Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

VIBRATION CONTROL OF ULTRA-HIGH PRECISION MAGNETIC LEADSCREW USING RECURRENT NEURAL NETWORK

by
Bhaskar Vinayak Dani

Ultra-high precision positioning is of strategic importance to modern industrial processes such as semiconductor manufacturing. Traditional drives with mechanical transmission elements exhibit nonlinearities such as friction, backlash and hysteresis which limit the system performance significantly. The magnetic leadscrew in this work belongs to the class of contactless drives which overcome the above mentioned limitations of contact-type drives. The operation is based on leadscrew/nut coupling but unlike mechanical leadscrews, the threads of the nut and the leadscrew are aligned magnetically and do not come in contact. Thus, "hard" nonlinearities are substantially reduced resulting in high precision and high resolution.

The dynamics of the system are, however, lightly damped and result in vibration of the nut upto tens of microns peak-to-peak. Due to the high frequency of the modes, typically a few hundred Hz, the dynamics are difficult to control using conventional techniques, limited actuator bandwidth being one of the reasons. Active control must therefore be employed. This work develops a passband control scheme based on the Hilbert Transform which gives the orthogonal components of the oscillating modes. The components are extracted using a neural network to enhance the robustness of the controller.

Performance of the controller is evaluated under self-resonance, forced oscillation and transient response. Self-resonance is shown to be completely eliminated while for forced oscillation, the axial gain is shown to be reduced. Stabilization time of the transient response is also significantly reduced, thereby confirming the vibration suppression capabilities of the controller.

VIBRATION CONTROL OF ULTRA-HIGH PRECISION MAGNETIC LEADSCREW USING RECURRENT NEURAL NETWORK

by
Bhaskar Vinayak Dani

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

May 1999

APPROVAL PAGE

VIBRATION CONTROL OF ULTRA-HIGH PRECISION MAGNETIC LEADSCREW USING RECURRENT NEURAL NETWORK

Bhaskar Vinayak Dani

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

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

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

Dr. Reggie Caudill, Committee Member — Date
Professor of Industrial and Manufacturing Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Bhaskar Vinayak Dani

Degree: Master of Science in Electrical Engineering

Date: May 1999

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, USA, 1999
- Bachelor of Engineering in Electronics and Telecommunication,
  University of Pune, Pune, India, 1996

Major: Electrical Engineering

To my parents and sister

ACKNOWLEDGMENT

I would like to place on record my profound gratitude towards Prof. Timothy Chang for the patience, guidance, support and encouragement he extended to me throughout the course of this thesis. I also thank the committee members Prof. Andrew Meyer, Prof. Zhiming Ji and Prof. Reggie Caudill for reviewing the thesis and suggesting important changes in the written work.

I am indebted to my parents and sister who put me where I am. Many thanks are are also due to my friends Deven Panse, Ketan Limaye, Amit Amte and Rahul Dhavalikar for their support. The countless intellectual discussions and research problem solving sessions with fellow student Roger Kwadzogah made invaluable contributions to this work and I thank him for the same. Finally, I would like to thank the NIST ATP Grant number 70NANB5H1092 for Precision Optoelectronics Assembly, for extending financial support for this thesis.

TABLE OF CONTENTS

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Objective

This work addresses the issue of vibration control of an ultra-high precision positioning system. The system under consideration is a contactless drive where magnetic coupling is employed between a nut and a leadscrew to achieve a resolution of about 10 nanometers over a range of 10 cm. Due to the use of aerostatic bearing between the nut and the leadscrew, a number of resonances exist at the nut. It was determined that the vibration of the nut is of the order of tens of microns and therefore requires active control. A learning controller based on the recurrent neural network scheme is implemented. The neural network separates the required in-phase and quadrature components of the oscillating modes of the nut which are then individually regulated regeneratively or degeneratively. Performance of this controller in the presence of self-resonance and forced oscillation is evaluated.

1.2 Background Work

Contactless drives is the enabling technology for modern industrial processes such as microlithography, precision manufacturing, optoelectronic assembly, etc. These processes have to meet increasingly tighter positioning constraints such as end point positioning in the range of microns or nanometers. It is difficult to achieve this performance using conventional systems with mechanical bearings or belt drives. This is primarily because systems with mechanical transmission elements exhibit nonlinearities such as friction, backlash and hysteresis which are the primary factors degrading the system performance thus making it difficult for these drives to conform to the stringent requirements of end point confines. For example, it is projected that manufacturing the next generation of DRAMs with capacities of 256 MB or 1 GB will require a positioning reproducibility of 20-30 nanometers [2]. Motivation for research

in contactless drives stems from the above mentioned difficulties in employing conventional drives in precision applications.

In contactless drives, there is no physical contact between the transmitting parts. This virtually eliminates the nonlinearities such as friction, backlash, etc. resulting in very accurate positioning. Different technologies such as aerostatic and hydrostatic bearings have been investigated which use externally pressurized fluids for physical separation of the nut and the leadscrew [3]. Intense research is also being pursued in the field of magnetic suspension. Magnetic suspension drives may have two types: nut-leadscrew assemblies employing magnetic coupling and linear motors. In conventional contact-type drives, the threads of the nut and the leadscrew are mechanically meshed. However, in contactless drives, the threads are aligned magnetically. This gives them the desired contactless property. Note, however, that the drive's performance will be sensitive to the air gap between the nut and the leadscrew.

Linear motors are an alternate approach for implementing magnetic suspension drives. These are direct drive systems because the drive force is directly applied to the payload. Also, because of the magnetic suspension, there is no theoretical limit on the speed of these motors. Practically, the speed is limited by the bandwidth of sensors employed and that of the power electronics driving the motor [4]. All this results in a high speed-high accuracy system, ideal for precision positioning applications. Many types of linear motors are available with applications ranging from merry-go-round rides to semiconductor wafer stepping. Selection depends on the type of application. For precision applications, linear servo motors are the most suitable. Most of the current research in precision positioning using linear motors addresses the field of semiconductor manufacturing. In [2], an xy-stage is outlined which has a positioning reproducibility of 20 nanometers and uses three linear DC motors in an H-shaped arrangement. Dual servo (piggyback) technique is employed with nanometer range

positioning achieved by PZT actuators. Similarly, [5] demonstrates a six degree-of-freedom, permanent magnet levitated linear motor delivering r.m.s. positioning as low as 5 nanometers. Extensive research is being pursued in the field and linear motors with peak forces as high as 20 kN are available. These examples show that linear motors are also proving to be promising for precision positioning applications.

Thus, magnetic leadscrews and linear motors are the two competing technologies among magnetic suspension systems both taking advantage of their contactless property. Linear motor technology is, however, expensive. With their advantages in terms of performance, they also bring inherent undesired properties (e.g. force ripple) which make their control more difficult than rotary motors [4]. Special drive circuits are required for linear motors as against the standard rotary motor drives required for magnetic leadscrews. Increasing the span-length of linear motors is expensive since it requires additional magnetic tracks to be stacked adjacent to each other. Further, each magnetic track requires separate cooling arrangement which makes it even more expensive to achieve longer travels. Magnetic leadscrews do not suffer from these limitations of linear motor drives. As mentioned before, they can be driven with conventional rotary motor drive circuits. Increasing their span-length amounts only to replacing the leadscrew with another having greater length, which is a much more economical affair as compared to linear motors. Also, resolution of the drive can be improved by improving the thread ratio between the nut and the leadscrew. Typically, they can deliver forces of the order of 400 N and higher peak forces may be achieved. In summary, magnetic leadscrews offer a cheaper option for precision positioning applications delivering the desired properties of linear motors.

However, due to their contactless nature, magnetic leadscrews exhibit very low damping. Friction between the transmitting parts is virtually eliminated resulting in a lightly damped system which exhibits more vibration than conventional (contact-type) drives. This is an undesirable property inherited by these drives due to their

contactless nature, and, active control schemes must be resorted to for meeting the stringent positioning demands on them. This is the issue addressed in this work.

Active control finds many applications like aircraft panel vibration control, machinery vibration control, active noise control, etc. A summary of various applications can be found in [6]. Although active control has been employed for decades, there is a renewed interest in this field because many control algorithms that were difficult to implement earlier are now feasible with the availability of low cost, high bandwidth Digital Signal Processors (DSPs). In a typical active control effort, vibrating modes of the plant are identified first using a frequency sweep test or a white noise excitation test. An algorithm is then devised for cancellation/control of each of these modes. Both single-mode and multimode control can be performed and has been demonstrated in the past, see e.g. [7]. A comparison of different algorithms for independent modal space control (IMSC) may be found in [8].

This work addresses the issue by developing a learning controller. A neural network is used to obtain suitable feedback components. The controller requires only a rough idea about the location of the vibrating modes. Lead Zirconate, Lead Titanate, Piezoelectric Ceramic (PZT) actuators are employed for the dual purpose of controlling and/or injecting vibration at the resonant modes. PZTs have many desirable properties such as high stiffness, precise controllable motion, high force and high bandwidth. They are gaining wide acceptance as actuators for active vibration control. Note however, that other actuators are also being examined for their potential as actuators for vibration control, e.g., [9] demonstrates positioning down to ± 1nm using a voice-coil actuator used as secondary actuator in the system.

1.3 General System Description

Instrumentation on the nut is as shown in Figure 2.1 while a cross-sectional view of the nut is shown in Figure 2.4. The drive is supported by a combination of externally pressurized air journal and thrust bearing on one end and by the nut on the other end. The nut moves along a slide through a rectilinear air bearing. A capacitive sensor and an accelerometer are mounted on the nut to measure the vibration in the axial (z) direction. Two stacks of PZT actuators are mounted on either side of the nut to inject or control the vibration as required. High voltage amplifiers (HVAs) are employed to provide the drive power for the PZTs. The controller is implemented in a PC-DSP data acquisition and signal processing board which uses the TMS320C31 floating point DSP from Texas Instruments. Figure 1.1 shows a basic block diagram of the system.

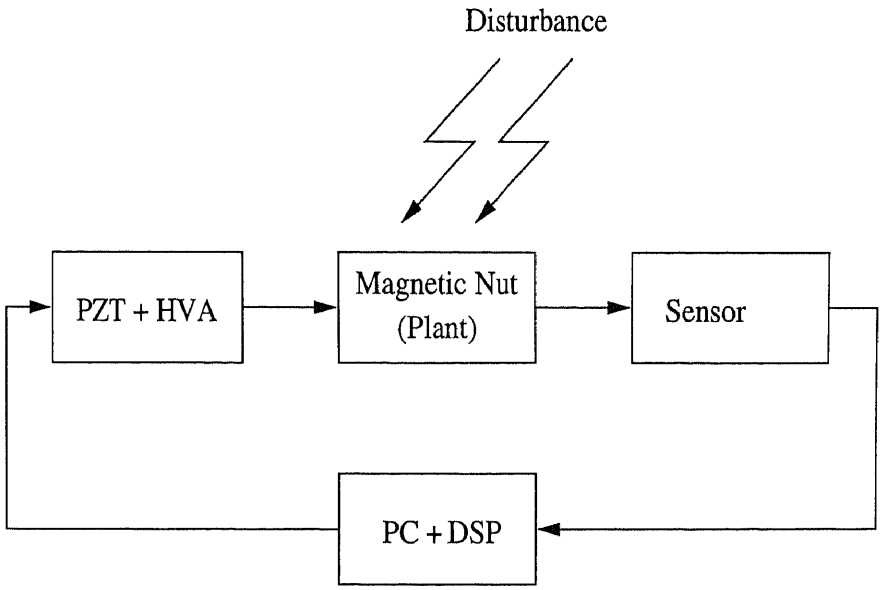

Figure 1.1 Basic Block Diagram of the System

Detailed hardware description of the system follows in Chapter 2. Chapter 3 explains the theory behind the neural network based controller and also the overall controller structure. Software development is the subject of Chapter 4. Test results and discussions are included in Chapter 5.

CHAPTER 2

HARDWARE DEVELOPMENT

This chapter describes the hardware platform for the project. An overview of the system is given first followed by the description of each of the individual components. The PC-DSP development system is explained in detail. A description of the experimental setup consisting of the magnetic leadscrew, and the sensors and actuators, along with their relevant interfacing circuitry, is given next. Parameter specifications of individual components are included in Appendix A.

2.1 System Overview

Figure 2.1 shows the experimental setup of the system.

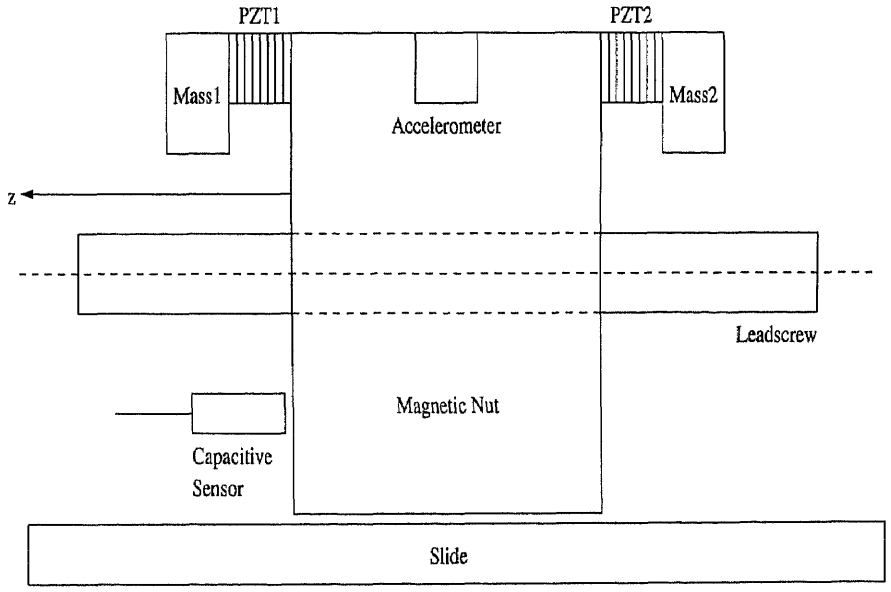

Figure 2.1 Experimental setup of the system

The system consists of a nut and a leadscrew which are magnetically coupled to each other. The nut is mounted on a slide at the bottom and travels in the axial i.e. the z-direction as shown. The system uses PZT actuators to control the nut vibration. Two separate PZT stacks attached with the desired proof mass are mounted on either side of the nut as shown. Each stack can serve the dual purpose of injecting the test

disturbance into the plant (nut) or producing the control motion to cancel/control the disturbance. A low G accelerometer and a capacitive sensor measure the vibration. Figure 2.2 shows the functional block diagram of the system.

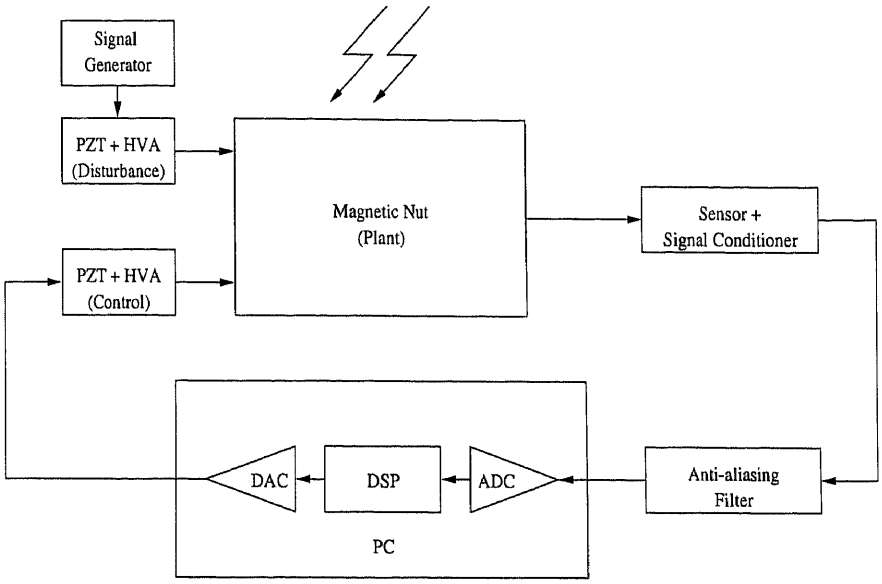

Figure 2.2 Block diagram of the system

The controller is implemented on a PC-DSP development platform which uses the "Model 310 Data Acquisition and Signal Processing Board" from Dalanco Spry. The board uses the TMS320C31 floating point DSP from Texas Instruments and is designed for the ISA (PC-AT) bus of the host PC which has an Intel Pentium-MMX CPU running at 166 MHz.

2.2 Components

2.2.1 PC-DSP Development System

The DSP board has 128 kword on-board RAM with provision for additional expansion. It is interfaced to the ISA bus of the PC. The RAM is dual ported and therefore is accessible by both the PC, as well as, the DSP. It is interfaced to the 16-bit ISA bus via a bus interface. Figure 2.3 shows a block diagram of the board.

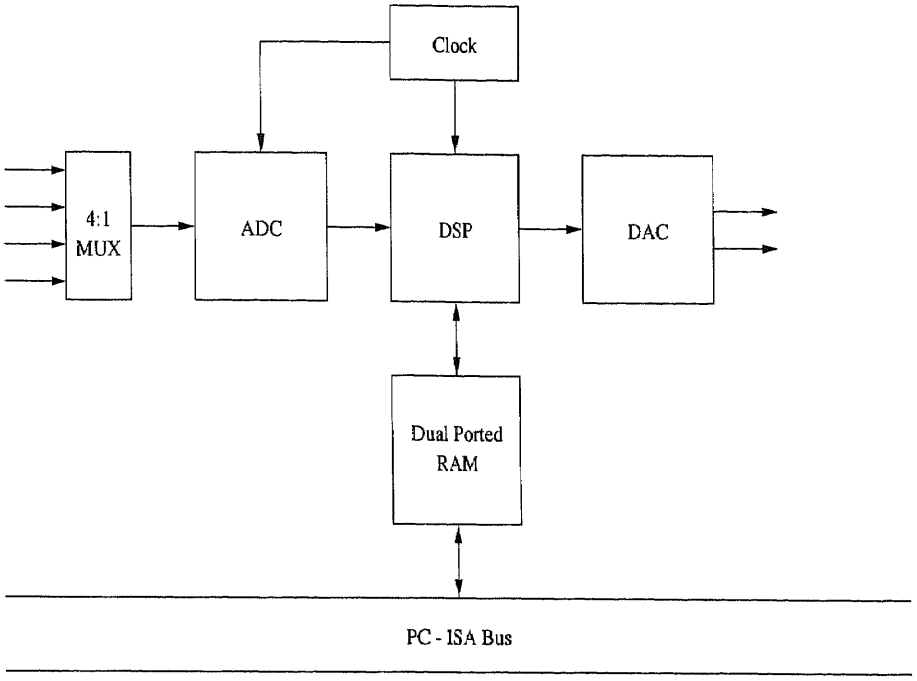

Figure 2.3 Block diagram of the DSP board

A four channel differential multiplexer on the input side facilitates four 14-bit ADC input ports while the output side has two 12-bit DAC ports. Simultaneous Write cycles on the DAC ports in the same cycle are also possible. The voltage range on both ADC and DAC ports is $\pm$ 5V. A programmable gain amplifier (PGA) on the board enables low amplitude signals to be read. Gain ranges from 1 to 1000 with fixed intermediate values provided by the manufacturer. Note that the ADC and the DAC are not dual ported in that they are accessible only by the DSP and not the PC. The DAC can output data with a maximum rate of 140 kHz while the ADC can read at a maximum rate of 300 kHz. The sampling rate for the ADC is programmed from the C31.

The board can be mapped in the PC's I/O space on 8-byte boundaries. Programming may be done in both C as well as assembly. The DSP board is compatible with the Texas Instruments Optimizing C Compiler for the C31 [11].

In addition, two user written libraries are used which contain the C code along with optimized assembly fragments for certain routine functions. This facilitates complete isolation of the programming environment from the architectural details of the board.

2.2.2 C31 Features

The TMS320C31 is a 32-bit floating point DSP from Texas Instruments which combines both system control and mathematical processing functions on a single chip. Key mathematical processing features of the C31 include single cycle MAC (multiply + accumulate) instruction, 32-bit barrel shifter, independent multiplier and ALU units, eight external precision registers with 32-bit mantissa and 8-bit exponent, for intermediate storage of operations and two independent Auxiliary Register Arithmetic Units (ARAUs) for fast address generation in all addressing modes. The separate multiplier and ALU units both handle 32-bit integer and 40-point floating point data. The key system control features which make the C31 a stand-alone single powerful processor include: on-chip 64-word instruction cache, two 32-bit wide on-chip banks of 1 kword RAM and one 32-bit wide on-chip bank of 4 kword ROM, two 32-bit memory-mapped timers, a full-duplex, bidirectional, memory-mapped serial port and most importantly, an on-chip memory-mapped DMA controller. The DMA controller is used to achieve CPU operation concurrent with I/O. The DMA controller handles data transfers between the different on-chip memory resources thereby relieving the CPU of I/O responsibilities. It also performs memory-to-memory transfers and can access any on-chip/off-chip memory address or memory mapped peripherals. The on-chip memory blocks can perform two memory accesses in a single cycle. The serial port can transfer data as 8-bit, 16-bit, 24-bit or 32-bit words. In the present system, the on-chip timer is used as a triggering signal for the ADC conversion cycle. A toggle on the TCLK pin gives a "Start of

Conversion" to the ADC. After "End of Conversion", data is sent from the ADC to the serial port. A read operation copies the data in the CPU.

2.2.3 Magnetic Leadscrew

A cross-sectional view of the contactless drive is shown in Figure 2.4.

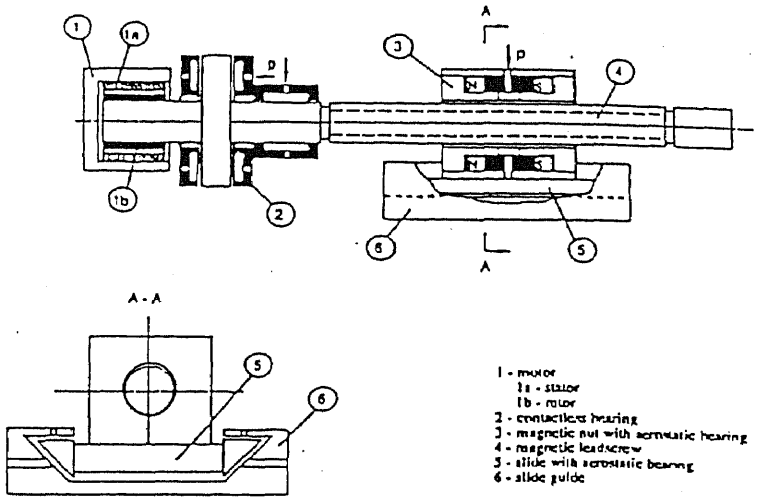

Figure 2.4 Magnetic Leadscrew

The drive is based on the principle of aerodynamic/magnetic suspension. In conventional drive systems, the nut and the leadscrew are mechanically coupled. In the present system, however, magnetic coupling is applied between the two to align the threads of the nut and the leadscrew. This avoids any physical contact between the leadscrew and the nut and eliminates the so-called "hard" nonlinearities such as backlash, hysteresis and surface friction, present in contact-type (mechanically coupled) drives. Superior performance is achieved as a result, giving high resolution,

longer life due to lower wear and tear, lower cost as compared to the conventional ball screw drives and also a wide range of travel. "The leadscrew and nut are made of soft magnetic material with fine rectangular thread. Their spacing is filled with non-magnetic epoxy. Permanent magnets are joined to the nut to supply energy to the magnetic circuit formed by the leadscrew and nut pair. The operating point and the subsequent performance of the permanent magnets depend on both the physical installation of the magnetic circuit and the magnetization of the magnetic circuit after assembly. As shown in the cross-sectional view above, the aerostatic leadscrew system is supported on one end by a combination of externally pressurized air journal and thrust bearing. It is also supported by the nut, with the nut acting as an externally pressurized air journal bearing. The nut moves on a slide through a rectilinear air bearing. High resolution linear movement is achieved by the pitch, which is 0.1 mm in the present system."[1]

2.2.4 PZT and High Voltage Amplifier

The system uses the Lead Zirconate Lead Titanate (PZT) class of actuators to inject/control the vibration of the nut. PZT belongs to the class of induced strain actuators (ISAs). Among other members of this class are electrostrictive actuators, magentostrictive actuators and also shape memory effects [10]. The multilayer stacks of PZTs used in this system exhibit high force and low displacement. Since the displacement (vibration) of the nut is within $\pm$ 25 microns, the stiffness requirement of the PZTs is not as high as that for span-lengths exceeding hundreds of microns. [2] Two separate piezoelectric stacks with suitable proof mass are mounted on either side of the nut. One stack (PZT1) is used to inject test disturbance while the other (PZT2) is used to produce the control motion. These functions can be interchanged. The PZTs require high voltage of the order of hundreds of volts to provide the desired control motion. The control inputs (from the DAC) and disturbance inputs (from

the signal generator) are therefore fed to high voltage amplifiers with an amplification factor of 100. Outputs of these High Voltage Amplifiers eventually feed the command to the PZTs.

2.2.5 Sensors

As shown in Figure 2.1, two sensors are used to measure the vibration of the nut. A low G accelerometer and a capacitive sensor are mounted on the drive as shown to measure the axial (z-direction) acceleration and displacement the nut, respectively.

2.2.5.1 Capacitive Sensor: The capacitive sensor has a range of $\pm$ 25 microns which corresponds to half rotation of the nut. A charge amplifier is used to convert the capacitive sensor's output to a dual polarity voltage signal. The resulting sensitivity is 0.4 V/micron. Parameter specifications are given in Appendix A.3.

Mounting of the capacitive sensor is a critical implementation issue. In the present system, a magnetic stand is used to hold the capacitive sensor. The stand is mounted on the same slide on which the nut travels. This reduces any common mode disturbances that may be present. However, because of the magnetic stand, the overall assembly is similar to a cantilever beam with the result that the vibration modes of the magnetic stand also appear in the frequency spectrum obtained from the capacitive sensor. If these modes "mix" with those of the nut then it becomes difficult to identify the "true" modes of the nut. It is extremely important to identify the correct resonant modes if proper cancellation is desired. Confronted with this situation, we take the following precautionary steps: First, the capacitive sensor is mounted as low as possible on the magnetic stand. This reduces the stand vibration significantly. In addition, the frequency spectra as obtained from the capacitive sensor and the accelerometer are compared. The accelerometer is mounted directly on the nut and therefore does not introduce any erroneous modes in its spectrum.

Lastly, the magnetic stand is placed at various positions and the position where the spectra obtained from the two sensors are in agreement, is selected. Sensor placement is an extremely critical part in the hardware implementation of this system.

2.2.5.2 Accelerometer: The system uses the model CXL04M3 triaxial accelerometer from Crossbow Technology. The silicon micromachined accelerometer operates on a single +5V supply and provides a direct analog output voltage which can be interfaced to the ADC channels directly. It is epoxyed on the magnetic nut such that the x-axis of the accelerometer is aligned with the z-direction of the nut. The factory calibrated zero-G output on the x-axis is 2.475 G. Following relation is used to calculate the output acceleration:

$$Acceleration = (Vout \ - \ zero - G \ offset \ voltage) * (scalefactor) \ (G) \quad (2.1)$$

Scale factor for the x-axis is 500 mV/G. Despite factory calibration, the accelerometer was re-calibrated before any tests were conducted and the zero-G voltage from the x-axis (after mounting the accelerometer on the nut) is 2.48 V. Note that the DC offset is adjusted (nullified in software) before any operations are performed on the data.

Due to ground vibration and other external mechanical vibration, the accelerometer data is extremely noisy for low amplitude vibrations. In such cases, a simple inverting amplifier stage with a suitable gain is introduced between the accelerometer and the ADC as shown in Figure 2.5. Parameter specifications of the accelerometer are given in Appendix A.4.

2.3 Anti-aliasing Filter

An analog low-pass filter precedes ADC in order to avoid aliasing effects. The controller operates at a 2 kHz sampling rate. Hence, an analog low-pass filter with an ideal cut-off at Nyquist Frequency of 1 kHz is desirable. However, a cut-off frequency

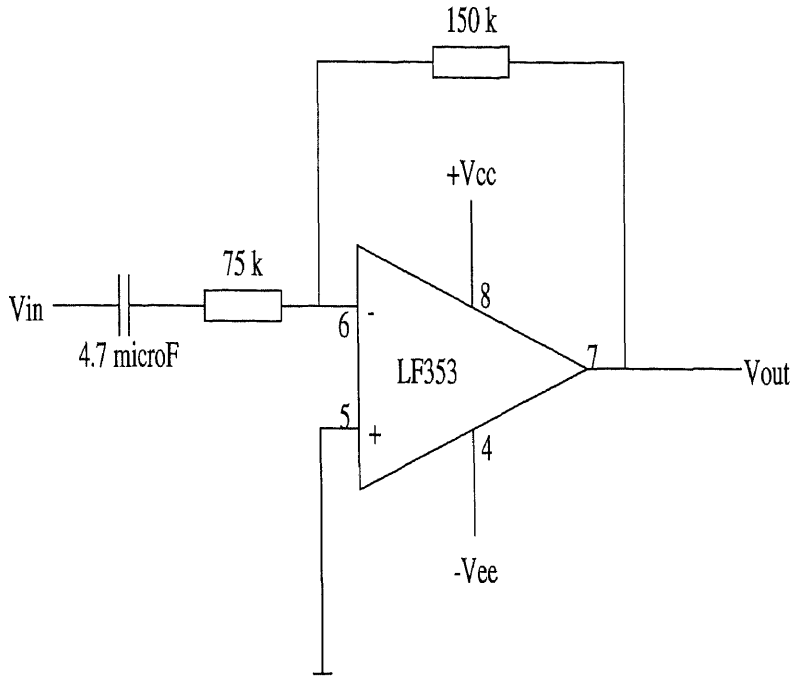

Figure 2.5 Amplifier Circuit

lower than this frequency is desirable for two reasons. First, the accelerometer has a first-order low-pass characteristic with bandwidth set at 100 Hz. And second, in its present position, the dominant modes of the magnetic stand used to mount the capacitive sensor are at 225 Hz and 256 Hz. Hence, an analog low-pass filter with a cut-off lower than these frequencies will facilitate the identification and later cancellation of the true modes of the nut. In the present system, the filter is designed to have a cut-off frequency of $\approx$ 200 Hz. Circuit diagram is shown in Figure 2.6.

$$f_c = \frac{1}{2\pi RC} \tag{2.2}$$

$$C_1 = \sqrt{2} \times C \tag{2.3}$$

$$C_2 = \frac{1}{\sqrt{2}} \times C \tag{2.4}$$

$$\Rightarrow C = \frac{1}{2\pi R f_c} \tag{2.5}$$

Choose $C_2 = 3.3\mu F$ which gives $C_1 = 6.6\mu F$.

Let $C_1 = 6.8\mu F$ which gives: $C \approx 4.67\mu F$,

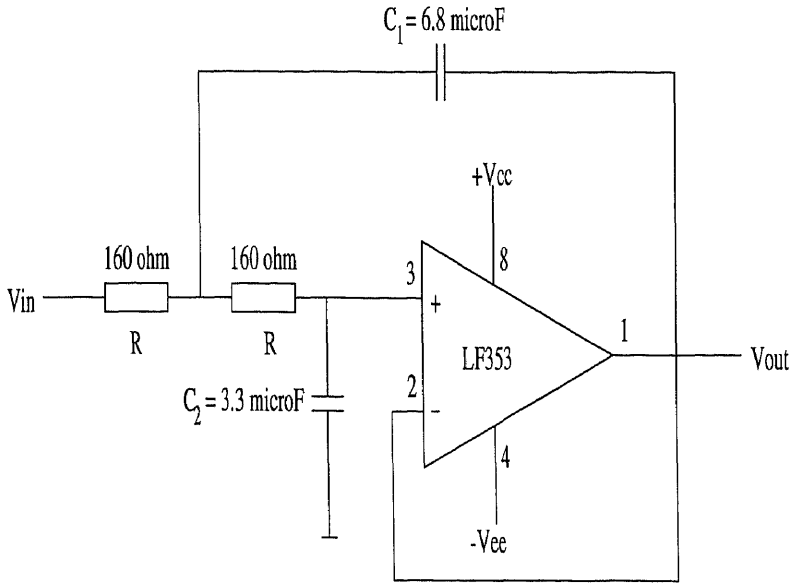

Figure 2.6 Anti-aliasing analog low-pass filter

$f_c = 200Hz \quad R \approx 170\Omega.$

We choose $R = 160\Omega$ which gives $f_c = 213Hz$.

Parameter Specifications of the various components discussed in this chapter are given in Appendix A.

CHAPTER 3

CONTROL ANALYSIS

This chapter explains the theory behind the neural network based controller. Plant structure is explained first and a low-frequency equivalent model is derived via the Hilbert transform. This is the model used for passband control. The Neural Mode Separator (NMS) is discussed next. Overall controller structure along with experimental procedure is given at the end.

3.1 Model Development

The magnetic nut has an open-loop response as shown in Figure 3.1.

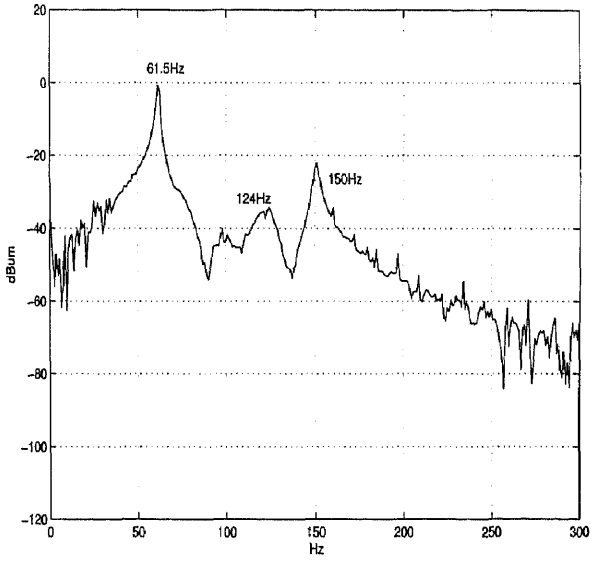

Figure 3.1 Open-loop response of the plant

We start by obtaining an approximate model for this response by implementing the method of curve-fitting. This is done in Matlab on an iterative basis until the fit matches/approximates the various slopes and breakpoints of the actual response. An exact fit which includes the "weak" mode at 124 Hz results in a transfer function of a very high order. Hence, only the two dominant modes at 61.5 Hz and 150 Hz are used to obtain an approximate fit which is shown in Figure 3.2.

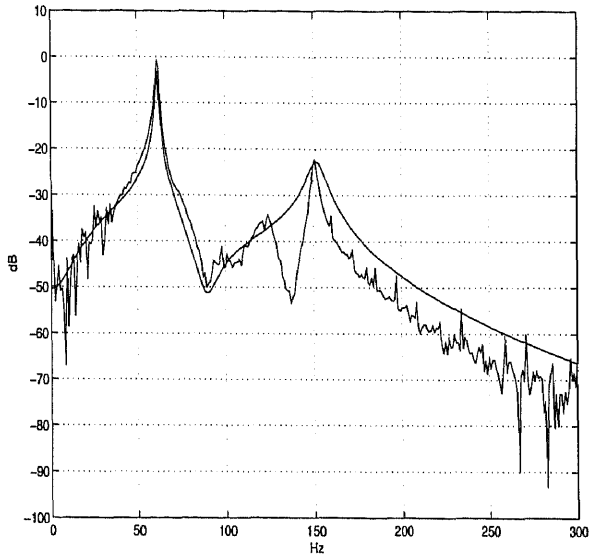

Figure 3.2 Resonant modes of the nut for m = 0 kg

This fit results in an open-loop transfer function given in Equation 3.1.

$$T(s) = 1 \times 10^3 \frac{A}{B} \tag{3.1}$$

where

$$\begin{aligned} A &= -1.25s^4 - 2.67 \times 10^2 s^3 - 4.11 \times 10^5 s^2 - 6.21 \times 10^7 s - 2.32 \times 10^9 \\ B &= s^9 + 3.35 \times 10^3 s^8 + 5.29 \times 10^6 s^7 + 6.02 \times 10^9 s^6 + 5.18 \times 10^{12} s^5 + \\ &\quad 3 \times 10^{15} s^4 + 1.20 \times 10^8 s^3 + 3.75 \times 10^{20} s^2 + 8.04 \times 10^{22} s + 7.75 \times 10^{24} \end{aligned}$$

The 9^{th} order open-loop transfer function has

zeros at:

$-27.96 \pm j585.5,$

-94.25 *and*

-62.83

and poles at:

$-28.65 \pm j954.61,$

$-1053.69,$

$-1053.69,$

$-3.83 \pm j383.25,$

$-565.49 \qquad and$

$-303.48 \pm j3.09 \times 10^{-5}$

Note that the transfer function is load dependent. In other words, whenever the load on the nut changes, the transfer function in Equation 3.1 no longer holds in its present form. This is because with change in the load conditions, the oscillating modes of the nut shift to a different location. In other words, the plant dynamics are variable. Hence, a controller based on an open loop transfer function of Equation 3.1 does not guarantee consistent performance which, in the present case, is stabilization of the oscillating modes. It is, therefore, of general interest to seek a robust scheme for the controller.

3.2 Passband Control

Vibration of the nut can be treated either as undamped/underdamped internal dynamics of the plant or as external disturbance. The controller developed in this work is based on the former approach and needs to have only a rough idea about the number and location of dominant modes.

As mentioned in Chapter 1, the contactless drive is a lightly damped structure. Most lightly damped structures have energy content in a number of separate passbands which are band-limited. Each band can be considered to be a separate plant and a modular structure can be obtained as shown in Figure 3.3.

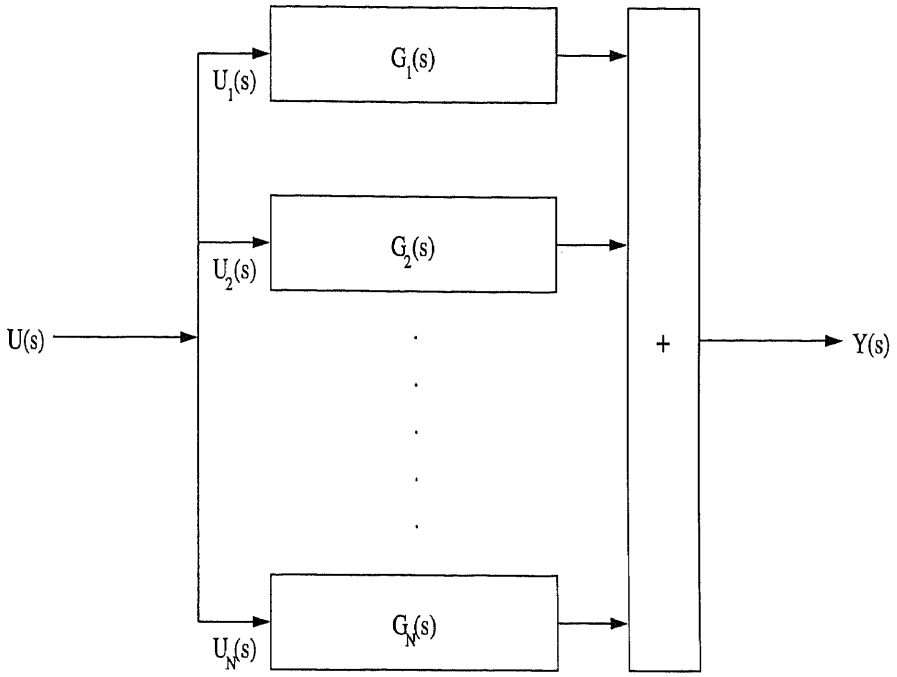

Figure 3.3 Modular plant structure

Band limited nature of passbands suggests the use of Hilbert Transform which is known Communication Systems community for the key role it plays in bandwidth conservation in Single Sideband (SSB) modulation [12]. Using Hilbert Transform, we can determine the pre-envelope and the baseband complex-envelope of a given signal as will be explained in later sections. Therefore, we look at each oscillating mode as a separate narrowband modulation frequency centered around a carrier frequency. Hilbert Transform can then be used to obtain an equivalent model in the baseband for each of the individual modes of the plant (subsections of the controller). We proceed by defining a linear plant and identifying the control objectives.

3.2.1 Plant Definition

The plant is assumed to be linear and time invariant given by:

$$\dot{x} = Ax + Bu \tag{3.2}$$

$$z = Cx \tag{3.3}$$

Let the transfer function be given by:

$$G(s) = C(sI - A)^{-1}B = \sum_{i=1}^{N} G_i(s) \tag{3.4}$$

where

$$G_i(s) = \frac{c_{i_1} + c_{i_2}s}{s^2 + 2\zeta_i\omega_i s + \omega_i^2} \qquad i = 1, 2, \ldots, N \tag{3.5}$$

This is the transfer function of the i^{th} subsystem of Figure 3.3.

Advantage of parallel decomposition is that all $G_i(s)$ are automatically decoupled in the frequency domain [1]. Therefore, for the i^{th} subsystem, we have,

$$Y_i(s) = G_i(s)U(s) \tag{3.6}$$

Since the open loop system is lightly damped, the output of each subsystem can be written in time domain as:

$$y_i(t) = A_i(t)sin(w_i^d t + \theta_i(t)) \tag{3.7}$$

$$= y_i^c(t)cos\omega_i^d t - y_i^s(t)sin\omega_i^d t \qquad i = 1, 2, \ldots, N \tag{3.8}$$

where $y_i^c(t), y_i^s(t) \in \Re$ are, respectively, the low-frequency in-phase and quadrature components of $y_i(t)$.

Let $amp(y_i(t)) = \sqrt{(y_i^c(t))^2 + (y_i^s(t))^2}$ be defined as the amplitude of $y_i(t)$ and let y_i^{ref} be defined as the reference signal. Let the regulation error be defined as:

$$e_i(t) = amp(y_i(t)) - y_i^{ref} \qquad i = 1, 2, \ldots, N \tag{3.9}$$

Then the control objectives [1] are:

1. Stabilization: $(amp(y_i(\infty)) = 0 \qquad i = 1, 2, \ldots, N$
2. Regulation: $e_i(\infty) = 0 \qquad i = 1, 2, \ldots, N$
3. Robustness: Properties (1) and (2) hold under small parametric perturbation in $\zeta_i, \omega_i, c_{i_1}$ and c_{i_2} *where* $i = 1, 2, \ldots, N$

Condition (2) implies condition (1) by setting y_i^{ref} to zero.

The i^{th} passband has a bandwidth of $2\zeta_i\omega_i$ and a center frequency of $\omega_i^r = \omega_i\sqrt{1 - 2\zeta_i^2}$.

A low-frequency equivalent model is now derived.

3.2.2 Hilbert Transform

The Hilbert Transform [12] of a continuous time signal $x(t)$ is given by:

$$\mathcal{H}[x(t)] = \hat{x}(t) = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{x(\tau)}{t-\tau}d\tau \tag{3.10}$$

and the inverse is given by:

$$\mathcal{H}^{-1}[\hat{x}(t)] = x(t) = -\frac{1}{\pi}\int_{-\infty}^{\infty}\frac{\hat{x}(\tau)}{t-\tau}d\tau \tag{3.11}$$

where the integrals are assumed to be Cauchy's principal values.

The frequency domain relationship is given by:

$$\hat{X}(\omega) = -jsgn(\omega)X(\omega) \tag{3.12}$$

where $\hat{X}(\omega)$ and $X(\omega)$ are the Fourier transforms of $\hat{x}(t)$ and $x(t)$, respectively. A pictorial representation of the Hilbert Transformer is shown in Figure 3.4.

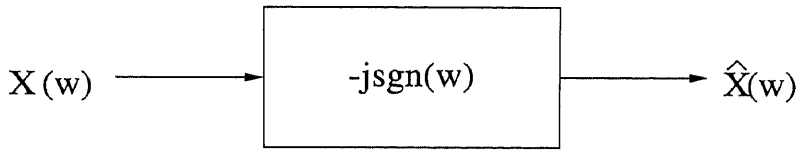

Figure 3.4 Hilbert Transformer

Essentially, the Hilbert Transform performs the function of shifting the phase of the input signal by 90^0.

3.2.3 Low-Frequency Equivalent Model

Let the output of each individual section be denoted by $y_i(t)$ and its Hilbert transform by $\hat{y}_i(t)$. Let the pre-envelope of $y_i(t)$ be given by:

$$\bar{y}_i(t) = y_i(t) + j\hat{y}_i(t) \qquad \in \mathcal{C} \tag{3.13}$$

Then, it is observed that,

$$y_i(t) = Re[\bar{y}_i(t)] \tag{3.14}$$

$$= Re[(\bar{y}_i(t)e^{-j\omega_i^d t})e^{j\omega_i^d t}] \tag{3.15}$$

$$= Re[\tilde{y}_i(t)e^{j\omega_i^d t}] \tag{3.16}$$

Equations 3.15 and 3.16 are obtained using the frequency shifting property of the Fourier Transform [12]. The quantity $\tilde{y}_i(t) = \bar{y}_i(t)e^{-j\omega_i^d t} \in \mathcal{C}$ is known as the complex envelope of $y_i(t)$. $\tilde{y}_i(t)$ is a low-pass signal with bandwidth Ω_i and is expressed as:

$$\tilde{y}_i(t) = y_i^c(t) + jy_i^s(t) \tag{3.17}$$

where $y_i^c(t)$ and $y_i^s(t)$ are the low-frequency in-phase and quadrature components of $y_i(t)$. Therefore, we can write,

$$y_i(t) = y_i^c(t)cos\omega_i^d t - y_i^s(t)sin\omega_i^d t \tag{3.18}$$

In other words, any signal $y_i(t)$ can be shifted to the baseband using the low-frequency in-phase and quadrature components. This is possible only because of the Hilbert Transform since the complex envelope can only be derived from the pre-envelope which is given by the Hilbert Transform. Equation 3.18 is the expression for the output of each of the subsections of Figure 3.3.

Proceeding on the same lines, we can also decompose the impulse response matrix of the section as:

$$G_i(t) = 2G_i^c(t)cos\omega_i^d t - 2G_i^s(t)sin\omega_i^d t \tag{3.19}$$

where the factor 2 is introduced for notational convenience. Note that $G_i^c(t)$ and $G_i^s(t)$ are stable iff. $G_i(t)$ is stable.

Thus, the section output $y_i(t)$ can be expressed as:

$$y_i(t) = G_i(t) * u(t) \tag{3.20}$$

$$= Re[\bar{G}_i(t) * \bar{u}(t)] \tag{3.21}$$

where $\bar{G}_i(t) = G_i(t) + j\hat{G}_i(t)$, "*" denotes convolution and $\hat{G}_i(t) = \mathcal{H}[G_i(t)]$.

Let

$$\tilde{y}_i(t) = \tilde{G}_i(t) * \tilde{u}_i(t) = (G_i^c(t) + jG_i^s(t)) * (u^c(t) + ju^s(t)) \tag{3.22}$$

Observe that

$$y_i(t) = y_i^c(t)\cos\omega_i^d t - y_i^s(t)\sin\omega_i^d t \tag{3.23}$$

where

$$y_i^c(t) = G_i^c(t) * u_i^c(t) - G_i^s(t) * u_i^s(t) \tag{3.24}$$

$$y_i^s(t) = G_i^s(t) * u_i^c(t) + G_i^c(t) * u_i^s(t) \tag{3.25}$$

where $u_i^c(t)$ and $u_i^s(t)$ are the low-frequency components of $u(t)$ aligned in the i^{th} passband.

The baseband equivalent model of the i^{th} resonant section is now obtained as:

$$Y_i^b(s) = G_i^b(s)U_i^b(s) \tag{3.26}$$

where

$$G_i^b(s) = \begin{bmatrix} \mathrm{G}_i^c(s) & \text{-}\mathrm{G}_i^s(s) \\ \mathrm{G}_i^s(s) & \mathrm{G}_i^c(s) \end{bmatrix} \tag{3.27}$$

$$Y_i^b = \begin{bmatrix} \mathrm{Y}_i^c & \mathrm{Y}_i^s \end{bmatrix}' \tag{3.28}$$

and

$$U_i^b = \begin{bmatrix} \mathrm{U}_i^c & \mathrm{U}_i^s \end{bmatrix}' \tag{3.29}$$

The plant output $y(t)$ is given as:

$$y(t) = \sum_i^N y_i^c(t)\cos\omega_i^d t - y_i^s(t)\sin\omega_i^d t \tag{3.30}$$

$$z(t) = Cy(t) \tag{3.31}$$

Therefore $y_i(t)$ can be extracted from $z(t)$ by suitable band-pass filtering or "mode separation".

Since the impulse response of $G_i(t)$ is given by:

$$\frac{H_i}{\omega_i^d} e^{-\zeta_i\omega_i t}[-\zeta_i\omega_i \sin\omega_i^d t - \omega_i^d \cos\omega_i^d t] \tag{3.32}$$

and $G_i^b(s)$ is calculated as:

$$G_i^b(s) = \frac{H_i}{2\omega_i^d(s+\zeta_i\omega_i)} \begin{bmatrix} \omega_i^d & \zeta_i\omega_i \\ \text{-}\zeta_i\omega_i & \omega_i^d \end{bmatrix} \tag{3.33}$$

If the in-phase (cosine) component only is used for feedback control, i.e.

$$u_i^c(t) = Ky_i^c(t)cos\omega_i^d t \tag{3.34}$$

$$= K(c_{i2}\dot{x}_i + c_{i2}\zeta_i\omega_i x_i) \tag{3.35}$$

it can be readily shown that the closed loop eigenvalues are given by:

$$s^2 + (2\zeta_i\omega_i - Kc_{i2})s + (\omega_i^2 - Kc_{i2}\zeta_i\omega_i) = 0, \qquad i = 1, 2, \ldots, N \tag{3.36}$$

Stabilization of the plant now requires shifting all N poles further into the left-half plane. A bandwidth-conservative controller can now be synthesized for each of the N resonant sections.

3.2.4 Fixed Regulator Synthesis

The low-frequency equivalent transfer function for the i^{th} subsystem with the in-phase component used for feedback is:

$$G_i^c(s) = \frac{c_{i2}}{2(s + \zeta_i\omega_i)} \tag{3.37}$$

For regulation, i.e. for sustained oscillation, a PI controller may be employed [1]. This supplies an equivalent closed-loop transmission zero at the origin. For the present work, however, we need stabilization. Hence, the integral action has been eliminated and a simple proportional controller has been employed. Successful functioning of the bandwidth conservative controller now depends on the availability of the in-phase component which is generally mixed at the output stage. The neural mode separator does the job of extracting this component.

3.2.5 Neural Mode Separator (NMS)

The NMS belongs to the class of Recurrent Neural Networks [13]. A general n-input, n-output neural network scheme is shown in Figure 3.5.

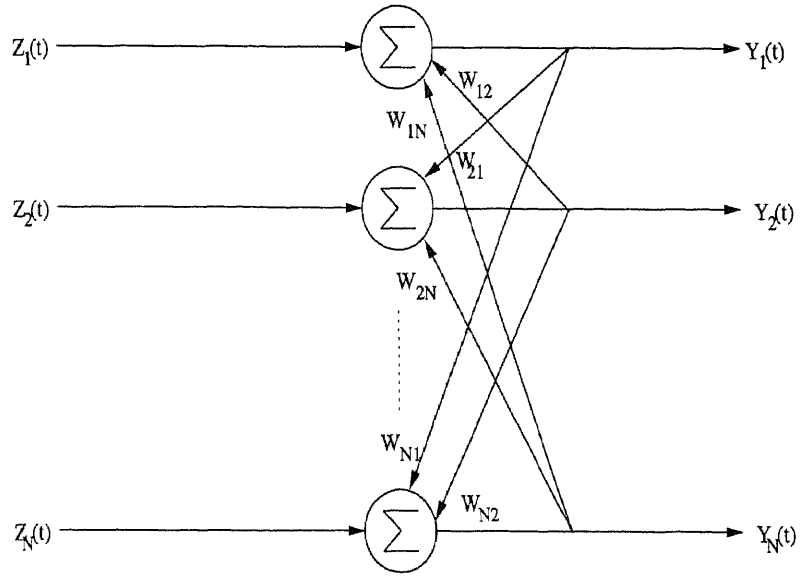

Figure 3.5 A general Recurrent Neural Network

The adaptation rule is given by:

$$w_{ij}^{+} = w_{ij}^{-} - \mu\alpha(\tilde{y}_i)\beta(\tilde{y}_j) \qquad i \neq j \qquad i,j \in [1,N] \tag{3.38}$$

The parameter μ represents the learning rate and $\alpha(.), \beta(.)$ are odd, locally smooth functions. For the case of n = 2, a robust update rule for the "blind separation" of the elementary signals that are statistically independent [13].

$$\omega_{ij}^{+} = \omega_{ij}^{-} - \mu\tilde{y}_i{}^3\tilde{y}_j, i \neq j, i,j \in [1,2] \tag{3.39}$$

For the present case, resonant modes are treated as elementary signals and orthogonality between them is treated as statistical independence.

Since,

$$\hat{y}(t) = W^{-1}z(t) = W^{-1}Cy(t) \tag{3.40}$$

where $C = [c_{ij}]$ and since

$$W = \begin{bmatrix} 1 & w_{12} & w_{13} & \cdots & w_{1N} \\ w_{21} & 1 & w_{23} & \cdots & w_{2N} \\ \vdots & & \cdots & \vdots & \\ w_{N1} & w_{N2} & w_{N3} & \cdots & 1 \end{bmatrix} \tag{3.41}$$

the objective of the weight adaptation scheme is to force $WH = P_N D$ where P_N is a general permutation matrix and D is a nonsingular diagonal matrix and the relations obtained are:

$$z = Hy = Cx \tag{3.42}$$

$$\tilde{y} = WHy \tag{3.43}$$

This gives the required statistical independence condition: For the present case, it translates to the following: each mode acts as one elementary signal. The other elementary signal is obtained by employing a filter introducing an arbitrary phase-shift. These two elementary signals can then be fed to the neural network whose job will be to make them statistically independent i.e. orthogonal, thus giving the desired in-phase and quadrature components.

3.2.6 Controller Structure

Overall controller structure is shown in Figure 3.6.

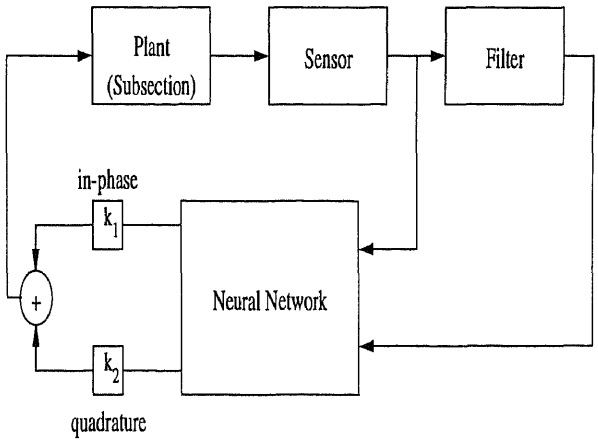

Figure 3.6 Overall controller structure

The controller is implemented in real-time inside the DSP. Output of the subsection (mode) obtained using a suitable sensor is fed to a front-end digital low-pass filter inside the controller. The cut-off frequency of the filter is designed to be higher than the frequency of the mode since, for the present system, the purpose of the filter is not to attenuate a particular component but only to introduce a phase-shift. (Note that

an all pass filter may also be used to perform the same task but design of a digital all pass filter is much more complicated than its counterpart). The original signal and its phase-shifted version are fed to the neural network which adjusts the phase-shift between the two signals to 90^0. Selection of the parameter value (μ) for the neural network is the deciding factor in the speed of learning. The issue is discussed in much detail in Chapter 5. Scalar gains k_1 and k_2 and the proceeding summing junction in Figure 3.6 are used to obtain a linear combination of the in-phase and quadrature components. This is required since these components, when applied individually, do not appear in the correct phase at the nut due to the transmission delay between the controller output at the DAC channel and the actuator mounted on the nut (plant). This is explained in greater detail in Chapter 4.

3.3 Experimental Procedure

For each load condition, the resonant modes of the nut are identified. Test procedure for identification of modes is given in Chapter 5. The digital low-pass filter coefficients are selected for the particular mode under consideration. Tests are conducted for self-resonance which is a ringing effect observed whenever additional mass (load) is placed on the nut. For self-resonance, the sensor output is simply fed to the filter. Suitable parameter value (μ) is chosen for the neural network. Gains k_1 and k_2 are tuned to stabilize the ringing. For another set of tests, an external sinewave disturbance is injected into the nut at the frequency of the desired mode and the same procedure is repeated. Once the parameters k_1, k_2 and k_p are tuned to obtain the maximum reduction in vibration, a third test is performed where a step disturbance is applied to the nut and the time required for stabilization of the transient part of the response without control action is recorded with that under control action. These test results are included in Chapter 5. Chapter 4 explains the software and the various implementation issues arising in the implementation of the controller.

CHAPTER 4

SOFTWARE

The control program is developed in C. Some user written libraries with embedded fragments of C31 assembly are also used. These libraries facilitate user friendly interfacing of the DSP board with the PC. The PC is used for performing supervisory functions, data acquisition and further processing while the DSP is used to implement the control algorithm. The control Program is written in the PC and the compiled code is downloaded into the DSP where the controller is implemented in real-time. A brief description of the user written libraries is given first. Control program is discussed next. Data exchange between the PC and the DSP is given in the end.

4.1 User Written C Library

This library performs two main tasks. First, it initializes the signal processing board and second, it performs data exchange between the analog I/O channels and the DSP. The interface provided by Dalanco - manufacturer of the board - for the I/O channels is time consuming and complex. The C library overcomes this drawback by encapsulating the required functions. This provides a high level interface between the PC and the DSP and the user is isolated from the details of the board architecture. Each of the functions implemented are now described in turn. Note that these functions are a part of the "d310bio.h" header file which is included at the beginning of the control program. Code listing for the header file is given in Appendix B.

4.1.1 DSP Board Initialization

The DSP board requires initialization which is performed by the *InitDsp()* function in "d310bio.h". The function call prototype is:

void InitDsp(void)

Note that the ADC on the board is connected to the serial port of the DSP. Hence, serial port, timer and the latch on the board need to be properly initialized during the *InitDsp()* function. The function begins by initializing some pointers. The latch in the DSP board contains two quantities: the current channel number of the ADC with default set to *channel0* and the gain of programmable gain amplifier (PGA), default being unity. Next, the function sets up the memory mapped chip registers of the DSP. These are the timer registers and the serial port control registers. The pointers mentioned above are used to write to the respective addresses of the register. This completes the initialization of the communication between the DSP and the ADC on the serial port. The *ReadAdc()* and *WriteDAC()* functions are now called as required to perform analog I/O as explained in the following sections.

4.1.2 Analog Output

Analog output is generated on the DAC channels by means of the *WriteDAC()* function which has the prototype:

int WriteDAC(int value, int channel)

The legal values for *channel* are 0 and 1 and those for *value* are in the range -2048 to +2047. To avoid roll over effects, values greater than +2047 are clamped to +2047 and those less than -2048 are clamped to -2048. The *WriteDAC()* function writes to only one channel at a time. If both channels are required to be updated in a single cycle, the function *WriteDACs()* may be used. The function prototype is:

int WriteDACs(int value0, int value1)

In a single cycle, both *channel0* and *channel1* are updated with the values *value0* and *value1* respectively.

4.1.3 Analog Input

To input analog values from the ADC, the function *ReadAdc()* is called which has the following prototype:

int ReadAdc(int channel)

This reads the value from the ADC for the voltage applied on the specified channel and returns values ranging from -2048 to +2047 for voltages ranging from -5 V to +5 V. The necessary sign extension is performed internally in the function. To read from more than one channel, multiple calls to *ReadAdc()* are necessary. Since the ADC has four channels, legal values for *channel* are from 0 to 3. The voltage at the ADC input is calculated by multiplying the ADC data by the scale factor (5/2047).

4.1.4 Determination of Sampling Rate

The sampling rate is determined by the constant TIMPER0 specified at the beginning of the control program. Since functions *WriteDAC()* and *ReadAdc()* both wait for the falling edge of the TCLK signal of the DSP, it is evident that Timer0 register of the DSP decides the sampling rate. The functions are active low. The function *InitDsp()* puts the value of the constant TIMPER0 into the Timer0 register which is used as a counter. The constant must be defined before the header file "d310bio.h" is included in the control program in order to preempt the assignment of the default value for the sampling rate which is declared in the header file. TIMPER0 is calculated from the formula:

$$SampleRate = \frac{SystemClock}{(TIMPER0)(numcalls)(8)} \tag{4.1}$$

where *System Clock* = 40 MHz. Both *ReadAdc()* and *WriteDAC()* consume one cycle each. The factor *numcalls* in the above formula is the total number of reads and writes taken together, in the control loop.

4.2 Control Program

The control program has two parts: initialization section and the control loop.

4.2.1 Initialization

The DSP board is initialized first by the function *InitDsp()*. Next, a suitable sampling rate is selected for the controller. From Figure 5.1, we see that, for no load condition, the frequency of the highest mode is 150 Hz. For any additional load, the modes shift to lower frequencies. Hence, 150 Hz may be taken as the highest "frequency of interest". This gives a Nyquist Rate of 300 Hz. Hence, a sampling frequency more than 200 Hz would meet the demands of the present application. However, too small a sampling frequency results in longer delays in the signal path introducing unnecessarily high phase shifts. Therefore a sampling frequency of 2 kHz is chosen for this application. The corresponding value of TIMPER0 as obtained from Equation 4.1 is 1250 with $numcalls = 2$ — the number of reads and writes in the present control loop.

Coefficients of the digital filter and its desired order are initialized next.

Data storage in the DSP memory is done by means of pointers. During initialization, pointers are assigned to memory locations in the DSP memory space. The size of the data depends on the interval of time for which data is to be recorded. Care should be taken that memory spaces do not overlap when multiple variables are being stored. Next, the neural network parameter (μ) is initialized to a suitable value depending upon the desired speed of learning. Weights of the neural network, declared as global pointers, are set to zero during initialization. Scalar gains for the in-phase and quadrature components, k_1 and k_2, are also initialized to suitable values. Note that μ, k_1, k_2 and k_p are declared as global variables because, once tuned, their values remain constant throughout the control program. Hence, declaring them locally during each iteration of the control loop is not lucrative keeping in mind the

precious "time between samples" available to the controller. Weights of the neural network are stored in global pointers since every "present" iteration refers to the weights calculated in the "last" iteration.

4.2.2 Control Loop

The ADC data is read from the selected channel. Any DC bias present in the data is removed by adding/subtracting the appropriate offset and the data is stored in the DSP memory at the current pointer location. Note that since storage is done by means of pointers, any other data of interest may also be stored by simply changing the pointer assignments. Storage has been done in two ways. For storing the transient response, storage begins as soon as control motion is applied and continues until the transient part settles. On the other hand, for storing the steady-state response, storage is disabled for the first few samples. The number of these samples depends on the time required for the sensor output to settle to its steady state value.

The *filter()* function is placed outside the control loop and is invoked during each iteration of the loop. Each sample is passed as an argument to *filter()*. The prototype of the function is:

int filter(int)

As explained in Chapter 3, the neural network requires, at its inputs, the original signal and its phase shifted version with an arbitrary phase shift between them. To obtain this phase shifted signal, the control loop implements the digital low-pass filter using the function *filter()*. The filter is implemented as a transposed direct form II structure whose initial/starting coefficients are selected during initialization of the control program and updated with successive iterations of the control loop. Extreme care should be taken to ensure that the starting coefficients are designed for the current sampling frequency since the same coefficients with a different sampling frequency give an altogether different, sometimes even unstable filter. This

is especially important during debugging. A general second order transposed direct form II transposed lattice structure is shown in Figure 4.1.

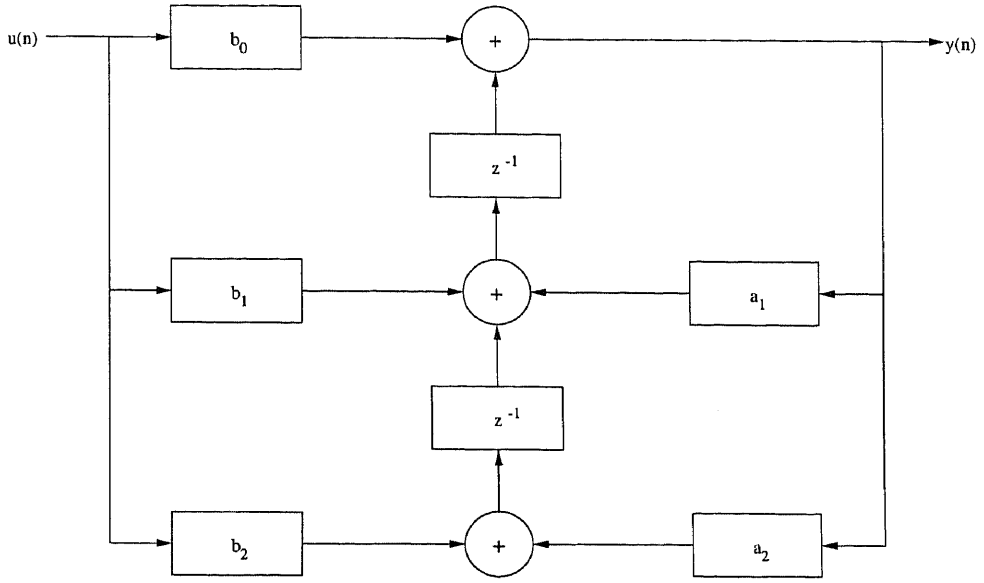

Figure 4.1 Structure of a second order transposed direct form II filter

Transfer function for this structure is given by:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{4.2}$$

This structure is preferred because it requires the least number of delay elements (equal to the order of the filter N) and summation points or "taps" as compared to other structures e.g. the direct form I structure which requires 2N number of delay elements. In other words, the transposed direct form II structure requires less memory and is therefore more suitable for real-time applications. Code fragment implementing the *filter()* function is given below.

```
int filter(int u) //2nd order transposed direct form II filter
{
  int y;
  int i;

  // calculate filter output
  y = u * aryB[0] + aryZ[1];
```

```
  //update aryZ[i]
  for(i=1;i<order;i++)
  {
    \hspace*{5ex}aryZ[i] = u * aryB[i] - y * aryA[i] + aryZ[i+1];
  }
  aryZ[order] = u * aryB[order] - y * aryA[order];
  return(y);
}
```

Notice that the code is flexible, in that, any order and any type (low-pass, high-pass, band-pass, band-stop, etc.) can be implemented by simply switching the coefficient values. Coefficients of the low-pass filter used in the present case, are selected using the Matlab function

$$[b,a] = \text{butter}(n,wn)$$

where n is the order of the filter and wn is the cut-off frequency - a number between 0 and 1 - where 1 corresponds to the Nyquist Frequency which is half the sampling frequency [15]. Note that since the purpose of the filter is only to shift the phase, a simple second order Butterworth filter may be used. The cut-off frequency is selected to be slightly higher than the frequency of mode under consideration. Order should not be too high since it introduces noise. The *filter()* function is written outside *main()* and is invoked during each iteration of the control loop.

As things stand, we now have two signals with the same frequency but different phase. In other words, we have the so called "elementary signals" mentioned in Chapter 3. We next make these signals "statistically independent" — a condition which is relaxed to orthogonality for the present case. The task is performed by the neural network which is implemented in the control program by the function *neural()*. The original signal and the filter output are passed as arguments to *neural()*. Like *filter()*, *neural()* too is implemented outside *main()* and is invoked during each iteration of the control loop. The prototype for the function is:

void neural(int, int)

The neural network bears a parameter μ which decides its rate of learning. Higher the value of μ, faster do the network weights "adapt" to their final value, faster do the outputs converge. Chapter 5 contains detailed plots showing the learning behavior of the neural network including a comparison of simulated learning rates for different parameter values. It is worth repeating that μ is initialized to a suitable value in a global variable during initialization section of the control program. Also, the weights are stored globally by means of pointers and are set to zero during initialization. Code fragment implementing the neural network is given below.

```
void neural(int sig1, int sig2)
{
  float var1,var2,command;

  //calculate network outputs
  *z1 = sig1 + ((*w12)*sig2);
  *z2 = ((*w21)*sig1) + sig2;

  //scale the network outputs
  var1 =  k1 * (*z1);
  var2 =  k2 * (*z2);

  //add the outputs
  command = var1 + var2;

  //apply suitable gain
  command = kp * command;

  //output ''command'' on both channels
  WriteDACs(command,command);

  //update weights
  *w12 = (*w12) - (u*(*z1)*(*z2));
  *w21 = (*w21) - (u*(*z1)*(*z2)*(*z2)*(*z2));
}
```

Any variables in the above fragment which are stored by means of pointers may be stored in the DSP memory, if required, by suitable assignments. Both in-phase and quadrature components obtained from the neural network can be used individually

for feedback. However, as briefly mentioned in Chapter 3, it is observed that the interface circuitry between the DAC and the PZT (which is actually mounted on the nut), introduces a transmission delay between the command written to the DAC channel and the command reaching the PZT. Hence, the in-phase or the quadrature components do not appear at the nut in the correct phase when applied individually and the desired cancellation is not achieved. As a remedy to this, each component is multiplied by a scalar gain stored in the global variables k_1 and k_2. Addition of the scaled outputs is initially written to the DAC channel. The gain values k_1 and k_2 are tuned so as to obtain maximum reduction in vibration of the nut as seen at the sensor output which is used as the ultimate performance criterion. Once this desired phase-shift is obtained, a proportional gain (k_p) is used to maximize the reduction. A schematic of the foregoing is shown in Figure 4.2.

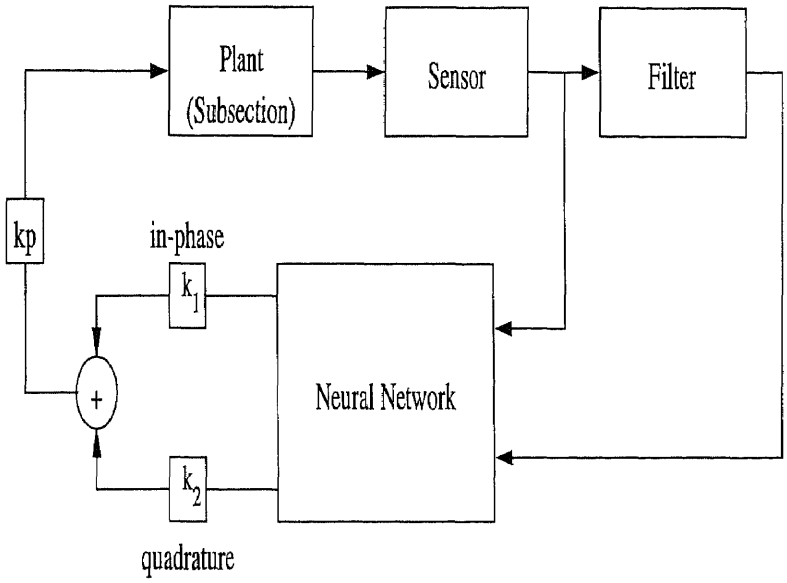

Figure 4.2 Combination of neural network outputs

Chapter 5 contains actual values for k_1, k_2 and k_p used to obtain the reduction of each mode.

Lastly, the data (typically sensor output) stored in the DSP memory is exchanged with the PC memory for further processing. This is explained in the next

section. Program listing for the entire program is given in Appendix B. An overall flowchart of the control program is shown in Figure 4.3.

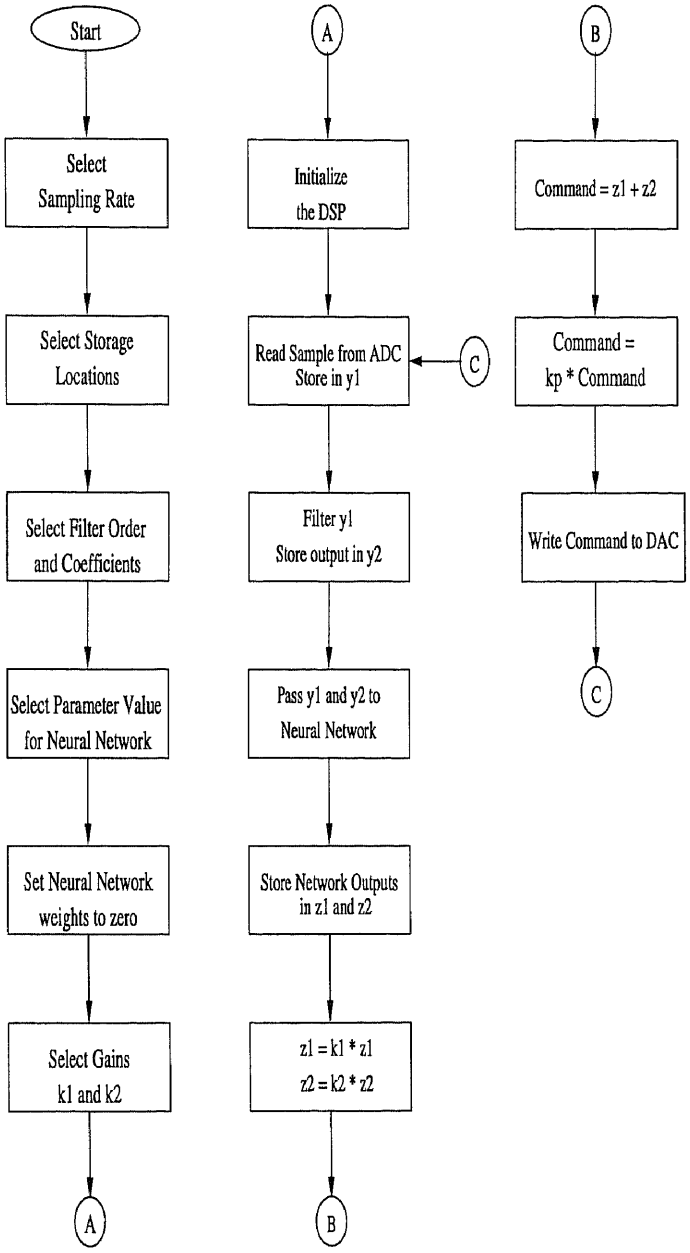

Figure 4.3 Flowchart of the Control Program

4.3 Data Exchange Between PC and DSP

Dalanco, manufacturer of the board, provides user libraries that have embedded functions for data transfer between the PC and the DSP. Additionally, they include functions for starting and halting the DSP. These functions are explained now.

4.3.1 Starting the DSP

The *go320()* function is used for starting the DSP. The function prototype is:

go320(unsigned baseio)

where *baseio* is the Base I/O address of Model 310B (300h at present). The function reads a byte from the address (*baseio*+6). This causes the DSP to start execution of the currently loaded program, provided the signal $\overline{IORD}$ is asserted. This logic is implemented in hardware.

4.3.2 Halting the DSP

The DSP is halted by the function *hlt320()* which has the prototype

hlt320(unsigned baseio)

As before, *baseio* is the Base I/O address of Model 310B. This function reads a byte from the address (*baseio*+7) causing the DSP to be halted.

4.3.3 Data Transfer between DSP Memory and PC Memory

Blocks of data can be exchanged by the PC with the DSP using the functions *sendio()* and *recvio()*. Both these functions have identical prototypes as follows:

sendio(long *x, unsigned length, long start, unsigned baseio)

recvio(long *x, unsigned length, long start, unsigned baseio)

where

x :	an array of 32-bit words.
length :	the number of words in array x. Altered depending on the length of time interval for recording the data.
start :	source start address in Model 310 memory. Altered as required.
baseio :	the base I/O address of Model 310. This is jumper selectable and is currently set to 300h in the PC's I/O space.
sendio :	copies the contents of x into the DSP board's memory starting at location *start*.
recvio :	copies the DSP board's memory contents starting at memory location *start* into x.

The function *recvio()* is used often to transfer the sensor data or any other intermediate signals from the DSP into the PC. The function is embedded into the C program "grab.c". The code for the program is self explanatory and is included in Appendix B.3. Test results are now discussed in Chapter 5.

CHAPTER 5

TEST RESULTS

Test procedure of the magnetic leadscrew/nut stabilization experiment consists of the following steps: First, the axial (z-direction) resonant modes are identified and characterized with respect to a range of loading conditions. Second, the Neural Mode Separator (NMS) passband control is applied to stabilize these modes under two conditions: self resonance and forced oscillation. For self resonance, the nut is allowed to go into undamped oscillation (driven by background noise). The control objective here is to increase the damping so that "ringing" is minimized. In case of forced oscillation, a sinewave tuned to the resonant frequency is injected into the disturbance PZT so that an oscillation is observed at the sensors. The goal is to reduce the transmission gain of the axial dynamics at the resonant frequencies. Plant dynamics are shown first, followed by a discussion on the neural network's operation. This is followed by tests for self resonance, forced oscillation and transient response.

5.1 Plant Dynamics

Resonant modes of the nut are shown in Figure 5.1.

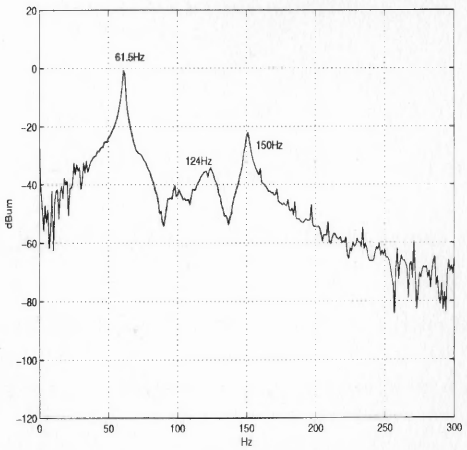

Figure 5.1 Resonant modes of the nut; m = 0 kg

These are the natural modes of the nut without any additional mass. Identification of these modes consists of the following procedure: A large step disturbance is applied to the nut which excites the resonant modes. An FFT is performed on the response obtained at the sensor, which gives the desired mode plot. Plant dynamics remain fixed for a given load condition. Hence, real-time FFT is not required. FFT is performed in Matlab after "grabbing" the sensor output from the DSP memory into the PC. Identical test procedure is followed for other load conditions. Matlab code for FFT is included in Appendix B.5.

5.2 Neural Mode Separator

Figure 5.2 shows the Lissajous Figure for the phase shift between the sensor data and its phase shifted version at the digital filter's output which are shown as y1 and y2, respectively, in the controller structure of Figure 3.6.

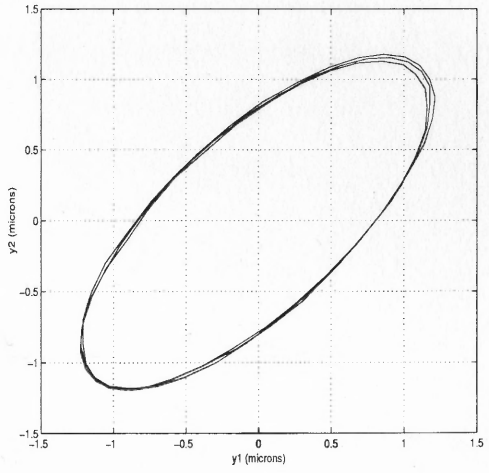

Figure 5.2 Initial phase-shift between filter outputs

These are the "elementary signals" mentioned in Chapters 3 and 4 that are fed to the neural network. With statistical independence relaxed to orthogonality, the Lissajous Figure must represent a normally centered ellipse. Figures 5.3 through 5.5 show the successive learning steps of the neural network for $\mu = 10^{-12}$.

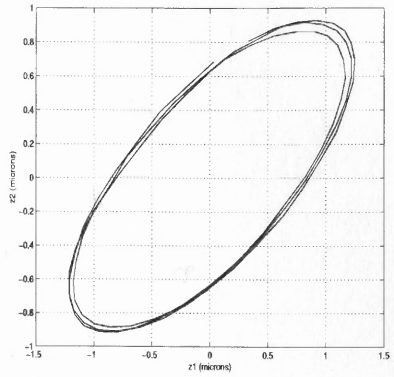

Figure 5.3 Neural Network outputs in learning mode: I

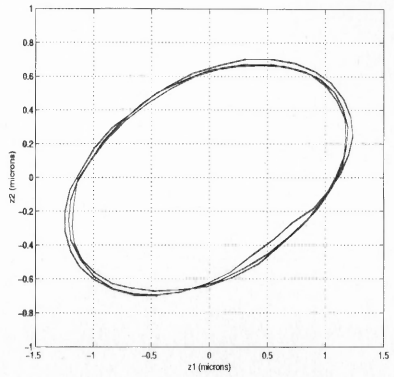

Figure 5.4 Neural Network outputs in learning mode: II

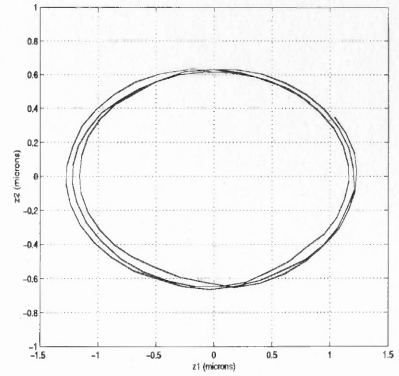

Figure 5.5 Neural Network outputs in learning mode: III

As can be seen, the outputs of the neural network start with an initial phase shift between each other that is equal to the phase shift between its inputs and converge to the orthogonality condition (phase shift = 90^0) as the network learns. The speed of learning of the network is determined by the parameter μ whose value

should be as high as possible for fastest possible convergence. Figure 5.6 shows the simulation results for the rate of learning. For the set of results shown, with $\mu = 0.01$, the network learns about three times faster than for $\mu = 0.0025$.

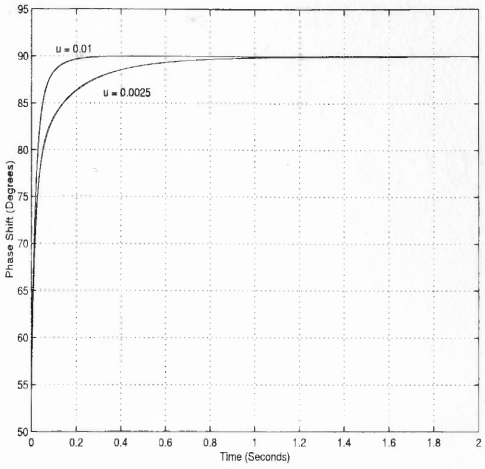

Figure 5.6 Simulation results showing rate of learning for different μ-values

However, there is an upper limit on the μ-value beyond which the network's outputs distort from the desired sinusoidal shape and eventually become unstable. Figure 5.7 shows the time domain plot of the network outputs for $\mu = 10^{-10}$.

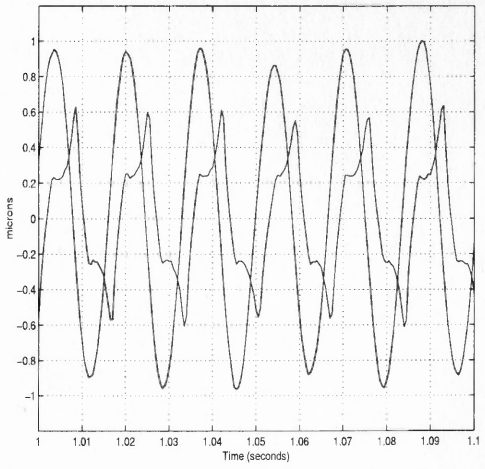

Figure 5.7 Unstable output due to high μ-value (time domain plot)

The corresponding Lissajous Figure is shown in Figure 5.8. Proper operation of the network must be ensured for all test conditions.

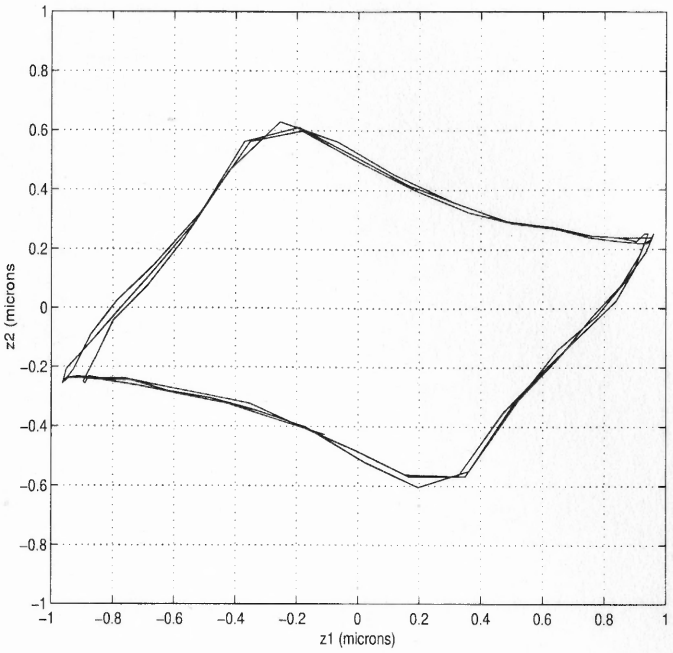

Figure 5.8 Unstable output due to high μ-value (Lissajous Figure)

5.3 Self Resonance Tests

The magnetic nut is subjected to considerable background noise and ground vibration. This results in a pronounced "ringing effect" whenever additional mass is placed on top of the nut. The ringing is undesirable considering the positioning demands on the drive. It is, however, observed that it has a single frequency of resonance and can therefore act as a candidate for external disturbance. Performance of the controller is evaluated in the presence of this self resonance.

It was observed that $\mu = 10^{-11}$ was the most suitable value for an appreciable rate of learning and is the value for all tests conducted for both self resonance as well as external disturbance.

5.3.1 m = 1.1 kg

A mass of 1.1 kg is added and the effect of control is observed. Figure 5.9 shows the transient response of the nut to the control action. Both responses, with and without control are shown for comparison.

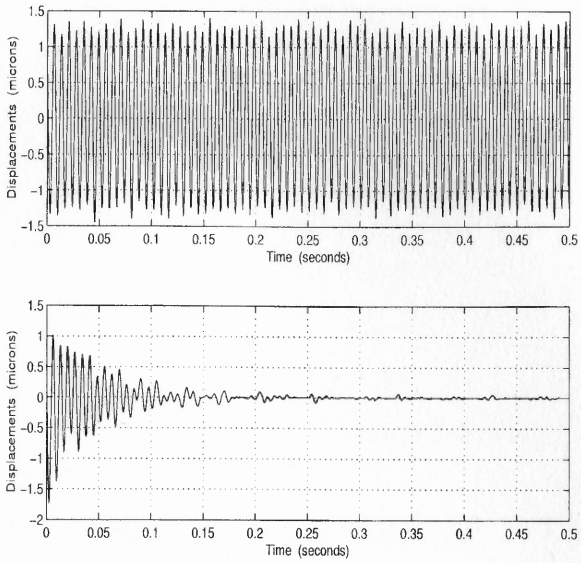

Figure 5.9 Self resonance cancellation (transient part); m = 1.1 kg

Figure 5.10 compares the steady state parts of the two responses.

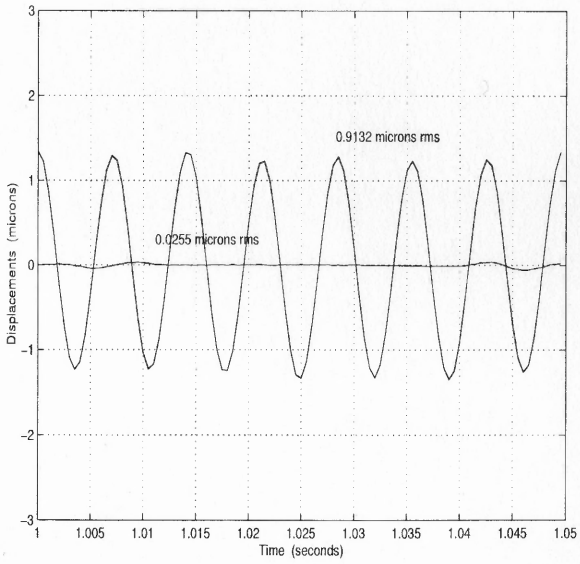

Figure 5.10 Self resonance cancellation (steady state part); m = 1.1 kg

A visibly convincing FFT of the nut response is shown in Figure 5.11. As can be seen, self-resonance component is virtually eliminated.

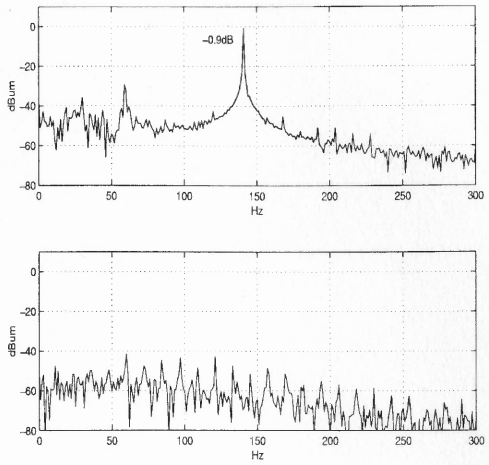

Figure 5.11 FFT of self resonance; m = 1.1 kg

Parameter value for this case with a 150 Hz filter cut-off are:

b = [0.0413, 0.0825, 0.0413]

a = [1, -1.3490, 0.5140]

Gain values: k_1 = 1.4; k_2 = -1; k_p = -7.

5.3.2 m = 2 kg

A second test run is conducted with an overall additional mass of 2 kg. Consistent performance is delivered by the controller as depicted in Figures 5.12 through 5.14.

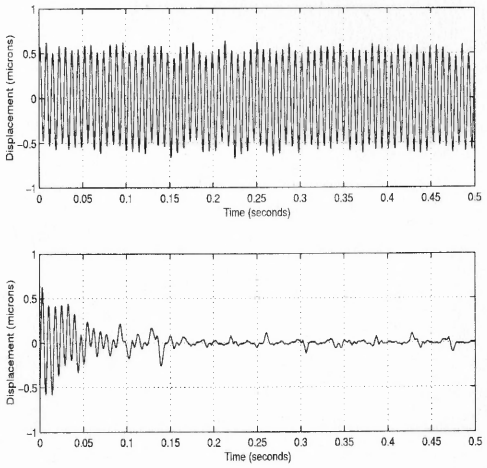

Figure 5.12 Self resonance cancellation (transient part); m = 2 kg

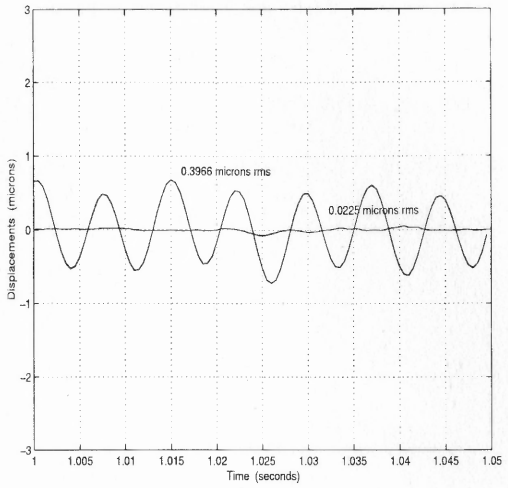

Figure 5.13 Self resonance cancellation (steady state part); m = 2 kg

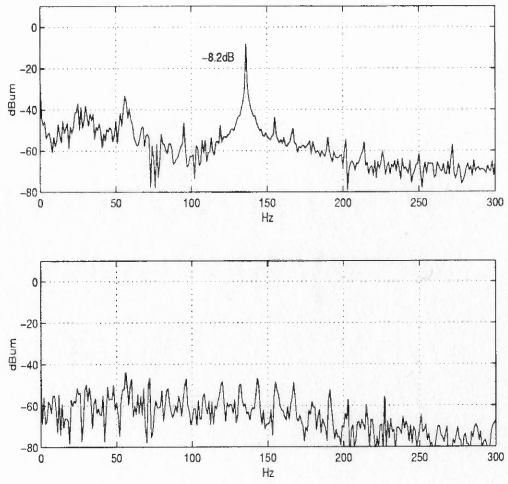

Figure 5.14 FFT of self resonance; m = 2 kg

Parameter values are:

Filter cut-off at 150 Hz gives:

b = [0.0413, 0.0825, 0.0413]

a = [1, -1.3490, 0.5140]

Gains: k_1 = 1.1; k_2 = -1.3; k_p = -4.

5.4 Forced Oscillation Tests

5.4.1 m = 0 kg

Procedure for identifying the resonant modes was discussed in Section 5.1. The mode plot was shown in Figure 5.1 and is repeated here in Figure 5.15.

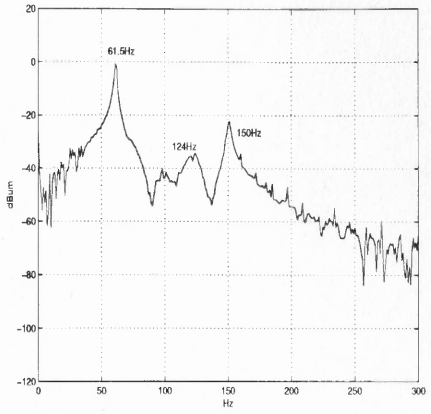

Figure 5.15 Resonant modes of the nut; m = 0 kg

For forced oscillation, each mode is excited individually by injecting a sinewave into the disturbance PZT at that particular frequency. Figure 5.16 shows the reduction obtained with external disturbance at mode 1 (61.5 Hz).

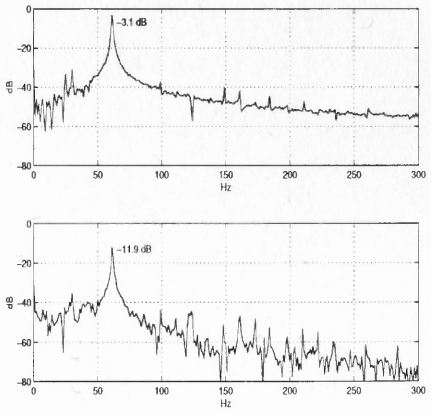

Figure 5.16 Vibration control at mode 1; m = 0 kg

Parameter values with 70 Hz filter cut-off are:

b = [0.0104, 0.0209, 0.0104]

a = [1, -1.6910, 0.7327]

Gains values are: k_1 = -1; k_2 = 1; k_p = 6.

As can be seen, an overall reduction of ≈ 9 dB is obtained. Figure 5.17 shows the reduction obtained for external disturbance at mode 2 (150 Hz).

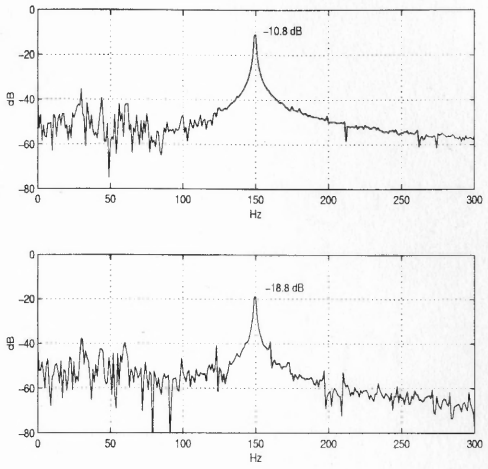

Figure 5.17 Vibration control at mode 2; m = 0 kg

With a 160 Hz cut-off for the filter, the parameter are:

b = [0.0461, 0.0923, 0.0461]

a = [1,-1.3073,0.4918]

Gains: $k_1 = 1$; $k_2 = 3$; k_p = -6.

The attenuation in this case is 8 dB.

Thus, consistent reduction of 8 to 9 dB is obtained in each case. Note that the control objective here is to increase the damping on the nut by shifting the roots further into the left-half plane which gives a reduced axial gain. The controller is not designed to cancel a particular mode as in case of a servo compensator (notch filter) approach. Hence, complete cancellation of the mode is not expected in presence of externally injected disturbance.

5.4.2 m = 1.1 kg

A mass of 1.1 kg is now placed on top of the nut and the same tests as in Section 5.4 are conducted for single tone excitation at each mode. Figure 5.18 shows the mode distribution. The modes shift down in frequency from their locations in the earlier

(m = 0 kg) case. This confirms that all modes obtained in the earlier case are true modes.

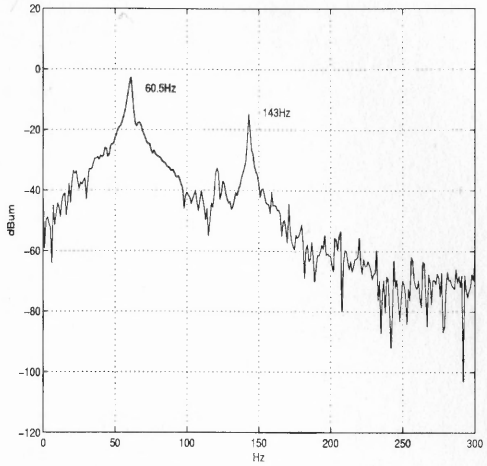

Figure 5.18 Resonant modes of the nut; m = 1.1 kg

Figure 5.19 shows the attenuation in mode 1 (60.5 Hz) under control action.

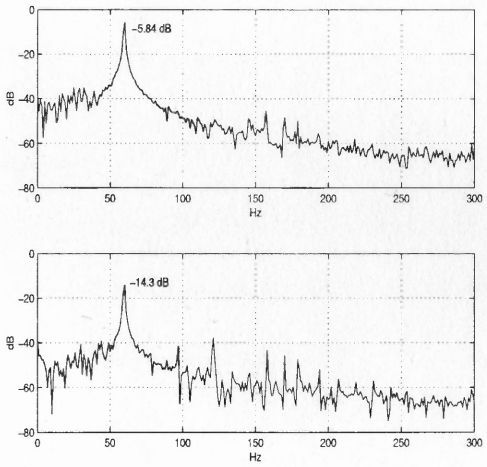

Figure 5.19 Vibration control at mode 1; m = 1.1 kg

With a 70 Hz filter cut-off, following parameters are obtained:

b = [0.0104, 0.0209, 0.0104]

a = [1,-1.6910,0.7327]

Gain values: k_1 = 0.3; k_2 = -1; k_p = -15.

Lastly, Figure 5.20 shows the reduction in axial gain with external disturbance at mode 2 (143 Hz) under control action.

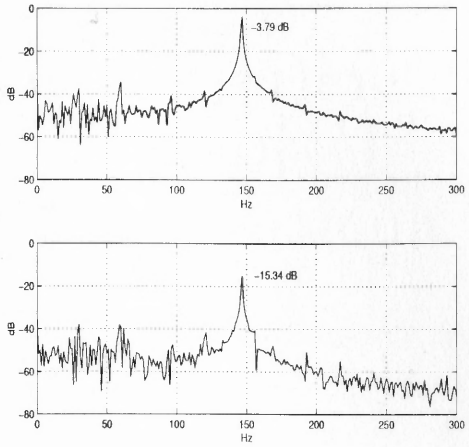

Figure 5.20 Vibration control at mode 2; m = 1.1 kg

Filter cut-off is at 150 Hz. Overall parameters are:

b = [0.0413, 0.0825, 0.0413]

a = [1, -1.3490, 0.5140]

Gains values are: $k_1 = 1$; $k_2 = 1$; k_p = -12.

5.4.3 Transient Response

A final test is conducted to examine the control of transient response. A step input is applied using a pendulum which makes an impact on the nut with the same starting angle each time. Results are shown in Figure 5.21.

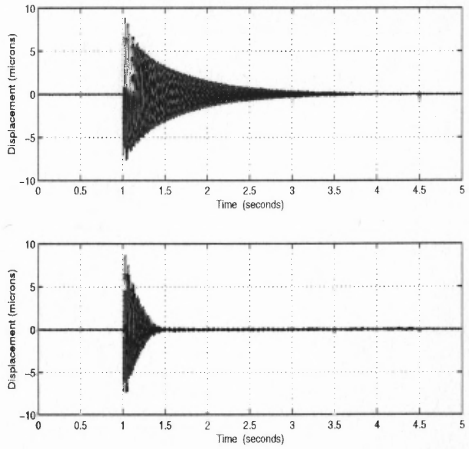

Figure 5.21 Control of transient response

As can be seen, the time required for stabilization is significantly reduced under control action. FFT of the two responses are shown in Figure 5.22.

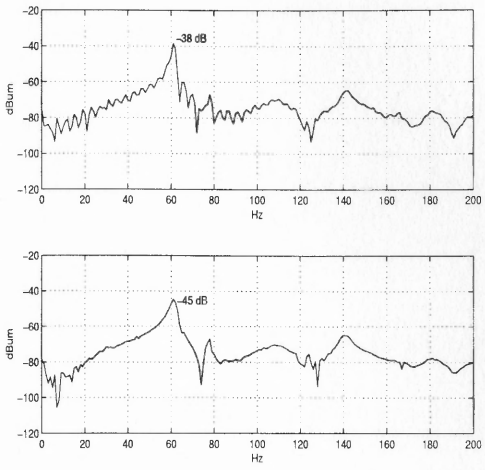

Figure 5.22 FFT of nut response

For this particular test run, control is applied at mode 1 (62.5 Hz) which is the dominant mode of the nut as shown in Figure 5.1 and is the significant mode contributing to the vibration. For further improvement in the time to stabilize, multimode control may be applied along with higher actuator power which are the issues discussed in the Section 5.5.

A summary of test results and corresponding parameter values is given in Tables 5.1 and 5.2 respectively.

Table 5.1 Test Results

Test Conditions	Mass	Mode	Without Control		With Control	
			dB	Ripple (μm rms)	dB	Ripple (μm rms)
Self Resonance	1.1 kg	141 Hz	-0.9	0.9132	-63.5	0.0255
	2 kg	136 Hz	-8.2	0.3966	-66.6	0.0225
Forced Oscillation	0 kg	61.5 Hz	-3.1	0.8718	-11.9	0.3131
		150 Hz	-10.8	0.4504	-18.8	0.1846
	1.1 kg	60.5 Hz	-5.84	0.6481	-14.3	0.2496
		143 Hz	-3.79	0.7736	-15.34	0.2077

Table 5.2 Test Parameters

Test Conditions	Mass	Mode	Filter	k_1	k_2	k_p
Self Resonance	1.1 kg	141 Hz	150 Hz	1.4	-1	-7
	2 kg	136 Hz	150 Hz	1.1	-1.3	-4
Forced Oscillation	0 kg	61.5 Hz	70 Hz	1	-1	-6
		150 Hz	160 Hz	1	3	-6
	1.1 kg	60.5 Hz	70 Hz	0.3	-1	-15
		143 Hz	150 Hz	1	1	-12

5.5 Conclusions and Future Directions

Active vibration control of a contactless magnetic leadscrew is presented in this thesis. A learning controller based on a recurrent neural network is developed. Its ability to separate the required in-phase and quadrature components of the oscillating modes is demonstrated. Self-resonance exhibited by the magnetic nut is cancelled completely while a reduction of typically 8 to 9 dB is obtained in case of forced oscillation. Significant improvement is also achieved in stabilizing the transient response of the nut to a step disturbance which proves the vibration suppression capability of the controller.

The natural first step in future research would be to attempt multimode control. Multiple sets of sensor data, filter coefficients and neural network weights will have to be stored and invoked in an interlaced fashion with commands for individual modes being sent on the DAC channels on successive samples. Number of variables and, consequently, the storage requirement will increase depending upon the intended number of modes to be controlled and duration of data required.

To achieve higher attenuation of forced oscillation, PZT stacks with higher peak force may be used along with additional proof mass. Other locations may also be attempted for the actuators.

Lastly, the present brushless DC motor may be connected to drive the leadscrew. Travel lengths of a few centimeters may be tried initially and the performance of the controller for online (during travel) vibration suppression may be evaluated.

APPENDIX A

PARAMETER SPECIFICATIONS

A.1 Model LTZ-2H Pzt Actuator

Free Dielectric Constant	3400
Loss Tangent	-0.15
Density(gm/cm^3)	7.45
Curie Temperature(C)	195
Coupling Coefficients k_{33}	0.75
Coupling Coefficients k_{31}	0.36
Coupling Coefficients k_p	0.62
Piezoelectric Constant $(m/V) \times 10^{12}$ d_{33}	590
Piezoelectric Constant $(m/V) \times 10^{12}$ d_{31}	-280
Piezoelectric Constant $(V/m)/(N/m) \times 10^3$ g_{33}	19.6
Piezoelectric Constant $(V/m)/(N/m) \times 10^3$ g_{31}	-8.3
Young's Modulli (N/m^2) $\times 10^{-10}$ E_{11}	7.27
Young's Modulli (N/m^2) $\times 10^{-10}$ E_{33}	4.88
Mechanical-Q Q	70
Transverse Frequency Constant $(l.f_r)$ $(kc.in)$ N_1	58
Radial Frequency Constant $(r.f_r)$ $(kc.in)$ N_2	42
Thickness Frequency Constant $(t.f_r)$ $(kc.in)$ N_3	76
Circumference Frequency Constant $(D_m.f_r)$ $(kc.in)$ N_c	40.5

A.2 The Tek Model 601B-PCB High Voltage Amplifier

Input Voltage Range (V)	0 - ± 10
Input Impedance $(k\Omega)$	10
Output Voltage Range (V)	± 500
Output Current Range (mA_{rms})	0 - ± 10 $(20mA_{peak})$
Slew Rate $(V/)$	< 35
Gain Bandwidth Product (MHz)	$> 3MHz$
Noise (mV_{peak})	< 500
Power Requirements	$115/230VAC/50Hz$ - $100Hz/30watts$

A.3 Capacitive Sensor

Supply (V_{dc})	± 15
Operating Range (V)	± 10
Bandwidth (jumper selectable)	5 kHz, 1 kHz (default), 100 Hz, 10 Hz
Scale Factor $(V/\mu m)$	0.4

A.4 The Crossbow Accelerometer Model CXL04M3

Span(G)	± 4
Sensitivity(mV/G)	500
Bandwidth (Hz)	DC_{100}
Noise(mG_{rms})	5
Zero G Output (V)	$+2.5 \pm 0.1$
Span Output (V)	$\pm 2.0 \pm 0.1$
Nonlinearity ($\%FS$)	± 0.2
Alignment($Degrees$)	± 2
Transverse Sensitivity ($\%FS$)	$\pm$ 3.5
Temperature Range (^0C)	-40 to +85
Supply Voltage (V)	$+5 \pm 0.25$
Supply Current (mA)	24

APPENDIX B

CODE LISTINGS

B.1 User Written C Library

```
//d310bio.h
#ifndef D310BIO_H                    // prevent multiple include
#define D310BIO_H

#define TIMER 0
#ifndef TIMPER0
#define TIMPER0         0x50
#endif

#define IOF_AMASK       0x0E
#define IOF_SET_XF1     0x62
#define IOF_RESET_XF1   0x22
#define CTRL            0x808000
#define TIMGB0CONHI     0x6
#define TIMGB0CONLO     0x2
#define TIMGB0START     0x3C1
#define TIMGB0STOP      0x381
#define TIMGB0RESTART   0x341
#define TIMGB1CONHI     0x6
#define TIMGB1CONLO     0x2
#define SERGLOBA        0x1d0144
#define SERGLOB0        0x0C1d0144
#define SERPRTX0        0x111
#define SERPRTR0        0x111
#define SERTIM0         0x3CF
#define SERTIM0VAL      0x01
#define XVALUES         0x809800
#define LATCH_VAL       0x0
#define LATCH_AREA      0x0FFFFFF
#define HIMASK          0x0FFFF0000


void  InitDsp(void)
{
int* p3;
int* p7;

// Initialize ADC 0 (default)
p3  = (int*)LATCH_AREA;
```

```
*p3 = LATCH_VAL;
*p3 = 0;

// Initialise serial port and timer
p7 = (int*)CTRL;

*(p7+64) = SERGLOBA;
*(p7+70) = SERTIM0VAL;
*(p7+68) = SERTIM0;
*(p7+66) = SERPRTX0;
*(p7+67) = SERPRTR0;
*(p7+64) = SERGLOB0;

*(p7+0x64) = 0x18;
*(p7+0x20) = 0;
*(p7+0x28) = TIMPER0;
*(p7+0x20) = TIMGB0START;
}


int ReadAdc(int channel)
{
int* p3;
int* p7;
int  adcvalue, x;
int  i = 0;

x  = channel;
p3 = (int*)LATCH_AREA;
p7 = (int*)CTRL;

// Set the ADC Input Channel
*p3 = x;

// Wait for TIM0 (conversion start) to go HIGH
while(((*(p7+0x20)) & 0x0800) == 0);

// Wait for TIM0 (conversion start) to go LOW
while(((*(p7+0x20)) & 0x0800) != 0);

// Start oscillator to use Serial Port
asm(" OR 60H,IOF");

// Wait for reception of ADC Data
```

```
while(((*(p7+64)) & 0x01) == 0);

// Turn Oscillator OFF to reduce noise
asm(" AND 0DFH,IOF");

// Retrieve the ADC data
adcvalue = (*(p7+76) & 0x0fff0);
adcvalue = adcvalue >> 4;

if((adcvalue & 0x800) != 0)
adcvalue = adcvalue | 0x0fffff000;

return(adcvalue);
}


int WriteDAC(int value,int channel)
{

int* p7;
static int channel_value[2];
int s[2];

ReadAdc(3);


        if((channel != 0) && (channel != 1)) {
exit(1);
}

if(value >  2047) value =  2047;
if(value < -2047) value = -2048;

channel_value[channel] = value;

s[0] = (channel_value[0]) & 0x0fff;
s[1] = (channel_value[1]) & 0x0fff;

asm(" NOP ");
asm(" NOP ");

s[1] = s[1] << 16;

asm(" NOP ");
```

```
asm(" NOP ");


value = s[0] | s[1];

p7 = (int*)CTRL;

while(((*(p7+64)) & 0x02) == 0);

*(p7+0x30) = TIMGB1CONLO;

asm(" NOP");
asm(" NOP");

*(p7+0x30) = TIMGB1CONHI;

*(p7+0x48) = value;

return(0);
}


int WriteDACs(int value0,int value1)
{

int *p7;
int s[2];

ReadAdc(3);

if(value0 >  2047) value0 =  2047;
if(value0 < -2047) value0 = -2048;

if(value1 >  2047) value1 =  2047;
if(value1 < -2047) value1 = -2048;

value0 = value0 & 0x0fff;
value1 = value1 & 0x0fff;
value0 = (value1 << 16) | value0;

p7 = (int*)CTRL;

while(((*(p7+64)) & 0x02) == 0);
```

```
*(p7+0x30) = TIMGB1CONLO;

asm(" NOP");
asm(" NOP");

*(p7+0x30) = TIMGB1CONHI;

*(p7+0x48) = value0;

return(0);
}


inline void InterruptPC(void)
{
asm(" OR 02H,IOF");
asm(" AND 0FBH,IOF");
asm(" OR 04H,IOF");
asm(" AND 0FBH,IOF");
}

#endif                                          // #ifndef D310BIO_H
```

B.2 Control Program

```
//net.c

//select sampling rate = 2 kHz
#define TIMPER0 1250

#include "d310bio.h"

//select filter coefficients
//150Hz LPF with 2kHz sampling
//float   aryB[3] = {0.0413,0.0825,0.0413};
//float   aryA[3] = {1.0000,-1.3490,0.5140};
//float   aryZ[3] = {0};

//select filter order
int order = 2;

//select neural network parameter value
float u = 1e-11;
```

```
//declare global pointers for the
//network's weights (w12, w21) and outputs (z1, z2)
float dummy;
float* w12 = &dummy;
float* w21 = &dummy;
float* z1 = &dummy;
float* z2 = &dummy;

//select storage location in DSP memory
int *s = (int*)0x1000;

//select gain values
float k1 = 1.4;
float k2 = -1;
float kp = -7;

int count, num;

void main()
{
  int sig1,sig2,temp,hold;
  int filter(int);
  void neural(int,int);

  *w12 = 0;
  *w21 = 0;
  count = 0;
  num = 0;

  InitDsp();

  while(1)
  {
    sig1 = ReadAdc(0);
    //correct dc offset
    sig1 -= 100;

    //grab data
    if(count >= 1000 && count<=11000)
    //if(count >= 3500 && count<=13500)
    //if(count < 10000)
    {
      *s = sig1;
```

```
      s ++;
      //count++;
    }
    count ++;

    //scale sensor data (if required)
    sig1 *= 10;

    //filter
    sig2 = filter(sig1);

    //scale filter output (if required)
    sig2 *= 2;

    //pass sensor data and filter output to neural network
    neural(sig1,sig2);
  }
}

//filter function
//implements a transposed direct form II filter
//order preselected during initialization
int filter(int u)
{
  int y;
  int i;

  //calculate output
  y = u * aryB[0] + aryZ[1];

  //update aryZ[i]
  for(i=1;i<order;i++)
  {
    aryZ[i] = u * aryB[i] - y * aryA[i] + aryZ[i+1];
  }
  aryZ[order] = u * aryB[order] - y * aryA[order];
  return(y);
}

//neural network function
void neural(int sig1, int sig2)
{
  float var1,var2;
```

```
  //calculate outputs
  *z1 = sig1 + ((*w12)*sig2);
  *z2 = ((*w21)*sig1) + sig2;

  var1 =  k1 * (*z1);
  var2 =  k2 * (*z2);

  command = var1 + var2;
  command = kp * command;


  //output suitable signals
  //(1) output sensor data and filter output
  //WriteDACs(sig1,sig2);

  //(2) output network outputs without scaling
  //WriteDACs(*z1,*z2);

  //(3) output scaled outputs of the network
  //WriteDACs((int)var1,(int)var2);

  //(4) output combination of network outputs simultaneously
  WriteDACs(command,command);

  //update network weights
  *w12 = (*w12) - (u*(*z1)*(*z2));
  *w21 = (*w21) - (u*(*z1)*(*z2)*(*z2)*(*z2));
}
```

B.3 Data Grab Program

```
//grab.c

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

void main()
{
  long *p,i;
  FILE *fp;
  //open file pointer and allocate required memory
  fp=fopen("reso1.out","wt");
  p=malloc((6000)*(sizeof(long)));
```

```
  //read data from DSP memory using the vendor supplied function
  recvio(p,0x1388,0x1000L,0x300);

  //copy data into the memory pointer
  for(i=0;i<5000;i++)
  fprintf(fp,"%ld\n",p[i]);

  fclose(fp);
}
```

B.4 Matlab Routine to Obtain Time Domain Plots

```
%spect.m
figure;
load test.out;
y=test;

%scale sensor data to volts
y=(y*5/2047);

%capacitive sensor
y=y*2.5;
%accelerometer
%y=y/0.505;

%cancel offset
y=y-mean(y);
%select time vector according to
%current sampling rate and required data length
t=0:0.0005:5;
t=t(1:length(t)-1);

plot(t,y);
%select suitable axes and display parameters
axis([0 0.5 -1 1]);
xlabel('Time (seconds)');
%capacitive sensor
ylabel('Displacement (microns)');
%accelerometer
%ylabel('Acceleration (G)')
grid;
zoom
```

B.5 Matlab Routine to obtain FFT

```
%specf.m
figure;
load test.out;
tdata = test;

%select required duration
tdata = tdata(4300:6300);

%select time and frequency vectors
ts = 0.0005;
i=1:2000;
fs=1/ts;

%scale sensor data
tyout = (tdata(i) * 5 / 2047) * 2.5;

fdata = fft(tyout);

ffdata = fdata(1:1001) * sqrt(2) * ts;
fyout = 20 * log10(abs(ffdata));

%generate a suitable frequency vector
f = fs*[0:1000]/2000;

%select suitable display parameters
plot(f,fyout);
grid;
zoom;
xlabel('Hz');
ylabel('dBum');
axis([0 300 -120 20]);
%gtext('61.5Hz')
%gtext('124Hz')
%gtext('150Hz')
```

B.6 Neural Network Simulation Routine

```
%net.m
%initialize weights
w12 = 0;
w21 = 0;
```

```
%select time vector
t = 0:1/5000:1;
t = t(1:length(t)-1);

%calculate input to the net
y1 = sin(2*pi*500*t + pi/2);
y2 = sin(2*pi*500*t + 1.3963);

for i=1:length(t)
  %calculate output of the net
  z1(i) = y1(i) + w12(i)*y2(i);
  z2(i) = y2(i) + w21(i)*y1(i);

  %calculate weights
  w12(i+1) = w12(i) - u*z1(i)*z2(i);
  w21(i+1) = w21(i) - u*z1(i)*z2(i)*z2(i)*z2(i);
end
```

B.7 Simulation Routine for Neural Network's Rate of Learning

```
%theta.m
%purpose: to determine the rate of change of phase-shift
%between two sinewaves as the neural network learns
%method:
%multiply two sinewaves
%2*sin(A)*sin(B) = cos(A-B) - cos(A+B);
%OR
%y = 2*sin(wt)*sin(wt + theta) = cos(wt - wt - theta) - cos(wt + wt + theta)
%                              = cos(-theta) - cos(2wt + theta)
%                              = cos(theta) - cos(2wt + theta)
%filter y using a LPF at wt cutoff or less
%this gives cos(theta)
%find cosine inverse [acos()] to get theta

figure;
%normalize the inputs (if required) so that
%acos() does not give imaginary values
%z1 = z1/max(z1);
%z2 = z2/max(z2);

%multiply the two sinewaves
for i = 1:length(t)
```

```
   z(i) = z1(i) * z2(i);
   %z(i) = z(i)/2;
end

%filter the product twice to minimize the
%starting and ending transients
[b,a] = butter(2,100/2500);
z=filtfilt(b,a,z);
z=filtfilt(b,a,z);
%find cosine inverse
z = acos(z);
z = z * 180/pi;
plot(t,z);
grid;
zoom;
```

REFERENCES

1. T. Chang, T. Wong, B. Dani, Z. Ji, M. Shimanovich, and R. Caudill, "Control of Ultra-High Precision Magnetic Leadscrew Using Recurrent Neural Networks", *Proceedings of International Symposium on Intelligent Systems and Advanced Manufacturing,* SPIE - The International Society for Optical Engineering, Boston, MA, 1998, Vol. 3518-26.

2. C. Lee and S. Kim, "An Ultraprecision Stage for Alignment of Wafers in Advanced Microlithography", *Precision Engineering,* Vol. 21, 1997, pp. 113-122.

3. T. Satomi, "Studies on Aerostatic Lead Screws", *World Congress of IFToMM,* 1987, pp. 1545-1548.

4. G. Otten, T. Vries, J. Amerongen, A. Rankers, and E. Gaal, "Linear Motor Motion Control Using a Learning Feedforward Controller", *IEEE/ASME Transactions on Mechatronics,* Vol. 2, No. 3, Sept. 1997.

5. W. Kim and D. Trumper, "High-Precision Magnetic Levitation Stage for Photolithography", *Precision Engineering,* Vol. 22, 1998, pp. 66-77.

6. R. Benning, M. Hodgins, and G. Zipfel, Jr., "Active Control of Mechanical Vibrations", *Bell Labs Technical Journal,* Spring 1997.

7. G. Song, S. Schmidt, and B. Agrawal, "Experimental study of Active Vibration Suppression of Flexible Structure using Modular Control Patch", *Proceedings of IEEE Conference on Aerospace Applications,* 1998, Vol. 1, pp. 189-201.

8. T. Hooper and R. Greif, "Active Vibration Control Using Modal Techniques", *Active/Passive Vibration Control and Nonlinear Dynamics of Structures,* ASME 1997.

9. B. Awabdy, W. Shih, and D. Auslander, "Nanometer Positioning of a Linear Motion Stage Under Static Loads", *IEEE/ASME Transactions on Mechatronics,* June 1998, Vol. 3, No. 2.

10. R. LeLetty, F. Claeyssen, N. Lhermet, and P. Bouchilloux, "New Amplified Piezoelectric Actuator for Precision Positioning and Active Damping", *Proceedings of SPIE - Smart Structures and Materials, Opt Engineering,* SPIE - The International Society for Optical Engineering, 1997.

11. *Model 310 Reference Manual,* Dalanco Spry, Rochester, NY, 1993.

12. S. Haykin, *Communication Systems*, Second ed., John Wiley and Sons, New York, NY, 1989.

13. T. Chang and N. Ansari, "A Learning Controller for the Regulation and Stabilization of Flexible Structures", *Proceedings of IEEE Conference on Decision and Control,* 1993, pp. 1268-1273.

14. C. Jutten and J. Hearault, "Blind Separation of Sources, Part I: An Adaptive Algorithm Based on Neuromimetic Architecture", *Signal Processing,* 1991, pp. 1-10.

15. *Signal Processing Toolbox for use with Matlab,* The Mathworks, Inc., Natick, MA, May 1993.