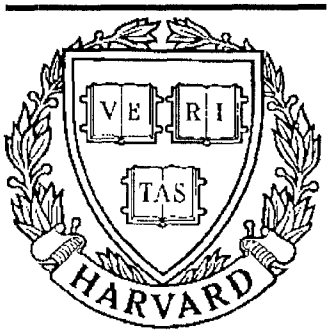
THESIS REPORT
Master's Degree

S Y S T E M S
R E S E A R C H
C E N T E R



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Design and Digital Signal Processor Implementation of a Controller for Flexible Structures

by: J.D. Bartusek
Advisor: P.S. Krishnaprasad

M.S. 90-1
Formerly TR 90-3

Design and Digital Signal Processor Implementation of a Controller for Flexible Structures

by

John D. Bartusek

Thesis submitted to the Faculty of The Graduate School

of The University Of Maryland in partial fulfillment

of the requirements for the degree of

Master of Science

1989

Advisory Committee:

Professor P. S. Krishnaprasad Advisor

Professor Mark Shayman

Associate Professor Andre L. Tits

Research Scientist Josip Lončarić

ABSTRACT

Title of Thesis: Design and Digital Signal Processor Implementation of a Real Time Controller for Flexible Structures

Name of Candidate: John D. Bartusek

Degree and Year: Master of Science, 1989

Thesis directed by: Dr. P.S. Krishnaprasad

Professor, Electrical Engineering Department

A control system for a single link light-weight flexible robot manipulator is designed and implemented on a Digital Signal Processing (DSP) chip, which controls the link dynamics via the applied motor torque. Different models of this flexible structure are studied including a nonlinear Galerkin model, linearized Galerkin model, and linear beam theory model; and these are compared through simulation to the empirical system response. A geometric Input-Output Linearization of the nonlinear system is achieved with respect to the hub angle and hub rate. Experimental system identification, performed on the flexible beam, suggests we can adequately model the flexible beam using linear theory and an optimization of the Feedback Controller is performed. Finally, this controller is implemented on a DSP chip and various aspects (programming, timing, synchronization, etc.) of DSP-based feedback control system implementation are presented.

DEDICATION

To my parents

ACKNOWLEDGEMENTS

I wish to acknowledge all those who have contributed to this thesis. I am especially indebted to my advisor, Dr. P.S. Krishnaprasad for his guidance, encouragement, and the many helpful discussions which led to the completion of this thesis. I am grateful to Dr. S. Paddock for the experience I received while participating in the NASA/USRA Advanced Design Program. Also, I would like to thank Dr. Mark Shayman, Dr. Andre Tits, and Dr. Josip Lončarić for serving as a member of my thesis advisory committee.

I wish to acknowledge the help of my friends and colleagues who also contributed. In particular, Dr. Josip Lončarić, David MacEnany, John Reilly, Serafin Rodriguez, Anthony Teolis, and Li-Sheng Wang were especially helpful. Thanks are due to Ms. Tien-Chun Wang for the drawing of the system hardware.

This research was supported in part by the National Science Foundation's Engineering Research Centers Program: NSFD CDR 8803012, the Air Force of Scientific Research under AFOSR-URI grant (AFOSR-87-0073,D) and by the Universities Space Research Association Advanced Design Program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER
ONE

Introduction

This thesis considers the problem of controlling a light-weight single link robot manipulator arm which sacrifices rigidity of the link for reduced mass and bulk. The increased flexibility in the robot manipulator link causes the resonant frequencies of the link to shift to lower frequencies that create undesirable vibrational effects in a robot manipulator. The goal of this thesis is to design a feedback control system which provides the robot arm with the ability to move in a swift and controlled manner while minimizing these vibrations. If the goal were to merely servo the base of the arm as fast as possible, without any regard to the flexible structure attached, the resonant modes of the beam would be excited. Depending on the amount of damping, it could then take a very long time for the tip position to settle to the reference position. Without a controlled system which takes into account the flexibility in the beam, we can not accurately position the endpoint of the robot arm in a reasonable amount of time.

Analysis of light-weight flexible structures is necessary if we are to understand the dynamics of space platforms which have controlled flexible appendages.

A controlled flexible structure could be a long and slender robot arm (with inherent flexibility) working in the NASA Space Shuttle bay or possibly an antenna system which is attached to a satellite undergoing a change of orientation. In both cases, we have a flexible structure extending radially from a rigid body which is subject to a driving force and/or torque. The only difference is whether, for control purposes, we are interested primarily in positioning the rigid body itself or the distal end of the flexible structure. Furthermore, control of such structures continues to be an active area of research in the literature.

Before we are able to control a flexible link manipulator, we need to determine the characteristics of the system and we will see that the resonant frequencies only partially describe the dynamics involved. The beam dynamics are described by the Euler Bernoulli beam equations, a partial differential equation whose terms depend on time and the distance along the beam. Thus the first task at hand is to identify the characteristics of the system and explicitly choose a suitable model. The models describing the dynamics that we will explore are simple linear models of finite dimension and a nonlinear model which arises from a spatial discretization (finite element approximation) of the distributed parameter system. These models will be simulated and compared to the experimental system response before designing the control system with the most appropriate model of the system. With these chosen models, we will empirically estimate or explicitly determine the model parameters so we can simulate and finally initiate a design of the feedback control system. The goal of the control system

design is to optimize the step response of the tip position to a given change in the reference command.

The last section describes the implementation of the control system which is achieved through use of a Digital Signal Processing (DSP) chip which has vastly improved processing power when compared to past experiments of this type [1] [2] using the 80286 processor on the IBM PC AT. Additionally, this thesis serves as an example of how to implement a general sampled data feedback control system on a high speed DSP chip and discusses some of the signal processing issues inherent in such a system.

CHAPTER
TWO

Description of Experiment and Hardware

2.1 Hardware Description of the Flexible Beam Experiment

The experimental hardware in figure 2.1 consists of a single-link robot manipulator with link flexiblility, an IBM PC AT computer with DSP board, three sensors and some additional electronics. The flexible link robot consists of a brushless DC torque motor, optical encoder collocated to measure shaft position, tachometer to measure shaft rate and a non-collocated accelerometer placed at the distal end of the flexible beam. The flexible beam is a one meter length of aluminum which is mounted to the hub of the motor and serves to model a full size flexible robot arm or space craft antennae system. Throughout this thesis, we will view the experiment as modelling a robot arm although the hardware could be thought to model other dynamical systems.

The electronics which we have installed to measure, process, and control the flexible robot consist of:

1. IBM PC AT Computer

2. Communication, Automation, and Control Inc. DSP board with AT&T

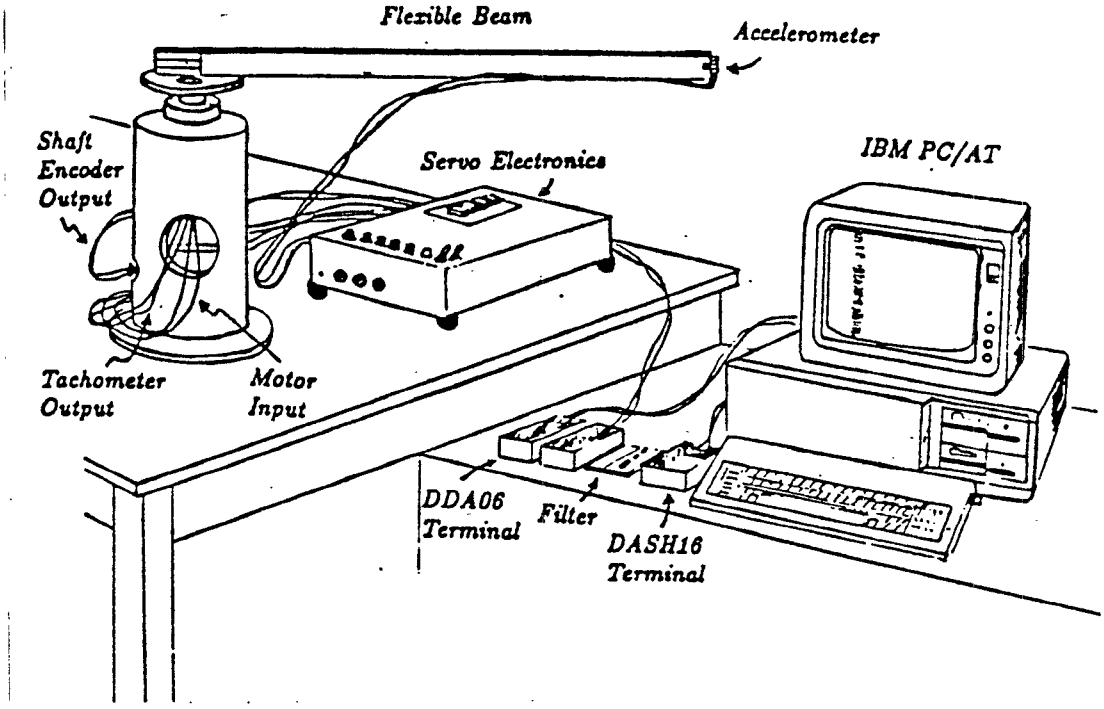


Figure 2.1: Experimental Hardware

DSP32 chip

3. Metrabyte DDA06 - Digital to Analog Converter (DAC)

4. Metrabyte DASH 16 - Analog to Digital Converter (ADC)

5. Sensors

   - Accelerometer
   - Tachometer
   - Optical Encoder

6. Analog Prefilter

7. Power Amplifier

Although we do not include here a tip position sensor, these electronics comprise the digital control system hardware which, as we will later see, will control the tip position of the flexible link manipulator through use of a *state observer*. This tip position state is available for feedback so this experiment is related to that of Schmitz [3] which made use of an optical tip position sensor for feedback.

The DDA06 board has in addition to the D/A output function to the motor, three digital I/O ports (with word size of 8); two of which are used to input the 12 bit word from the optical shaft encoder to the computer.

2.2 Digital Controller Processor

The DSP board by Communications, Automation, & Control Inc. is a low cost plug in board which uses AT&T's DSP32 floating point processor and also has I/O consisting of serial and parallel ports as well as a Codec telephone sampler. This DSP chip peaks at 8 MFLOPS. The DSP chip has on-board memory of 4 Kbytes and 56 Kbytes additional off chip memory which is directly addressable and located in four memory chips adjacent to the DSP chip on the board. For a more detailed description of the DSP chip as well as examples of DSP programming, see chapter 5 and the appendix.

The DSP chip communicates to the IBM PC AT via the parallel port of

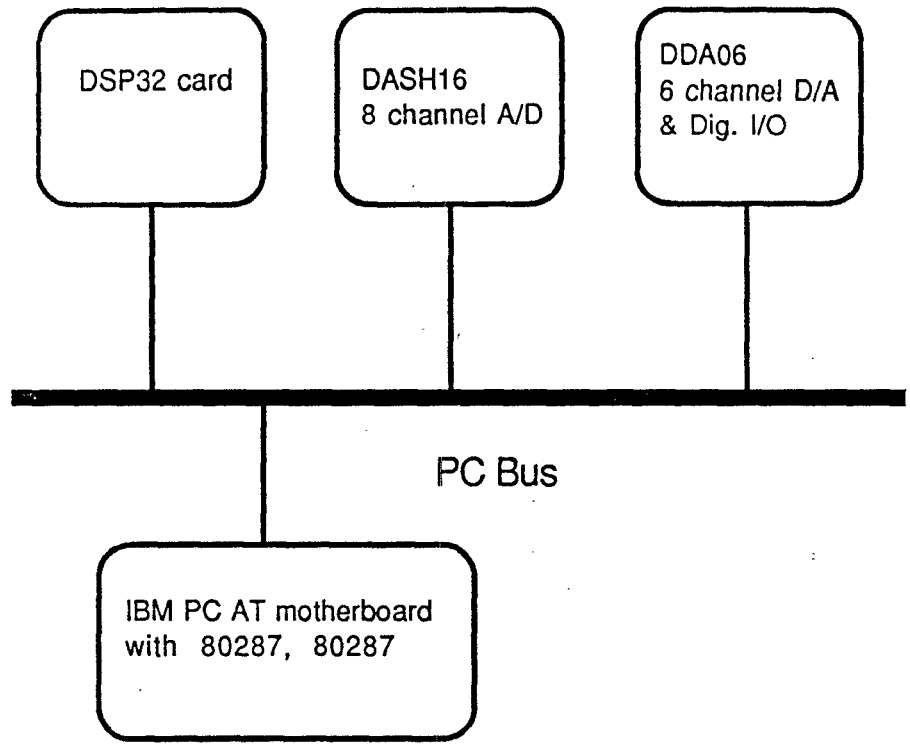


Figure 2.2: Bus Diagram

the chip through the PC bus. This is a 16 bit bus with a clock rate of 6 Mhz. Also located on the PC bus is the ADC and DAC boards which communicate to the IBM PC AT. The tachometer and accelerometer sensors are sampled by the DASH16 ADC board operating under computer control by the IBM processor. Timing and synchronization are controlled by the IBM processor which uses one of the clocks on the DASH16 board.

2.3 Sampling Rate Selection and Prefilter Design

2.3.1 Sampling Rate Selection

The sampling rate of the digital controller must be adequately high to accurately sense the majority of signals that are present in our system. A rule of thumb [4] which gives good performance is to have approximately twenty samples for the highest mode that is present in the system, ie. the desired sampling rate f_s is determined by the equation

$$f_s = 20 \times f_{max}. \tag{2.3.1}$$

Since our system is an infinite dimensional system, we must assume that all modes above a given frequency are negligible and therefore assume a fixed number of flexible modes. As we will see in the next chapter, the first three modes or vibration frequencies of this system are the dominant ones and they are approximately at $9Hz$, $19Hz$, and $49Hz$. To fully model all these modes would require an eigth order linear model since the motor accounts for a second order system. Since we want the order of our observer to be of reasonable size, we will try to limit the order of our modelled system to six by neglecting the mode at $49Hz$. This is reasonable for our hardware but on the other hand, if we were to try to model the mode at $49Hz$, according to the above rule, we would need to sample the sensor data at $20 \times 49 \approx 1KHz$. Additionally, the DSP processor would be required to compute the estimate of eight states in less time. We could build this eigth order controller but the goal here was to build

a more practical low order controller.

If we attempt to model our system as a sixth order model, we will be able to capture the dynamics of the motor as well as retain the first two modes of the flexible beam. Since the second flexible mode is approximately 20 Hz, we would thus like to sample at 400 Hz, and the feedback controller will be designed and implemented accordingly. In addition to this chosen sampling rate, another version of the digital controller will be designed and implemented with a sampling rate of 200 Hz to determine if there is any degradation in the performance at the lower sampling rate.

2.3.2 Prefilter Design

To prevent the aliasing of the sampled sensor data, analog prefilters were placed between the two analog sensor outputs and the A/D input board. A second order low pass butterworth filter ($\zeta = .71$) with transfer function

$$G_f(s) = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2} \tag{2.3.2}$$

was realized with a LC 1458 dual op amp, some resistors and capacitors (see Table 2.1) to achieve the desired cut off frequency of the filter. The bandwidth of the analog prefilter was selected according to the following rule [4] which depends on the sampling rate h.

$$\omega_c h \approx 0.5 \text{ to } 1 \tag{2.3.3}$$

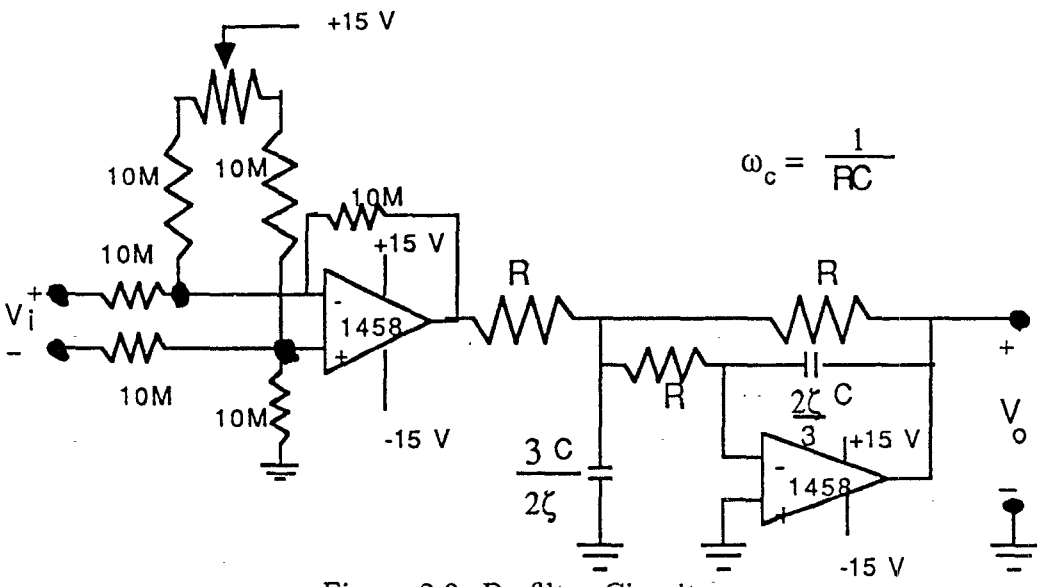


Figure 2.3: Prefilter Circuit

R	$1.47 \times 10^5 \Omega$
C_1	$0.08 \mu F$
C_2	$0.02 \mu F$

Table 2.1: Prefilter Circuit Components

According to the above rule, with a sampling rate of either 200Hz or 400Hz , the cutoff frequency $f_c = 40$ would be adequate but we select the cutoff frequency

$$f_c = 27.2Hz. \tag{2.3.4}$$

to provide more attenuation of the mode at 49 Hz while retaining the first two modes at 8.2 and 19 Hz. The circuit for the analog prefilter is shown if figure 2.3 with the component values given in table 2.1.

CHAPTER
THREE

Modeling and System Identification

The flexible link manipulator is modeled as a flexible beam with a rigid body attached at the base end which represents the hub and motor and the input to the system is the motor torque. The Euler Bernoulli beam equations governing the flexible beam are arrived at through Hamilton's principle. Finally, the open loop transfer functions are found by taking the Laplace transform of the Euler Bernoulli beam equations. We will then recognize some important properties of these transfer functions which will aid us in the experimental determination of the transfer function parameters. The *nonminimum phase* properity as well as the *pole zero alternation* properity will give us some additional constraints on the numerator and denominator polynomials of our transfer function estimate. Our estimate of the transfer function will be obtained by finding the parameters, under these constraints, which best matches both the magnitude and phase of the transfer function estimate Bode plot to that of the empirically determined frequency response.

3.1 Hamilton's Principle

At any instant in time, the flexible beam system is described by the state of its n generalized coordinate vectors within the configuration space. As time goes on, the state of the system changes and this state traces a curve which is known as the path of the system. The virtual displacement δ is defined as an infinitesimal and arbitrary change in the actual motion or path of a system which satisfies any constraints that are placed on the motion between times t_1 and t_2.

Unlike the symbol d which designates differentials, the virtual displacement has no time change associated with it, but the operations concerning δ obey those of elementary calculus.

The *Lagrangian* $\mathcal{L}$ is the difference between the kinetic energy and potential energies

$$\mathcal{L} = T - V.$$

The motion of the system from time t_1 to time t_2 is such that the line integral

$$\int_{t_1}^{t_2} \mathcal{L} dt \tag{3.1.1}$$

has a stationary value for the correct or actual path of motion so Hamilton's principle is summarized by saying that the motion is such that the variation of the line integral 3.1.1 for fixed t_1 and t_2 is zero:

$$\delta \int_{t_1}^{t_2} \mathcal{L} dt = 0 \tag{3.1.2}$$

Hamilton's principle is invariant with respect to coordinate changes, and the conditions which render the integral stationary lead to the equations of motion of the system.

3.2 Euler Bernoulli Beam Equation

The Euler Bernoulli Beam equations which describe the dynamics of the beam are derived using Hamilton's Principle and are outlined here. For the full derivation see [5].

The total kinetic energy of the hub, beam of length L, and tip mass m_t is given by

$$T_k = I_H\dot{\theta}^2 + \int_0^L (\frac{\partial w(x,t)}{\partial t} + x\dot{\theta})^2 \rho dx + m_t(\frac{\partial w(x,t)}{\partial t} + x\dot{\theta})^2|_{x=L} \qquad (3.2.3)$$

where I_H is the hub inertia and $w(x,t)$ is a function of length along the beam x, at time t which represents the perpendicular distance of the beam from a reference line fixed to the hub. The strain energy is

$$V_s = \frac{1}{2}\int_0^L EI\left(\frac{\partial^2 w(x,t)}{\partial x^2}\right)^2 dx \qquad (3.2.4)$$

where E is Young's Modulus and I is the second moment of area of the beam's cross section. The motor torque is an external torque and is

$$V_{ext} = -T\theta$$

where T is the input torque. The Lagrangian $\mathcal{L}$ for this system is

$$\mathcal{L} = T_k - V_s - V_{ext}$$

and if we apply Hamilton's Principle , i.e, set,

$$\delta \int_{t_1}^{t_2} (T_k - V_s - V_{ext}) dt = 0 \tag{3.2.5}$$

we can derive the the partial differential equations describing the beam dynamics. Let ρ be a constant mass density of the beam material and after some manipulations using integration by parts we can arrive at the Euler Bernoulli beam equation

$$EI\frac{\partial^4 w(x,t)}{\partial x^4} + \rho\frac{\partial^2 w(x,t)}{\partial t^2} = -\rho x\ddot{\theta} \tag{3.2.6}$$

with the boundary conditions

$$EI\frac{\partial^2 w(x,t)}{\partial x^2}|_{x=0} + T - I_H\ddot{\theta} = 0 \tag{3.2.7}$$

$$w(0,t) = 0 \tag{3.2.8}$$

$$EI\frac{\partial^2 w(x,t)}{\partial x^2}|_{x=L} = 0 \tag{3.2.9}$$

$$EI\frac{\partial^3 w}{\partial x^3}|_{x=L} = m_t(\frac{\partial^2 w}{\partial t^2} + x\ddot{\theta})|_{x=L}. \tag{3.2.10}$$

If we define

$$y(x,t) = w(x,t) + x\theta(t)$$

the equations can be recast into the form

$$EI\frac{\partial^4 y(x,t)}{\partial x^4} + \rho\frac{\partial^2 y(x,t)}{\partial t^2} = 0 \tag{3.2.11}$$

with the boundary conditions

$$EI\frac{\partial^2 y(x,t)}{\partial x^2}|_{x=0} + T - I_H\ddot{\theta} = 0 \tag{3.2.12}$$

$$y(0,t) = 0 \tag{3.2.13}$$

$$EI\frac{\partial^2 y(x,t)}{\partial x^2}|_{x=L} = 0 \tag{3.2.14}$$

$$EI\frac{\partial^3 y}{\partial x^3}|_{x=L} = m_t\frac{\partial^2 y}{\partial t^2}|_{x=L}. \tag{3.2.15}$$

If we allow a beam with nonuniform mass distribution $m(x)$ or nonuniform inertia $I(x)$, then the Euler Bernoulli equation becomes

$$\frac{\partial^2}{\partial x^2}\left[EI(x)\frac{\partial^2 y(x,t)}{\partial x^2}\right] = -m(x)\frac{\partial^2 y(x,t)}{\partial t^2}. \tag{3.2.16}$$

3.3 Open Loop Transfer Functions

The open loop transfer functions for the flexible beam and rotating hub system are found by taking the Laplace transform of the Euler Bernoulli beam equations solving the equations as in [6] [5] [3]. We then define complex number λ which is related to s by the equations

$$\beta^4 = -\frac{\rho}{EI}s^2$$

$$\lambda = \beta L$$

If $Y(x,s)$ is the Laplace transform of $y(x,t)$, then the solution to the system is given by

$$Y(x,s) = A\sin\beta x + B\sinh\beta x + C\cos\beta x + D\cosh\beta x$$

We seek transfer functions for the three sensors we have as well as the tip position so we can estimate the tip position. The exact open loop transfer functions ,as

derived by Schmitz [3] for a similar experiment, are given by the expressions.

$$\theta(s) = \frac{\partial Y(0,s)}{\partial x} \quad \text{Hub Angle} \tag{3.3.17}$$

$$\Omega(s) = s\frac{\partial Y(0,s)}{\partial x} \quad \text{Hub Angular Rate} \tag{3.3.18}$$

$$\alpha_{tip}(s) = s^2 Y(L,s) \quad \text{Tip Acceleration} \tag{3.3.19}$$

$$y_{tip}(s) = Y(L,s) \quad \text{Tip Position} \tag{3.3.20}$$

and we can define following numerator polynomials $N()$ and denominator polynomials $N()$ for the transfer functions

$$\begin{bmatrix} \theta(s) \\ \Omega(s) \\ \alpha_{tip}(s) \\ y_{tip}(s) \end{bmatrix} = \begin{bmatrix} N_\theta(s) \\ N_\Omega(s) \\ N_{\alpha_{tip}}(s) \\ N_{y_{tip}}(s) \end{bmatrix} D(s)^{-1} \tag{3.3.21}$$

3.3.1 Hub Position Transfer Function

The hub position transfer function is of the form

$$\frac{\theta(s)}{T(s)} = \frac{1}{I_T s^2} \prod_{i=1}^{i=\infty} \frac{(1+\frac{s^2}{\Omega_i^2})}{(1+\frac{s^2}{\omega_i^2})} \tag{3.3.22}$$

where the total inertia of the system I_t is given by

$$I_T = I_B + I_H + m_t L^2.$$

This transfer function has an interesting property that between any two successive roots of the numerator N_θ there is always a unique root of the denominator. Thus the poles and zeroes alternate along the $j\omega$ axis and we will

call this phenomenon the pole zero alternation property. Also it is apparent that at low frequencies the beam behaves as $\frac{1}{I_T s^2}$. This fact will help us to estimate the transfer function from the empirically determined frequency response using prior knowledge on the theory of beams.

Constraining our transfer function to a sixth order model and observing the pole zero alternation property, we still need to choose the parameters of the transfer function

$$\frac{k}{I_t s(s+a)} \prod_{i=1}^{2} \frac{\left(\frac{s^2}{\Omega_i^2} + 2\zeta_i \frac{s}{\Omega_i} + 1\right)}{\left(\frac{s^2}{\omega_i^2} + 2\xi_i \frac{s}{\omega_i} + 1\right)} \tag{3.3.23}$$

where $\omega_{i-1} < \Omega_i < \omega_i$ for $i = 1, 2$ and $\zeta \geq 0$.

3.3.2 Tip Position Transfer Function

For the tip position, the numerator of the transfer function has as its zeros the roots of the equation [3]

$$N_{y_t}(\lambda) = \frac{-2}{I_T \beta^2}(\sin\lambda + \sinh\lambda)$$

and the transfer function is given by

$$\frac{y_t(s)}{T(s)} = \frac{L}{I_T s^2} \prod_{i=1}^{i=\infty} \frac{(1 - \frac{s^2}{\Omega_i^2})}{(1 + \frac{s^2}{\omega_i^2})} \tag{3.3.24}$$

For this transfer function we know that the zeroes are all on the real axis having symmetric pairs about the origin. Since there exists zeroes in the positive right half plane, this transfer function is *nonminimum phase.*

A second property of this transfer function is that the zeros don not depend on the tip mass since they correspond to input frequencies in which the tip does

not move. A mass place at the tip while responding to the sinusoidal input at the frequency of the zeroes will have no effect on the system.

Again, by constraining the transfer function to a sixth order model and observing the pole zero alternation property, we still need to choose the parameters of the transfer function

$$\frac{k}{I_t s(s+a)} \prod_{i=1}^{2} \frac{\left(\frac{s^2}{\Omega_i^2} - 2\zeta_i \frac{s}{\Omega_i} + 1\right)}{\left(\frac{s^2}{\omega_i^2} + 2\xi_i \frac{s}{\omega_i} + 1\right)} \tag{3.3.25}$$

where $\omega_{i-1} < \Omega_i < \omega_i$ for $i = 1, 2$ and $\zeta \geq 0$.

3.3.3 Nonminimum Phase Property

Because of the nonminimum phase property of the flexible beam, we know that if the system is driven with a step input, that the tip response can first move in the opposite direction to the hub response after an initial delay of quasi stationary motion [7]. The amount of delay in the system depends on the placement of the zero which is closest to the origin. Furthermore, we should expect the tip accelerometer to first move in the negative direction and this is precisely what was experimentally observed.

3.4 Spatial Discretization - Galerkin Model

The Euler Bernoulli Beam equation is a partial differential equation which is function of time and position on the beam and is thus a distributed parameter system. One method for solving the distributed parameter system is to approxi-

mate the continuous beam by a finite number of rigid beams or elements, which are adjoined by freely rotating pin joints connected in a lengthwise manner. At each joint is also connected a spring to provide a linear restoring force and a damping element can be included in each joint as well. Posbergh [6] has shown that as the number of elements in this approximation approaches infinity, the solution converges to that of the continuous model.

If we write the equations for this finite dimensional system, called a Galerkin model of the beam, we get a set of nonlinear equations in which the order is twice the number of elements in the finite approximation. Taking the approach of Brockett and Isidori, we could base our control system design on this nonlinear model, by first finding the feedback which linearizes this system using differential geometric methods and then using a linear feedback to control the system.

The Galerkin model with linearizing feedback requires a full state vector at all times and would require a very complex nonlinear state observer. It was felt that the equations which linearize the system would not be implementable on the digital signal processor which has somewhat limited memory, therefore, we chose to implement a linear feedback system based on the linear system open loop transfer function if we could verify that this was an acceptable model.

We intend to use the Galerkin model in simulation to show that the linear systems, both experimental and simulated, give very similar results which will justify our use of a linear system in the actual implementation. To verify this approach, we compared the openloop system responses to a pulse input for the

following three systems.

1. the Nonlinear Galerkin Model simulation,

2. the Empirical Transfer Function Estimate (ETFE) simulation,

3. and the Experimental DATA.

Before we show the results, let us develop the equations which describe the nonlinear Galerkin model.

3.4.1 Galerkin Model with 3 Rigid Bodies

We consider a flexible arm with some unknown parameters such as mass distribution along the rod, hub inertia, joint friction, rod stiffness. The flexible arm is governed by the Euler Bernoulli beam equation which is derived in [5]

$$\frac{\partial^2}{\partial x^2}\left[EI(x)\frac{\partial^2 y(x,t)}{\partial x^2}\right] = -m(x)\frac{\partial^2 y(x,t)}{\partial t^2}, \tag{3.4.26}$$

This equation is easily solved for the case of uniform mass function $m(x)$, but in the general case it is not easily solved. Part of the difficulty lies in the problem of finding the mode shapes when the mass is nonuniform.

We can make an finite approximation to the Euler Bernoulli beam equation; the beam is broken down to a series of links which consists of N rigid bodies coupled together with elastic springs at revolute joints.

The model used to approximate the continuum model will be a 3 body approximation which consists of one body for the hub and two rigid bodies forming

the flexible beam. The rigid bodies are connected by a rotary joint and coupled only with a torsional spring which depends linearly, with spring constant k_i, on the angle between the two rigid bodies. Each link has a center of mass located at some point along the i^{th} rigid body which will be denoted by ϵ_i where ϵ_i represents where the mass is placed. If the mass is placed at the joint nearest the base, then $\epsilon_i = 0$; if the mass is placed at the distal end of a link, then $\epsilon_i = 1$, thus $\epsilon_i \in [0, 1]$. Also each link has a moment of inertia which is specified by I_i about the center of mass of the i^{th} link.

Posbergh [6] has outlined the equations of motion and for N-body case which include all nonlinear centrifugal and coriolis terms and they are.

$$\begin{aligned} 0 &= \sum_{i=1}^{N} m_i \left\{ \sum_{j=1}^{i} \left(\frac{\partial f_i}{\partial \theta_j} \frac{\partial f_i}{\partial \theta_k} + \frac{\partial g_i}{\partial \theta_j} \frac{\partial g_i}{\partial \theta_k} \right) \ddot{\theta}_j \right\} \\ &+ \sum_{i=1}^{N} m_i \left\{ \sum_{j=1}^{i} \sum_{l=1}^{i} \left(\frac{\partial^2 f_i}{\partial \theta_j \partial \theta_l} \frac{\partial f_i}{\partial \theta_k} + \frac{\partial^2 g_i}{\partial \theta_j \partial \theta_l} \frac{\partial g_i}{\partial \theta_k} \right) \dot{\theta}_j \dot{\theta}_l \right\} \\ &+ I_k \ddot{\theta}_k - k_k(\theta_{k+1} - \theta_k) + k_{k-1}(\theta_k - \theta_{k-1}) \\ &+ \sum_{i=1}^{N} m_i \left\{ \ddot{x}_1 \frac{\partial f_i}{\partial \theta_k} + \ddot{y}_1 \frac{\partial g_i}{\partial \theta_k} \right\}. \end{aligned} \tag{3.4.27}$$

where f_i and g_i are defined as b

$$\begin{aligned} f_i(\theta_i, \ldots, \theta_1) &= \sum_{j=1}^{i} r_j \epsilon_j \cos(\theta_j) + r_{j-1}(1 - \epsilon_{j-1}) \cos(\theta_{j-1}) \\ g_i(\theta_i, \ldots, \theta_1) &= \sum_{j=1}^{i} r_j \epsilon_j \sin(\theta_j) + r_{j-1}(1 - \epsilon_{j-1}) \sin(\theta_{j-1}), \end{aligned} \tag{3.4.28}$$

Here k_i is found by noting that in the limit as $N \to \infty$ we have that, $k_i r_i \to EI$, $m_i/r_i \to \rho$ while $\sum_i r_i = L$ and $\sum_i m_i = m_{tot}$. Let us also define the notation, $r_i = l = L/2$ for our 3 body approximation and this gives us a relation

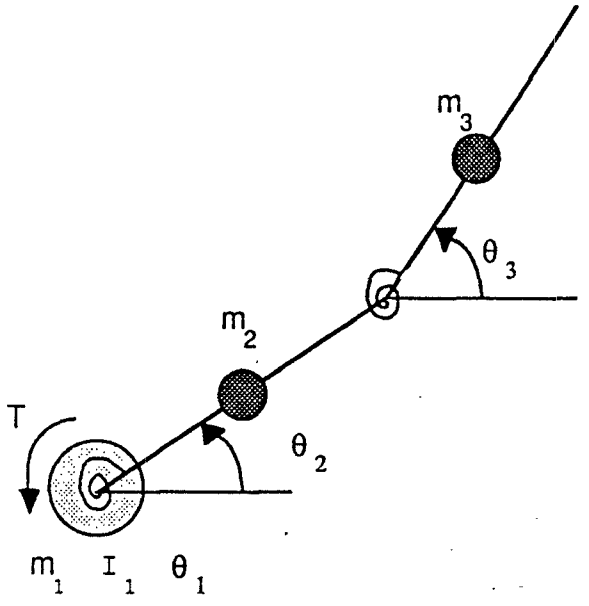


Figure 3.1: Finite Element Approximation

for k_i,

$$k_i = \frac{EI}{r_i} = \frac{EI}{l}.$$

Using the symbolic mathematics tool Macsyma [8], these equations reduce for the case when $N = 3$ to .

$$\begin{aligned} T_{motor} &= I_1\ddot{\theta}_1 - k_1(\theta_2 - \theta_1) \\ 0 &= (I_2 + l\epsilon_2^2 m_2 + l m_3)\ddot{\theta}_2 + l\epsilon_3 m_3 \cos(\theta_3 - \theta_2)\ddot{\theta}_3 \\ &\quad -l\epsilon_3 m_3 \sin(\theta_3 - \theta 2)\dot{\theta}_3^2 - k_2(\theta_3 - \theta_2) + k_1(\theta_2 - \theta_1) \\ 0 &= l\epsilon_3 m_3 \cos(\theta_3 - \theta_2)\ddot{\theta}_2 + l(\epsilon_3^2 m_3 + I_3)\ddot{\theta}_3 \\ &\quad +l\epsilon_3 m_3 \sin(\theta_3 - \theta_2)(\dot{\theta}_2)^2 + k_2(\theta_3 - \theta_2) \end{aligned}$$

The equations of motion for the case when $N = 3$ of the rigid body approximation of the beam and hub system can be written in a vector notation as in

[9] where Craig applies this equation to a robotic manipulator.

$$T = M(\theta_1, \theta_2, \theta_3) \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} + V(\theta_1, \theta_2, \theta_3, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3) + K \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \tag{3.4.29}$$

where we have the so called mass matrix,

$$M(\Theta) = \begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2 + m_3 l^2 + \epsilon_2^2 m_2 l^2 & l^2 \epsilon_3 m_3 \cos(\theta_3 - \theta_2) \\ 0 & l^2 \epsilon_3 m_3 \cos(\theta_3 - \theta_2) & I_3 + \epsilon_3^2 m_3 l^2 \end{bmatrix}$$

the stiffness matrix,

$$K = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix}$$

and the torque vector,

$$T = \begin{bmatrix} T_{motor} \\ 0 \\ 0 \end{bmatrix}.$$

If we define the orientation vector Θ as

$$\Theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix},$$

then the second term of the above equation can be written as a matrix multiplied by a vector, as shown in [9]. This allows us to represent the second term of the

right hand side of 3.4.29 as the product of a matrix and a joint angular velocity vector,

$$V(\Theta, \dot{\Theta}) = V_m(\Theta, \dot{\Theta})\dot{\Theta},$$

where the subscript m stands for matrix. This term $V_m(\Theta, \dot{\Theta})$ represents the centrifugal and coriolis force terms which are fully represented in this model. This matrix then takes the following form:

$$V_m(\Theta, \dot{\Theta}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -l^2\epsilon_3 m_3 \sin(\theta_3 - \theta_2)\dot{\theta}_3 \\ 0 & l^2\epsilon_3 m_3 \sin(\theta_3 - \theta_2)\dot{\theta}_2 & 0 \end{bmatrix} \tag{3.4.30}$$

Thus we have

$$T = M(\Theta)\begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} + V_m(\Theta, \dot{\Theta})\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} + K\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}. \tag{3.4.31}$$

describing the Galerkin model dynamics.

3.4.2 State Space Equations

It is desired to reformulate the equations of motion so they are in the form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x}, u). \tag{3.4.32}$$

If we rewrite 3.4.29 as:

$$M(\Theta)\ddot{\Theta} = -V(\Theta, \dot{\Theta}) - K\Theta + T \tag{3.4.33}$$

and then since the mass matrix is positive definite,

$$\ddot{\Theta} = M(\Theta)^{-1}\left(-V(\Theta,\dot{\Theta}) - K\Theta + T\right). \tag{3.4.34}$$

Then if we define the state vector as:

$$\mathbf{x} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{3.4.35}$$

The nonlinear state equations become

$$\begin{aligned}
\dot{\theta}_1 &= \omega_1 \\
\dot{\theta}_2 &= \omega_2 \\
\dot{\theta}_3 &= \omega_3 \\
\dot{\omega}_1 &= \frac{k_1}{I_1}(\theta_2 - \theta_1) + \frac{T_q}{I_1} \\
\dot{\omega}_2 &= \Big\{\epsilon_3 l^2 m_3 \cos(\theta_3 - \theta_2)(\epsilon_3 l^2 m_3 \omega_2^2 \sin(\theta_3 - \theta_2) + k_2(\theta_3 - \theta_2)) \\
&\quad + (\epsilon_3 l^2 m_3 + I_3)(\epsilon_3 l^2 m_3 \omega_3^2 \sin(\theta_3 - \theta_2) + k_2(\theta_3 - \theta_2) - k_1(\theta_2 - \theta_1))\Big\}/\Delta(\mathbf{x}) \\
\dot{\omega}_3 &= \Big\{-\epsilon_3 l^2 m_3 \cos(\theta_3 - \theta_2)(\epsilon_3 l^2 m_3 \omega_3^2 \sin(\theta_3 - \theta_2) + k_2(\theta_3 - \theta_2) - k_1(\theta_2 - \theta_1)) \\
&\quad - (l^2 m_3 + \epsilon_2^2 l^2 m_2 + I_2)(\epsilon_3 l^2 m_3 \omega_2^2 \sin(\theta_3 - \theta_2) + k_2(\theta_3 - \theta_2))\Big\}/\Delta(\mathbf{x})
\end{aligned}$$

where

$$\Delta(\mathbf{x}) = (l^2 m_3 + \epsilon_2^2 l^2 m_2 + I_2)(\epsilon_3^2 l^2 m_3 + I_3) - \epsilon_3^2 l^4 m_3^2 \cos^2(\theta_3 - \theta_2).$$

Let us define the following notation for $\mathbf{f(x)}$ and $\mathbf{g}$;

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ f_1(\theta_1, \theta_2) \\ f_2(\theta_1, \theta_2, \theta_3, \omega_2, \omega_3) \\ f_3(\theta_1, \theta_2, \theta_3, \omega_2, \omega_3) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ I_1^{-1} \\ 0 \\ 0 \end{bmatrix}. \tag{3.4.36}$$

3.4.3 Output Equations

The three outputs connected to the system are the optical shaft encoder, shaft tachometer, and tip accelerometer. The equations for the encoder and the tachometer outputs are just the states θ_1 and $\dot{\theta}_1$ respectively.

Tip Accelerometer

To find the tip accelerometer output equation, we first write the equation for the forward kinematics of the 3 links. The tip position is related to the base coordinates by the following expressions for the x coordinate and y coordinate respectively.

$$f_3(\theta_3, \theta_2, \theta_1) = \sum_{j=1}^{3} r_j \epsilon_j \cos\theta_j + r_{j-1}(1 - \epsilon_{j-1}) \cos\theta_{j-1} \tag{3.4.37}$$

$$g_3(\theta_3, \theta_2, \theta_1) = \sum_{j=1}^{3} r_j \epsilon_j \sin\theta_j + r_{j-1}(1 - \epsilon_{j-1}) \sin\theta_{j-1} \tag{3.4.38}$$

Thus for our case, we have $r_1 = 0, r_2 = r_3 = l$ and we can assume for now that $\epsilon_j = 1$. The endpoint position vector $X \in \mathbb{R}^3$ of the tip with respect to the base is,

$$X(\theta_3, \theta_2) = \begin{bmatrix} l\cos\theta_2 + l\cos\theta_3 \\ l\sin\theta_2 + l\sin\theta_3 \\ 0 \end{bmatrix}.$$

The first derivative is,

$$\dot{X}(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = \begin{bmatrix} -l\dot{\theta}_2\sin\theta_2 - l\dot{\theta}_3\sin\theta_3 \\ l\dot{\theta}_2\cos\theta_2 + l\dot{\theta}_3\cos\theta_3 \\ 0 \end{bmatrix}$$

and the second derivative is,

$$\ddot{X}(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2) = \begin{bmatrix} -l\theta_2^2\cos\theta_2 - l\ddot{\theta}_2\sin\theta_2 - l\theta_3^2\cos\theta_3 - l\ddot{\theta}_3\sin\theta_3 \\ -l\theta_2^2\sin\theta_2 + l\ddot{\theta}_2\cos\theta_2 - l\theta_3^2\sin\theta_3 + l\ddot{\theta}_3\cos\theta_3 \\ 0 \end{bmatrix}$$

Now since the tip accelerometer only measures acceleration normal to the third link, we must form the dot product of the acceleration vector with the vector normal to the third rigid body,

$$N(\theta_3) = \begin{bmatrix} -\sin\theta_3 \\ \cos\theta_3 \\ 0 \end{bmatrix}$$

i.e.

$$a_{tip} = \ddot{X}(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2) \cdot N(\theta_3).$$

After some cancellation and use of trigonometric identities, the acceleration reduces to

$$a_{tip} = l\dot{\theta}_2^2 \sin(\theta_3 - \theta_2) + l\ddot{\theta}_2 \cos(\theta_3 - \theta_2) + l\ddot{\theta}_3. \tag{3.4.39}$$

In the equation 3.4.39, the accelerometer output is not a function of the states exclusively, since there are the terms $\ddot{\theta}_2$and $\ddot{\theta}_3$. First we can write the tip accelerometer using some of the other state variables,

$$a_{tip} = l\omega_2^2 \sin(\theta_3 - \theta_2) + l\dot{\omega}_2 \cos(\theta_3 - \theta_2) + l\dot{\omega}_3.$$

and then we notice in the state equations 3.4.36 that since the input u is only in the fourth state equation. We can substitute in $f_2()$ and $f_3()$ for $\dot{\omega}_2$ and $\dot{\omega}_3$, respectively yielding finally

$$a_{tip} = l\omega_2^2 \sin(\theta_3 - \theta_2) + lf_2(\theta_1, \theta_2, \theta_3, \omega_2, \omega_3) \cos(\theta_3 - \theta_2) + lf_3(\theta_1, \theta_2, \theta_3, \omega_2, \omega_3).$$

Here we have the output as a function of states only and not any state derivatives and now the equations of motion conform to the the general nonlinear system 3.4.32.

3.4.4 Simulation of the Nonlinear Galerkin Equations

The pulse response of the 3-body equations 3.4.36 was simulated using Simnon [10] and the output of the hub position (θ_1), rate (ω_1) and tip accelerometer is shown in Figures 3.2 through 3.4. These can be compared to both the pulse response of the linearized model simulated in the next section and the empirical

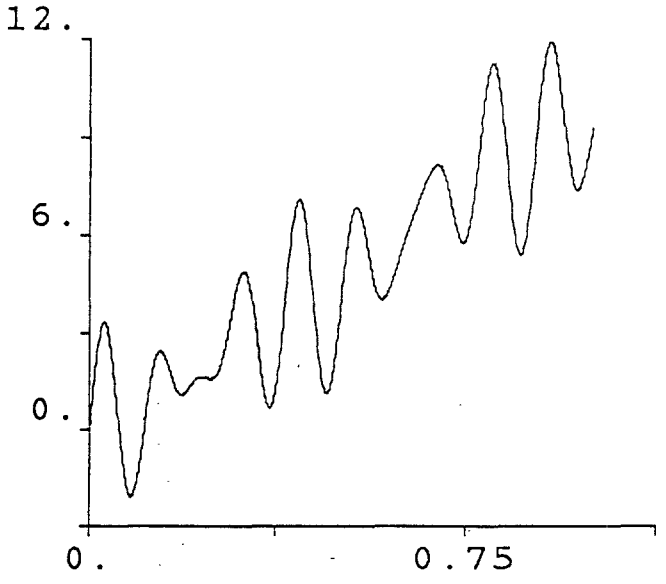


Figure 3.2: 3 Body Pulse Response of Hub Position θ_1

pulse response. In addition, the pulse response of the angles of the first and second links, θ_2 and θ_3 respectively, are simulated in Figure 3.5. In this plot, the "1" corresponds to θ_2 and the "2" corresponds to θ_3.

3.4.5 Linearization of the Galerkin Model

A model such as the nonlinear Galerkin model will includes the coriolis and centrifugal terms and is thus better suited for modeling flexible beams under large deformations. Our system, on the other hand has only small deformations caused by the high stiffness of the beam so we can safely neglect these terms and consider a linearized model.

One approach would be to linearize the nonlinear Finite Element model around a stable equilibrium point. A stable equilibrium point is simply the beam in a relaxed state which is given by $\theta_1 = \theta_2 = \theta_3$ and $\dot{\theta}_1 = \dot{\theta}_2 = \dot{\theta}_3 = 0$.

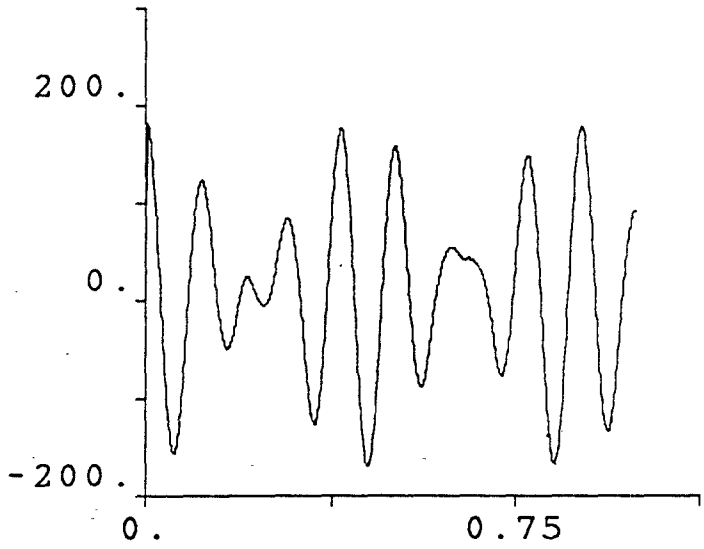


Figure 3.3: 3 Body Pulse Response of Hub Rate ω_1

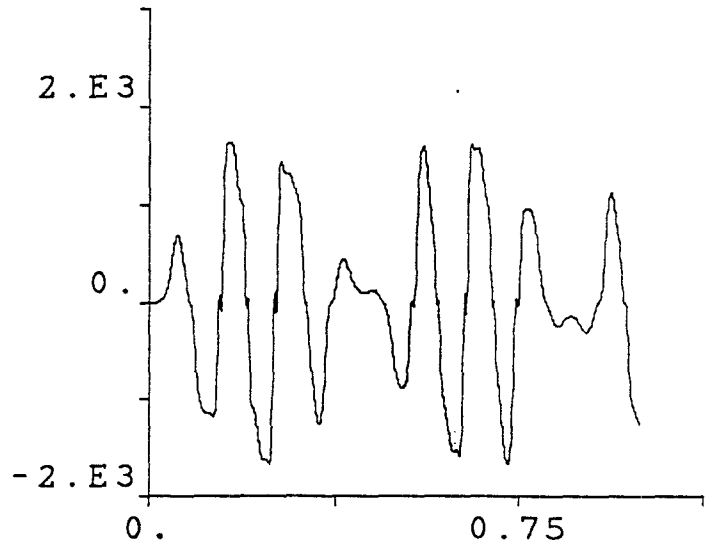


Figure 3.4: 3 Body Pulse Response of Tip Acceleration a_{tip}

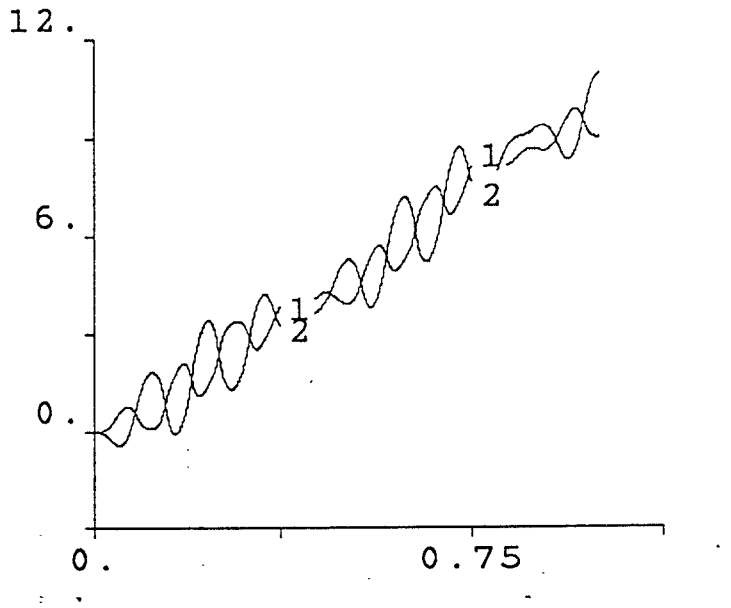


Figure 3.5: 3 Body Pulse Response of angles of link 1 and link 2,i.e. θ_2 and θ_3

Since we have a finite dimensional system, we can use the truncated Taylor expansion of the nonlinear system to determine the linearized system

$$\dot{\mathbf{x}} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{x=x_0} \mathbf{x} + \mathbf{b}u. \tag{3.4.40}$$

This linearization results in a linear system of the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}u \tag{3.4.41}$$

around the chosen equilibrium point

$$\mathbf{x_o} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.4.42}$$

where **A** is

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -k_1 & k_2 & 0 & 0 & 0 & 0 \\ \frac{k_1(\epsilon_3^2 l^2 m_3 + I_3)}{\Delta} & \frac{(-k_2-k_1)(\epsilon_3^2 l^2 m_3 + I_3) - \epsilon_3 k_2 l^2 m_3}{\Delta} & \frac{k_2(\epsilon_3^2 l^2 m_3 + I_3) + \epsilon_3 k_2 l^2 m_3}{\Delta} & 0 & 0 & 0 \\ \frac{-\epsilon_3 k_1 l^2 m_3}{\Delta} & \frac{k_2(l^2 m_3 + \epsilon_2^2 l^2 m_2 + I_2) - \epsilon(-k_2-k_1) l^2 m_3}{\Delta} & \frac{k_2(l^2 M_3 + \epsilon_2^2 m_2 + I_2) - k_3 k_2 l^2 m_3}{\Delta} & 0 & 0 & 0 \end{bmatrix}$$

where

$$\Delta = (l^2 m_3 + \epsilon_2^2 l^2 m_2 + i_2)(\epsilon_3^2 l^2 m_3 + i_3) - \epsilon_3 l^4 m_3^2$$

The hub mass and inertia are easily computed from the hubs physical dimensions. The hub has a radius $r = 11.4cm$, a height of $h = 0.87cm$ and is made of aluminum which weighs $w = 26.6kN/m^3$. The mass density of the aluminum hub is thus $\rho_{Al} = 26.6 \times 10^3/9.806 = 2.71262 \times 10^3 kg/m^3$. Since the hub is very nearly a flat cylinder, the total mass is computed $m_1 = \rho_{Al}\pi\rho^2 h = 0.8971kg$ and the inertia is computed as

$$I_1 = m_1 \frac{r^2}{2} = 5.427 \times 10^{-3} kg/m^3.$$

The mass of both the second and third link is computed from the mass density and the dimensions of the beam elements

$$m_2 = m_3 = (2712.62)(0.0030625)(0.04828)(0.5) = 0.20054.$$

The stiffness of the aluminum beam is computed by noting that the modulus of elasticity for aluminum is $E = 71.0GPa$ and the dimensions of the beam are width $w = 0.30625cm$, height $h = 4.826cm$, and length $L = 100.0cm$.

The inertia about the first y-axis joint is

$$I = \int_A x^2 dA = \rho_{Al} hw \int_0^{0.5} x^2 dx = \frac{\rho_{Al} hw}{24}$$

and is thus

$$I_2 = I_3 = \frac{(0.0030625)(0.04826)(2.7162)}{24} = 1.6705 \times 10^{-5}$$

and we have $\epsilon_1 = \epsilon_2 = \frac{1}{2}$.

With these particular parameters for the flexible beam experiment, the linearized Galerkin model has eigenvalues which correspond to the vibratory modes of the system:

$$\begin{aligned} \lambda_0 &= 0 \quad \text{(multiplicity 2)} \\ \lambda_1 &= \pm\frac{j75.20}{\sqrt{2}} \\ \lambda_2 &= \pm\frac{j170.17}{\sqrt{2}} \end{aligned}$$

from which we can compute the modal frequencies

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 8.46 \\ f_2 &= 19.15 \end{aligned}$$

using the relation $\lambda = 2\pi f$.

3.5 Empirical System Identification

3.5.1 Empirical Transfer Function Estimate

To determine the mathematical model of the open loop system and verify some of the results of Frank [1], some linear system identification procedures were done on the experimental hardware to find the frequency response of the system excited by a impulse. This methodology assumes that the system is linear.

To obtain the Bode plots of the system, the following methodology is performed on each sensor output. An impulse was approximated by applying the maximum motor torque for one sampling period (in this case, $T_s = 0.01$) to the motor which excites the system. The outputs of the tachometer and accelerometer were sampled via the analog to digital converter (ADC) board on the IBM PC. The 1024 point Discrete Fourier Transforms (DFT) of these sampled outputs were then determined using a subroutine from the DSP32 Software Library [11] on the DSP32 chip. The magnitude (converted to dB) and phase of the DFT approximate the Bode plot of the motor input to the corresponding sensor output. This approximates the transfer function of the openloop system from which we will base the control system design.

One problem is that we can not take the DFT of the encoder to obtain the Bode plot for the hub position because the DFT requires that you have a time sequence which is of finite length i.e., there exists some N such that $f(k) = 0$ for $k \geq N$. Both the tachometer and accelerometer time sequences satisfy this

condition. To obtain the transfer function for the hub position we must divide the transfer function obtained from the tachometer by the Laplace variable s.

3.5.2 Obtaining the Pole-Zero Transfer Function from the Empirical Data

Since we have data for the frequency response, we would like to obtain a transfer function $H(s)$ which approximates the empirical transfer function $\widehat{H}(s)$. The goal of this experiment is to model the infinite dimensional system by a six order finite dimensional system. In addition we will need a second order transfer function

$$H_{pf}(s) = \frac{1}{\frac{s^2}{\omega_c^2} + \frac{2\zeta s}{\omega_c} + 1} \tag{3.5.43}$$

to model the analog prefilter which is placed after the sensors to reduce aliasing. The cascade of both transfer functions results in an eighth order system. We will perform the curve fitting for both the sixth order model without the prefilter in cascade and the eight order model which includes the prefilter. A sixth order model would be preferable if it a gives reasonable estimate of the DFT.

One possible method of determining the transfer function would be to construct an optimization procedure in which the mean square error is minimized. The parameters which minimize the sum of the mean square error of the ETFE for each sensor describes the poles, zeros, and damping ratios of the system.

From Chapter 3, we have considerable prior knowledge as to the form or expected form of the open loop transfer functions. Also the alternation property

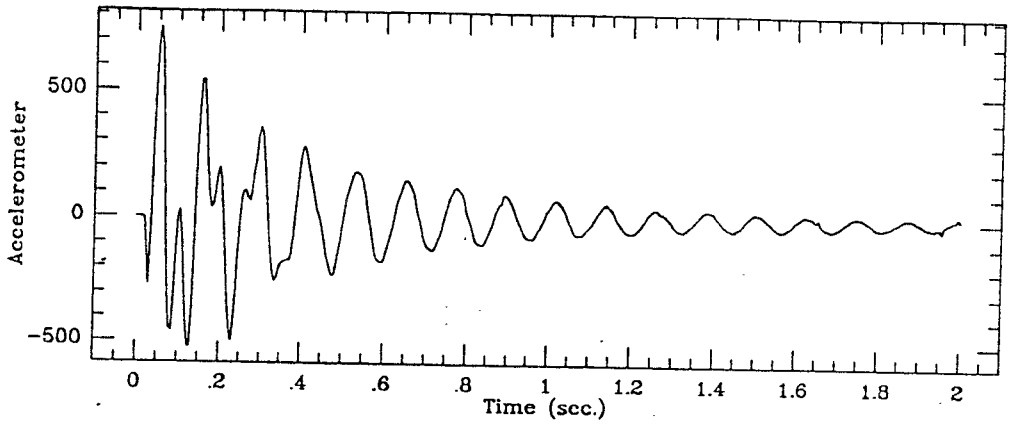


Figure 3.6: Impulse Response of Tip Accelerometer

of the poles and zeroes will be helpful in fitting the curves. Since we are constructing a low order controller, and with this knowledge, it is quite reasonable to simply construct the ETFE by hand. From the accelerometer data, we can determine the poles of the system and adjust the zeros to best match the magnitude and phase of the system. Care was taken to match the phase as much as possible. The damping ratio of each pole was adjusted by matching the slope of the ETFE phase response with that from the experimental phase response.

The estimate for the transfer function from the motor input to accelerometer is

$$H_{ac}(s) = \frac{-1.5s^2\left[\left(\frac{s}{2\pi 2.4}\right)^2 - 1\right]\left[\left(\frac{s}{2\pi 17.2}\right)^2 - \left(\frac{0.06s}{2\pi 17.2}\right) + 1\right]}{s\left[\left(\frac{s}{2\pi 0.25}\right) + 1\right]\left[\left(\frac{s}{2\pi 8.2}\right)^2 + \left(\frac{0.08s}{2\pi 8.2}\right) + 1\right]\left[\left(\frac{s}{2\pi 19.8}\right)^2 + \left(\frac{0.06s}{2\pi 19.8}\right) + 1\right]} \tag{3.5.44}$$

and the comparison of this estimate to the experimental data is given in figure 3.7. The tachometer transfer function has the same poles as the accelerometer transfer function so we only need to fit the numerator to the experimental data.

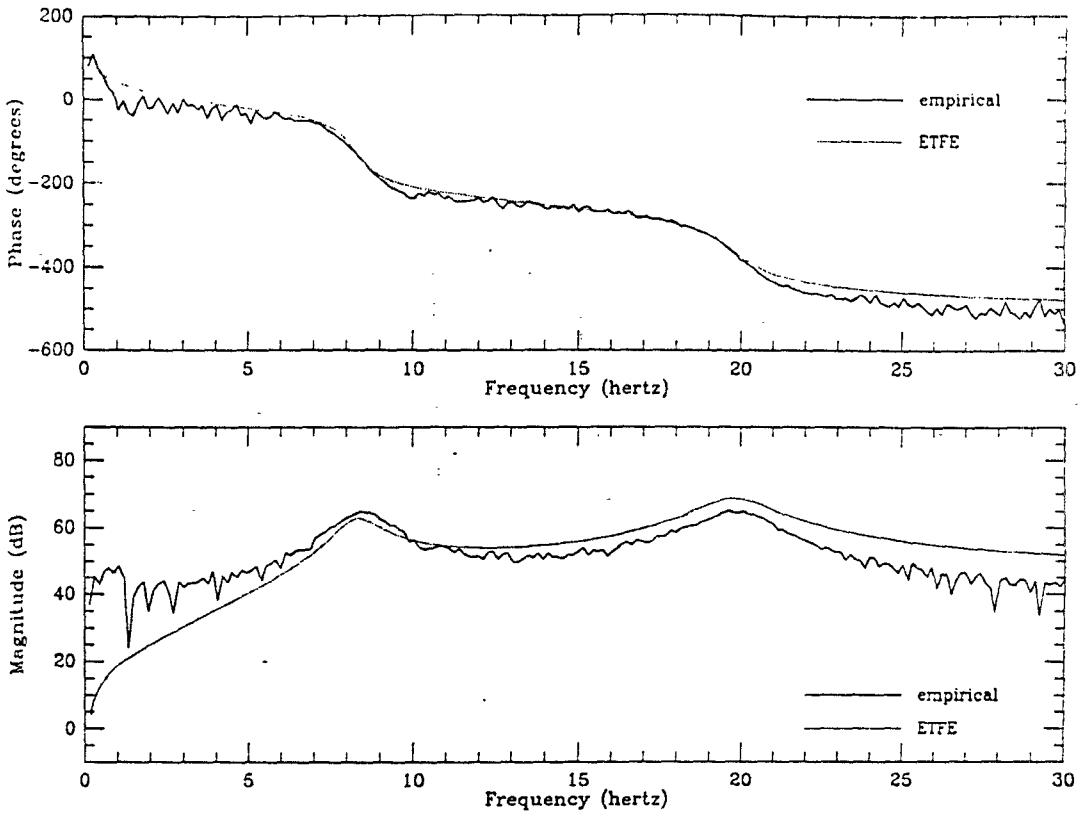


Figure 3.7: DFT of Tip Accelerometer

This yields after curve fitting 3.9, the transfer function

$$H_v(s) = \frac{130s\left[\left(\frac{s}{2\pi 2.4}\right)^2 + \left(\frac{0.16s}{2\pi 2.4}\right) + 1\right]\left[\left(\frac{s}{2\pi 17.2}\right)^2 + \left(\frac{0.06s}{2\pi 17.2}\right) + 1\right]}{s\left[\left(\frac{s}{2\pi 0.25}\right) + 1\right]\left[\left(\frac{s}{2\pi 8.2}\right)^2 + \left(\frac{0.08s}{2\pi 8.2}\right) + 1\right]\left[\left(\frac{s}{2\pi 19.8}\right)^2 + \left(\frac{0.06s}{2\pi 19.8}\right) + 1\right]}. \tag{3.5.45}$$

Since the optical encoder is collocated to the same shaft as the tachometer, the transfer function for the encoder is

$$H_p(s) = k\frac{1}{s}H_v(s), \tag{3.5.46}$$

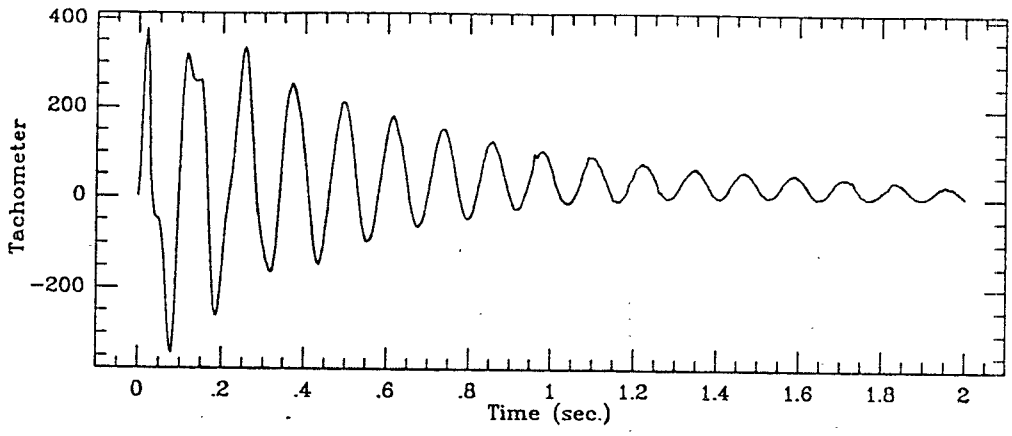


Figure 3.8: Impulse Response of Hub Angular Velocity

where k is a scale factor due to the different gains of the sensor outputs. The scale factor k was found to be 15.0, thus we have the encoder transfer function,

$$H_p(s) = \frac{130 \times 15 \left[\left(\frac{s}{2\pi 2.4}\right)^2 + \left(\frac{0.16s}{2\pi 2.4}\right) + 1\right] \left[\left(\frac{s}{2\pi 17.2}\right)^2 + \left(\frac{0.06s}{2\pi 17.2}\right) + 1\right]}{s \left[\left(\frac{s}{2\pi 0.25}\right) + 1\right] \left[\left(\frac{s}{2\pi 8.2}\right)^2 + \left(\frac{0.08s}{2\pi 8.2}\right) + 1\right] \left[\left(\frac{s}{2\pi 19.8}\right)^2 + \left(\frac{0.06s}{2\pi 19.8}\right) + 1\right]}. \tag{3.5.47}$$

The transfer function for the analog prefilter is

$$H_{pf}(s) = \frac{1}{\left[\left(\frac{s}{2\pi 27.2}\right)^2 + \left(\frac{1.5s}{2\pi 27.2}\right) + 1\right]} \tag{3.5.48}$$

although this was not included in the final model since we want to limit the order of the controller. Satisfactory results are given by the sixth order model and the curve fit does not warrant an increased order. In fact, if an eighth order model was to be approximated with the best curve fit, a superior approximation could be obtained by cascading in series the second order term for the next mode

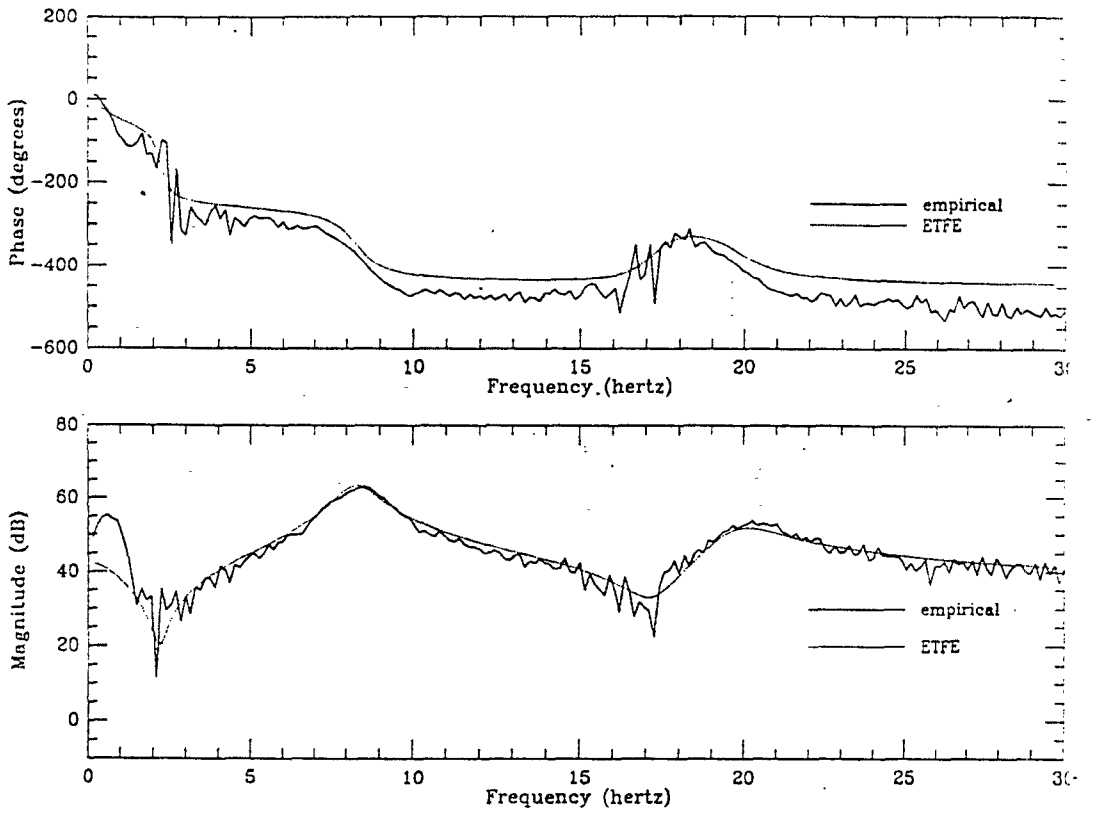


Figure 3.9: DFT of Hub Tachometer

instead of including the prefilter transfer function to our sixth order model.

3.5.3 Simulation of ETFE

The impulse response of the ETFE continuous time model is verified by simulating the pulse response of the open loop model for each of the three outputs as well as the tip position, figures 3.11 through 3.14, and and comparing the results to the empirical pulse responses.

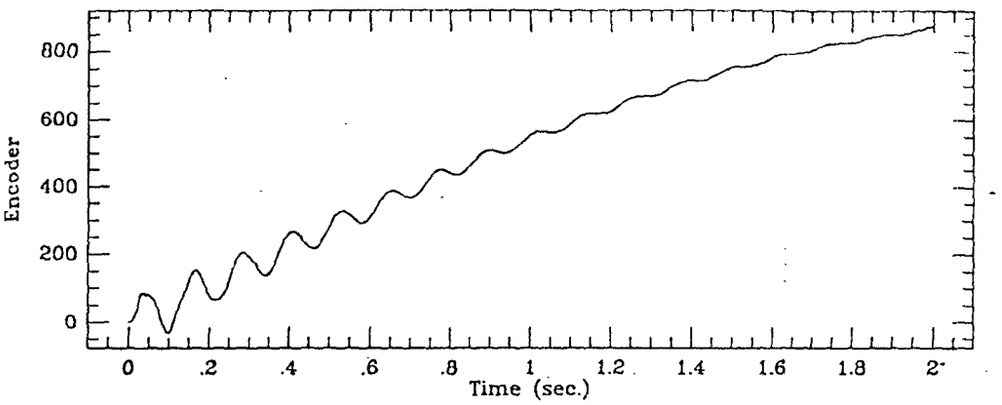


Figure 3.10: Impulse Response of Hub Position

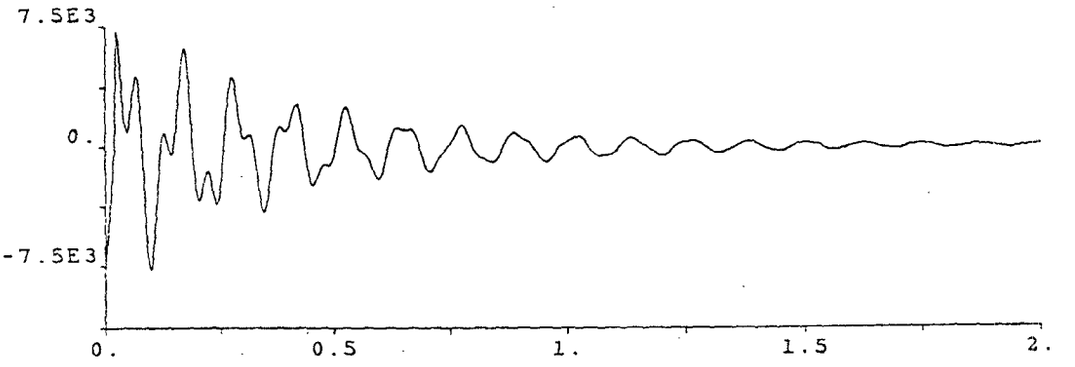


Figure 3.11: Pulse Response of Open Loop Model- Accelerometer

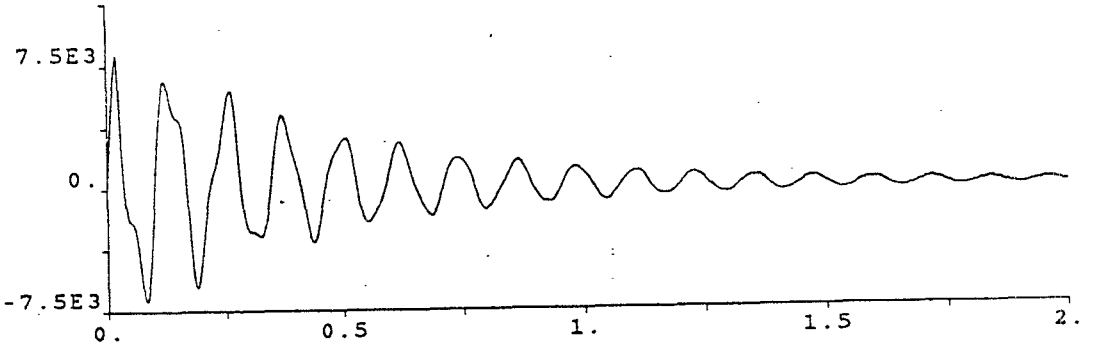


Figure 3.12: Pulse Response of Open Loop Model- Tachometer

3.6 Disturbances

3.6.1 Friction Compensation

The motor and hub assembly has a considerable amount of friction which must be modeled and compensated for. The system has Coulomb, static friction, viscous friction as well as air drag on the flat beam. Coulomb and static friction can be lumped as a single bearing friction while viscous friction is a linear function of hub rate.

$$T_{vf} = k_{vf}\dot{\theta} \tag{3.6.49}$$

Caution must be used when implementing the viscous friction compensator because the system could go unstable if we overcompensate with a higher value of

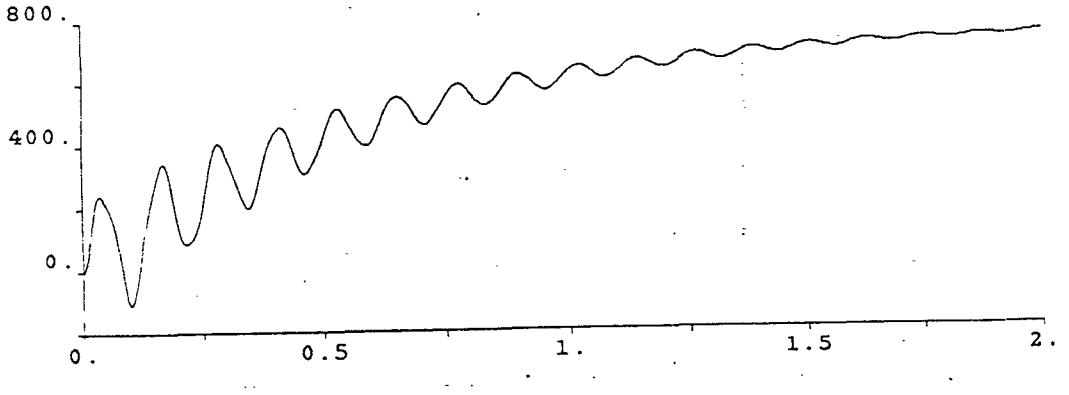


Figure 3.13: Pulse Response of Open Loop Model- Encoder

k_{vf}. Ideally, we would like to compensate the viscous friction so as to eliminate it entirely, but if the value for the coefficient is too large, the system will be unstable, so we must tolerate a small amount of viscous friction to give us a safe stability margin by slightly undercompensating. This instability can be explained because if we were to implement a high gain friction compensator, we would be supplying more energy to the system than the friction would be dissipating.

A simple method to determine the value of the coefficient is to write a program that compensates for the friction while we adjust the coefficient to be a large as possible but also requiring the motor to have a small stability margin.

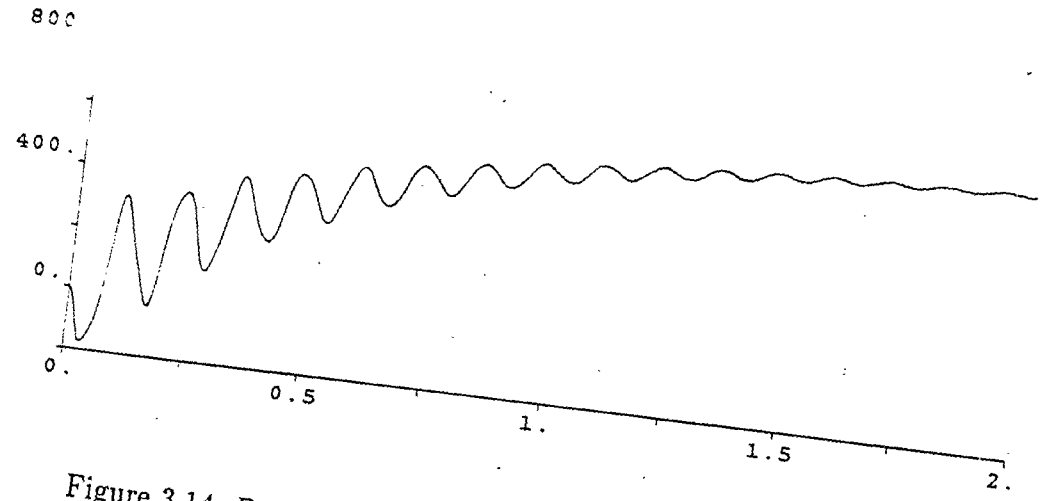


Figure 3.14: Pulse Response of Open Loop Model- Tip Position

Instability is easily observed by perturbing the system (setting the hub in motion) and observing the hub rate increases or decreases. With the flexible beam removed from the hub, the coefficient determined in this manner is $K_{vf} = -0.66$. With the compensator program running with this value and the hub set in motion, the hub rate will slowly decay to a zero after many revolutions. If we attach the flexible beam, we find that the hub rate decays much faster since air drag provides some additional damping. Although the air drag is usually thought of as a cubed term and not a linear term, we can improve the compensation by increasing k_{vf} to -1.05. Experimental testing suggests that this approximation is reasonable since we are not operating the beam at high speeds to warrant a

cubed term for the air drag compensator. With $K_{vf} = -1.05$, the hub and beam system is stable yet it behaves as very nearly a frictionless bearing at all allowable hub rates.

3.6.2 Ripple Torque Compensation

The motor has some undesirable ripple torque which we would like to eliminate so that it doesn't excite any vibratory modes of the flexible beam as we slew the robot arm. The ripple torque is seen by applying a constant small input signal to the motor ($u = 40$out of a maximum of 2000), (which is enough to cause the hub to rotate at a moderately slow rate), then the velocity is measured as the ripple torque acts on the system. For observing the ripple torque, we do *not* want to compensate for viscous friction. With a constant input and the friction acting on the system the beam will reach a constant steady state velocity. Any detectable variation in speed can then be attributed to ripple torque acting on the motor shaft.

Figure 3.15 shows that the hub rate is not constant as the motor rotates from $-\pi$ to π while figure 3.16 shows more detail. The variation in hub rate is really only detectable at very slow speeds while at high speeds there is almost basically no ripple torque visible. Also seen in these graphs is that the magnitude of the velocity variation varies as a function of hub position.

Due to the sinusoidal envelope, the ripple torque must be compensated with

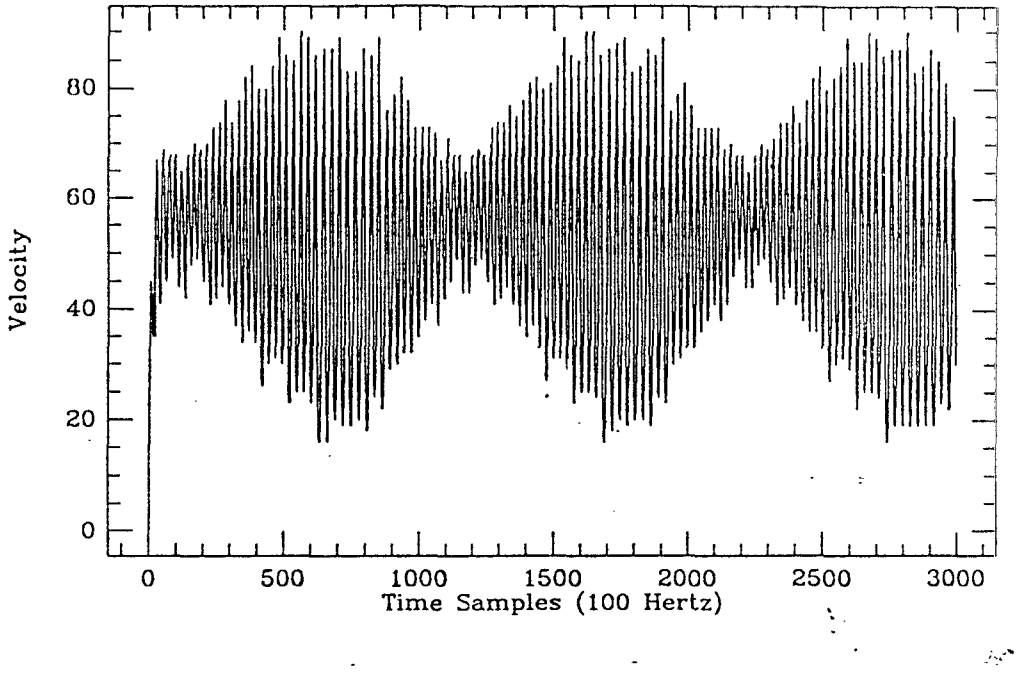


Figure 3.15: Variation of Velocity with respect to Position.

the compensating function

$$u = \alpha(1 - \beta\cos(\theta - \phi))\sin(\omega\theta - \psi) \tag{3.6.50}$$

A compensator program was implemented which applied the feedforward law with the variables listed in Table 3.1, and the compensator showed only a negligible improvement. Other parameters were tried as well with most giving the same if not worse performance.

The ripple torque, being a sinusoidal function, has a spatial frequency of 41 cycles in one revolution of the hub which is caused by the poles of the motor magnets. In each cycle we have a both a stable and an unstable position with zero ripple torque as well as the peak values of ripple torque between these

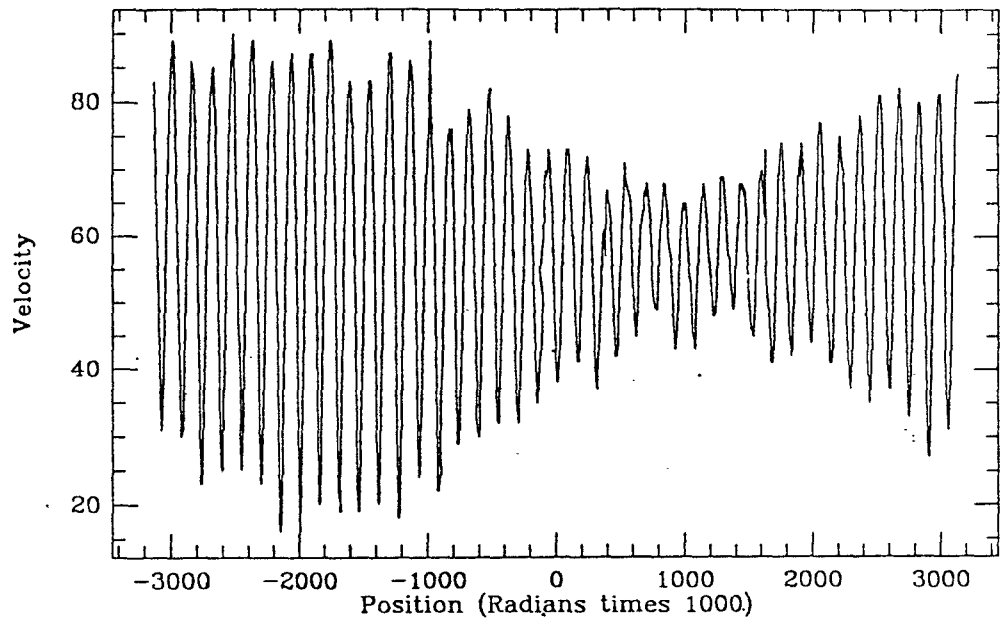


Figure 3.16: Velocity Change in One Revolution

Variable	Description	Value
α	gain	20.0
β	Variation	0.16
θ	motor position	
ϕ	envelope offset	1.0
ω	ripples per revolution	41
ψ	ripple offset	2.1

Table 3.1: Ripple Torque Variables

points. To measure the maximum ripple torque we find the maximum applied motor torque such that the hub remains static in the stable portion of one ripple cycle. Experimentally, we slowly increase the amount of applied motor torque in small increments until the shaft goes beyond a stable point and jumps into the next cycle.

The conclusion of this experiment was that a motor signal of 5 caused the hub to remain in equilibrium with the ripple torque and any signal higher caused the hub to start rotating. Thus relative to the maximum motor output of ± 2048, the ripple torque is almost negligible. Perhaps, this is why the compensation showed little improvement in measured velocity variation. In effect, the feed forward term of the ripple torque compensator takes values between ± 5 and since we are working with integers, we have a rather coarse quantization effect.

Also, the output of the encoder sensor, figure 3.17, suggests that the hub rate is considerably smoother than 3.16 suggests. Although some actual ripple torque has been measured, some of the variation in hub rate could be due to the tachometer instead of the actual hub rate since it operates via magnetic poles just as the motor does.

With the negligible improvement of the ripple torque compensator, we chose not to implement it in the final closed loop system and, in fact, no induced vibrations were noticed in the final step response of the closed loop system.

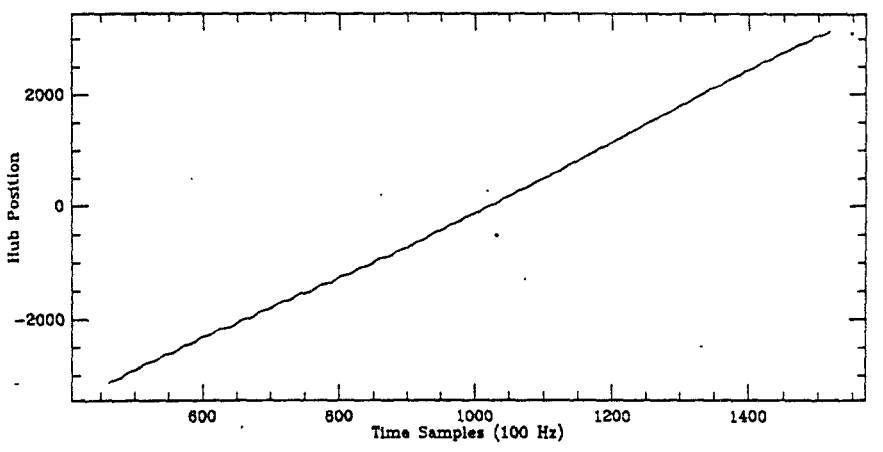


Figure 3.17: Ripple torque identification: Position data

CHAPTER
FOUR

Control

4.1 Nonlinear Control

4.1.1 Input Output Linearization

We consider the nonlinear system with one input described by

$$\dot{x} = f(x) + g(x)u \tag{4.1.1}$$

$$y = h(x) \tag{4.1.2}$$

and we would like to linearize this system with respect to the input output response. A system of N connected rigid bodies was treated by Sreenath [12] for the case of *no external torque* and Posbergh [6] gives examples of the Input Output Linearization methodology for the case of a 2-body and a 3-body chain of rigid bodies (see pages 131-142). These examples linearize the equations with respect to one of the outputs, namely, the hub rate. A hub position sensor output would also be linear as well since it is collocated to the hub rate sensor, and therefore, the I/O linearized arm could be controlled using linear feedback.

The I/O Linearization method is based on chapter 5 of Isidori [13] and linearizes the system by way of the feedback

$$u(t) = \alpha(x) + \beta(x)v(t). \tag{4.1.3}$$

This feedback requires that the complete state x vector be known in order to carry out the linearization and this emphasizes the need for a good observer to the nonlinear system.

A system is said to be linear with respect to the input and output if the relationship between y and u can be put in the form

$$y(t) = y(t, x_o) + \int_0^t k(t-\tau)u(\tau)d\tau, \tag{4.1.4}$$

where $k(t-\tau)$ is the first order kernel of the Volterra Series of 4.1.2.

Theorem 4.1 (Isidori) *A necessary and sufficient condition for the system 4.1.2 to be I/O linearizable is that the first order kernel of the Volterra Series*

$$w(t, \tau, x)$$

depend on the difference $(t-\tau)$ and does not depend on x, in a neighborhood U of the initial point x_o.

Alternatively, a necessary and sufficient condition for this kernel to be independent of x and depend only on $t-\tau$ is that the Lie derivative

$$L_g L_g^k h_j(x) \quad \text{is independent of } x \ \forall k \geq 0, 1 \leq j \leq l.$$

In general, for a specific nonlinear system, this is not satisfied and we may wish to find a feedback 4.1.3 that satisfies this requirement.

Thus the input output linearization problem can be stated as follows. Given (f, g, h) and an initial state x_o, find a neighborhood U of x_o and a pair of feedback functions α and β, defined on U such that for all $k \geq 0$ and all $1 \leq j \leq l$

$$L_{g\beta} L_{f+g\alpha} h_j(x) = \text{independent of } x \text{ on } U.$$

A convenient way to solve this problem is to arrange the functions $L_{g_i} L_f^k h_i(x)$, which characterize the Taylor series expansions of the kernels $w_j(t, 0, x)$ around $t = 0$, in a $l \times m$ matrix denoted by $T_k(x)$,

$$T_k(x) = [t_{ij}(x)] \tag{4.1.5}$$

where

$$t_{ij} = L_{g_j} L_f^k h_i(x).$$

We will attempt to linearize the nonlinear system with respect to the chosen state variables $\theta_1, \theta_2, \theta_3, \omega_1, \omega_2, \omega_3$, so temporarily define the output functions as

$$y = \begin{bmatrix} h_1(x) \\ h_2(x) \\ h_3(x) \\ h_4(x) \\ h_5(x) \\ h_6(x) \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}. \tag{4.1.6}$$

For our case, we have only one input ($m = 1$), so $T_k(x)$ is just a column vector of the form

$$T_k(x) = \begin{bmatrix} L_g L_f^k h_1(x) \\ L_g L_f^k h_2(x) \\ L_g L_f^k h_3(x) \\ L_g L_f^k h_4(x) \\ L_g L_f^k h_5(x) \\ L_g L_f^k h_6(x) \end{bmatrix} \tag{4.1.7}$$

which is defined for the formal power series

$$T(s,x) = \sum_{k=0}^{\infty} T_k(x) s^{-k-1}. \tag{4.1.8}$$

For our system,

$$\begin{aligned}
L_g h_1(x) &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} g(x) = 0 \\
L_g h_2(x) &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} g(x) = 0 \\
L_g h_3(x) &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} g(x) = 0 \\
L_g h_4(x) &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} g(x) = I_1^{-1} \\
L_g h_5(x) &= \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} g(x) = 0 \\
L_g h_6(x) &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} g(x) = 0
\end{aligned}$$

thus

$$T_o(x) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ I_1^{-1} \\ 0 \\ 0 \end{bmatrix}.$$

Theorem 4.2 (Isidori) *There exists a solution at x_o to the Input-Output Linearization Problem if and only if there exists a formal power series*

$$K(s) = \sum_{k=0}^{\infty} K_k s^{-k-1}$$

whose coefficients are $l \times m$ matrices of real numbers, and a formal power series

$$R(s,x) = R_{-1} + \sum_{k=0}^{\infty} R_k(x) s^{-k-1}$$

whose coefficients are $m \times m$ matrices of smooth functions defined on a neighborhood U of x_o, with invertible R_{-1} , which factorizes the formal power series $T(s,x)$ as

$$T(s,x) = K(s)R(s,x).$$

4.1.2 The Structure Algorithm

Following the steps outlined in Isidori for the structure algorithm to find a linearizing feedback, we proceed with step 1.

Step $i = 1$. We need to find the permutation matrix V_1 which which performs the row permutations such that

$$V_1 T_0(x) = \begin{bmatrix} S_1(x) \\ 0 \end{bmatrix} \tag{4.1.9}$$

We choose:

$$V_1 = \begin{bmatrix} P_1 \\ K_1^1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1.10}$$

where

$$\begin{aligned} \delta_1 &= r_0 \\ &= \text{rank } S_1(x) \\ &= 1 \\ \gamma_1(x) &= P_1 h(x) \\ &= h_4(x) \\ &= \omega_1 \\ \bar{\gamma}_1(x) &= K_1^1 h(x) \end{aligned}$$

$$= \begin{bmatrix} h_1(x) \\ h_2(x) \\ h_3(x) \\ h_5(x) \\ h_6(x) \end{bmatrix}$$

$$= \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

and note that

$$L_g\gamma_1(x) = S_1(x)$$

$$L_g\bar{\gamma}_1(x) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Now step $i = 2$. Consider the matrix;

$$\begin{bmatrix} L_g\gamma_1(x) \\ L_gL_f\bar{\gamma}_1 \end{bmatrix} = \begin{bmatrix} S_{i-1}(x) \\ L_gL_f\bar{\gamma}_1 \end{bmatrix}$$

since

$$d\bar{\gamma}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and we have the Lie derivative of $\bar{\gamma}_1$ with respect to f,

$$L_f\bar{\gamma}_1(x) = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ f_2(\theta_1, \theta_2, \theta_3, \omega_2, \omega_3) \\ f_2(\theta_1, \theta_2, \theta_3, \omega_2, \omega_3) \end{bmatrix}$$

and then since

$$dL_f\bar{\gamma}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \frac{\partial f_2}{\partial \theta_3} & 0 & \frac{\partial f_2}{\partial \omega_2} & \frac{\partial f_3}{\partial \omega_3} \\ \frac{\partial f_3}{\partial \theta_1} & \frac{\partial f_3}{\partial \theta_2} & \frac{\partial f_3}{\partial \theta_3} & 0 & \frac{\partial f_3}{\partial \omega_2} & \frac{\partial f_3}{\partial \omega_3} \end{bmatrix}$$

we see that

$$L_g L_f \bar{\gamma}_1(x) = \begin{bmatrix} I_1^{-1} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Now if we choose V_2 as

$$V_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} I_{\delta_1=1} & \\ K_1^2 & K_2^2 \end{bmatrix}$$

we get

$$V_2 \begin{bmatrix} L_g \gamma_1(x) \\ L_g L_f \bar{\gamma}_1(x) \end{bmatrix} = \begin{bmatrix} S_2(x) \\ 0 \end{bmatrix}$$

and

$$S_2(x) = S_1(s) = I_1^{-1}$$

so we are done. Also

$$\begin{aligned} \delta_2 &= r_1 - r_0 \\ \gamma_2(x) &= \text{does not exist} \\ \bar{\gamma}_2 &= K_1^2 \gamma_1(x) + K_2^2 L_f \bar{\gamma}_1(x) \end{aligned}$$

$$= \begin{bmatrix} I_1^{-1} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ f_2() \\ f_3() \end{bmatrix}$$

Next we form

$$\Gamma(x) = \begin{bmatrix} \gamma_1(x) \\ \gamma_2(x) \end{bmatrix} = \omega_1$$

and then α and β is found by the relationship

$$\begin{aligned} [L_g\Gamma(x)]\,\alpha(x) &= -L_f\Gamma(x) \\ [L_g\Gamma(x)]\,\beta(x) &= [1] \end{aligned}$$

so we have

$$\begin{aligned} \left[I_1^{-1}\right]\alpha(x) &= -f_1(x) \\ \left[I_1^{-1}\right]\beta(x) &= 1 \end{aligned}$$

and solving for α and β

$$\begin{aligned} \alpha(x) &= -I_1 f_1(\theta_1, \theta_2) \\ \beta(x) &= I_1 \end{aligned}$$

results in the linearizing feedback

$$u = -I_1 f_1(\theta_1, \theta_2) + I_1 v. \tag{4.1.11}$$

We have linearized the Galerkin model with respect to the hub rate and hub position but the structure algorithm degenerated before we could linearize the system with respect to the other outputs.

4.1.3 Implementation

In general, the method of feedback linearization requires a full state vector of the nonlinear system and this either requires a nonlinear observer, which must be implementable on a processor of reasonable cost and having sufficiently high sampling rate, or some distributed array of sensors to put on line the measured estimates of the states. In our particular system, the linearizing feedback requires only the two states θ_1 and θ_2 and is, therefore; in some sense simpler. Even a nonlinear observer for this case would be difficult to implement on a low cost digital signal processor like the AT&T WE DSP32, since it requires a long software subroutine to carry out most nonlinear functions. On the other hand, the DSP32 is very proficient at multiplication succeeded by addition which suggests its use for a linear control system.

Implementing the state vector estimator for the nonlinear system is achieved in a cost effective manner by use of sensors which approximate, in some sense, the states that are required for feedback. The optical shaft encoder measures θ_1; but θ_2, the angle of the first rigid element of the Finite Element approximation to the continuous beam, is currently unavailable. An approximation of this angle could be obtained through a strain gauge placed on the beam near its

connection to the hub. With such a strain gauge, the required state vector for feedback linearization would be on line with little delay due to preprocessing and at a relatively low cost.

Another drawback of the nonlinear control presented here is that it doesn't dampen the flexible modes of the beam; it merely servos the hub to the desired location. Since the linearized system is

$$\ddot{\theta}_1 = u,$$

the hub position could be servoed with high speed using a bang-bang controller, but this method makes no attempt to control the end effector other than through the natural damping of the beam. This could take some time for the tip position to settle to the actual angle indicated by the hub for systems with little or no damping. On the other hand, such a linearized system which servos only the hub to the reference position might be very useful in a satellite application which has some flexible structures attached that make control difficult.

For applications related to robotics, we are more concerned with end effector control than hub position control and a linear system of finite dimension could model the lower dominant modes of the system while truncating the remaining modes which have negligible amplitude. A model of this type, while only valid for modest slew rates, will approximate the response of a the nonlinear Galerkin model.

A possible strategy for control system design could to use linear state feed-

back which places the closed loop poles to achieve sufficiently fast response of the lowest rigid body mode of the hub while only the damping of the higher flexible modes is increased while the frequency of these higher modes remains unchanged. Such a system is implementable on a low cost DSP chip with a high sampling rate as this thesis demonstrates.

4.2 Discretization of the Linear Continuous Time System

The system identification procedures lead to an open loop transfer function for the flexible beam manipulator which describes the dynamics of the beam.

$$H(s) = \begin{bmatrix} H_{ac}(s) \\ H_{tach}(s) \\ H_{hub-pos}(s) \\ H_{tip-pos}(s) \end{bmatrix} \tag{4.2.12}$$

To discretize the system we use the operation

$$G(z) = (1 - z^{-1})\mathcal{Z}\ \mathcal{L}^{-1}\left(\frac{H(s)}{s}\right) \tag{4.2.13}$$

where $\mathcal{Z}$ signifies the Z transform operator and $\mathcal{L}^{-1}$ represents the inverse Laplace operator.

Since the discretization depends on the sampling rate, we will need two separate discretizations,i.e. one for 200 Hz and one for 400 Hz. A Macsyma package was written which, for a given sampling rate, discretizes the equations and converts the proper transfer function to a strictly proper one. Thus the discretization of the openloop system takes the form:

$$H(z) = \frac{b_1 z^5 + b_2 z^4 + b_3 z^3 + b_4 z^2 + b_5 z + b_6}{z^6 + a_1 z^5 + a_2 z^4 + a_3 z^3 + a_4 z^2 + a_5 z + a_6} + d \tag{4.2.14}$$

and the control canonical realization

$$\begin{aligned} x(k+1) &= \begin{bmatrix} -a_1 & -a_2 & -a_3 & -a_4 & -a_5 & -a_6 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \end{bmatrix} + du(k) \end{aligned}$$

is obtained and let us define Φ, Γ, and $\mathbf{C}$ as

$$x(k+1) = \Phi x(k) + \Gamma u(k) \tag{4.2.15}$$

$$y(k) = \mathbf{C}x(k). \tag{4.2.16}$$

The discretized transfer functions for a sampling rate of 200 Hz are:

$$\begin{aligned} H_{ac}(z) &= \frac{6.57z^5 - 25.97z^4 + 40.16z^3 - 29.47z^2 - 9.46z - 0.763}{z^6 - 5.49z^5 + 12.92z^4 - 16.74z^3 + 12.58z^2 - 5.21z + 0.925} - 36.45 \\ H_{tach}(z) &= \frac{15.2z^5 - 7.10z^4 + 136.44z^3 - 135.11z^2 + 69.04z - 14.5}{z^6 - 5.49z^5 + 12.92z^4 - 16.74z^3 + 12.58z^2 - 5.21z + 0.925} \\ H_{hub-pos}(z) &= \frac{0.58z^5 - 1.57z^4 + 1.02z^3 + 0.876z^2 - 1.44z + 0.53}{z^6 - 5.49z^5 + 12.92z^4 - 16.74z^3 + 12.58z^2 - 5.21z + 0.925} \\ H_{tip-pos}(z) &= \frac{-0.44z^5 + 1.23z^4 - 0.85z^3 - 0.66z^2 + 1.2z - 4.3}{z^6 - 5.49z^5 + 12.92z^4 - 16.74z^3 + 12.58z^2 - 5.21z + 0.925} \end{aligned}$$

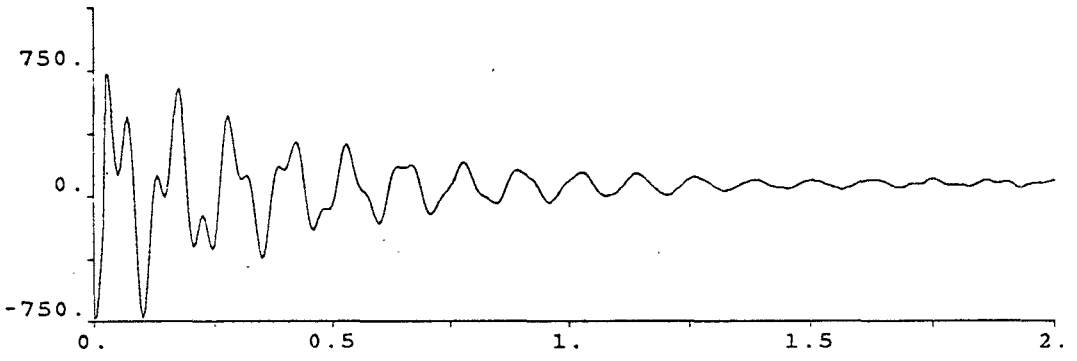


Figure 4.1: Pulse Response of Discretized Open Loop Model- Accelerometer

4.2.1 Simulation of Discretized Model

The discretized model was simulated so the system response could be compared to the response of the continuous time model to ensure that the discretization steps were error free. The results are in the following figures.

4.3 State Feedback Control System Design

Linear state feedback is employed to relocate the poles of the system. The feedback control signal sent to the motor which is computed at every sample is given by

$$u(k) = v_{ref} - Kx(k) \tag{4.3.17}$$

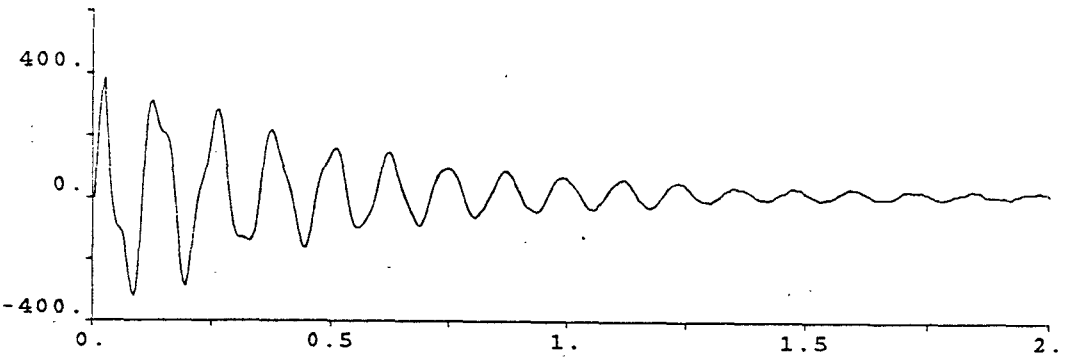


Figure 4.2: Pulse Response of Discretized Open Loop Model- Tachometer

where K is a 6×1 matrix of gains. The gains are chosen to maximize the performance and the optimization of the gains is computed using Console [14]. The reference voltage v_{ref} is the new input to the system and it corresponds to the desired location of the robot arm. For this experiment v_{ref} is a unit step input. The feedback control control computation above is dependent on the knowledge of the six states in x and to obtain these we construct a state estimator or *observer.*

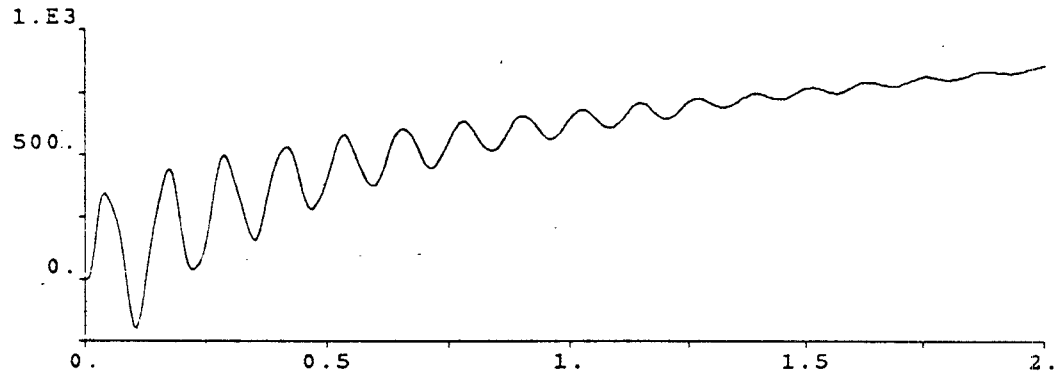


Figure 4.3: Pulse Response of Discretized Open Loop Model- Encoder

4.4 State Estimators

4.4.1 Full Order State Estimator

The full order state observer is constructed using the equation

$$\widehat{x}(k+1) = \Phi\widehat{x}(k) + \Gamma u(k) + L[y(k) - C\widehat{x}(k)] \tag{4.4.18}$$

which is a new dynamical system based on the dynamics of the open loop system. The new dynamical system is driven by the same input as the robot arm as well as a correcting term which is based on the difference between the measured and the estimated outputs. The 6×3 matrix L is chosen such that the estimate of the states has adequate convergence to the actual states of the system.

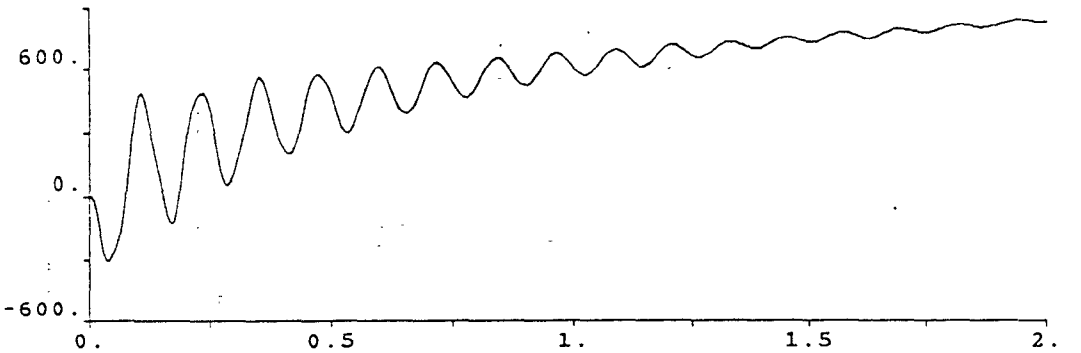


Figure 4.4: Pulse Response of Discretized Open Loop Model- Tip Position

The observer equation above reduces to

$$\hat{x}(k+1) = (\Phi - LC)\hat{x}(k) + \Gamma u(k) + Ly(k)$$

and since we have computed u above we get

$$\hat{x}(k+1) = (\Phi - LC - \Gamma K)\hat{x}(k) + \Gamma v_{ref} + Ly(k) \qquad (4.4.19)$$

This discrete time dynamical system is computed between every two consecutive samples to update the states for use in the feedback equation above. The observer 4.4.1 is a sixth order linear filter with infinite impulse response, having a polynomial transfer function and since DSP chips are excellent at filtering, they should be also be excellent at feedback control.

4.4.2 Reduced Order State Estimator

Since we directly measure the outputs of three sensors, all of this sensor information can be utilized by the observer to reduce the order of the observer. We can reduce the order of the state estimator from six for the full order observer to 3 for the reduced order observer. This is possible since the output matrix C has rank 3, i.e. all three outputs are linearly independent. A change of basis of the system will result in three states being transformed to a new system in which the sensor outputs correspond to the first three states in the state vector $\bar{x}$, i.e.

$$C = \begin{bmatrix} I_3 & 0_3 \end{bmatrix}$$

while the remaining 3 states do not represent any output but are estimated by the observer.

There is a trade off when constructing reduced order observers which is caused by sensor noise. Reduced order observers have the benefit of less computational steps but suffer if the sensor outputs are noisy since this noise is fed back into the plant. This could be a problem for noisy sensors like the tachometer and particularly the accelerometer, but with adequate filtering of the signals before sampling, the control signal u should not be adversely corrupted by the sensor noise.

The reduced order observer is constructed by first making a change of basis of the state space. The transformation matrix is defined as

$$P = \begin{bmatrix} C \\ R \end{bmatrix}$$

where R is an 3×6 arbitrary submatrix of real constants such that P is non-singular. The inverse of P

$$Q = P^{-1} = \begin{bmatrix} Q_1 & \vdots & Q_2 \end{bmatrix}$$

is needed as well. Here Q_1 and Q_2 are both 3×3 matrices. Next the control canonical realization, obtained previously, is transformed by $\bar{x} = Px$ so

$$\begin{aligned} \bar{x}(k+1) &= P\Phi P^{-1}\bar{x}(k) + P\Gamma u(k) \\ y(k) &= CP^{-1}\bar{x}(k) = CQ\bar{x}(k) = \begin{bmatrix} I_3 & 0_3 \end{bmatrix} \bar{x}(k) \end{aligned}$$

Now the transformed system is partitioned as

$$\begin{aligned} \begin{bmatrix} \bar{x}_1(k+1) \\ \bar{x}_2(k+1) \end{bmatrix} &= \begin{bmatrix} \bar{\Phi}_{11} & \bar{\Phi}_{12} \\ \bar{\Phi}_{21} & \bar{\Phi}_{22} \end{bmatrix} \begin{bmatrix} \bar{x}_1(k) \\ \bar{x}_2(k) \end{bmatrix} + \begin{bmatrix} \bar{\Gamma}_1 \\ \bar{\Gamma}_2 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} I_3 & 0_3 \end{bmatrix} \bar{x}(k) = \bar{x}_1(k) \end{aligned}$$

Here $\bar{x}_1$ is the first 3 elements of $\bar{x}$ and $\bar{x}_2$ is the latter 3 elements of $\bar{x}$. Using the output equation $y = \bar{x}_1$ as well as introducing the shift operator notation ($qf = f(k+1)$), we can rewrite 4.4.20 as

$$\begin{aligned} qy &= \bar{\Phi}_{11}y + \bar{\Phi}_{12}\bar{x}_2 + \bar{\Gamma}_1 u \\ q\bar{x}_2 &= \bar{\Phi}_{22}\bar{x}_2 + \bar{\Phi}_{11}y + \bar{\Gamma}_2 u. \end{aligned}$$

and if we define two new variables

$$\begin{aligned} \bar{u} &= \bar{\Phi}_{21}y + \bar{\Gamma}_2 u \\ w &= qy - \bar{\Phi}_{11}y - \bar{\Gamma}_1 u \end{aligned}$$

the system becomes

$$q\bar{x}_2 = \bar{\Phi}_{22}\bar{x}_2 + \bar{u} \tag{4.4.20}$$

$$w = \bar{\Phi}_{12}\bar{x}_2. \tag{4.4.21}$$

Here $\bar{u}$ and w can still be thought of as input to a linear system (even though they contain y and qy) because they are known signals driving the observer.

Theorem 4.3 *The pair $\{\Phi, C\}$ is observable if and only if the pair $\{\bar{\Phi}_{22}, \bar{\Phi}_{12}\}$ is observable. ($\bar{\Phi}_{22}$ is $(n-q)\times(n-q)$ and $\bar{\Phi}_{12}$ is $q\times(n-q)$)*

Proof

The pair $\{\bar{\Phi}, \bar{\mathrm{C}}\}$ is observable $\Longleftrightarrow$

$$\operatorname{rank}\begin{bmatrix} s\mathbf{I} - \bar{\Phi} \\ \bar{\mathrm{C}} \end{bmatrix} = n \quad \forall s \in \mathbb{C}.$$

$\Longleftrightarrow$

$$\operatorname{rank}\begin{bmatrix} s\mathbf{I} - \bar{\Phi}_{11} & -\bar{\Phi}_{12} \\ -\bar{\Phi}_{21} & s\mathbf{I} - \bar{\Phi}_{22} \\ \bar{\mathrm{I}}_q & 0 \end{bmatrix} = n \quad \forall s \in \mathbb{C}$$

$$= \operatorname{rank}\mathbf{I}_q + \operatorname{rank}\begin{bmatrix} -\bar{\Phi}_{12} \\ s\mathbf{I} - \bar{\Phi}_{22} \end{bmatrix} \quad \forall s \in \mathbb{C}$$

$$= \quad q + \text{rank} \begin{bmatrix} s\mathbf{I} - \bar{\Phi}_{22} \\ \\ \bar{\Phi}_{12} \end{bmatrix} \quad \forall s \in \mathbb{C}$$

Thus the pair $\{\bar{\Phi}, \bar{\mathrm{C}}\}$ is observable

$\Longleftrightarrow$

$$\text{rank} \begin{bmatrix} s\mathbf{I} - \bar{\Phi}_{22} \\ \\ \bar{\Phi}_{12} \end{bmatrix} = n - q \quad \forall s \in \mathbb{C}$$

$\Longleftrightarrow$

$$\{\bar{\Phi}_{22}, \bar{\Phi}_{21}\} \quad \text{is observable.}$$

□

Since the system 4.4.21 is observable there exists a third order estimator of $\bar{x}_2$ in the form

$$q\hat{\bar{x}} = (\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12})\hat{\bar{x}} + \bar{L}w + \bar{u} \tag{4.4.22}$$

where the eigenvalues of $(\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12})$ can be assigned arbitrarily by appropriate choice of the 3×3 matrix $\bar{L}$.

Substituting w and $\bar{u}$ into the estimator yields

$$q\hat{\bar{x}}_2 = (\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12})\hat{\bar{x}}_2 + \bar{L}(qy - \bar{\Phi}_{11}y - \bar{\Gamma}_1 u) + (\bar{\Phi}_{21}y + \bar{\Gamma}_2)u \tag{4.4.23}$$

but this equation contains qy which is the output at the next sampling interval. The term qy can be eliminated if we define

$$z = \hat{\bar{x}}_2 - \bar{L}y$$

with the observer

$$\begin{aligned} qz &= (\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12})(z + \bar{L}y) + (\bar{\Phi}_{21} - \bar{L}\bar{\Phi}_{11})y + (\bar{\Gamma}_2 - \bar{L}\bar{\Gamma}_1)u \\ &= (\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12})z + (\bar{\Gamma}_2 - \bar{L}\bar{\Gamma}_1)u \\ &+ [(\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12})\bar{L} + (\bar{\Phi}_{21} - \bar{L}\bar{\Phi}_{11})]y \end{aligned}$$

where

$$\hat{\bar{x}}_2 = z + \bar{L}y$$

is the estimate of $\bar{x}_2$. The full state vector is then given by

$$\begin{aligned} z(k+1) &= (\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12})z(k) + (\bar{\Gamma}_2 - \bar{L}\bar{\Gamma}_1)u(k) \\ &+ \left[(\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12})\bar{L} + (\bar{\Phi}_{21} - \bar{L}\bar{\Phi}_{11})\right] y(k) \qquad (4.4.24) \\ \hat{\bar{x}}(k) &= \begin{bmatrix} \hat{\bar{x}}_1(k) \\ \\ \hat{\bar{x}}_2(k) \end{bmatrix} = \begin{bmatrix} y(k) \\ \\ \bar{L}y(k) + z(k) \end{bmatrix}. \qquad (4.4.25) \end{aligned}$$

4.4.3 Observer Pole Selection and Placement

We wish to design the observer so as to have the error between estimate and actual state converge asymptotically to zero, so we design to observer to have faster poles than the highest mode frequency of the closed loop system. All poles were chosen to be three times faster than the highest mode frequency of the modeled beam, i.e.

$$f_o = 3 \times 20 = 60.$$

If the poles were made any faster, the estimated states would be overly sensitive to any sensor noise that enters into the observer since the observer also should

have a desirable noise smoothing effect in addition to estimating the states.

4.4.4 An Efficient Observer Algorithm

An algorithm for the implementation of the observer achieves an increased level of efficiency over the general reduced order observer. In particular the update of $z(k)$ in equation 4.4.24 is made more efficient by choosing the 3×3 matrix L such that the matrix

$$\bar{\Phi}_{22} - \bar{L}\bar{\Phi}_{12} = \begin{bmatrix} e^{-2\pi f_o t_s} & 0 & 0 \\ 0 & e^{-2\pi f_o t_s} & 0 \\ 0 & 0 & e^{-2\pi f_o t_s} \end{bmatrix}. \tag{4.4.26}$$

Here both Φ_{22} and $\bar{\Phi}_{12}$ are 3×3 matrices as well and t_s is the sampling period.

This diagonalizes and decouples the observer dynamics and as a result simplifies the computation of the update of $z(k)$. Also since all poles are identical, the memory space requirements are reduced although this is not really an issue in this particular implementation since we have more than enough. To diagonalize the system, we must solve for each element of the 3×3 matrix L in equation 4.4.26.

The reason we can achieve this diagonalization is because we have three sensors and thus the 9 values of the L matrix can be solved for the 9 particular values that we want in the matrix 4.4.26. The computation of L which places the poles and diagonalizes 4.4.26 is a linear set of 9 equations with 9 unknowns. Diagonalization would not be possible if we had less than three sensor measure-

ments because we would not have enough parameters in the L matrix to choose such that the elements of matrix 4.4.26 are diagonal in addition to placing the poles of the observer.

4.5 Console-Simnon Optimization of Feedback Gains

Linear state feedback of the form

$$u(k) = k_0(v_{ref} - Kx(k)) \tag{4.5.27}$$

is used to control the flexible robot where K is a 6×1 feedback gain matrix

$$K = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 & k_5 & k_6 \end{bmatrix}.$$

The feedback gains are optimized using Console [14] where some stringent performance criterion is placed on the system response. The criterion specifies the rise time and overshoot and Console optimizes the seven feedback gains k_i, $i = 0, 6$, to best achieve the performance specifications.

Unlike the work of Frank [1] and Wang [2], an additional scale factor k_0 is also optimized to scale the overall error from the reference position. Since our motor has a maximum voltage input of ± 10 Volts, corresponding to an internal computer representation of ± 2048, a nonlinear saturator is present in the system. This limits our ability to place the poles of the system arbitrarily by restricting the magnitude of the control to the motor. The additional optimization parameter k_0 allows the feedback error magnitude to be so large in magnitude

that it is into the saturation range. The optimization is performed with the saturator present and this ensures that the controller uses all of the available motor torque. In the thesis of both Frank and Wang the reference value was fixed and the magnitude of the control signal was well under the limit of the saturator. This seems to suggest that we can significantly increase the performance by scaling up k_0 to use more of the available motor torque to control the flexible manipulator.

Another benefit of having a larger k_0 is that ripple torque does not alter the final steady state position as much by forcing the shaft position toward one of the stable points in the ripple torque cycle. Under final steady state conditions, ripple torque present in the system tends to force the the hub into the location of a "potential well". If this position does not correspond to the desired reference position, then a steady state positioning error will be caused by the ripple torque pulling the hub away from the desired position. With an increased scale factor k_0, the overall gain on just the position error is increased and this will cause the final position accuracy to be less sensitive to the ripple torque.

The discretized closed loop system was optimized for both a sampling rate of 200 and 400 Hz with identical criterions on performance specifications. The performance specifications are defined in the following convert file in figure 4.5.

The console optimization was iterated with a simple PD controller as the initial reference starting gains and allowed to be optimized from there. The system is optimized over all k_i and the final optimal set of feedback gains for

```
design_parameter k1 init= 0 variation=1.0e-4
design_parameter k2 init= 2.3 variation=2.3
design_parameter k3 init= 0.9 variation=0.9
design_parameter k4 init= 0 variation=1.0e-3
design_parameter k5 init= 0 variation=1.0e-3
design_parameter k6 init= 0 variation=1.0e-3
design_parameter k0 init= 3.0 variation=3.0

initialization {
   simnon( "syst fb12");
   simnon( "store v y1 y2 y3 y4 y5 y6" );
   simu( 0.0, 1.6);
   }

functional_objective "overshoot"
   for t from 0 to 1.6 by .005
   minimize {
      double output();
      return output( "y4", t);
      }
   good_curve = {
       if( t <= 0.65 ) return 1000;
       else        return 1000;
       }
   bad_curve = {
       if( t <= 0.65 ) return 1200;
       else        return 1100;
       }

functional_objective "rise"
   for t from 0.33 to 1.6 by .005
   maximize {
      double output();
      return output( "y4", t);
      }
   good_curve = {
       if( t <= 0.4 ) return 900;
       else        return 999;
       }
   bad_curve = {
       if( t <= 0.5 ) return 600;
       else        return 900;
       }
exit
```

Figure 4.5: Convert File of Performance Specifications

```
Name         Value         Variation  wrt 0   Prev   Iter=14
k1           1.88801e-07   1.0e-04    ****%    7%
k2           2.27634e+00   2.3e+00      -1%    0%
k3           1.05849e+00   9.0e-01      17%    0%
k4           1.15239e-05   1.0e-03    ****%    0%
k5           6.66089e-06   1.0e-03    ****%    2%
k6          -1.18617e-05   1.0e-03    ****%    0%
k0           3.45315e+00   3.0e+00      15%    0%

Pcomb (Iter= 14)  (Phase 2)  (MAX_COST_SOFT= 0.0359949)

SPECIFICATION    PRESENT   GOOD                 G          B              BAD
FO1  overshoot   1.01e+03  1.00e+03 ==========*          |              1.20e+03
FO2  rise        9.85e+02  9.99e+02           *==========================  6.00e+02
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]Optimal code 3

<14> cleaning SIMNON ...
```

Figure 4.6: Optimal Feedback Gains for closed loop system sampled at 200 Hz

the two sampling rates are listed in figure 4.6 and 4.7.

4.6 Sampling Rate Comparison

4.6.1 200 Hertz

The step response of the experimental system with the feedback observer controller in place is shown in figures 4.8, 4.9, and 4.10.

Figure 4.10 shows the response of both the hub position and the tip position which is generated from the observer. Although this plot shows some noise, it does follow the expected path of the tip and the nonminimum phase property is clearly present.

```
Name         Value          Variation  wrt 0   Prev
k1          -9.64316e-06    1.0e-04    ****%     0%
k2           2.39264e+00    2.3e+00       4%     0%
k3           1.33221e+00    9.0e-01      48%     0%
k4          -1.80057e-04    1.0e-03    ****%     0%
k5           1.74842e-04    1.0e-03    ****%     0%
k6          -1.01043e-04    1.0e-03    ****%     0%
k0           3.57706e+00    3.0e+00      19%     0%

Pcomb (Iter= 20)  (Phase 2)  (MAX_COST_SOFT= 0.0449483)

SPECIFICATION     PRESENT   GOOD              G          B               BAD
F01  overshoot    1.01e+03  1.00e+03 =========*          |               1.20e+03
F02  rise         9.81e+02  9.99e+02          *==========================6.00e+02
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]
[SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON] [SIMNON]
[SIMNON] [SIMNON] [SIMNON]Optimal code 3

<20> cleaning SIMNON ...
```

Figure 4.7: Optimal Feedback Gains for closed loop system sampled at 400 Hz

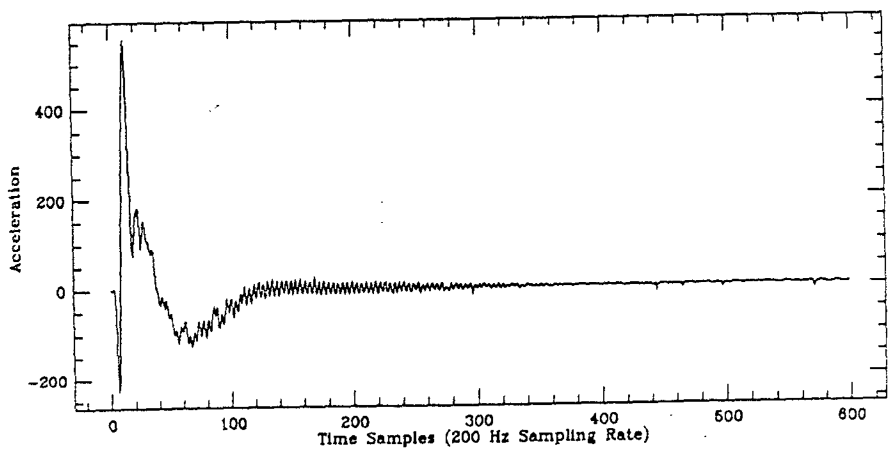


Figure 4.8: Accelerometer step response of physical closed loop system: 200 Hz

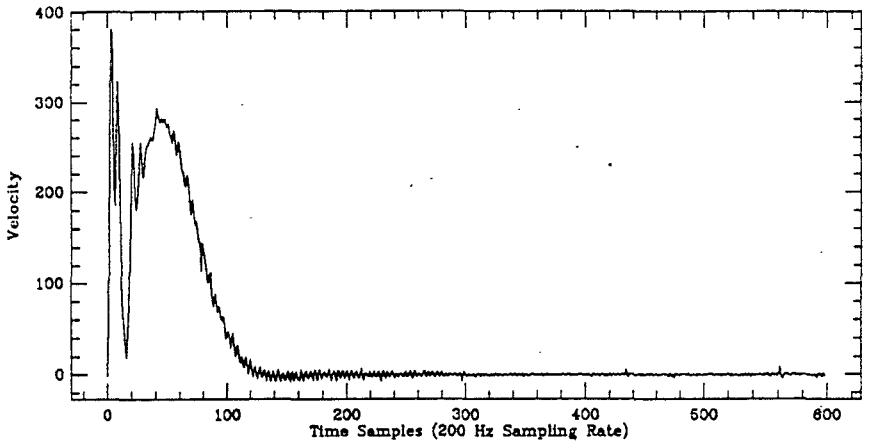


Figure 4.9: Tachometer step response of physical closed loop system: 200 Hz

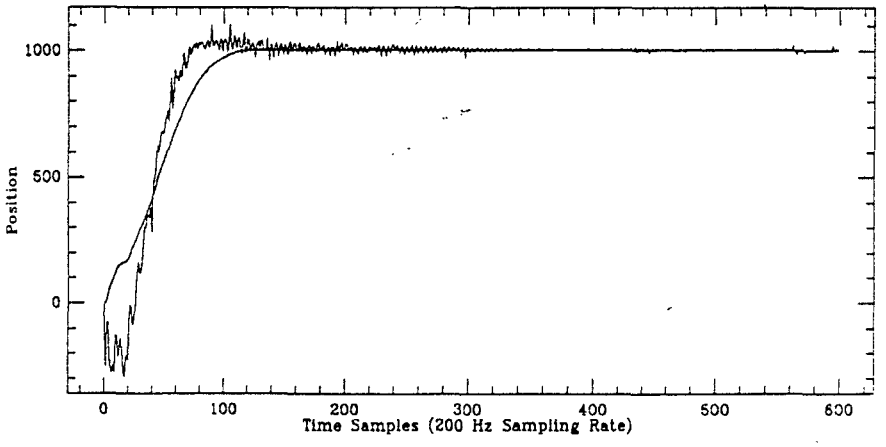


Figure 4.10: Tip and Hub position step response of physical closed loop system: 200 Hz

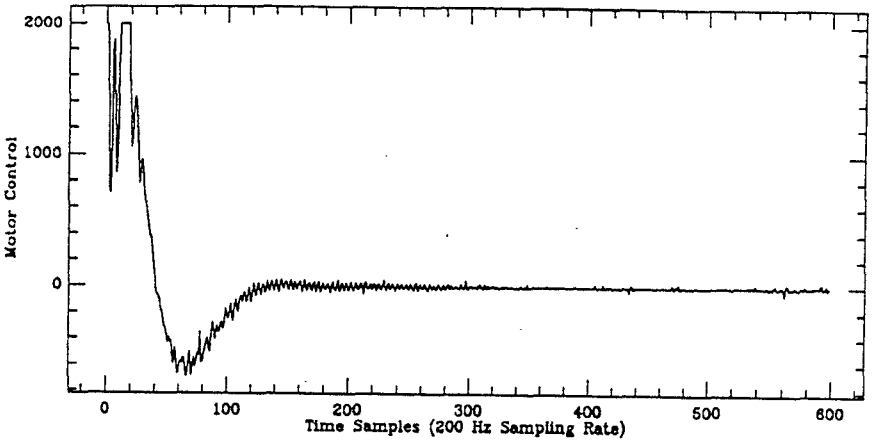


Figure 4.11: Control Signal to Motor: 200 Hz

4.6.2 400 Hertz

The step response of the experimental closed loop system sampled at 400 Hz with the feedback observer controller in place is shown in figures 4.8, 4.9, and 4.10.

4.7 Performance Comparison

To evaluate the performance of the control system on the DSP32 chip, a step function was input to the motor to study how the manipulator arm moved to its new position. Figure 5 shows the motion of the tip position as a function of time. The tip reaches its destination approximately 0.5 second later with no overshoot. Also visible from the graph are some vibrations due to the flexibility

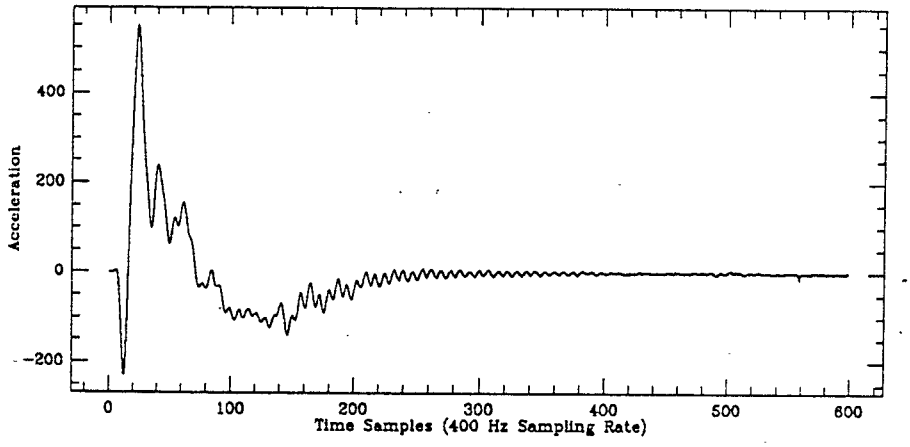


Figure 4.12: Accelerometer step response of physical closed loop system: 400 Hz

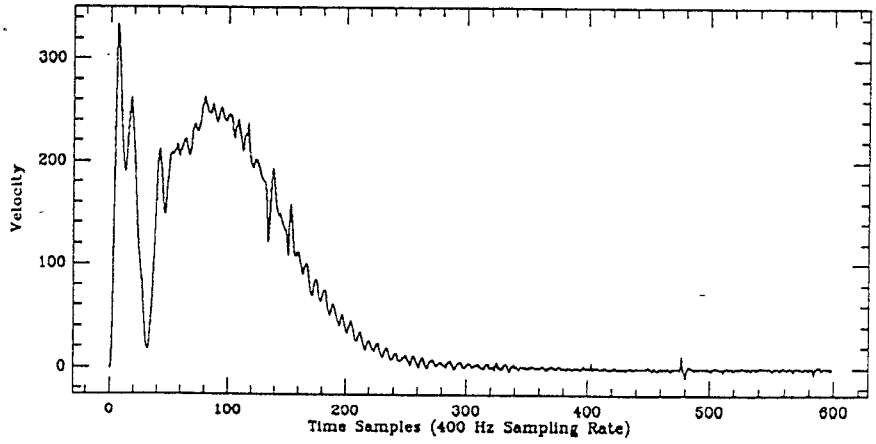


Figure 4.13: Tachometer step response of physical closed loop system: 400 Hz

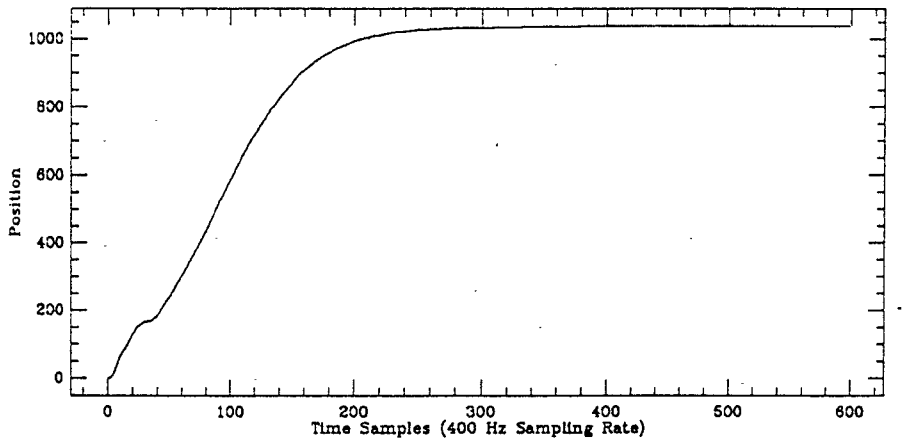


Figure 4.14: Hub position step response of physical closed loop system: 400 Hz

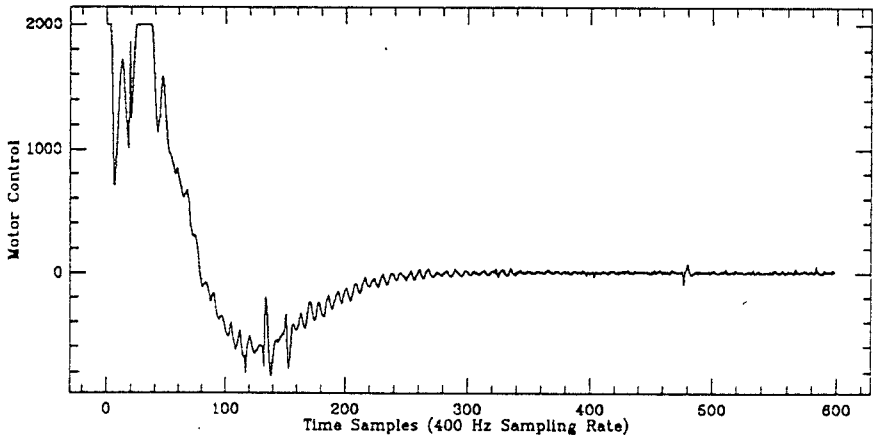


Figure 4.15: Control Signal to Motor: 400 Hz

of the beam which are damped out as the tip nears its destination. Both the 200 Hz and the 400 Hz have nearly identical responses.

CHAPTER
FIVE

Digital Signal Processing with the DSP32

5.1 DSP32 Architecture

Digital signal processing applications generally require a large number of repetitive mathematical operations such as multiply and accumulate. If floating point arithmetic is to be used, conventional microprocessors and co-processors, which use microcode and software simulation for floating point arithmetic operations, suffer in performance compared to the DSP chips which have floating point arithmetic hardware. The AT&T WE DSP32 uses a fast hardware floating point arithmetic unit which can peak at 25 MFLOPS in the latest CMOS version, the DSP32C. The floating point ALU is highly pipelined and can achieve two floating point operations per clock cycle, allowing a multiply and an accumulate operation execute in a single instruction cycle.

The performance and costs [15] of various DSP chips summarized in Table 5.1 is adequate for many real-time applications such as speech, servo-control, and graphics. Previously, dsp chips were lacking in development software, but the situation is improving with Optimizing C compilers available and Software

Processor	On-Chip Memory (bits)	Instruction Cycle Time	Mult./Accum. Time	Single Chip Cost
AT&T DSP32	1K x 32	160 ns	160 ns, 250ns	$190
AT&T DSP32C	1K x 32	80 ns	80 ns , 100ns	NA
TI TMS 320C1X	256 x 16	160 ns	320 ns	$50
TI TMS 32020	288 x 16	200 ns	200 ns	$120
TI TMS 320C30	2K x 32	60 ns	60 ns	$1300
AD ADSP2100	16 x 24	125 ns	80 ns , 100ns	$411

Table 5.1: Comparison of Some Commericially Available DSP chips

Libraries of common arithmetic functions. The DSP32 architecture is shown in Figure 1 and has a separate address bus (16 bit) and data bus (32 bit), Data Arithmetic Unit (DAU), Control Arithmetic Unit (CAU), parallel IO port, and Serial IO port, and four banks of reconfigurable memory.

5.1.1 Data Arithmetic Unit

The DAU is the main data processing unit for DSP algorithms and it consists of a floating point multiplier, an adder, and four static 40-bit accumulators. The floating point numbers are represented in a special 32-bit format which is compatible with the DAU hardware. Inputs to the multiplier and adder can be from memory, IO, or another accumulator. The multiplier and adder operate in parallel and, as a result, there are certain latency effects that must be recognized when performing particular software tasks. The pipeline achieves the single cycle execution for all instructions, but it also makes control of the processor more difficult and this emphasizes the need for a separate Control Arithmetic Unit (CAU) on the DSP chip.

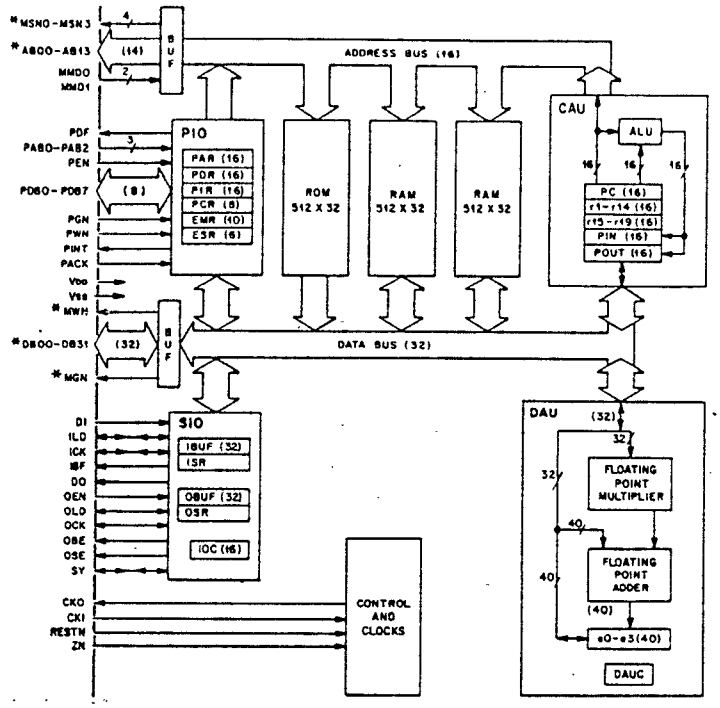


Figure 5.1: DSP32 Architecture Block Diagram, source: [17]

5.1.2 Control Arithmetic Unit

The function of the CAU is to generate the addresses to memory so it operates on 16 bit integers and they essentially resemble microprocessor instructions. The CAU is not limited to generating memory addresses exclusively and can process integer data as well in its own ALU. There are twenty-one 16-bit general purpose registers (r1-r21) and a 16-bit program counter. Some of the registers have special functions such as the increment registers (r15-r19) which will post modify an address in the memory pointers. This is very useful in signal processing algorithms where for example, the tap weights of a filter are located in consecutive memory locations and the filter can just step through the weights as it computes

the output of the filter. The DSP32 is capable of Serial DMA and the pointer in (r20) and pointer out (r21) are used for these transfers.

5.1.3 Memory

The memory in the DSP32 may be accessed as 8, 16, or 32 bit words. In addition to the single cycle instruction execution, up to four memory accesses per instruction cycle are possible; instruction fetch, two operand fetches and a memory write. The four blocks of on-chip memory consist of one block with 2Kbytes ROM, and three blocks each with 4 Kbytes of dynamic RAM (512×32 words). Memory is expandable off chip up to a maximum of 56 Kbytes which is directly accessible by addresses generated in the CAU. The four blocks and the external memory can be configured in four modes as shown in figure 2. All of the modes split the four blocks and off-chip memory into an upper and lower bank. The reason for splitting the memory into two is for the DSP32 to achieve maximum throughput, memory accesses must alternate between the two banks and the different modes allow us to specify how to best organize the memory for our application. For this control system application Memory Mode 2 is used since it maximizes the amount of total memory available. In this pipelined architecture, as one bank is being accessed, the other is being addressed and the pipelining effectively reduces the memory access time by one half. On the other hand, it is possible to access the same memory bank consecutively, but the control unit will insert a wait state (= 1/4 instruction cycle) and thus reduce

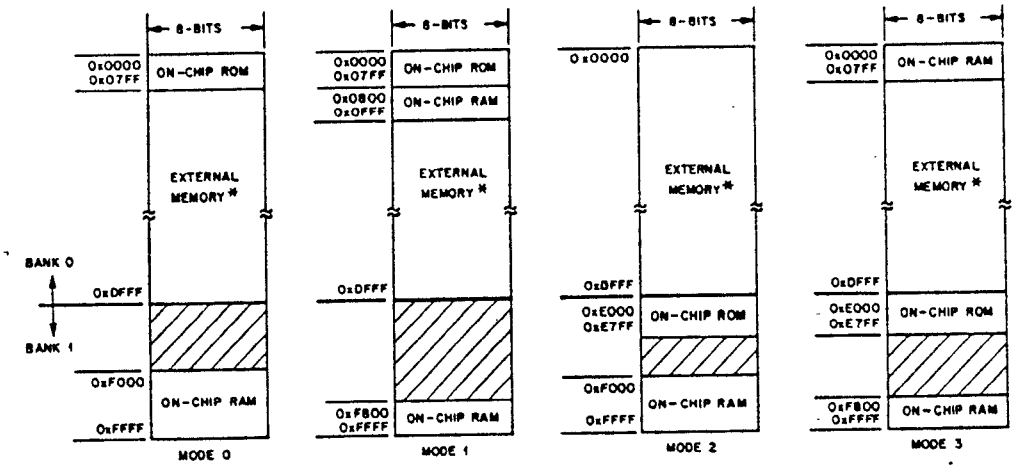


Figure 5.2: Memory Modes, source: [17]

the speed of the computation.

5.1.4 Processor Cycle and Pipelining

The processor cycle for the multiply and accumulate instruction

$$*r3 = a0 = a1 + *r1 * *r2$$

is executed in four states:

1. fetch X and Y fields (*r1 and *r2)

2. multiply $*r1 * *r2$

3. accumulate product with a1 and put result in a0

4. write result to memory (optional).

Because of the pipelining, we get single cycle execution even with a write to memory. This is useful when performing such tasks as windows, adaptive filtering, or matrix operations that can take a block of data form memory, process it and write it back to memory all in one instruction.

5.1.5 Instruction Set and Latency Effects

The two types of instructions that are found on the DSP32 are the Data Arithmetic (DA) instructions and the Control Arithmetic (CA) instructions. The DA instructions consist of multiply/accumulate instructions like

$$[Z =]aN = [-]aM\{+,-\}Y * X$$

and some special functions such as: float(), int(), round() . The CA instructions are executed in the CAU and consist of the control, arithmetic/logic, and data move instructions. Examples of these are respectively:

- if (DA COND) goto r3
- r2 = r5 + r6
- r3 = memory-location

There are three classes of latencies that we will discuss here:

- Memory Writes
- Accumulator as Multiplier Input

- Conditional Branching on DA conditions

When a DA instruction writes to memory, the value written is not available to be read until four instructions later.

Example:

$I1 \quad *r3 = a0 = a0$

$I2 \quad *r3 = a3 = a3$

$I3 \quad .$

$I4 \quad .$

$I5 \quad a1 = *r3$

Here the value read in I5 is the value written to memory in I1 not I2.

When an accumulator, a0-a3 is used as an input to the multiplier, its value is established at least three instructions prior to the multiply instruction.

Example:

$I1 \quad a0 = a0 + *r1 * *r2$

$I2 \quad a0 = a0 + a1$

$I3 \quad .$

$I4 \quad a2 = a0 * a0$

The value of a0 used in I4 is computed in I1 not I2.

A DAU condition tested in a conditional branch is established by the DA instruction which is at least four instructions before the test. Example:

$I1$ $a0 = a0 + a1$

$I2$ $a2 = a0 * a2$

$I3$.

$I4$.

$I4$ if (agt) goto next

The condition "agt", which is *test if accumulator is greater than zero*, is established by I1.

5.2 Implementation of Observer-Controller on DSP32

A linear control system for the flexible link manipulator was implemented on the DSP32 with a sampling rate of 200 Hz and 400 Hz. There are two separate programs, one for the IBM PC and one for the DSP32. The IBM PC program is written in C and compiled using the Microsoft C compiler.

The C program initializes the analog to digital converters, initializes the DSP32 and downloads the DSP32 executable file to the memory of the DSP board. The IBM program also handles the calls to retrieve input from sensors and to send out a new control signal to the motor. It makes use of some key subroutines supplied by Communications, Automation, and Control [16] for initialization, downloading programs and data and uploading results.

The program for the DSP32, written in assembly language and assembled with the assembler provided by AT&T, is where the algorithm (see figure 5.3

) for the feedback control system resides. First, the program waits in a loop until the IBM PC downloads the new sensor samples and imediately computes the new control output to the motor. Then the DSP algorithm signals the IBM PC that the motor control is ready (located in a known memory location on the DSP32 board) and to upload the value. After finishing the computation of the new motor control, it imediately starts updating the three states using the reduced order state observer. This architecture/algorithm achieves some degree of parallelism here since the DSP chip does not wait for the PC to finish uploading the results before it can start the state observer. After the observer computation computes the states necessary for the next sampling instant, the algorithm loops back and waits in a "wait loop" for the new sensor values from the PC and the signal to start.

The signal to start is given by downloading a 1 to the memory location "flag" and the signal to the IBM PC that results are ready is that "flag" is set to 0. The IBM program constantly monitors this memory location "flag" to determine when the control is ready for uploading.

The reduced order observer calculation is split into two parts which enables the motor control to be output virtually imeadately upon receiving the latest sensor data. Only that part of the observer which uses the latest sensors is used placed before the control output. The part of the observer which uses the

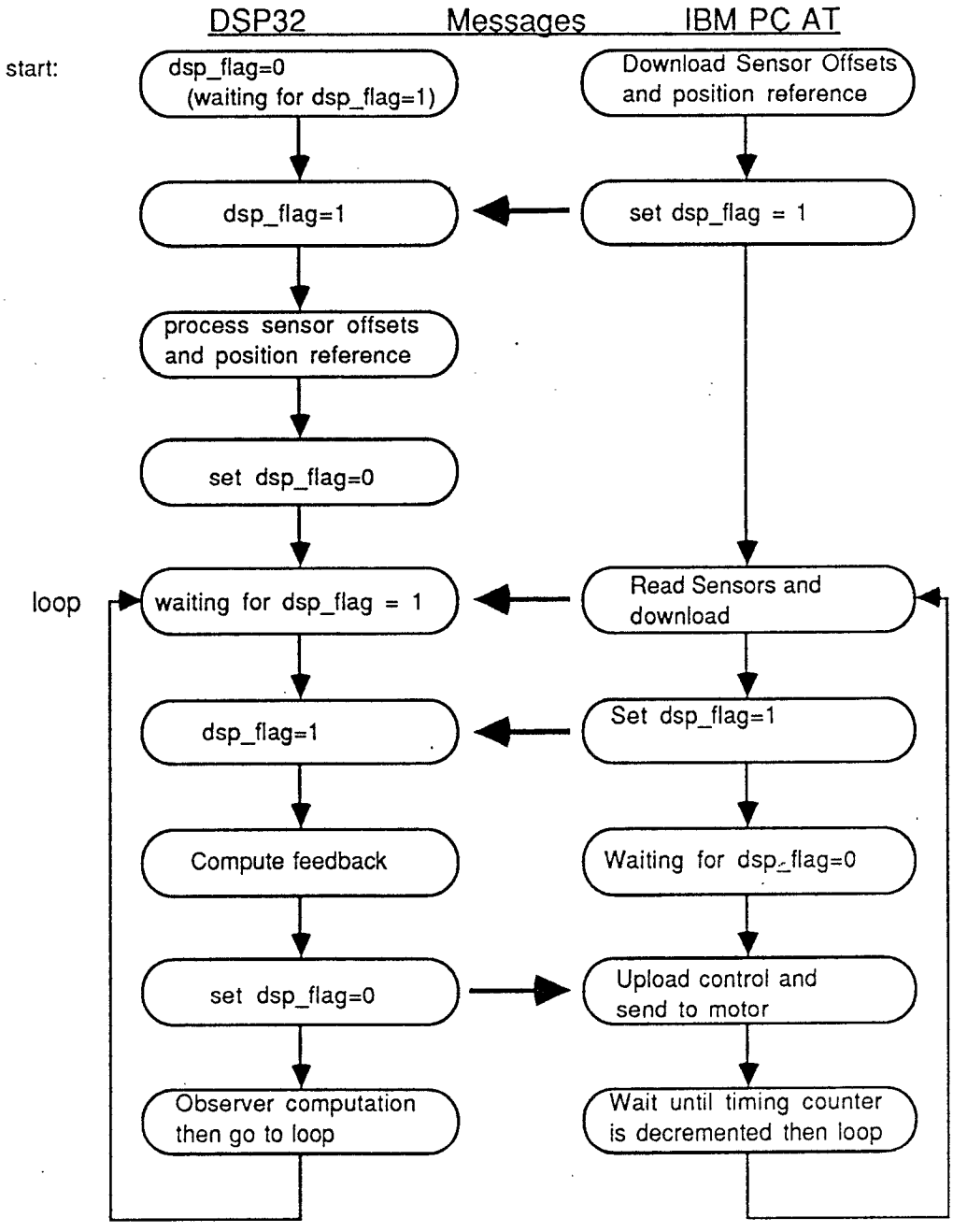


Figure 5.3: Flowchart of IBM PC program synchronized with DSP32 Program

previous sensor values , i.e.

$$z(k+1) = Az(k) + bu(k) + Cy(k) \tag{5.2.1}$$

was done at the end of the previous loop while the control was being uploaded and is stored until needed. At the start of the next loop, $z(k+1)$ is used in the calculation

$$x_2(k+1) = Ly(k+1) + z(k+1) \tag{5.2.2}$$

and then together with the three sensors, we have the whole state vector for feedback. The feedback is computed, sent through the saturator, and then offset by the motor offset value 2005 before it is uploaded to the IBM and sent to the motor. The time starting from the point where the DSP chip receives the new sensors to when the DSP chip sends the control to the IBM is 11×10^{-6} seconds, while the total sampling rate is either 5×10^{-3} or 2.5×10^{-3} so the control is output almost instaneously once the new sensors are in.

A time saving assumption is made in the syncronization scheme and that is that after the new sensor values are downloaded to the DSP board and the flag is set to 1, the DSP board is assumed to be finished with the observer calculation. The assumption reduced the time of the algorithm since the C program does not have to upload a flag to check this condition. We can make this assumption since the computational speed of the DSP board is so greast that it finishes one loop of its algorithm much faster than the inter-sample time.

I/O Task	Rate
Upload Rate	0.728 Mbytes/sec
Download Rate	0.412 Mbytes/sec

Table 5.2: Data Transfer Rates

DSP Task	Time Estimate
Download 3 sensors (6 bytes)	14.1μ sec
Upload motor control (2 bytes)	2.7μ sec
Download Flag (2 bytes)	4.7μsec
Upload Flag (2 bytes)	2.7μsec
Linear Feedback Computation	93×250 nsec $= 23.25\mu$ sec
Observer Computation	45×250 nsec $= 11.25\mu$ sec
Total Time Estimate	59.15 μ sec

Table 5.3: Time Estimates of DSP Algorithmic Processes

5.2.1 Time Estimates of DSP Tasks

The data transfer rates were measured between the DSP board memory and the IBM PC memory and the estimates are tabulated in Table 5.2

With these data transfer rates and the number of operations in the algorithm, we can estimate the time for tasks with the assumption that we have one operation per clock cycle with a clock cycle of 250 nanoseconds. Table 5.3 gives the estimates of individual tasks as well as the total time estimate. The time estimate is felt to be a lower bound on the total time since wait states occaisionaly are placed to get required values out of the pipeline before they can be used for a subsequent calculation. We therefore need to allow some extra time as a margin of safety to ensure that the two processors remain synchronised.

CHAPTER
SIX

Conclusion

This thesis demonstrates that successful control system design is achievable provided we have the proper tools at hand. To summarize what we have designed and implemented, we have:

- Increased the processing power of the digital controller through use of a DSP Chip.
- Achieved some computational improvements by implementing a diagonalized reduced order observer which takes advantage of the availability of all three sensor outputs.
- Significantly improved the sampling rate to adequately measure the 19Hz mode and allow the system to respond to disturbances quicker.
- Implemented an observer feedback matrix of unrestricted dimension, i.e. L is a 3×3 matrix with all nine elements available for pole placement in the reduced order model. In previous experiments [1] [2], which used a full sixth order observer, the observer feedback matrix L was restrained

by the condition

$$L = \tilde{L}\Gamma \tag{6.0.1}$$

where L is a 6×3 matrix, $\tilde{L}$ is 1×3, and Γ is fixed to be the 6×1 vector defined by the open loop system

$$x(k+1) = \Phi x(k) + \Gamma u \tag{6.0.2}$$

$$y(k) = Cx(k) \tag{6.0.3}$$

Thus only three parameters were available to the control system designer in constructing the observer.

- Optimized the feedback gains K to allow an overall scaling which lets the controller make full use of all available motor torque. Although the motor saturator causes the torque to occasionally be clipped, the overall system step response is approximately three times faster than previous experiments by [2] and the saturator does not seem to cause any undesirable effects in the system. Since the system in optimized using Console with the saturator in place , the final optimal control is such that this nonlinearity is accounted for and the feedback is found which optimizes performance under the control magnitude constraint. With a large slew angle (greater or equal to one radian), the signal to the motor is occasionally clipped since the controller sends to the motor the maximum motor torque allowed.

- Performed the system identification with the friction compensator operating to achieve a more accurate model. As a test, the system identification was performed without the friction compensator installed and this system was highly damped and exhibited minimal vibrations in the beam.

In addition to these steps which let to the successful implementation, a useful architecture for a feedback control system was described and examples of programming the processor are given in the appendix. The Digital Signal Processor seems to be very well suited for linear feedback control systems and if any changes were to be made in the future, I would suggest a more direct method of input for the sensors into the DSP chip. Timing estimates of the tasks for control are measured or estimated , as the case may be, and this allows us to determine an upper bound on the sampling rate for the algorithm we have implemented on this hardware.

The control system, as we have designed it, has undergone a state variable transformation which makes as one of the states (x_4) correspond to the tip position. This information on tip position is not available directly from a sensor and we must settle for an estimated value. The tip position is available for feedback along with the other states to achieve the performance we obtained. Console proved to be an excellent tool for successful optimization based control system design, and gives some insight to the optimal control. The feedback from the tip position state for the optimized system was small suggesting that the controller doesn't find this state extremely useful in controlling the tip. Although

this seems paradoxical, perhaps it can be explained by imagining a system based on tip position feedback alone. Such a system has some instabilities due to the nonminimum phase property which can result in the tip being 180 degrees out of phase with the hub position [3].

Furthermore, an analysis of the nonlinear Galerkin model was undertaken and through simulation, we can compare this "inextensible" model to that of the linear model than we ultimately implemented. In addition to this comparison, the *nonminimum phase* property of the flexible beam is clearly visible from the impulse response of the 3 body Galerkin model. Such analysis and simulation is useful for successful control system design by bringing to light addition insight to beams with either more flexibility or quicker slew rates which will require an inextensible Galerkin model from which to design the control system.

Future implementations of DSP based controllers should be able to achieve over 1KHz without much difficulty. Even now current DSP processors are achieving better than 25 MFLOPS while the one used in this experiment has only 8 MFLOPS. While the speed for this feedback controller processor is definitely I/O bound, we can still achieve excellent overall performance. and future architectures with better I/O rates will significantly improve the level of digital feedback control that is achievable.

Appendix

A

IBM PC AT Program to Control the Flexible Structure

```
/*****************************************************************/
/* Program    control.c                                          */
/*                                        DSP32 controller       */
/*****************************************************************/
#include <process.h>
#include <conio.h>
#include "dsp_util.c"

extern int dash16();
extern int dda6();
extern void exit();

int mode, based16=784, d16io[5], flag;
int basedda6=768, pa, pb, dda6ofst=2005;
int three=3, eleven=11, four=4, twelve=12;

main()                                                          main
{
        FILE *fp1,*fp2, *fp3, *fp4, *fp5, *fp6;

        float sensoff[3];
        float pos, initpos;
        float offset();
        float velplot[800], accelplot[800];
        float tempf,scale;
        int posplot[800];
        int control[800];
        int motor[800];
        int timing[800];
        int upsensor[3];
        int up1[800];
        int up2[800];
        int up3[800];
        int tippos[800];
        int sensor[3], cflag;
        int mot_inp, noloop;
        int i, j;
        int loopcount =9500;
        int loopstop = 9000;
        int temp,pos_ref;
        unsigned int refad, motad, flagad, controlad, sensorad, offad;
        unsigned int x2ad, yad;


/*****   Initialization of dash16      *****/
        initd16();
```

```
        mot_inp = dda6ofst;
        dda6( mot_inp );

        outp( basedda6+15, 0xbe );

/*****   Open Files   *****/
        if( (fp1 = fopen( "encoder.plo", "w" )) == NULL ) {
                fprintf( stderr, "couldn't open file count.plo\n" );
                exit( 1 );
                }

        if( (fp2 = fopen( "timing.plo", "w" )) == NULL ) {
                fprintf( stderr, "couldn't open file count.plo\n" );
                exit( 1 );
                }

        if( (fp3 = fopen( "accel.plo", "w" )) == NULL ) {
                fprintf( stderr, "couldn't open file control.plo\n" );
                exit( 1 );
                }
        if( (fp4 = fopen( "control.plo", "w" )) == NULL ) {
                fprintf( stderr, "couldn't open file itgl2st.plo\n" );
                exit( 1 );
                }

        if( (fp5 = fopen( "tach.plo", "w" )) == NULL ) {
                fprintf( stderr, "couldn't open file itgl2.plo\n" );
                exit( 1 );
                }

        if( (fp6 = fopen( "sensors.plo", "w" )) == NULL ) {
                fprintf( stderr, "couldn't open file itgl2st.plo\n" );
                exit( 1 );
                }

/*****   Compute the offset value for velocity and acceleration   *****/
        sensoff[1] = offset( 3, -1.0 );     /* Tachometer   */
        sensoff[0] = offset( 4, 1.0 );     /* Accelerometer   */
        printf("tach offset: %f  accel offset: %f\n", sensoff[1],sensoff[0]);

/*****   Start Execution   *****/
        printf(" Start execution\n");
        noloop = 599;

/**   Input initial position data   ***/
        pa = inp( basedda6+12 );
        pb = inp( basedda6+13 );
        initpos = ((pb*256 + pa) >> 4)*3.141592653/2048;
        printf(" Current position is %f\n", initpos );
        printf(" Input new position ( in radian < 2)  ");
        scanf("%f", &pos );
        if(pos > 2.0 ) pos=2.0;
```

```
        printf(" Input scale factor ");
        scanf("%f", &scale);
        pos_ref = (int) (pos*scale*1000.0);
        printf("pos_ref = %d\n",pos_ref);
        default_addr();
        if(!dsp_dl_exec("cntrl"))                                          110
                exit(1);
        dsp_run();
        refad = get_addr("posref");
        motad = get_addr("motor");
        flagad = get_addr("dsp_flag");
        sensorad = get_addr("sensor");
        offad = get_addr("offset");
        yad = get_addr("yi");
        x2ad = get_addr("x2i");
        printf("code is downloaded to DSP and its running\n");             120

        /*Set Multiplexer Scan limits  */

        mode = one;
        d16io[0] = three;              /*  Tachometer    channel three  */
        d16io[1] = four;               /*  Accelerometer channel four   */
        dash16( &mode, d16io, &flag);

/*****   Start the loop    *****/
        dsp_dl_int(refad, pos_ref);                                        130

/*  Download Sensor Offsets     */
        dsp_dl_array(offad, 4, sensoff);  /* 4 since floats  */
        cflag=dsp_up_int(flagad);
        printf("cflag=%d  (should be 0)\n",cflag);
        dsp_dl_int( flagad, 1 );
        cflag=dsp_up_int(flagad);
        printf("cflag=%d  (should be 1)\n",cflag);

        for( j = 0; j < noloop; j++ )  {                                   140

      /*   Load Counter 0 = loopcount */
                mode = eleven;
                d16io[0] = loopcount;
                dash16( &mode, d16io , &flag );

      /*****   Input position data    *****/
                pa = inp( basedda6+12 );
                pb = inp( basedda6+13 );
                sensor[2] = ((pb*256+pa) >> 4)*3.141592653/2.048;          150
                posplot[j] = sensor[2];

/*****   Input velocity and acceleration   *****/
                mode = three;
                dash16( &mode, d16io, &flag );
                sensor[1] = d16io[0];
```

```
                dash16( &mode, d16io, &flag);
                sensor[0] = d16io[0];

                dsp_dl_array(sensorad, 3, sensor);                          160
                dsp_dl_int(flagad,1);
                cflag=dsp_up_int(flagad);
                while(0!=(cflag=dsp_up_int(flagad)))
                        printf("%d    %d\n",j,cflag);
                mot_inp = dsp_up_int(motad);
                dsp_up_array(yad,3,upsensor);
                tippos[j] = dsp_up_int(x2ad);
                dda6( mot_inp );
                control[j] = mot_inp;
                velplot[j] = - sensor[1] - sensoff[1];                      170
                accelplot[j] = sensor[0] - sensoff[0];
                up1[j] = upsensor[0];
                up2[j] = upsensor[1];
                up3[j] = upsensor[2];

/*  DSP is now computing observer */

/*  Use up remaining time before taking next samples              */
                mode = twelve;
                d16io[0] = one;          /*   Latch before read   */     180
                dash16( &mode, d16io, &flag );
                timing[j]=d16io[1];
                while( loopstop <= d16io[1] )dash16( &mode, d16io, &flag);
        }
        dsp_halt();
/***    Input final position data    ***/
        pa = inp( basedda6+12 );
        pb = inp( basedda6+13 );
        initpos = ((pb*256 + pa) >> 4)*3.141592653/2048;
        printf(" Final position is %f\n", initpos );                        190

/*****   The End    *****/
   dda6( 2005 );
   printf("End of loop\n");
   for( j = 0; j < noloop; j++ )  {
                fprintf( fp1,"%d,%d\n", j, posplot[j]);
                fprintf( fp2,"%d,%d\n",j, timing[j]);
                fprintf( fp3,"%d,%f\n", j, accelplot[j]);
                fprintf( fp4,"%d,%d\n", j, 2005 - control[j]);
                fprintf( fp5,"%d,%f\n", j, velplot[j]);                     200
fprintf( fp6,"%d %d %d %d %d\n",j,up1[j],up2[j],up3[j],tippos[j]);
      }
   printf("The End.\n");
}

get_addr(s)                                                          get_addr
char *s;
{
```

```
        unsigned int i;
        if ((i=find_label_name(s)) == -1){                                    210
        printf("\nGlobal label '%s' missing from DSP32 program",s);
        exit(1);
        }
        return i;
}
```

Appendix

B

DSP32 Program to Control the Flexible Structure

```
/*************************************************************/
/*************************************************************/
/**  cntrl.s      by John Bartusek   200 Hz Sampling Rate  **/
/**  DSP32 assembly language program to control            **/
/**  Flexible Beam Manipulator                             **/
/*************************************************************/
/*************************************************************/

.rsect ".bank0"
.global dsp_flag, sensor, motor, offset, posref, start
.global loop1,yi,x2i
          r1 = 0
          *dsp_flag = r1
          3*nop
start:    r1 = *dsp_flag
          nop
          if (eq) goto start
          nop
          call _dsp32 (r14)
          nop
      int 2
      int offset
          r18 = -8
          r1 = 0
          *dsp_flag = r1
          3*nop
loop1:    r1 = *dsp_flag
          nop
          if (eq) goto loop1
          nop

          r5 = sensor   /* sensors downloaded as integers */
          r10 = posref
          r11 = flref   /* floating point of posref  */

/* Convert sensors to floating point  */
          a0 = float(*r5++)             /* accel  */
```

```
          a1 = float(*r5++)              /* tach   */
          a2 = float(*r5)                /* pos    */
  *r11 =  a3 = float(*r10)               /* posref */

          r6 = y  /* sensors converted to floating pt & scaled */
          r7 = sensgain   /* sensor gains                      */
          r8 = offset     /* sensor offsets                    */
          r9 = l          /* Observer Feedback Matrix L        */

  *r6++ = a0 = - *r8++ + a0 * *r7++      /* Sensor gains  */
  *r6++ = a1 = - *r8-- + a1 * *r7++      /* and Offsets   */
  *r6++r18  = a2 =  a2 * *r7             /*  used for ref */
          nop

/* Compute x2 = L.y(k) + z(k)  */
          r3 = z
          r6 = y
          r13 = x2   /* 3 vector in observer x2=L.y(k)+z(k) */
          r7 = ymd
          r8 = d
          r12 = u
          a0 = *r12
          3*nop

  *r7++ = a1 = *r6++ - a0 * *r8++
  *r7++ = a2 = *r6++ - a0 * *r8++
  *r7   = a3 = *r6 - a0 * *r8
          nop
          r7=ymd

          a0 = *r9++ * *r7++
          a0 = a0 + *r9++ * *r7++
          a0 = a0 + *r9++ * *r7++r18
  *r13++ =a0 = a0 + *r3++
          r7=ymd

          a0 = *r9++ * *r7++
          a0 = a0 + *r9++ * *r7++
          a0 = a0 + *r9++ * *r7++r18
  *r13++ =a0 = a0 + *r3++
          r7=ymd

          a0 = *r9++ * *r7++
```

```
          a0 = a0 + *r9++ * *r7++
          a0 = a0 + *r9 * *r7++r18
*r13++r18=a0 = a0 + *r3++r18

/* Compute Feedback control law:   K.x */
          r4 = k
          r6 = y
          r13 = x2

          a0 =      *r6++ * *r4++
          a0 = a0 + *r6++ * *r4++
          a0 = a0 + *r6++r18 * *r4++
          a0 = a0 + *r13++ * *r4++
          a0 = a0 + *r13++ * *r4++
          a0 = a0 + *r13++r18 * *r4

/*  Subtract from reference value  v * 4000 - K.x */
          r9  = flref
          a3  = - a0 + *r9      /*  ref - control   */

/* Saturator   */
          r1 = satur
          r2 = u

          a2 = -*r1       /* initialize a2 with negative limit  */
          a1 = -a3 + *r1  /* compare a3 with positive limit     */
          a3 = ifalt(*r1) /* if a3>2000, replace a3 with 2000   */
          a1 = a3 + *r1   /* compare a3 with negative limit     */
    *r2 = a3 = ifalt(a2)  /* if a3<-2000, replace a3 with -2000 */
          3*nop

/* Motor Offset  */
          r9 = tipscale
          r10 = motor
          r11 = motofst
          r12 = y
          r13 = x2
          r1 = yi
          r2 = x2i

          a1   = -a3 + *r11       /*  2005 - u  */
          a3 = *r13 * *r9
          5*nop
```

```
          *r10 = a0= int(a1)
          *r1++ = a1 = int(*r12++)
          *r1++ = a2 = int(*r12++)
          *r1   = a0 = int(*r12)
          *r2   = a0 = int(a3)
          4*nop

/*  Send out Control Signal                          */
          r1 = 0
          *dsp_flag = r1

/*  Update State Vector                              */

          r2 = a
          r3 = z
          r4 = b
          r5 = c
          r6 = y
          r7 = ymd
          r8 = d
          r12 = nextz
          r13 = u
          a0 = *r13
          3*nop
          *r7++ = a1 = *r6++ - a0 * *r8++
          *r7++ = a2 = *r6++ - a0 * *r8++
          *r7   = a3 = *r6 - a0 * *r8
          nop
          r7=ymd

          a0 = *r2 * *r3++        /*  a * z   */
          a0 = a0 + *r4++ * *r13  /* + b * u  */
          a0 = a0 + *r5++ * *r7++ /* + c * y  */
          a0 = a0 + *r5++ * *r7++
  *r12++ =a0 = a0 + *r5++ * *r7++r18
          nop
          r7=ymd

          a0 = *r2 * *r3++
          a0 = a0 + *r4++ * *r13
          a0 = a0 + *r5++ * *r7++
          a0 = a0 + *r5++ * *r7++
  *r12++ =a0 = a0 + *r5++ * *r7++r18
```

```
          nop
          r7=ymd

          a0 = *r2 * *r3++r18      /* skip back 8  */
          a0 = a0 + *r4++ * *r13
          a0 = a0 + *r5++ * *r7++
          a0 = a0 + *r5++ * *r7++
*r12++r18=a0 = a0 + *r5++ * *r7++r18
          nop
          r3=z
          r12=nextz


/*  Update nextz to z     */

          *r3++ = a0 = *r12++
          *r3++ = a1 = *r12++
          *r3   = a2 = *r12

          goto loop1
          nop

posref:   float 0.0
offset:   2*float 0.0
motofst:  float 2005.0
u:        float 0.0
nextz:    3*float 0.0
satur:    float 2000.0
y:        3*float 0.0
z:        3*float 0.0
x2:       3*float 0.0
ymd:      3*float 0.0
flref:    float 0.0
tipscale: float 0.091
motor:    int 0
dsp_flag: int 0
sensor:   3*int 0
x2i:      3*int 0
yi:       3*int 0
.align 4

 /*
          _dsp32 (n,A)
```

```
   const  int    n;
          float  *A;
   {
     scratch register short r1, r2, r3, r5;
     scratch pointer register short r4, r6, r14;
     scratch increment register short r15, r16;
     scratch register float a0;
#asm
*/

_dsp32:
         r5 = *r14++       /* r5 = Length of Array */
         r4 = *r14++       /* r4 points to start of Array */
         r5 = r5 - 2       /* r5 is a loop counter */
         r6 = _dsp32T      /* r6 points to float 2.0 */
         r15 = 3           /* pointer bumper */
         r16 = -3          /* pointer bumper */

         r1 = *r4++         /* w0 */
_dsp32A: r2 = *r4           /* w1 */
         *r4-- = r1h        /* b2 */
         *r4-- = r1l        /* b1 */
         r3 = r2*2
         *r4++r15 = r3h     /* b0, i.e., exponent field */
         r2 = r2 & 0x807f
         if(pl) goto _dsp32B
         *r4++r16 = r2l      /* b3 */
         goto _dsp32C
         *r4++ = a0 = - *r4 * *r6    /* negaitve number */
         nop              /* Dummy to make assembler work */
_dsp32B: *r4++ = a0 = *r4 * *r6      /* positive number */
_dsp32C: if(r5-- >=0) goto _dsp32A
         r1 = *r4++

         nop

_dsp32T: float 2.0

/*
#endasm
   }
*/
.rsect ".hi_ram"
```

```
a:          float 0.1518358
b:          float -3.0847574,-1.1355494, -1.1145619
c:          float 9.798564, -25.883911, -177.5527
            float 3.6014428, -9.766553, -70.725845
            float 3.539703,-9.679377, -77.52354
kp:         float 4.28e-5, 11.0, 5.045
            float -4.24e-4, 0.0, 0.0
k:          float 6.5195817e-7,6.860543471,3.45315
            float 3.9793755e-5,2.300105e-5,-4.095845e-5
l:          float -3.4019299, 20.70356,  220.12013
            float -1.2499559, 5.564637, 84.12455
            float -1.2383009, 5.3820543, 85.07537
sensgain:   float 1.0, -1.15, 1.0
d:          float -0.036449, 4.40807e-8, 4.1436e-5
```

BIBLIOGRAPHY

[1] G. H. Frank, *Design and Real-Time Control of a Flexible Arm.* Master's thesis, University of Maryland, College Park, 1986.

[2] L. S. Wang, *Control System Design for a Flexible Arm.* Master's thesis, University of Maryland, College Park, 1987.

[3] E. Schmitz, *Experiments on the End-Point Position Control of a Very Flexible One-Link Manipulator.* PhD thesis, Stanford University, 1985.

[4] K. J. Astrom and B. Wittenmark, *Computer Controlled Systems: Theory and Design.* Englewood Cliffs, N.J.: Printice-Hall, Inc., 1984.

[5] L. Meirovitch, *Analytical Methods in Vibrations.* 866 Third Avenue, New York, New York 10022: The Macmillan Company, 1967.

[6] T. A. Posbergh, *Modeling and Control of Mixed and Flexible Structures.* PhD thesis, University of Maryland, College Park, 1988.

[7] K. Kwakernak and R. Sivan, *Linear Optimal Control Systems.* New York: Wiley-Interscience, 1972.

[8] *Macsyma.* Symbolics Computers, Inc.

[9] J. J. Craig, *Adaptive Control of Mechanical Manipulators.* New York: Addison-Wesley Publishing Company, 1988.

[10] H. Elmqvist, *An Interactive Simulation Program For Nonlinear Systems.* Lund Institute of Technology.

[11] *WE DSP32-SL Support Software Library User Manual.* AT&T, AT&T Microelectronics, Dept. 50AL203140, 555 Union Boulevard, Allentown, PA 18103, March 1987.

[12] N. Sreenath, *Modeling and Control of Multibody Systems.* PhD thesis, University of Maryland, College Park, 1987.

[13] A. Isidori, *Nonlinear Control Systems: An Introduction.* Vol. 72, Berlin; Heidelberg; New York; Tokyo: Springer-Verlag, 1985.

[14] M. Fan, J. Koninckx, L. Wang, and A. Tits, "Console: a cad tandem for optimization-based design interacting with arbitrary simulators," Technical Report, University of Maryland, Systems Research Center, 1987.

[15] D. Shear, "Edn's dsp benchmarks," *EDN*, vol. 23, p. 126, September 1988.

[16] *D3EMU (release 1.7) Software Emulator for AT&T's DSP32 Floating Point Digital Signal Processor and Hardware Reference Manual for the DSP32-*

PC Plug-In Board for the PC/XT/AT. Communications, Automation & Control.

[17] *WE DSP32 Digital Signal Processor Information Manual.* AT&T, AT&T Microelectronics, Dept. 51Al230230, 555 Union Boulevard, Allentown, PA 18103, March 1988.

[18] *WE DSP32 Digital Signal Processor Application Guide.* AT&T, AT&T Microelectronics, Dept. 51Al230230, 555 Union Boulevard, Allentown, PA 18103, June 1988.

[19] T. J. and H. S.A., *Spectral Analysis Using the WE DSP32 Digital Signal Processor.* AT&T, AT&T Microelectronics, Dept. 51Al230230, 555 Union Boulevard, Allentown, PA 18103.

[20] G. Sideris, "A new version of at&t's dsp will peak at 25 megaflops," *Electronics*, pp. 57–59, July 1987.

[21] C. Chen, *Linear System Theory and Design.* New York: Holt, Rinehart and Winston, 1970.

[22] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems.* New York: Addison-Wesley Publishing Company, 1986.

[23] Hanselmann, "Implementation of digital controllers-a survey," *Automatica*, vol. 23, no. 1, pp. 7–32, 1987.

[24] T. Kailath, *Linear Systems. Prentice Hall Information and System Sciences Series*, Englewood Cliffs, N.J.: Printice-Hall, Inc., 1980.

[25] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing.* New Jersey: Prentice-Hall, 1975.

[26] H. Goldstein, *Classical Mechanics.* Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1980.