name=example.lvm
type=CIN
codeDir=d: \ xuemei \ labv \ cinsub \ communicate
CINTOOLSDIR=f:\ LabVIEW \ Cintools
!include $(CINTOOLSDIR) \ ntlvsb.mak
/*————— end of file —————*/
```

The *makefile* name should have extension of *.lvm* to indicate that it is a Labview *makefile*. Under Visual C++, The CIN code can be compiled using the following command,

nmake /f filename.lvm

3.4 Summary

A digital signal processor board is applied to implement data acquisition, data processing and real-time control. This chapter describes the architecture of the TMS320 C31 microprocessor, which is helpful in developing programs for real-time control strategies and graphical user interface.

Labview programming language is powerful in data acquisition and signal processing, and Code Interface Reference is the bridge to connect C program in DSP memory and Labview.

Software, including DSP driver programs and Labview programming language are described in the second part of this chapter. An example is applied to illustrate how to develop and compile the interface code. The whole packages tailored to the specific real time control strategies are listed in Appendix B-E, consisting of the source code for Labview, CIN and DSP. The structures of the block diagram in each program are the same, including (1) starting or stopping DSP program, (2) parameter communication between PC and DSP through a CIN node, (3) data acquisition and analysis.

CHAPTER 4

HYSTERESIS CHARACTERISTICS

Hysteresis is an inherent nonlinear phenomenon in piezoelectric actuators, and its modeling and compensation have been proposed [5-7][9][12-14]. However, its characteristics have not been well analyzed. This is because the popular hysteresis models in piezoelectric actuators, such as Maxwell and Preisach, are non-analytic. In this chapter, simulation and experiment are applied to study the characteristics of the hysteresis. The analytic models in (1.10) and (1.11) are applied to simulation studies with parameters obtained experimentally, as shown below:

$$\gamma = 1.1612 \times 10^3 \quad k_n = 1.1893 \times 10^7 \quad k_v = 0.43058 \tag{4.1}$$

$$k_1 = 1.1 \times 10^7 \quad F_c = 7.5 \times 10^{-6} \tag{4.2}$$

4.1 Experimental and Simulated Results

The experiment was done on y-axis of the cruciform actuator. However, the results are applicable to x-axis as well because of the same construction and material. Furthermore, because x and y axes are orthogonal, the results also hold when both axes are actuated. The applied input sinusoid has frequency of $0.1Hz$, and the sampling rate of the system is $20Hz$.

Simulation was applied with input sinusoid of $10Hz$. The sampling rate is $200Hz$ and the total time is two seconds. Although the simulated results between 0.1 and $10Hz$ have little difference, the speed is much faster by choosing the faster frequency.

4.1.1 Hysteresis Characteristics in Time Domain

Figures 4.1-4.3 show the experimental results on the shape of hysteresis loops for different magnitudes of input sinusoid. Figure 4.1 shows all the curves in one graph,

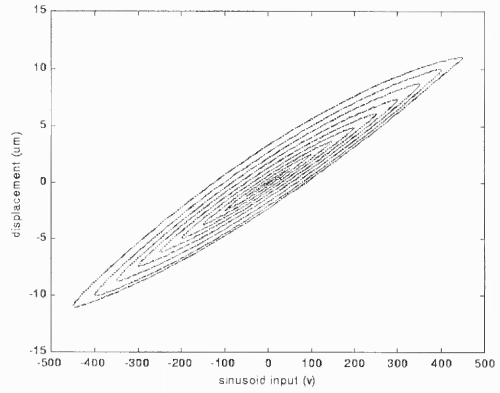

Figure 4.1 Experimental hysteresis loops with different magnitudes of input sinusoid (at $0.1Hz$)

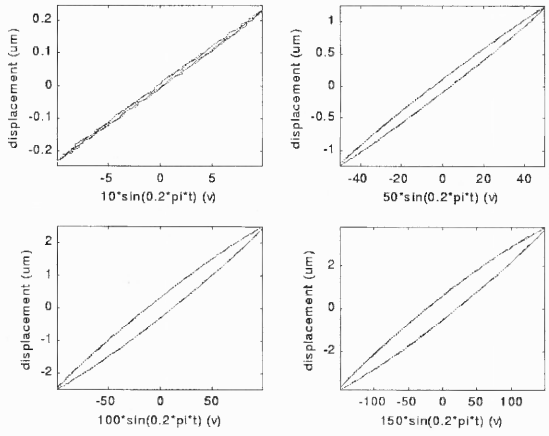

Figure 4.2 Experimental hysteresis loops with different magnitudes of input sinusoid (at $0.1Hz$)

and Figures 4.2 and 4.3 show the curves separately. Figures 4.4-4.6 are the corresponding simulated results. It is observed that

1. The shape of the hysteresis loop is elliptic with no saturation

2. The size of the hysteresis increases as the applied voltage increases

4.1.2 Hysteresis Characteristics in Frequency Domain

In the previous section, hysteresis characteristics are interpreted based on observing the size and shape of the "hysteresis loops" which are commonly used in the literature

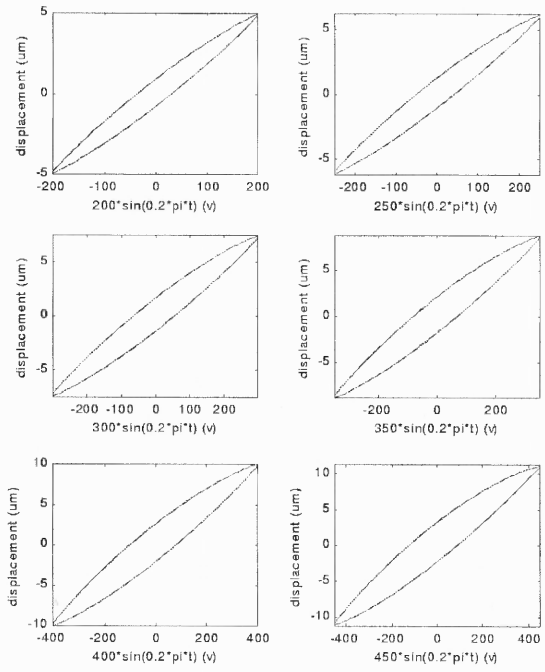

Figure 4.3 Experimental hysteresis loops with different magnitudes of input sinusoid (Continued)

as a qualitative criterion. However, these loops are difficult to quantify, rendering them useless in implementing hysteresis compensation. A primary objective of this work is therefore to develop a quantitative measure of the degree of hysteresis nonlinearity. This measure should be easy to derive, and valid for a wide range of experimental conditions (such as drive amplitude and frequency). It will be shown in this section that the output harmonics of the piezoelectric actuator contain sufficient information to describe the hysteresis phenomenon while maintaining enough "robustness" under different drive conditions (different frequencies and magnitudes).

First and Third harmonics

The spectra corresponding to the experimental and simulated responses have been obtained using Discrete Fourier Transform (DFT). The first and the third harmonics are shown in Figures 4.7 and 4.8 with different magnitudes of input

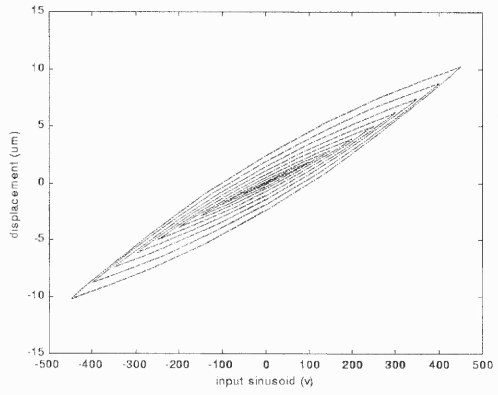

Figure 4.4 Simulated hysteresis loops with different magnitudes of input sinusoid (at $10Hz$)

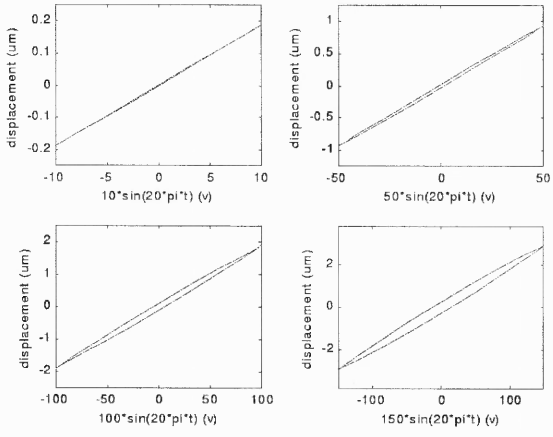

Figure 4.5 Simulated hysteresis loops with different magnitudes of input sinusoid (at $10Hz$)

sinusoid, where mag is the magnitude of the input sinusoid, and h_1 and h_3 stand for the amplitudes of the first and the third harmonics, respectively. It is observed that

1. The first harmonics are approximately linear to mag

2. The third harmonics are proportional to mag^2

Weak Scalability

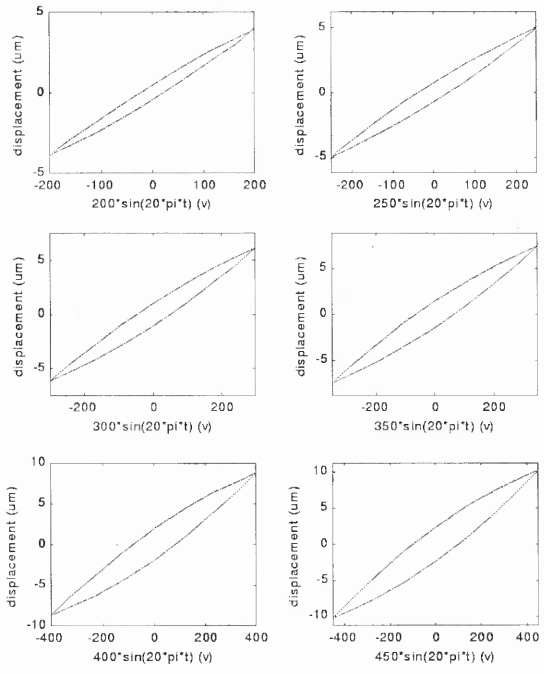

Figure 4.6 Simulated hysteresis loops with different magnitudes of input sinusoid (Continued)

Now since the fundamental and third harmonics of the actuator output behave in a consistent, predictable manner, it appears that the amplitude ratio between the harmonics may be used as a metric for the hysteresis. In this section, the notions of harmonic ratio and weak scalability are introduced to support the effort in quantifying the degree of hysteresis nonlinearity.

Definition (Normalized Ratio): The normalized ratio is defined as

$$N_R = \frac{\text{magnitude ratio of the third harmonic to the first harmonic}}{\text{magnitude of input sinusoid}}$$

This ratio can be readily determined experimentally by applying a sinusoidal excitation to the PZT actuator and measuring the output in frequency domain. The notion of weak scalability deals with the robustness of the normalized ratio under different drive conditions.

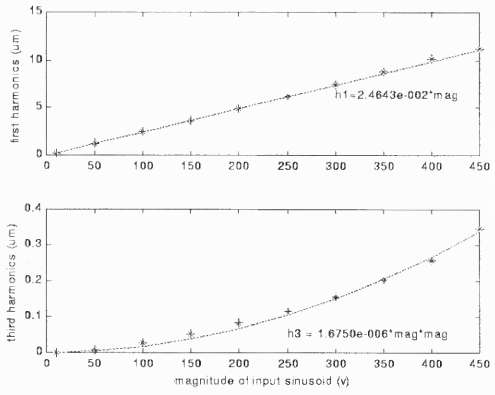

Figure 4.7 Experiment: Curve fitting of the magnitudes of the first and third harmonics

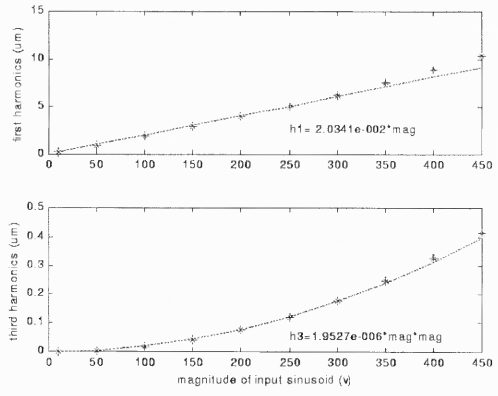

Figure 4.8 Simulation: Curve fitting of the magnitudes of the first and third harmonics

Definition (Weak Scalability): A system is said to possess weak scalability if at low input frequency the normalized ratio is approximately constant and independent of input amplitude.

Figures 4.9 and 4.10 respectively show the experimental and simulated normalized ratio for the hysteresis of the piezoelectric actuator. Both figures indicate that the hysteresis possesses weak scalability. Table 4.1 summarizes the numerical results for convenience of comparison. It should be noted that the experimental N_R corresponding to $10Volts$ drive falls out of the predicted range. This is

primarily due to weak signal to noise ratio when the noise floor limits the resolution of measurement.

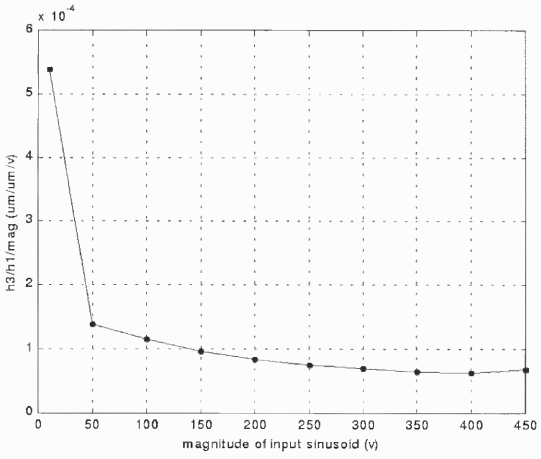

Figure 4.9 Experiment: Weak scalability

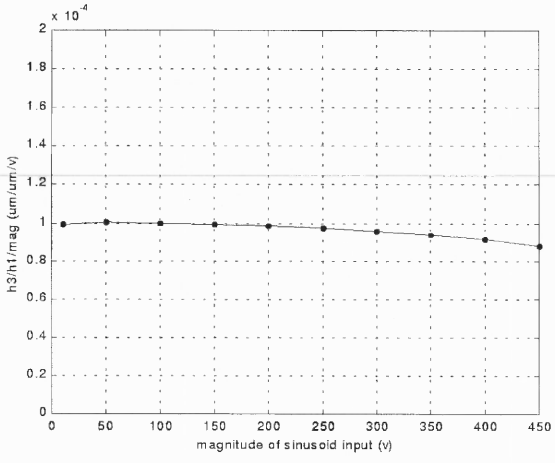

Figure 4.10 Simulation: Weak scalability

4.2 Effects of Hysteresis Parameter Variation

In equations (1.10) and (1.11), the nonlinearity is parameterized by the scalars k_1 and F_c. It is of interest to determine the effects of varying k_1 and F_c on the hysteresis loops as well as the normalized ratio N_R.

Figure 4.11 shows the hysteresis loops with different values of k_1, where $F_c = 7.5 \times 10^{-6}$ and equation (4.1) is used. As k_1 increases from 10^6 to 4.5×10^7, the

Magnitude ($Volts$)	Normalized ratio ($\mu m/\mu m/Volts$)	
	experiment	simulation
10	5.3852×10^{-4}	9.9396×10^{-5}
50	1.3849×10^{-4}	1.0001×10^{-4}
100	1.1624×10^{-4}	9.9694×10^{-5}
150	9.6930×10^{-5}	9.9105×10^{-5}
200	8.4688×10^{-5}	9.8279×10^{-5}
250	7.5807×10^{-5}	9.7166×10^{-5}
300	6.9780×10^{-5}	9.5686×10^{-5}
350	6.5755×10^{-5}	9.3751×10^{-5}
400	6.4016×10^{-5}	9.1284×10^{-5}
450	6.8610×10^{-5}	8.8259×10^{-5}

Table 4.1 Weak scalability

range of output response decreases, while the area of hysteresis loop first increases and then decreases. The normalized ratios are plotted in Figure 4.13. It is observed that the normalized ratio provides a concise and quantized measure of the effects of k_1.

Similarly, the parameter F_c is varied from 5×10^{-7} to 3×10^{-5}. The hysteresis loops and the normalized ratios are shown in Figures 4.12 and 4.14 respectively where $k_c = 1.1 \times 10^7$ and equation (4.1) is used. Again it is observed that the variation of the area of the hysteresis loops and that of the normalized ratio are closely matched.

4.3 Summary

In this chapter, both the experimental and simulated results are applied to study the characteristics of the hysteresis in the piezoelectric actuator. In time domain, the hysteresis expresses itself as an elliptic shape, and the area of the hysteresis loop increases as the magnitude of input sinusoid increases. In frequency domain, the first harmonics are propositional to the magnitudes of the input sinusoid, and the third harmonics are proportional to the square of the magnitudes. Frequency analysis also shows that the hysteresis possesses a certain degree of weak scalability.

The similarity between the experimental and simulated results also verifies that the model can predict the behavior of the hysteresis very well. Hence quantitative measure method of hysteresis is proposed using the normalized ratio based on simulation. It can be applied to measure the level of hysteresis nonlinearity and the effect of hysteresis compensation.

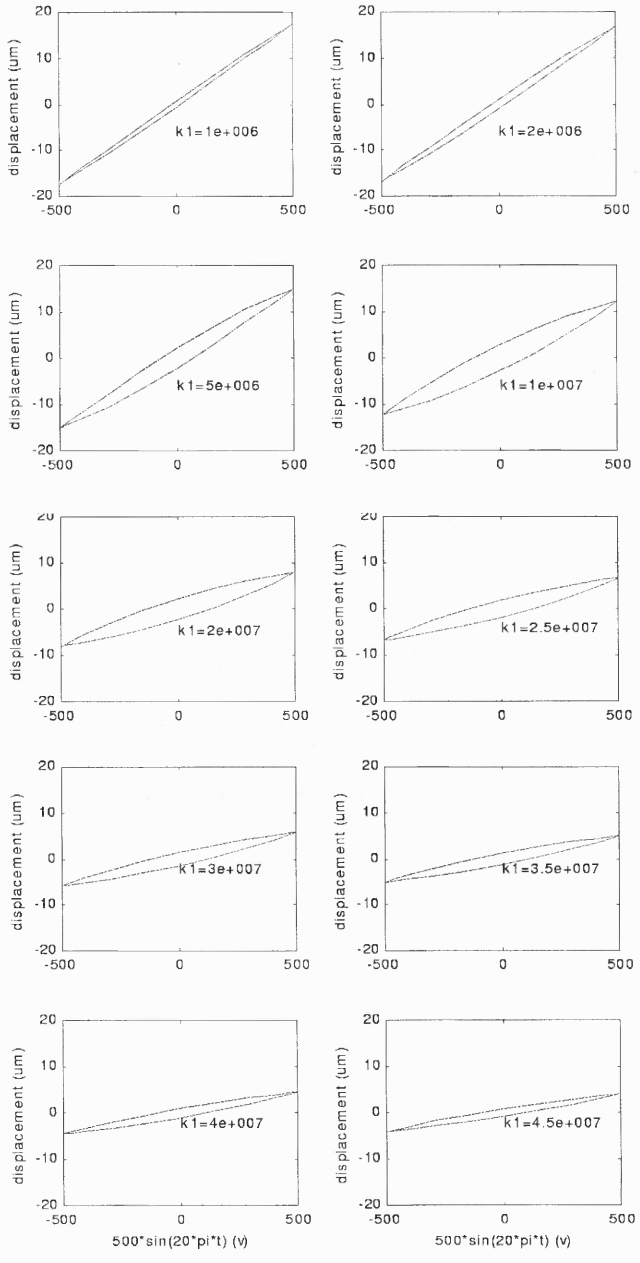

Figure 4.11 Simulation: Effect of k_1 on hysteresis loops (at $10Hz$)

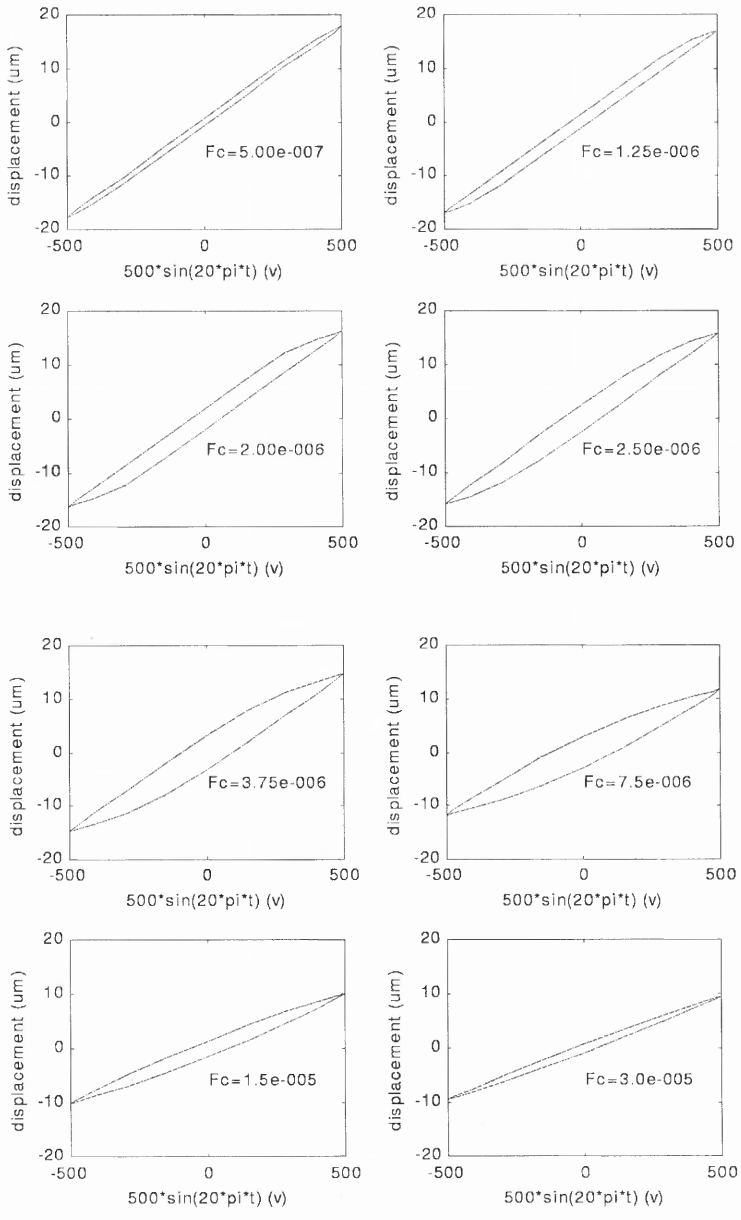

Figure 4.12 Simulation: Effect of F_c on hysteresis loops (at $10Hz$)

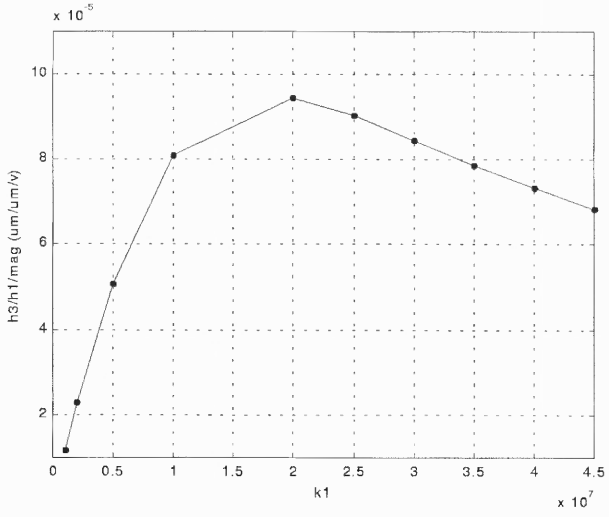

Figure 4.13 Simulation: Level of nonlinearity (at $10Hz$)

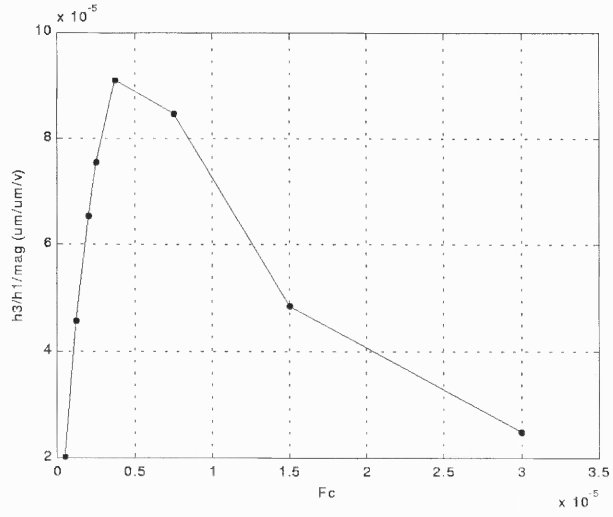

Figure 4.14 Simulation: Level of nonlinearity (at $10Hz$)

CHAPTER 5

ANALYTIC EXPRESSIONS OF HYSTERESIS HARMONICS

In Chapter 4, hysteresis characteristics in piezoelectric actuators have been observed from the experimental and simulated results. The analytic expressions of the hysteresis harmonics will be derived in this chapter. Not only do the harmonic expressions provide a theoretical explanation on the observed phenomena, but they can also be applied to estimate the hysteresis parameters.

5.1 Analytic Expressions of the Hysteresis Harmonics

In the following, Perturbation Method [24] is applied to derive the analytic expressions of the hysteresis harmonics. Perturbation Method is an iterative method such that the results of iteration N are applied in iteration $N+1$ for further calculation. The following assumptions are imposed to simplify the iterative process:

Assumptions:

1. $F(0)$ is chosen such that the DC term of $F(t)$ is zero
2. $\dot{x}(0) = x(0) = 0$
3. The sign of dx/dt is the same as its fundamental term

Assumption 1 removes the effects of the secular terms. Assumption 3 is applied to facilitate the calculation of the Fourier Series of $\left|\frac{dx}{dt}\right|$. For instance, let

$$\frac{dx}{dt} = h_1 \sin(\omega t + \varphi_1) + h_3 \sin(3\omega t + \varphi_3)$$

where h_1 and h_3 are the amplitudes of the first and third harmonics, respectively. The Fourier Series expression of $\left|\frac{dx}{dt}\right|$ is computed as,

$$\left|\frac{dx}{dt}\right| = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos(n\omega t) + b_n \sin(n\omega t))$$

$$a_n = \frac{2}{T}\int_a^{a+T}\left|\frac{dx}{dt}\right|\cos(n\omega t)dt \quad (n \geq 0)$$

$$b_n = \frac{2}{T}\int_a^{a+T}\left|\frac{dx}{dt}\right|\sin(n\omega t)dt \quad (n \geq 1)$$

From Assumption 3, we have $sgn(\frac{dx}{dt}) = sgn(sin(\omega t + \varphi_1))$, where $sgn()$ is defined in equation (1.2). Applying the relationship, a_n and b_n can be calculated as

$$\begin{aligned} a_n &= \frac{\omega}{\pi}\int_{-\varphi_1/\omega}^{(\pi-\varphi_1)/\omega}[h_1 sin(\omega t + \varphi_1) + h_3 sin(\omega t + \varphi_3)]\cos(n\omega t)dt - \\ &\quad \frac{\omega}{\pi}\int_{(\pi-\varphi_1)/\omega}^{(2\pi-\varphi_1)/\omega}[h_1 sin(\omega t + \varphi_1) + h_3 sin(\omega t + \varphi_3)]\cos(n\omega t)dt \\ b_n &= \frac{\omega}{\pi}\int_{-\varphi_1/\omega}^{(\pi-\varphi_1)/\omega}[h_1 sin(\omega t + \varphi_1) + h_3 sin(\omega t + \varphi_3)]\sin(n\omega t)dt - \\ &\quad \frac{\omega}{\pi}\int_{(\pi-\varphi_1)/\omega}^{(2\pi-\varphi_1)/\omega}[h_1 sin(\omega t + \varphi_1) + h_3 sin(\omega t + \varphi_3)]\sin(n\omega t)dt \end{aligned}$$

The assumption is not restrictive since $h_1 \gg h_3$.

5.1.1 The First Iteration

Considering equations (1.10) and (1.11) with input expressed by $u = mag\sin(\omega t)$, where mag and ω stand for the magnitude and the frequency of the sinusoid, respectively. The first iteration is derived by ignoring the second term in the right hand side of equation (1.11), as below.

$$F_0 = x_0 + C_0 \tag{5.1}$$

$$\frac{d^2x_0}{dt^2} + \gamma\frac{dx_0}{dt} + k_n x_0 = A\sin(\omega t) - k_1 F_0 \tag{5.2}$$

where C_0 is the constant to eliminate the DC term in x_0, and $A = k_v mag$. Since the DC term of x_0 has no effect on the next iteration, and the analysis is focused on harmonic derivation, the DC terms of x_0 and F_0 are ignored and the iteration results become,

$$x_0(t) = \frac{A(k_n + k_1 - \omega^2)\sin(\omega t) - A\omega\gamma\cos(\omega t)}{k_n^2 + 2k_n k_1 + k_1^2 + (\gamma^2 - 2k_n - 2k_1)\omega^2 + \omega^4} \tag{5.3}$$

$$F_0 = x_0 \tag{5.4}$$

5.1.2 The Second Iteration

The second iteration is processed by adding the nonlinear term in equation (1.11), as shown in equations (5.5) and (5.6) below,

$$\frac{dF_1}{dt} = \frac{dx_0}{dt} - \frac{F_0}{F_c}\left|\frac{dx_0}{dt}\right| \tag{5.5}$$

$$\frac{d^2x_1}{dt^2} + \gamma\frac{dx_1}{dt} + k_n x_1 = A\sin(\omega t) - k_1 F_1 \tag{5.6}$$

From equation (5.5), it is observed that if F_0 has DC term, F_1 will include the secular term $C \times t$, where t is time and C is a constant. This is because $\left|\frac{dx_0}{dt}\right|$ contains DC term and such there is constant term in the right hand side of equation (5.5). Hence the system becomes unstable. The application of Assumption 1 removes this secular effect. For convenience, x_0 in equation (5.3) is expressed as,

$$x_0(t) = A_0\sin(\omega t + \varphi) \tag{5.3'}$$

And its derivative is given by,

$$\frac{dx_0(t)}{dt} = \omega A_0\sin(\omega t + \varphi + \frac{\pi}{2})$$

To solve equation (5.5), $\left|\frac{dx_0}{dt}\right|$ is approximated by its Fourier Series expression,

$$\left|\frac{dx_0}{dt}\right| \approx \frac{6\omega A_0 + 4\omega A_0\cos(2\varphi + 2\omega t)}{3\pi}$$

Ignoring the DC terms of x_1 and F_1, the second iteration can be obtained as,

$$x_1(t) = x_{11}\cos(\omega t) + x_{12}\sin(\omega t) + x_{31}\cos(3\omega t) + x_{32}\sin(3\omega t) \tag{5.7}$$

where

$$x_{11} = \frac{-A\omega\gamma F_c\pi + k_1(4/3A_0^2\omega^2 + A_0\gamma F_c\pi\omega - 4/3A_0^2 k_n)\cos(\varphi) + k_1(A_0\omega^2 F_c\pi - A_0 F_c\pi k_n - 4/3A_0^2\gamma\omega)\sin(\varphi)}{F_c\pi(\omega^2\gamma^2 + k_n^2 - 2k_n\omega^2 + \omega^4)}$$

$$x_{12} = \frac{AF_c\pi(k_n - \omega^2) + k_1(A_0\omega^2 F_c\pi - A_0 F_c\pi k_n - 4/3A_0^2\gamma\omega)\cos(\varphi) + k_1(4/3A_0^2 k_n - 4/3A_0^2\omega^2 - A_0 F_c\pi\gamma\omega)\sin(\varphi)}{F_c\pi(\omega^2\gamma^2 + k_n^2 - 2k_n\omega^2 + \omega^4)}$$

$$x_{31} = \frac{k_1 A_0^2(-2/3\omega\gamma\sin(3\varphi) + (2\omega^2 - 2/9k_n)\cos(3\varphi))}{F_c\pi(9\omega^2\gamma^2 + k_n^2 - 18k_n\omega^2 + 81\omega^4)}$$

$$x_{32} = \frac{k_1 A_0^2((2\omega^2 - 2/9k_n)\sin(3\varphi) - 2/3\omega\gamma\cos(3\varphi))}{F_c\pi(9\omega^2\gamma^2 + k_n^2 - 18k_n\omega^2 + 81\omega^4)} \tag{5.8}$$

And

$$F_1(t) = A_0\sin(\omega t + \varphi) + \frac{4A_0^2}{3F_c\pi}\cos(\omega t + \varphi) + \frac{2A_0^2}{9F_c\pi}\cos(3\omega t + 3\varphi) \tag{5.9}$$

5.1.3 The Third Iteration

Similar to the second iteration, the third iteration starts from the equations below,

$$\frac{dF_2}{dt} = \frac{dx_1}{dt} - \frac{F_1}{F_c}\left|\frac{dx_1}{dt}\right| \tag{5.10}$$

$$\frac{d^2x_2}{dt^2} + \gamma\frac{dx_2}{dt} + k_n x_2 = A\sin(\omega t) - k_1 F_2 \tag{5.11}$$

with x_1 expressed as

$$x_1(t) = h_1\sin(\omega t + \varphi_1) + h_3\sin(3\omega t + \varphi_3) \tag{5.7'}$$

and $\frac{dx_1}{dt}$ as

$$\frac{dx_1(t)}{dt} = \omega h_1\sin(\omega t + \varphi_1 + \frac{\pi}{2}) + 3\omega h_3\sin(3\omega t + \varphi_3 + \frac{\pi}{2})$$

$\left|\frac{dx_1}{dt}\right|$ is approximated by its Fourier Series as below,

$$\begin{aligned}\left|\frac{dx_1}{dt}\right| \approx\ & 0.6366h_1 + 0.2122h_3 cos(3\varphi_1 - \varphi_3) + [-0.4244h_1\cos(2\varphi_1) \\ & +0.1273h_3\cos(5\varphi_1 - \varphi_3) + 0.6366h_3\cos(\varphi_1 - \varphi_3)]\cos(2\omega t) \\ & +[0.4244h_1\sin(2\varphi_1) - 0.1273h_3\sin(5\varphi_1 - \varphi_3) \\ & +0.6366h_3\sin(\varphi_1 - \varphi_3)]\sin(2\omega t) + [-0.0849h_1\cos(4\varphi_1) \\ & +0.0909h_3\cos(7\varphi_1 - \varphi_3) - 0.6366h_3\cos(\varphi_1 + \varphi_3)]\cos(4\omega t) \\ & +[0.0849h_1\sin(4\varphi_1) - 0.0909h_3\cos(7\varphi_1 - \varphi_3) \\ & +0.6366h_3\sin(\varphi_1 + \varphi_3)]\sin(4\omega t)\end{aligned}$$

The third iteration result can be expressed as,

$$\begin{aligned} x_2(t) &= x_{11}\cos(\omega t) + x_{12}\sin(\omega t) + x_{31}\cos(3\omega t) + x_{32}\sin(3\omega t) + x_{51}\cos(5\omega t) \\ &\quad + x_{52}\sin(\omega t) + x_{71}\cos(7\omega t) + x_{72}\sin(7\omega t) \end{aligned} \tag{5.12}$$

Since the expressions of the parameters in $x_2(t)$ are very complex, they are not listed. The iteration program in Matlab is shown in Appendix F.

Comparing the expressions of the first harmonic in equations (5.3) and (5.8), it is not difficult to observe that the order of A_0 is one in x_0, and two in x_1. Further analysis on the third iteration in equation (5.10) shows that even higher order terms of A_0 exist in x_2. Hence, the smaller A_0 is, the less effects the higher order terms of A_0 have on the iteration results, and more accurate results can be achieved. It should also be noted that in equation (5.8), the coefficients related to A_0^2 might not be accurate since A_0^2 appears for the first time. Hence, the modified second iteration results are given,

$$x_{11} = \frac{-A\omega\gamma F_c\pi + k_1(4/3k_aA_0^2\omega^2 + A_0\gamma F_c\pi\omega - 4/3k_aA_0^2k_n)\cos(\varphi) + k_1(A_0\omega^2F_c\pi - A_0F_c\pi k_n - 4/3k_aA_0^2\gamma\omega)\sin(\varphi)}{F_c\pi(\omega^2\gamma^2 + k_n^2 - 2k_n\omega^2 + \omega^4)}$$

$$x_{12} = \frac{AF_c\pi(k_n - \omega^2) + k_1(A_0\omega^2F_c\pi - A_0F_c\pi k_n - 4/3k_aA_0^2\gamma\omega)\cos(\varphi) + k_1(4/3k_aA_0^2k_n - 4/3k_aA_0^2\omega^2 - A_0F_c\pi\gamma\omega)\sin(\varphi)}{F_c\pi(\omega^2\gamma^2 + k_n^2 - 2k_n\omega^2 + \omega^4)}$$

$$x_{31} = \frac{k_1k_bA_0^2(-2/3\omega\gamma\sin(3\varphi) + (2\omega^2 - 2/9k_n)\cos(3\varphi))}{F_c\pi(9\omega^2\gamma^2 + k_n^2 - 18k_n\omega^2 + 81\omega^4)}$$

$$x_{32} = \frac{k_1k_bA_0^2((2\omega^2 - 2/9k_n)\sin(3\varphi) - 2/3\omega\gamma\cos(3\varphi))}{F_c\pi(9\omega^2\gamma^2 + k_n^2 - 18k_n\omega^2 + 81\omega^4)} \tag{5.8'}$$

where k_a and k_b can be obtained by curve fitting.

5.2 Hysteresis Harmonic Expressions at Low Frequencies

In this section, the relations among the coefficients in equations (1.10) and (1.11) at low frequencies are derived first. The results are then applied to simplify the iteration results in equations (5.3) and (5.8').

Now since the piezoelectric actuators are underdamped and their bandwidth is on the order of several hundred hertz, i.e.,

$$\omega_n \geq 1000 \quad \varsigma < \frac{1}{\sqrt{2}} \tag{5.13}$$

where ω_n and ς are the natural frequency in rad/sec and the damping ratio, respectively. Hence, we have

$$\frac{k_n}{\gamma} = \frac{\omega_n}{2\varsigma} > \frac{\omega_n}{\sqrt{2}}$$

For

$$\omega \leq 100/\sqrt{2}\ rad/\sec \tag{5.14}$$

$$\frac{k_n}{\gamma} \gg \omega \tag{5.15}$$

where ω stands for the frequency of the input sinusoid.

5.2.1 Simplified First Iteration

Applying equations (5.14) and (5.15) to (5.3), it can be achieved,

$$x_0(t) = A_0 \sin(\omega t) \tag{5.16}$$

where $A_0 = \frac{A}{k_n + k_1}$ is the first harmonic or fundamental term.

5.2.2 Simplified Second Iteration

Similarly, applying equations (5.14) and (5.15) to (5.8'), we obtain

$$\begin{aligned} x_{11} &= \frac{-k_n A_0 \omega \gamma F_c \pi - 4/3 k_1 k_a A_0^2 k_n}{F_c \pi k_n^2} \\ x_{12} &= \frac{k_n A_0 F_c \pi k_n - 4/3 k_1 k_a A_0^2 \gamma \omega}{F_c \pi k_n^2} \\ x_{31} &= \frac{-2/9 k_1 k_b A_0^2}{F_c \pi k_n} \\ x_{32} &= \frac{-2/3 k_1 k_b A_0^2 \omega \gamma}{F_c \pi k_n^2} \end{aligned} \tag{5.17}$$

The first harmonic is calculated from equation (5.17) as

$$h_1 = \sqrt{x_{11}^2 + x_{12}^2} = \frac{A_0 \sqrt{(3F_c \pi k_n)^2 + (4k_a A_0 k_1)^2}}{3F_c \pi k_n} \tag{5.18}$$

Considering x_{31} and x_{32} in equation (5.17), since

$$\frac{x_{32}}{x_{31}} = \frac{-2/3k_1A_0^2\omega\gamma}{-2/9k_1A_0^2k_n} \to 0$$

the third harmonic can be written as

$$h_3 = \frac{2k_1k_bA_0^2}{9F_c\pi k_n} \tag{5.19}$$

Hence, we have

$$x_1(t) = h_1\sin(\omega t) - h_3\cos(3\omega t) \tag{5.20}$$

The above results can be summarized as:

Remarks: For the present application,

1. $k_n \gg \omega\gamma$ for input frequency $\omega \leq 100/\sqrt{2}\ rad/sec$

2. The first and third harmonics are approximated as,

$$h_1 = \frac{A_0\sqrt{(3F_c\pi k_n)^2+(4k_aA_0k_1)^2}}{3F_c\pi k_n} \quad h_3 = \frac{2k_1k_bA_0^2}{9F_c\pi k_n} \tag{5.21}$$

   where $A_0 = A/(k_n + k_1)$.

3. h_3 is proportional to A^2; and h_1 is proportional to A if $A \ll \frac{k_nF_c(k_n+k_1)}{k_ak_1}$ is satisfied.

   The results can be derived from Remark 2 directly by replacing A_0 by $\frac{A}{k_n+k_1}$. Experimental results in Chapter 4 show that the first harmonic is approximately proportional to the magnitude of input sinusoid. Hence, in piezoelectric actuators, A is much smaller than $\frac{k_nF_c(k_n+k_1)}{k_ak_1}$.

4. For $A \ll \frac{k_nF_c(k_n+k_1)}{k_ak_1}$, the normalized ratio can be expressed as

$$N_R = \frac{2k_1k_bk_v}{9F_c\pi k_n(k_1+k_n)}$$

   where k_v is shown in equation (1.10).

5.3 Example

In the following example, the system in equations (1.10) and (1.11) with parameters described by equations (4.1) and (4.2) is applied to calculate the hysteresis harmonics. The frequency of the input sinusoid is $1Hz$, and the magnitude is varied from $10Volts$ to $600Volts$ in nine steps, i.e.,

$$mag = \begin{bmatrix} 10 & 20 & 50 & 100 & 200 & 300 & 400 & 500 & 600 \end{bmatrix} (Volts)$$

Figure 5.1 shows the first and the third harmonics obtained from equation (5.3) for the first iteration, (5.8) for the second iteration, and from simulation, where $A = k_v \times mag$. It can be observed that the first harmonics for the three cases are close to each other, and as expected the second iteration has better results than the first one. For the third harmonics, however large error exists between the second iteration and the simulation, and the error increases as the magnitude of the input sinusoid increases. The maximum relative errors of the first and the third harmonics between the second iteration and the simulation are 15.254% and 54.815%, respectively.

Choosing $k_a = 1.6$ and $k_b = 0.66$, the modified iteration results are shown in Figure 5.2. The results of the simplified and the original iteration are both closely matched with the simulation data.

5.4 Summary

Perturbation Method has been applied to derive the analytic expressions of the hysteresis harmonics. For piezoelectric actuators, the results are simplified and summarized in the Remarks. The Remarks also summarize the hysteresis characteristics in frequency domain for frequency less than $10Hz$. Hence, The explicit expressions of the harmonics give a theoretical explanation on the experimental phenomena observed in Chapter 4.

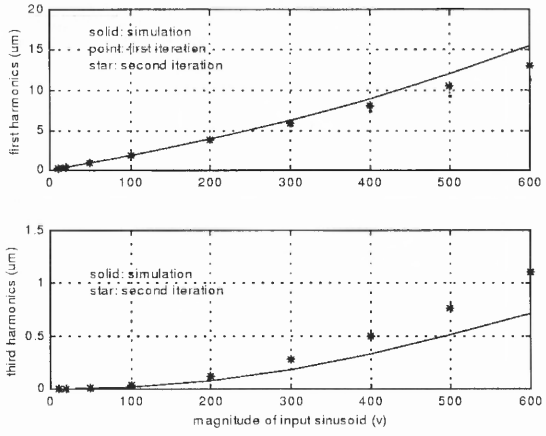

Figure 5.1 Comparison between iterations and simulation

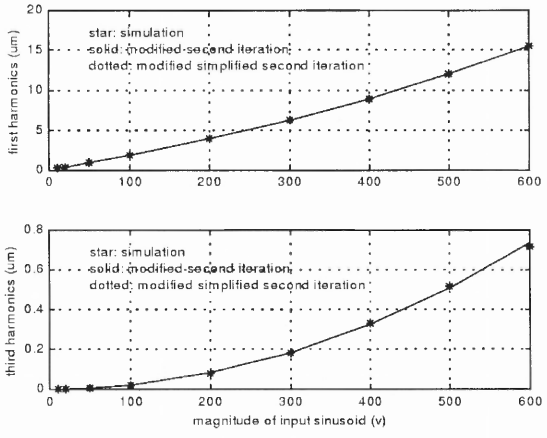

Figure 5.2 Comparison between modified iterations and simulation

To improve the accuracy of the second iteration results, gains k_a and k_b are introduced. The accurate expression makes it possible to study the influence of frequency on hysteresis parameters, which will be discussed later in Chapter 6.

CHAPTER 6

PIEZOELECTRIC ACTUATOR MODEL

In this chapter, the modeling effort is applied to the piezoelectric actuator. The effects of frequency on the hysteresis parameter estimation are then analyzed.

6.1 Modeling and Compensating the Nonlinear Scale Factor

The parameters γ and k_n are readily determined by the standard step response method. It remains to account for the nonlinear scale factor and hysteresis effects. The nonlinear scale factor is caused by the deformation of the orthogonal piezoelectric members (e.g. the y-axis element when x-axis is actuated). In the following, polynomial curve fitting is applied to model the scale factor, and a gain compensator is designed to eliminate the nonlinear effect.

With a $0.05Hz$ sinusoidal excitation, and magnitude ranging from $10Volts$ to $500Volts$ uniformly 50 steps, Figure 6.1 shows the relationship between the magnitudes of the input and output responses, where the sampling rate is $5Hz$. Defining the scale factor as the ratio of the magnitude of the output to that of the input, it is clearly seen that the scale factor is nonlinear. Curve fitting shows that the scale factor can be expressed as,

$$v = C_4x^4 + C_3x^3 + C_2x^2 + C_1x \tag{6.1}$$

where $C_1 = 73.353$, $C_2 = -7.3374$, $C_3 = 0.67792$ and $C_4 = -0.024937$; x and v represent the magnitudes of the output response in *meters* and the input sinusoid in $Volts$, respectively.

Figure 6.2 shows the block diagram of the system compensated for the nonlinear scale factor, where the gain compensator is designed such that

$$v = f(u)u \qquad x = \tfrac{u}{100} \times 2.5 \times 10^{-6} \tag{6.2}$$

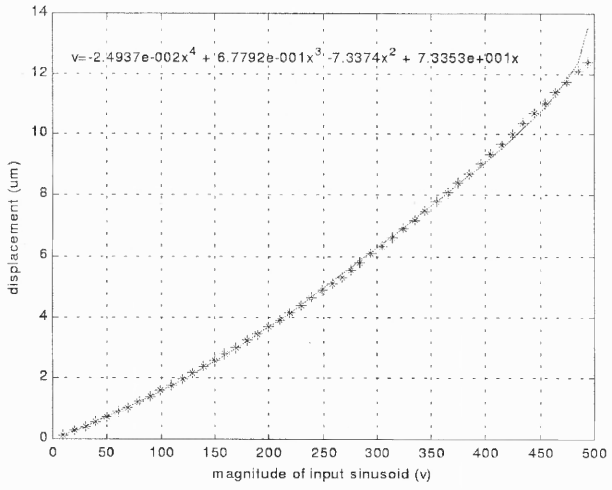

Figure 6.1 Scale factor of the uncompensated system and polynomial fit

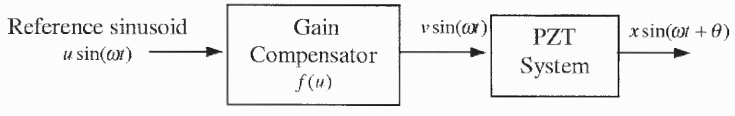

Figure 6.2 Block diagram of the nonlinear gain compensation

where u is the magnitude of the input sinusoid in $Volts$, and $f(u)$ stands for the function of the nonlinear gain compensator. Substituting x and v in equation (6.2) to (6.1),

$$f(u) = (2.5\times10^{-8})^4\times C_4u^3+(2.5\times10^{-8})^3\times C_3u^2+(2.5\times10^{-8})^2\times C_2u+2.5\times10^{-8}\times C_1 \tag{6.3}$$

For x in μm, equation (6.2) is expressed as,

$$v = f(u)u \qquad x = \frac{u}{100}\times 2.5 \tag{6.2'}$$

$$f(u) = (0.025)^4\times C_4u^3 + (0.025)^3\times C_3u^2 + (0.025)^2\times C_2u + 0.025\times C_1 \tag{6.3'}$$

Figure 6.3 shows the compensated gain applied in this system, and the linear relationship between the magnitudes of the reference input and those of the output of the compensated system is obtained, as shown in Figure 6.4.

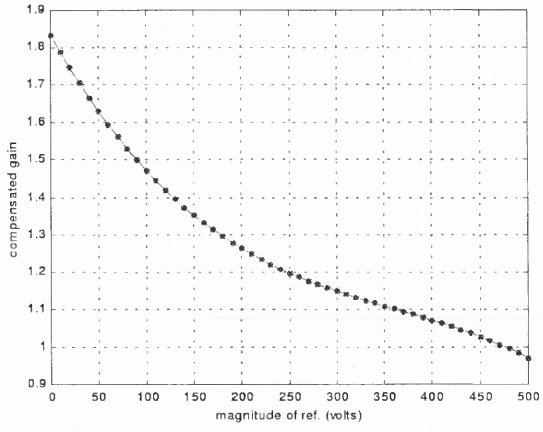

Figure 6.3 Compensated gain (x in μm)

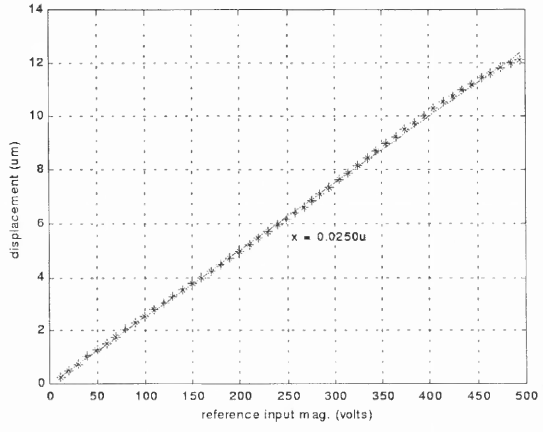

Figure 6.4 Scale factor of the compensated system and polynomial fit

6.2 Off-Line Hysteresis Parameter Estimation

In the following, the analytic expressions of the first and the third harmonics in Chapter 5 are applied in hysteresis parameter estimation. Rewrite the analytic expressions in equation (5.21) as

$$h_1 = A_0\sqrt{1+\left(\frac{4k_aA_0k_1}{3k_nF_c\pi}\right)^2} \tag{6.4}$$

$$F_c = \frac{2k_b k_1 A_0^2}{9\pi k_n h_3} \tag{6.5}$$

where

$$A_0 = \frac{A}{k_1 + k_n} \tag{6.6}$$

Substituting equation (6.5) to (6.4) and simplifying the result, the following equation is obtained:

$$A_0 = \sqrt{h_1^2 - \left(\frac{6k_a h_3}{k_b}\right)^2} \tag{6.7}$$

Letting $k_a = k_b = 1$, k_1 and F_c can be obtained by first calculating A_0 in (6.7), then solving (6.6) and (6.5). To obtain better estimated results, k_1 and F_c are then adjusted such that the simulated hysteresis loop is approach to the experimental one. Knowing k_1 and F_c, k_a and k_b can then be obtained such that the first and the third harmonics in simulation are sufficiently close to those of the experiment.

The systemic method to obtain k_1, F_c, k_a, k_b is as follows:

1. Apply polynomial curve fit between h_1^2 and A^2, where h_1^2 and A^2 have the relationship of $h_1^2 = (\frac{A}{k_1+k_n})^2 + \left(\frac{4k_a k_1}{3k_n F_c \pi}(\frac{A}{k_1+k_n})^2\right)^2$ from equations (6.4) and (6.6).

2. Calculate k_1 and $\frac{F_c}{k_a}$ as:

$$k_1 = 1/\sqrt{C_2} - k_n \tag{6.8}$$

$$\frac{F_c}{k_a} = \frac{4k_1}{3k_n \pi (k_1 + k_n)^2 \sqrt{C_1}} \tag{6.9}$$

   where C_1 and C_2 are the coefficients of the curve fit polynomial.

3. Calculate A_0 from equation (6.6) using k_1 obtained in equation (6.8)

4. Apply polynomial curve fit between h_3 and A_0^2, and calculate $\frac{F_c}{k_b}$ from equation (6.5),

$$\frac{F_c}{k_b} = \frac{2k_1}{9\pi k_n C_3} \tag{6.10}$$

   where C_3 is the coefficient of the curve fitting.

	First harmonics (μm)	Third harmonics (μm)
Experiment	11.250	0.34733
Simulation	10.400	0.41305

Table 6.1 Experimental and simulated harmonics

5. Apply polynomial curve fitting between $h_1^2 - A_0^2$ and h_3^2, and calculate $\frac{k_a}{k_b}$ from equation (6.7):

$$\frac{k_a}{k_b} = \frac{\sqrt{C_4}}{6} \tag{6.11}$$

where C_4 is the coefficient of the curve fitting.

Hence k_1, F_c, k_a, k_b are obtained by solving equations (6.8)-(6.11).

Figure 6.5 shows the experimental and simulated hysteresis loops with hysteresis parameters shown in equation (4.2) and $k_a = 1.6$, $k_b = 0.66$. The frequency and the magnitude of the reference sinusoid are $10Hz$ and $450Volts$, respectively. The harmonics are tabulated in Table 6.1, where it is noted that a good fit is achieved in both time and frequency domains.

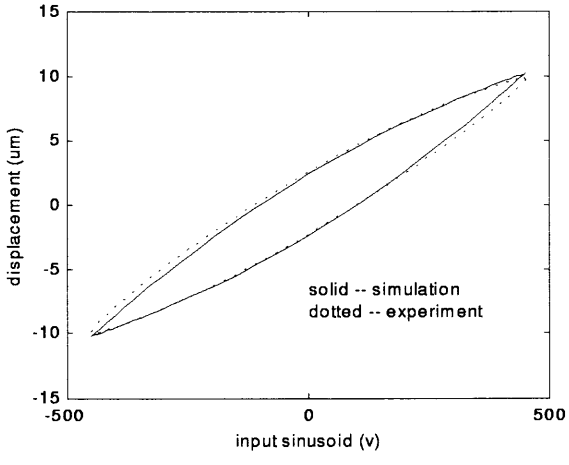

Figure 6.5 Hysteresis loop fitting using the estimated parameters (at $10Hz$)

6.3 Effect of Frequency on Hysteresis Parameter Estimation

In the following, the modified second iteration results are applied to calculate the hysteresis parameters when the frequency of the sinusoidal excitation is varied from $0.2Hz$ to $100Hz$ in 11 steps, i.e.,

$$\left[\begin{array}{ccccccccccc} 0.2 & 0.5 & 1 & 2 & 5 & 10 & 20 & 40 & 60 & 80 & 100 \end{array}\right] Hz$$

Three sets of data are obtained corresponding to the magnitude of the sinusoid 100, 200 and $300Volts$. Figure 6.6 shows the hysteresis loops corresponding to 0.2 and $100Hz$. The hysteresis loops with frequency between 0.2 and $100Hz$ fall to the area bounded by the two loops. In general, the variation of the hysteresis loop is small. The relationships between the first harmonics and the frequency, and the third harmonics and the frequency are shown in Figure 6.7. It is observed that the first and the third harmonics are approximately constant in the frequency range of $[0.2\ 100]Hz$ and $[0.2\ 40]Hz$, respectively.

Define the relative errors of the estimated hysteresis parameters as

$$\begin{aligned} e_{a\,\max} &= \frac{\max\{abs\,[a - a_{true}]\}}{a_{true}} \\ e_{a\,\min} &= \frac{\min\{abs\,[a - a_{true}]\}}{a_{true}} \\ e_{aavg} &= \frac{mean\,\{abs\,[a - a_{true}]\}}{a_{true}} \end{aligned} \tag{6.12}$$

where a_{true} stands for the true value of a parameter, and a is the vector containing the estimated values of the parameter corresponding to different frequencies; $max(\cdot)$, $min(\cdot)$, $mean(\cdot)$, $abs(\cdot)$ are functions to calculate the vector's maximum, minimum, average and absolute values respectively. The relative errors of the estimated parameters are listed in Table 6.2, where the maximum value is about 10%.

Figures 6.8 and 6.9 show the experimental results of the effect of frequency on the hysteresis harmonics. Similar conclusions as those of the simulated results can

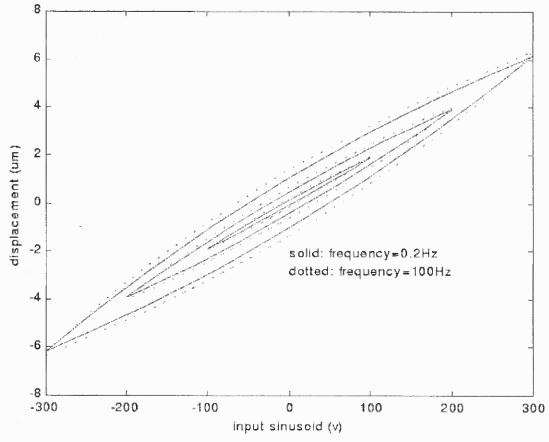

Figure 6.6 Simulation: Effect of frequency on hysteresis loops

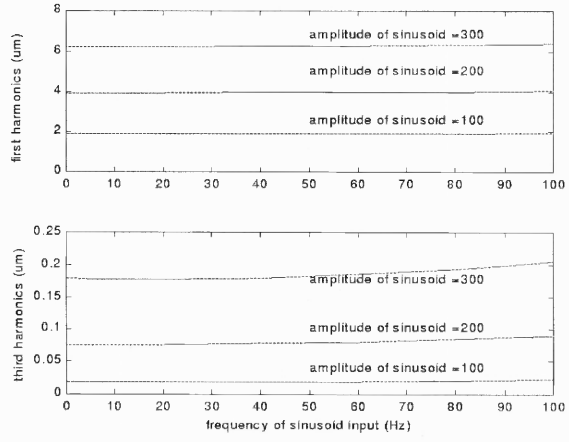

Figure 6.7 Simulation: Effect of frequency on hysteresis harmonics

be reached. Define the relative errors as

$$\begin{aligned} e_{a\max} &= \frac{\max\{abs[a - mean(a)]\}}{mean(a)} \\ e_{a\min} &= \frac{\min\{abs[a - mean(a)]\}}{mean(a)} \\ e_{aavg} &= \frac{mean\{abs[a - mean(a)]\}}{mean(a)} \end{aligned} \tag{6.13}$$

The relative estimated errors of the hysteresis parameters are calculated and listed in Table 6.3. The maximum relative mean errors of k_1 and F_c are also within 10%.

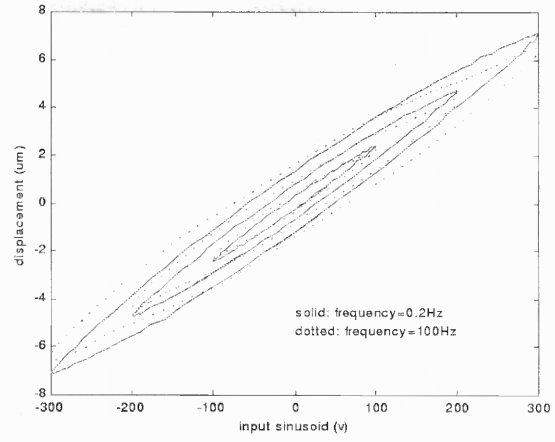

Figure 6.8 Experiment: Effect of frequency on hysteresis loops

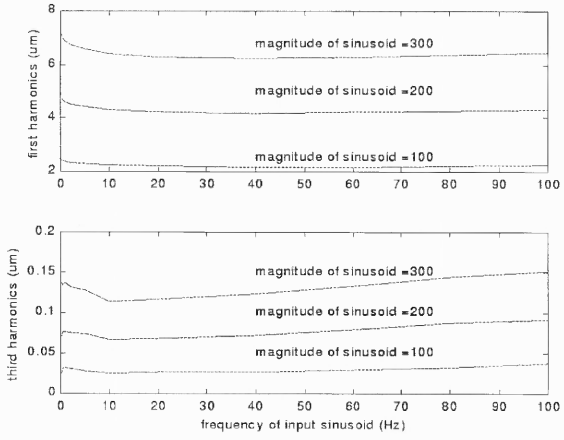

Figure 6.9 Experiment: Effect of frequency on hysteresis harmonics

6.4 Summary

In this chapter, the models of the piezoelectric actuator system are estimated from the experimental data. Step response is applied in modeling the actuator dynamics, and polynomial curve fitting in modeling the nonlinear scale factor. The explicit expressions of hysteresis parameters are derived, and off-line estimation procedure is given.

Furthermore, nonlinear scale factor compensator is proposed such that the scale factor of the compensated system is constant.

$mag(Volts)$	e_{k_1} max	e_{k_1} min	$e_{k_1 avg}$
100	2.7556×10^{-2}	2.9234×10^{-3}	8.0090×10^{-3}
200	1.2362×10^{-2}	6.4419×10^{-3}	8.1744×10^{-3}
300	1.3607×10^{-2}	1.2558×10^{-3}	8.5894×10^{-3}

$mag(Volts)$	e_{F_c} max	e_{F_c} min	$e_{F_c avg}$
100	9.6562×10^{-2}	2.9420×10^{-2}	7.0010×10^{-2}
200	9.1888×10^{-2}	1.2281×10^{-2}	5.4333×10^{-2}
300	1.0295×10^{-1}	1.2043×10^{-2}	3.6822×10^{-2}

Table 6.2 Simulation: Effect of frequency on hysteresis parameters

$mag(Volts)$	e_{k_1} max	e_{k_1} min	$e_{k_1 avg}$
100	0.15307	2.2073×10^{-2}	9.0997×10^{-2}
200	0.19213	1.4352×10^{-2}	9.8903×10^{-2}
300	0.20709	9.0032×10^{-3}	1.0076×10^{-1}

$mag(Volts)$	e_{F_c} max	e_{F_c} min	$e_{F_c avg}$
100	0.20843	1.8027×10^{-2}	9.9144×10^{-2}
200	0.16149	9.2930×10^{-3}	6.9683×10^{-2}
300	0.15865	5.3886×10^{-3}	7.5628×10^{-2}

Table 6.3 Experiment: Effect of frequency on hysteresis parameters

The modified second iteration results are applied to study the effect of frequency on hysteresis harmonics and hysteresis parameter estimation. Both simulated and experimental results show that the first and the third harmonics are approximately constant in the frequency range of $[0.2 \quad 100]Hz$ and $[0.2 \quad 40]Hz$, respectively. The maximum relative mean errors of the estimated hysteresis parameters are with 10%.

CHAPTER 7

NONLINEAR OBSERVER HYSTERESIS COMPENSATION

Although conventional closed loop controllers such as PI or PID can suppress the hysteresis effects [26], ignoring the hysteresis phenomenon may cause the closed loop system to be unstable if sufficient phase margin is not provided [27], since hysteresis causes nonlinear phase distortion.

Two kinds of hysteresis compensated schemes have been presented. In [18], an adaptive dither compensation method is presented. The idea is that a dither signal with a fast frequency can be applied to the system to reduce the hysteresis effect. The larger the magnitude of the dither signal, the better the compensated effect will be. The advantage of this method is that the hysteresis model is not needed. However, experimental tests in the monolithic piezoelectric actuator show that the method is not effective when the magnitude of the input sinusoid is greater than $50 Volts$.

The other compensation methods require hysteresis models. The motivation is that hysteresis models are applied to predict the hysteresis output which is applied to eliminate the actual hysteresis. [7] applied Maxwell model and proposed the scheme of a PID controller augmented with a feedback linearization loop. [13, 14] applied Preisach model and presented the scheme of a feedforward loop with a PID feedback controller. However, since PID can alleviate the effect of hysteresis, how well the compensated effect of either feedback linearization loop or a feed-forward loop is hard to measure. Furthermore, as the models of the hysteresis are not analytic, hysteresis estimation has to be processed off-line.

To compensate friction, [28] proposed friction observer algorithm based on an analytic friction model. The predicted friction output was applied to cancel the actual friction. Adaptive friction compensation schemes were further proposed in [29] and [30], where the friction parameter was estimated on-line.

The observer compensated method is applied in this work, where a reduced order nonlinear observer is proposed to predict the hysteresis output which is applied to compensate the hysteresis nonlinearity.

7.1 Discrete-Time State Space Model

Consider the continuous time model of the piezoelectric actuator in equations (1.10) and (1.11) with the estimated hysteresis parameters expressed by $\hat{k}_1$ and $\hat{F}_c$. Letting $x_1 = x$, $x_2 = \dot{x}$, the state space model can be expressed as,

$$\begin{aligned} \dot{X}(t) &= A_c X(t) + B_c U(t) \\ y(t) &= C_c X(t) \\ \dot{F} &= x_2 - \frac{F}{\hat{F}_c}|x_2| \end{aligned} \tag{7.1}$$

where

$$A_c = \begin{pmatrix} 0 & 1 \\ -k_n & -\gamma \end{pmatrix} \quad B_c = \begin{pmatrix} 0 & 0 \\ k_v & -\hat{k}_1 \end{pmatrix} \quad C_c = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad U = \begin{pmatrix} u \\ F \end{pmatrix}$$

The corresponding discrete time state space model can be obtained as

$$\begin{aligned} X(n+1) &= A_d X(n) + B_d U(n) \\ y(n) &= C_d X(n) \\ F(n+1) &= F(n) + T_s(x_2(n) - \frac{F(n)}{\hat{F}_c}|x_2(n)|) \end{aligned} \tag{7.2}$$

where

$$A_d = e^{A_c T_s} \qquad B_d = \int_0^{T_s} e^{A_c \tau} B_c d\iota \qquad C_d = C_c$$

and T_s is sampling time.

7.2 Nonlinear Observer Hysteresis Compensator

Equation (7.2) is a third-order nonlinear system which makes it difficult to estimate the states directly. [9] avoided the problem by replacing $\dot{x}$ with the known $\dot{u}$.

However, how well the model can predict the hysteresis phenomena was not discussed.

In the present work, the reduced order nonlinear observer is proposed so that state x_2 is first estimated using the reduced order observer in which F is as an input, then F is calculated using the estimated x_2. Hence,

$$\hat{x}_2(n+1) = (a_{22} - l_r a_{12})\hat{x}_2(n) + l_r x_1(n+1) + (a_{21} - l_r a_{11})x_1(n) + (b_{21} - l_r b_{11})u(n) + (b_{22} - l_r b_{12})\hat{F}(n) \tag{7.3}$$

$$\hat{F}(n+1) = \hat{F}(n) + T_s(\hat{x}_2(n) - \frac{\hat{F}(n)}{\hat{F}_c}|\hat{x}_2(n)|)$$

where l_r is computed such that $(a_{22} - l_r a_{12})$ has the desired observer pole; a_{ij} and b_{ij} are the elements of A_d and B_d, respectively; $\hat{x}_2$ and $\hat{F}$ are the estimated states of x_2 and F, respectively. The hysteresis compensator is designed such that the control law will cancel the effect of the actual hysteresis,

$$u(n) = v(n) + \frac{\hat{k}_1}{k_v}\hat{F}(n) \tag{7.4}$$

Substituting the continuous time version of equation (7.4) to (1.10), it can be seen that if $\hat{F}(n) = F(n)$ and $\frac{\hat{k}_1}{k_v} = \frac{k_1}{k_v}$, the effect of hysteresis is canceled. The procedure of computer calculation is as following:

1. Set initial values $\hat{x}_2(0) = 0$ and $\hat{F}(0) = 0$
2. Calculate $\hat{F}(n+1)$ and $\hat{x}_2(n+1)$ from equation (7.3)
3. Apply controller output $u(n)$, calculated from equation (7.4), to the system
4. Repeat steps 2 and 3

Figure 7.1 shows the block diagram of the hysteresis compensated system, where the block of Gain Compensator has been discussed in Chapter 6, and the blocks of Gain Adjust k_{comp} and Average Filter will be described in Sections 7.4 and 7.5, respectively.

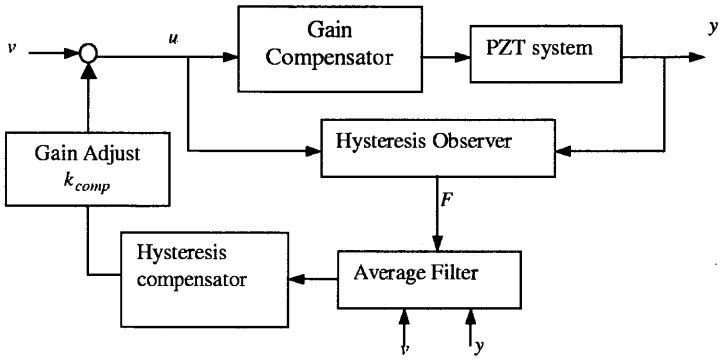

Figure 7.1 Block diagram of hysteresis compensated system

7.3 Stability Analysis

The properties of the hysteresis model in equation (1.11) were discussed in [37], and Property 1 in that work is described below.

Property 7.1 [37]: Assume that $0 < F_c \leq a$. If $|F(0)| \leq a$ then $|F(t)| \leq a$ $\forall\, t \geq 0$.

Applying Property 7.1, following results can be further derived.

Lemma 7.1: Assuming $0 < \hat{F}_c \leq a$ and choosing $\hat{F}(0) = 0$, then $\left|\hat{F}(t)\right| \leq a$ $\forall\, t \geq 0$.

Lemma 7.2: The compensated system is bounded if

1. $\hat{F}(0) = 0$ and $0 < \hat{F}_c \leq a$
2. v in equation (7.4) is designed such that the linear system ($k_1 = 0$) is stable
3. l_r is chosen such that the pole of the observer is within the unit circle

Proof: $\hat{F}$ is bounded which can be reached directly from condition 1 and Lemma 7.1. If condition 2 is further satisfied, u in equation (7.4) is bounded, and furthermore x (or x_1) in equation (1.10) is bounded. Rewrite equation (7.3) as

$$\hat{x}_2(n+1) = (a_{22} - l_r a_{12})\hat{x}_2(n) + l_r x_1(n+1) + (a_{21} - l_r a_{11})x_1(n) +$$

$$(b_{21} - l_r b_{11})v(n) + \left((b_{21} - l_r b_{11})\tfrac{\hat{k}_1}{k_v} + b_{22} - l_r b_{12}\right)\hat{F}(n)$$

If condition 3 is met, $\hat{x}_2$ is bounded. Therefore, Lemma 7.2 is proved.

7.4 Simulation

In this section, an example is applied to illustrate the implementation of the compensation scheme, and to study the effects of hysteresis parameter estimated errors and gain k_{comp} on the compensation.

7.4.1 Control Law

In order to study the effect of hysteresis parameter estimated errors on hysteresis compensation, symbols $\hat{k}_1$ and $\hat{F}_c$ are applied instead of their numerical values in the following derivations. Considering the model in equations (1.10) and (1.11) with parameters defined in equation (4.1), the coefficient matrices in equation (7.1) can be obtained directly as

$$A_c = \begin{pmatrix} 0 & 1 \\ -1.1893 \times 10^7 & -1.1612 \times 10^3 \end{pmatrix}$$

$$B_c = \begin{pmatrix} 0 & 0 \\ 0.43058 & -\hat{k}_1 \end{pmatrix} \qquad C_c = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

The coefficient matrices of the discrete time state space model in equation (7.2) with a 5KHz sampling rate are computed as,

$$A_d = \begin{pmatrix} 0.78800 & 1.6467 \times 10^{-4} \\ -1.9584 \times 10^3 & 0.59679 \end{pmatrix}$$

$$B_d = \begin{pmatrix} 7.6753 \times 10^{-9} & -1.7826 \times 10^{-8} \times \hat{k}_1 \\ 7.0903 \times 10^{-5} & -1.6467 \times 10^{-4} \times \hat{k}_1 \end{pmatrix} \qquad C_d = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

Hence, choosing the desired pole as 0.5, the gain of the reduce order observer l_r is calculated as 5.8777×10^2. The control law in equations (7.3) and (7.4) can then be written as

$$\begin{aligned} \hat{F}(n+1) &= \hat{F}(n) + \tfrac{1}{5000}(\hat{x}_2(n) - \tfrac{\hat{F}(n)}{\hat{F}_c}\,|\hat{x}_2(n)|) \\ \hat{x}_2(n+1) &= 0.5\hat{x}_2(n) + 5.8777 \times 10^2 x_1(n+1) - 2.4216 \times 10^3 x_1(n) \\ &\quad +6.6392 \times 10^{-5} u(n) - 1.5419 \times 10^{-4} \hat{k}_1 \hat{F}(n) \\ u(n) &= v(n) + \tfrac{\hat{k}_1}{0.43058}\hat{F}(n) \end{aligned} \tag{7.5}$$

7.4.2 Hysteresis Compensation

Figure 7.2 shows the responses of the systems with (right) and without (left) hysteresis compensation. It is observed that with known hysteresis parameters, the compensation scheme eliminates the hysteresis effects completely.

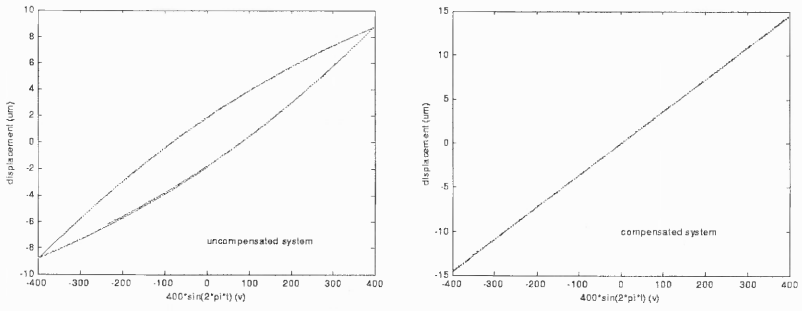

Figure 7.2 Simulated hysteresis loops (at $1Hz$): uncompensated (left) and compensated system (right)

7.4.3 Effect of Hysteresis Parameter Estimated Error

Applying perturbed values of $\hat{F}_c$ or $\hat{k}_1$ to the compensation scheme along with a $400Volts$ sinusoid reference at $1Hz$, the performance in time domain is simulated and plotted in Figures 7.3 and 7.5. Figures 7.4 and 7.6 show their corresponding frequency performance, measured by the normalized ratio. The results in time and frequency domains indicate that the compensation scheme is robust when $\hat{F}_c$ or $\hat{k}_1$ is larger than its true value.

The compensated scheme is further tested using the perturbed hysteresis parameters,

$$\hat{F}_c = 1.0664 \times 10^{-5} \quad \hat{k}_1 = 8.0092 \times 10^6$$

The results are shown in Figure 7.7. The normalized ratio between the third and the first harmonics is 2.473×10^{-3}. Hence the method is quite robust.

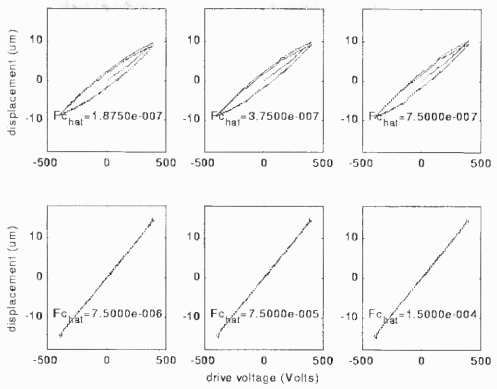

Figure 7.3 Simulated results in time domain: Effect of estimated error of F_c on hysteresis compensation

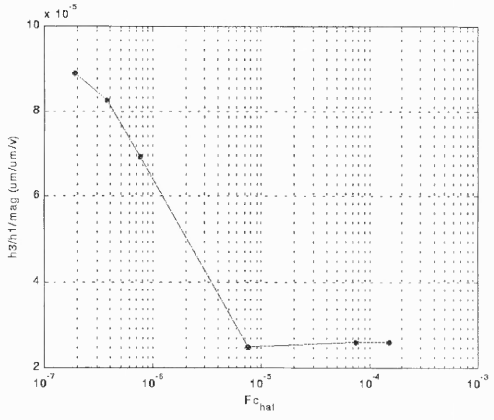

Figure 7.4 Simulated results in frequency domain: Effect of estimated error of F_c on hysteresis compensation

7.4.4 Compensation on the Parameter Estimated Error

The block Gain Adjust k_{comp} in Figure 7.1 is added to compensate the hysteresis parameter estimated error. Figures 7.8 and 7.9 show the effect of k_{comp} on hysteresis compensation when $\hat{F}_c$ and $\hat{k}_1$ equal their true values. When $k_{comp} = 1$, the hysteresis loop is completely eliminated and the normalized ratio of the first and the third harmonics is minimum. When $k_{comp} > 1$, the hysteresis is over-compensated and there appears a phase lead loop. On the other hand, when $k_{comp} < 1$ the hysteresis is under-compensated and the loop is phase lag. Hence by adjusting the value of k_{comp} such that the hysteresis loop is minimum (the system without phase shift), the effect of the hysteresis parameter estimated error is compensated.

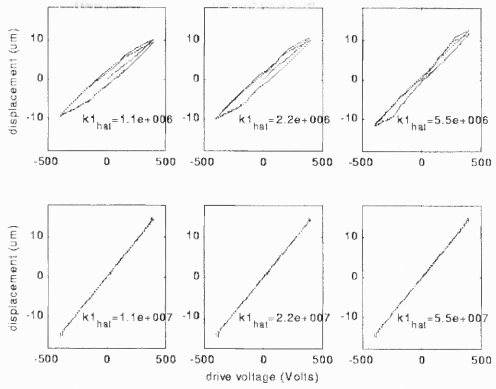

Figure 7.5 Simulated results in time domain: Effect of estimated error of k_1 on hysteresis compensation

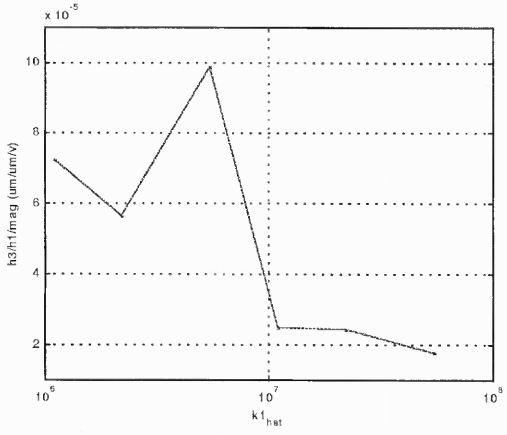

Figure 7.6 Simulated results in frequency domain: Effect of estimated error of k_1 on hysteresis compensation

7.5 Experiment

In this section, the compensation scheme shown in Figure 7.1 is experimentally verified.

7.5.1 Average Filter

The Average Filter calculates and outputs the average value of every N samples. It has two functions: eliminating background noise and down-sampling the sampling rate to suit the need of the hysteresis compensator which is essentially a low frequency operation.

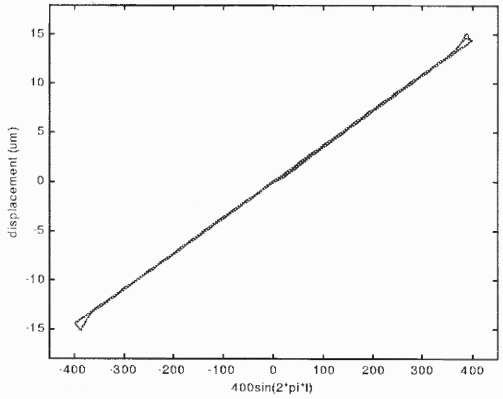

Figure 7.7 Simulated response of the hysteresis compensated system with $\hat{k_1} = 8.0092 \times 10^6$ and $\hat{F_c} = 1.0664 \times 10^{-5}$

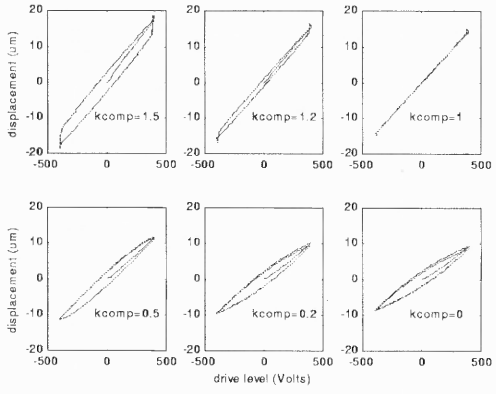

Figure 7.8 Simulation in time domain: Effect of over-compensation and under-compensation by k_{comp}

Figure 7.10 shows the effect of the Average Filter on eliminating background noises. The inputs to the A/D channels are grounded, and hence the noise is caused by the electric parts inside PC. With sampling rate of $10000Hz$, the spectral responses without and with 40-point Average Filter are shown in Figure 7.10. The noise level is reduced from $-40dB$ to $-80dB$.

The Average Filter is basically a low-pass filter. The sampling rate in Figure 7.10 is $10000Hz$. Without Average Filter, the noise with frequency less than $5000Hz$ is recovered. With 40-point Average Filter, only the noise with frequency less than $125Hz$ is recovered. Thus the background noise is significantly reduced.

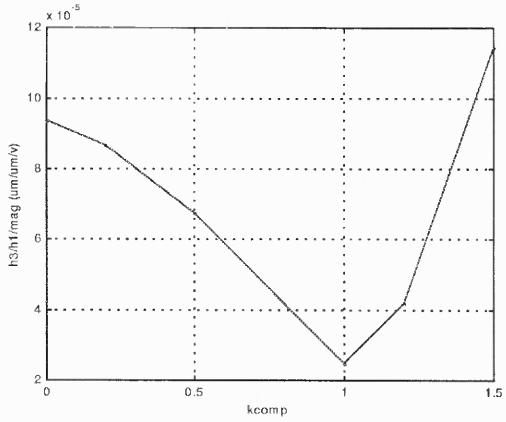

Figure 7.9 Simulation in frequency domain: Effect of over-compensation and under-compensation

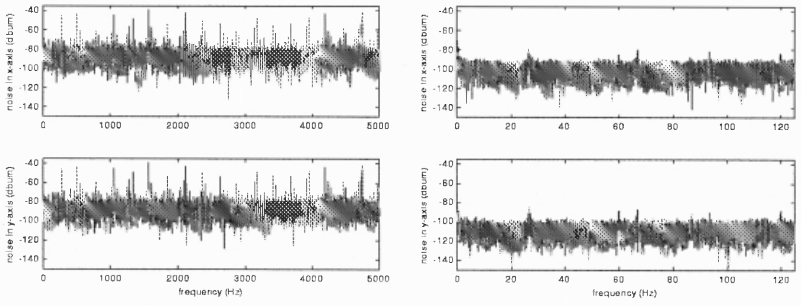

Figure 7.10 Spectrum responses with (right) and without (left) the average filter

The other function of the Average Filter is down-sampling. In this experiment, the sampling rate of the observer is $5000Hz$. Applying a 20-point Average Filter, the sampling rate of the hysteresis compensator is $250Hz$.

7.5.2 Hysteresis Compensation

Without compensation, Figures 7.11 and 7.12 show the effect of the hysteresis in time and frequency domains, respectively. Applied the nonlinear observer hysteresis compensator in equation (7.5), with the hysteresis parameters described by equation (4.2), Figures 7.13 and 7.14 show the results of hysteresis compensation. Comparing Figure 7.11 with 7.13, it can be seen that the compensated system eliminates the

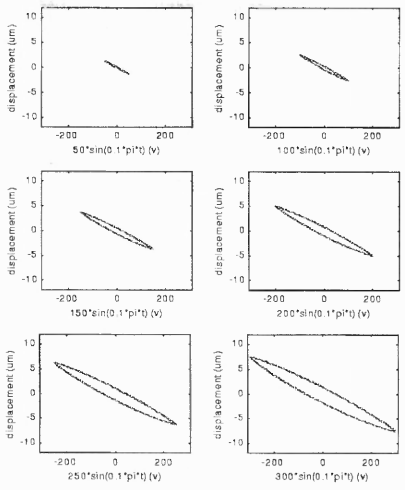

Figure 7.11 Experimental responses in time domain without hysteresis compensation

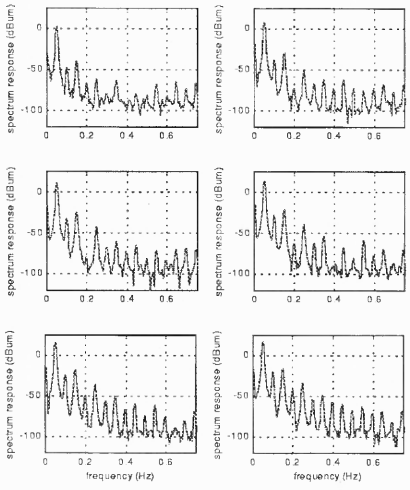

Figure 7.12 Experimental responses in frequency domain without hysteresis compensation

hysteresis loops completely. And as shown in Figures 7.12 and 7.14, the harmonics of compensated system are greatly reduced.

Table 7.1 summarizes the ratios of the third, the fifth and the seventh harmonics to the fundamental terms. It is noted that the ratios of the third harmonics to the fundamental terms are reduced about $20dB$. The small change of the other ratios is caused by the sensitivity of the equipment.

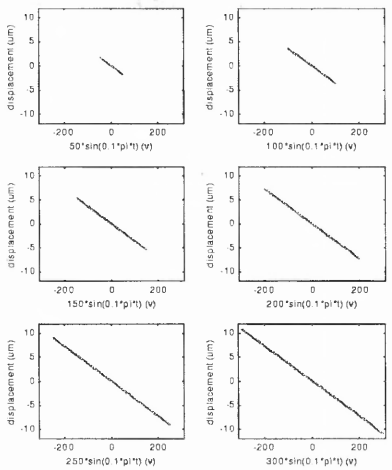

Figure 7.13 Experimental responses in time domain with hysteresis compensation

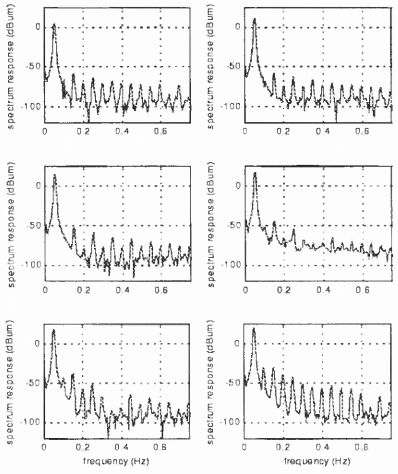

Figure 7.14 Experimental responses in frequency domain with hysteresis compensation

7.6 Performance Analysis

In this section, simulation is first applied to compare the tracking performance to a sinusoidal input. Experiment is then applied to verify the simulated results.

Figure 7.15 shows the tracking performance of the three systems: open-loop nonlinear system described by equations (1.10) and (1.11), its corresponding linear system ($k_1 = 0$) with PI control, and the nonlinear system with PI-control and observer compensator. The reference input is a sinusoid with magnitude of $400 Volts$ and frequency of $1Hz$. And the sampling rate is $5000Hz$. The coefficients of

PI-controller are $k_p = 2 \quad k_i = 1000$. It can be observed that the tracking error is greatly reduced by adding the hysteresis compensator, and the linear system response is approximately restored.

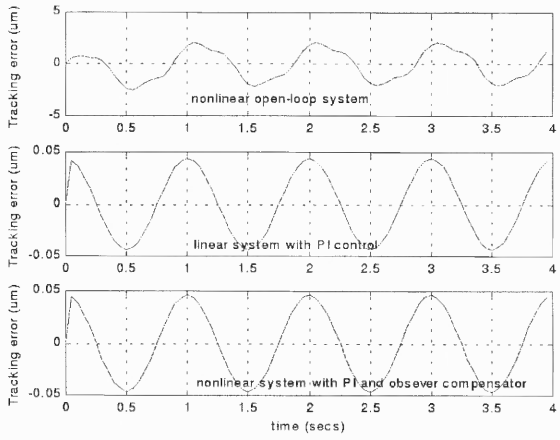

Figure 7.15 Simulation: Tracking error to the reference sinusoid

Tables 7.2 and 7.3 show the tracking error for the different frequencies and magnitudes of the input sinusoid. The ratios between the average mean square error of the hysteresis compensated system and the open loop system are 1.6898×10^{-2} in Table 7.2 and 3.9896×10^{-2} in Table 7.3. The average mean square error ratios between the hysteresis compensated system and the linear system are 1.0522 in Table 7.2 and 1.1228 in Table 7.3.

Figure 7.16 shows the experimental results of the tracking performance with input sinusoid of $100 Volts$ and $1 Hz$. The sampling rate is $250 Hz$, and the coefficients of PI-controller are $k_p = 0.1 \quad k_i = 6$. Clearly, the tracking performance is greatly improved by adding the nonlinear compensator. Same result can be obtained by comparing the tracking error shown in Figure 7.17. Tables 7.4 and 7.5 list the mean square error for different frequencies and magnitudes of the input sinusoid. The average mean square error ratio between hysteresis compensated system and the open loop system are 0.22554 in Table 7.4 and 2.4802×10^{-3} in Table 7.5.

$mag(Volts)$	uncompensated system (dB)			compensated system (dB)		
	h3/h1	h5/h1	h7/h1	h3/h1	h5/h1	h7/h1
50	-40.693	-63.137	-65.006	-63.338	-68.516	-72.173
100	-37.447	-57.486	-69.475	-69.313	-73.404	-76.284
150	-36.045	-54.079	-72.054	-67.463	-74.040	-79.342
200	-35.179	-52.700	-67.222	-61.395	-72.585	-88.901
250	-34.157	-51.765	-66.373	-56.503	-69.451	-103.644
300	-32.925	-51.267	-65.931	-50.291	-63.105	-75.423

Table 7.1 Experiment: Harmonic ratios with and without hysteresis compensation

frequency (Hz)	open-loop	linear system with PI control	nonlinear system with PI + observer
1	2.0806	9.3860×10^{-4}	1.0736×10^{-3}
2	2.0873	3.7512×10^{-3}	4.2011×10^{-3}
4	2.1006	1.4952×10^{-2}	1.5205×10^{-2}
6	2.1140	3.3439×10^{-2}	3.4111×10^{-2}
8	2.1273	5.8941×10^{-2}	5.9745×10^{-2}
10	2.1406	9.1134×10^{-2}	9.9424×10^{-2}

Table 7.2 Simulation: MSE (μm^2) for different sinusoidal frequencies

$mag(Volts)$	open-loop	linear system with PI control	nonlinear system with PI +observer
50	4.7430×10^{-2}	1.4240×10^{-3}	1.6396×10^{-3}
100	1.8541×10^{-1}	5.6959×10^{-3}	6.5056×10^{-3}
150	4.0153×10^{-1}	1.2816×10^{-2}	1.4502×10^{-2}
200	6.7770×10^{-1}	2.2784×10^{-2}	2.5703×10^{-2}
250	9.9490×10^{-1}	3.5599×10^{-2}	3.9951×10^{-2}
300	1.3400	5.1263×10^{-2}	5.7197×10^{-2}

Table 7.3 Simulation: MSE (μm^2) for different sinusoidal magnitudes

frequency (Hz)	open-loop	PI control	PI + observer compensator
0.5	0.30874	3.6495×10^{-3}	1.5848×10^{-3}
1	0.25722	1.2131×10^{-2}	2.7783×10^{-3}
2	0.26865	4.7868×10^{-2}	6.1587×10^{-3}
5	0.27711	3.0665×10^{-1}	4.6702×10^{-2}
10	0.40531	1.1442	2.8492×10^{-1}

Table 7.4 Experiment: MSE (μm^2) for different sinusoidal frequencies

$mag(Volts)$	open-loop	PI control	PI + observer
50	2.1480×10^{-2}	1.0542×10^{-3}	4.6568×10^{-4}
100	2.4890	4.0631×10^{-3}	1.7461×10^{-3}
150	3.2033	9.6104×10^{-3}	3.8362×10^{-3}
200	3.2334	1.6002×10^{-3}	6.6707×10^{-3}
250	3.4628	2.4171×10^{-2}	1.0420×10^{-2}
300	2.9296	3.5016×10^{-2}	1.4907×10^{-2}

Table 7.5 Experiment: MSE (μm^2) for different sinusoidal magnitudes

frequency(Hz)	normalized ratios($dB(\mu m/\mu m/Volts)$)		
	PI control	deadbeat observer (without PI)	deadbeat observer and PI control
1	-113.0972	-161.6712	-139.8552
10	-94.5392	-132.2042	-124.3892
50	-90.1212	-106.0042	-112.8102
100	-90.4342	-104.3442	-111.1062
150	-89.7272	-103.3732	-105.1732
200	-88.4092	-95.6542	-101.5812

Table 7.6 Simulation: Hysteresis compensation with different frequencies

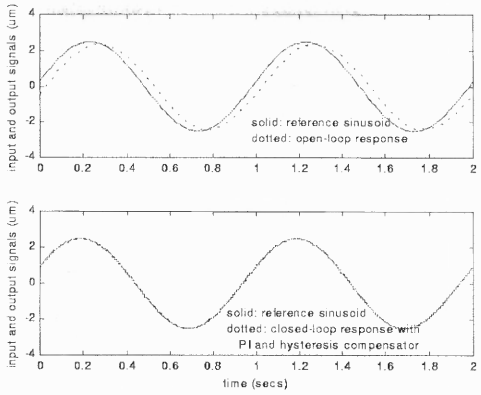

Figure 7.16 Experiment: Tracking performance to the reference sinusoid

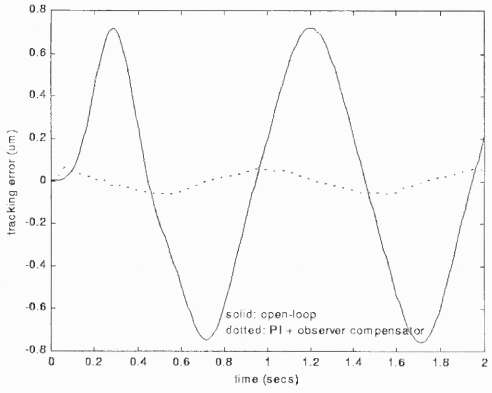

Figure 7.17 Experiment: Tracking error to the reference sinusoid

Table 7.6 compares the effect of the hysteresis compensation for the systems of nonlinear closed-loop PI control, nonlinear open-loop system with hysteresis observer compensator, and nonlinear closed-loop PI control with hysteresis observer compensator. And Table 7.7 lists the results of tracking errors of the the two closed loop system. It can be noted that the system with hysteresis observer compensator can achieve better results on eliminating the hysteresis phenomena than that with PI control. The system with PI and hysteresis observer compensator combines the advantages of the PI and hysteresis observer compensator and achieves the best results in terms of reducing tracking error and nonlinearity. Comparing the results of PI-control with and without hysteresis observer compensator, the average normalized ratio is reduced by about $20dB\mu m$ by adding the observer compensator,

frequency(Hz)	peak to peak values of the tracking errors (μm)	
	PI control	deadbeat observer and PI control
1	0.14054	8.6750×10^{-2}
10	1.4350	0.84590
50	5.2174	3.3053
100	6.7414	4.3674
150	7.0495	4.7482
200	7.3452	5.0075

Table 7.7 Simulation: Tracking errors with and without hysteresis compensation

$mag(Volts)$	100	200	300	400	500	600
error (μm)	0.78491	0.92760	1.6928	3.0652	4.6808	6.4091

Table 7.8 Simulation: Steady state error for different square magnitudes

and the relative error is reduced by 36%. Figures 7.18-7.20 show the spectrum responses of the three systems. In experiment, Tables 7.4 and 7.5 show that the improvement of the tracking error is 76% and 57.9% by introducing the hysteresis compensator.

Simulation results in Figure 7.21 show the tracking performance to the input square wave with different magnitudes, a situation commonly encountered with command shaping strategies (see Chapter 9), where a feedforward component is generated for high speed of response. However, the presence of hysteresis causes substantial steady state errors. The use of the hysteresis observation alleviates this problem significantly. For this study, deadbeat hysteresis observer is applied. Comparing the results in Figure 7.21, it is noted that the hysteresis observer compensated system has high overshoot during the first half period, which is caused by the finite convergence rate of state estimation. After the first half period, hysteresis compensated system produces better performance than the open-loop system measured in terms of zero steady state error and quicker transient characteristics. Table 7.8 shows the steady state error with different magnitudes of square wave.

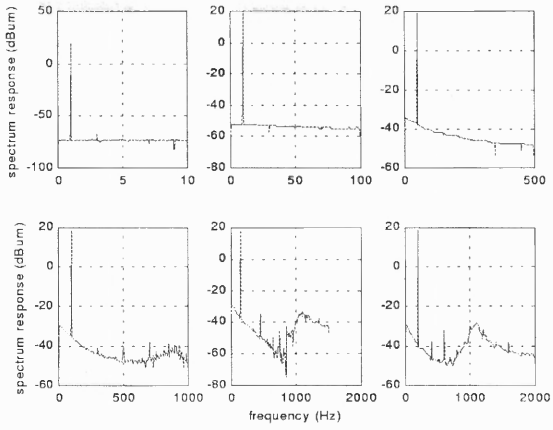

Figure 7.18 Simulated spectrum responses of the system with PI control and hysteresis compensator

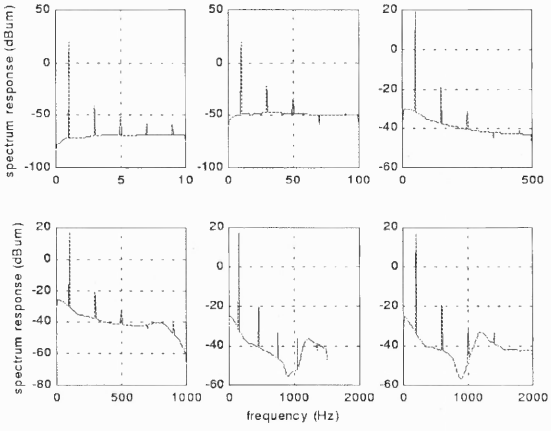

Figure 7.19 Simulated spectrum responses of the system with PI control

7.7 Summary

Nonlinear observer hysteresis compensator is proposed in this chapter to eliminate the hysteresis effect in the piezoelectric actuator. Stability analysis shows that the closed-loop system is stable if the controller and the observer designed assuming $k_1 = 0$ are stable.

Simulation is applied to study the efficiency and the robustness of the compensator. It is shown that the compensator is robust when $\hat{F}_c$ and $\hat{k}_1$ are bigger than their corresponding true values (20 and 5 times respectively). To

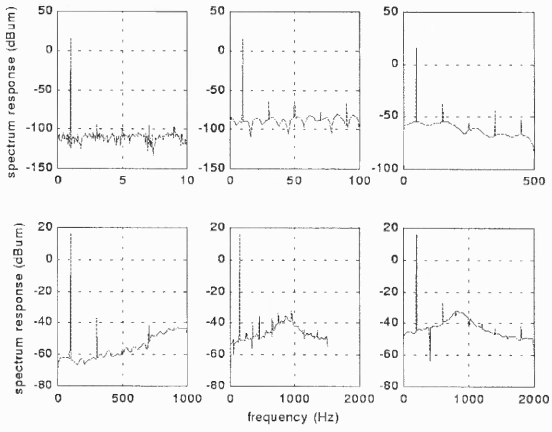

Figure 7.20 Simulated spectrum responses of the system with hysteresis compensator

compensate the effect of hysteresis parameter estimated error, an adjustable gain is added to the compensation scheme.

Both experiment and simulation show the efficiency of the hysteresis compensated scheme. The hysteresis loop is eliminated in time domain and the harmonics are greatly reduced in frequency domain.

Experiment and simulation are further applied to study the tracking error to the reference sinusoid with different frequencies and magnitudes. Comparing with open-loop control, the PI control with hysteresis compensator greatly reduces the tracking error. Simulation also shows that the linear system performance can be restored by adding the hysteresis compensator.

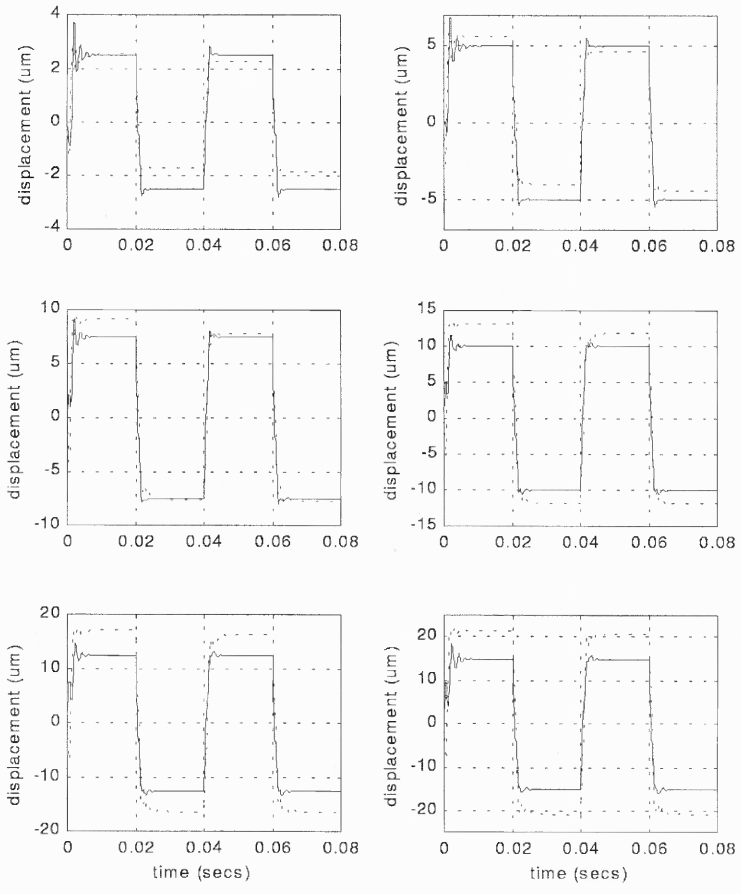

Figure 7.21 Simulation: Tracking performance to the square wave input (solid line: hysteresis compensated output; dashed line: open-loop response)

CHAPTER 8

ADAPTIVE HYSTERESIS COMPENSATOR

The application of piezoelectric actuators in precision and fine motion control requires highly consistent performance and accuracy. Due to variations in environmental factors such as temperature, humidity, as well as the uncertainty in operating conditions, an adaptive mechanism to update the control system is necessary. This chapter describes an adaptive hysteresis compensator which tracks the hysteresis parameters. The ideas is that the hysteresis parameters are estimated in real-time, and the hysteresis compensator is updated accordingly to compensate the slowly changing hysteresis nonlinearity.

Figure 8.1 shows the block diagram of the adaptive hysteresis compensator, where the block of adaptive algorithm will be discussed in detail.

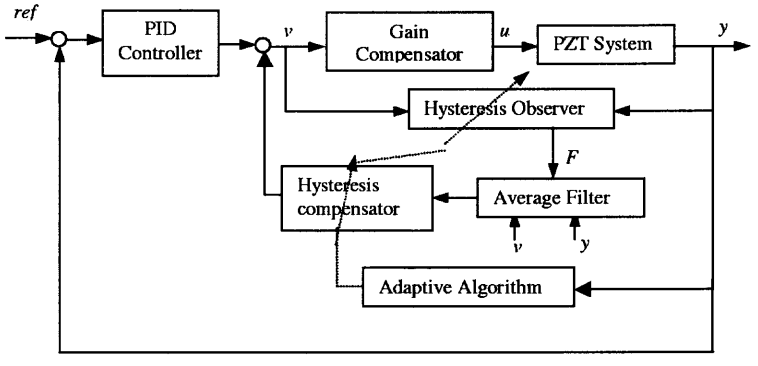

Figure 8.1 Block diagram of adaptive hysteresis compensator

8.1 Adaptive Hysteresis Compensated Scheme

8.1.1 Adaptive Algorithm

Assume the linear coefficients γ, k_n, k_v in equation (7.1) are known and constant, while the hysteresis parameters F_c and k_1 are unknown and may change slowly. Hence the discrete time coefficient matrix A_d in equation (7.2), and the gain of the reduce order observer l_r are constant. The adaptive algorithm is designed such that

the hysteresis parameters are estimated on-line, and the hysteresis compensator is updated based on the estimated parameters. The procedure of the hysteresis compensation in Chapter 7 is modified as:

1. Set initial values $\hat{F}_c(0) > 0$, $\hat{k}_1(0) = 0$, $\hat{x}_2(0) = 0$ and $\hat{F}(0) = 0$
2. Compute states $\hat{F}(n+1)$ and $\hat{x}_2(n+1)$ from equation (7.3)
3. Calculate the first and the third harmonics h_1 and h_3 using small sample DFT discussed below
4. Update hysteresis parameters $\hat{F}_c$ and $\hat{k}_1$ as,

$$A_0 = \sqrt{h_1^2 - 36h_3^2} \quad \tilde{k}_1 = \frac{A}{A_0} - k_n \quad \tilde{F}_c = \frac{2\tilde{k}_1 A_0^2}{9\pi k_n h_3} \tag{8.1}$$

$$\hat{k}_1(n) = \hat{k}_1(n-1) + \tilde{k}_1 \quad \hat{F}_c(n) = \hat{F}_c(n-1) + \tilde{F}_c \tag{8.2}$$

5. Output $u(n)$, calculated form equation (7.4), to the PZT system
6. Repeat steps 2-5

8.1.2 Harmonic Calculation Using Small Sample of Data

To estimate the hysteresis parameters in real-time, a small sample of data should be applied to reduce the delay caused by collecting data and calculating DFT or FFT. The small sample size can be efficiently utilized if signal leakage is properly accounted for. That is to say, if sampling frequency is a harmonic multiple of a signal's frequency, there is no leakage to the adjacent frequency bias. Harmonics can then be calculated accurately without using windows.

Let $freq$ stand for the frequency of an input sinusoid, and F_{sa} be sampling rate. If $F_{sa} = 20 \times freq$, the harmonics in frequency domain can then be expressed as,

$$harmonics\ frequency = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix} \times freq$$

This is because the number of samples in frequency domain equals $(F_{sa}/2)/freq$. Hence, the first frequency point in FFT is

$$(F_{sa}/2)/(\text{number of samples in frequency domain}) = freq$$

Discrete Fourier Transform (DFT) is applied to calculate the harmonics, as below:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} \qquad 0 \leq k \leq N-1$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi kn}{N}} \qquad 0 \leq n \leq N-1$$

where $N = 20$. From the above frequency vector of the harmonics, it can be seen that the second and the fourth elements of $x(n)$ stand for the first and the third harmonics, respectively.

8.2 Simulation

Considering the example in Section 7.4, the hysteresis compensation algorithm is shown in equation (7.5). In the following, the example is applied to illustrate the on-line hysteresis parameter estimation and the hysteresis compensation.

8.2.1 Real Time Hysteresis Parameter Estimation

For an input sinusoid with frequency of $1Hz$ and amplitude of $400Volts$, Figure 8.2 shows the estimated parameters with different initial value of $\hat{F}_c$ and $\hat{k}_1 = 1.1 \times 10^7$, and Figure 8.3 shows the estimated parameters with different initial value of $\hat{k}_1$ and $\hat{F}_c = 7.5 \times 10^{-6}$. During the first period (time less than $1sec$), the estimated hysteresis parameters are equal to their initial values. During the second period, the data in the first period time are applied to calculate the hysteresis parameters which are updated to the new values, and so on. The mechanism of the hysteresis parameter estimation is that the $(N+1)$th hysteresis parameter estimation are calculated using the data in the Nth period. It can be seen that although the estimated parameters do not approach to their true values, they are convergent,

$\hat{F}_c \to 1.0664 \times 10^{-5}$ and $\hat{k}_1 \to 8.0092 \times 10^6$. The bias is mainly caused the inaccurate harmonics expressions (k_a and k_b are chosen as one). The update speed can be improved by increasing the frequency of the input sinusoid, and the bias can be reduced by choosing $k_a = 1.6$ and $k_b = 0.66$, as shown in Figures 8.4 and 8.5 where $\hat{F}_c \to 6.3447 \times 10^{-6}$ and $\hat{k}_1 \to 1.2145 \times 10^7$.

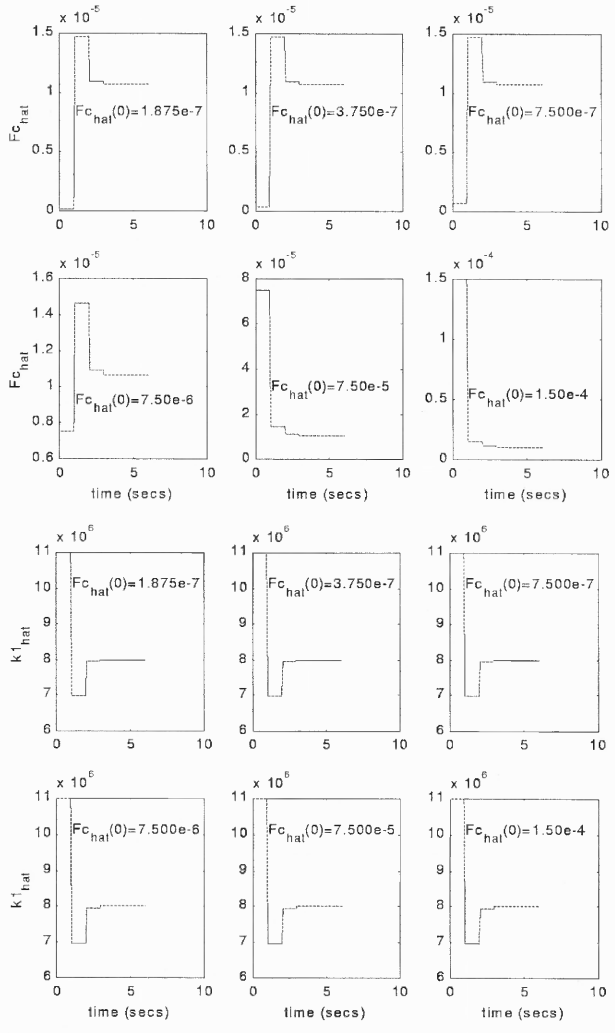

Figure 8.2 Hysteresis parameter estimation curves with different $\hat{F}_c(0)$ ($\hat{k_1}(0) = 1.1 \times 10^7$)

8.2.2 Adaptive Hysteresis Compensation

Since $\hat{F}_c$ can not be zero, an arbitrary value is chosen as its initial value. Letting $\hat{k}_1(0) = 0$ and $\hat{F}_c(0) = 5 \times 10^{-5}$, and applying the hysteresis compensated scheme in

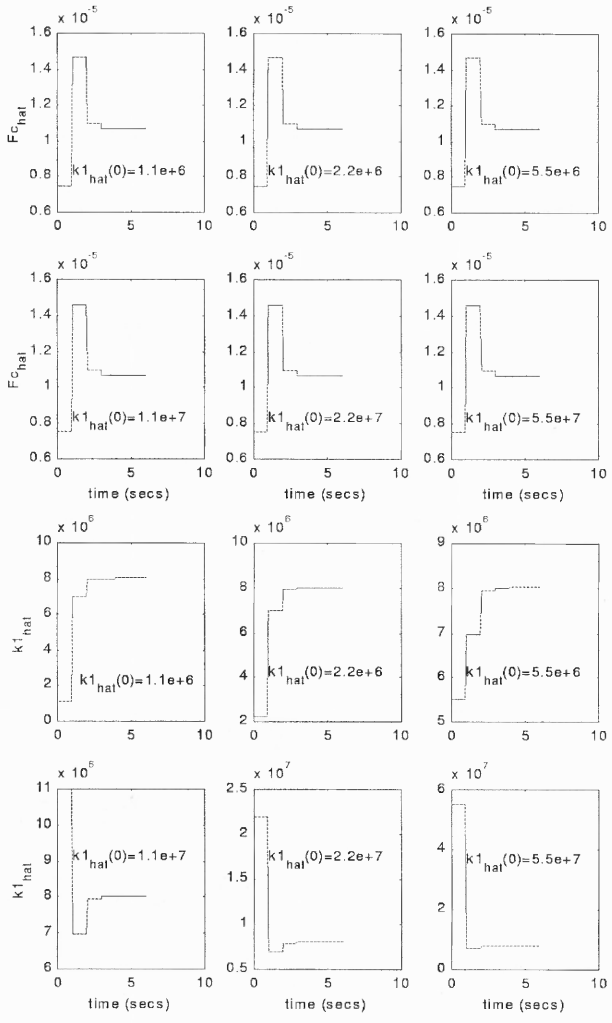

Figure 8.3 Hysteresis parameter estimation curves with different $\hat{k_1}(0)$ ($\hat{F}_c(0) = 7.5 \times 10^{-6}$)

Section 8.1, the tracking error of the compensated system is shown in Figures 8.6, together with the tracking errors of the linear and the nonlinear open-loop systems. It can be observed that the tracking performance of the compensated system is quite similar as that of the linear system, and the nonlinearity existed in the open-loop system are eliminated. Figure 8.7 shows the responses of the three systems in frequency domain. It can be seen that the harmonics in the hysteresis compensated system are very small. Hence the results in time and frequency domains show that

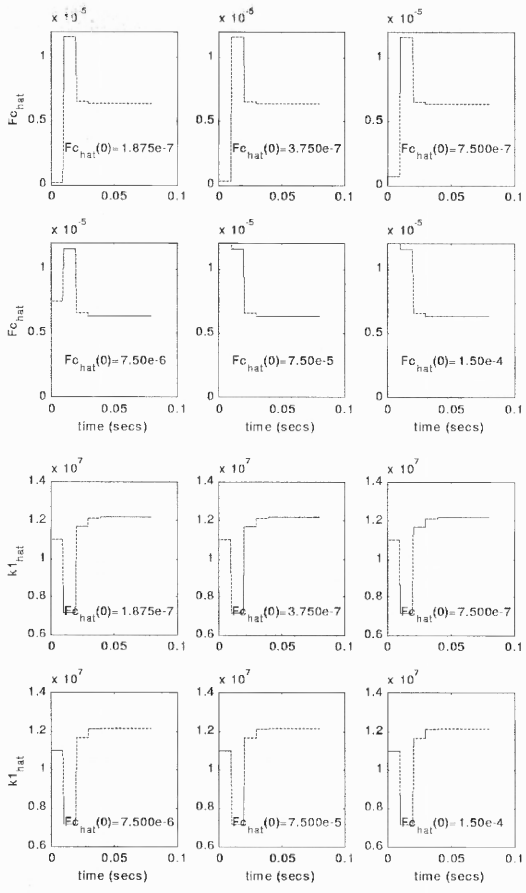

Figure 8.4 Hysteresis parameter estimation curves with different $\hat{F}_c(0)$ ($\hat{k_1}(0) = 1.1 \times 10^7$)

the compensator effectively reduces the hysteresis effect, and the linear performance is approximately achieved. The process of the hysteresis parameter estimation is shown in Figure 8.8.

8.3 Experiment

Figures 8.9-8.12 show the responses of the PZT system with input sinusoid of $0.5Hz$ and magnitude varied from $50Volts$ to $300Volts$ in six steps, i.e.,

$$mag = [\ 50 \quad 100 \quad 150 \quad 200 \quad 250 \quad 300\ ]Volts$$

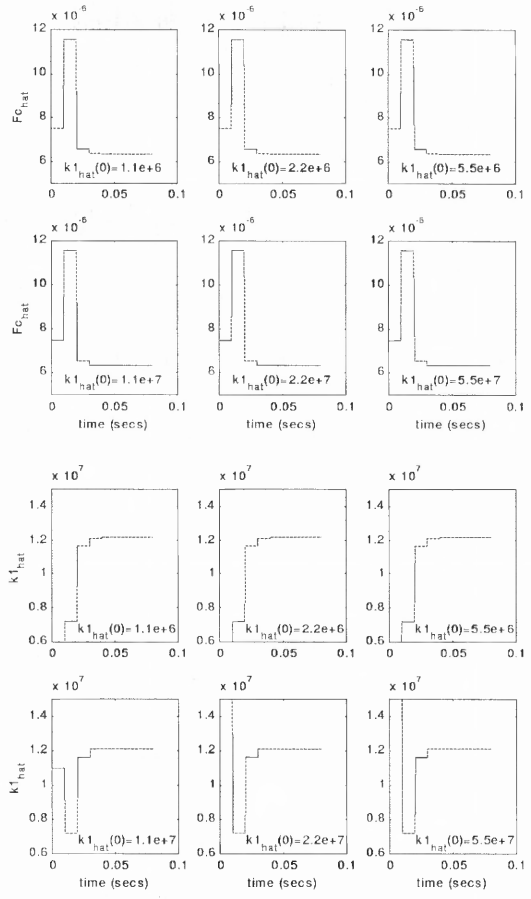

Figure 8.5 Hysteresis parameter estimation curves with different $\hat{k}_1(0)$ ($\hat{F}_c(0) = 7.5 \times 10^{-6}$)

where the sampling rate of the observer is $5000Hz$, and that of the controller is $250Hz$. Figures 8.9 and 8.10 show the open loop responses. Hysteresis nonlinearity can be observed in time domain as hysteresis loops, and in frequency domain as harmonics. Applying the control strategy shown in Figure 8.1 with initial values of $\hat{k}_1(0) = 0$ and $\hat{F}_c(0) = 7.5 \times 10^{-6}$, and the coefficients of PID as $k_i = 0.5 \quad k_p = 0 \quad k_d = 0$, the effect of hysteresis compensation is shown in Figures 8.11 and 8.12. It is observed that the hysteresis loops are eliminated and the harmonics are reduced significantly. For easy comparison, the normalized ratios of the uncompensated and compensated systems are listed in Table 8.1. It is noted that adding the hysteresis compensator, the ratios are reduced by $30dB\mu m$. The

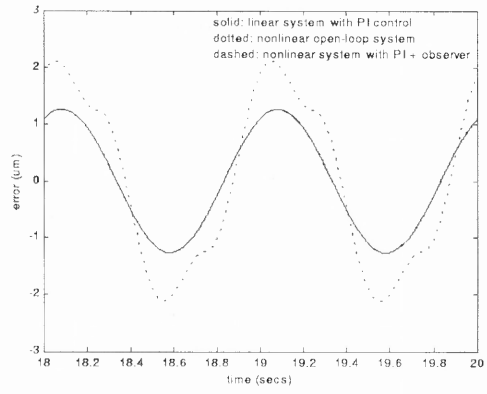

Figure 8.6 Simulation: The effect of hysteresis compensation in time domain

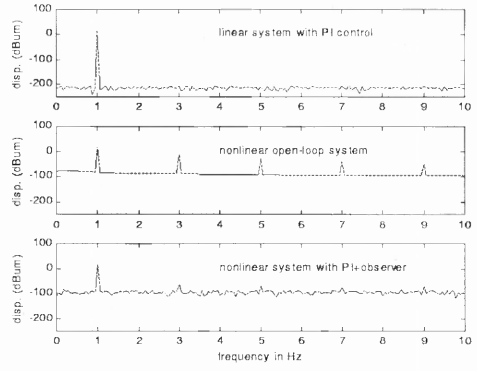

Figure 8.7 Simulation: The effect of hysteresis compensation in frequency domain

value of $\hat{F}_c(0)$ is chosen based on our best knowledge from Chapter 6, and $\hat{k}_1(0) = 0$ eliminates the hysteresis compensation during the first period in which the estimated hysteresis parameters may be far from their true values.

8.4 Summary

An adaptive hysteresis compensator is proposed for an input sinusoid to compensate slowly changed hysteresis parameters. Hysteresis parameters are calculated iteratively from the input and output of the PZT system. They are then applied in hysteresis compensation. Simulation shows that the estimated hysteresis parameters

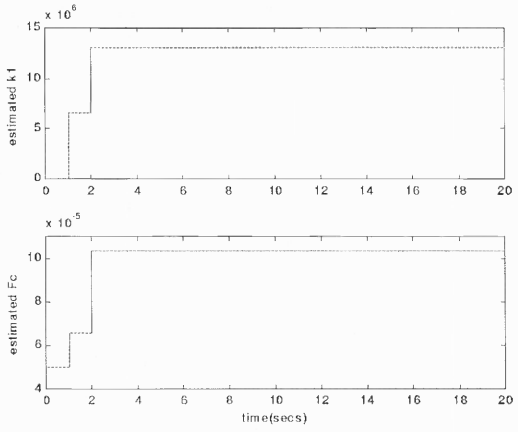

Figure 8.8 Hysteresis parameter estimation

$mag(Volts)$	normalized ratios($dB(\mu m/\mu m/Volts)$)	
	uncompensated	compensated
50	-75.3494	-113.1864
100	-78.3840	-111.5060
150	-80.8948	-111.5578
200	-82.4316	-114.2246
250	-83.7028	-114.7558
300	-84.5774	-115.3044

Table 8.1 The effect of adaptive hysteresis compensation

are convergent. Both simulated and experimental results show that the compensated scheme can effectively eliminate the hysteresis effect.

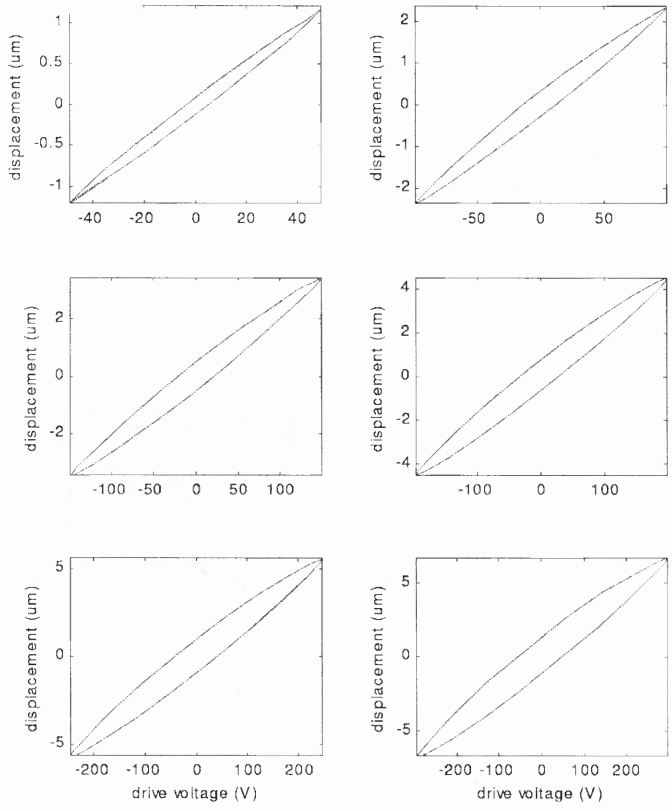

Figure 8.9 Experiment: Time domain response of the hysteresis uncompensated system

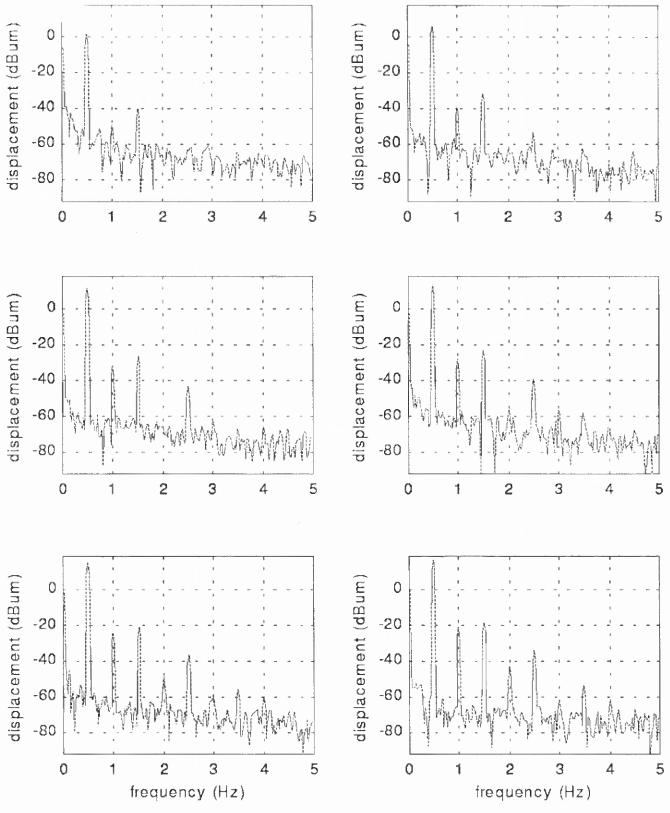

Figure 8.10 Experiment: Frequency domain response of the hysteresis uncompensated system

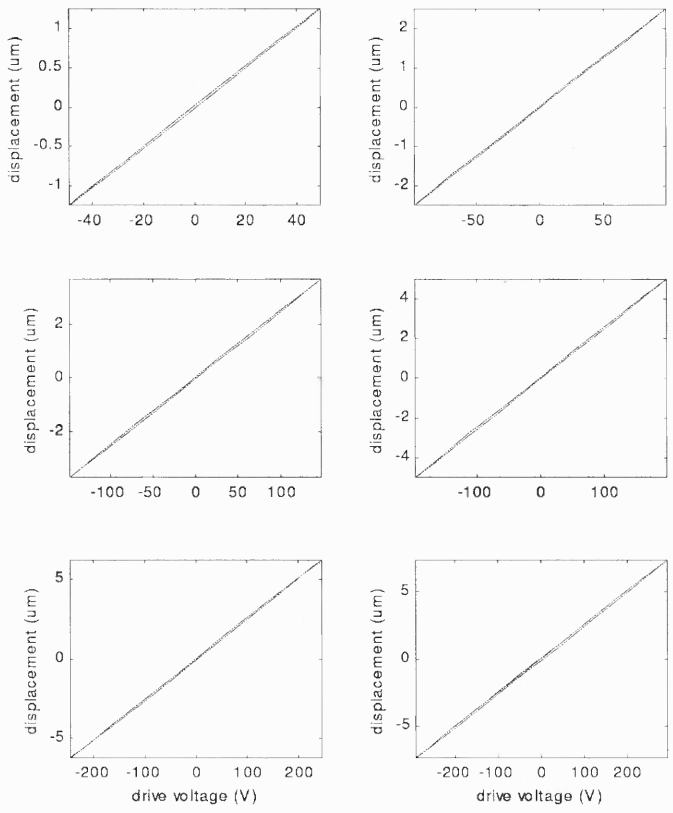

Figure 8.11 Experiment: Time domain response of the hysteresis compensated system

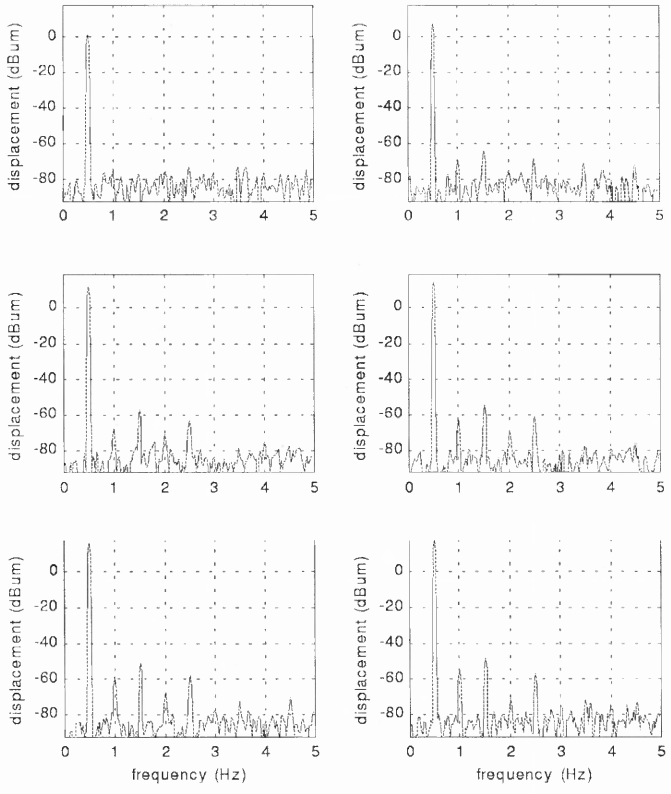

Figure 8.12 Experiment: Frequency domain response of the hysteresis compensated system

CHAPTER 9

VIBRATION AND PRECISION CONTROL TIME DIVISION MULTI-CONTROLLER

As discussed in Chapter 1, piezoelectric actuators are generally underdamped. Without compensation, the open loop transient response exhibits high degree of overshoot and residual vibration. The common control strategies such as PID control or servo compensation are generally ineffective when constraints such as actuator saturation and high speed of response are imposed. In this chapter, the strategy of time division multi-controller is proposed to solve the problem.

9.1 Time Division Multi-Control Strategy

The block diagram of the control strategy is shown in Figure 9.1, where Gain Compensator and Average Filter have been discussed in Chapters 6 and 7 respectively. In the following, Input Shaper and time-division multi-controller strategy will be discussed in detail.

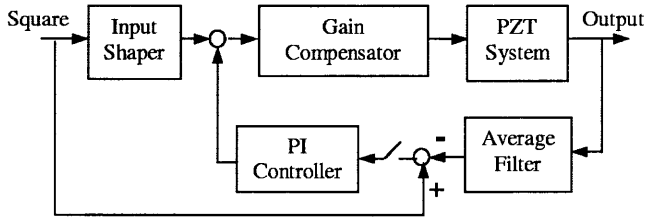

Figure 9.1 Block diagram of time division multi-control system

9.1.1 Input Shaper

Input shaper has been successfully applied in vibration control, even in the presence of modeling uncertainties and structural nonlinearities [31-34]. A number of design methods have been proposed: Zero Vibration (ZV), Zero Vibration and Derivative

(ZVD), Extra Insensitive (EI), and Optimal [35]. Here ZVD is chosen to increase the robustness of the impulse shaping sequence to modeling errors.

Input shaping is an open-loop scheme in which the square (or step) input is re-shaped such that the system oscillations are canceled. The re-shaped input involves a sequence of impulses with properly assigned weights and temporal separation. Hence the system output is the addition of all the impulse responses. Although vibration exists for each impulse response, the system oscillations can be canceled by properly choosing weights and temporal separation. To simplify the discussions, consider a second order system with the following transfer function,

$$G(s) = \frac{k}{s^2 + 2\varsigma\omega s + \omega^2} \tag{9.1}$$

The unit impulse response of (9.1) is given by:

$$y(t) = \frac{\omega}{\sqrt{1-\zeta^2}} e^{-\zeta\omega(t-t_0)} sin(\omega\sqrt{1-\zeta^2}(t-t_0)) \tag{9.2}$$

Let $y_i(t_N)$ be the impulse response to the impulse $A_i\delta(t-t_i)$, $i = 1, ..., M$, where M is the number of impulses. A_i and t_i are respectively, the impulse weight and application time; then the total response at $t = t_N$ can be written as:

$$y(t_N) = \sum_{i=1}^{M} y_i(t_N) \tag{9.3}$$

The goal is to minimize the level of residual vibration at $t = t_N$:

$$e^{-\varsigma\omega t_N} \sqrt{\sum_{i=1}^{M} \frac{A_i\omega}{\sqrt{1-\varsigma^2}} e^{-\varsigma\omega t_i} \cos\omega(t_N - t_i) + \sum_{i=1}^{M} \frac{A_i\omega}{\sqrt{1-\varsigma^2}} e^{-\varsigma\omega t_i} \sin\omega(t_N - t_i)} \tag{9.4}$$

For a ZVD shaper, the impulse parameters are given by,

$$A_1 = \frac{1}{1+2M_p+M_p^2} \quad A_2 = \frac{2M_p}{1+2M_p+M_p^2} \quad A_3 = 1 - A_1 - A_2 \quad t_d = \frac{\pi}{\omega_d} \tag{9.5}$$

where M_p is percentage maximum overshoot and $\omega_d = \omega_n\sqrt{1-\varsigma^2}$ with ω_n as natural undamped frequency and ς as damping ratio. These values can be obtained directly from a step response. Figure 9.2 shows the output of the input shaper of a square

input with magnitude of 0.5 and period of t_p. The parameters are calculated from equation (9.5), where the subscripts + and − stand for the rising and falling steps respectively. The input shaper can thus be expressed as,

$$output_{shaper} = 2A \times gshaper \times \begin{cases} A_{1+} - 0.5 & for \quad 0 \le t < t_{d+} \\ A_{1+} + A_{2+} - 0.5 & for \quad t_{d+} \le t < 2t_{d+} \\ 0.5 & for \quad 2t_{d+} \le t < t_p/2 \\ A_{2-} + A_{3-} - 0.5 & for \quad t_p/2 \le t < t_p/2 + t_{d-} \\ A_{3-} - 0.5 & for \quad t_p/2 + t_{d-} \le t < t_p/2 + 2t_{d-} \\ -0.5 & for \quad t_p/2 + 2t_{d-} \le t < t_p \end{cases} \tag{9.6}$$

where t_p and A are the period and the magnitude of the square wave; $gshaper$ is chosen such that the gain of the system with input shaper is equal to one. Hence $gshaper = \frac{\omega^2}{k}$.

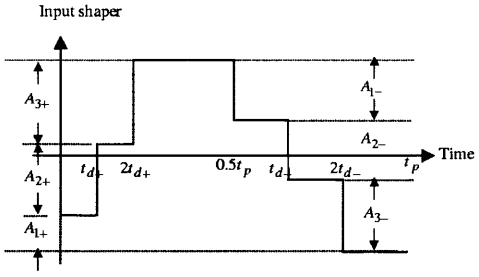

Figure 9.2 Parameter allocation of input shaping

Input shaping has the following advantages:

1. Very practical, since only the application time and the impulse weights need to be stored

2. Very efficient in eliminating vibration, which will be illustrated later in this chapter

3. Efficient under actuator saturation constraint

   As shown in Figure 9.2, instead of applying the whole input drive at one time, the input drive is divided into three parts. Only one part is applied at one time, and hence the transient control effort is greatly reduced.

9.1.2 Time Division Multi-Controller Strategy

Although input shaper is efficient in reducing vibration and remaining fast transient response, as an open-loop control scheme, it could not eliminate the steady state error caused by drift and noise. Time division multi-control strategy effectively combines input shaper and PI control and thus implements vibration and precision control.

The time allocation of the strategy is as following: as shown in Figure 9.2, during $[0\ 2t_{d+}]$, input-shaping control is active and the output of the shaper follows the signal in Figure 9.2. At the same time, PI controller is passive and its output keeps the value before $t < 0$. During $[2t_{d+}\ t_p/2]$, on the other hand, the input shaper is passive and its output maintains the value at $t = 2t_{d+}$, while PI controller is active and its output is calculated based on the error between the filtered output and the reference input. Similar strategy is applied for the falling step.

Two sampling rates are applied in the strategy: fast sampling rate $10000Hz$ for input shaping since the switching time is critical, and slow sampling rate $250Hz$ for PI control. The reasons to choose the slow sampling rate for PI controller are as following: Firstly, the fast sampling rate like $10000Hz$ will make the system more sensitive to numerical errors because the poles of the discrete system approach $z = 1$. Secondly, PI controller is used in the period of fine motion, and thus fast sampling rate is not necessary. And finally, the Average Filter can be applied to eliminate background noise and hence improve resolution.

9.2 Simulation

In the following, an example is applied to illustrate the implementation of the control strategy. Considering the model in equations (1.10) and (1.11) with parameters described by (4.1) and (4.2), the parameters of M_p and t_d in equation (9.5) are firstly obtained from the square response with frequency of $25Hz$ and magnitude of

$50Volts$. The parameters of the input shaper are then calculated as

$$A_1 = 0.645 \quad A_2 = 0.31624 \quad A_3 = 0.038763 \quad t_1 = 6.625 \times 10^{-4} \quad gshaper = 1.15$$

Figure 9.3 shows the responses of the systems with open loop control, input shaping and time division multi-control strategy, where the coefficients of PI controller are chosen as $K_p = 0.1$ and $K_i = 100$. It can be seen that input shaper and time division multi-controller can greatly reduce vibration and hence improve transient performance. Comparing with input shaper, Time division multi-controller is more efficient in eliminating the steady state error caused by hysteresis, drift and noise, which is to be discussed in the next section.

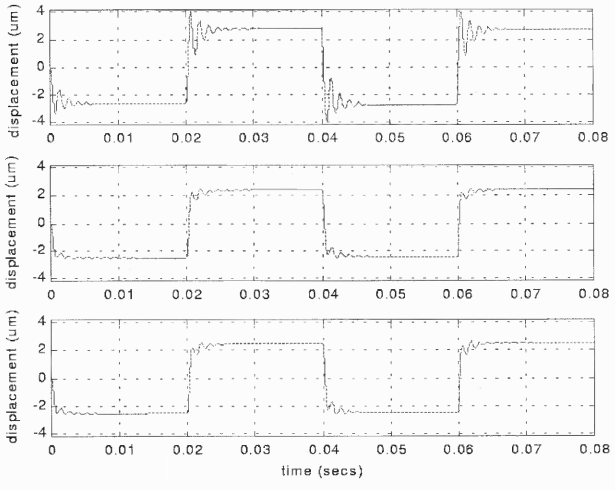

Figure 9.3 Simulated responses of the three control strategies: top: open-loop; middle: input shaping; bottom: time division multi-controller

9.3 Experiment

Experiment is applied in this section to verify the control strategy. Some of the parameters are given first.

1. The parameters of the PI controller are obtained using on-line tuning, $K_p = 0.1$ and $K_i = 200$

2. The Gain Compensator is applied to eliminate the effect of the nonlinear scale factor, as discussed in Chapter 6, with coefficients as

$$C_4 = 0 \quad C_3 = 3.2223 \times 10^{-2} \quad C_2 = -0.33328 \quad C_1 = 2.0258 \quad \text{for x-axis}$$

$$C_4 = 0 \quad C_3 = 3.2920 \times 10^{-2} \quad C_2 = -0.32127 \quad C_1 = 1.9468 \quad \text{for y-axis}$$

3. The parameters of the input shaper are obtained from square response, as following

x-axis:

$$\begin{array}{llll} A_{1+} = 0.15477 & A_{2+} = 0.47728 & A_{3+} = 0.36796 & t_{d+} = 7.5 \times 10^{-4} \\ A_{1-} = 0.28099 & A_{2-} = 0.49819 & A_{3-} = 0.22082 & t_{d-} = 7.5 \times 10^{-4} \\ gshaper = 2.7027 & & & \end{array}$$

y-axis:

$$\begin{array}{llll} A_{1+} = 0.21386 & A_{2+} = 0.49718 & A_{3+} = 0.28896 & t_{d+} = 7.45 \times 10^{-4} \\ A_{1-} = 0.21386 & A_{2-} = 0.49718 & A_{3-} = 0.28896 & t_{d-} = 7.5 \times 10^{-4} \\ gshaper = 2.1368 & & & \end{array}$$

Figure 9.4 shows the open-loop square response. The frequency and the magnitude of the square wave are respectively $1Hz$ and $50Volts$. High degree of overshoot and vibration during the transient period are clearly seen. Figure 9.5 shows the system response with the input shaper only. It is observed that the vibration is eliminated completely, though steady state error (mainly caused by drift) still exists. Figure 9.5 shows the steady state error of $0.9\mu m$ in x-axis and $1.3\mu m$ in y-axis. Applying the scheme of time division multi-controller, Figure 9.6 shows the system responses with different magnitudes of the reference square wave. The rise time is $1.5msecs$ and the steady state error is zero. Hence the controller implements the objectives of vibration reduction, fast transient response and zero steady state error. Figure 9.6 also shows the efficiency of the Average Filter in reducing background noise and hence improving system's resolution.

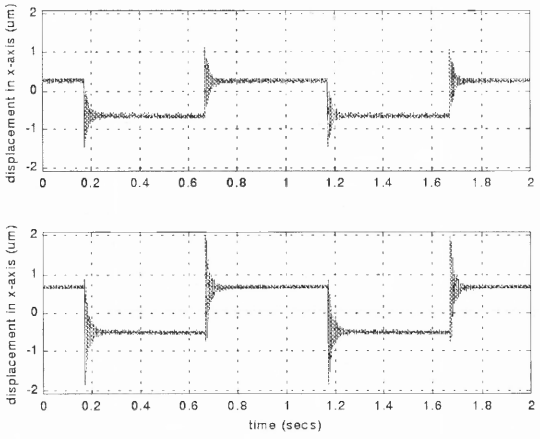

Figure 9.4 Experiment: Open-loop square-wave response

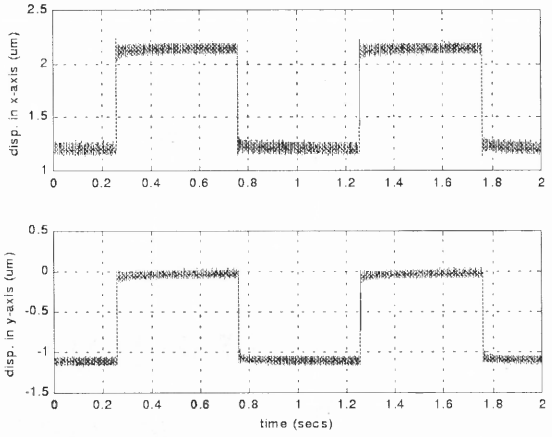

Figure 9.5 Experiment: Vibration control using input shaping

9.4 Summary

Time division multi-control strategy is proposed in this chapter to reduce vibration, implement fast transient response, and eliminate steady state error. The strategy effectively combines input shaping and PI control in different time zones such that input shaping is applied in intensive motion control and PI-control in fine motion control. The efficiency of the strategy is verified by simulated and experimental results.

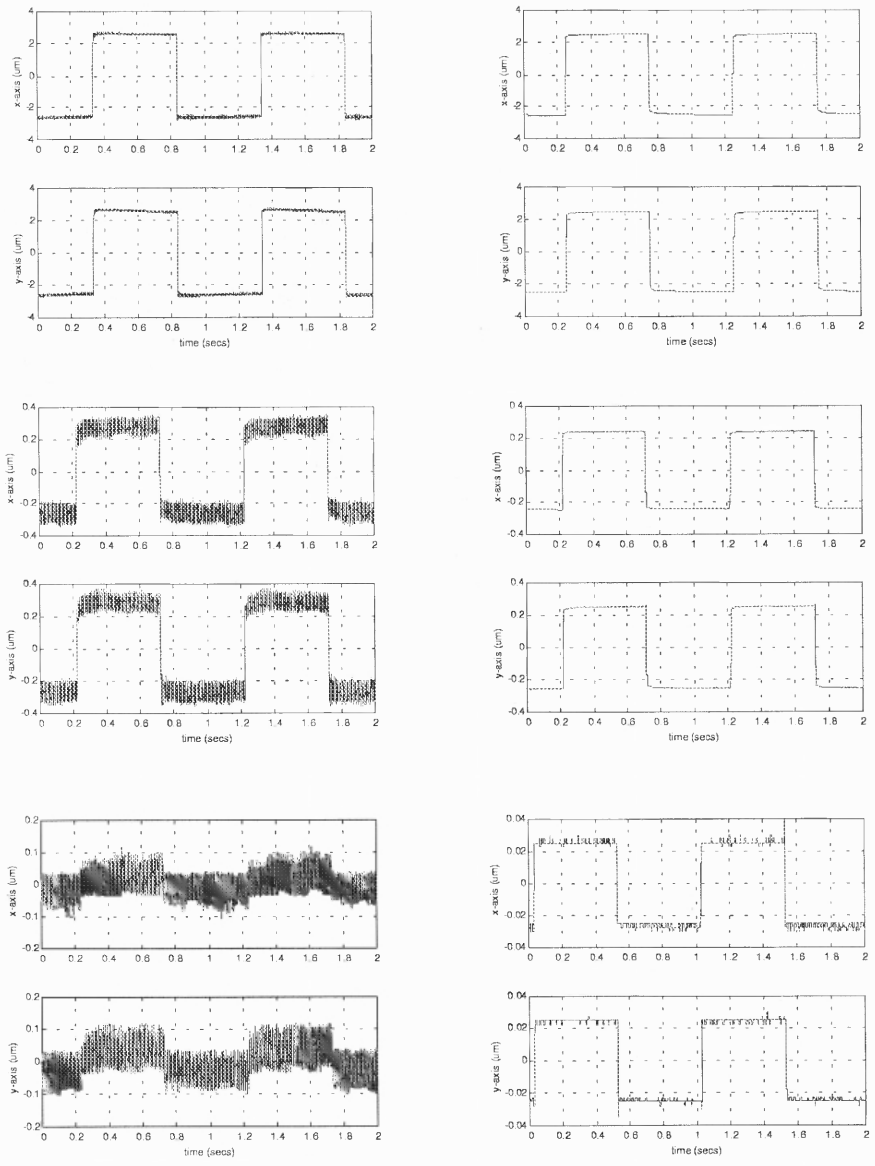

Figure 9.6 Experiment: Vibration and precision control using time division multi-controller: without filter (left) and with filter (right)

CHAPTER 10

CONCLUSIONS

Characteristics of the monolithic piezoelectric nano-actuator have been studied, covering the areas of hysteresis characteristics analysis, system identification, nonlinear compensation, vibration and precision control, and real-time control software development.

Frequency domain analysis shows that the hysteresis nonlinearity possesses a certain degree of weak scalability, so that a normalized harmonic ratio can be applied to quantitatively measure the degree of hysteresis nonlinearity. Simulation and experimental results also verify that the proposed simplified Dahl model can effectively describe the hysteresis in the monolithic piezoelectric actuator. The hysteresis parameter estimation is carried out using the first and the third harmonics.

A reduced order nonlinear hysteresis observer is proposed to compensate the hysteresis nonlinearity. Stability analysis shows that the compensated system is stable if the corresponding linear system is stable. Experimental and simulated results verify the effectiveness of the compensation which reduces the normalized ratio by about $20dB$. Simulated results also show that the linear system performance can be recovered. Furthermore, an adaptive hysteresis compensator is proposed to track the slowly changed hysteresis parameters. This is followed by time-division multi-control strategy which combines input shaping and PI control in different time scales to carry out vibration suppression and precision control.

Real-time control strategies, such as PID-control, input shaping, reduced order observer and low-pass filter are developed and successfully bench tested on a TMS320C31 based digital signal processing platform. Furthermore, GUI is developed in the LabVIEW environment to implement communication between the code in DSP memory and LabVIEW. Hence the system signals can be monitored,

and the control variables be adjusted in real-time through LabVIEW Front Panel. The GUI greatly simplifies the control system operation and control strategy test.

In summary, the main contributions of the work are:

1. Verified the hysteresis model in the monolithic piezoelectric actuator.
2. Introduced frequency domain method to analyze the hysteresis nonlinearity.
3. Proposed the method of multi-controller time division control.
4. Effectively reduced the hysteresis nonlinearity.
5. Developed reusable real-time control package.

Suggestions for the future work on the PZT system are listed below:

1. Investigate the effect of load on the system models.
2. Design an adaptive controller for slowly changed model parameters and arbitrary input.
3. Replace the PI controller with a servo compensator, which is more general and capable of handling a wider range of exogenous signals.
4. Improve the analytic expressions of the hysteresis harmonics.
5. Study the PZT system with six degrees of freedom.

APPENDIX A

DSP MEMORY DISTRIBUTION

Table A.1 defines the DSP memory for the variables in the user programs.

Address	Variables	Function
0x12F4	BIASF	frequency offset
0x12F5	FREQX	frequency of the reference signal in x-axis
0x12F6	CHECK1	1=data in memory is ready; 0=not ready
0x12F7	TIMER	Sampling time
0x12F8	GRAB1	1=begin to store data; 0=clear counter
0x12FB	FREQY	frequency of the reference signal in y-axis
0x12FC	MAGY	magnitude of the reference signal in y-axis
0x12FD	CONTY	control signal in y-axis
0x12FE	OUTY	output signal in y-axis
0x12FF	KPY	coefficient of proportional control in y-axis
0x1300	KIY	coefficient of integral control in y-axis
0x1301	KDY	coefficient of derivative control in y-axis
0x1302	REFX	reference signal in x-axis
0x1303	REFY	reference signal in y-axis
0x1306	KCOMP	adjustable gain in observer compensation
0x1308	SAFE1	switch from open-loop control to close-loop
0x1382	CONTX	control signal in x-axis
0x1383	OUTX	output signal in x-axis
0x1384	MAGX	magnitude of the reference signal in x-axis
0x1385	KDX	coefficient of derivative control in y-axis
0x1386	KPX	coefficient of proportional control in y-axis
0x1387	KIX	coefficient of integral control in y-axis
0x1388		start area for store experimental data

Table A.1 DSP memory distribution for user programs

APPENDIX B
NONLINEAR SCALAR FACTOR

The following code automatically generates a sinusoid with changed magnitude. The signal is applied to the monolithic piezoelectric actuator to study the nonlinearity of the scalar factor.

Section 1 implements the GUI between the code in DSP memory and Labview. The program allows the user to input the maximum value of the magnitude, decide when to store experimental data. Through the interface, the experimental results are uploaded and shown in the Labview.

Section 2 implements real-time control strategy. Here a sinusoid with changed magnitudes is generated and outputted to the PZT system, the system output and the reference input are stored.

Section 3 is the graphical programs in Labview. Front Panel shows the signals (reference input and system output) and the controlled variables. Block Diagram is the code to implement the functions.

B.1 Graphical User Interface Between DSP and Labview

```
/*---------- CIN source file - modeln.c ----------
Objective:
Implement the connection between the control buttons in Labview and DSP
----------*/

#include "extcode.h"
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define CHECK1 0x12F6
#define GRAB1 0x12F8
```

```
#define REF2PC 0x1382
#define Y 0x1383
#define REF2DSP 0x1384
#define SCALING 500000.
#define MASK 0x0000FFFFL
CIN MgErr CINRun(float64 *ref2dsp, float64 *ref2pc, float64 *output, int32
*check, int32 *var5);
CIN MgErr CINRun(float64 *ref2dsp, float64 *ref2pc, float64 *output, int32
*check, int32 *var5) {
    long int longv, longh, longl;
    outp(0x306,0x0); /*page value*/
    // read check1 from DSP memory
    outpw(0x302,CHECK1); /*0x12F6*/
    longl = inpw(0x300) & 0x0000FFFF;
    longh = inpw(0x300) & 0x0000FFFF;
    longv = longl | (longh <<16) ;
    *check = (int32) (longv);
    //Write grab1 to DSP memory
    outpw(0x302, GRAB1);        /*0x12F8*/
    outpw(0x300,*var5);
    outpw(0x300,(*var5)>>16);
    //read ref from DSP memory
    outpw(0x302,REF2PC); /*0x1382*/
    longl = inpw(0x300) & 0x0000FFFF;
    longh = inpw(0x300) & 0x0000FFFF;
    longv = longl | (longh << 16) ;
    *ref2pc = (float64) (longv)/2047*4.9976;
```

```
        //read output from DSP memory
        outpw(0x302,Y);    /*0x1383*/
        longl = inpw(0x300) & 0x0000FFFF;
        longh = inpw(0x300) & 0x0000FFFF;
        longv = longl | (longh << 16);
        *output = (float64) (longv)/8191*4.99939;
        //Write magnitude of reference sinusoid to DSP memory
        longv = (long int)((*ref2dsp)*SCALING);
        outpw(0x302, REF2DSP);        /*0x1384*/
        outpw(0x300,longv);
        outpw(0x300,longv >>16);
        return noErr;
    }//end of file
/*————————- CIN source file – modeln_d.c ————————————-

Objective:
Upload the experimental data in DSP memory when ready.
Outputs:
sampling_freq – sampling rate;  y, ref – signals uploaded
————————————————————————————————————*/

    #include "extcode.h"
    #include < stdio.h >
    #include < math.h >
    #include < conio.h >
    #define Timer 0x12F7
    #define Y 0x1388
    #define REF 0x3A98
    #define MASK 0x0000FFFFL
```

```
#define DATASIZE 5000
#define SCALING 500000.
typedef struct {
    int32 dimSize;
    float64 arg1[1];
} TD1;
typedef TD1 **TD1Hdl;
CIN MgErr CINRun(float64 *Sampling_freq, TD1Hdl y, TD1Hdl ref);
CIN MgErr CINRun(float64 *Sampling_freq, TD1Hdl y, TD1Hdl ref) {
    long int longv, longh, longl,i;
    MgErr err=noErr;
    outp(0x306,0x0);
    outpw(0x302,Timer);
    longl = inpw(0x300) & 0x0000FFFF;
    longh = inpw(0x300) & 0x0000FFFF;
    longv = longl | (longh << 16) ;
    *Sampling_freq = (float64) 400.0*SCALING/longv;
if (err = NumericArrayResize(fD, 1L, (UHandle *) &y, 2*DATASIZE))
goto out;
(*y)->dimSize = 2*DATASIZE;
for (i=0;i < 2*DATASIZE;i++) {
    outpw(0x302,( Y + i ));
    longl = inpw(0x300) & 0x0000FFFF;
    longh= inpw(0x300) & 0x0000FFFF;
    longv= longl | (longh <<16);
    (*y)->arg1[i]= (float64) (longv)/8191*4.99939;
  }
```

```
if (err = NumericArrayResize(fD, 1L, (UHandle *) &ref, 2*DATASIZE))
goto out;
(*ref)->dimSize = 2*DATASIZE;
for (i=0;i < 2*DATASIZE;i++) {
  outpw(0x302,( REF + i ));
  longl = inpw(0x300) & 0x0000FFFF;
  longh = inpw(0x300) & 0x0000FFFF;
  longv = longl | (longh  << 16);
  (*ref)->arg1[i]= (float64) (longv)/2047*4.9976;
}
out:
return noErr;
}//end of file
```

B.2 C Source Code for Real-time Operation System (DSP)

```
/*———————————— sinel2.C ————————————
Objective: generate a sinusoid with changed magnitude
Notes:
Sampling rate = 50*10^6/(TIMPER0*8*numcalls)
where numcalls mean ReadAdc(=1) only
Fs = 500Hz, 100-point average filter (NFILTER = 100)
nperiod = Fs/freq = 500/0.05 = 10000
total time = 10000/(500/100)/60 = 34mins
#data stored = NMAG * nperiod * 2/NFILTER = 10k (per signal)
————————————————————————————————————*/
#define TIMPER0 12500
#include < math.h >
```

```
#include "d310biox.H"
//—- Memory address distribution —
#define CHECK1 0x12F6
#define GRAB1 0x12F8
#define TIMER 0x12F7
#define REF2PC 0x1382
#define Y 0x1383
#define V 0x1384
#define MEMORY1_START 0x1388
#define MEMORY2_START 0x3A98
#define DATA_SIZE 5000
#define PI 3.14159265358979
#define SCALING 500000.
#define NMAG 50
#define NFILTER 100
int *grab1;
long *check1, * ptr_to_v,*int_ptr[2];
/*———— store date ————————————
Objective: filter and store data.
inputs:
data1 – Y from A/D converter;
data2 – reference generated from computer
grab1 – control signal
DATA_SIZE – #data stored
NFILTER – #data averaged
————————————————————————*/
void StoreData(int data1,int data2) {
```

```
static long ndata=1,ndummy=0,sumf[2]={0,0};
    if ((*grab1) == 1 ) {
        if (ndata  <= 2* DATA_SIZE ) {
            if (ndummy  < NFILTER) {
                sumf[0] = sumf[0] + data1;
                sumf[1] = sumf[1] + data2;
                ndummy++;
            }
            if (ndummy == NFILTER) {
                *(int_ptr[0]+ndata) =(long)((double)(sumf[0])/NFILTER);
                *(int_ptr[1]+ndata) =(long)((double)(sumf[1])/NFILTER);
                ndata++;
                sumf[0] = sumf[1] = ndummy = 0;
            }
        }
        else
            *check1= (int)(1);
    }//endif
    else
    {
        *check1 =(int)(0);
        ndata=0;
    }
}
//————————— Main() function ————————-
void main() {
    double gain,ref = 0, freq=0.05,
```

```
Fs=(double)(50000000)/(TIMPER0*8*1);
double coe[4]={-9.7409e-003,1.0593e-001,-4.5859e-001,1.8338};
int i=1, k=0, index1, *int_ref, *int_y;
long *int_time, nperiod;
InitDsp();
//————— set memory address ————————
//for StoreData:
check1 = (long int *) CHECK1; //0x12F6
* check1 = (int) (0); //1=data in mem. ready
grab1 = (int *) GRAB1; //0x12F8
* grab1 = (int) (0); //1=store data; 0=reset
int_ptr[0] = (long *) MEMORY1_START; //0x1388
int_ptr[1] = (long *) MEMORY2_START; //0x3A98
//for main function:
int_time = (long int *) TIMER; //0x12F7
* int_time = (long int) (1/Fs*SCALING*400);
int_ref = (int *) REF2PC; //0x1382 control
int_y = (int *) Y; //0x1383 ReadAdc
ptr_to_v = (long int *) V; //0x1384 mag. of ref.
* ptr_to_v = (int) (0*SCALING);
nperiod = (long)(Fs/freq);
WriteDACs(0,0);
while(1) {
    for (i=1;i <=NMAG*2;i++){
    /* test magnitude i=NMAG -> mag=maximum */
    //i=NMAG;
    //if (i==NMAG){
```

```
                //—- magnitude ——
                if (i < =NMAG) index1=i;
                else index1=NMAG*2-i;
                //— run one period ——
                for (k=0;k  < nperiod;k++){
                    *int_y = ReadAdc(1);
                    ref =(double)(*ptr_to_v)/SCALING*index1/NMAG;
                    // dynamic gain compensation
                    gain=ref*ref;
                    gain = coe[0]*gain*ref+coe[1]*gain+coe[2]*ref+coe[3];
                    ref =(1-*check1)*ref*sin((float)k*2*PI/nperiod);
                    *int_ref = (int) (2047.0/4.9976 *ref*gain);
                    WriteDAC(*int_ref,0);
                    StoreData(*int_y,(int)(2047.0/4.9976*ref));
                    //—— reset ref signal ——-
                    if((*ptr_to_v)==0) {
                        k=nperiod;
                        i=0;
                    } //endif
                } //endfor
            } //endfor
        }//endwhile
    } //end of file
```

B.3 Labview Program

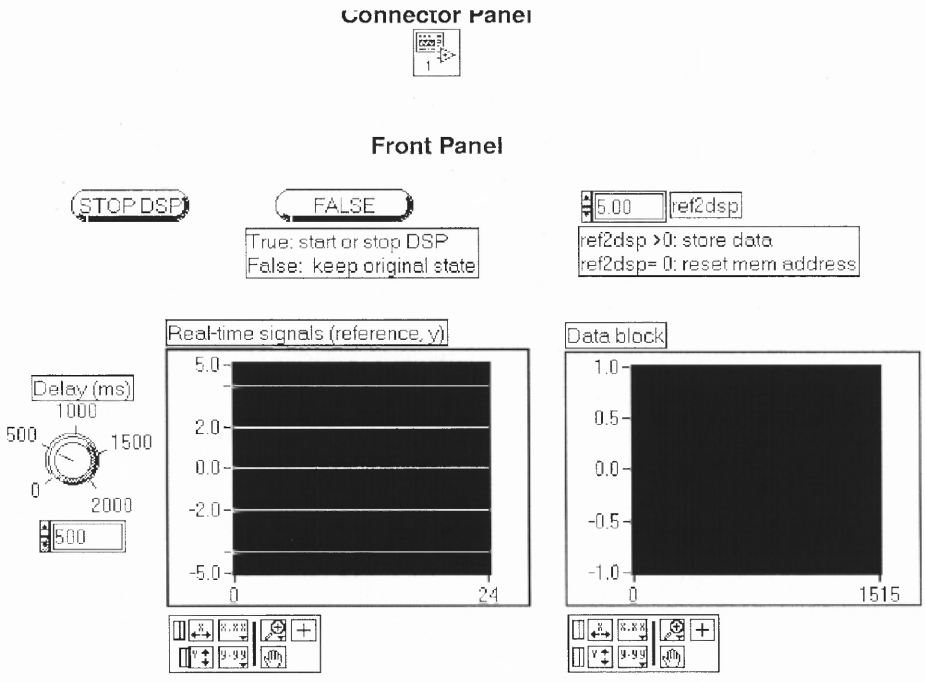

Figure B.1 Labview program for modeling PZT system

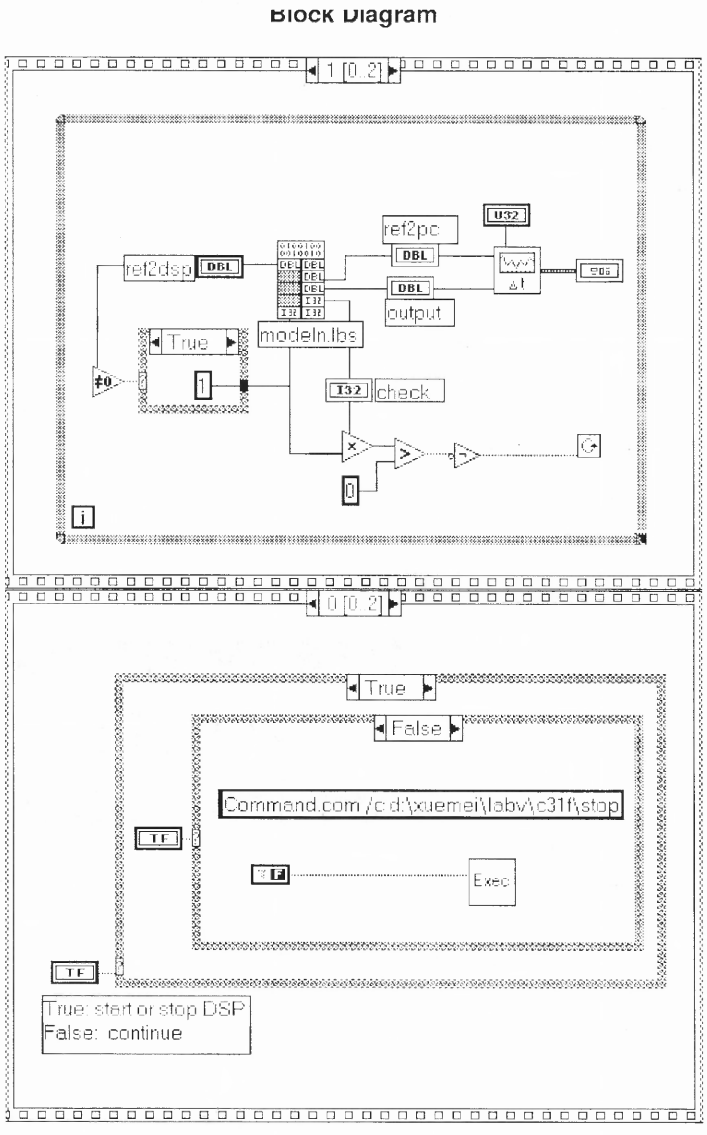

Figure B.2 Labview program for modeling PZT system (continued)

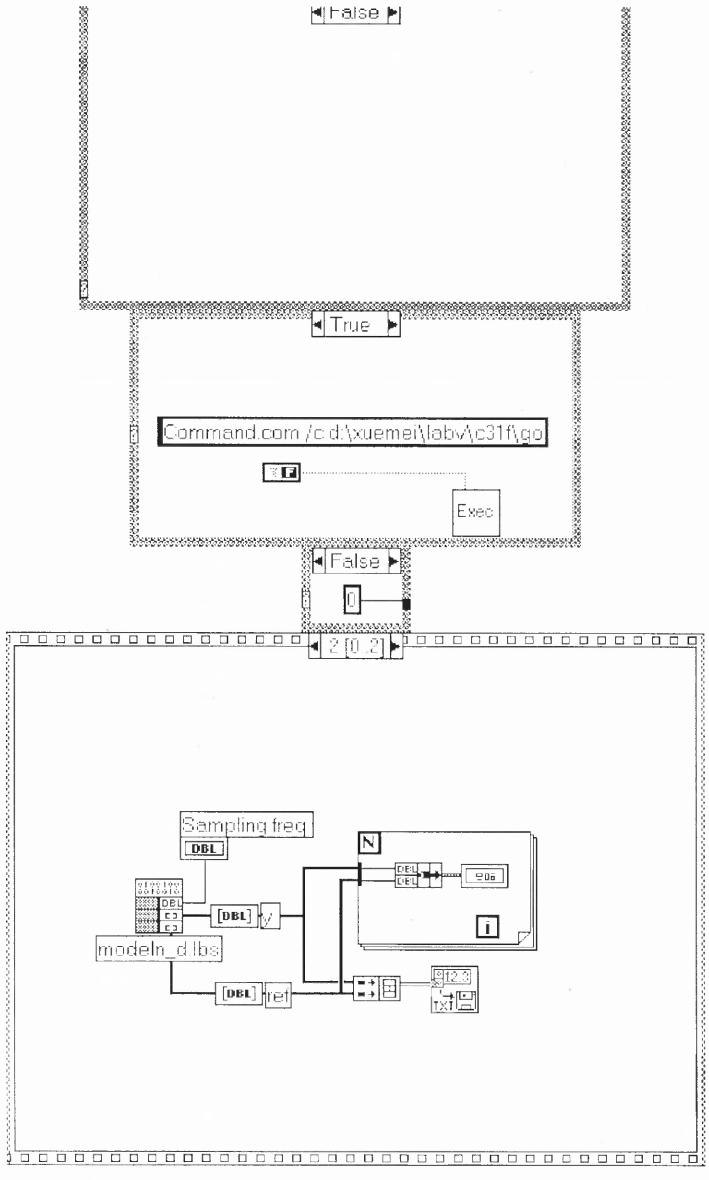

Figure B.3 Labview program for modeling PZT system (continued)

APPENDIX C

HYSTERESIS OBSERVER COMPENSATION

The following code implements the control strategy of the nonlinear hysteresis observer compensation, where reference input is sinusoid.

Section 1 implements the GUI between the code in DSP memory and Labview. The program allows the user to change the magnitude and the frequency of the sinusoid, the estimated hysteresis parameters, and the PID coefficients, decide when to store the experimental data, and choose one of the control strategies.

Section 2 implements the real-time control strategies, where average filter is applied to implement the functions of eliminating background noise and down-sampling. The code is designed to have the flexibility of switching among open-loop control, closed-loop with PID control, and closed-loop with PID and hysteresis compensation. The filtered system output and the reference input are stored.

Section 3 is the graphical programs in Labview. Front Panel shows the signals and the controlled variables, where pushbutton *(unsafe)* shows the open-loop control is chosen. If *safe* is chosen by clicking the pushbutton again, PID control is applied if $k1\_a = 0$, and PID with hysteresis compensation is applied otherwise. The experimental data are uploaded and shown in time and frequency domains.

C.1 Graphical User Interface between DSP and Labview

```
/*————————— CIN source file - observ6.c —————————
Objective:
implement the connection between the code in DSP memory and Labview
————————————————————————————————————————*/

    #include "extcode.h"
    #include < stdio.h >
```

```
#include < math.h >
#include < conio.h >
#define FREQX 0x12F5
#define CHECK1 0x12F6
#define GRAB1 0x12F8
#define SAFE1 0x12F9
#define REFX 0x1302
#define CONTR 0x1382
#define Y 0x1383
#define MAGX 0x1384
#define KDX 0x1385
#define KPX 0x1386
#define KIX 0x1387
#define K1_A 0x1306
#define FC_A 0x1307
#define SCALING 500000.
#define FSCALING 5000.
CIN MgErr CINRun(float64 *mag, float64 *freq, float64 *kp, float64 *ki,
float64 *kd, float64 *Fc_a, float64 *k1_a, int32 *safe1, float64 *ref,
float64 *control, float64 *output, int32 *check, int32 *grab1);
CIN MgErr CINRun(float64 *mag, float64 *freq, float64 *kp, float64 *ki,
float64 *kd, float64 *Fc_a, float64 *k1_a, int32 *safe1, float64 *ref,
float64 *control, float64 *output, int32 *check, int32 *grab1) {
    long int longv, longh, longl;
    outp(0x306,0x0); /*page value*/
    //Write freq1 to DSP memory
    longv = (long)((*freq)*FSCALING);
```

```
outpw(0x302, FREQX);        /*0x12F5*/
outpw(0x300,longv);
outpw(0x300,longv>>16);
// read check1 from DSP memory
outpw(0x302,CHECK1); /*0x12F6*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh<<16) ;
*check = (int32) (longv);
//Write grab1 to DSP memory
outpw(0x302, GRAB1);        /*0x12F8*/
outpw(0x300,*grab1);
outpw(0x300,(*grab1)>>16);
//Write safe1 to DSP memory
outpw(0x302, SAFE1);        /*0x12F9*/
outpw(0x300,*safe1);
outpw(0x300,(*safe1)>>16);
//read ref from Dsp memory
outpw(0x302,REFX);      /*0x1302*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh<<16) ;
*ref = (float64) (longv)/2047*4.9976;
//Write k1a to DSP memory
longv = (long)((*k1_a)*FSCALING);
outpw(0x302, K1_A);         /*0x1306*/
outpw(0x300,longv);
```

```
outpw(0x300,longv>>16);
//Write Fca to DSP memory
longv = (long)((*Fc_a)*FSCALING);
outpw(0x302, FC_A);          /*0x1307*/
outpw(0x300,longv);
outpw(0x300,longv>>16);
//read control from Dsp memory
outpw(0x302,CONTR); /*0x1382*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh<<16) ;
*control = (float64) (longv)/2047*4.9976;
//output
outpw(0x302,Y);    /*0x1383*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh << 16);
*output = (float64) (longv)/8191*4.99939;
//Write mag. to DSP memory
longv = (long)((*mag)*SCALING);
outpw(0x302, MAGX);          /*0x1384*/
outpw(0x300,longv);
outpw(0x300,longv>>16);
//Write kd to DSP memory
longv = (long)((*kd)*SCALING);
outpw(0x302, KDX);          /*0x1385*/
outpw(0x300,longv);
```

```
        outpw(0x300,longv>>16);
        //Write kp to DSP memory
        longv = (long)((*kp)*SCALING);
        outpw(0x302, KPX);          /*0x1386*/
        outpw(0x300,longv);
        outpw(0x300,longv>>16);
        //Write ki to DSP memory
        longv = (long)((*ki)*SCALING);
        outpw(0x302, KIX);          /*0x1387*/
        outpw(0x300,longv);
        outpw(0x300,longv>>16);
        return noErr;
    }//end of file
/*————————— CIN source file – d_observ.c ————————————
Objective: upload the experimental data stored in DSP memory when ready.
Outputs:
sampling_freq – sampling rate
Y2 – signal to be uploaded
————————————————————————————————————————*/
    #include "extcode.h"
    #include<stdio.h>
    #include<math.h>
    #include<conio.h>
    #define Timer 0x12F7
    #define Y 0x1388
    #define DATASIZE 5000
    #define SCALING 500000
```

```
typedef struct {
    int32 dimSize;
    float64 arg1[1];
} TD1;
typedef TD1 **TD1Hdl;
CIN MgErr CINRun(float64 *Fs, TD1Hdl Y2);
CIN MgErr CINRun(float64 *Fs, TD1Hdl Y2) {
    long int longv, longh, longl,i;
    MgErr err=noErr;
    outp(0x306,0x0);
    outpw(0x302,Timer);
    longl = inpw(0x300) & 0x0000FFFF;
    longh = inpw(0x300) & 0x0000FFFF;
    longv = longl | (longh<<16) ;
    *Fs = (float64) 400.0*SCALING/longv;
    if (err = NumericArrayResize(fD, 1L, (UHandle *) &Y2, 4*DATASIZE))
        goto out;
    (*Y2)->dimSize = 4*DATASIZE;
    for (i=0;i<4*DATASIZE;i++) {
    outpw(0x302,( Y + i ));
    longl = inpw(0x300) & 0x0000FFFF;
    longh = inpw(0x300) & 0x0000FFFF;
    longv = longl | (longh<<16) ;
        if (i < 2*DATASIZE)
            (*Y2)->arg1[i]= (float64) (longv)*4.99939/8191.0;
        else
            (*Y2)->arg1[i]= (float64) (longv)*4.9976/2047.0;
```

```
    }
    out:
    return noErr;
}//end of file
```

C.2 C Source Code for Real-time Operation System (DSP)

```
/* ———————— observer.c ————————
Objective: implement control strategies of open-loop control, closed-loop with PID
control and closed-loop with PID and hysteresis compensation
Notes:
Sampling rate = 50*10^6/(TIMPER0*8*numcalls)
where numcalls mean ReadAdc only
maximum freq of ref. sinusoid = 50Hz
Sampling rate of observer Fs = 5kHz
Sampling rate of controller Fsc=5000/20=250Hz (NSTORE=20)
sampling rate of storing data Fsd=Fs/NFILTER
Total data stored = Fs/NFILTER/freq*period
————————————————————————————————————————*/
    #define TIMPER0 1250
    #include <math.h>
    #include"d310biox.H"
    //— mem. distribution —-
    #define FREQX 0x12F5
    #define CHECK1 0x12F6
    #define TIMER 0x12F7
    #define GRAB1 0x12F8
    #define SAFE1 0x12F9
```

```
#define REFX 0x1302
#define K1_A    0x1306
#define FC_A 0x1307
#define CONTR 0x1382
#define Y 0x1383
#define MAGX 0x1384
#define KDX 0x1385
#define KPX 0x1386
#define KIX 0x1387
#define MEMORY1_START 0x1388
#define MEMORY2_START 0x3A98
#define DATA_SIZE 1000
#define PI 3.14159265358979
#define SCALING 500000.
#define FSCALING 5000.
#define NSTORE 20
#define NFILTER 11
int * check1, *grab1, *safe1, *contr, *int_y;
long * int_ptr[2],*mag, *freq,*k1a,*fca,*kp,*ki,*kd;
double Fs= (double)50000000/(TIMPER0*8*1),ref = 0;
/* ---------- Signal function ----------
```

Objective: generate reference sinusoidal

Inputs:

(*freq)/FSCALING – frequency of reference signal

(*mag) /SCALING – magnitude of reference signal

Fs – sampling rate

Outputs:

```
ref – reference sine wave(−5 => +5)
gain_c – for compensating dynamic gain
Note: Fs=5KHz,max_freq=50Hz <=Fs/freq=100
------------------------------------------------------------ */
    double Signal(void){
        static double msin[100] = {0, 6.279051952931337e-002,
            1.253332335643043e-001, 1.873813145857246e-001,
            2.486898871648548e-001, 3.090169943749474e-001,
            3.681245526846780e-001, 4.257792915650727e-001,
            4.817536741017153e-001, 5.358267949789967e-001,
            5.877852522924731e-001, 6.374239897486898e-001,
            6.845471059286887e-001, 7.289686274214116e-001,
            7.705132427757893e-001, 8.090169943749475e-001,
            8.443279255020151e-001, 8.763066800438637e-001,
            9.048270524660196e-001, 9.297764858882515e-001,
            9.510565162951535e-001, 9.685831611286311e-001,
            9.822872507286887e-001, 9.921147013144779e-001,
            9.980267284282716e-001, 1.000000000000000,
            9.980267284282716e-001, 9.921147013144778e-001,
            9.822872507286886e-001, 9.685831611286311e-001,
            9.510565162951535e-001, 9.297764858882514e-001,
            9.048270524660195e-001, 8.763066800438635e-001,
            8.443279255020150e-001, 8.090169943749475e-001,
            7.705132427757893e-001, 7.289686274214114e-001,
            6.845471059286885e-001, 6.374239897486895e-001,
            5.877852522924733e-001, 5.358267949789967e-001,
            4.817536741017152e-001, 4.257792915650725e-001,
```

3.681245526846778e-001, 3.090169943749471e-001,

2.486898871648548e-001, 1.873813145857246e-001,

1.253332335643041e-001, 6.279051952931314e-002,

1.224606353822377e-016, -6.279051952931335e-002,

-1.253332335643043e-001, -1.873813145857248e-001,

-2.486898871648546e-001, -3.090169943749473e-001,

-3.681245526846779e-001, -4.257792915650727e-001,

-4.817536741017154e-001, -5.358267949789968e-001,

-5.877852522924730e-001, -6.374239897486896e-001,

-6.845471059286887e-001, -7.289686274214116e-001,

-7.705132427757894e-001, -8.090169943749473e-001,

-8.443279255020149e-001, -8.763066800438636e-001,

-9.048270524660198e-001, -9.297764858882515e-001,

-9.510565162951535e-001, -9.685831611286310e-001,

-9.822872507286887e-001, -9.921147013144779e-001,

-9.980267284282716e-001, -1.000000000000000,

-9.980267284282716e-001, -9.921147013144779e-001,

-9.822872507286886e-001, -9.685831611286311e-001,

-9.510565162951536e-001, -9.297764858882512e-001,

-9.048270524660196e-001, -8.763066800438634e-001,

-8.443279255020150e-001, -8.090169943749476e-001,

-7.705132427757890e-001, -7.289686274214116e-001,

-6.845471059286883e-001, -6.374239897486896e-001,

-5.877852522924734e-001, -5.358267949789963e-001,

-4.817536741017153e-001, -4.257792915650722e-001,

-3.681245526846779e-001, -3.090169943749476e-001,

-2.486898871648545e-001, -1.873813145857247e-001,

```
        -1.253332335643038e-001, -6.279051952931326e-002};
    double static coe_g[4]={-9.7409e-003,1.0593e-001,-4.5859e-001,1.8338};
    static long m=0, k=0;
    double dmag, gain_c;
    int ref_c=0;
    //dynamic gain compensation
    dmag =(double) (*mag)/SCALING;
    gain_c=dmag*dmag;
    gain_c = coe_g[0]*gain_c*dmag+coe_g[1]*gain_c+coe_g[2]*dmag+coe_g[3];
    //ref sinusoid
    ref_c = (int)(Fs*FSCALING / (* freq)/100);
    if (m == ref_c) {
        if (k >= 100) k=0;
        ref = msin[k] * dmag;
        k++;
        m=-1;
    }
    if (m > ref_c) m=-1;
    m++;
    return gain_c;
}
/* ---------- Observer function ----------
Objective: predict the output of hysteresis
Inputs:
*contr - normalized controller output(v);
gain_c - dynamic gain compensation
*int_y - normalized sensor output(v), 1st state
```

```
Fs  - sampling rate
*safe1 - initialize state of observer
Output:
Fout - hysteresis output
------------------------------------------------------ */
    double Observer(double gain_c) {
        static int x1_old = 0;
        static double x2 = 0, Fout = 0,
        //e:\xuemei\backup\thesis\proposal\estimate\exp\param.m
        coe[5] = { 5.0e-001, 3.587455094129822e-001, -1.478006271299153,
        6.483578033886975, -1.696103904526040e+003};
        if((*safe1)==1){      //open loop, initialize
            x2=Fout=0;
        }
        //nonlinear part
        Fout = Fout+(x2-Fout*fabs(x2)/(*fca)*FSCALING)/Fs;
        //linear part
        x2 = coe[0]*x2 + coe[1]*(*int_y) + coe[2]*x1_old +coe[3]*(*contr)/gain_c
+ coe[4]*Fout;
        //iteration
        x1_old = *int_y;
        return Fout;
    }
/*---------- store date ----------------
Objective: filter and store data.
inputs:
data1 - Y from A/D converter;
```

```
data2 - reference generated from computer
* grab1 - control signal
DATA_SIZE - #data stored
NFILTER - #data averaged
-------------------------------------*/
    void StoreData(long data1,long data2) {
        static long ndata=0,dummy=0,sumd[2]={0,0};
        //int NFILTER;
        //NFILTER=(int)(10.0*FSCALING /(double) (* freq))+1;
        if ((*grab1) == 1 ) {
            if (ndata < DATA_SIZE ) {
                if (dummy < NFILTER) {
                    sumd[0] = sumd[0] + data1;
                    sumd[1] = sumd[1] + data2;
                    dummy++;
                }
                if (dummy == NFILTER) {
                    *(int_ptr[0]+ndata)=(long)((double)(sumd[0])/NFILTER);
                    *(int_ptr[1]+ndata)=(long)((double)(sumd[1])/NFILTER);
                    ndata++;
                    dummy = 0;
                    sumd[0] = sumd[1]=0;
                }
            }
            else
                *check1= (int)(1);
        }//endif
```

```
        else
        {
            *check1 =(int)(0);
            ndata= dummy=0;
            sumd[0]=sumd[1]=0;
        }
    }
/*————————— PID controller —————————
Objective: eliminate drift effect and implement precision control
Inputs:
avg_ref – normalized ref input
avg_y   – normalized system output
*ki, *kp, *kd – PI control parameters in interger
SCALING – coefficient to calculate ki and kp
Fs – sampling rate
NSTORE – coeff. to compute control sampling rate
Output:
errc – output of PID controller
————————————————————————————————————*/
    double Control(double avg_ref, double avg_y) {
        double errc;
        static double sumc, err_old = 0;
        if ( (*ki)==0 ) sumc=0;      //reset integral term
        errc = avg_ref-avg_y/8191.0*2047.0;
        sumc = sumc + (double)(*ki)/SCALING/Fs*NSTORE*errc;
        if((sumc>=2047.0) && (errc>0))
            sumc = 2047.0;
```

```
        if((sumc<=-2047.0) && (errc<0))
            sumc = -2047.0;
        errc = errc*(*kp)/SCALING + sumc + (errc-err_old)*(*kd)/SCALING;
        err_old = errc;
        return errc;
    }
/*————————— Filter1 function ————————————
Objectives: eliminate noise and implement down-sampling
Inputs:
y1 – data from A/D converter;
ref – reference sinusoid input(−5 => +5)
Fout  – observer output
NSTORE – #data averaged
(*k1a) – coefficient for hysterisis compensation
gain_c – nonlinear scalar factor compensation Outputs:
y_avg – avg. output
*contr – controller output
Note:
set y_avg to be static in order to keep it constant between NSTORE points
————————————————————————————————*/
    double Filter1(int y1,double Fout,double gain_c) {
        static long ndummy=0;
        static double sumf[3]={0,0,0}, y_avg=0, ref_avg=0, F_avg=0;
        // average filter
        if (ndummy < NSTORE) {
            sumf[0] = sumf[0] + (double)y1;
            sumf[1] = sumf[1] + ref;
```

```
            sumf[2] = sumf[2] + Fout;
            ndummy++;
        }
        if (ndummy == NSTORE) {
            y_avg =sumf[0]/NSTORE;
            ref_avg =sumf[1]/NSTORE*2047/4.9976;
            F_avg =sumf[2]/NSTORE;
            ndummy=0;
            sumf[0]= sumf[1]= sumf[2]=0;
        }
        //controller output
        if ((*safe1) == 1 ) {          //open-loop sinusoid with Fs=5KHz
            //*contr = (int)(ref*2047.0/4.9976); //test sinusoid
            *contr=(int)(ref*2047.0/4.9976*gain_c);//scalar factor compensation
        }
        else          //hysteresis compensation:
            //*contr = (int)((ref_avg + (double)(*k1a) / FSCALING * F_avg) *
gain_c);
            //hysteresis compensation (*k1a!=0) + PID controller
            *contr = (int)((Control(ref_avg,y_avg) + (double)(*k1a) / FSCALING
* F_avg) * gain_c);
        return y_avg;
    }
//---------- main function ----------

    void main() {
        double yf, y_obv, gain_c;
        long *int_time, *int_ref;
```

```
InitDsp();
//————————— set mem. address ————————————-
//for store data
check1      = (int *) CHECK1; //0x12F6
* check1   = (int) (0); //1: data in mem. ready
grab1       = (int *) GRAB1; //0x12F8
* grab1    = (int) (0); //1=store;0=reset
safe1 = (int *) SAFE1; //0x12F9
* safe1 = (int) (1); //0=compensated;1=open loop
int_ptr[0]   = (long *) MEMORY1_START; //0x1388
int_ptr[1]   = (long *) MEMORY2_START; //0x3A98
//for signals
freq      = (long *) FREQX; //0x12F5
*freq = (int) (0.05*FSCALING);
int_time   = (long *) TIMER; //0x12F7
*int_time   = (long) (SCALING*400/Fs*NSTORE);
int_ref = (long *) REFX;    //0x1302
*int_ref = (int)(0);
contr   = (int *) CONTR;    //0x1382 control
*contr   = (int) (0);
int_y      = (int *) Y;          //0x1383 ReadAdc
*int_y      = (int)(0);
mag      = (long *) MAGX; //0x1384 mag. of ref.
*mag      = (int) (0.0*SCALING);
kd = (long *) KDX; //0x1385
* kd = (int) (0*SCALING);
kp = (long *) KPX; //0x1386
```

```
    * kp = (int) (0*SCALING);
    ki = (long *) KIX; //0x1387
    * ki = (int) (0*SCALING);
    //adjust hysteresis parameters
    k1a     = (long *) K1_A; //0x1306
    *k1a = (int) (0*FSCALING);
    fca   = (long *) FC_A; //0x1307
    *fca = (int) (3.0*FSCALING);
    WriteDACs(0,0);
    while(1) {
        *int_y = - ReadAdc(1);
        gain_c = Signal();       //output=ref, gain_c
        y_obv= Observer(gain_c);    // output=F[n]
        *int_ref = (long)(ref*2047.0/4.9976);
        yf = Filter1(*int_y, y_obv, gain_c);//outputs=avg. *int_y, *contr
        WriteDACs((int)(y_obv*2047/4.9976),*contr);
        //WriteDAC(*contr,1);
        StoreData((long)*int_y, *int_ref);
    } //endwhile
}//end of file
```

C.3 Labview Program

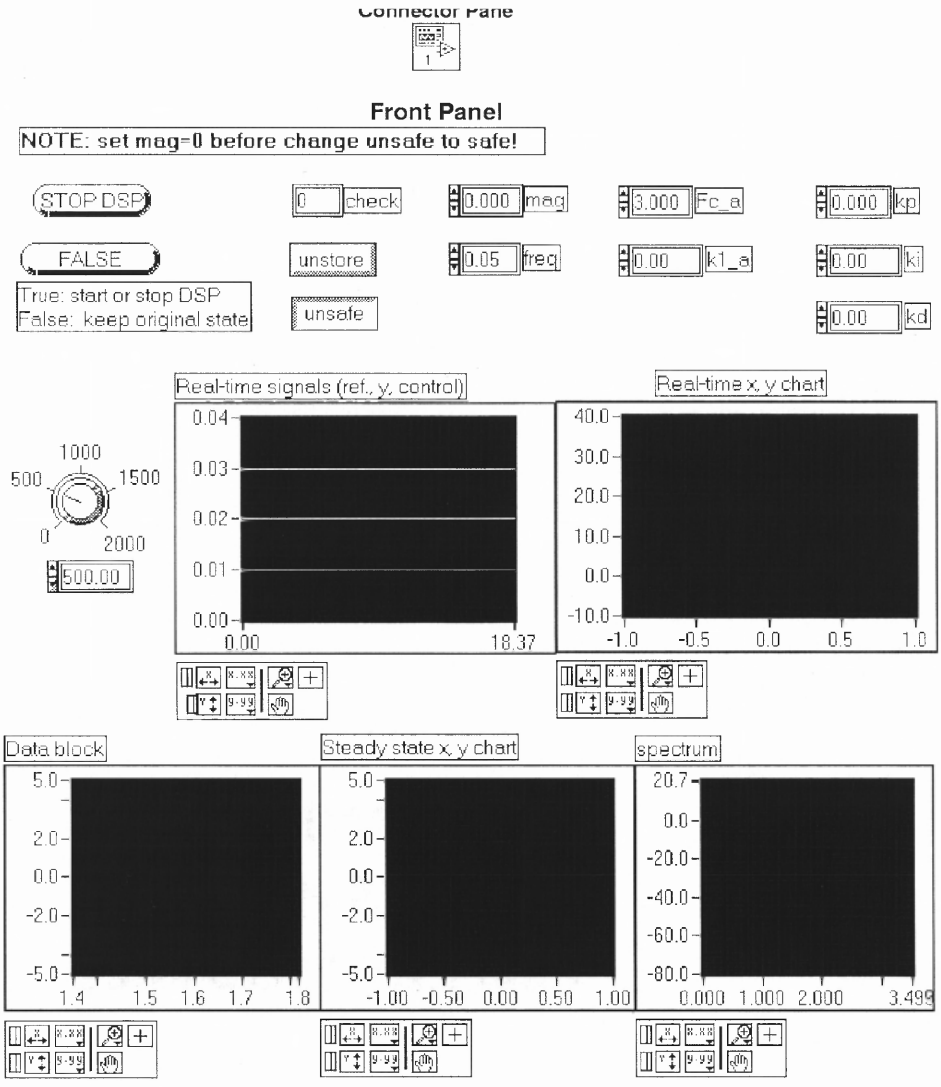

Figure C.1 Labview program of hysteresis compensation

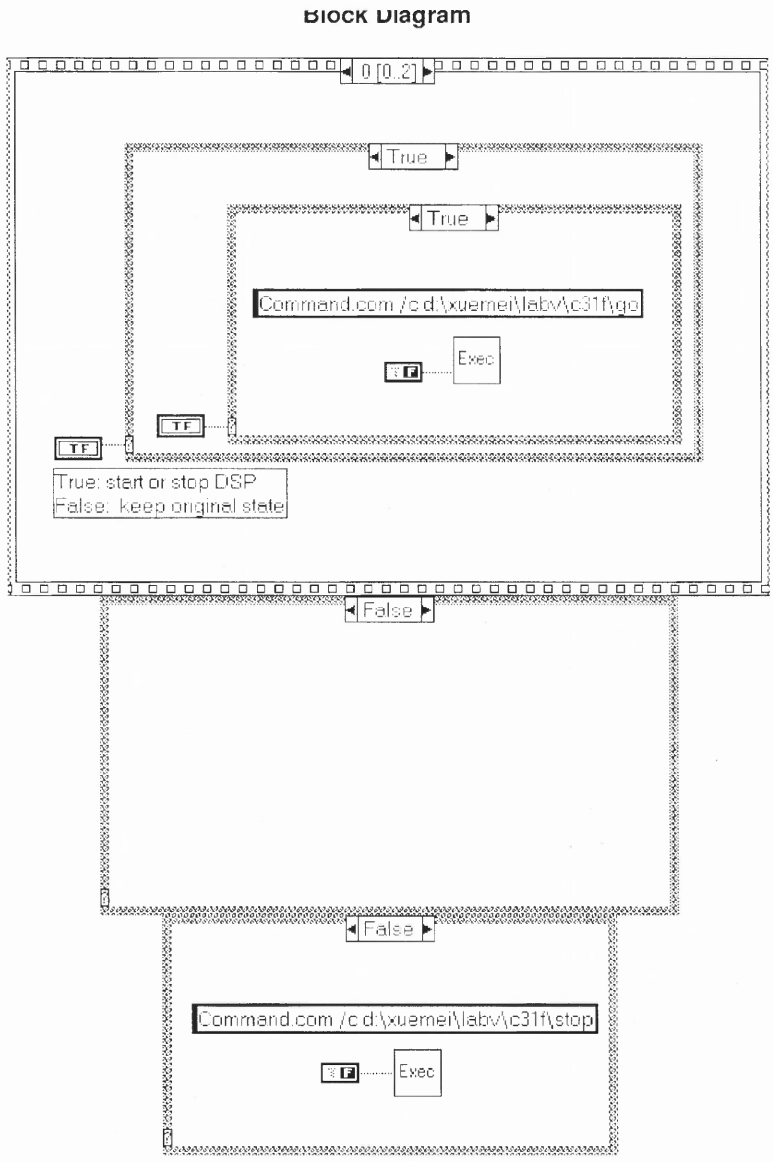

Figure C.2 Labview program of hysteresis compensation (continued)

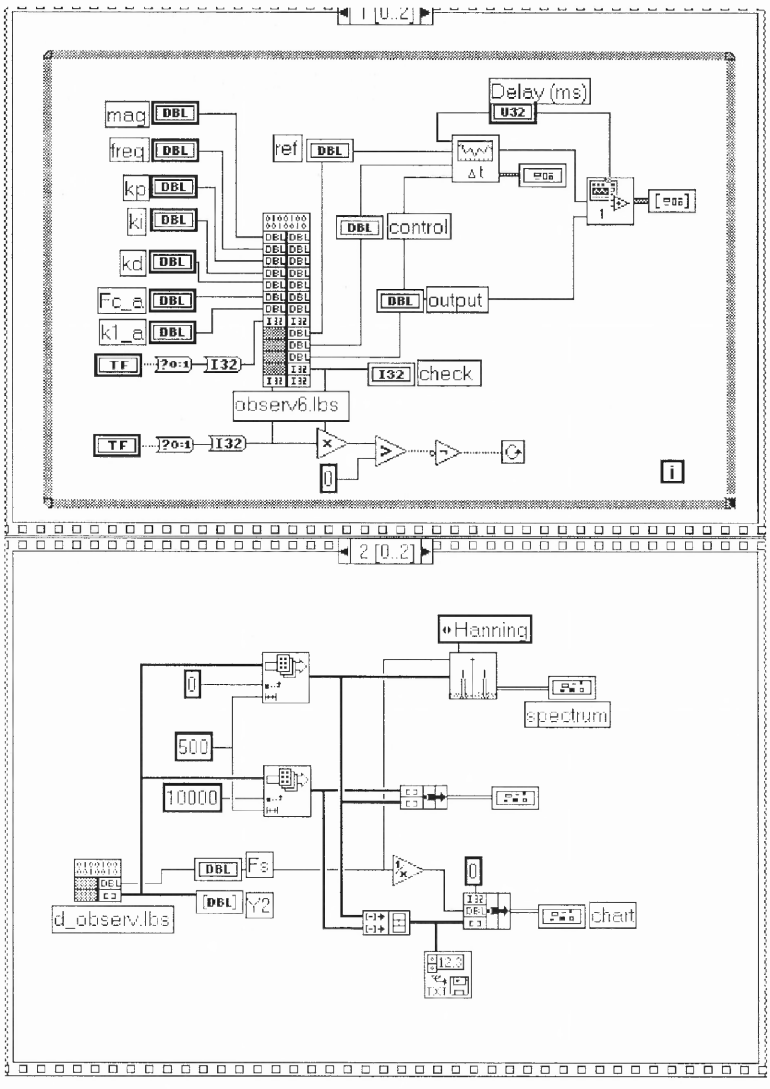

Figure C.3 Labview program of hysteresis compensation (continued)

APPENDIX D

ADAPTIVE HYSTERESIS COMPENSATION

The following code implements the control strategy of the adaptive hysteresis observer compensation, where reference input is sinusoid.

Section 1 implements the GUI between the code in DSP memory and Labview. The program allows the user to change the magnitude and the frequency of the sinusoid and the PID coefficients, decide when to store the experimental data, switch between open-loop and closed-loop control strategies, and define the filter in Labview.

Section 2 implements the real-time control strategies, where average filter is applied to implement the functions of eliminating background noise and down-sampling. The code is designed to have the flexibility of switching between open-loop control, and closed-loop with PID and adaptive hysteresis compensation. The filtered system output and the reference input are stored.

Section 3 is the graphical programs in Labview. Front Panel shows the signals and the controlled variables, where pushbutton *(unsafe)* shows the open-loop control is chosen. The closed-loop PID control with adaptive hysteresis compensation is chosen by clicking the pushbutton again(*safe* will be shown). The experimental data are uploaded and shown in time and frequency domains.

D.1 Graphical User Interface between DSP and Labview

```
/*————————— CIN source file - obadp2.c —————————
Objective: implement the connection between the code in DSP memory and Labview
————————————————————————————————————————*/

    #include "extcode.h"
    #include <stdio.h>
```

```
#include <math.h>
#include <conio.h>
#define FREQX 0x12F5
#define CHECK1 0x12F6
#define GRAB1 0x12F8
#define SAFE1 0x12F9
#define REFX 0x1302
#define CONTR 0x1382
#define Y 0x1383
#define MAGX 0x1384
#define KDX 0x1385
#define KPX 0x1386
#define KIX 0x1387
#define SCALING 500000.
#define FSCALING 5000.
```

CIN MgErr CINRun(float64 *mag, float64 *freq, float64 *kp, float64 *ki, float64 *kd, int32 *safe1, float64 *ref, float64 *control, float64 *output, int32 *check, int32 *grab1);

CIN MgErr CINRun(float64 *mag, float64 *freq, float64 *kp, float64 *ki, float64 *kd, int32 *safe1, float64 *ref, float64 *control, float64 *output, int32 *check, int32 *grab1) {

```
long int longv, longh, longl;
outp(0x306,0x0);        /*page value*/
//Write freq1 to DSP memory
longv = (long)((*freq)*FSCALING);
outpw(0x302, FREQX);      /*0x12F5*/
outpw(0x300,longv);
```

```
outpw(0x300,longv>>16);
// read check1 from DSP memory
outpw(0x302,CHECK1);        /*0x12F6*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh << 16) ;
*check = (int32) (longv);
//Write grab1 to DSP memory
outpw(0x302, GRAB1);        /*0x12F8*/
outpw(0x300,*grab1);
outpw(0x300,(*grab1)>>16);
//Write safe1 to DSP memory
outpw(0x302, SAFE1);        /*0x12F9*/
outpw(0x300,*safe1);
outpw(0x300,(*safe1)>>16);
//read ref from Dsp memory
outpw(0x302,REFX);          /*0x1302*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh <<16) ;
*ref = (float64) (longv)/2047*4.9976;
//read control from Dsp memory
outpw(0x302,CONTR);         /*0x1382*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh <<16) ;
*control = (float64) (longv)/2047*4.9976;
```

```
//output
outpw(0x302,Y);    /*0x1383*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh << 16);
*output = (float64) (longv)/8191*4.99939;
//Write mag. to DSP memory
longv = (long)((*mag)*SCALING);
outpw(0x302, MAGX);       /*0x1384*/
outpw(0x300,longv);
outpw(0x300,longv >>16);
//Write kd to DSP memory
longv = (long)((*kd)*SCALING);
outpw(0x302, KDX);       /*0x1385*/
outpw(0x300,longv);
outpw(0x300,longv>>16);
//Write kp to DSP memory
longv = (long)((*kp)*SCALING);
outpw(0x302, KPX);       /*0x1386*/
outpw(0x300,longv);
outpw(0x300,longv>>16);
//Write ki to DSP memory
longv = (long)((*ki)*SCALING);
outpw(0x302, KIX);       /*0x1387*/
outpw(0x300,longv);
outpw(0x300,longv>>16);
return noErr;
```

```
} //end of file
/*------------ CIN source file - d_observ.c ------------

Objective: upload the experimental data from the DSP memory to Labview

-------------------------------------------------------*/

#include "extcode.h"
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define Timer 0x12F7
#define Y 0x1388
#define DATASIZE 5000
#define SCALING 500000
typedef struct {
        int32 dimSize;
        float64 arg1[1];
} TD1;
typedef TD1 **TD1Hdl;
CIN MgErr CINRun(float64 *Fs, TD1Hdl Y2);
CIN MgErr CINRun(float64 *Fs, TD1Hdl Y2) {
        long int longv, longh, longl,i;
        MgErr err=noErr;
        outp(0x306,0x0);
        outpw(0x302,Timer);
        longl = inpw(0x300) & 0x0000FFFF;
        longh = inpw(0x300) & 0x0000FFFF;
        longv = longl | (longh << 16) ;
        *Fs = (float64) 400.0*SCALING/longv;
```

```
        if (err = NumericArrayResize(fD, 1L, (UHandle *) &Y2, 4*DATASIZE))
                goto out;
        (*Y2)->dimSize = 4*DATASIZE;
        for (i=0;i<4*DATASIZE;i++) {
                outpw(0x302,( Y + i ));
                longl = inpw(0x300) & 0x0000FFFF;
                longh = inpw(0x300) & 0x0000FFFF;
                longv = longl | (longh<<16) ;
                if (i < 2*DATASIZE)
                        (*Y2)->arg1[i]= (float64) (longv)*4.99939/8191.0;
                else
                        (*Y2)->arg1[i]= (float64) (longv)*4.9976/2047.0;
        }
        out:
        return noErr;
} //end of file
```

D.2 C Source Code for Real-time Operation System (DSP)

```
/* ———— Adaptive Observer obs_adp2.c ————
Objective:
implement the control strategies of open loop control and
closed-loop with PID and adaptive hysteresis observer compensation
Notes:
Sampling rate of observer Fs = 5kHz
Sampling rate of controller Fsc = 5000/20 = 250Hz (NSTORE = 20)
Sampling rate of storing data Fsd = Fs/NFILTER
Total data stored = Fs/NFILTER/freq*period
```

```
NFILTER*freq = 10 => nsample/period = 500
(period = 2 and DATA_SIZE = 1000)
-------------------------------------------------------*/
    #define TIMPER0 1250
    #include <math.h>
    #include "d310biox.H"
    //--- mem. distribution ---
    #define FREQX 0x12F5
    #define CHECK1 0x12F6
    #define TIMER 0x12F7
    #define GRAB1 0x12F8
    #define SAFE1 0x12F9
    #define REFX 0x1302
    #define CONTR 0x1382
    #define Y 0x1383
    #define MAGX 0x1384
    #define KPX 0x1386
    #define KIX 0x1387
    #define MEMORY1_START 0x1388
    #define MEMORY2_START 0x3A98
    #define DATA_SIZE 1000
    #define PI 3.14159265358979
    #define SCALING 500000.
    #define FSCALING 5000.
    #define NSTORE 20
    #define NFILTER 100
    int * check1, *grab1, *safe1, *contr, *int_y;
```

```
        long * int_ptr[2],*mag, *freq,*kp,*ki;
        double Fs= (double)50000000/(TIMPER0*8*1),ref = 0,dmag, freqd, k1a=0,
fca=3;
/* ————————- Signal function ————————————
Objective: generate reference sinusoid
Inputs:
(*freq)/FSCALING – frequency of the sinusoid
(*mag) /SCALING – magnitude of the sinusoid
Fs – sampling rate
Outputs:
ref – reference sine wave(between -5 and 5)
gain_c – for compensating dynamic gain
freqd – for calculate hysteresis param
Note: Fs=5KHz,max_freq=50Hz <=Fs/freq=100
————————————————————————————— */
        double Signal(void) {
            static double msin[100] = { 0,   6.279051952931337e-002,
                1.253332335643043e-001,   1.873813145857246e-001,
                2.486898871648548e-001,   3.090169943749474e-001,
                3.681245526846780e-001,    4.257792915650727e-001,
                4.817536741017153e-001,    5.358267949789967e-001,
                5.877852522924731e-001,    6.374239897486898e-001,
                6.845471059286887e-001,    7.289686274214116e-001,
                7.705132427757893e-001,    8.090169943749475e-001,
                8.443279255020151e-001,    8.763066800438637e-001,
                9.048270524660196e-001,    9.297764858882515e-001,
                9.510565162951535e-001,   9.685831611286311e-001,
```

9.822872507286887e-001, 9.921147013144779e-001,

9.980267284282716e-001, 1.000000000000000,

9.980267284282716e-001, 9.921147013144778e-001,

9.822872507286886e-001, 9.685831611286311e-001,

9.510565162951535e-001, 9.297764858882514e-001,

9.048270524660195e-001, 8.763066800438635e-001,

8.443279255020150e-001, 8.090169943749475e-001,

7.705132427757893e-001, 7.289686274214114e-001,

6.845471059286885e-001, 6.374239897486895e-001,

5.877852522924733e-001, 5.358267949789967e-001,

4.817536741017152e-001, 4.257792915650725e-001,

3.681245526846778e-001, 3.090169943749471e-001,

2.486898871648548e-001, 1.873813145857246e-001,

1.253332335643041e-001, 6.279051952931314e-002,

1.224606353822377e-016, -6.279051952931335e-002,

-1.253332335643043e-001, -1.873813145857248e-001,

-2.486898871648546e-001, -3.090169943749473e-001,

-3.681245526846779e-001, -4.257792915650727e-001,

-4.817536741017154e-001, -5.358267949789968e-001,

-5.877852522924730e-001, -6.374239897486896e-001,

-6.845471059286887e-001, -7.289686274214116e-001,

-7.705132427757894e-001, -8.090169943749473e-001,

-8.443279255020149e-001, -8.763066800438636e-001,

-9.048270524660198e-001, -9.297764858882515e-001,

-9.510565162951535e-001, -9.685831611286310e-001,

-9.822872507286887e-001, -9.921147013144779e-001,

-9.980267284282716e-001, -1.000000000000000,

```
    -9.980267284282716e-001,   -9.921147013144779e-001,
    -9.822872507286886e-001,   -9.685831611286311e-001,
    -9.510565162951536e-001,   -9.297764858882512e-001,
    -9.048270524660196e-001,   -8.763066800438634e-001,
    -8.443279255020150e-001,   -8.090169943749476e-001,
    -7.705132427757890e-001,   -7.289686274214116e-001,
    -6.845471059286883e-001,   -6.374239897486896e-001,
    -5.877852522924734e-001,   -5.358267949789963e-001,
    -4.817536741017153e-001,   -4.257792915650722e-001,
    -3.681245526846779e-001,   -3.090169943749476e-001,
    -2.486898871648545e-001,   -1.873813145857247e-001,
    -1.253332335643038e-001,   -6.279051952931326e-002};
double static coe_g[4]={-9.7409e-003,1.0593e-001,-4.5859e-001,1.8338};
static long m=1,k=0;
double gain_c;
int ref_c=0;
//dynamic gain compensation
dmag =(double) (*mag)/SCALING;
gain_c=dmag*dmag;
gain_c=coe_g[0]*gain_c*dmag+coe_g[1]*gain_c+coe_g[2]*dmag+coe_g[3];
//ref sinusoid
freqd=(double)(* freq)/FSCALING;
ref_c = (int)(Fs/freqd/100);
if (m == ref_c) {
    if (k >= 100) k=0;
    ref = msin[k] * dmag;
    k++;
```

```
            m=0;
        }
        if (m > ref_c) m=0;
        m++;
        return gain_c;
    }
/* ---------- Observer function ----------
Objective: implement a reduced order obserser to predict hysteresis output
Inputs:
contr – normalized controller output(v)
gain_c – dynamic gain compensator
int_y – normalized sensor output(v), 1st state
Fs – sampling rate
safe1 – initialize state of observer
k1a,fca – hysteresis parameters
Output: Fout – hysteresis output
---------- */
    double Observer(double gain_c) {
        static int x1_old = 0;
        static double x2 = 0, Fout = 0,
        coe[5] = {5.0e-001, 3.587455094150691e-001, -1.478006271301299,
                  6.483578033887466, -1.696103904526040e+003};
        coe[4] =-1.541912640478336e-004*k1a;
        if((*safe1)==1) { //open loop, initialize
            k1a = x1_old = x2 = Fout = 0;
        }
        //nonlinear part
```

```
        Fout = Fout+(x2-Fout*fabs(x2)/fca)/Fs;
        //linear part
        x2 = coe[0]*x2 + coe[1]*(*int_y) + coe[2]*x1_old +
        coe[3]*(*contr)/gain_c+coe[4]*Fout;
        //iteration
        x1_old = *int_y;
        return Fout;
    }
/*---------- adaptive algorithm ----------

Objective: calculate hysteresis parameters Fc, k1
Input: data1   - Y from A/D converter
Outputs: k1a, fca - estimated hysteresis parameters
-----------------------------------*/
    int Parm(int data1) {
        static long ndata=0,dummy=0,sumd=0,N=20;
        static double avg_y[20], kn = 1.1893e+007, kv = 1.72231e+007;
        long double X1[2]={0,0}, X3[2]={0,0};
        long n, block;
        /*---------- DFT----------*/
        block= (long)(Fs/(N*freqd));
        if (ndata < N) {
            if (dummy > block){
                ndata = dummy = sumd = 0;
            }
            else if (dummy < block) {
                sumd = sumd + data1;
                dummy++;
```

```
        }
        if (dummy == block){
            avg_y[ndata] =((double)(sumd)/block);// Y filtered
            ndata++;
            sumd = dummy = 0;
        }
    }
    if (ndata == N) {
        dummy = sumd = ndata=0;
        X1[0]=X1[1]=X3[0]=X3[1]=0;
        for (n=0; n<N; n++) {
            // X1=X1+avg_y(n)*exp(-j*2*PI*n/N);
            X1[0]=X1[0]+avg_y[n]*cos(2*PI*n/N);
            X1[1]=X1[1]-avg_y[n]*sin(2*PI*n/N);
            //X3=X3+avg_y(n)*exp(-j*2*PI*3*n/N);
            X3[0]=X3[0]+avg_y[n]*cos(2*PI*3*n/N);
            X3[1]=X3[1]-avg_y[n]*sin(2*PI*3*n/N);
        }
        X1[0]=sqrt(X1[0]*X1[0]+X1[1]*X1[1])*2/N;
        X3[0]=sqrt(X3[0]*X3[0]+X3[1]*X3[1])*2/N;
        /*————— hysteresis parameters ————————
        A0=sqrt(h1*h1-36*h3*h3); k1a=kv*mag/A0-kn;
        fca=2*k1a*A0^2/(9*pi*kn*h3); 20*log10(h3/h1/mag)>-60
        ————————————————————————————————*/
        if (dmag<0.1) k1a=0;
        else if (X1[0]>6*X3[0] && X3[0]>0 && X3[0]/X1[0]/dmag>0.001){
            X1[1]=sqrt(X1[0]*X1[0]-36*X3[0]*X3[0]);
```

```
                X3[1] = k1a+kv*dmag/X1[1]-kn;
                if (X3[1]>10){
                    k1a=X3[1];
                    fca=fca+2*k1a*X1[1]*X1[1]/(9*PI*X3[0]*kn);
                }
            }
        }
        return (int)(X1[0]*2047/4.9976);
    }
/*————————— store date ————————————————-
Objective: filter and store data
Inputs:
data1 – Y from A/D converter
data2 – reference generated from computer
grab1 – control signal from "pi2.vi"
DATA_SIZE – #data stored
NFILTER – #data averaged
————————————————————————————————-*/
    void StoreData(long data1,long data2) {
        static long ndata=0,dummy=0,sumd[2]={0,0};
        if ((*grab1) == 1 ){
            if (ndata < DATA_SIZE ){
                if (dummy < NFILTER){
                    sumd[0] = sumd[0] + data1;
                    sumd[1] = sumd[1] + data2;
                    dummy++;
                }
```

```
            if (dummy == NFILTER) {
                *(int_ptr[0]+ndata) =(long)((double)(sumd[0])/NFILTER);
                *(int_ptr[1]+ndata) =(long)((double)(sumd[1])/NFILTER);
                ndata++;
                sumd[0]=sumd[1]=dummy = 0;
            }
        }
        else
            *check1= (int)(1);
        }//endif
        else {
            *check1 =(int)(0);
            ndata= dummy=0;
            sumd[0]=sumd[1]=0;
        }
    }
/*————————— PI controller ————————————-
Objective: implement PI control which is used to eliminate steady state error
Inputs:
avg_ref – normalized ref input
avg_y   – normalized system output
ki,*kp – PI control parameters in interger
SCALING– coefficient to calculate ki and kp
Fs   – sampling rate
NSTORE– coeff. to compute control sampling rate
Output: errc
————————————————————————————————*/
```

```
double Control(double avg_ref, double avg_y) {
    double errc;
    static double sumc;
    if ( (*ki)==0 ) sumc=0; //reset integral term
    errc = avg_ref-avg_y/8191.0*2047.0;
    sumc = sumc + (double)(*ki)/SCALING/Fs*NSTORE*errc;
    if((sumc >=2047.0) && (errc > 0))
    sumc = 2047.0;
    if((sumc <=-2047.0) && (errc < 0))
    sumc = -2047.0;
    errc = errc*(*kp)/SCALING + sumc;
    return errc;
}
```

/*———————— Filter1 function ————————

Objective:

implement average filter to eliminate background noise and down-sampling

Inputs:

y1 – data from A/D converter

ref – reference input from "Signal"($-5 \Rightarrow +5$)

Fout – observer output

NSTORE – #data averaged

k1a – coefficient for hysteresis compensation

gain_c – dynamic gain compensation

Outputs:

y_avg – average output

contr – controller output

```
Note: set y_avg to be static in order to keep it constant between NSTORE points
-------------------------------------------------------------*/
    double Filter1(int y1,double Fout,double gain_c) {
        static long ndummy=0;
        static double sumf[3]={0,0,0},y_avg=0, ref_avg=0 ,F_avg=0;
        // average filter
        if (ndummy < NSTORE){
            sumf[0] = sumf[0] + (double)y1;
            sumf[1] = sumf[1] + ref;
            sumf[2] = sumf[2] + Fout;
            ndummy++;
        }
        if (ndummy == NSTORE) {
            y_avg =sumf[0]/NSTORE;
            ref_avg =sumf[1]/NSTORE*2047/4.9976;
            F_avg = sumf[2]/NSTORE;
            ndummy=0;
            sumf[0]=sumf[1]=sumf[2]=0;
        }
        //open-loop control with sampling rate Fs=5KHz
        if ((*safe1) == 1 ) {
            *contr = (int)(ref*2047.0/4.9976); //test sinusoid
            //scale factor compensation
            //*contr = (int)(ref*2047.0/4.9976*gain_c);
        }
        //hysteresis compensation
        else
```

```
            //*contr = (int)((ref_avg + 2.378181665153533e-005 * k1a * F_avg)
* gain_c);
            //with PID control, *k1a = 0 => PID controller only
            *contr = (int)((Control( ref_avg,y_avg) + 2.378181665153533e-005 *
k1a * F_avg) * gain_c);
        return y_avg;
    }
    //---------- main function ----------
    void main() {
        double yf, y_obv, gain_c;
        long *int_time, *int_ref;
        int harm;
        InitDsp();
        //---------- set mem. address ----------
        //for store data
        check1 = (int *) CHECK1;      //0x12F6
        * check1 = (int *) (0);       //1=data in mem. ready
        grab1 = (int *) GRAB1;        //0x12F8
        * grab1 = (int) (0);          //1=store;0=reset
        safe1 = (int *) SAFE1;        //0x12F9
        * safe1 = (int) (1);          //0=compensated;1=open loop
        int_ptr[0] = (long *) MEMORY1_START;      //0x1388
        int_ptr[1] = (long *) MEMORY2_START;      //0x3A98
        //for signals
        freq = (long *) FREQX;        //0x12F5
        *freq = (int) (0.1*FSCALING);
        int_time = (long *) TIMER;    //0x12F7
```

```
    *int_time = (long) (SCALING*400/Fs*NSTORE);
    int_ref = (long *) REFX;       //0x1302
    *int_ref = (int)(0);
    contr = (int *) CONTR;         //0x1382 control
    *contr = (int) (0);
    int_y = (int *) Y;         //0x1383 ReadAdc
    *int_y = (int)(0);
    mag = (long *) MAGX;       //0x1384 mag. of ref.
    *mag = (int) (0.0*SCALING);
    kp = (long *) KPX;         //0x1386
    * kp = (int) (0*SCALING);
    ki = (long *) KIX;         //0x1387
    * ki = (int) (0*SCALING);
    WriteDACs(0,0);
    while(1) {
        *int_y = - ReadAdc(1);
        gain_c = Signal();         //outputs = ref,dmag freqd, gain_c
        harm = Parm(*int_y);       //outputs = k1a, fca
        y_obv= Observer(gain_c);       //output = F[n]
        *int_ref = (long)(ref*2047.0/4.9976);
        /* ———— dither compensation ———*/
        yf=Filter1(*int_y, y_obv, gain_c); //outputs=avg. *int_y, *contr
        //WriteDACs((int)(*int_ref),*contr);
        WriteDAC(*contr,1);
        StoreData((long)*int_y, *int_ref);
    } //endwhile
} //end of file
```

D.3 Labview Program

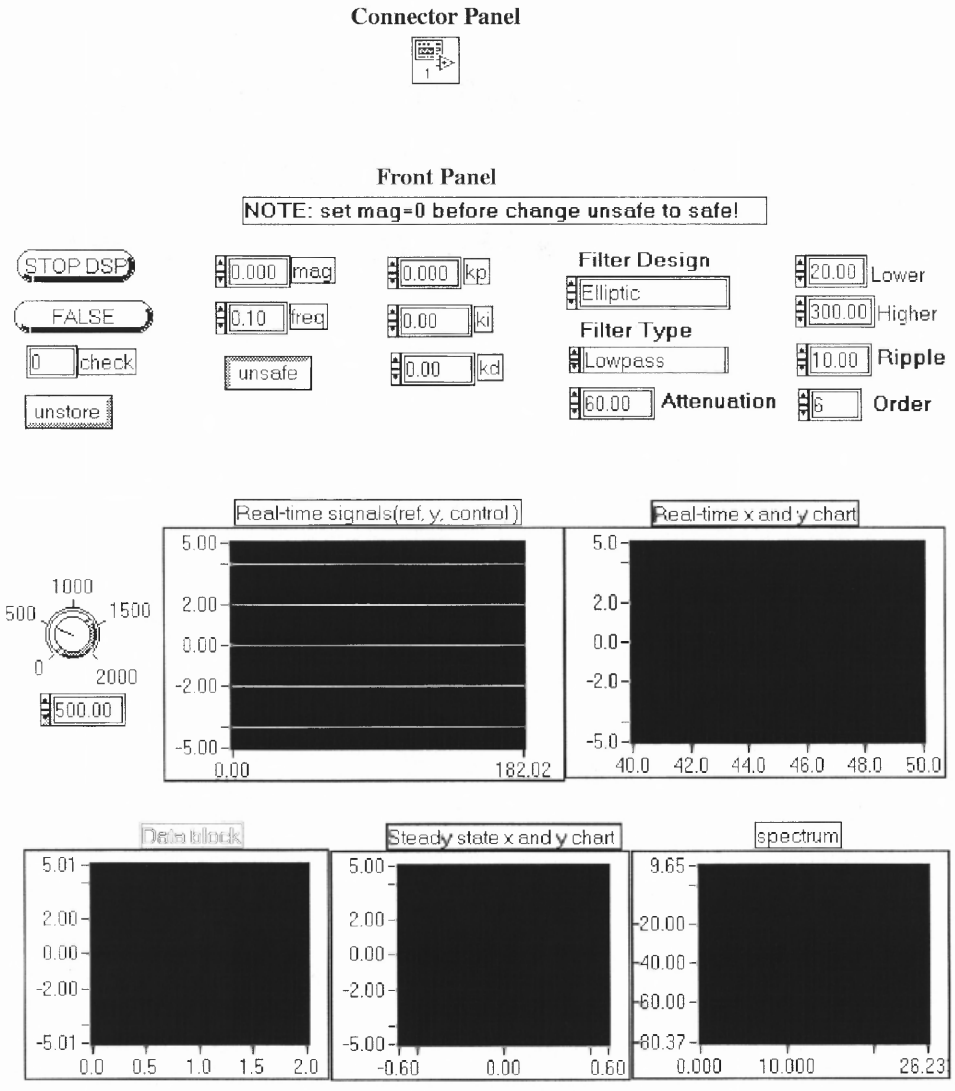

Figure D.1 Labview program of adaptive hysteresis compensation

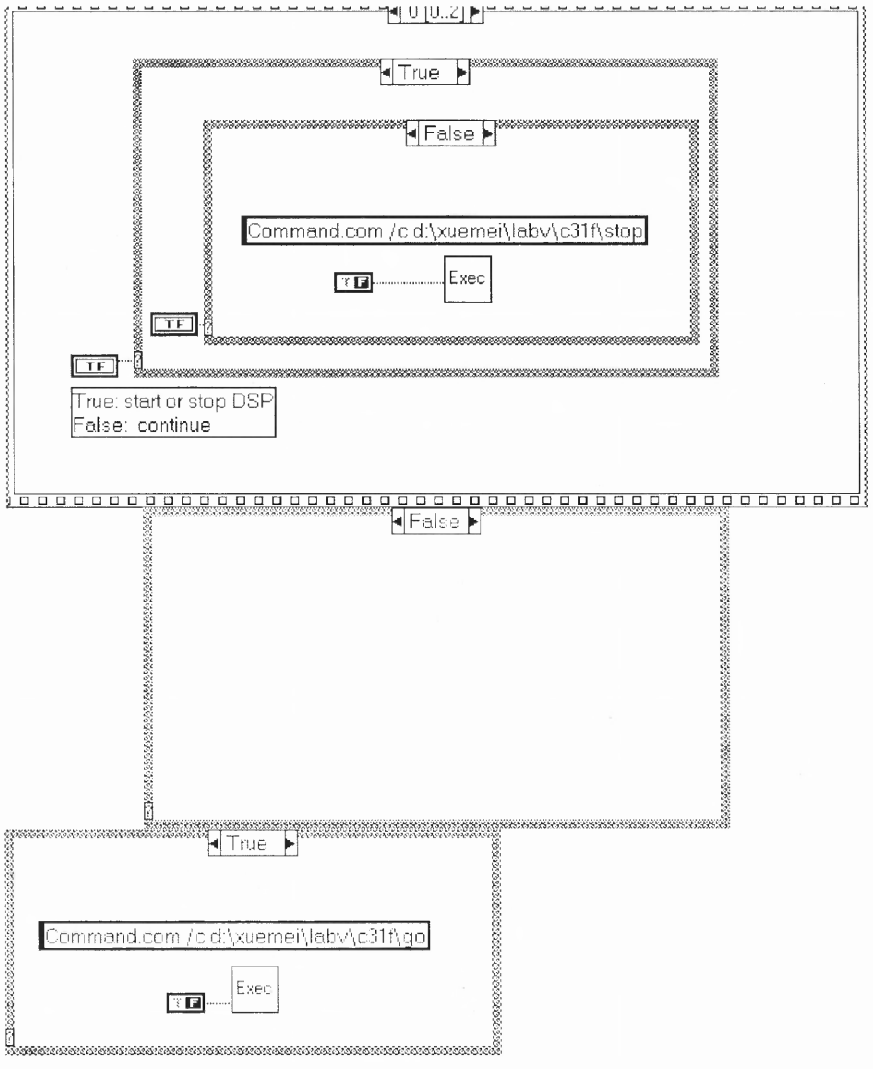

Figure D.2 Labview program of adaptive hysteresis compensation (continued)

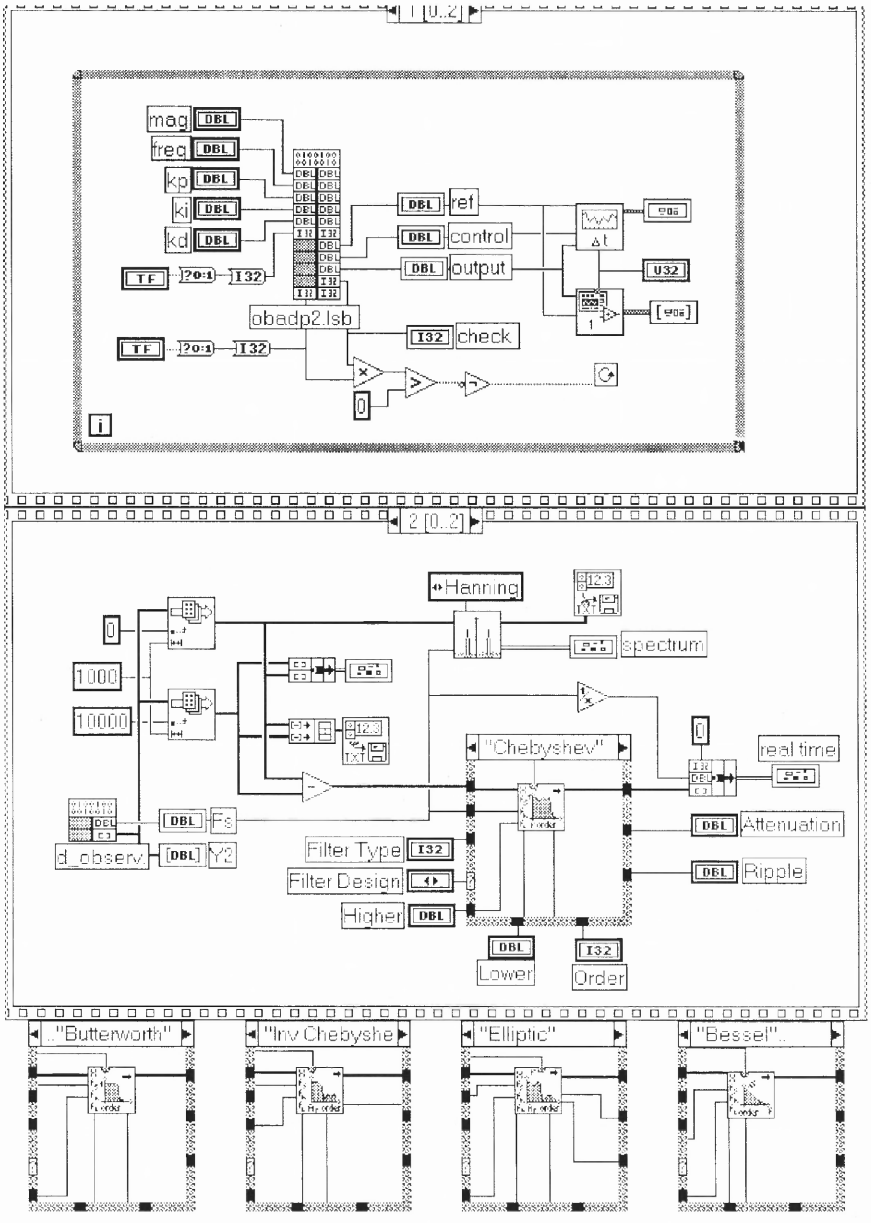

Figure D.3 Labview program of adaptive hysteresis compensation (continued)

APPENDIX E

FINE MOTION CONTROL AND TIME DIVISION MULTI-CONTROLLER

The following code implements the system control in two axes at the same time. Fine motion control is to study the system's resolution, and time division multi-controller is to implement fast transient response, low vibration, and zero steady state error.

Section 1 implements the GUI between the code in DSP memory and Labview. The program allows the user to input the magnitudes and frequencies of the input signals and the coefficients of PI-controller, and decide when to store the experimental data. Through the interface, the experimental data stored in the DSP memory are uploaded and shown in the Labview.

Section 2 implements the real-time control strategies. For time division multi-controller, the code is designed to have the flexibility of choosing open-loop control with input square or input shaper, and closed-loop PI-control with input square or input shaper. The filtered system output and the reference input are stored.

Section 3 is the graphical programs in Labview. Front Panel shows the signals and the controlled variables. Block Diagram is the code to implement the functions.

E.1 Graphical User Interface between DSP and Labview

```
/*————————— CIN source file - pi2m.c —————————————
Objective: implement the connection between the code in DSP memory and Labview
————————————————————————————————————————*/

    #include "extcode.h"
    #include <stdio.h>
    #include <math.h>
    #include <conio.h>
```

```
//— memory address distribution ———
#define FREQX    0x12F5
#define CHECK1    0x12F6
#define GRAB1    0x12F8
#define FREQY    0x12FB
#define MAGY    0x12FC
#define CONTY    0x12FD
#define OUTPUTY    0x12FE
#define KPY    0x12FF
#define KIY    0x1300
#define REFY    0x1303
#define REFX    0x1302
#define CONTX    0x1382
#define OUTPUTX    0x1383
#define MAGX    0x1384
#define KPX    0x1386
#define KIX    0x1387
#define SCALING      500000
#define FSCALING      5000
#define MASK   0x0000FFFFL
```

CIN MgErr CINRun(float64 *magx, float64 *kpx, float64 *kix, float64 *freqx, float64 *magy, float64 *kpy, float64 *kiy, float64 *freqy, float64 *refx, float64 *outputx, float64 *controlx, float64 *refy, float64 *outputy, float64 *controly, int32 *check, int32 *var14);

CIN MgErr CINRun(float64 *magx, float64 *kpx, float64 *kix, float64 *freqx, float64 *magy, float64 *kpy, float64 *kiy, float64 *freqy, float64 *refx, float64

```
*outputx, float64 *controlx, float64 *refy, float64 *outputy,float64 *controly, int32
*check, int32 *var14) {
        long int longv, longh, longl;
        outp(0x306,0x0);        /*page value*/
        //Write freqx to DSP memory
        longv = (long int)((*freqx)*FSCALING);
        outpw(0x302, FREQX);      /*0x12F5*/
        outpw(0x300,longv);
        outpw(0x300,(longv)>>16);
        // read check1 from DSP memory
        outpw(0x302,CHECK1);        /*0x12F6*/
        longl = inpw(0x300) & 0x0000FFFF;
        longh = inpw(0x300) & 0x0000FFFF;
        longv = longl | (longh<<16) ;
        *check = (int32) (longv);
        //Write grab1 to DSP memory
        outpw(0x302, GRAB1);      /*0x12F8*/
        outpw(0x300,*var14);
        outpw(0x300,(*var14)>>16);
        //Write freqy to DSP memory
        longv = (long int)((*freqy)*FSCALING);
        outpw(0x302, FREQY);      /*0x12FB*/
        outpw(0x300,longv);
        outpw(0x300,(longv)>>16);
        //Write magy to DSP memory
        longv = (long int)((*magy)*SCALING);
        outpw(0x302, MAGY);      /*0x12FC*/
```

```
outpw(0x300,longv);
outpw(0x300,(longv)>>16);
//read control_y from Dsp memory
outpw(0x302,CONTY);       /*0x12FD*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh<<16) ;
*controly = (float64) (longv)/2047*4.9976;
//read output_y from DSP
outpw(0x302,OUTPUTY);       /*0x12FE*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh << 16);
*outputy = (float64) (longv)/8191*4.99939;
//Write kpy to DSP memory
longv = (long int)((*kpy)*SCALING);
outpw(0x302, KPY);      /*0x12FF*/
outpw(0x300,longv);
outpw(0x300,(longv)>>16);
//Write kiy to DSP memory
longv = (long int)((*kiy)*SCALING);
outpw(0x302, KIY);      /*0x1300*/
outpw(0x300,longv);
outpw(0x300,(longv)>>16);
//Read refx from DSP memory
outpw(0x302,REFX);       /*0x1302*/
longl = inpw(0x300) & 0x0000FFFF;
```

```
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh << 16);
*refx = (float64) (longv)/2047*4.9976;
//Read refy from DSP memory
outpw(0x302,REFY);        /*0x1303*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh << 16);
*refy = (float64) (longv)/2047*4.9976;
//read control_x from Dsp memory
outpw(0x302,CONTX);       /*0x1382*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh<<16) ;
*controlx = (float64) (longv)/2047*4.9976;
//read output_x from DSP
outpw(0x302,OUTPUTX);       /*0x1383*/
longl = inpw(0x300) & 0x0000FFFF;
longh = inpw(0x300) & 0x0000FFFF;
longv = longl | (longh << 16);
*outputx = (float64) (longv)/8191*4.99939;
//Write magx to DSP memory
longv = (long int)((*magx)*SCALING);
outpw(0x302, MAGX);       /*0x1384*/
outpw(0x300,longv);
outpw(0x300,(longv)>>16);
//Write kpx to DSP memory
```

```
        longv = (long int)((*kpx)*SCALING);
        outpw(0x302, KPX);      /*0x1386*/
        outpw(0x300,longv);
        outpw(0x300,(longv)>>16);
        //Write kix to DSP memory
        longv = (long int)((*kix)*SCALING);
        outpw(0x302, KIX);      /*0x1387*/
        outpw(0x300,longv);
        outpw(0x300,(longv)>>16);
        return noErr;
    } //end of file
/*————————— CIN source file - data_s.c ————————————
```

Objective: upload the experimental data stored in the DSP memory when ready.

Outputs:

sampling_freq - sampling rate

Y2 - signal to be plotted

```
————————————————————————————————————————*/
    #include "extcode.h"
    #include <stdio.h>
    #include <math.h>
    #include <conio.h>
    #define Timer 0x12F7
    #define Y 0x1388
    #define MASK 0x0000FFFFL /* Should this be a long? */
    #define DATASIZE 5000
    #define SCALING 500000.
    typedef struct {
```

```
    int32 dimSize;
    float64 arg1[1];
} TD1;
typedef TD1 **TD1Hdl;
CIN MgErr CINRun(float64 *Sampling_freq, TD1Hdl Y2);
CIN MgErr CINRun(float64 *Sampling_freq, TD1Hdl Y2) {
    long int longv, longh, longl,i;
    MgErr err=noErr;
    outp(0x306,0x0);
    outpw(0x302,Timer);
    longl = inpw(0x300) & 0x0000FFFF;
    longh = inpw(0x300) & 0x0000FFFF;
    longv = longl | (longh<<16) ;
    *Sampling_freq = (float64) 400.0*SCALING/longv;
    if (err = NumericArrayResize(fD, 1L, (UHandle *) &Y2, 8*DATASIZE))
        goto out;
    (*Y2)->dimSize = 8*DATASIZE;
    for (i=0;i<8*DATASIZE;i++) {
        outpw(0x302,( Y + i ));
        longl = inpw(0x300) & 0x0000FFFF;
        longh = inpw(0x300) & 0x0000FFFF;
        longv = longl | (longh<<16) ;
        (*Y2)->arg1[i]= (float64) (longv)/8191*4.99939;
    }
    out:
    return noErr;
} //end of file
```

E.2 C Source Codes for Real-time Operation System (DSP)

E.2.1 Time Division Multi-Controller

```
/* ———————— shadpi5.c ————————

Objective: implement the strategy of time division multi-controller

Note: Check the maximum sampling rate.

————————————————————————————————————-*/

    #define TIMPER0 0x138

    #include <math.h>

    #include "d310biox.H"

    //— memory address distribution ———

    #define CHECK1 0x12F6

    #define TIMER 0x12F7

    #define GRAB1 0x12F8

    #define MAGY 0x12FC

    #define CONTY 0x12FD

    #define OUTPUTY 0x12FE

    #define KPY 0x12FF

    #define KIY 0x1300

    #define FREQX 0x12F5

    #define CONTX 0x1382

    #define OUTPUTX 0x1383

    #define MAGX 0x1384

    #define KPX 0x1386

    #define KIX 0x1387

    #define MEMORY1_START 0x1388

    #define MEMORY2_START 0x61A8

    //———— table of constant ————-

    #define DATA_SIZE 20000
```

```
#define PI 3.14159265358979
#define SCALING 500000.
#define FSCALING 5000.
#define NSAMPLES 40
#define NSTORE 1
/* ———— global variables ————

Objective: use global variables to increase the program's speed

————————————————————————*/
int * check1, *grab1, *cont[2], delayl[2], delayh[2];
long * int_ptr[2],*output1[2],*kp[2], *ki[2],*mag[2];
double Fs = (double)50000000/(TIMPER0*8*2),dref[2],
poly[2][4]={0,0,0,0,-9.7409e-003,1.0593e-001,-4.5859e-001,1.8338},
Mph[2]={ 1.5419,7.0e-001},Mpl[2]={8.8649e-001,7.0e-001},
tdelayh[2]={0,9.2417e-004}, tdelayl[2]={0,9.2417e-004},
// tdelayh[2]={0,0}, tdelayl[2]={0,0},
gshaper[2]={1,1}, A1h[2],A2h[2],A3h[2],A1l[2],A2l[2],A3l[2];
/*———————————— Input shaping ————————————

Objective: implement ZVD shaper signal
Inputs:
nsamples – total #samples in half square period
ncurrent – current sample time
mag[2] – amplitude of square wave
poly[2][4] – coefficients. of nonlinear scalar factor x and y axis
gshaper[2] – nonlinear scalar factor compensator
Outputs:
gshaper[2] – used in PI control to obtain reference
```

```
dref[2] – double shaper signals
-------------------------------------------------------*/
    void Shaperf(long nsamples,long ncurrent,int channel) {
        double dmag;
        dmag= (double)(*mag[channel])/SCALING;
        gshaper[channel]=dmag*dmag;
        gshaper[channel]=poly[channel][0]*gshaper[channel]*dmag + poly[channel][1]
    *gshaper[channel]+poly[channel][2]*dmag +poly[channel][3];
        dref[channel]=gshaper[channel]*dmag*2;
        if (ncurrent < delayh[channel])
            dref[channel]= dref[channel]*(A1h[channel]-0.5);
        else if (ncurrent < (2*delayh[channel]))
            dref[channel]= dref[channel]*(A1h[channel]+A2h[channel]-0.5);
        else if (ncurrent < nsamples)
            dref[channel]=0.5*dref[channel];
        else if (ncurrent < (delayl[channel] + nsamples))
            dref[channel]= dref[channel]*(A3l[channel]+A2l[channel]-0.5);
        else if (ncurrent < (2*delayl[channel]+ nsamples))
            dref[channel]= dref[channel]*(A3l[channel]-0.5);
        else
            dref[channel]= -0.5*dref[channel];
        }//endif
/*---------- store data ----------
```

Objective: filter and store data.

Inputs:

data1 – Y from A/D converter

```
data2 - reference generated from computer
grab1 - control signal from "pi2.vi"
DATA_SIZE - #data stored
NSTORE - #data averaged
-----------------------------------------*/
    void StoreData(void) {
        static long ndata=1,ndummy=0,sumf[2]={0,0};
        if ((*grab1) == 1 ) {
            if (ndata <=DATA_SIZE/NSTORE ) {
                if (ndummy < NSTORE) {
                    sumf[0] = sumf[0] + (*output1[0]);
                    sumf[1] = sumf[1] + (*output1[1]);
                    ndummy++;
                }
                if (ndummy == NSTORE) {
                    // Y filtered:
                    *(int_ptr[0]+ndata) =(long)((double)(sumf[0])/NSTORE);
                    *(int_ptr[1]+ndata) =(long)((double)(sumf[1])/NSTORE);
                    ndata++;
                    ndummy=1;
                    sumf[0]=(*output1[0]);
                    sumf[1]=(*output1[1]);
                }
            }
            else
                *check1= (int)(1);
            }//endif
```

```
            else {
                *check1 =(int)(0);
                ndata=0;
            }
        }
/*————————— PI controller ————————————————————
Objective: implement PI-control
Inputs:
refc – reference signal
yc – system output or DSP input from A/D converter
(filtered – *output1[2]; without filtered – input[2])
channel – the channel from which yc come from
Fs/NSAMPLES– sampling rate used in computing I-control
Output:
errc – control output
————————————————————————————————————————*/
        double PIcontrol(double refc,long yc, int channel) {
            double dyc, errc;
            static double sumc[2]={0,0};
            //reset integral term:
            if ( (*ki[channel])==0 ) sumc[channel]=0;
            dyc =(double)(yc)*6.10351e-004;
            errc = refc-dyc;
            sumc[channel] = sumc[channel] +
            (double)(*ki[channel])/SCALING/(Fs/NSAMPLES)*errc;
            if((sumc[channel]>=4.9976) && (errc>0))
                sumc[channel] = 4.9976;
```

```
    if((sumc[channel]<=-4.9976) && (errc<0))
        sumc[channel] = -4.9976;
    errc = errc*(*kp[channel])/SCALING + sumc[channel];
    return errc;
}
/*————————— controller (shaper + filter PI) ————————————
Objective: generate controller signals
Inputs:
    square_c – current sampling time
    totaln– total sampling number in half square period
    input– system output or A/D input signals
    dref– shaping reference
Outputs:
    *output1[2] – data to be stored;
    *cont[2] – control signals
Notes:
    ki=kp=0, input shaping + average steady state signal.
    open-loop control: PI control-> "dcon[2]" = constant,
    shaper "dref[2]" = various
    closed-loop control: PI control-> "dcon[2]" = various,
    shaper "dref[2]" = constant
————————————————————————————————————————*/
void Control2(long square_c,long totaln, int input, int channel) {
    static long pi_c[2]={0,0}, sumi[2]={0,0};
    static int dcont[2]={0,0};
    if ( (square_c < 2*delayh[channel]) || (square_c >= totaln && square_c <
totaln+2*delayl[channel]))
```

```
            *output1[channel]=(long)input;
        else if ((square_c == 2*delayh[channel]) ||
        ( square_c == totaln+2*delayl[channel])) {
            *output1[channel]=(long)input;
            pi_c[channel]= sumi[channel]=0;
        }
        else {
            if ( pi_c[channel] < NSAMPLES ) {
                sumi[channel]=sumi[channel]+input;
                pi_c[channel]++;
            }
            if ( pi_c[channel] == NSAMPLES ) {
                *output1[channel]=(long)(sumi[channel]/NSAMPLES);
                dcont[channel]= (int)(PIcontrol(dref[channel]/gshaper[channel],
*output1[channel],channel)*409.5966);
                sumi[channel]= input;
                pi_c[channel]=1;
            }
        }
        *cont[channel]= (int)(dref[channel]*409.5966)+dcont[channel];
    }
    /*———————- Main() function ———————————-
    void main() {
        int input[2];
        long i, *int_time, counter, *freq;
        InitDsp();
        // ——— set memory address ———————————-
```

```
//for store data
check1 = (int *) CHECK1;        //0x12F6
* check1 = (int) (0);          //1: data in mem. ready
int_time = (long *) TIMER;      //0x12F7
*int_time = (long ) (SCALING*400/Fs*NSTORE);
grab1 = (int *) GRAB1;          //0x12F8
* grab1 = (int) (0);           //1=store data; 0=reset
int_ptr[0] = (long *) MEMORY1_START;
int_ptr[1] = (long *) MEMORY2_START;
//for main()
mag[1] = (long *) MAGY;         //0x12FC
* mag[1] = (int) (0*SCALING);
cont[1] = (int *) CONTY;        //0x12FD
output1[1] = (long *) OUTPUTY;      //0x12FE
kp[1] = (long *) KPY;           //0x12ff
* kp[1] = (int) (0*SCALING);
ki[1] = (long *) KIY;           //0x1300
* ki[1] = (int) (0*SCALING);
freq = (long *) FREQX;          //0x12F5
* freq = (int) (1.0*FSCALING);
mag[0] = (long *) MAGX;         //0x1384
* mag[0] = (int) (0*SCALING);
cont[0] = (int *) CONTX;        //0x1382
output1[0] = (long *) OUTPUTX;      //0x1383
kp[0] = (long *) KPX;           //0x1386
* kp[0] = (int) (0*SCALING);
ki[0] = (long *) KIX;           //0x1387
```

```
* ki[0] = (int) (0*SCALING);
//------- set shaper parameters ----------
for (i=0;i<=1;i++) {
    A1h[i]= 1/(1+2*Mph[i]+Mph[i]*Mph[i]);
    A2h[i] = 2*Mph[i]/(1+2*Mph[i]+Mph[i]*Mph[i]);
    A3h[i]=1-A1h[i]-A2h[i];
    A1l[i]= 1/(1+2*Mpl[i]+Mpl[i]*Mpl[i]);
    A2l[i] = 2*Mpl[i]/(1+2*Mpl[i]+Mpl[i]*Mpl[i]);
    A3l[i]=1-A1l[i]-A2l[i];
    delayh[i] = (int)(tdelayh[i]*Fs);
    delayl[i] = (int)(tdelayl[i]*Fs);
}
//-------------------------------------
i=0;
WriteDACs(0,0);
counter = (long)(Fs*FSCALING/2/(*freq));      //half period
while(1) {
    input[0] = ReadAdc(0);
    counter = (long)(Fs*FSCALING/2/(*freq));      //half period
    if (i>=2*counter) i=0;
    Shaperf(counter,i,0);      //output= double dref[0],gshaper[0]
    /*---------- 1. Open-loop control ----------
    tdelay =0 => square input (mag=gshaper*(*mag))
    tdelay!=0 => shaper input (mag=gshaper*(*mag))
    NSTORE =1 => without average. filter
    ----------------------------------*/
    /*
```

```
*cont[0] = (int)(dref[0]*2047/4.9976);
*output1[0] = (long)input[0];
input[1] = ReadAdc(1);
Shaperf(counter,i,1);       //output= double dref[1],gshaper[1]
*cont[1] = (int)(dref[1]*2047/4.9976);
WriteDACs(*cont[0],*cont[1]);
*output1[1] = (long)input[1];
*/
/*————————— 2. open loop + average —————————
kp=ki=0:
tdelay =0 => square(mag=(*mag))+average
tdelay!=0 => square(mag=(*mag))+average when t>2*tdelay
————————————————————————————————————*/
/*
dref[0] =(double)(*mag[0])/SCALING;       //square wave – start
if (i >= counter)      dref[0] = - dref[0];       //square wave – end
//average: output=*cont[0],*output1[0]
Control2(i,counter,input[0],0);
input[1] = ReadAdc(1);
dref[1] =(double)(*mag[1])/SCALING;
if (i >= counter)      dref[1] = - dref[1];
Control2(i,counter,input[1],1);       //output=*cont[1], *output1[1]
WriteDACs(*cont[0],*cont[1]);
*/
/*————————— 3. closed-loop control —————————
ki=kp=0 => shaper + average
ki=kp=0; tdelay=0 => square(mag=gshaper*(*mag)) + average
```

```
            ki=kp!=0; tdelay=0 => square +pi+ average
            ki=kp!=0; tdelay!=0 => shaper +pi+ average
            ------------------------------------------*/
            Control2(i,counter,input[0],0);      //output=*cont[0], *output1[0]
            input[1] = ReadAdc(1);
            Shaperf(counter,i,1);      //output= double dref[1],gshaper[1]
            Control2(i,counter,input[1],1);      //output=*cont[1], *output1[1]
            WriteDACs(*cont[0],*cont[1]);
            StoreData();
            i++;
        } //endwhile
    } //end of file
```

E.2.2 Fine Motion Control

```
/*------------ fine motion control (pifine) ---------
Objective: implement fine motion control to study the system's resolution
Notes:
Check the frequency of the reference sinusoid, which will be far from the expected
value if the sampling rate is too big to be implemented
Fs=5000Hz,  Fs_pi=5000/NSTORE=5000/100=50Hz
#data_stored = 2*NMAG*NSAMPLES=10000 (per signal);
total time=10000/50=200sec
------------------------------------------*/
    #define TIMPER0 0x271
    #include <math.h>
    #include"d310biox.H"
```

```
//--- mem. distribution ---
#define FREQX 0x12F5
#define CHECK1 0x12F6
#define TIMER 0x12F7
#define GRAB1 0x12F8
#define MAGY 0x12FC
#define CONTY 0x12FD
#define OUTPUTY 0x12FE
#define KPY 0x12FF
#define KIY 0x1300
#define REFY 0x1303
#define REFX 0x1302
#define CONTX 0x1382
#define OUTPUTX 0x1383
#define MAGX 0x1384
#define KPX 0x1386
#define KIX 0x1387
#define MEMORY1_START 0x1388
#define MEMORY2_START 0x61A8
#define DATA_SIZE 10000
#define PI 3.14159265358979
#define SCALING 500000.
#define NMAG 10
#define NSAMPLES 500
#define NSTORE 100
int * check1, *grab1,*cont[2];
long *output[2], * int_ptr[2], *kp[2], *ki[2];
```

```
    double Fs= (double)50000000.0/(TIMPER0*8*2),refs[2];
/*------- Magref function ----------

Objective: generate stair triangle signal

Inputs:

NMAG - # amplitude level

mref1, mref2 - maximum magnitude of the signals in x and y axes

i - current amplitude loop => generate the current amplitude

check1 - reset D/A signal when finish

Output:

refs[2] - ref. in x, y axis

-----------------------------------*/
    void Magref(int i,long int mref1,long int mref2) {
        double index1;
        if (i<=NMAG) index1=(double) i;
        else
            index1=(double)(NMAG*2-i);
        refs[0] =(double)(mref1)/SCALING*(index1/NMAG)*(1-*check1);
        refs[1] =(double)(mref2)/SCALING*(index1/NMAG)*(1-*check1);
    }
/*-------- StoreData function ----------

Objective: store data

Inputs:

y1,y2 - Y from A/D converter

grab1 - 1: acquire data; 0: reset counter

DATA_SIZE - #data stored

-----------------------------------*/
    void StoreData(long y1,long y2) {
```

```
        static long ndata=1;
        if ((*grab1) == 1 ) {
            if (ndata <=DATA_SIZE) {
                *(int_ptr[0]+ndata) = y1;
                *(int_ptr[1]+ndata) = y2;
                ndata++;
            }
            else
                *check1= (int)(1);
            }//endif
        else {
            *check1 =(int)(0);
            ndata=0;
        }
    }
/*————————- PIcontrol function ————————————

Objective: implement PI-control
Inputs:
refs – reference signal
yc – system response or DSP input
(filtered – *output[2]; without filtered – input[2])
channel – channel of A/D converter
Fs/NSTORE – sampling freq. used to compute I-control
Output:
control – output of PI-controller
————————————————————————————————————*/
    int PIcontrol(long yc, int channel) {
```

```
        double errc;
        static double sumc[2]={0,0};
        int control;
        errc = refs[channel]- 4.99939/8191.0*yc;
        if ( (*ki[channel])==0 ) sumc[channel]=0;      //reset integral term
        sumc[channel] = sumc[channel] + (double)(*ki[channel]) / SCALING /
Fs * NSTORE * errc;
        if((sumc[channel]>=4.9976) && (errc>0))
            sumc[channel] = 4.9976;
        if((sumc[channel]<=-4.9976) && (errc<0))
            sumc[channel] = -4.9976;
        control = (int)(2047./4.9976 * (errc * (*kp[channel]) / SCALING +
sumc[channel]));
        return control;
    }
/*------------ Control2 function ------------
Objectives: implement PI-control with down-sampling
Inputs:
input - system response or A/D input signals
Channel - channel of A/D converter
outputs:
output1[2] - filtered system response
cont[2] - control signals
-------------------------------------------*/
    void Control2(int input, int channel) {
        static long pi_c[2]={0,0}, sumi[2]={0,0};
        if ( pi_c[channel] < NSTORE ) {
```

```
            sumi[channel]=sumi[channel]+input;
            pi_c[channel]++;
        }
        if ( pi_c[channel] == NSTORE ) {
            *output[channel]=(long)(sumi[channel]/NSTORE);
            *cont[channel]=PIcontrol(*output[channel],channel);
            sumi[channel]= input;
            pi_c[channel]=1;
            if(channel == 1)
                StoreData(*output[0],*output[1]);
        }
}
/*————————— Main() function ————————-*/
void main() {
    int i, k, *freqx, input[2];
    long int *mag[2], *int_time, *ref[2];
    InitDsp();
    //———— set mem. address ————————-
    //for store data
    check1 = (int *) CHECK1;      //0x12F6
    * check1 = (int) (0);      //1: data in mem. ready
    grab1 = (int *) GRAB1;      //0x12F8
    * grab1 = (int) (0);      //1 store data
    int_ptr[0] = (long *) MEMORY1_START;      //0x1388 store y1
    int_ptr[1] = (long *) MEMORY2_START;      //0x2710 store ref1
    //for main()
    freqx = (int *) FREQX;      //0x12F5
```

```
* freqx = (int) (0);        //0: reset
int_time = (long *) TIMER;        //0x12F7
*int_time = (long) (SCALING*400/Fs*NSTORE);        //Ts<10
cont[0] = (int *) CONTX;        //0x1382 x-controller
*cont[0] = 0;
output[0] = (long *) OUTPUTX;        //0x1383 y_response
mag[0]    = (long int *) MAGX;        //0x1384 p-p ref.
* mag[0] = (int) (0.0*SCALING);
ref[0] = (long *) REFX;        //0x1302 refx
kp[0] = (long int *) KPX;        //0x1386
* kp[0] = (int) (0.0*SCALING);
ki[0] = (long int *) KIX;        //0x1387
* ki[0] = (int) (0.0*SCALING);
cont[1] = (int *) CONTY;        //0x12FD y-controller
*cont[1] = 0;
output[1] = (long *) OUTPUTY;        //0x12fe ReadAdc
mag[1] = (long int *) MAGY;        //0x12fc p-p ref.
* mag[1] = (int) (0.0*SCALING);
ref[1] = (long *) REFY;        //0x1303 refx
kp[1] = (long int *) KPY;        //0x12ff
* kp[1] = (int) (0.0*SCALING);
ki[1] = (long int *) KIY;        //0x1300
* ki[1] = (int) (0.0*SCALING);
WriteDACs(0,0);
while(1) {
    for (i=1;i<=NMAG*2;i++){
//———— test magnitude of ref sinusoid————
```

```
            // i=NMAG, mag. of ref. Sinusoid = max
            //i=NMAG;
            //if (i==NMAG){
        //-------------------------------------
                Magref(i,*mag[0],*mag[1]);      //double refs[2]
                *ref[0]=(long)(refs[0]*2047/4.9976);
                *ref[1]=(long)(refs[1]*2047/4.9976);
                for (k=0;k<NSAMPLES*NSTORE;k++) {
                    input[0] =(long) ReadAdc(0);
                    Control2(input[0],0);       //output=*cont[0],*output[0]
                    if((*freqx)==0){
                        k=NSAMPLES;
                        i=0;
                    }
                    input[1] =(long) ReadAdc(1);
                    Control2(input[1],1);       //output=*cont[1],*output[1]
                    WriteDACs(*cont[0],*cont[1]);
                    //test ref. signal
                    //WriteDACs((short)(*ref[0]),(short)(*ref[1]));
                    //test pi control
                    //WriteDACs((short)(*ref[0]),*cont[0]);
                } //endfor
            } //endfor
        } //endwhile
} // end of file
/*--------- PI control with sinusoid ref. (pi2sine.c) ---------
#define TIMPER0 0x30D4
```

```
#include <math.h>
#include"d310biox.H"
//—- mem. distribution —
#define FREQX 0x12F5
#define CHECK1 0x12F6
#define TIMER 0x12F7
#define GRAB1 0x12F8
#define FREQY 0x12FB
#define MAGY 0x12FC
#define CONTY 0x12FD
#define OUTPUTY 0x12FE
#define KPY 0x12FF
#define KIY 0x1300
#define REFY 0x1303
#define REFX 0x1302
#define CONTX 0x1382
#define OUTPUTX 0x1383
#define MAGX 0x1384
#define KPX 0x1386
#define KIX 0x1387
#define MEMORY1_START 0x1388
#define MEMORY2_START 0x3A98
#define MEMORY3_START 0x61A8
#define MEMORY4_START 0x88B8
#define DATA_SIZE 5000
#define PI 3.14159265358979
#define SCALING 500000.
```

```
#define FSCALING 5000
int * check1, *grab1;
double refs[2];
long * int_ptr[4];
double Fs= (double)50000000.0/(TIMPER0*8*2);
/*———— StoreData function ————————
Objective: store data
inputs:
y1,y2 - Y from A/D converter
grab1 - control signal from "pi2.vi"
DATA_SIZE - #data stored
————————————————————————————————*/
void StoreData(long y1,long y2) {
    static long int data_count=0;
    if ((*grab1) == 1 ) {
        if (data_count < 2*DATA_SIZE ) {
            *(int_ptr[0]+data_count) = y1;
            *(int_ptr[1]+data_count) = (long)(refs[0]*8191.0/4.99939);
            *(int_ptr[2]+data_count) = y2;
            *(int_ptr[3]+data_count) = (long)(refs[1]*8191.0/4.99939);
            data_count++;
        }
        else
            *check1= (int)(1);
        }//endif
    else {
        *check1 =(int)(0);
```

```
            data_count=0;
        }//endelse
    }
/*————————— PIcontrol function ————————————
Objective: implement PI-control
Inputs:
refc – reference signal
yc – system output or DSP input
(filtered – *output[2];      without filtered – input[2])
channel – from which yc comes
Fs – sampling rate
kic, kpc – coefficients of PI controller
Output: control
————————————————————————————————————*/
    int Control(double refc,long yc, long kic,long kpc,int channel) {
        double errc;
        static double sumc[2]={0,0};
        int control;
        if ( kic==0 ) sumc[channel]=0; //reset integral term
        errc = refc- 4.99939/8191.0*yc;
        sumc[channel] = sumc[channel] + (double)kic/SCALING/Fs*errc;
        if((sumc[channel]>=4.9976) && (errc>0))
            sumc[channel] = 4.9976;
        if((sumc[channel]<=-4.9976) && (errc<0))
            sumc[channel] = -4.9976;
        control = (int)(2047./4.9976*(errc*kpc/SCALING + sumc[channel]));
        return control;
```

```
}
/*————————— Main() function ————————————-*/
void main() {
    int n, m, *cont[2];
    long *mag[2], *freq[2], *int_time, *kp[2], *ki[2], count[2],
    *output[2], *ref[2];
    InitDsp();
    //for store data
    check1 = (int *) CHECK1;       //0x12F6
    * check1 = (int) (0);       //1: data in mem. ready
    grab1 = (int *) GRAB1;       //0x12F8
    * grab1 = (int) (0);       //1 store data
    int_ptr[0] = (long *) MEMORY1_START;       //0x1388 store y in x-axis
    int_ptr[1] = (long *) MEMORY2_START;       //0x3A98
    int_ptr[2] = (long *) MEMORY3_START;       //0x61A8 store y in y-axis
    int_ptr[3] = (long *) MEMORY4_START;       //0x88B8
    //for main()
    int_time = (long *) TIMER;       //0x12F7
    *int_time = (long) (SCALING*400/Fs);       //Ts<10
    // x_direction
    freq[0] = (long *) FREQX;       //0x12F5
    * freq[0] = (int) (1.0*FSCALING);       //1: data in mem. ready
    cont[0] = (int *) CONTX;       //0x1382 x-controller
    *cont[0] = 0;
    output[0] = (long *) OUTPUTX;       //0x1383 y_response
    mag[0] = (long *) MAGX;       //0x1384 peak-to-peak of ref.
    * mag[0] = (int) (0*SCALING);
```

```
ref[0] = (long*) REFX;        //0x1302 refx
kp[0] = (long int *) KPX;     //0x1386
* kp[0] = (int) (0.001*SCALING);
ki[0] = (long int *) KIX;     //0x1387
* ki[0] = (int) (0.0*SCALING);
// y_direction
freq[1] = (long int *) FREQY;      //0x12FB
* freq[1] = (int) (1.0*FSCALING);      //1: data in mem. ready
cont[1] = (int *) CONTY;      //0x12FD y_controller
*cont[1] = 0;
output[1] = (long *) OUTPUTY;      //0x12fe ReadAdc
mag[1] = (long int *) MAGY;      //0x12fc p-p ref.
* mag[1] = (int) (0*SCALING);
ref[1] = (long *) REFY;      //0x1303 refx
kp[1] = (long int *) KPY;      //0x12ff
* kp[1] = (int) (0.001*SCALING);
ki[1] = (long int *) KIY;      //0x1300
* ki[1] = (int) (0.0*SCALING);
count[0] = (long)(Fs*FSCALING/(*freq[0]));
count[1] = (long)(Fs*FSCALING/(*freq[1]));
WriteDACs(0,0);
while(1) {
    *output[0] = (long)ReadAdc(0);
    *output[1] = (long)ReadAdc(1);
    count[0]=(long int)(Fs*FSCALING/(*freq[0]));
    count[1]=(long int)(Fs*FSCALING/(*freq[1]));
    if (n>=count[0]) n = 0;
```

```
            refs[0]=(double)(*mag[0])/SCALING*sin(2*PI*n/count[0]);
            n++;
            if (m>=count[1]) m = 0;
            refs[1]=(double)(*mag[1])/SCALING*sin(2*PI*m/count[1]);
            m++;
            *cont[0]=Control(refs[0],*output[0],*ki[0],*kp[0],0);
            *cont[1]=Control(refs[1],*output[1],*ki[1],*kp[1],1);
            WriteDACs(*cont[0],*cont[1]);
            *ref[0]=(long)(refs[0]*2047/4.9976);
            *ref[1]=(long)(refs[1]*2047/4.9976);
            //test ref. signal
            //WriteDACs((short)(*ref[0]),(short)(*ref[1]));
            //test pi control
            //WriteDACs((short)(*ref[0]),*cont[0]);
            StoreData(*output[0],*output[1]);
        } //endwhile
} //end of file
```

E.3 Labview Program

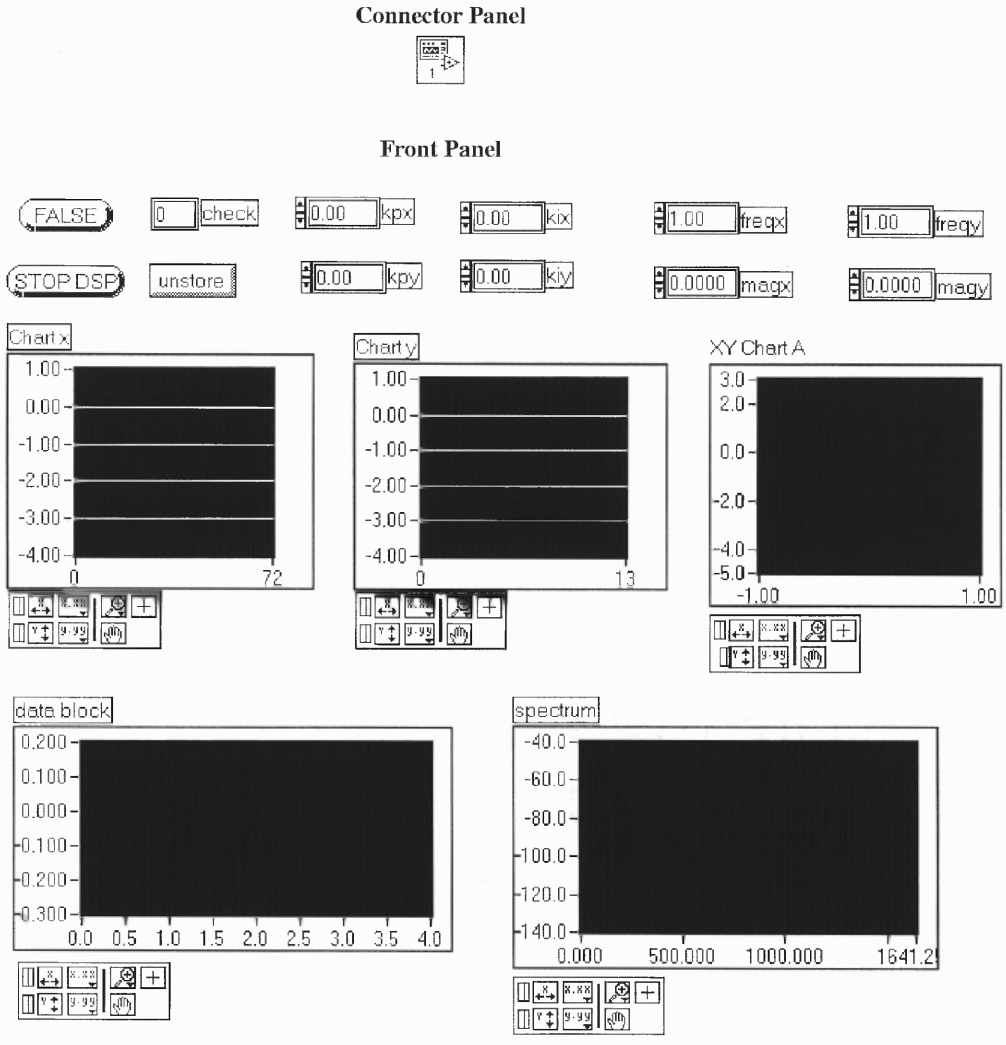

Figure E.1 Labview program for time division multi-control and fine motion control

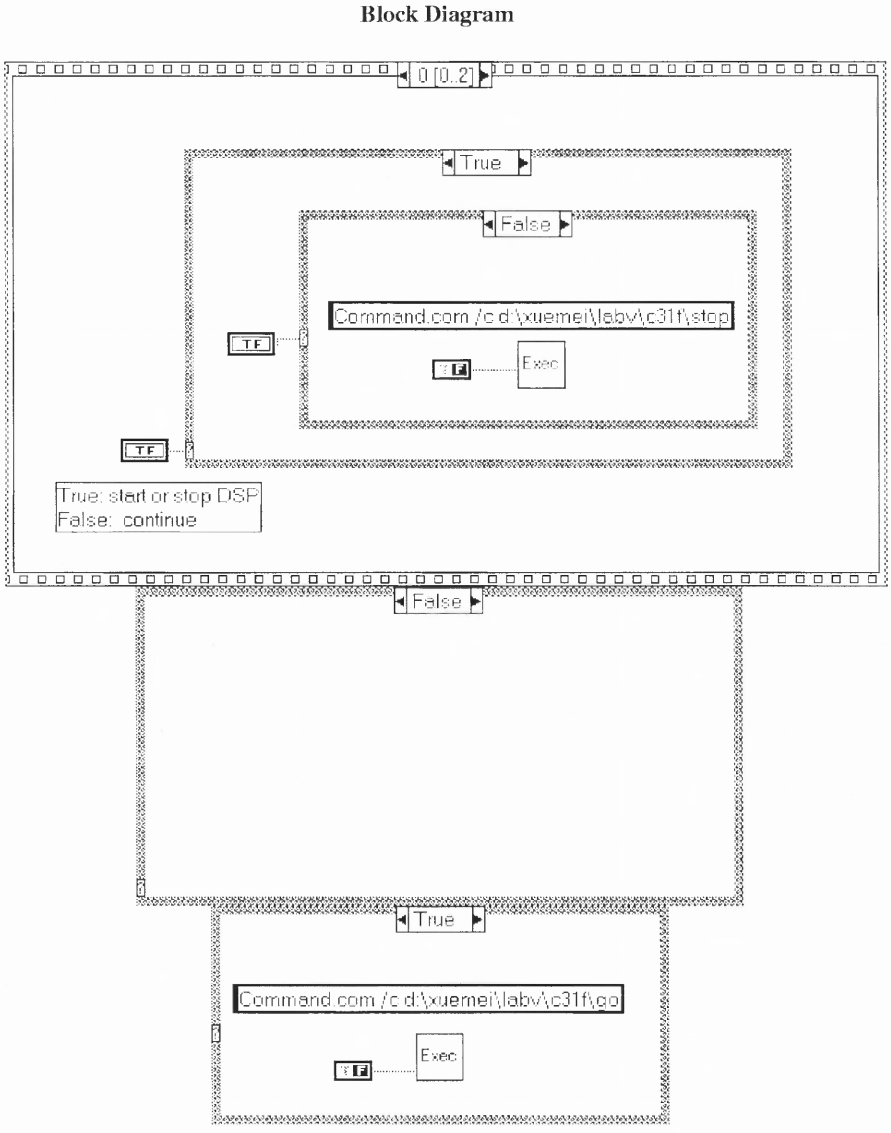

Figure E.2 Labview program for time division multi-control and fine motion control (continued)

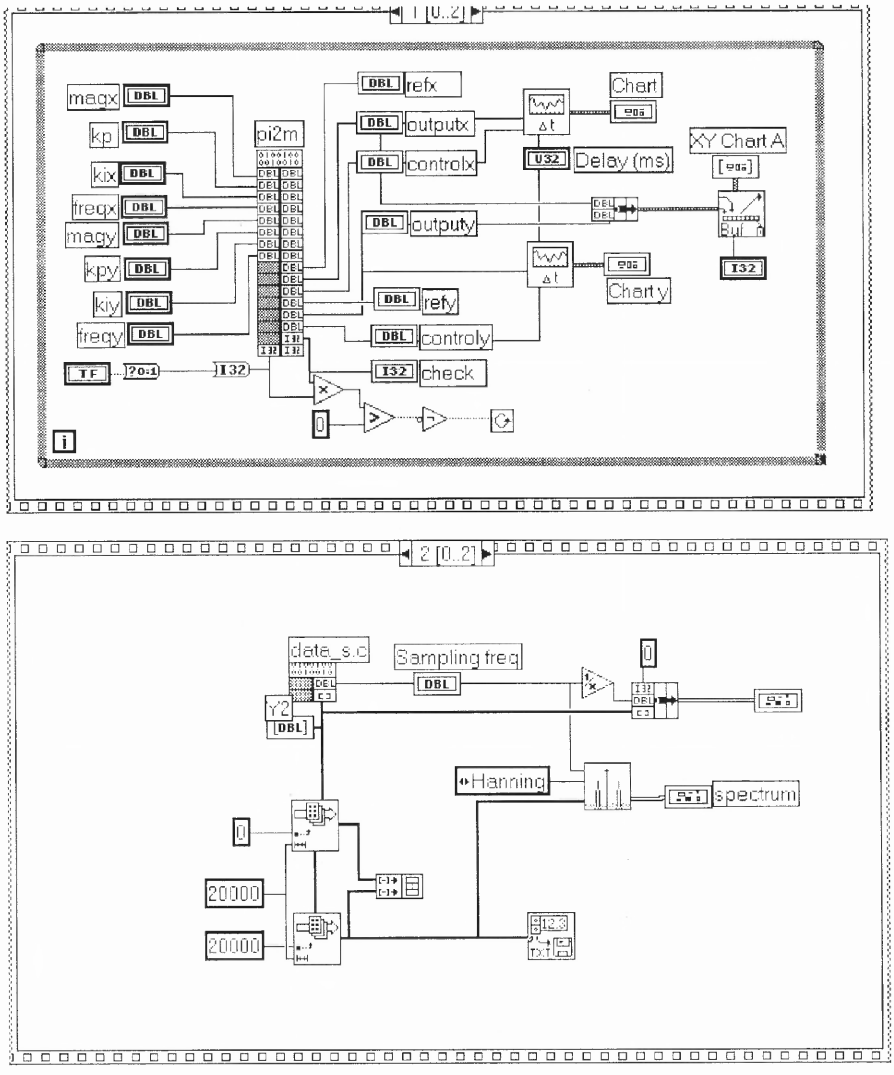

Figure E.3 Labview program for time division multi-control and fine motion control (continued)

APPENDIX F

ITERATION DERIVATION USING MATLAB

Matlab symbolic toolbox is applied to derive the hysteresis harmonics, as below.

```
% ———————————— Iteration derivation ————————————

% Objective: derive the analytic expression of hysteresis harmonics

% Model:ddot(x)+r*dot(x)+kn*x=A*sin(w*t)-k1*F

%  dot(F)=dot(x)-F/Fc* |dot(x)|;

% ————————————————————————————————————————————

    % 1st iteration:

    % Model: dot(x0)+r*dot(x0)+kn*x0=A*sin(w*t)-k1*F0

    %          F0=x0;

    % Results:  x0=(A*(-w^2+k1+kn)*sin(w*t)-A*r*w*cos(w*t))...

                /(kn^2+2*kn*k1-2*kn* w^2+k1^2- 2*k1* w^2+w^2 * r^2+w^4)

    syms r kn w k1 A s t

    X0=laplace(A*sin(w*t),t,s)/(s^2+r*s+kn+k1);

    x0=ilaplace(X0,s,t)

    % 2nd iteration results

    % Model:  ddot(x1)+r*dot(x1)+kn*x1=A*sin(w*t)-k1*F1

    %          dF1=dx0-F0/Fc * | dx0 |;

    % Results:

    %  x1 = x11*cos(w*t)+x12*sin(w*t)+x13*cos(3*w*t)+x14*sin(3*w*t)

    %  x11= -1/3*(3*A*w*r*Fc*pi-4*k1*A0^2* w^2*cos(phi)...

    %          -3*k1*A0*r*Fc*pi*cos(phi)*w...

    %          +3*k1*A0*Fc*pi*sin(phi)*kn+4*k1*A0^2*r*sin(phi)*w...

    %          -3*k1*A0*w^2*Fc*pi*sin(phi)+4*k1*A0^2*cos(phi)*kn)...

    %          *cos(w*t)/Fc/pi/(w^2*r^2+kn^2-2*kn*w^2+w^4)
```

```
%  x12=  1/3 * (-3 * A * Fc * pi * w^2+3 * A * Fc * pi * kn ...
%        -3*k1*A0*r*Fc*pi*sin(phi)*w...
%        -3*k1*A0*Fc*pi*cos(phi)*kn-4*k1*A0^2*w^2*sin(phi)...
%        +3*k1*A0*w^2*Fc*pi*cos(phi)+4*k1*A0^2*sin(phi)*kn...
%        -4*k1*A0^2*r*cos(phi)*w)...
%        *sin(w*t)/Fc/pi/(w^2*r^2+kn^2-2*kn*w^2+w^4)
%  x13=  -2/9*k1*A0^2*(3*w*sin(3*phi)*r...
%        -9*w^2*cos(3*phi)+cos(3*phi)*kn)...
%        *cos(3*w*t)/Fc/pi/(9*w^2*r^2+81*w^4-18*kn*w^2+kn^2)
%  x14=  -2/9*k1*A0^2*(3*w*cos(3*phi)*r...
%        -sin(3*phi)*kn+9*w^2*sin(3*phi))...
%        *sin(3*w*t)/Fc/pi/(9*w^2*r^2+81*w^4-18*kn*w^2+kn^2)
%  F1=  (A0*cos(phi)-4/3*A0^2/Fc/pi*sin(phi))*sin(w*t)...
%        +(A0*sin(phi)+4/3*A0^2/Fc/pi*cos(phi))*cos(w*t)...
%        +2/9*A0^2/Fc/pi*cos(3*phi)*cos(3*w*t)
%        -2/9*A0^2/Fc/pi*sin(3*phi)*sin(3*w*t)
syms r kn k1 Fc w phi n A0 A t s
T=2*pi/w;  x0=A0*sin(w*t+phi); F0=x0;  %1st iteration results
dx0=diff(x0,t);  %dx0=A0*cos(w*t+phi)*w=A0*w*sin(w*t+phi+pi/2)
%  for f(t)=h*sin(w*t+phi1),
%  abs_f(t)=1/3*(6*h-4*h*cos(2*phi1+2*w*t))/pi, seen"Fourier1.m"
h = A0*w;  phi1 = phi+pi/2;
dF1 = diff(x0,t)-F0/Fc*(2*h/pi-4*h/3/pi*cos(2*w*t+2*phi1));
F1 = ilaplace(laplace(dF1,t,s)/s,s,t)
X1 = laplace(A*sin(w*t),t,s)/(s^2+r*s+kn)- k1*laplace(dF1,t,s)/s/(s^2+r*s+kn);
x1 = ilaplace(X1,s,t);
% end of file
```

REFERENCES

1. L. Fulton, "Two-axis motion apparatus utilizing piezoelectric material," *U. S. Patent 5170089*, 1992.

2. M. Toda, "Piezoelectric position device," *U.S. Patent 4678955*, 1982.

3. T. Chang, "Piezoelectric multiple degree of freedom actuator," *U.S Patent pending*, 1998.

4. Patrick M. Sain, Michael K. Sain and B. F. Spencer, "Models for hysteresis and application to structural control," *Proceedings of the American Control Conference*, pp. 16-20, 1997.

5. Michael Goldfarb and Nikola Celanovic, "Modeling piezoelectric stack actuators for control of micromanipulation," *IEEE Control Systems Magazine*, vol. 17, n. 3, pp. 69-79, June 1997.

6. Michael Goldfarb and Nikola Celanovic, "Behavioral implications of piezoelectric stack actuators for control of micromanipulation," *Proceedings of the 1996 IEEE International Conference on Robotics and Automation Minneapolis*, Minnesota, April 1996.

7. Gi Sang Choi, Hie-Sik Kim and Gi Heung Choi, "A study on position control of piezoelectric actuators," *ISIE'97.*

8. Yi-Kwei Wen and M. Asce, "Method for random vibration of hyterestic systems," *Journal of the Engineering Mechanics Division*, vol. 102, pp. 249-263, 1976.

9. Musa Jouaneh and Huawei Tian, "Accuracy enhancement of a piezoelectric actuator with hysteresis," *Japan/USA Symposium on Flexible Automation*, vol. 1, ASME 1992.

10. Mayergoyz, *Mathematical Models of Hysteresis*, New York: Springer-Verlag, 1991.

11. T. Doong and I. Mayergoyz, "On numerical implementation for hysteresis models," *IEEE Trans. Magnetics*, vol. 21, pp. 1853-1855, 1985.

12. Ping Ge and Musa Jouaneh, "Modeling hysteresis in piezoceramic actuators," *Precision Engineering*, vol. 17, No. 3, July 1995.

13. Ping Ge and Musa Jouaneh, "Generalized preisach model for hysteresis nonlinearity of piezoceramic actuators," *Precision Engineering*, vol. 20, No. 2, March 1997.

14. Ping Ge and Musa Jouaneh, "Tracking control of a piezoceramic actuators," *IEEE Transactions on Control Systems Technology*, vol. 4, No. 3, May 1996.

15. P. R. Dahl, "Solid fraction damping of mechanical vibrations," *AIAA*, vol. 14, Dec. 1976.

16. P. R. Dahl, "Math model of hysteresis in piezo-electric Actuators for precision pointing systems," *EL Segundo*, California, Feb. 1985.

17. Tian Hong, *Thesis: Control of Smart Structure Using Adaptive Dither*, Oct. 1994.

18. Tian Hong and Timothy N. Chang, "Control of nonlinear piezoelectric stack using adaptive dither," *Proceedings of the 1995 American Control Conference*, Seattle, WA., pp. 76-80.

19. *Operator's Manual – Model 601B-PCB High-Voltage Amplifier*, Trek Incorporated.

20. *ADE 380 OEM Gaging System – User's Guide, ADE P/N 024418-01*, 3800 Preliminary User's Guide.

21. *Model 310 Data Acquisition and Signal Processing Board for the IBM PC and ISA Bus compatible*, Dalanco Spry

22. *Labview User Manual*, National Instruments, January 1998, part number 320999B-01.

23. *Labview Code Interface Reference Manual*, National Instruments Corporation, January 1996, part number 320539c-01.

24. Robert W. Newcomb, *Nonlinear System Analysis*, Prentice-Hall Electrical Engineering Series.

25. Timothy N. Chang, Xuemei Sun, Zhiming Ji, Reggie Caudill, "Analysis and control of monolithic piezoelectric nano-actuator," *Proceeding of the 2000 ACC*, Chicago, IL, pp.3086-3090.

26. E. Kouno, "A fast response piezoelectric actuator for servo correction of systematic errors in precision machining," *Annals of the CIRP*, vol. 33, pp. 369-372, Jan. 1984.

27. E. Crawely and E. Anderson, "Detailed models of piezoceramic actuation of beams," *Proceedings of the 30th Structures, Structural Dynamics and Materials Conference, AIAA*, pp. 2000-2010, April 1989.

28. Hayesh Amin, Bernard Friedland and Avraham Harnoy, "Implementation of a friction estimation and compensation technique," *IEEE Control Systems*, pp. 71-76, August 1997.

29. Bernard Friedland and Young-kin Park, "On adaptive friction compensation," *Proceedings of the 30th Conference on Decision and Control*, pp. 2899-2902, December 1991.

30. Bernard Friedland and Sophia Mentzelopoulou, "On adaptive friction compensation without velocity measurement," *1992 IEEE*, pp. 1076-1081.

31. N. Singer and W. Seering, "Pre-shaping command inputs to reduce system vibration," *ASME J. Dyn. Sys., Meas., and Contr.*, 112(1), 1990.

32. T. Singh and S. R. Vadali, "Robust time-optimal control: a frequency domain approach," *AIAA J. Guid., Contr., and Dyn.*, 17(2), 1990.

33. W. Singhose, W. Seering, and N. Singer, "Residual vibration reduction using vector diagrams to generate shaped inputs," *ASME J. Mech. Design*, 116(2),1994.

34. B. Wie, R. Sinha, and Q. Liu, "Robust time-optimal control of uncertain structural dynamic systems," *AIAA J. Guid., Contr., and Dyn.*, 16(5), 1993.

35. L. Pao, T. Chang, and E. Hou, "Input shaper design for minimizing the expected level of residual vibration in flexible systems," *Proc. American Control Conference*, pp. 3542-3546, 1997.

36. P. Chen and S. Montgomery, "A macroscopic theory for the existence of the hysteresis and butterfly loops in ferroelectricity," *Ferroelectr.*, vol. 23, pp. 199-207, 1980.

37. C. Canudas de Wit, H. Olsson, K. J. Astrom and P. Lischinsky, "A new model for control of systems with friction," *IEEE Transactions on Automatic Control*, vol. 40, No. 3, pp. 419-425, March 1995.

38. Timothy Chang, X. Sun, Z. Ji, R. Caudill, "High precision multiple degree of freedom piezoelectric manipulator," *Proceedings to the 1998 SPIE International Symposium on Intelligent Systems and Advanced Manufacturing*, Nov. 1-5, 1998.